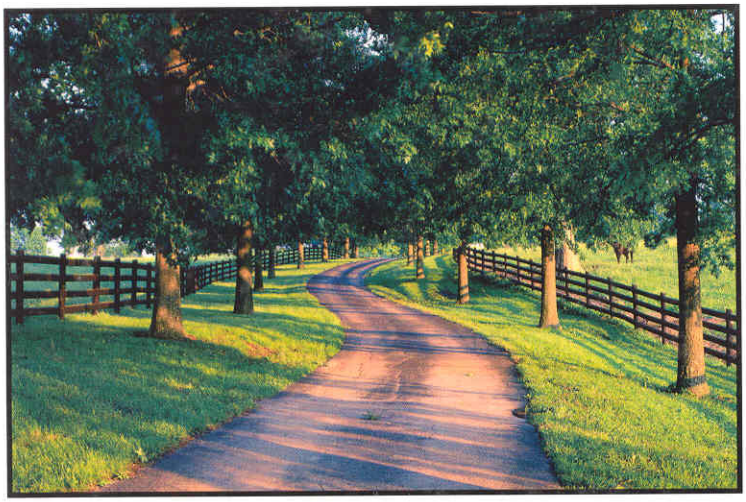
Профессиональное руководство по SQL Server

Структура и реализация



Кен Хендерсон

Профессиональное руководство по SQL Server

Структура и реализация

Кен Хендерсон

Москва • Санкт-Петербург • Киев
2006

ББК 32.973.26-018.2.75
Х38
УДК 681.3.07

Издательский дом "Вильямс"

Зав. редакцией *С.Н. Тригуб*

Перевод с английского и редакция *К.А. Птицына*

По общим вопросам обращайтесь в Издательский дом "Вильямс"
по адресу: info@williamspublishing.com, http://www.williamspublishing.com
115419, Москва, а/я 783; 03150, Киев, а/я 152

Хендерсон, Кен.

Х38 Профессиональное руководство по SQL Server: структура и реализация.: Пер. с англ. — М. : Издательский дом "Вильямс", 2006. — 1056 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0912-0 (рус.)

Подробное описание системы управления базами данных SQL Server, которая в настоящее время получила очень широкое распространение. Рассматривается весь состав программных средств, лежащих в основе этой СУБД и показано, как работают и взаимодействуют ее многочисленные компоненты. Приведены необходимые сведения о фундаментальных технологиях Windows, на которых основана работа программы SQL Server. В книге принят комплексный подход к изложению сведений по архитектуре, внутренней организации и настройке SQL Server, дополняющих существующие справочные руководства и официальную документацию.

Книга предназначена для широкого круга читателей, может использоваться в качестве учебного пособия и позволяет достичь глубокого понимания принципов работы SQL Server.

ББК 32.973.26-018.2.75

ISBN 5-8459-0912-0 (рус.)
ISBN 0-201-70047-6 (англ.)

Оглавление

Предисловие 22
Историческая перспектива 25
Предисловие автора 29
Благодарности 32
Введение 34
Об авторе 45

ЧАСТЬ I. ОСНОВНЫЕ СВЕДЕНИЯ 47
ГЛАВА 1. Краткий обзор средств SQL Server 48
ГЛАВА 2. Основы Windows 57
ГЛАВА 3. Процессы и потоки 70
ГЛАВА 4. Основные принципы организации памяти 159
ГЛАВА 5. Основные принципы ввода-вывода 246
ГЛАВА 6. Основы организации сетей 360
ГЛАВА 7. Технология COM 405
ГЛАВА 8. Язык XML 420

ЧАСТЬ II. ПОДСИСТЕМЫ, КОМПОНЕНТЫ И ТЕХНОЛОГИИ 455
ГЛАВА 9. Программа SQL Server как серверное приложение 456
ГЛАВА 10. Планировщик непривилегированного режима 465
ГЛАВА 11. Управление памятью программы SQL Server 484
ГЛАВА 12. Процессор запросов 507
ГЛАВА 13. Транзакции 554
ГЛАВА 14. Курсоры 580
ГЛАВА 15. Средства ODSOLE 620
ГЛАВА 16. Полнотекстовый поиск 674

ЧАСТЬ III. СЛУЖБЫ ДАННЫХ 695
ГЛАВА 17. Объединения серверов 696
ГЛАВА 18. Технологии SQLXML 708
ГЛАВА 19. Службы рассылки извещений 820
ГЛАВА 20. Службы Data Transformation Services 865
ГЛАВА 21. Репликация снимка 936
ГЛАВА 22. Транзакционная репликация 953
ГЛАВА 23. Репликация путем слияния 974

ЧАСТЬ IV. НЕДОКУМЕНТИРОВАННЫЕ СРЕДСТВА ПРОГРАММЫ SQL SERVER 991
ГЛАВА 24. Поиск недокументированных средств 992
ГЛАВА 25. Программа DTSDIAG 1002
Предметный указатель 1014

Содержание

Предисловие 22

Историческая перспектива 25

Предисловие автора 29

Благодарности 32

Введение 34

Оперативная документация Books Online 34
Отладчик WinDbg 35
Основные понятия 36
Практические рекомендации 37
Широта изложения 38
Применение языка C++ 38
Применение языка Visual C++ 6.0 40
Разделы с определениями терминов и вопросами для самопроверки 40
Версии SQL Server 41
Различия между квалифицированным программистом и квалифицированным пользователем 41

Об авторе 45

ЧАСТЬ I. ОСНОВНЫЕ СВЕДЕНИЯ 47

Глава 1. Краткий обзор средств SQL Server 48

Краткий обзор содержимого глав 48
Глава 2. Основы Windows 48
Глава 3. Процессы и потоки 48
Глава 4. Основные принципы организации памяти 49
Глава 5. Основные принципы ввода-вывода 49
Глава 6. Основы организации сетей 49
Глава 7. Технология COM 50
Глава 8. Язык XML 50
Глава 9. Программа SQL Server как серверное приложение 50
Глава 10. Планировщик непривилегированного режима 50
Глава 11. Управление памятью программы SQL Server 51
Глава 12. Процессор запросов 51
Глава 13. Транзакции 51
Глава 14. Курсоры 51
Глава 15. Средства ODSOLE 52
Глава 16. Полнотекстовый поиск 52
Глава 17. Объединения серверов 52
Глава 18. Технологии SQLXML 52

Глава 19. Службы рассылки извещений 52
Глава 20. Службы Data Transformation Services 53
Глава 21. Репликация снимка 53
Глава 22. Транзакционная репликация 53
Глава 23. Репликация путем слияния 53
Глава 24. Поиск недокументированных средств 53
Глава 25. Программа DTSDIAG 54
Изложение аналогичного материала в разных главах 54
Использование кода 55

Глава 2. Основы Windows 57
API-интерфейс Win32 57
Сравнение непривилегированного и привилегированного режимов 58
Процессы и потоки 59
Особенности виртуальной и физической памяти 59
Подсистемы 61
Библиотеки динамического связывания 62
Инструментальные средства 65
Программа TList 65
Программа Pviewer 65
Программа Pview 65
Программа Perfmon 66
Программа WinDbg 66
Резюме 68
Вопросы для самопроверки 68

Глава 3. Процессы и потоки 70
Процессы 70
Процессы. Основные термины и определения 70
Краткий обзор 71
Процессы. Основные функции API-интерфейсов 72
Основные инструментальные средства анализа процессов 72
Основные счетчики программы Perfmon 72
Внутренняя организация процесса 72
Завершение процесса 75
Функция SetErrorMode 76
Процессы. Резюме 82
Процессы. Вопросы для самопроверки 82
Потоки 83
Потоки. Основные термины и определения 84
Потоки. Основные функции API-интерфейсов 84
Основные инструментальные средства, относящиеся к потокам 85
Основные счетчики программы Perfmon 85
Внутреннее устройство потока 86
Первичный поток 87
Сравнение процессов и потоков 88
Создание и уничтожение потоков 89
Завершение работы потока 90

Функции _beginthreadex и _endthreadex 91
Функции потока 92
Потоки и обработка исключительных ситуаций 93
Функция GetLastError 94
Микропотоки 95
Потоки. Резюме 103
Потоки. Вопросы для самопроверки 103
Планирование потоков 105
Планирование потоков. Основные термины и определения 105
Краткий обзор 106
Планирование потоков. Основные функции API-интерфейсов 107
Планирование потоков. Основные инструментальные средства 107
Внутренняя организация планирования потоков 107
Состояния потока 109
Приоритеты потока 110
Потоки реального времени 113
Планирование очередей 114
Переключение контекста 115
Родственность процессора для потоков 115
Выбор потока 119
Приостановка и возобновление работы потоков 120
Перевод потока в состояние ожидания 121
Планирование потоков. Резюме 130
Планирование потоков. Вопросы для самопроверки 130
Синхронизация потоков 133
Синхронизация потоков. Основные термины и определения 133
Синхронизация потоков. Основные функции API-интерфейсов 134
Синхронизация потоков. Основные инструментальные средства 135
Синхронизация с использованием конструкций непривилегированного режима 135
Синхронизация с использованием объектов привилегированного режима 143
Синхронизация потоков. Резюме 156
Синхронизация потоков. Вопросы для самопроверки 156

Глава 4. Основные принципы организации памяти 159

Основы организации памяти 159
Основы организации памяти. Основные термины и определения 159
Основы организации памяти. Основные функции API-интерфейса Win32 161
Основы организации памяти. Основные инструментальные средства 162
Основные счетчики программы Perfmon 162
Адреса 163
Основные службы управления памятью 164
Значения степени детализации 164
Защита памяти процесса 165
Секции 166
Системный файл подкачки 171
Адресные расширения для работы с окнами 172

Настройка памяти приложения 173
Преобразование адресов 176
Расширение физического адреса 179
Основы функционирования памяти. Резюме 189
Основы функционирования памяти. Вопросы для самопроверки 189
Виртуальная память 192
Виртуальная память. Основные термины и определения 192
Виртуальная память. Основные функции API-интерфейсов 193
Атрибуты защиты страницы 194
Копирование при записи 195
Резервирование памяти 196
Закрепление виртуальной памяти 198
Инициализация и модификация страниц 199
Освобождение памяти 200
Блокировка страниц в памяти 201
Виртуальная память. Резюме 213
Виртуальная память. Вопросы для самопроверки 214
Динамические области памяти 216
Динамические области памяти. Основные термины и определения 216
Динамические области памяти. Основные функции API-интерфейсов 217
Применяемая по умолчанию динамическая область памяти 217
Распределение памяти из динамической области памяти 218
Пользовательские динамические области памяти 219
Динамические области памяти. Резюме 227
Динамические области памяти. Вопросы для самопроверки 228
Совместно используемая память 229
Совместно используемая память. Основные термины и определения 230
Совместно используемая память. Основные функции API-интерфейсов 230
Программа SQL Server и совместно используемая память 230
Объекты секции памяти 231
Файлы, отображаемые на память 232
Отображения файлов образов 233
Совместно используемая память. Резюме 243
Совместно используемая память. Вопросы для самопроверки 244

Глава 5. Основные принципы ввода-вывода 246

Основы ввода-вывода 247
Основы ввода-вывода. Основные термины и определения 247
Основы ввода-вывода. Основные функции API-интерфейсов 248
Основы ввода-вывода. Основные инструментальные средства 249
Основные счетчики программы Perfmon 249
Краткий обзор 250
Средства синхронного ввода-вывода 251
Основы ввода-вывода. Резюме 254
Основы ввода-вывода. Вопросы для самопроверки 255
Асинхронный и небуферизованный ввод-вывод 256
Асинхронный и небуферизованный ввод-вывод. Основные термины и определения 256

Асинхронный и небуферизованный ввод-вывод. Основные функции API-интерфейсов 257
Краткий обзор 257
Небуферизованный ввод-вывод 268
Асинхронный и небуферизованный ввод-вывод. Резюме 285
Асинхронный и небуферизованный ввод-вывод. Вопросы для самопроверки 287
Ввод-вывод со сборкой-разборкой 289
Ввод-вывод со сборкой-разборкой. Основные термины и определения 289
Ввод-вывод со сборкой-разборкой. Основные функции API-интерфейсов 290
Краткий обзор 290
Ввод-вывод со сборкой-разборкой. Резюме 304
Ввод-вывод со сборкой-разборкой. Вопросы для самопроверки 305
Порты завершения ввода-вывода 306
Порты завершения ввода-вывода. Основные термины и определения 306
Порты завершения ввода-вывода. Основные функции API-интерфейсов 307
Краткий обзор 307
Порты завершения ввода-вывода. Резюме 337
Порты завершения ввода-вывода. Вопросы для самопроверки 337
Ввод-вывод с помощью отображаемых на память файлов 339
Ввод-вывод с помощью отображаемых на память файлов. Основные термины и определения 340
Ввод-вывод с помощью отображаемых на память файлов. Основные функции API-интерфейсов 340
Краткий обзор 340
Ввод-вывод с помощью отображаемых на память файлов. Резюме 358
Ввод-вывод с помощью отображаемых на память файлов. Вопросы для самопроверки 358

Глава 6. Основы организации сетей 360

Краткий обзор 360
Организация сетей. Основные термины и определения 360
Организация сетей. Основные функции API-интерфейсов 361
Эталонная модель OSI 362
Компоненты организации сетевого взаимодействия Windows 364
API-интерфейс Named Pipes 366
API-интерфейс Windows Sockets 380
Приложения Winsock с установлением логических соединений 381
Приложения Winsock без установления логических соединений 389
Сокеты и API-интерфейс Win32 389
Расширения Winsock 391
Сравнение сокетов и каналов 392
Дистанционный вызов процедур 400
Маршалинг 401
Асинхронные средства RPC 401
Библиотека этапа прогона RPC 402
Резюме 402
Вопросы для самопроверки 402

Глава 7. Технология COM 405
Краткий обзор 405
Эпоха в программировании, предшествующая появлению технологии COM 406
Появление технологии COM 409
Основная архитектура 411
Интерфейсы 411
Подсчет ссылок 412
Метод QueryInterface 412
Маршалинг 412
Агрегирование 413
Практическое применение технологии COM 413
Модели многопотоковой поддержки 415
Главный контейнер STA 417
Технология COM и программа SQL Server 417
Резюме 418
Вопросы для самопроверки 418

Глава 8. Язык XML 420
Краткий обзор 421
Недостатки языка HTML 422
Краткая история языка XML 423
Сравнение возможностей языков XML и HTML на примере 424
Нюансы обозначений 427
Сравнение формально правильных и допустимых документов 429
Определения типа документа 430
Схемы XML 433
Преобразование документа XML в документ HTML с использованием таблицы стилей 436
Объектная модель документа 444
Обработка документов XML с помощью средств MSXML 445
Средства MSXML и модель DOM 445
Средства MSXML и интерфейс SAX 446
Дополнительные источники информации 450
Дополнительные материалы для чтения 450
Инструментальные средства 451
Резюме 452
Вопросы для самопроверки 453

ЧАСТЬ II. ПОДСИСТЕМЫ, КОМПОНЕНТЫ И ТЕХНОЛОГИИ 455

Глава 9. Программа SQL Server как серверное приложение 456
Программа SQL Server и организация сетей 456
Исполняемый файл программы SQL Server 459
Библиотеки DLL программы SQL Server 460
Средства ввода-вывода программы SQL Server 460
Компоненты программы SQL Server 462
Резюме 463
Вопросы для самопроверки 464

Глава 10. Планировщик непривилегированного режима 465
Цели проекта UMS 465
Сравнение средств планирования в непривилегированном и привилегированном режимах 466
Сравнение вытесняющего и кооперативного управления задачами 467
Выполнение задач планирования компонентом UMS 468
Планировщик UMS 469
Списки планировщика UMS 470
Список рабочих (микро)потоков 470
Список работоспособных (микро)потоков 472
Список ожидающих (микро)потоков 474
Список запросов ввода-вывода 474
Контролируемый по таймеру список 476
Цикл простоя 477
Переход в режим вытеснения 477
Режим микропотоков 479
Скрытые планировщики 480
Команда DBCC SQLPERF(umsstats) 480
Резюме 481
Вопросы для самопроверки 482

Глава 11. Управление памятью программы SQL Server 484
Области памяти 484
Определение размеров областей памяти 485
Область BPool 487
Хэширование 489
Примитивные операции распределения 489
Массивы страниц 491
Массив BUF 491
Структура отображения с информацией о закреплении 492
Средства AWE 492
Средства отложенной записи 493
Вычисление объема физической памяти 494
Сброс на диск и освобождение страниц 494
Контрольная точка 496
Секции 497
Диспетчеры памяти 497
Диспетчер памяти Connection 497
Диспетчер памяти Query Plan 497
Диспетчер памяти Optimizer 498
Диспетчер памяти Utility 498
Диспетчер памяти General 498
Операции распределения памяти с помощью диспетчера памяти OS 498
Диспетчер памяти, применяемый в условиях нехватки памяти 499
Интерфейс IMalloc 500
Описание общей картины 500
Резюме 504
Вопросы для самопроверки 505

Глава 12. Процессор запросов 507
Основные термины и определения 507
Синтаксический анализ 508
Этапы оптимизации 509
Тривиальная оптимизация плана 509
Упрощение 511
Полная оптимизация 511
Преобразование 514
Пределы оптимизации 514
Перехват параметров 515
Автоматическая параметризация 518
Применение индексов 519
Местонахождение данных об индексах 520
Покрывающие индексы 522
Проблемы производительности 523
Пересечение индексов 523
Фрагментация индекса 524
Индексы на представлениях и вычисленных столбцах 528
Блокировка и индексы 532
Статистические данные 532
Кардинальность 533
Плотность 533
Избирательность 533
Проблемы производительности 533
Организация хранения статистических данных 535
Статистические данные о столбцах 536
Структура статистических данных 536
Обновление статистических данных 537
Оценка избирательности 539
Индексируемые выражения 540
Преобразование операторов 541
Свертывание 543
Порядок соединений и выбор типа 543
Соединения с помощью вложенных циклов 545
Соединения слиянием 546
Хэшированные соединения 547
Подзапросы и альтернативные варианты соединений 548
Логические и физические операторы 549
Конструкция DISTINCT 550
Конструкция GROUP BY 550
Конструкция ORDER BY 551
Накопление промежуточных результатов 551
Резюме 552
Вопросы для самопроверки 553

Глава 13. Транзакции 554
Проверка соблюдения свойств ACID 554
Неразрывность 554

Непротиворечивость 555
Изолированность 555
Устойчивость 556
Принципы выполнения транзакций программы SQL Server 556
Типы транзакций 557
Автоматические транзакции 557
Неявные транзакции 558
Определяемые пользователем транзакции 558
Распределенные транзакции 559
Полный отказ от использования транзакций 559
Команды, требующие минимального объема записи в журнал 559
Транзакции и модели восстановления 560
Базы данных, допускающие только чтение, и однопользовательские базы данных 560
Автоматическое управление транзакциями 561
Уровни изоляции транзакции 563
Уровень изоляции транзакции READ UNCOMMITTED 563
Уровень изоляции транзакции READ COMMITTED 565
Уровень изоляции транзакции REPEATABLE READ 566
Уровень изоляции транзакции SERIALIZABLE 567
Команды управления транзакциями и синтаксическая структура этих команд 568
Вложенные транзакции 568
Команда SAVE TRAN и точки сохранения 572
Случайно выполняемые команды ROLLBACK 573
Синтаксические конструкции языка T-SQL, недопустимые в транзакциях 575
Отладка транзакций 575
Оптимизация кода транзакций 577
Резюме 578
Вопросы для самопроверки 579

Глава 14. Курсоры 580
Краткий обзор 580
Несколько слов о курсорах и базах данных ISAM 581
Типы курсоров 584
Курсоры с перемещением только вперед 584
Динамические курсоры 586
Статические курсоры 587
Курсоры ключевого набора 588
Рекомендации по правильному использованию курсоров 590
Динамические запросы 591
Операции, ориентированные на обработку строк 593
Прокручиваемые формы 596
Синтаксические конструкции для работы с курсорами, применяемые в языке Transact-SQL 596
Команда DECLARE CURSOR 597
Команда OPEN 601
Оператор FETCH 602

Оператор CLOSE 606
Оператор DEALLOCATE 606
Настройка конфигурации курсоров 606
Асинхронные курсоры 607
Способ автоматического закрытия курсоров, определяемый стандартом ANSI/ISO 608
Определение применяемого по умолчанию режима создания глобальных или локальных курсоров 611
Обновление курсоров 612
Переменные курсора 613
Хранимые процедуры для работы с курсорами 615
Оптимизация производительности курсоров 616
Резюме 618
Вопросы для самопроверки 618

Глава 15. Средства ODSOLE 620

Краткий обзор 620
COM-объекты и модели многопотоковой поддержки 621
Сравнение раннего и позднего связывания 624
Процедуры sp_OA 625
Процедура sp_OACreate 625
Процедура sp_OAMethod 627
Процедуры sp_OASetProperty и sp_OAGetProperty 627
Процедура sp_OAGetErrorInfo 628
Процедура sp_OADestroy 628
Процедура sp_OAStop 628
Переход по компонентам имени объекта 628
Именованные параметры 629
Автоматизация с помощью средств ODSOLE 629
Процедура sp_checkspelling 630
Процедура sp_vbscript_reg_ex 632
Автоматизация доступа к классам инфраструктуры .NET Framework с помощью интерфейса COM Interop 635
Использование COM-объектов в определяемых пользователем функциях 638
Автоматизация интерфейса SQL-DMO с помощью средств ODSOLE 641
Процедура sp_exporttable 641
Процедура sp_generate_script 647
Использование средств ODSOLE для автоматизации производных объектов 659
Создание массивов в языке T-SQL с помощью COM-объектов 663
Резюме 671
Вопросы для самопроверки 672

Глава 16. Полнотекстовый поиск 674

Краткий обзор 674
Подробные сведения об организации полнотекстовых средств поиска SQL Server 675
Полнотекстовый поиск в данных, отличных от данных SQL Server 678
Средства полнотекстового поиска в двоичных данных 678

Подготовка полнотекстовых индексов 680
Полнотекстовые предикаты 685
Предикат CONTAINS 686
Предикат FREETEXT 689
Функции набора строк 690
Функция набора строк CONTAINSTABLE 690
Функция набора строк FREETEXTTABLE 693
Резюме 693
Вопросы для самопроверки 694

ЧАСТЬ III. СЛУЖБЫ ДАННЫХ 695

Глава 17. Объединения серверов 696

Секционированные представления 696
Оператор BETWEEN и запросы к секционированному представлению 703
Распределенные секционированные представления 705
Резюме 707
Вопросы для самопроверки 707

Глава 18. Технологии SQLXML 708

Краткий обзор 709
Синтаксический анализатор MSXML 710
Конструкция FOR XML 715
Использование конструкции FOR XML 718
Запрос SELECT...FOR XML (серверный) 718
Режим RAW 719
Режим AUTO 720
Опция ELEMENTS 721
Режим EXPLICIT 722
Директива hide 728
Директива cdata 729
Директивы id, idref и idrefs 730
Запрос SELECT...FOR XML (клиентский) 732
Функция OPENXML 734
Использование функции OPENXML 736
Параметр flags 740
Формат краевой таблицы 741
Вставка данных с помощью функции OPENXML 742
Доступ к программе SQL Server по протоколу HTTP 745
Настройка виртуального каталога 746
Запросы URL 749
Использование запросов URL 750
Специальные символы 753
Таблицы стилей 753
Тип информационного наполнения 755
Получение результатов в коде, отличном от XML 756
Хранимые процедуры 757

Запросы с шаблонами 758
Параметризованные шаблоны 760
Таблицы стилей 761
Применение таблиц стилей в клиентской программе 763
Клиентские шаблоны 765
Схемы отображения 766
Схемы отображения XDR 767
Схемы отображения XSD 771
Шаблоны обновления 779
Принципы действия шаблонов обновления 780
Отображение данных 781
Значения NULL 783
Параметры 783
Способ обновления нескольких строк 785
Полученные результаты 786
Значения столбца identity 786
Глобально уникальные идентификаторы 788
Компонент XML Bulk Load 789
Использование компонента XML Bulk Load 789
Фрагменты XML 790
Принудительное применение ограничений проверки и целостности 791
Дублирующиеся ключи 791
Столбцы IDENTITY 792
Значения NULL 793
Блокировка таблиц 794
Транзакции 794
Ошибки 796
Создание объектов схем базы данных 796
Управляемые классы 798
Поддержка службы Web SQLXML (SOAP) 801
Ограничения средств SQLXML 807
Процедура sp_xml_concat 808
Процедура sp_run_xml_proc 811
Резюме 816
Вопросы для самопроверки 817

Глава 19. Службы рассылки извещений 820

Принципы работы служб Notification Services 820
Утилита NSControl 822
Базы данных экземпляра и приложения 822
Файлы конфигурации 822
Исполняемый файл NSService.exe 824
Компоненты приложения для рассылки извещений 825
Средства доступа к данным о событиях 826
Генератор 827
Функция рассылки извещений 828
Распределительные серверы 833
Подписчики и подписки 835

Создание собственного приложения рассылки извещений 839
Процесс разработки приложения Notification Services 839
Создание файлов конфигурации 840
Создание определяемой пользователем базы данных 846
Создание баз данных экземпляра и приложения 848
Регистрация экземпляра 848
Разрешение на использование экземпляра 849
Запуск службы 849
Создание приложения управления подпиской 850
Проверка приложения рассылки извещений 857
Возможные усовершенствования 857
Резюме 862
Вопросы для самопроверки 863

Глава 20. Службы Data Transformation Services 865

Краткий обзор 866
Пакеты 867
Соединения 868
Задачи 869
Средство многофазной перекачки данных 870
Задача Bulk Insert 872
Задача Data Driven Query 873
Преобразования ActiveX 876
Преобразования других типов 877
Поисковые запросы 877
Свойства потока данных 879
Свойство Close connection on completion 879
Свойство Execute on main package thread 880
Свойство Step priority 880
Службы DTS и транзакции 881
Управление потоком данных пакета с помощью средств поддержки сценариев 882
Организация цикла с использованием отдельного пакета 882
Организация цикла с использованием свойства ExecutionStatus 883
Реализация способа выполнения по условию 887
Организация выполнения по условию с помощью свойства ExecutionStatus и задач ActiveX Script 888
Способ выполнения по условию с учетом успешного или неудачного завершения задачи 889
Параметризованные пакеты DTS 889
Средство доступа к набору строк DSO 891
Использование средств DTS для преобразования данных, репликация которых осуществляется с помощью подписок 892
Принципы работы пакета DTS, используемого для преобразования данных подписки 894
Определяемые пользователем задачи 895
Создание новой определяемой пользователем задачи 895
Создание новой определяемой пользователем задачи на основе примера задачи 904

Отладка компонентов определяемой пользователем задачи 912
Настройка испытательного приложения 915
Управление службами DTS с помощью средств Automation 916
Простое приложение служб DTS, действующее на основе средств Automation 916
Приложение DTSPkgGuru 918
Резюме 933
Вопросы для самопроверки 933

Глава 21. Репликация снимка 936

Краткий обзор 937
Программа Snapshot Agent 937
Функции программ Snapshot Agent и Distribution Agent 941
Задачи программы Snapshot Agent 941
Задачи программы Distribution Agent 945
Обновляемые подписки 945
Немедленное обновление 946
Обновление в порядке очереди 948
Немедленное обновление с обновлением в порядке очереди в качестве резервного варианта 948
Активизация удаленного агента 949
Удаление устаревших данных репликации 950
Резюме 951
Вопросы для самопроверки 951

Глава 22. Транзакционная репликация 953

Краткий обзор 953
Таблица MSrepl_commands 954
Процедура sp_replcmds 955
Кэш статей 956
Параметры процедуры sp_replcmds 957
Процедура sp_repldone 958
Хранимые процедуры обновления 959
Параллельная обработка снимка 961
Обновляемые подписки 964
Немедленное обновление 964
Обновление в порядке очереди 966
Немедленное обновление с обновлением в порядке очереди в качестве резервного варианта 967
Проверка достоверности реплицированных данных 967
Пропуск ошибок 971
Удаление устаревших данных 971
Резюме 972
Вопросы для самопроверки 972

Глава 23. Репликация путем слияния 974

Краткий обзор 974
Разрешение конфликтов 977
Поколения 979

Применение критериев выборки 981
Оптимизация процесса синхронизации 982
Динамически определяемые критерии выборки 983
Критерии выборки на основе соединения 984
Динамически сопровождаемые снимки 984
Управление диапазонами идентификаторов 985
Резюме 988
Вопросы для самопроверки 989

ЧАСТЬ IV. НЕДОКУМЕНТИРОВАННЫЕ СРЕДСТВА ПРОГРАММЫ SQL SERVER 991

Глава 24. Поиск недокументированных средств 992
Богатства, скрытые в таблице syscomments 994
Недокументированные команды DBCC 995
Недокументированные флажки трассировки 995
Другие недокументированные объекты таблицы syscomments 995
Недокументированные объекты таблицы sysobjects 996
Недокументированные расширенные процедуры 996
Недокументированные функции 997
Поддержка сценариев для недокументированных и системных объектов 998
Россыпь сокровищ программы Profiler 999
Изучение инсталляционных сценариев 1000
Импорт библиотек DLL 1000
Резюме 1001
Вопрос для самопроверки 1001

Глава 25. Программа DTSDIAG 1002
Краткое описание 1002
Структура приложения 1003
Резюме 1013

Предметный указатель 1014

Предисловие

Составляя программы для компьютеров, я вначале не утруждал себя изучением системы команд, которая использовалась для разработки. Мне удавалось заставить работать большинство приложений, но процесс отладки при возникновении нетривиальных проблем оставался весьма трудоемким. А по мере усложнения программ трудности возрастали. В конечном итоге я уперся в глухую стену. Тогда я на время отказался от бесплодных попыток продолжить разработку и стал изучать систему команд и основополагающую архитектуру. Вскоре произошли удивительные события: я внезапно преодолел возникшую передо мной стену и добился гораздо более значительных успехов. Разочарование сменилось уверенностью, проблемы, казавшиеся сложными, перестали быть таковыми, и я начал решать задачи, которые раньше казались неосуществимыми.

Вы, несомненно, тоже испытывали нечто подобное. Можно провести аналогию с обучением игре на музыкальном инструменте. Научиться играть гаммы или простые мелодии довольно просто. Но для того, чтобы стать виртуозом, необходимо иметь глубокие знания теории и великолепно владеть самим инструментом. В этом и состоит различие между поверхностными знаниями (приобретение которых дает возможность выполнять простейшие действия) и глубокими (которые позволяют понять, как действительно устроен тот инструмент, которым вы хотите овладеть, а затем синтезировать новые знания на основе глубокого понимания предмета).

Кен Хендерсон (Ken Henderson) затратил невероятные усилия на раскрытие секретов программы SQL Server компании Microsoft и недавно выпустил две книги, *The Guru's Guide to Transact-SQL* (Addison-Wesley, 2000) и *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML* (Addison-Wesley, 2002). Кен – один из тех людей, которые стремятся овладеть глубокими знаниями. Ему недостаточно просто понять, как действовать с помощью того или иного инструмента; он хочет знать, как действует сам инструмент. Прежде чем написать свою последнюю книгу, представленную вниманию читателя, Кен, начиная от самых глубоких основ, изучал, как работает программа SQL Server, а затем изложил накопленные им знания в книге.

В части I, "Основные сведения", описана фундаментальная инфраструктура, на основе которой сформирована программа SQL Server. В этой части рассматриваются различные службы и средства Windows, поэтому ее изучение позволяет понять, как программа SQL Server (или другие высокопроизводительные приложения Windows) взаимодействуют с операционной системой Windows.

Часть II, "Подсистемы, компоненты и технологии", посвящена описанию устройства основной реляционной машины SQL Server. В этой части Кен рассматривает UMS (User Mode Scheduler – планировщик непривилегированного режима) – базовый компонент программы SQL Server, который позволяет эффективно

использовать эту программу для обслуживания тысяч пользователей и обрабатывать десятки тысяч транзакций в секунду. Наиболее важной главой этой части является глава с описанием процессора запросов. Кену удалось исключительно полно описать весь процесс преобразования запроса из текста на языке SQL (подготовленного приложением или пользователем) в последовательность физических операторов, передаваемых машине выполнения для решения задачи, поставленной в запросе. Он показал, как в этом процессе осуществляются стадии синтаксического анализа, нормализации, оптимизации и окончательного преобразования в последовательность физических операторов. Во всей данной части Кен описывает многочисленные источники входной информации для оптимизатора, способы выбора плана и стратегии выполнения. Кроме того, в данной части приведен ряд оригинальных рекомендаций, позволяющих пользователю изучить работу рассматриваемого сложного компонента SQL Server и определить, почему оптимизатор выбрал тот или иной конкретный план выполнения предъявленного ему сложного запроса.

В части III, "Службы данных", обсуждаются различные способы, с помощью которых пользователь может обеспечить взаимодействие с основной машиной данных SQL Server. В этой части Кен описал средства XML программы SQL Server и показал, как можно использовать службы DTS (Data Transformation Services — службы преобразования данных) для преобразования и перемещения данных из базы данных и в базу данных SQL Server. Рассматриваются также недавно включенные в состав программного продукта службы Notification Services (службы рассылки извещений), которые могут применяться для создания масштабируемых и высокопроизводительных служб доставки данных, основанных на обработке событий и подготовке подписок. Наконец, Кен очень подробно обрисовал различные технологии репликации, поддерживаемые программой SQL Server.

Обычно в предисловии указывается, для какой аудитории предназначена книга. В данном случае рекомендация является вполне однозначной — данная книга необходима всем, кто стремится превратить свои, возможно, поверхностные представления о программе SQL Server в самые глубокие знания. У многих людей есть естественное стремление изучать любую новую тему досконально. Если читателю недостаточно просто знать о существовании некоторого механизма, и он хочет разобраться в том, как и почему работает этот механизм, то книга Кена действительно поможет ему удовлетворить творческий голод. Возможно, читатель обнаружил, что ранее достаточные знания программы SQL Server больше не позволяют находить ответы на сложные вопросы и решать проблемы, с которыми ему приходится сталкиваться. Книга Кена позволяет выйти далеко за рамки оперативной документации Books Online, относящейся к программе SQL Server, и понять внутреннее функционирование этого программного продукта. С другой стороны, читатель, возможно, просто хочет знать, как спроектировано и создано такое сложное и высокопроизводительное приложение, как SQL Server. И в этом случае настоящая книга полностью удовлетворит его потребности.

Я могу со всей уверенностью гарантировать, что любой, кто регулярно использует программу SQL Server, может узнать для себя что-то новое, прочитав эту книгу (что относится даже к специалистам компании Microsoft из

штаб-квартиры компании в городе Редмонт, повседневно использующим программу SQL Server в своей работе).

Дэвид Кэмпбелл
Июнь 2003 года

Дэвид Кэмпбелл (David Campbell) был принят на работу в компанию Microsoft в 1994 году в качестве разработчика основной машины хранения данных для программы SQL Server компании Microsoft. С тех пор Дэвид неизменно входит в состав группы разработчиков SQL Server, а в настоящее время занимает должность руководителя производственного подразделения группы реляционного сервера.

Историческая перспектива

Трудно поверить, что уже прошло десять лет с тех пор, как я был приглашен на работу в компанию Microsoft, чтобы я мог участвовать в создании нового программного продукта SQL Server для операционной системы Windows NT. В то время я и не думал, что достигну того уровня, на котором сейчас работаю, или что вся наша группа сумеет создать столь совершенный программный продукт. До тех пор я работал в качестве программиста на языке C и разработчика баз данных для систем UNIX, и мне приходилось в основном сталкиваться с СУБД Oracle и Ingres. Ко времени поступления на работу в компанию Microsoft я считал, что эта компания занимается исключительно разработками для настольных компьютеров. Единственной базой данных для персонального компьютера, с которой мне когда-либо приходилось знакомиться, была база данных dBase. Поэтому естественно, что я, рассматривая предложение о поступлении на новую работу, относился к нему скептически. Разве сможет компания Microsoft когда-либо создать программный продукт, который будет способен конкурировать с программами, завоевавшими колоссальный авторитет в индустрии баз данных? К счастью для меня, Андреа Стоппани (Andrea Stoppani), занимавший в 1993 году должность директора службы SQL Support компании Microsoft, сумел убедить меня, что не только компания Microsoft достигнет успеха в разработке программы SQL Server, но и сам я добьюсь завидной карьеры в этой компании, поскольку буду иметь невиданные раньше возможности учиться самому, учить других, заниматься отладкой и разбираться в самых глубоких внутренних нюансах машины реляционной базы данных.

Прошло десять лет, и данное мне обещание исполнилось. Занимая должность инженера-испытателя в компании Microsoft, я действительно получил возможность учить и учиться, заниматься отладкой и разбираться во внутреннем устройстве той реляционной машины, которая является приводным механизмом для программы SQL Server. Поскольку я обладал приобретенными таким образом глубокими знаниями, меня регулярно просили давать консультации, сообщать свое мнение и предоставлять рекомендации группе разработчиков, особенно после появления каждого нового выпуска. В течение всех этих лет я был свидетелем эволюционного и революционного развития программного продукта SQL Server, начиная с того начального периода поддержки версии SQL Server 4.20 для операционной системы OS/2, когда заказчики, работающие на однопроцессорных компьютерах, могли использовать лишь ограниченный объем оперативной памяти на 16 Мбайт, и заканчивая версией SQL Server 2000 Enterprise Server уровня предприятия, которая работает на компьютере с 64-битовым процессором и оперативной памятью 512 Гбайт, устанавливая все новые и новые рекорды при выполнении тестов TPC.

Развитие самой машины базы данных, которое началось с создания самого раннего прототипа, происходило буквально на моих глазах. Машина хранения и процессор

запросов для всех версий, начиная с SQL Server 4.20 и заканчивая SQL Server 6.5, были основаны на оригинальном архитектурном проекте, который был взят из перенесенной в операционную систему Windows машины базы данных Sybase, разработанной для операционной системы OS/2. В течение этих лет произошли заметные изменения и были сделаны важные дополнения, благодаря которым машина базы данных стала работать гораздо быстрее и превратилась в надежную и доступную платформу для многих пользователей, стремящихся развертывать на своих предприятиях системы баз данных. Однако все усилия, предпринимавшиеся в рамках указанного подхода, в конечном итоге достигли своих пределов. Именно поэтому выпуск версии SQL Server 7.0 оказался столь важным. Компания Microsoft привлекла к работе некоторых ведущих разработчиков в индустрии баз данных для того, чтобы они приняли участие в проектировании и реализации новой архитектуры для машины баз данных и создали фундамент, на котором будут основаны все новые и новые программные продукты в течение многих последующих лет.

Но модернизация и эволюция распространялись не только на саму машину SQL Server. Нельзя отрицать то, что в первые годы развития программы SQL Server основные усилия разработчиков были направлены на усовершенствование машины базы данных, поскольку и сам этот программный продукт в основном формировался вокруг этой машины – программы `sqlservr.exe`. В тот период разработчики приложений сосредоточивали свои усилия, главным образом, на создании приложений с использованием библиотеки DB-Library на языках Visual Basic или C, обеспечивая обмен данными с помощью именованных каналов или протоколов IPX/SPX. В те дни Кайл Гейгер (Kyle Geiger) еще только приступал к реализации интерфейса ODBC на своем компьютере. А в настоящее время инженерам службы поддержки Microsoft чаще всего приходится сталкиваться с многоуровневыми приложениями на основе Web для баз данных, которые поддерживают оперативные приложения для розничных предприятий с многочисленными пользователями, обменивающимися данными по протоколам TCP/IP. Поэтому стремление обеспечить дальнейшее развитие "только машины базы данных" стало неоправданным. Программный продукт SQL Server развивается не только вглубь, но и вширь, охватывая богатую инфраструктуру служб данных, в том числе Multi-Server Job Scheduling (Многосерверное планирование заданий), Data Replication (Репликация данных), XML, Data Transformation Services (Службы преобразования данных) и Notification Services (Службы рассылки извещений).

Размышляя над тем, какие изменения были внесены в машину базы данных SQL Server и какие основные дополнения способствовали расширению возможностей столь популярного программного продукта, я вспоминаю о том, что пользователи этой программы и служащие компании Microsoft, работающие в другой области, часто задавали мне такие вопросы: "Как больше узнать о тех скрытых возможностях, благодаря которым программа SQL Server становится такой мощной? Как получить знания на уровне эксперта об особенностях внутреннего устройства машины SQL Server, чтобы добиться наибольших успехов в работе с этим программным продуктом?". Я всегда отвечал на эти вопросы следующим образом: "Рассматривайте эту программу с точки зрения программиста". Точнее, на эти

вопросы следует отвечать так: "Рассматривайте эту программу с точки зрения программиста, работающего в операционной системе Windows".

Я понял важность приобретения твердых знаний основ технологии, которая используется машиной SQL Server для выполнения работы. К этому относится целый ряд средств программирования, реализованных в операционной системе Windows, таких как процессы, потоки, синхронизация, асинхронный ввод-вывод, динамически загружаемые библиотеки, виртуальная память, сетевой обмен данными и технология COM. Проходят годы, появляются все новые и новые выпуски программы SQL Server, но изучение указанных средств остается существенно важным для понимания внутреннего устройства этого программного продукта. Чтобы изучить программные средства SQL Server, недостаточно просто читать документацию с их описанием. Необходимо применять полученные знания и познавать весь смысл рассматриваемых концепций в реальной действительности. Например, недостаточно прочитать о том, какую структуру имеют средства обработки исключительных ситуаций; необходимо также понять, почему функции обработки исключительных ситуаций стали столь важным компонентом, используемым в машине SQL Server. Часть I настоящей книги поможет читателю не только получить глубокие знания, но и достичь понимания основ программирования для операционной системы Windows. Но недостаточно просто прочитать главы этой книги. Проработайте все примеры и убедитесь в том, что вы способны сформулировать ответы на все вопросы, приведенные в каждой главе. Овладев твердыми знаниями всех рассматриваемых понятий, вы приобретете надежный фундамент и концептуальную основу для изучения части II этой книги, в которой рассматривается внутреннее устройство основных компонентов, составляющих машину SQL Server. Вооружившись этой информацией, вы сможете расширить и углубить свои знания, изучая технологии, описанные в части III, которые дополняют возможности самой машины SQL Server и позволяют пользователю получить в свое распоряжение тот программный продукт, обладающий полным набором служб баз данных, в который превратилась программа SQL Server.

СУБД SQL Server постепенно развилась до такой степени, что стала основой технологии и ведущей силой в индустрии баз данных. Одним из важных показателей реального успеха в области создания этого программного продукта является большое количество посвященных ему высококачественных книг. Когда я впервые приступил к работе в компании Microsoft в 1993 году, в продаже не было книг по Microsoft SQL Server (и только в следующем году была выпущена первая книга). Сегодня в Web и в книжных магазинах можно найти множество книг, посвященных этому программному продукту, в которых рассматриваются не только основные его возможности, но и такие сложные темы, как настройка производительности, администрирование базы данных, поддержка XML и т.д. А окончательным доказательством успеха данного программного продукта является появление настоящей книги. В этой книге автор стремится добиться широкого распространения знаний о наиболее важных темах, касающихся программы SQL Server, для того чтобы достигнутый им уровень понимания стал достоянием разработчиков в масштабах всего мира. Знания становятся силой, поэтому и данная книга становится в руках

пользователей, разработчиков и администраторов SQL Server могучей силой, позволяющей добиться максимальной отдачи от этого программного продукта.

Боб Уард
Июнь 2003 года

Боб Уард (Bob Ward) поступил на работу в компанию Microsoft в 1993 году в качестве инженера службы поддержки Microsoft SQL Server. В настоящее время Боб занимает должность инженера-испытателя в подразделении SQL Server Support.

Предисловие автора

Я вырос на ферме в самом сердце Соединенных Штатов. С восьми лет я жил в небольшом деревянном доме в сельском районе штата Оклахома со своими родителями и сестрами, и расстался с этим домом только после того, как отправился учиться в колледж. Я находился в такой же среде, как и все обычные деревенские мальчики, со всеми сопутствующими этому образу жизни факторами: здоровое питание, обилие приключений и умение полагаться на самого себя.

Я появился на свет, когда водопровод и электричество получили повсеместное распространение даже в сельской местности штата Оклахома, поэтому не могу рассказать о себе страшных историй о том, что я не пользовался основными бытовыми удобствами, жил в сарае или ходил по немытым полам. Но до тех пор, пока я не пошел в школу, мне приходилось ежедневно доить пять коров; я сгребал сено летом и заготавливал дрова осенью. Вместе с сестрами я помогал родителям сажать овощи в огороде весной и собирать урожай летом. Я кормил цыплят, свиней и других домашних животных, присутствовал при появлении на свет телят и выращивал бычков на убой.

Причины, побудившие моих родителей переехать в сельскую местность, так и остались для меня не совсем ясными. Мой отец работал инженером в правительственном учреждении, что позволяло нашей семье вести комфортное существование в одном из пригородов большого города. Поэтому, на первый взгляд, ничто не предвещало, что у родителей возникнет необходимость совершить такой важный переезд. Но когда мне исполнилось восемь лет, родители собрали вещи и отправились вместе с нами, своими детьми, навстречу к той жизни, о которой мы, обитатели города, даже никогда и не помышляли. До того времени я никогда не видел корову, кроме как по телевизору, и тем более никогда не ездил на лошади. Мы расстались с прошлым, переехали ближе к земле, и все в нашей жизни изменилось.

Я до сих пор помню, как моя мать усадила нас рядом с собой за день до отъезда и сообщила, что мы покидаем город, чтобы получить шанс узнать новые и замечательные стороны жизни, приобрести более широкие перспективы, увидеть вещи другими глазами, словом, получить то, что недоступно большинству горожан. Она с непоколебимым оптимизмом ответила на нашу лавину слез и жалоб и заверила, что у нас не только все получится на новом месте, но и фактически все станет еще лучше. Она считала, что любой опыт дает шанс узнать что-то новое. Подобно Генри Торо, она хотела познать самую суть жизни. Она давно решила, что ее дети должны расти в здоровых условиях фермерского хозяйства, и, добиваясь этого, сделала все, что было в ее силах. Только повзрослев, я понял, для чего был затеян весь этот переезд.

Без сомнения, переезд в сельскую местность предоставил мне замечательную возможность постигать уроки повседневности. Жизнь предстала передо мной в самом естественном виде; в реках, в лесах и повсюду вокруг меня так явно совершался цикл рождения и смерти. Для мальчика восьми лет не было лучшего места, чтобы действительно чему-то научиться. Я пробирался сквозь густой лес, спускался на плоту вниз по ручью, ловил рыбу в пруду, срывал свежие плоды с деревьев и съедал их немытыми; ни одного дня не проходило без приключений,

и это было самое лучшее время для того, чтобы больше узнать о наблюдаемом мною мире. Я получил все уроки, которые должна была преподать мне жизнь, и в таких обстоятельствах, которые, скорее всего, так и не сложились бы, если бы мы остались в городе, и я всегда буду за это благодарен.

Моя мать прилагала большие усилия, чтобы наше образование не страдало из-за того, что нас пересадили с городской клумбы на сельскую грядку. Она завела личную библиотеку для каждого из нас и постаралась привить нам такую же любовь к чтению, какой она сама отличалась всю жизнь. После того как окончилась неудачей попытка добиться от властей округа, чтобы они открыли где-то рядом с нами новую библиотеку, она уговорила администрацию действующей библиотеки развернуть программу обслуживания читателей с помощью передвижной библиотеки. В нашей жизни появился праздник — День передвижной библиотеки. В долгожданный день веселая толпа малышей наперегонки мчалась к старой сельской церкви, у которой останавливалась передвижная библиотека. В специально приспособленном для этого автомобиле для активного отдыха стены были заставлены книгами, а прохлада кондиционированного воздуха становилась для нас замечательной передышкой после дня, проведенного под горячим солнцем штата Оклахома. Мы оставались в библиотеке до тех пор, пока нас не выгоняли, и уносили каждый раз охапку книг, чтобы вернуть в следующий День передвижной библиотеки.

Именно в то время я впервые начал постигать тайны жизни самостоятельно. Я хотел знать, откуда все произошло, как все устроено. Я читал буквально ненасытно; мой восьмилетний мозг жадно поглощал каждую научно-популярную книгу и книгу по электронике, какую я мог достать. Я хотел раскрыть секреты всего, что меня окружает; я хотел знать, в чем состоит основная суть всего происходящего. Я хотел знать, как устроена жизнь, как устроен мир, как устроено все вокруг. Я хотел понять буквально все, что заставляет мир кружиться вокруг нас.

Именно в те дни я впервые прочитал книги по физике и понял, что напал на след новых для себя открытий. Я нашел путь, который мог привести меня к пониманию того, к чему я стремился. Я читал о всемирном тяготении и магнетизме, о сильных и слабых частицах. Я сформировал в своем мозгу модель того, как работает вселенная. Я обрел понимание (хотя и не столь совершенное, как хотелось бы) того, как все в мире взаимодействует, как устроен мир и какова воспринимаемая мной реальность, которая сводится к немногим фундаментальным понятиям, которые я мог легко наблюдать в действии в окружающем меня естественном мире. Я пришел к пониманию основной "системы" бытия, той совокупности идей, которая позволяла объяснить смысл почти всего, что существовало вокруг меня. Внезапно все, что происходило в нашей сельской местности, в самой природе и во всем том мире, который меня окружал, стало приобретать смысл.

С того времени я подходил к изучению почти всего, с чем мне приходилось сталкиваться, с таким же жадным любопытством. Я хотел знать, как все работает, и стремился познать новое во всей его полноте. Я упорно работал над тем, чтобы уйти от поверхностных объяснений или неполного понимания. До сих пор я стремлюсь досконально узнать, как действует рассматриваемая мной система, как взаимодействуют и чем связаны ее составные части. Я считаю, что это — единственный реальный способ чем-то овладеть, по-настоящему понять суть вещей.

Этот философский подход лег в основу настоящей книги. Я написал эту книгу, чтобы воплотить в ней все полученные мною знания о программе SQL Server и о фундаментальных технологиях, на основе которых спроектирована эта программа, показать, как работают и взаимодействуют ее компоненты. Я написал данную книгу, поскольку получаю огромное удовольствие, изучая работу программы SQL Server. В предыдущих своих книгах я показал, как эксплуатировать и составлять программы для SQL Server, а в настоящей книге я подробно описал, как устроена программа SQL Server с точки зрения ее архитектуры. Завершив эту работу, я надеюсь, что сумею пробудить у читателя такое же восхищение, такую же любовь к технологии и всем компонентам SQL Server, какую вызывает у меня эта программа.

Я уверен, что дорога к настоящему овладению программой SQL Server или любой другой технологией лежит через исследование ее проекта. Безусловно, важно знать, как обеспечить практическое использование технологии, но эти знания начинаются с понимания того, как действует сама технология и для чего она предназначена. Глубокое изучение проекта программы SQL Server позволит вам стать более квалифицированным пользователем этой программы и достичь невиданных высот.

Я давно распрощался со светловолосым мальчиком, который бегал босиком по дорогам сельской местности штата Оклахома. Теперь я живу в городе, но воспоминания о сельской жизни меня не оставили. Я часто возвращаюсь мыслями к своим прогулкам по полям при лунном свете, вспоминаю о том, как ходил под открытым небом, восхищался всем, что было, и думал обо всем, что могло быть. Я все еще вспоминаю запах свежей люцерны, который доносится с вечерним ветерком, храню в душе безграничную радость, которую испытываешь, бегая стремглав по золотым холмам, не забываю, какое отстранение от всех жизненных забот происходит в этот яркий момент. Мне не хватает тех приключений и той самостоятельности в жизни, которые я когда-то испытывал. Я становлюсь задумчивым, услышав далекий крик кукушки; мне не хватает вечерних сумерек в лесу. Я до сих пор помню, как катался на надувной резиновой шине по пруду, и хочу снова почувствовать вкус свежего, спелого зерна кукурузы, оторванного от початка.

Еще раз хочу сказать, что я, хоть и уехал из сельской местности, так и не смог расстаться со своими воспоминаниями о ней. То же самое относится и к моему ненасытному желанию исследовать и понять все о мире, который меня окружает, и о тех вещах, которые пробуждают мой интерес. Хотя я уже не раз переезжал и время от времени менял работу, жажда приключений, побуждающая меня исследовать новые и неизвестные места, которую я обнаружил в себе в восьмилетнем возрасте, еще меня не покинула. С тех пор я провожу свою жизнь, совершая одно путешествие за другим, исследуя один за другим все новые и новые миры в надежде изучить все, что я способен познать. Меня еще не оставило желание выяснить о работе программы SQL Server буквально все, что возможно. Я работаю с данной технологией с 1990 года, но для меня еще многое в ней осталось нераскрытым. Поэтому я выражаю надежду, что читатель присоединится ко мне в стремлении проложить новый путь в познании этой технологии, изучая новые места и открывая еще невиданные возможности. И я надеюсь, что это путешествие понравится читателю так же, как и мне.

Кен Хендерсон
11 марта 2003 года

Благодарности

За каждой написанной и изданной мною книгой стоит буквально целая армия людей, которые помогли мне довести начатое дело до конца. Настоящая книга не является исключением. Прежде всего я хочу поблагодарить многих специалистов в области разработки и эксплуатации программного обеспечения SQL Server, которые ознакомились с данной книгой и высказали целый ряд конструктивных и полезных замечаний. Я особенно благодарен Грэгу Линвуду (Greg Linwood), Тони Роджерсону (Tony Rogerson), Аллену Митчеллу (Allan Mitchell), Даррену Грину (Darren Green), Эрону Бертрану (Aaron Bertrand), Т. К. Динешу (T. K. Dinesh) и Уэйну Снайдеру (Wayne Snyder). Без вашей помощи стало бы невозможным появление этой книги на свет в том виде, какой она имеет сейчас.

Я хотел бы также выразить признательность моим друзьям из компании Microsoft, которые прочитали книгу в рукописи и помогли улучшить ее в такой степени, что я просто не мог бы на это надеяться. Особенно весомым оказался вклад, сделанный Бартом Дунканом (Bart Duncan), Робертом Дорром (Robert Dorr), Китом Элмором (Keith Elmore), Бобом Уардом (Bob Ward), Дайаной Ларсен (Diane Larsen), Кристой Карпентьер (Christa Carpentiere), Диком Дивендорфом (Dick Dievendorff) и теми сотрудниками группы разработчиков SQL Server, которые подготовили свою рецензию на книгу. Кроме того, я хотел бы поблагодарить Верна Амина (Vern Ameen) из компании Microsoft за то, что он первым поверил в меня и высказал мнение, что я действительно могу принести пользу компании Microsoft. Спасибо, Верн. Я всегда буду ценить вашу поддержку. Я очень признателен Рону Соукапу (Ron Soukup), бывшему сотруднику компании Microsoft и первому "создателю" программы SQL Server, за то, что он выразил мне свое доверие и поддержал мою работу. Рон, все решил тот разговор, который состоялся между нами вскоре после того, как вы оставили компанию. Еще раз благодарю вас за сказанное.

От себя лично я хотел бы поблагодарить моих друзей Джона Кокрена (John Cochran) и Лоррэйн Бодетт (Lorraine Beaudette), которые просмотрели отдельные части рукописи и порекомендовали мне продолжить работу, когда я был весьма склонен ее завершить. Ваша поддержка моей работы и ваша дружба, которая длится все эти годы, важны для меня больше, чем я могу выразить.

Мой друг Нейл Кой (Neil Coy), как всегда, был на протяжении всего процесса реализации проекта данной книги горячим сторонником ее создания и оставался моим твердым наставником. Я думаю, что все, кто когда-либо встречался с Нейлом, испытали его стимулирующее влияние. Я также, безусловно, нахожусь у него в долгу. Под благоприятным воздействием его идей и эта книга стала намного лучше, как и все мои предыдущие книги. В моей жизни есть несколько положительных факторов, от которых зависит многое. Одним из них является тесная дружба с Нейлом Коем. Нейл, вы – программист из программистов. Спасибо за все, чему вы научили меня за все эти прошедшие годы и продолжаете учить.

Я чувствовал бы себя просто неблагодарным должником, если бы не адресовал несколько слов признательности замечательным людям, сотрудникам издательства Addison-Wesley, благодаря которым стало возможным появление на свет этой книги. Мой редактор, Карен Геттмен (Karen Gettman), напряженно боролась за то, чтобы проект подготовки данной книги завершился успешно. Спасибо вам, Карен. Вы помогли мне не сойти с дистанции. Кроме того, невероятные усилия в осуществление данного проекта вложили Эмили Фрей (Emily Frey), Карен Хансен (Karin Hansen) и Керт Джонсон (Curt Johnson). Мне было также очень приятно работать с сотрудниками производственной группы — Элизабет Райен (Elizabeth Ryan) из компании Addison-Wesley, Ким Арни Малкэхи (Kim Arney Mulcahy), Моникой Грот Фаррар (Monica Groth Farrar) и Кристой Мэдоубрук (Chrysta Meadowbrooke). Я хочу выразить особую признательность Кристе. Криста, вы — лучший редактор, с которым мне когда-либо приходилось работать. Желаю вам и дальше поддерживать столь высокий уровень профессионализма. Я хотел бы также поблагодарить Картера Шанклина (Carter Shanklin), который впервые представил меня в издательстве Addison-Wesley, а также Майкла Слотера (Michael Slaughter) и Мэри О'Брайен (Mary O'Brien), редакторов моих предыдущих книг, во многом благодаря которым был достигнут успех при выпуске этих изданий.

Наконец, я хочу выразить глубокую признательность моей жене. Она сделала все, чтобы стало возможным появление на свет и этой, и всех других написанных мною книг. Она проявляла такую заботу о наших детях, что я почти не испытывал чувства вины за то, что встречался с ними лишь урывками, потому что на долгие часы задерживался на работе, подготавливая очередную книгу. Именно она дает мне ту крупицу поддержки, позволяющую немного легче справляться с трудностями, о которой говорил Трумен Капоте. И именно она придает хоть какой-то смысл моей работе, которой приходится заниматься с утра и до позднего вечера.

Введение

Автор написал эту книгу, чтобы глубже раскрыть устройство программы SQL Server. В данной книге автор хотел показать, что можно многое узнать об этом программном продукте и технологиях, на которых он основан, с использованием бесплатно предоставляемого отладчика, нескольких хорошо подобранных расширенных процедур и большого трудолюбия. Книга, лежащая перед глазами читателя, стала результатом этого эксперимента.

Две предыдущие книги автора, посвященные программе SQL Server, в большей степени сосредоточены на прагматических аспектах, касающихся этой программы. В частности, в этих книгах описано, как составлять программы для SQL Server и использовать на практике многочисленные средства указанного программного продукта. С другой стороны, настоящая книга в соответствии с ее названием в большей степени направлена на изложение архитектурного проекта самого программного продукта. В данной книге больше внимания уделено вопросам технической реализации программы SQL Server, чем темам, касающимся использования этой программы. По мнению автора, лучшее понимание работы программы SQL Server позволяет успешнее применять ее на практике. Благодаря этому читатель сможет лучше эксплуатировать этот программный продукт и более продуктивно использовать его средства в своей работе, поскольку будет иметь глубокое понимание того, как функционируют эти средства и для чего они предназначены.

Оперативная документация Books Online

В основе замысла настоящей книги, как и двух предыдущих книг автора, лежало стремление избегать ненужного повторения информации, изложенной в оперативной документации Books Online. Для реализации этого замысла потребовалось исключить несколько тем, которые читатель мог рассчитывать найти в книге, подобной этой. Например, автор первоначально планировал ввести в книгу обзорную главу, которая содержала бы описание структурной компоновки рассматриваемого программного продукта с точки зрения основных его составляющих. Автор также намеревался предусмотреть отдельную главу с описанием устройства машины хранения. Но после повторного ознакомления с изложением этих вопросов в оперативной документации Books Online (см. тему *SQL Server Architecture Overview* и связанные с ней подтемы) и в других источниках автор пришел к выводу, что не сможет существенно дополнить это изложение.

Цель автора состоит не в том, чтобы заполнить страницы настоящей книги информацией, которая и так является легко доступной; автор решил продолжить работу с того, на чем заканчивается описание, приведенное в документации программного продукта (а также в других книгах и фирменных материалах),

и перевести обсуждение на более высокий уровень. Поэтому предполагается, что читатель уже ознакомился с оперативной документацией Books Online и изучил основные понятия, касающиеся программы SQL Server.

Отладчик WinDbg

Значительная часть материала данной книги основана на использовании символического отладчика WinDbg компании Microsoft, который свободно предоставляется для бесплатной загрузки. У читателя может сразу же возникнуть вопрос, почему для изучения программы SQL Server требуется отладчик. В конечном итоге, ведь мы вряд ли собираемся "отлаживать" программу SQL Server, а также, безусловно, не имеем необходимого для этого исходного кода, поэтому не можем заниматься пошаговым просмотром кода, как обычно действуют программисты, использующие отладчик.

Причина, по которой в данной книге применяется отладчик, состоит в том, что он позволяет ознакомиться с работой действующих процессов так полно, как ни одно иное инструментальное средство. Отладчик позволяет узнать, какие потоки в настоящее время функционируют внутри процесса, ознакомиться с текущим содержимым стеков вызовов потоков, определить состояние виртуальной памяти и динамических областей памяти в процессе, а также получить многие другие важные данные, относящиеся ко всему процессу и к отдельным потокам. Отладчик позволяет задавать точки останова, просматривать регистры и наблюдать за тем, как происходит загрузка библиотек DLL в *самом* процессе или перебазирование этих библиотек в операционной системе Windows. Отладчик позволяет останавливать выполнение, выводить на внешнее устройство информацию о содержимом областей памяти, сохранять и восстанавливать полные данные о состоянии процесса. Короче говоря, отладчик служит в качестве своего рода "рентгеновского аппарата" – инструмента, позволяющего заглянуть внутрь процесса и узнать, что в нем действительно происходит. В данном случае интересующим нас объектом является программа SQL Server, но основные навыки отладки, полученные при изучении данной книги, могут помочь при исследовании любого другого приложения Win32. Одна из основных целей этой книги состоит в том, чтобы вооружить читателя некоторыми наиболее важными навыками составления и отладки программ, чтобы дать ему возможность продолжить освоение программы SQL Server самостоятельно.

Желая проникнуть внутрь программного продукта и понять, как он работает, вы просто обязаны пользоваться отладчиком. Попытка понять внутреннее функционирование какой-то технологии только путем чтения сведений об этой технологии, изложенных в книгах или фирменных материалах, равносильна попытке изучить чужую страну, не посещая ее. Дело в том, что невозможно ничем заменить реальное знакомство с изучаемой предметной областью.

Отладчик WinDbg, доступный для бесплатной загрузки с Web-узла Microsoft, обладает необходимыми возможностями и является относительно удобным в использовании, поэтому в наибольшей степени подходит для достижения описанных выше целей. Кроме того, WinDbg – это символический отладчик,

позволяющий использовать отладочную информацию, которая входит в поставку программы SQL Server и предоставляется для общего пользования по Internet, поэтому указанный отладчик вполне подходит для изучения внутреннего функционирования и структурной компоновки рассматриваемого программного продукта.

Основные понятия

Настоящая книга позволяет достичь глубокого понимания технологий, лежащих в основе программы SQL Server, и лучше разобраться в том, как работает эта программа. Поэтому несколько глав данной книги посвящено изложению основных сведений, относящихся к процессам и потокам, управлению памятью, вводу-выводу Windows, организации сетевого взаимодействия и к некоторым другим темам. Для тех, кому еще не приходилось глубоко изучать работу программы SQL Server, указанные темы могут показаться в лучшем случае лишь косвенно касающимися этой программы. В конце концов, неужели так нужно знать об асинхронном вводе-выводе, чтобы разобраться в работе программы SQL Server? Тем не менее необходимо знать кое-что не только об этом, но и о других фундаментальных технологиях, на которых основана программа SQL Server, для того, чтобы иметь базовый запас сведений и приобрести глубокие познания в том, как работает сам этот программный продукт. Необходимо владеть фундаментальными понятиями Windows, на которых основана работа такого сложного приложения Windows, как SQL Server, по тем же причинам, по которым студент-медик должен понимать основы биологии, прежде чем поступить в медицинский институт. Без таких фундаментальных знаний вы утратите перспективу и базу, необходимые для полного понимания и освоения более сложных понятий, которые пытаетесь изучить. Люди учатся с помощью ассоциативного мышления, связывая новые сведения с уже приобретенными знаниями. Не имея твердого понимания основ проектирования приложений Windows, вы не сможете приобрести базовые знания, необходимые для формирования систематических ассоциаций, позволяющих познать функционирование сложных приложений Windows, подобных SQL Server.

Безусловно, вы всегда сумеете получить поверхностное представление о том, как работает программа SQL Server (например, прочитав, что в этой программе используется ввод-вывод со сборкой-разборкой) без реального понимания того, что именно подразумевается под этим описанием. Если же вы действительно хотите полностью овладеть этим программным продуктом (т.е., если вы действительно хотите изучить его буквально до мельчайших подробностей), то должны иметь определенное понимание того, на основе каких технологий реализован этот программный продукт. Узнав о том, как функционирует ввод-вывод со сборкой-разборкой, вы немедленно обретете понимание того, почему такие средства ввода-вывода используются в программе SQL Server и как они способствуют повышению производительности. Такое же утверждение остается справедливым применительно к виртуальной памяти, синхронизации потоков, организации сетевого взаимодействия и ко многим другим основополагающим темам, рассматриваемым в данной книге. Эти темы не только непосредственно

касаются программы SQL Server, но и настолько важны, что без их усвоения невозможно полностью понять работу программы SQL Server. Не имея полного представления о тех фундаментальных технологиях, на которых основана работа программы SQL Server (процессы и потоки Win32, виртуальная память, асинхронный ввод-вывод, технология COM, организация сетевого взаимодействия Windows и многие другие), вы не сможете иметь в своем распоряжении инструментальные средства и знания, позволяющие полностью понять, как работает этот программный продукт, или полностью освоить эксплуатацию этого программного продукта.

Автор вполне отдает себе отчет в том, что не каждый читатель стремится глубоко изучить технологии Windows и API-интерфейсы, на которых основаны функциональные возможности программы SQL Server. Это не означает, что при изучении данной книги у него возникнут какие-либо сложности. Если вас не интересуют подробные сведения об API-интерфейсах Win32, о том, как их использовать, и о том, как эти интерфейсы обычно применяются в таких приложениях, как SQL Server, просто пропустите часть I, "Основные сведения", данной книги. В остальных частях книги есть еще очень много полезной информации, и, чтобы воспользоваться этой информацией, не обязательно изучать каждый нюанс каждого API-интерфейса.

Практические рекомендации

Автор пытался сделать все от него зависящее для описания структурных особенностей устройства различных компонентов, на которых основана программа SQL Server, и в то же время не упустить из виду необходимость обеспечения практического использования этой программы. Излагая теоретические сведения, автор в душе всегда оставался программистом-практиком, поэтому в настоящей книге можно найти весьма значительный объем информации практического назначения. Достаточно отметить, что автор подготовил для комплектации компакт-диска, прилагаемого к данной книге, около 900 файлов исходного кода. Таким образом, общее количество файлов намного увеличилось даже по сравнению с двумя последними книгами автора, которые, как уже было сказано, были в значительной степени направлены на обеспечение практического использования программы SQL Server.

Стараясь лучше раскрыть центральную тему трех своих книг, посвященных программе SQL Server (а именно, обеспечение наиболее производительной эксплуатации данного программного продукта), автор стремился достичь наиболее обобщенного уровня описания структурных компонентов, составляющих рассматриваемый программный продукт, и вместе с тем не оставить без внимания интересы основной части своих читателей, интересующихся прикладными аспектами. Автор надеется, что любой читатель этой книги найдет в ней то, что искал, независимо от того, ожидал ли он получить в свое распоряжение наглядные примеры кода и практическую информацию, или хотел лучше понять, как работает программа SQL Server, чтобы повысить эффективность ее использования.

Широта изложения

В настоящей книге рассматривается широкий перечень средств и технологий, лежащих в основе программного продукта SQL Server. Эта книга не ограничивается просто изложением функциональных возможностей, предоставляемых программой `sqlservr.exe`; в ней предпринята попытка описать весь программный продукт. По мнению автора, книга, предназначенная для описания внутреннего функционирования и структурного проекта такого сложного программного продукта, как SQL Server, должна охватывать весь этот программный продукт, а не только те функциональные средства, которые были давно реализованы в основном исполняемом файле или оставались неизменными из года в год. Перечень функциональных возможностей программы SQL Server отнюдь не ограничивается функциями, поддерживаемыми всего лишь единственным исполняемым файлом. Вполне можно допустить, что версии SQL Server, предшествующие версии 7.0 этого программного продукта, можно было рассматривать с точки зрения лишь функциональных возможностей, предоставляемых основным исполняемым файлом, но такое положение изменилось навсегда, причем это произошло много лет тому назад. Рассматриваемый программный продукт достиг полной зрелости, и с каждым новым выпуском спектр его возможностей существенно расширяется.

Поэтому автор не планировал создать эту книгу как посвященную исключительно программе `sqlservr.exe`. В данной книге рассматривается вся комплектация СУБД SQL Server и показано, как работают и взаимодействуют многочисленные составные части этого сложного приложения. Поэтому в настоящей книге можно найти описание технологий, которые при поверхностном подходе могут рассматриваться как относящиеся к программе SQL Server лишь косвенно, например, Full-Text Search (Полнотекстовый поиск), Notification Services (Службы рассылки извещений) и SQLXML. В этой книге рассматривается система репликации, службы DTS, а также множество других технологий SQL Server, не реализованных в основном исполняемом файле SQL Server. Из-за реальных ограничений автор не смог досконально описать каждое средство рассматриваемого программного продукта или даже привести столь полные сведения, какие ему хотелось бы. Для написания подобной книги потребовалось бы десять лет, а по объему эта книга достигла бы 5 тысяч страниц. Тем не менее автор пытался выдерживать компромисс между требованием изложить необходимые сведения настолько глубоко, насколько читатели привыкли ожидать от книг автора, и стремлением охватить достаточно широкий перечень средств и технологий, чтобы можно было полностью понять общий проект и структуру программы SQL Server как программного продукта.

Применение языка C++

Автор отнюдь не сомневается в том, что многие пользователи программы SQL Server более уверенно чувствуют себя, работая на языке Visual Basic, чем на любом диалекте языка C или C++. Тем не менее автор использовал языки C и C++ для описания основ программирования для Windows и изложения многих других тем в данной книге по ряду причин.

Во-первых, на языке C написан сам API-интерфейс Win32. Безусловно, описанию способов доступа к API-интерфейсу Win32 из программ на языке VB посвящены целые книги, но опыт автора показывает, что в некоторых обстоятельствах доступ с помощью языка VB к некоторым функциям API-интерфейса становится сложным или вообще невозможным, в зависимости от рассматриваемой функции API-интерфейса. Дело в том, что сам API-интерфейс Win32 был первоначально написан на языке C, поэтому языки C и C++ предоставляют самый простой и непосредственный способ доступа к этим функциям. Любые другие подходы (независимо от того, основаны ли эти подходы на использовании VB, Delphi, C# или какого-то другого языка или инструментального средства) вводят дополнительный уровень абстракции, который скрывает суть излагаемого материала за ненужными подробностями.

Во-вторых, автор использует язык C++, поскольку считает, что этот язык не столь трудоемок в изучении, и что основная часть программистов, работающих на языке VB, более чем способны приобрести основные навыки программирования на языке C++ и могут успешно разбираться в коде C++, даже если не верят в это сами. Создается впечатление, что в сообществе пользователей VB просто существует естественное отторжение или даже боязнь всего, что касается языка C++. Но автор считает, что такое отрицательное отношение к языку C++ по большей части является необоснованным и создает перед специалистами ненужные препоны, не позволяющие по-настоящему понять работу самой операционной системы Windows и таких сложных приложений для Windows, как SQL Server. Автор хочет дать совет — даже если вы не знаете язык C++ и чувствуете себя беспомощным, читая код C++, не поддавайтесь панике. Прорабатывайте примеры данной книги, выполняйте приведенные в ней указания и следите за тем, куда приведут вас проведенные исследования. Если это возможно, ознакомьтесь с книгой начального уровня, посвященной языку C++. Вы вполне можете обнаружить, что этот язык не настолько сложен в изучении, как вам казалось на первых порах, и полученный опыт принесет вам пользу (возможно, даже весьма значительную).

Как уже было сказано, C++ далеко не является единственным языком, используемым в настоящей книге. Автор исходил из того, что нет такого одного языка, который использовался бы всеми, поэтому пытался поддерживать в данной книге определенный баланс между используемыми языковыми инструментальными средствами. Значительная часть примеров кода, приведенных в этой книге, разработана с использованием некоторой разновидности языка Visual Basic — VB6, VBScript или VB.NET. Например, в главе 15, "Средства ODSOLE", автор показал, как создавать COM-объекты на языке VB6. В главе 18, "Технологии SQLXML", описано, как получить доступ к средствам SQLXML с помощью языка VBScript. А в главе 19, "Службы рассылки извещений", показано, как реализовать приложение управления подпиской с помощью языка VB.NET. Много информации приведено также в отношении C#, Delphi, командных файлов и даже дано несколько фрагментов, касающихся языка ассемблера. Кроме того, разумеется, данная книга содержит большой объем кода на языке Transact-SQL (сокращенно, T-SQL). Поэтому любой читатель найдет в этой книге программы, которые его заинтересуют, независимо от того, с каким языком (языками) он предпочитает работать.

Применение языка Visual C++ 6.0

Многим специалистам принятое автором решение об использовании версии языка Visual C++ 6.0 для подготовки большинства примеров кода на языке C++ в настоящей книге может показаться не совсем оправданным. Но автор предпочел версию VC6, а не версию Visual Studio .NET по двум причинам: во-первых, версия VC6 была выпущена намного раньше по сравнению с другой версией, поэтому получила гораздо более широкое распространение, и, во-вторых, в версии интегрированной среды разработки Visual Studio .NET (выпуска и 2001 года, и 2003 года) проекты VC6 автоматически преобразуются в проекты новой версии сразу после их первой загрузки в среду. Поэтому проекты C++, которые находятся на компакт-диске, прилагаемом к данной книге, должны открываться вполне успешно, независимо от того, применяется ли для работы с ними интегрированная среда разработки Visual Studio 6 или Visual Studio .NET. Читатель имеет возможность компилировать и вызывать эти проекты на выполнение без каких-либо затруднений. Кроме того, описывая в данной книге такие основные понятия Windows, как синхронизация потоков и управление памятью, автор не использует каких-либо средств, относящихся лишь к конкретной версии, поэтому применение Visual Studio .NET вместо VC6 не дало бы никаких преимуществ.

Разделы с определениями терминов и вопросами для самопроверки

Читатели предыдущих книг автора могли заметить, что в настоящей книге появилось значительное количество "вспомогательного" материала, особенно в начальных главах. В частности, заметным нововведением являются определения терминов, которые предшествуют изложению основной темы в конкретной главе, а также вопросы для самопроверки, приведенные после завершения каждой темы. Читателям не следует беспокоиться — автор по-прежнему считает, что в книге не должно быть материала, предназначенного просто для увеличения ее объема, и прилагает значительные усилия, чтобы избежать появления в книге ненужных снимков с экрана, резюме и прочих разделов, которые обычно используются в специальной литературе для наращивания количества страниц.

Безусловно, лично сам автор предпочитает не накапливать таблицы определений терминов, вопросы для самопроверки и тому подобные материалы, поэтому не включал их в свои предыдущие книги, но все большее количество читателей обращалось к нему с просьбами о введении подобных дополнений для того, чтобы выпускаемые автором книги стали более подходящими для использования в качестве учебной литературы. Как оказалось, некоторые из предыдущих книг автора регулярно используются при обучении, даже несмотря на то, что они не были для этого предназначены. Поэтому автор в конечном итоге решил что-то предпринять в указанном направлении. Читатели, считающие, что подобные разделы не слишком нужны, вправе просто их пропустить. Все сведения, при-

веденные в определениях терминов, можно также найти в тексте глав, поэтому, не читая разделы с определениями терминов, вы ничего не потеряете. Несмотря на сказанное выше, читатель может заметить, что ознакомление с основными сведениями, касающимися некоторых терминов и понятий, еще до того, как будет приведено более подробное описание темы, может оказаться полезным. В действительности решение о том, знакомиться ли с терминами заранее или изучать их в связном изложении, — дело личного вкуса.

Автор намеренно не включил в книгу ответы на задания из разделов "Вопросы для самопроверки", чтобы в дальнейшем узнать от читателей, насколько широко используются эти разделы. Еще раз отметим, что появление в книге таких разделов — это способ адаптации к новым потребностям, цель которого состоит в создании книги, более пригодной для использования в качестве учебной литературы. Автор может продолжить работу в этом направлении при подготовке будущих книг или отказаться от указанного подхода, в зависимости от того, насколько он окажется оправданным. Если вы хотите ознакомиться с ответами на задания из разделов "Вопросы для самопроверки", отправьте письмо по электронной почте по адресу khen@khen.com, и такие ответы будут вам предоставлены.

Версии SQL Server

Настоящая книга посвящена одной из новейших версий SQL Server, которая в настоящее время получила наиболее широкое распространение, — версии SQL Server 2000. Можно считать, что все упоминания программы SQL Server, приведенные в данной книге, безусловно, относятся к версии SQL Server 2000, а также, возможно, к другим версиям. Сам автор редко упоминает конкретную версию SQL Server, поскольку считает, что из-за этого текст книги становится более трудным для восприятия. Несмотря на сказанное выше, если у читателя возникнут какие-либо сомнения, следует считать, что рассматриваемый материал относится к версии SQL Server 2000.

Различия между квалифицированным программистом и квалифицированным пользователем

Безусловно, в настоящей книге приведен весьма значительный объем кода и широко обсуждаются вопросы, относящиеся к функционированию программного обеспечения, поэтому на первый взгляд может показаться, что автор написал эту книгу не для того, чтобы подготовить читателя для квалифицированной работы в качестве пользователя программы SQL Server, а чтобы сделать из него квалифицированного программиста. Но это предположение весьма далеко от истины. Чтобы рассеять сомнения, рассмотрим, прежде всего, каковы отличительные особенности квалифицированного программиста.

Во-первых, квалифицированным программистом, скорее всего, становится тот, кто зарабатывает себе на жизнь написанием программ. Никто не сможет приобрести навыки программирования на уровне эксперта и постоянно поддерживать свою высокую квалификацию, просто изучая код, написанный другими людьми, или читая книги по программированию. Необходимо самому заниматься этим делом и выполнять практическую работу, причем указанная деятельность должна быть постоянной. Технология непрерывно изменяется, а сфера разработки программного обеспечения развивается настолько быстро, что для программиста просто нет иного способа поддерживать свою квалификацию, чем каждый день заниматься программированием.

Во-вторых, квалифицированным программистом нельзя назвать того, кто просто быстро набирает исходный код. Автору довелось работать с одним человеком, который как-то заявил, что самой важной отличительной особенностью специалиста в области программного обеспечения являются великолепные навыки работы на клавиатуре! Это утверждение действительно показалось автору просто смехотворным, поскольку навыки специалиста по программированию ничего не имеют общего с навыками работы на клавиатуре; автору приходилось встречаться с опытными программистами, которые набирали текст на клавиатуре буквально одним пальцем. Приведенное выше утверждение напомнило автору то, что сказал Трумен Капоте[1], когда ему задали вопрос о том, как он оценивает произведения Джека Керуака: "Он не писатель, а просто машинистка". Так же, как талант писателя не сводится к умению работать на машинке, так и квалификация программиста не ограничивается умением быстро вводить текст на клавиатуре. Если человек научился набивать огромные объемы исходного кода, это отнюдь не означает, что он стал квалифицированным программистом. В действительности характерной особенностью программ, составленных специалистами в области программирования, является краткость и эффективность, благодаря чему такие программы часто выполняют потрясающий объем работы с помощью невероятно малого объема кода. Весь секрет состоит не в том, чтобы написать большую программу; важнее всего создать хорошую программу. Здесь решающим является качество, а не количество.

В-третьих, квалифицированный программист имеет всестороннюю подготовку. Это означает, что он знает много языков программирования и работает во многих операционных системах и на разных платформах. Он не использует один язык программирования, забывая о других, какая бы проблема перед ним не стояла. Он знает, какой инструмент лучше всего подходит для выполнения стоящего перед ним задания, постоянно стремится расширить свой кругозор и все глубже овладеть наукой и искусством программирования для компьютеров. Квалифицированный программист – это не тот человек, который знает "ничего обо всем и все ни о чем". Он должен обладать навыками на уровне эксперта сразу в нескольких областях.

В-четвертых, квалифицированный программист не только мастерски владеет языками программирования, но и глубоко изучает среду операционной системы и фундаментальные технологии, используемые в работе. Он знает, что недостаточно просто владеть языком, с помощью которого ему довелось работать над

[1] Truman Capote. Цитата приведена из публикации в New Republic, Feb. 9, 1959.

конкретным проектом; он должен также глубоко понимать функционирование операционной системы и тех основополагающих компонентов, с помощью которых он создает приложения. Работает ли он с помощью технологий COM или EJB, использует ли операционную систему Windows или Linux, квалифицированный программист понимает, что он обязан также обладать знаниями на уровне эксперта о той среде, в которой работает созданный им код, и о тех компонентах, на основе которых создан этот код, поскольку лишь в этом случае он сможет создать надежное, эффективное и расширяемое программное обеспечение.

В-пятых, квалифицированный программист всегда следит за развитием технологии программирования и за новейшими достижениями в области разработки программного обеспечения. Квалифицированный программист может рассказать вам, в чем состоит различие между декораторным шаблоном проекта (decorator design pattern) и фасадным шаблоном проекта (façade design pattern). С ним можно обсудить вопрос о том, почему организация программы на основе технологии COM предпочтительнее по сравнению с организацией, основанной на использовании библиотек DLL, и побеседовать о преимуществах инфраструктуры .NET Framework перед технологией COM. Он может высказать собственное мнение по поводу того, какое место занимают средства Java среди других ведущих средств программирования и в чем состоит сходство и различие языка Java в сравнении с другими языками программирования. В разговоре с ним можно употребить термин "рефакторинг" (реорганизация кода), не встретив в ответ недоумевающего взгляда, и он способен описать взаимосвязь между экстремальным программированием (eXtreme Programming) и аспектно-ориентированным программированием (Aspect-Oriented Programming). Квалифицированный программист может не сталкиваться с этими понятиями и технологиями в своей повседневной деятельности, но остается достаточно осведомленным в том, что происходит в той индустрии, в которой он работает, чтобы понять их концептуально, быть способным объяснить отношения между ними и достаточно содержательно их обсудить.

В-шестых, квалифицированный программист весьма начитан. Он знает, кто такой Мартин Фоулер (Martin Fowler). Он читает книги Кента Бека (Kent Beck) и хорошо знает, чем занимается Эрих Гамма (Erich Gamma). Он читает не только литературу с описанием конкретных технологий, но и книги, в которых разработка программного обеспечения рассматривается как научная дисциплина. Он читал работы Стива Макконнелла (Steve McConnell) и изучал труды Дональда Кнута (Donald Knuth). Он знает, кем является Джон Бентли (Jon Bentley), а также знаком с работами Брайена Кернигана (Brian Kernighan). Он глубоко изучил работы Гради Буча (Grady Booch) и читал книги Чарльза Петцольда (Charles Petzold). В наши дни, когда теория и практика программирования развиваются с такой невиданной скоростью, что стала весьма нелегкой задача освоиться в этой сфере деятельности и перевести свои знания в область практического использования, никто не может считать себя достаточно хорошо подготовленным или оставаться слишком долго без дополнительных усилий в курсе всех новейших достижений в индустрии программирования. Квалифицированный программист всегда помнит об этом и посвящает все свою жизнь задаче продолжения своего образования.

Итак, учитывая сказанное выше, автор, по-видимому, сумел доказать, что не пытается превратить кого-либо в квалифицированного программиста. Настоящая книга не посвящена разработке программного обеспечения и полностью относится к программе SQL Server. Стремление автора затронуть темы, которые на первый взгляд кажутся более связанными с программированием, чем с изучением СУБД SQL Server, объясняется просто: автор пытается помочь читателям приобрести элементарные навыки в области программирования и отладки для того, чтобы они могли лучше понять, почему СУБД SQL Server спроектирована именно так, а не иначе, и могли самостоятельно продолжить исследование программы SQL Server. В целом настоящая книга должна служить для обретения настолько глубокого понимания работы SQL Server, насколько возможно, чтобы читатели могли лучше использовать эту программу в своей работе.

Об авторе

Кен Хендерсон (Ken Henderson) женат, имеет детей и проживает со своей семьей в Далласе, штат Техас. Он — автор семи ранее выпущенных книг, разработчик программного обеспечения с большим стажем, консультант и частый докладчик на конференциях. Кен — горячий болельщик баскетбольной команды Dallas Mavericks и проводит свободное время, играя со своими детьми, следя за спортивными событиями и импровизируя на музыкальных инструментах. Кен Хендерсон имеет свой Web-узел — `http://www.khen.com`; с ним можно связаться по электронной почте по адресу `khen@khen.com`.

Ждем ваших отзывов

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail:	`info@williamspublishing.com`
WWW:	`http://www.williamspublishing.com`

Адреса для писем:

из России:	115419, Москва, а/я 783
из Украины:	03150, Киев, а/я 152

ЧАСТЬ I

Основные сведения

ГЛАВА 1

Краткий обзор средств SQL Server

Ниже дано сокращенное описание основных тем, рассматриваемых в настоящей книге, и приведен небольшой обзор содержимого каждой главы. Это позволит читателю заранее ознакомиться с материалом, которому посвящена сама книга, и узнать, о чем пойдет речь в отдельных главах.

Краткий обзор содержимого глав

Глава 2. Основы Windows

В главе 2 рассматривается функционирование операционной системы Windows с точки зрения структуры программного обеспечения ОС. В этой главе обсуждаются различные компоненты Windows и способы создания приложений Windows, приведены сведения о библиотеках DLL, виртуальной памяти, режимах работы процессора и многих других особенностях устройства Windows, от которых зависит функционирование такого сложного приложения Windows, как SQL Server.

Глава 3. Процессы и потоки

Глава 3 посвящена описанию архитектуры Windows, обеспечивающей поддержку процессов и потоков. В этой главе дано общее определение понятия "процесс", показано, чем процессы отличаются от потоков, и приведены сведения о функционировании программы SQL Server в качестве процесса. Прочитав эту главу, читатель узнает, что такое планировщик Windows и как осуществляется планирование потоков на выполнение, а также подробно ознакомится с темой синхронизации потоков. Для демонстрации того, как действуют процессы и потоки под управлением операционной системы Windows и как это отражается на работе программы SQL Server, приведено несколько приложений C++ и расширенных процедур. С этой главы начинается разработка ряда примеров приложений, предназначенных для поиска подстрок в текстовых файлах, которая продолжается во всех остальных главах части I, "Основные сведения". При этом будет создано несколько разных версий приложения поиска в текстовых файлах (и во всех приложениях будут использоваться

разные основополагающие понятия Windows), что позволяет наглядно ознакомиться с тем, как обычно используются в реальных приложениях различные технологии Windows. Сведения, изложенные в этой главе, используются при описании планировщика непривилегированного режима (User Mode Scheduler — UMS) программы SQL Server ниже в данной книге (см. главу 10).

Глава 4. Основные принципы организации памяти

В этой главе показано, как осуществляется управление памятью Windows. Ознакомившись с главой 4, читатель узнает, чем виртуальная память отличается от физической, а также рассмотрит различия между виртуальной памятью и динамическими областями памяти. Кроме того, в главе описано, как операционная система Windows распределяет память по приложениям и как осуществляется преобразование виртуальных адресов памяти в физические. В этой главе продолжается тема создания средств поиска в текстовых файлах, начатая в предыдущей главе, и показано, как используются в приложениях средства управления памятью, предоставляемые операционной системой Windows. В главе приведены наглядные примеры приложений, созданных на основе указанных средств. Понятия, представленные в главе 4, становятся основой изложения после перехода к описанию средств управления памятью SQL Server ниже в данной книге (в главе 11).

Глава 5. Основные принципы ввода-вывода

В главе 5 приведен обзор основополагающих средств ввода-вывода Windows. В этой главе продолжается тема создания средств поиска в текстовом файле и проводится разработка приложений, в которых используется синхронный и асинхронный ввод-вывод, небуферизированный файловый ввод-вывод, ввод-вывод со сборкой-разборкой и файловый ввод-вывод с использованием отображаемых на память файлов. Изучение этой главы позволяет понять на примере реального приложения, как используются средства ввода-вывода Windows. А в главе 9, "Программа SQL Server как серверное приложение", показано, как эти средства используются в самой программе SQL Server.

Глава 6. Основы организации сетей

В главе 6 рассматриваются интерфейсы прикладного программирования (API-интерфейсы), которые относятся к области организации сетей Windows. Изучение этой главы позволяет многое узнать о сокетах и именованных каналах Windows, а также об API-интерфейсе RPC. Вместе с тем, в данной главе продолжается исследование приложений поиска подстрок в тексте и разрабатываются приложения, в которых сетевые API-интерфейсы Windows обеспечивают взаимодействие между компонентами приложения, а также организуют обработку входных и выходных данных. Кроме того, в главе 6 показано, как используются средства организации сетевого взаимодействия Windows в программе SQL Server.

Глава 7. Технология COM

В главе 7 рассматриваются основы технологии COM (Component Object Model – модель компонентных объектов) Microsoft и показано, как используется указанная технология в программе SQL Server. Прочитав эту главу, можно узнать о моделях многопотоковой поддержки, интерфейсах, маршалинге, подсчете количества ссылок, а также ознакомиться со многими другими понятиями COM. Кроме того, в главе указано, как обычно используются средства COM в приложениях Windows, и приведена информация о том, как эти средства применяются в программе SQL Server. В данной главе изложены основные сведения, необходимые для изучения главы 15, "Средства ODSOLE".

Глава 8. Язык XML

В главе 8 даны сведения о языке XML (eXtensible Markup Language – расширяемый язык разметки). Изучение этой главы позволяет узнать, как можно создать собственные документы XML и в чем состоят существенные различия между HTML и XML. В главе описываются атрибуты, элементы и схемы, а также способы применения таблиц стилей XML для преобразования данных. Данная глава содержит исходную информацию, которая необходима для изучения главы 18, "Технологии SQLXML".

Глава 9. Программа SQL Server как серверное приложение

В главе 9 речь идет о том, как функционирует программа SQL Server в роли серверного приложения Windows. В этой главе средства, представленные в предыдущих главах книги, описаны более подробно и показано, как они используются в программе SQL Server. Например, рассматриваются способы использования сетевых API-интерфейсов Windows в программе SQL Server для приема новых запросов на установление соединений. Кроме того, показано, как планируется дальнейшая обработка запросов с помощью компонента UMS (User Mode Scheduler – планировщик непривилегированного режима). В этой главе также рассматриваются импорт и применение библиотек DLL. К тому же, анализируется вопрос о том, какое место занимает программа SQL Server в общей классификации приложений Windows.

Глава 10. Планировщик непривилегированного режима

В главе 10 показано, каким образом в программе SQL Server происходит планирование работы, которая должна быть выполнена с помощью рабочих потоков и микропотоков. Изучение этой главы позволяет узнать, чем отличаются планировщики SQL Server и Windows, а также ознакомиться со сведениями об

использовании в программе SQL Server средств планирования Windows и синхронизационных объектов. Подробное изучение принципов функционирования планировщика UMS позволяет лучше понять, как происходит обработка клиентских запросов в программе SQL Server.

Глава 11. Управление памятью программы SQL Server

Содержание главы 11 базируется на приведенном ранее описании основных принципов управления памятью Windows. В этой главе показано, как рассматриваемые принципы применяются для управления памятью в программе SQL Server. Изучение данной главы позволяет получить сведения об областях памяти BPool и MemToLeave, а также узнать о том, как используются средства AWE (Address Windowing Extensions — адресные расширения для работы с окнами) для управления памятью.

Глава 12. Процессор запросов

Глава 12 содержит сведения о том, как организовано внутреннее функционирование процессора запросов SQL Server, а также показано, как осуществляются обработка и оптимизация запросов. В этой главе описаны четыре основные стадии оптимизации запросов. Кроме того, в данной главе показано, как используются индексы, статистические данные и ограничения для создания эффективных планов, а также описано, какая структура запросов обеспечивает максимальную производительность.

Глава 13. Транзакции

В главе 13 подробно описаны транзакции SQL Server. Эта глава содержит несколько примеров запросов на языке Transact-SQL, в которых используются транзакции, а затем показано, как действуют эти запросы и как в целом обрабатываются синтаксические конструкции управления транзакциями в программе SQL Server. В данной главе рассказывается, как избежать распространенных ошибок и правильно использовать средства управления транзакциями SQL Server в своих приложениях.

Глава 14. Курсоры

В главе 14 показано, как действуют курсоры SQL Server. Изучение данной главы позволяет ознакомиться с различными типами курсоров, освоить способы их применения и понять, как можно избежать распространенных ошибок. Кроме того, в этой главе описано взаимодействие транзакций и курсоров и показаны способы решения проблем снижения производительности, вызванных неправильным использованием курсоров.

Глава 15. Средства ODSOLE

В главе 15 описаны способы использования COM-объектов в сценариях Transact-SQL на основе средств связывания и внедрения объектов открытых служб данных (Open Data Services Object Linking and Embedding – ODSOLE) программы SQL Server. В этой главе рассмотрены расширенные процедуры sp_OA, показаны способы применения этих процедур, а также отмечено, при каких обстоятельствах их не следует использовать. Кроме того, показано, как создать некоторые более сложные COM-объекты, предназначенные для применения в языке Transact-SQL, включая ряд финансовых функций, функций работы с массивами и манипулирования строками.

Глава 16. Полнотекстовый поиск

В главе 16 рассматриваются средства полнотекстового поиска (Full-Text Search – FTS) программы SQL Server. Описывается, как действуют данные средства и как они спроектированы. Кроме того, в главе рассматриваются запросы FTS и показано, как использовать средства FTS в своем коде.

Глава 17. Объединения серверов

В главе 17 приведена информация о распределенных секционированных представлениях и показано, как они применяются в рамках объединений серверов. В главе представлено несколько примеров планов выполнения.

Глава 18. Технологии SQLXML

В главе 18 рассматриваются многие аспекты применения технологии XML в программе SQL Server. Изучение данной главы позволяет ознакомиться с конструкцией `FOR XML`, функцией `OPENXML()`, процедурой `sp_xml_preparedocument`, шаблонами обновления и поиска, управляемыми классами SQLXML и т.д. Материал главы 18 основан на тех сведениях о языке XML, которые были приведены в главе 8. Кроме того, в главе 18 показано, как в программе SQL Server предоставляется доступ к мощной коллекции средств, основанных на использовании языка XML, и как можно применить эти средства в своих приложениях.

Глава 19. Службы рассылки извещений

В главе 19 описывается технология Notification Services (служба рассылки извещений) программы SQL Server. В этой главе показано, на каком проекте основаны службы Notification Services и какую структуру имеет типичное приложение рассылки извещений. Изучение данной главы позволяет узнать, насколько просто платформа Notification Services позволяет создавать и развертывать такие приложения для работы с извещениями, которые являются полнофункциональными, мощными и масштабируемыми. В конце этой главы на примере показано, как создать собственное приложение Notification Services и приложение управления подпиской, используя интегрированную среду разработки VB.NET.

Глава 20. Службы Data Transformation Services

В главе 20 рассматриваются службы преобразования данных (Data Transformation Services — DTS) программы SQL Server. В этой главе речь идет о том, какой проект лежит в основе служб DTS, из каких основных компонентов состоят пакеты DTS и какие способы позволяют использовать эти службы для создания мощных и разносторонних приложений преобразования данных. Приведены примеры создания целого ряда пакетов и на основе этих примеров рассматриваются многие средства и возможности служб DTS. В завершение главы 20 описан проект, который позволяет узнать, как осуществляется доступ и управление пакетами DTS с помощью средств Automation.

Глава 21. Репликация снимка

В главе 21 рассматривается репликация снимков и показано, как обычно данные передаются с базы данных сервера публикаций на базу данных подписчика в ходе репликации снимка. Здесь вы научитесь обеспечивать полный контроль над этими данными. Кроме того, в этой главе приведены сведения о базе данных распределительного сервера и показано, как осуществляется процесс репликации с помощью программы Snapshot Agent.

Глава 22. Транзакционная репликация

Как и предыдущая, глава 22 посвящена репликации. Но в данном случае рассматривается транзакционная репликация и показано, чем она отличается от репликации снимка и репликации путем слияния. Изучение этой главы позволяет узнать, как осуществляется чтение журнала транзакций программой Log Reader Agent и как эта программа передает подписчикам данные об изменениях с помощью базы данных распределительного сервера. Кроме того, здесь описано, как действуют базы данных подписчика с немедленным обновлением и обновлением в порядке очереди, а также представлено внутреннее функционирование этих баз данных.

Глава 23. Репликация путем слияния

Глава 23 посвящена подробному описанию средств репликации путем слияния. В этой главе показано, как определить несколько подписок и проследить, как пользователи этих подписок участвуют в общем сценарии репликации путем слияния. Кроме того, в данной главе показано, как используются номера поколений и средства разрешения конфликтов, а также приведена информация о том, каким образом репликация путем слияния предоставляет возможность создавать разносторонние (но вместе с тем и сложные) средства репликации данных.

Глава 24. Поиск недокументированных средств

В главе 24 показано, как проводить поиск недокументированных средств. В своих предыдущих книгах автор подробно описывал недокументированные

средства программы SQL Server, а в главе 24 настоящей книги показал, как находить подобную ценную информацию самостоятельно. Изучение этой главы позволяет узнать, как использовать программу Profiler для поиска недокументированных средств и команд, а также ознакомиться с тем, как проводить поиск в тексте системных процедур для обнаружения недокументированных команд DBCC и флажков трассировки. Обладая навыками и сведениями, приобретенными в результате изучения данной главы, читатель сможет самостоятельно открывать недокументированные средства и команды.

Глава 25. Программа DTSDIAG

В главе 25 приведено вводное описание утилиты, реализованной в виде коллекции пакетов DTS, которая позволяет собирать диагностические данные, формируемые в процессе работы программы SQL Server. Применение такого инструментального средства позволяет собирать одновременно данные трассировок Profiler, выходные данные сценариев обнаружения блокировок, отчеты SQLDIAG, журналы Perfmon, журналы событий и много другой полезной диагностической информации. В коде этого инструментального средства воплощается несколько полезных способов использования служб DTS, в том числе способ автоматизации пакета DTS с помощью языка Visual Basic, способ структуризации приложения DTS путем разбиения его на отдельные пакеты, а также способ применения пакета DTS в качестве диспетчера потока данных для других задач.

Изложение аналогичного материала в разных главах

Такая организация книги легко прослеживается и без этого, но автор все же должен явно объяснить, с чем связано его намерение изложить близкий по тематике материал в нескольких разных главах. В данном случае во многом аналогичная информация разбивается на две главы. Например, автор предполагает, что рядовой администратор базы данных не обладает глубокими знаниями о том, как работает планировщик Windows (и поэтому не может объяснить, почему в программе SQL Server реализован собственный планировщик). В связи с этим автор подготовил отдельную главу, в которой подробно рассматривается функционирование планировщика Windows. Настоящая книга предназначена для профессионального пользователя базы данных, который не обязан досконально знать внутреннюю организацию работы операционной системы Windows (тем не менее не исключено, что он обладает такими знаниями). Поэтому автор считает себя обязанным объяснить некоторые понятия, лежащие в основе функционирования Windows, а не исходить из необоснованного предположения, что большинство читателей данной книги уже хорошо знакомы с ними. С другой стороны, автор мог бы на каждом шагу давать ссылки на другие источники, однако этот путь легок для автора, но утомителен для читателя. Кроме того, автор уверен, что сам способен внести много нового в описание основ приложений Windows.

Информация, изложенная во вступительной части, служит фундаментом, когда речь идет о различных компонентах SQL Server. Например, в главе 10, "Планировщик непривилегированного режима", используются сведения, приведенные при обсуждении планировщика Windows в главе 3, "Процессы и потоки". Такие же утверждения относятся к главам 4, "Основные принципы организации памяти", и 11, "Управление памятью программы SQL Server": первая из указанных глав закладывает основу для второй. То же касается главы 6, "Основы организации сетей", и 9, "Программа SQL Server как серверное приложение", т.е. изложение, начатое в одной главе, продолжается во второй. Каждая из ведущих глав в части I, "Основные сведения", является базой для одной или нескольких глав, приведенных в следующих частях настоящей книги, в которых рассматриваются конкретные технологии или компоненты SQL Server. В связи с этим целесообразно внимательно ознакомиться с информацией части I, прежде чем приступать к изучению остальных глав. Если вам покажется, что изучение начальных глав не имеет смысла, не беспокойтесь — значимость этой информации станет более очевидной после изучения остальных глав книги.

Если читатель хорошо знаком с внутренней организацией работы операционной системы Windows, технологиями COM, XML и тому подобным, он может пропустить все главы части I, "Основные сведения". Как было сказано в разделе "Введение", автор понимает, что подробная информация этих глав может не потребоваться для многих читателей. Если у вас возникают сомнения, действительно ли эта информация так необходима, можете перейти к описанию, относящемуся непосредственно к самой программе SQL Server (например, к главе 9, "Программа SQL Server как серверное приложение"), и проверить, все ли вам понятно без предварительной подготовки (например, без изучения главы 3, "Процессы и потоки"). Если вы ответите утвердительно, значит все великолепно. В противном случае вам придется ознакомиться с недостающими сведениями. Так или иначе, но читатель сможет найти в данной книге все необходимое для того, чтобы достичь полного понимания структуры и внутренней организации работы программы SQL Server.

Использование кода

В этой книге приведен значительный объем кода и сделан акцент на исследовании СУБД SQL Server как программы, поэтому на первый взгляд может показаться, что читателю надо быть весьма подготовленным программистом. Но автор не намеревался написать книгу, рассчитанную исключительно на программистов. Его цель состояла в том, чтобы реализовать новый и перспективный подход к обсуждению архитектуры программы SQL Server и найти удобный способ описать структуру и взаимодействие различных компонентов программы, поэтому в настоящей книге программный продукт SQL Server рассматривается с точки зрения профессионального разработчика. По мнению автора, такой подход вполне оправдан, поскольку СУБД SQL Server, в конечном итоге, является таким же приложением, как и любое другое. Очевидно, что разработчики при создании этого приложения использовали такие же средства, как и при создании любых

других приложений. Поэтому невозможно лучше понять, как работает приложение (поскольку нельзя достичь более высокого уровня мастерства), чем попытавшись разобраться в его работе, используя тот же подход, на основе которого это приложение было создано. Изучая SQL Server как приложение, мы пытаемся проникнуть в замыслы создавших его специалистов, понять, чего эти люди хотели достичь. Безусловно, существуют пределы того, чего можно добиться на этом пути, ведь мы же не создавали само приложение и не располагаем его исходным кодом! Но, подходя к изучению программы SQL Server на основе указанных принципов и рассматривая его наравне с любым другим сложным приложением Windows, можно действительно узнать очень многое. Используя такие инструментальные средства, как WinDbg, Perfmon и другие, мы можем, так сказать, "приподнять завесу тайны" и приобрести глубокое понимание многих нюансов архитектуры и внутренней организации рассматриваемого программного продукта. По мнению автора, просто не существует лучшего способа подробно изучить программу SQL Server или любое другое приложение стороннего поставщика.

Итак, чтобы стать квалифицированным пользователем SQL Server, вы должны быть программистом или обязаны разобраться во всем коде, приведенном в настоящей книге? Разумеется, нет, но это было бы не лишним. Если вы не считаете себя программистом, все равно не следует огорчаться. Прочитайте текст и примеры, приведенные в книге, выполните все упражнения, какие сможете, и работайте в том темпе, какой вас устраивает. Сам факт, что книга посвящена глубокому изучению конкретной программы, не свидетельствует о том, что издание предназначено только для программистов. Автор как раз и надеется на то, что многие читатели, которые отнюдь не считают себя разработчиками программного обеспечения, обнаружат в себе скрытые способности к программированию и, подняв свои навыки в разработке программ еще на один уровень, достигнут такого понимания работы программы SQL Server, которое в ином случае не было бы возможно и которого они никогда еще не добивались прежде. И автор надеется, что после этого они станут достаточно подготовленными для того, чтобы самостоятельно продолжить исследование SQL Server. Автор стремился не просто открыть перед читателями готовый набор подробных сведений (а такую тенденцию часто можно обнаружить даже в самых лучших образцах специальной литературы), но и показать читателям, как открывать эти сведения самостоятельно, исследуя такие сложные приложения Windows, как SQL Server. Навыки проведения исследований, приобретенные с помощью настоящей книги, будут полезны независимо от рассматриваемого программного продукта или программы. В частности, такие навыки должны позволить вам продолжать исследования программы SQL Server в течение многих последующих лет.

Основы Windows

Как было сказано во "Введении", знания принципов работы операционной системы (ОС) Windows существенно важны для понимания работы таких сложных приложений Windows, как SQL Server. Без полного представления о том, как функционирует эта операционная система, читатель не будет иметь ни инструментов, ни точек отсчета, позволяющих разобраться в работе программы SQL Server. Люди хранят знания в своих "нейронных сетях", создаваемых в результате связывания новых знаний с существующими. Знания и понимание, полученные читателем при изучении программы SQL Server, должны укладываться в более крупную инфраструктуру представлений о том, как функционирует ОС Windows и как в целом создаются приложения Windows. Цель этой главы состоит в том, чтобы ознакомить читателя с некоторыми фундаментальными компонентами Windows и заложить основу для более глубокого обсуждения этих тем в последующих главах.

API-интерфейс Win32

API-интерфейс Win32 представляет собой интерфейс программирования для 32-разрядной операционной системы Windows. Приложения Windows выполняют вызовы функций этого API-интерфейса в целях обращения к службам ОС. Фактически код, входящий в состав этого интерфейса, находится в коллекции таких библиотек динамического связывания (Dynamic-Link Library — DLL), как `Kernel32.DLL`, `User32.DLL` и `GDI32.DLL`, а также, безусловно, в самом ядре операционной системы, код которого размещается в основном в файле `NTOSKRNL.EXE`.

Для использования с первоначальной версией 32-разрядной ОС Windows предназначался не API-интерфейс Win32, а API-интерфейс OS/2 Presentation Manager. Но в ходе разработки операционной системы, которая в дальнейшем стала первой версией Windows NT, произошел выпуск Windows 3.0 с 16-разрядным API-интерфейсом и началось стремительное распространение Windows как платформы для разработки прикладного программного обеспечения. Вдохновленное этими успехами руководство компании Microsoft решило, что новый 32-разрядный программный API-интерфейс должен стать совместимым с существующим 16-разрядным, что позволило бы упростить перенос приложений из среды Windows 3.x в среду новой операционной системы. В этом и заключаются причины создания API-интерфейса Win32, а также причины того, что иногда данный интерфейс кажется немного внутренне несогласованным или негармоничным; дело в том, что он был предназначен для достижения максимально возможной совместимости со старым 16-разрядным API-интерфейсом Windows.

Поскольку SQL Server — это сложное приложение Windows, то в нем, безусловно, API-интерфейс Win32 используется очень интенсивно. Многие функции API-интерфейса Win32, применяемые в программе SQL Server, рассматриваются в первой половине данной книги. Читатель узнает, как они работают и используются, а также получит определенное представление о применении этих функций в программе SQL Server. Это позволит понять связь между некоторыми ключевыми средствами программы SQL Server и функциями API-интерфейса Win32, на которые опираются эти средства.

Читатель может загрузить пакет инструментальных средств разработчика программного обеспечения Platform SDK, который содержит файлы заголовков и библиотеки C, а также оперативную документацию по API-интерфейсу Win32, непосредственно с Web-узла Microsoft. Кроме того, инструментальные средства Platform SDK поставляются в комплекте с некоторыми инструментальными средствами разработки компании Microsoft и включены в библиотеку MSDN Library.

Сравнение непривилегированного и привилегированного режимов

Для того чтобы можно было предотвратить дестабилизацию системы под действием неправильно работающего прикладного кода, в системе Windows используются два режима работы процессора — непривилегированный и привилегированный. Прикладной код пользователя работает в непривилегированном режиме, а код операционной системы и драйверы устройств — в привилегированном. Привилегированный режим характеризуется более высоким уровнем привилегий доступа к аппаратным средствам по сравнению с непривилегированным и обеспечивает доступ ко всей памяти системы и ко всем командам процессора. Операционная система Windows действует с более высокими привилегиями, чем прикладное программное обеспечение, и поэтому способна предотвратить непосредственную дестабилизацию системы под влиянием неправильно работающего приложения.

В действительности семейство процессоров Intel x86 поддерживает четыре режима работы (называемых также *кольцами*). Они обозначены номерами от 0 до 3, и каждый режим изолирован от другого с помощью аппаратных средств; это позволяет предотвратить дестабилизацию работы в режимах с более высокими приоритетами под действием аварийного нарушения, происшедшего при работе в режиме с более низким приоритетом. Операционная система Windows была первоначально предназначена для поддержки таких процессоров, как Alpha компании Compaq и MIPS компании Silicon Graphics, в которых предусмотрено только два режима процессора, поэтому в ней используются лишь два из четырех режимов семейства процессоров Intel x86 (кольца 0 и 3, применяемые соответственно для привилегированного и непривилегированного режимов). А теперь указанные процессоры компаний Compaq и Silicon Graphics уже не поддерживаются, поэтому, вероятно, имело бы смысл предусмотреть в системе Windows по меньшей мере еще один, дополнительный, режим работы из предусмотренных в семействе процессоров Intel x86. Это позволило бы, например, обеспечить работу драйверов

устройств с более низким уровнем привилегий по сравнению с самой операционной системой и исключить возможность останова всей системы из-за единственного неправильно работающего драйвера.

Процессы и потоки

Экземпляр приложения, работающего на компьютере, принято называть *процессом*. Но фактически такое словоупотребление является неправильным. В действительности работают не процессы, а потоки. Каждый процесс состоит по меньшей мере из одного (называемого *главным*) или нескольких потоков. Каждый поток представляет собой независимый механизм выполнения кода. Любой код, функционирующий в рамках любого приложения, выполняется в виде одного из потоков.

Каждому процессу выделяется собственное пространство адресов виртуальной памяти. Этим пространством виртуальной памяти совместно пользуются все потоки, принадлежащие к одному процессу. Многочисленные потоки, модифицирующие один и тот же ресурс, должны синхронизировать доступ к этому ресурсу в целях предотвращения ошибочного поведения и возможных нарушений доступа. Процесс, в котором правильно организован доступ к ресурсам, совместно используемым многочисленными потоками, называется *потокобезопасным*.

Каждый поток в процессе получает свой собственный набор временных регистров. Временный регистр представляет собой программный эквивалент регистра процессора. Для того чтобы можно было дать возможность любому потоку поддерживать контекст, независимый от других потоков, всем потокам предоставляются собственные наборы временных (программных) регистров, которые используются для сохранения и восстановления аппаратных регистров. Каждый раз, когда операционная система Windows выделяет процессор потоку (*планирует* поток для выполнения на процессоре) или отменяет выделение процессора потоку (т.е. *деплапирует* выполнение потока), эти временные регистры копируются в прямом и обратном направлениях в регистры процессора. Такая операция называется *переключением контекста*.

Процессы могут инициироваться приложениями разных типов. К примерам исполняемых файлов, которые могут быть вызваны на выполнение для создания экземпляра процесса, относятся терминальные приложения, приложения с графическим пользовательским интерфейсом (Graphical User Interface — GUI), службы Windows, внепроцессные COM-серверы и т.д. Программа SQL Server может использоваться и как терминальное приложение, и как служба Windows.

Особенности виртуальной и физической памяти

Операционная система Windows предоставляет всем процессам пространство виртуальной памяти объемом 4 Гбайт, в котором они функционируют. Слово "виртуальная" подчеркивает, что эта память не является памятью в традицион-

ном смысле этого слова. Она представляет собой просто ряд адресов, с которым явно не связана какая-либо физическая память. А по мере того как процесс распределяет память из выделенного ему пространства, эти адреса отмечаются как используемые, и с ними ассоциируются адреса физической памяти. Но эти адреса физической памяти не обязательно находятся в самой физической (оперативной) памяти (обычно даже имеет место совсем иное распределение). Как правило, эти адреса соответствуют пространству памяти на диске. А это пространство находится в файле (файлах) подкачки операционной системы. Именно поэтому обеспечивается возможность одновременной эксплуатации многочисленных приложений в системе с объемом памяти 128 Мбайт и выделения каждому из этих приложений пространства виртуальных адресов с объемом 4 Гбайт: это не настоящая память, хотя и представляется таковой для приложения. Операционная система Windows обеспечивает прозрачное прямое и обратное копирование данных в файл подкачки для того, чтобы приложение могло распределять больше памяти, чем физически имеется на компьютере, что позволяет предоставлять многочисленным приложениям равноправный доступ к физической оперативной памяти компьютера.

Указанное пространство адресов с объемом 4 Гбайт разбито на два раздела — непривилегированного и привилегированного режимов. По умолчанию каждый из них имеет размер по 2 Гбайт, хотя это значение можно изменить с помощью параметров файла BOOT.INI в семействе Windows NT этой операционной системы (к семейству Windows NT относятся Windows NT, Windows 2000, Windows XP и Windows Server 2003; Windows 9x и Windows ME к нему не относятся).

Хотя каждый процесс получает свое собственное пространство адресов виртуальной памяти, код операционной системы и код драйверов устройств совместно используют единственное пространство закрытых адресов. Каждая страница виртуальной памяти связана с одним определенным режимом процессора. Для получения доступа к этой странице необходимо перевести процессор в соответствующий режим. Это означает, что пользовательские приложения не могут непосредственно получить доступ к виртуальной памяти привилегированного режима; чтобы стала доступной память привилегированного режима, система должна перейти в привилегированный режим.

Дополнительные сведения о виртуальной памяти, а также о том, как ею управляет операционная система Windows, приведены в главе 4, "Основные принципы организации памяти". На данный момент достаточно понять, что виртуальная память не обязательно должна соответствовать физической. Доступ к виртуальной памяти — это сервис, предоставляемый системой Windows, который позволяет приложениям (и самой ОС Windows) распределять и использовать больший объем адресов первичной памяти (или оперативной памяти), чем физически существует на компьютере, без необходимости выполнять самостоятельно страничный обмен данными, передавая их в прямом и обратном направлении на устройства вторичной памяти (жесткие диски).

Подсистемы

В поставку Windows входят три подсистемы, обеспечивающие создание разных вариантов среды, – Win32, OS/2 и POSIX. Каждая из этих подсистем позволяет создать отдельную среду для этой операционной системы или обеспечить выполнение индивидуальных требований ОС. Безусловно, среди всех подсистем Win32 является доминирующей. Это связано с тем, что она является обязательной (и должна всегда функционировать независимо от выбранной подсистемы создания среды и невзирая на то, зарегистрирован ли в системе какой-либо пользователь), а также потому, что она предоставляет самый непосредственный и самый полный доступ к самой ОС Windows. Другие подсистемы создания среды используются не столь часто и не предоставляют такой же уровень функциональных возможностей, как подсистема Win32. Приложения, которые работают в операционной системе Windows, компилируются и связываются с библиотеками в целях дальнейшего выполнения в конкретной подсистеме создания среды. Безусловно, большинство из этих приложений, включая SQL Server, являются приложениями Win32.

Подсистему создания среды Win32 можно разделить на перечисленные ниже основные компоненты.

- Процесс подсистемы создания среды, `Csrss.exe`, который поддерживает создание процессов, потоков и терминальных окон, а также предоставляет доступ к отдельным частям 16-разрядной виртуальной машины DOS и к разнообразным функциям.
- Драйвер устройства привилегированного режима, `Win32k.sys`, который включает два средства: во-первых, диспетчер окон, т.е. средство, обеспечивающее получение входных данных от клавиатуры и мыши, управление выводом на экран и передачу сообщений приложениям, и, во-вторых, интерфейс графических устройств (Graphics Device Interface – GDI). Последнее средство обеспечивает вывод данных на графические устройства.
- Библиотеки DLL подсистемы (в число которых входят `Kernel32.DLL`, `User32.DLL`, `GDI32.DLL` и `Advapi32.DLL`), преобразующие функции API-интерфейса Win32 в вызовы служб привилегированного режима.

Приложения Windows взаимодействуют с ядром этой операционной системы с помощью библиотек DLL используемой подсистемы. В этих библиотеках DLL скрыты фактические встроенные вызовы операционной системы (которые не отражены в документации), поступающие от приложения. Назначением этих библиотек DLL является преобразование вызовов API-интерфейса Win32 в вызовы служб операционной системы. Такие вызовы могут сопровождаться передачей сообщения в процесс подсистемы создания среды, под управлением которого работает приложение.

Поскольку выше было сказано, что код непривилегированного режима не может обращаться к памяти привилегированного, может оказаться непонятным, как же код непривилегированного режима вызывает код и обращается к данным, которые явно находятся в ядре, ведь все основные функциональные средства операционной

системы реализованы в ядре, потому оно так и называется. Но это взаимодействие между режимами организовано таким образом, что приложения, действующие в непривилегированном режиме, выполняют вызовы API-интерфейса Win32 к функциям, экспортируемым из библиотек DLL используемой подсистемы. Затем из этих библиотек DLL выполняются вызовы к функциям недокументированного встроенного API-интерфейса, который реализован в библиотеке NTDLL.DLL. После этого в функциях NTDLL.DLL вызываются соответствующие используемой платформе команды для переключения микросхемы процессора в привилегированный режим и вызова соответствующего кода в ядре операционной системы. Этот код может находиться в исполняемом файле ядра, NTOSKRNL.EXE, или в драйвере устройства привилегированного режима, Win32k.sys. (Педанты могут возразить, что в действительности происходят немного более сложные действия, но приведенное выше описание позволяет получить основное представление о том, как приложения непривилегированного режима взаимодействуют с ядром Windows.)

Библиотеки динамического связывания

Любая библиотека DLL представляет собой двоичный файл, служащий в качестве совместно используемой библиотеки процедур, которые на этапе прогона могут динамически загружаться и выгружаться приложениями, использующими эти процедуры. В виде библиотек DLL могут быть оформлены библиотеки этапа прогона и библиотеки классов для таких языковых продуктов, как Visual C++ и Delphi. В виде библиотек DLL, таких как Kernel32.DLL и User32.DLL, оформлена также часть API-интерфейса Win32 для непривилегированного режима. Одним из преимуществ библиотеки DLL над статической библиотекой является то, что единственная библиотека DLL может совместно использоваться всеми многочисленными приложениями. При этом операционная система Windows гарантирует, что в память будет отображаться только единственная копия кода каждой процедуры из библиотеки DLL, независимо от того, сколько приложений обращаются к этой процедуре.

Функции, предоставляемые библиотеками DLL, становятся видимыми для внешнего мира благодаря тому, что они экспортируются. Таблицу экспорта любой библиотеки DLL можно просматривать с помощью таких внешних инструментальных средств, как Depends, которое входит в состав программы Visual Studio, или утилита dumpbin, входящая в состав нескольких продуктов Microsoft. Процедуру DLL можно экспортировать, указав имя, порядковый номер или то и другое. Кроме того, библиотеки DLL (и исполняемые файлы) имеют таблицы импорта. В этих таблицах перечислены библиотеки DLL, от которых они зависят, и функции, импортируемые ими статически. Описание особенностей статического импорта приведено немного ниже.

Процесс может загружать библиотеки DLL с помощью одного из двух способов: либо неявно, после своего запуска, либо явно, с помощью вызова API-интерфейса LoadLibrary(Ex). Способ, с помощью которого загружается конкретная библиотека DLL, определяется тем, по какому принципу исполняемая программа ссылается на функцию, экспортируемую (предоставляемую для внешнего мира)

в используемой библиотеке DLL. Такие ссылки могут осуществляться с помощью одного из двух способов. Во-первых, в исполняемом файле можно предусмотреть статический импорт функций, экспортируемых библиотекой DLL, во время его компиляции и связывания с библиотеками путем импортирования файла `.LIB` применяемой библиотеки DLL. (Не существует единого требования, чтобы все компиляторы и редакторы связи применяли для статического связывания файлы `.LIB`, и чаще всего такой способ используется в продуктах C и C++; например, файлы `.LIB` не используются и не требуются ни в программных продуктах Visual Basic, ни в программных продуктах Delphi.) Статический метод импорта вызывает автоматическую загрузку библиотеки DLL при запуске исполняемого файла. А если нужную библиотеку DLL не удается найти, исполняемый файл не запускается. Статическое импортирование обычно представляет собой именно тот способ, с помощью которого библиотеки DLL загружаются в приложениях Windows. Для этого способа загрузки требуется меньше кода и он осуществляется в основном под управлением самой операционной системы. По крайней мере, все приложения Windows статически импортируют библиотеку `Kernel32.DLL`, а большинство импортирует также библиотеку `User32.DLL`, поскольку в этих библиотеках DLL содержится львиная доля функций API-интерфейса Win32.

Любая библиотека DLL может быть также загружена исполняемым файлом на этапе прогона с помощью вызова API-интерфейса LoadLibrary. В этом случае интерфейсу LoadLibrary передается имя загружаемой библиотеки DLL, а он возвращает дескриптор модуля, если находит модуль (а если не находит, то возвращает `NULL`). Затем дескриптор модуля передается в функцию `GetProcAddress` для получения адреса конкретной функции, экспортируемой библиотекой DLL. После этого тип функции, находящейся по данному адресу, может быть приведен к такому типу функции, который может использоваться приложением для последующего вызова. Например, именно так в программе SQL Server вызываются расширенные процедуры и загружаются библиотеки SQL Server Net-Libraries. Если разработчик приложения во время компиляции и связывания приложения с библиотеками не знает, будет ли находиться в системе библиотека DLL, которая может ему потребоваться, он вынужден использовать для ее загрузки интерфейс LoadLibrary. Например, если в приложении загружаются сменные драйверы базы данных с помощью интерфейса ODBC, то через этот интерфейс загружаются содержащие эти драйверы библиотеки DLL с использованием интерфейса LoadLibrary, поскольку не существует способа узнать еще на этапе компиляции и связывания с библиотеками, какие драйверы будут присутствовать в данной конкретной системе.

Поэтому после загрузки определенной библиотеки DLL в пространство адресов процесса эта библиотека становится кодом, который может быть вызван процессом. Каждая библиотека DLL имеет применяемый по умолчанию адрес загрузки в пространстве адресов процесса с объемом 4 Гбайт, который задается во время ее связывания. Если в вызывающем процессе никакая другая информация не занимает диапазон адресов, на загрузку в который настроена конфигурация DLL, библиотека загружается по этому адресу. А если здесь уже находится что-то другое, то библиотека должна быть "перебазирована"; для выполнения этой операции требуется перечитать целый образ библиотеки и обновить все адресные

привязки, отладочную информацию, контрольные суммы и значения временных отметок, чтобы можно было использовать другой базовый адрес. Поскольку для этого требуется обновлять страницы, содержавшиеся в образе DLL, то ОС Windows должна загрузить каждую страницу, подлежащую модификации, в виртуальную память и внести изменения в память.

Как было сказано выше и подробно описано в главе 4, нормальный режим осуществления операций размещения кода и данных в пространстве адресов процесса состоит в том, что эти операции поддерживаются с помощью файла подкачки (или физической памяти). Но в случае двоичного кода приходится делать исключение, поскольку исполняемый код обычно предназначен только для чтения, и его копирование в файл подкачки равносильно созданию еще одной ненужной копии. Вместо этого диапазон адресов виртуальной памяти в пространстве адресов процесса, отведенный для исполняемого файла или библиотеки DLL, "резервируется" в самом файле EXE или DLL (т.е. для этого диапазона предусматривается пространство физической памяти). Поэтому, выполняя вызов к части исполняемого файла или библиотеки DLL, не находящейся в физической памяти, процесс не обращается к файлу подкачки для получения нужной страницы файла EXE или DLL. Вместо этого он находит соответствующий двоичный файл и загружает страницу в физическую память непосредственно из него. В этом смысле файлы EXE и DLL становятся *расширениями файла подкачки*, предназначенными только для чтения.

Обычно в памяти поддерживается только одна копия страниц, из которых состоит файл DLL или EXE, независимо от того, сколько процессов используют этот файл. Исключение из данного правила возникает, если процесс вносит изменение в глобальную или статическую переменную DLL или EXE. Если это происходит, система Windows создает копию страницы, которая является локальной для данного процесса, вносит требуемое изменение, а затем изменяет характеристики самого процесса, чтобы он в дальнейшем ссылался на новую версию указанной страницы. Механизм, с помощью которого осуществляется эта операция, известен как *память с копированием при записи*.

Способность использовать файл DLL или EXE в качестве физического средства хранения для диапазона адресов виртуальной памяти, который отведен для этого файла, реализуется с помощью средства отображения файла на память операционной системы Windows, которое фактически может использоваться для файлов любых типов, а не только для двоичных файлов. Файл, отображаемый на память, служит в качестве реальной памяти для виртуальной памяти, которую он занимает. Вместо копирования этого файла в системный файл подкачки Windows использует его так, как если бы он сам представлял собой файл подкачки, и автоматически сохраняет (загружает) страницы, осуществляя прямой и обратный обмен данными с этим файлом как с виртуальной памятью, в которую отображена ссылка на данный файл. В этом и состоит суть использования виртуальной памяти — она обеспечивает единообразный доступ, но фактически не существует. Виртуальная память не представляет реальную память до тех пор, пока для нее не отводится реальная память. Средство отображения файла на память операционной системы Windows предоставляет приложениям возможность обращаться к файлам так, как если бы они представляли собой оперативную память, а операционная система Windows автоматически выполняет все операции ввода-вывода,

необходимые для поддержки этого средства. В самой операционной системе Windows во время доступа к файлам EXE и DLL также используется указанное средство. Все описанные здесь темы рассматриваются более подробно в главе 4.

Инструментальные средства

В этом разделе кратко описаны некоторые инструментальные средства, применяемые во всей книге при исследовании возможностей программы SQL Server. Эти инструментальные средства можно получить из многих источников, но читателю, безусловно, не нужно держать в своем распоряжении их все, чтобы иметь возможность работать с настоящей книгой. Автор упоминает их в разных местах для того, чтобы подчеркнуть их важность и дать читателю определенные рекомендации по использованию данных инструментов для решения конкретной задачи или исследования конкретного фрагмента данных. Несмотря на это, для выполнения большинства упражнений данной книги практически не требуется что-либо более сложное по сравнению с инструментальными средствами, входящими в состав отладчика WinDbg, который является одним из компонентов операционной системы Windows и предоставляется для бесплатной загрузки компанией Microsoft.

Программа TList

Это инструментальное средство содержится на компакт-диске *Windows Support Tools*. Его можно использовать для ознакомления со списком работающих процессов и со списком модулей, загруженных в каждом процессе. С его помощью можно также получить много другой информации, касающейся процессов.

Программа Pviewer

Инструментальное средство Pviewer также находится на компакт-диске *Windows Support Tools*. Оно позволяет просматривать информацию о работающих процессах и потоках. С его помощью можно уничтожать процессы и изменять классы приоритетов процессов. А еще одна особенность данного инструмента, которая действительно может оказаться очень удобной, состоит в том, что он может использоваться для ознакомления с процессами не только на локальном, но и на удаленном компьютере, подключенном к сети.

Программа Pview

Это инструментальное средство входит в состав комплекта Platform SDK и, по существу, аналогично инструментальному средству Pviewer. Оно предоставляет такие же функциональные возможности и отображает информацию такого же типа. Безусловно, предоставляемые им возможности зависят от выпуска самого комплекта SDK, а также от того, насколько современной является его версия, но в основном Pview ничем не отличается от инструментального средства Pviewer.

Программа Perfmon

Монитор производительности (Performance Monitor) операционной системы Windows, или Perfmon, по-видимому, представляет собой одно из наиболее ценных инструментальных средств, включенных в поставку этого продукта, которое позволяет заглянуть внутрь операционной системы и узнать, что в ней происходит. Программа Perfmon (или Sysmon, как теперь ее принято называть) предоставляет возможность следить за несколькими важными статистическими показателями, касающимися работающих процессов и потоков, степени загрузки памяти и процессора, интенсивности использования диска, а также знакомиться со многими другими интересными объектами и диагностическими данными.

Суть применения программы Perfmon состоит в том, что к некоторой коллекции данных добавляются *счетчики*, после чего этому инструментальному средству дается разрешение осуществлять с помощью счетчиков выборку данных в ходе дальнейшей работы системы с учетом изменений во времени. Изменяющееся содержимое этих счетчиков можно рассматривать в виде линий на диаграмме литеральных (фактических) значений или столбиков на гистограмме. Журналы, созданные программой Perfmon, можно сохранять в виде двоичных файлов, текстовых файлов с разграничителями и таблиц SQL Server. Автор будет указывать различные счетчики Perfmon, которые могут применяться для изучения конкретной технологии или подсистемы в рамках Windows или SQL Server.

Программа WinDbg

Как было указано во "Введении", в настоящей книге для ознакомления с внутренним функционированием программы SQL Server будет часто использоваться доступный для бесплатной загрузки автономный отладчик WinDbg компании Microsoft. Но при этом WinDbg фактически не будет применяться в тех целях, в которых обычно используются отладчики общего назначения — отладка приложений. Вместо этого WinDbg будет служить в качестве своего рода "рентгеновского аппарата" для программы SQL Server, т.е. в качестве инструментального средства, позволяющего видеть, что происходит в этой программе незаметно для постороннего взгляда.

Одна из версий отладчика WinDbg входит в состав операционной системы Windows, и ее также можно загрузить с общедоступного Web-узла Microsoft (ко времени написания данной книги эту версию можно было найти по адресу `http://www.microsoft.com/ddk/debugging/default.asp`). Автор рекомендует для выполнения упражнений, приведенных в книге, получить отладчик WinDbg с Web-узла Microsoft, чтобы быть уверенным в наличии у себя его последней версии.

ПРИМЕЧАНИЕ. Кроме того, компания Microsoft включает в свой пакет Debugging Tools for Windows отладчик `cdb.exe` с интерфейсом командной строки. Он может оказаться более удобным по сравнению с WinDbg для тех, кто предпочитает работать с интерфейсом командной строки, а не с графическим интерфейсом. В этом отладчике используется такая же "машина" отладки, как и в WinDbg, а команды отладчика, приведенные в данной книге, столь же успешно подходят и для него.

По-видимому, одно из наиболее важных требований, которые необходимо учитывать при использовании WinDbg или любого другого символического отладчика, состоит в том, что для успешной отладки основной части любой программы необходимо иметь в своем распоряжении файлы отладочной информации, а обозначение пути к отладочной информации в отладчике должно быть задано так, чтобы он мог их найти. Отладочная информация вырабатывается используемым на компьютере продуктом, который обеспечивает компиляцию (связывание) с библиотеками. Для продуктов Microsoft стандартным форматом файла отладочной информации является формат PDB (Program DataBase), а сами файлы отладочной информации вырабатываются автоматически для отладочных версий программ на языке Visual C++ и для любых исполняемых файлов на языке VB, для которых задано свойство проекта Create Symbolic Debug Info. Как правило, если в распоряжении читателя имеется файл отладочной информации, то он может найти файл PDB, имя которого соответствует имени созданного файла EXE или DLL, в том же каталоге, где находится сам исполняемый файл.

Исключением из этого правила является программа SQL Server. Файлы отладочной информации программы SQL Server находятся в подкаталогах `exe` и `dll` главного каталога `Binn` программы SQL Server. (Файлы отладочной информации для программы `sqlservr.exe`, главного исполняемого файла SQL Server, находятся в каталоге `exe`, а файлы отладочной информации для основных библиотек DLL расположены в каталоге `dll`.) Эти файлы отладочной информации представляют собой файлы, предназначенные для широкого круга пользователей; в них исключены многие компоненты, необходимые для полноценной отладки, такие как определения типов параметров, локальные переменные и т.п. Эти файлы отладочной информации непригодны для полноценной отладки, но вполне подходят для наших целей. Общедоступные файлы отладочной информации являются очень удобными, поскольку позволяют раскрывать внутреннее функционирование любого приложения и исследовать любой работающий процесс.

Как уже было сказано, для успешной отладки любой программы с помощью WinDbg необходимо правильно задать путь к файлам отладочной информации. Этот путь можно задать в программе WinDbg, нажав клавиши <Ctrl+S> или выбрав команды File | Symbol File Path в системе меню.

Общедоступные файлы отладочной информации для многих версий Windows (и для многих других продуктов) можно получить по Internet с сервера файлов отладочной информации компании Microsoft. Читателю может даже не потребоваться загружать эти файлы. Достаточно просто указать отладчику на этот сервер, после чего он будет по мере необходимости автоматически загружать (и кэшировать) файлы отладочной информации через открытое соединение с Internet. Ко времени написания данной книги достаточно было указать в пути к файлам отладочной информации адрес `http://msdl.microsoft.com/download/symbols`, чтобы задать ссылку на сервер компании Microsoft, доступный для широкого круга пользователей Internet. У компании Microsoft по адресу `http://www.microsoft.com/ddk/debugging/symbols.asp` имеется превосходная Web-страница, на которой описано, как именно действует этот сервер и как воспользоваться его услугами. Прочитайте эту страницу и установите соответствующим образом путь к файлам

отладочной информации в отладчике WinDbg. В настоящее время путь к файлам отладочной информации в отладчике WinDbg, используемом автором, задан с помощью параметра `SRV*c:\temp\symbols*http://msdl.microsoft.com/download/symbols`.

При отладке компонента или программы, для которой имеется исходный код, важно правильно установить путь к нему в отладчике WinDbg. Это можно сделать, нажав клавиши <Ctrl+P> или выбрав команды File | Source File Path в системе меню WinDbg. Путь к исходному коду, заданный в отладчике WinDbg, позволяет обеспечить пошаговое выполнение приложения, устанавливать точки останова и выполнять другие операции.

ПРИМЕЧАНИЕ. Автор обязан подчеркнуть следующее: нет никаких гарантий, что в будущем для программы SQL Server будут по-прежнему предоставляться файлы отладочной информации любого вида. В настоящее время они включены в поставку данного продукта, но в какой-то момент в будущем это решение может быть отменено. А если случится подобное, то останется лишь надеяться на то, что компания Microsoft предоставит доступ к ним через свой сервер файлов отладочной информации, как и для некоторых других продуктов.

Резюме

Развитая, надежная операционная система Windows состоит из нескольких подсистем. Будучи запрограммированной с использованием API-интерфейса Win32, эта операционная система предоставляет механизм, позволяющий пользовательским приложениям (косвенно) направлять свои вызовы в код ядра системы.

Для Windows предусмотрена архитектура, позволяющая предотвратить дестабилизацию работы этой операционной системы под действием пользовательских приложений, содержащих ошибки. Она предоставляет средства виртуальной памяти, которые позволяют исключить необходимость реализовывать собственные диспетчеры виртуальной памяти непосредственно в приложениях. Кроме того, в ней предусмотрена поддержка файлов с отображением на память и средств копирования при записи, благодаря которым выполнение усовершенствованных операций управления памятью становится легким и эффективным.

Вопросы для самопроверки

1. Какой объем виртуальной памяти предусмотрен по умолчанию для той части процесса, которая функционирует в непривилегированном режиме?
2. Сколько режимов работы (колец) поддерживает семейство процессоров Intel x86?
3. Подтвердите или опровергните следующее утверждение. Фактически в процессе не выполняется какой-либо код, поскольку для выполнения кода приложения требуется поток.

4. Объясните назначение памяти для копирования при записи.
5. Подтвердите или опровергните следующее утверждение. Чтобы использовать отладчик WinDbg для отладки любого продукта компании Microsoft, необходимо загрузить отладочную информацию с Web-узла Microsoft и скопировать ее на локальный сервер файлов отладочной информации.
6. Подтвердите или опровергните следующее утверждение. В операционной системе Windows используются все режимы работы, поддерживаемые любым процессором, на котором она функционирует.
7. Опишите связь между функциями `LoadLibrary` и `GetProcAddress` API-интерфейса Win32.
8. Что такое *переключение контекста*?
9. Подтвердите или опровергните следующее утверждение. В операционной системе Windows предусмотрен механизм, который может применяться в любом приложении для выборочной загрузки библиотек DLL на этапе прогона.
10. В чем состоит наиболее важное требование, которое необходимо выполнить, чтобы обеспечить возможность отладки любого процесса с помощью символического отладчика?

Процессы и потоки

В этой главе рассматриваются процессы и потоки, функционирующие в операционной системе Windows. Здесь описано, чем отличаются процессы от потоков и в чем они похожи друг на друга, а также показана уникальная роль, выполняемая процессами и потоками в операционной системе Windows.

Кроме того, в этой главе подробно описано, как действует планировщик потоков Windows и как происходят планирование потоков для выполнения на процессорах и отмена планирования. В ней также приведены сведения о синхронизации потоков и о том, как в многопотоковых приложениях, подобных SQL Server, используются синхронизационные объекты для упорядочения доступа к совместно применяемым ресурсам и обеспечения безопасности работы потоков.

Процессы

Процессы. Основные термины и определения

- **Процесс.** Воплощение всего, что связано с программой, работающей в операционной системе Windows. Процесс предоставляет контекст, в котором потоки могут выполнять работу приложения.
- **Адресное пространство процесса.** Пространство адресов виртуальной памяти для приложения. Для 32-битовых приложений Windows оно ограничено объемом 4 Гбайт. Адреса в приложениях Win32 ограничены объемом 4 Гбайт, поскольку 4 Гбайт — это наибольшее целочисленное значение, которое может храниться в 32-битовом указателе. Из этих 4 Гбайт по умолчанию 2 Гбайт зарезервировано для доступа в привилегированном режиме, а остальные 2 Гбайт предназначены для доступа в непривилегированном режиме. В некоторых версиях операционных систем из семейства Windows NT пространство адресов для непривилегированного режима может быть увеличено до 3 Гбайт (за счет пространства адресов для привилегированного режима) с помощью параметра `/3GB` файла `BOOT.INI`. Такой вариант предусмотрен для приложений, которые были созданы редактором связей с использованием специального параметра, позволяющего воспользоваться этим увеличенным объемом памяти. Вся память, распределяемая приложением, берется из пространства адресов для непривилегированного режима.
- **Главный поток.** Первый поток приложения. Операционная система Windows автоматически распределяет главный поток для каждого запускаемого ею процесса. Этот поток часто называют также *первичным потоком приложения*.

- **Функция точки входа.** Функция, с адреса которой начинается выполнение потока. Для главного потока функцией точки входа является точка входа приложения (часто это — функция, имеющая имя `main()` или подобное); для всех других потоков функция точки входа задается при создании потока и, по сути, представляет собой простую процедуру обратного вызова.

Краткий обзор

Процесс Win32 является полностью пассивным, поскольку в действительности не выполняется и не осуществляет каких-либо действий. Поэтому процесс просто предоставляет контейнер для потоков. При запуске или останове процесса фактически происходит лишь запуск или останов его потоков, поскольку сам процесс не функционирует. Кроме того, с формальной точки зрения прекращение работы процесса не происходит; прекращается лишь работа его потоков. Процесс предназначен исключительно для предоставления ресурсов и контекста, в котором могут функционировать потоки, фактически выполняющие всю работу приложения.

Каждый процесс состоит в основном из двух компонентов — объекта привилегированного режима процесса и пространства виртуальных адресов. Операционная система использует объект привилегированного режима для управления процессом и для предоставления приложениям средств взаимодействия с процессом. Пространство виртуальных адресов содержит код и данные исполняемого модуля, код и данные загружаемых им библиотек DLL, а также результаты распределения динамической памяти, такие как стеки потоков и динамические области памяти.

Процессы предоставляют пространство виртуальных адресов для приложения, а это означает, что для них требуется гораздо больше системных ресурсов, чем для потоков. В частности, значительный объем системных ресурсов необходим для создания пространства виртуальных адресов процесса. Слежение и контроль за этим пространством осуществляется диспетчером памяти операционной системы с использованием дескрипторов виртуальных адресов (Virtual Address Descriptor — VAD), а для этих целей требуются ресурсы, не относящиеся к функционированию самих потоков.

Автор обязан здесь упомянуть, что исполняемый код (и данные), который содержится в пространстве виртуальных адресов процесса, фактически не загружается до тех пор, пока в этом не возникает необходимость. Код и данные лишь отображаются на пространство виртуальных адресов. При этом для кода и данных просто резервируются некоторые диапазоны адресов в этом пространстве, тогда как физической памятью для этих участков остается сам файл EXE или DLL. Код и данные загружаются или выгружаются по мере необходимости операционной системой в виде фрагментов с объемом в одну страницу (в подсистеме Win32, работающей на процессоре x86, объем страницы равен 4 Кбайт). Каждому исполняемому файлу или файлу DLL, отображаемому на пространство адресов процесса, присваивается уникальный дескриптор экземпляра.

Кроме объекта привилегированного режима и пространства виртуальных адресов, в состав процесса входят также перечисленные ниже компоненты.

- Относящаяся к данному процессу таблица открытых дескрипторов системных ресурсов, таких как файлы, обработчики событий, мьютексы (взаимоисключающие блокировки) и семафоры.

- Лексема доступа, которая определяет контекст защиты процесса и идентифицирует пользователя, привилегии и группы защиты, связанные с этим процессом.
- Уникальный идентификатор, называемый *идентификатором процесса* или *идентификатором клиента*.
- По меньшей мере один поток.

Процессы. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к процессам, приведены в табл. 3.1.

Таблица 3.1. Основные функции API-интерфейсов, относящиеся к процессам

Функция	Описание
`CreateProcess`	Создать новый дескриптор процесса
`ExitProcess`	Выйти из текущего процесса
`OpenProcess`	Открыть существующий процесс
`TerminateProcess`	Завершить существующий процесс

Основные инструментальные средства анализа процессов

Как было указано в главе 2, возможности различных инструментальных средств получения информации о системе, а также возвращаемые ими данные, которые касаются системных объектов, во многом перекрываются. Инструментальные средства, относящиеся к процессам, не являются исключением из этого правила. В табл. 3.2 приведены итоговые сведения о том, какую информацию, касающуюся процессов, возвращают некоторые из этих инструментальных средств.

Основные счетчики программы Perfmon

Когда речь идет о работе с процессами, программа Perfmon становится буквально незаменимым инструментальным средством. Некоторые из наиболее важных счетчиков программы Perfmon, относящихся к процессам, перечислены в табл. 3.3.

Внутренняя организация процесса

Каждый поток функционирует в контексте процесса, к которому он принадлежит. Процесс предоставляет потоку ресурсы и среду, в которой поток выполняет свой код. Благодаря тому, что в операционной системе Windows процессы отделены друг от друга, поток не может получить доступ к пространству адресов другого процесса без обращения к секции совместно используемой памяти, а также без применения функций `ReadProcessMemory` или `WriteProcessMemory`.

Таблица 3.2. Инструментальные средства, относящиеся к процессам, и отображаемая ими информация

	Идентификатор процесса	Имя исполняемого файла	Класс приоритета	Процентная доля процессорного времени	Количество дескрипторов	Процентная доля времени непривилегированного режима	Процентная доля времени привилегированного режима	Общее затраченное время	Командная строка
Perfmon	+		+	+	+	+	+	+	+
Pstat	+	+							
Pviewer	+	+	+	+					
Qslice	+	+		+		+	+		
TaskMgr	+	+	+	+	+				
TList	+	+							+

Таблица 3.3. Счетчики программы Perfmon, относящиеся к процессам

Счетчик	Описание
`Process:% Privileged Time`	Время, затраченное процессом в привилегированном режиме
`Process:% User Time`	Время, затраченное процессом в непривилегированном режиме
`Process:% Processor Time`	Общее процессорное время для процесса; значение этого счетчика должно быть равно сумме значений первых двух счетчиков, поэтому может превышать 100% на многопроцессорном компьютере
`Process:Elapsed Time`	Количество секунд, истекших со времени создания процесса
`Process:ID Process`	Внутренний идентификатор процесса
`Process:Creating Process ID`	Внутренний идентификатор процесса, создавшего рассматриваемый процесс
`Process:Thread Count`	Количество потоков, содержащихся в настоящее время в рассматриваемом процессе
`Process:Handle Count`	Количество дескрипторов в относящейся к процессу таблице дескрипторов

По своей внутренней организации любой процесс представлен в виде блока исполнительного процесса (Executive PROCESS – EPROCESS), который хранится в пространстве памяти системы наряду с относящимися к нему структурами данных. Из этих внутренних структур в пространстве адресов процесса хранится только блок среды процесса (Process Environment Block – PEB).

Сразу после создания объекта процесса (с помощью вызова функции `CreateProcess`) операционная система Windows создает объект привилегированного режима процесса с первоначальным значением счетчика использования, равным 1. Следует отметить, что этот объект представляет собой не сам процесс,

а небольшую структуру данных, которая используется ОС Windows для управления процессом и отслеживания статистической информации о процессе.

После создания объекта привилегированного режима операционная система Windows создает пространство виртуальных адресов процесса и отображает на это пространство код и данные исполняемого модуля (а также любых других необходимых модулей).

Следует отметить, что функция `CreateProcess` может вернуть код завершения TRUE еще до того, как процесс был полностью инициализирован, и даже до того, как загрузчик операционной системы находит все библиотеки DLL, необходимые для исполняемого файла. Если же требуемый модуль не удается найти или попытка его инициализации оканчивается неудачей, то ОС Windows завершает новый процесс. При этом, поскольку функция `CreateProcess` уже могла возвратить код завершения TRUE, прежде чем об этом стало известно, процесс, создающий новый процесс, может не получить информации о возникшей проблеме и предпринять ошибочную попытку использовать дескриптор процесса или потока, возвращенный функцией `CreateProcess`. Одним из способов предотвращения такой ситуации является проверка состояния процесса с помощью функции API-интерфейса `GetExitCodeProcess` до осуществления попытки использовать дескриптор процесса или дескриптор главного потока. Функция `GetExitCodeProcess` при попытке передать ей недопустимый дескриптор возвращает FALSE. Кроме того, можно заключать вызовы функции `CloseHandle` в блоки обработки исключительных ситуаций, с тем, чтобы эти блоки перехватывали исключительные ситуации `INVALID_HANDLE`, которые активизируются, если функции `CloseHandle` передается неправильный дескриптор.

После создания процесса система автоматически создает его первичный (или главный) поток. Каждый процесс имеет по меньшей мере один поток, представленный с помощью блока исполнительного потока (Executive THREAD – ETHREAD) в памяти системы.

После создания нового процесса с использованием функции `CreateProcess` необходимо закрыть дескриптор его главного потока для того, чтобы дать возможность системе освободить соответствующий объект потока после того, как он больше не будет нужен. В результате закрытия этого дескриптора работа потока не прекращается, поскольку при этом лишь освобождается ссылка на него из вызывающей функции. Сохранение дескриптора первичного потока, запущенного на выполнение процессом, не дает никаких преимуществ, если только в замысел экспериментатора не входит попытка непосредственно манипулировать этим потоком с помощью вызовов API-интерфейса. Если же требуется дождаться завершения процесса перед переходом к дальнейшей работе, можно передать в функцию ожидания сам дескриптор процесса.

Во время инициализации нового процесса указатель команд для первичного потока процесса устанавливается на недокументированную и неэкспортируемую функцию, называемую `BaseProcessStart`. Именно с этого начинается выполнение всех новых процессов в подсистеме Win32.

Другие внутренние структуры

Блоки PEB, EPROCESS и относящиеся к ним структуры не являются единственными внутренними структурами, используемыми операционной системой Windows для слежения за процессами. Например, параллельную структуру для каждого процесса, в котором выполняется любая программа Win32, поддерживает процесс подсистемы Win32 (`Csrss`). Кроме того, в относящейся к привилегированному режиму части подсистемы Win32 (`Win32k.sys`) для каждого процесса поддерживается по одной структуре данных, создаваемой при первом же вызове в любом потоке процесса такой функции, как функция API-интерфейса USER или GDI подсистемы Win32, которая реализована в привилегированном режиме.

Завершение процесса

Процесс может быть завершен с использованием одного из перечисленных ниже четырех методов.

1. Возврат из функции точки входа первичного потока. Это — самый предпочтительный метод, поскольку он гарантирует выполнение описанных ниже действий.
   - 1.1. Для любых объектов C++, созданных потоком, вызываются их деструкторы и обеспечивается возможность выполнить код очистки, предусмотренный в библиотеке этапа прогона (RunTime Library — RTL).
   - 1.2. Операционная система Windows немедленно освобождает память, используемую стеком потока.
   - 1.3. В качестве кода выхода процесса задается значение, возвращаемое функцией точки входа потока.
   - 1.4. Операционная система Windows уменьшает значение в счетчике использования объекта привилегированного режима соответствующего процесса.
2. Вызов функции `ExitProcess` в одном из потоков процесса. Следует избегать использования такого варианта, поскольку он исключает возможность вызова деструкторов объектов C++ и кода очистки RTL.
3. Вызов функции `TerminateProcess` в потоке другого процесса. Этого варианта также следует избегать, поскольку он не позволяет передать в библиотеки DLL процесса извещение о его завершении, а также не дает возможности вызывать на выполнение деструкторы объектов и процедуры очистки RTL. Следует отметить, что функция `TerminateProcess` является асинхронной; это означает, что не гарантируется завершение процесса к моменту возврата из этой функции. Чтобы приостановить выполнение вызывающего потока до тех пор, пока процесс не будет отмечен как сигнальный (не завершится), можно применить функцию `WaitForSingleObject`.
4. Самостоятельное завершение работы всеми потоками в процессе. Такая ситуация встречается довольно редко.

После нормального завершения любого процесса не происходит абсолютно никаких утечек ресурсов. Операционная система Windows гарантирует, что все ресурсы, распределенные процессом, освобождаются после выхода из этого процесса.

Говоря о завершении процесса, следует рассмотреть, что происходит, когда процесс завершается.

- Закрываются все дескрипторы объектов, открытые процессом.
- Завершаются все потоки в процессе.
- Объект процесса привилегированного режима отмечается как сигнальный.
- Все потоки процесса отмечаются как сигнальные.
- Функция `GetExitCodeProcess` возвращает значение кода выхода процесса, а не значение `STILL_ACTIVE`.

Функция `SetErrorMode`

С каждым процессом связано множество флажков. Эти флажки сообщают системе, как данный процесс должен отвечать на серьезные ошибки, подобные необработанным исключительным ситуациям, отказам носителей и ошибкам при открытии файлов. Для установки этих флажков применяется вызов функции `SetErrorMode` API-интерфейса. В табл. 3.4 приведены сводные данные о доступных при этом опциях. Программа SQL Server вызывает эту функцию при запуске, и, как правило, ее не следует никогда вызывать из кода, работающего в рамках процесса SQL Server (например, из расширенной процедуры или из внутрипроцессного COM-объекта), поскольку при этом может нарушиться способность сервера обрабатывать ошибки.

Таблица 3.4. Параметры функции `SetErrorMode` и их значения

Значение	Описание
0	Выполнять предусмотренное по умолчанию действие системы – отображать все диалоговые окна по исправлению ошибки
`SEM_FAILCRITICALERRORS`	Не отображать диалоговые окна по исправлению критических ошибок; вместо этого передавать сообщение об ошибке в вызывающий процесс
`SEM_NOALIGNMENTFAULTEXCEPT`	Автоматически исправлять все проблемы выравнивания адресов памяти (этот параметр применяется только в системах с процессорами RISC)
`SEM_NOGPFAULTERRORBOX`	Не отображать диалоговое окно General Protection Fault
`SEM_NOOPENFILEERRORBOX`	Не отображать диалоговое окно, когда происходит ошибка поиска файла; вместо этого передавать сообщение об ошибке в вызывающий процесс

Упражнения

В этих упражнениях показано, как осуществляется текущий контроль различных аспектов функционирования процесса SQL Server. В ходе их выполнения будут проверены степень загрузки процессора, а также состояния потоков, созданных в процессе SQL Server, и состав загруженных в нем модулей. Точно такие же методы могут использоваться для контроля над процессами Win32 других типов.

Упражнение 3.1. Текущий контроль загрузки процессора в рассматриваемом процессе

В данном упражнении показано, как осуществляется текущий контроль ситуаций пиковой загрузки процессора в процессе SQL Server. С его помощью можно узнать, как обеспечить текущий контроль загрузки процессора с помощью программы Task Manager. Выполнение данного упражнения состоит из перечисленных ниже шагов.

1. Запустите программу SQL Server (система, в которой проводятся эксперименты, не должна быть производственной системой), если она еще не запущена, и подключитесь к ней с помощью программы Query Analyzer. В идеальной ситуации вы должны быть единственным пользователем в этой системе.
2. Запустите программу Task Manager (например, с помощью нажатия клавиш <Ctrl+Shift+Esc>).
3. Щелкните на вкладке Processes, затем — на столбце CPU, чтобы отсортировать список для перевода в верхнюю часть списка процессов с высокой степенью загрузки процессора.
4. Сверните окно программы Task Manager, затем снова переключитесь на программу Query Analyzer и выполните следующий запрос:

```
declare @var int
set @var=1
while @var<100000 begin
  set @var=@var+1
end
```

5. Пока выполняется данный запрос, переключитесь на программу Task Manager. В списке Processes вы должны обнаружить, что строка с именем программы `sqlservr.exe` переместилась в верхнюю часть списка.

Этот эффект будет более ярко выражен на однопроцессорных компьютерах и, безусловно, на компьютерах с более медленными процессорами, но в любом случае в той или иной форме должен обнаруживаться пик загрузки процессора под действием рассматриваемого процесса SQL Server.

Для того чтобы увидеть графическое изображение такого пика, снова выполните данный запрос и переключитесь на вкладку Performance в программе Task Manager. На однопроцессорном компьютере должна быть получена диаграмма, аналогичная приведенной на рис. 3.1.

Упражнение 3.2. Текущий контроль над созданием потоков в процессе SQL Server

В многопотоковых приложениях обычно применяется один из двух подходов при принятии решения о создании новых потоков. В первом подходе все потоки создаются

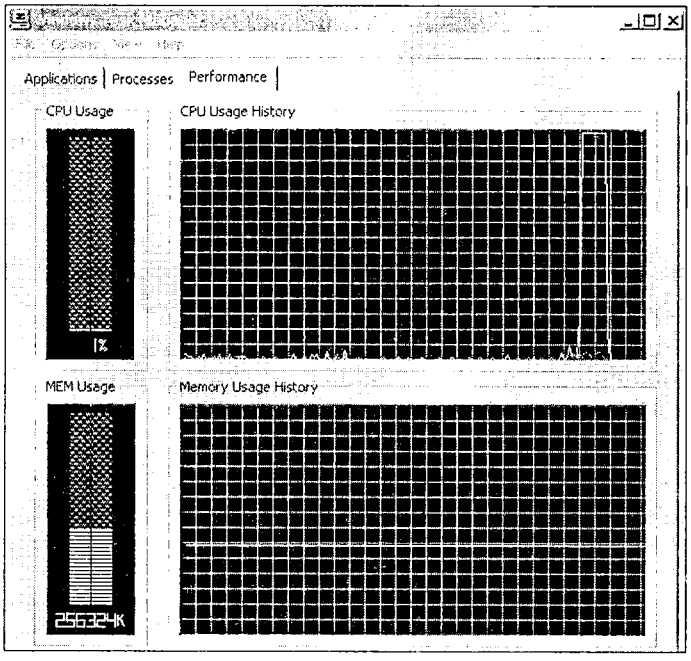


Рис. 3.1. Пик загрузки процессора под действием процесса SQL Server, который показывает, что потребность этого процесса в ресурсах процессора может намного превысить потребности всех других процессов, работающих на компьютере

при запуске приложения, а при втором — в ходе работы приложения. В программе SQL Server новые рабочие потоки создаются по мере необходимости вплоть до максимально допустимого количества рабочих потоков, заданных с помощью параметра процедуры `sp_configure` (или с помощью программы Enterprise Manager). Чтобы понять, как осуществляется эта настройка, выполните следующие шаги.

1. Остановите и перезапустите программу SQL Server. Этот эксперимент не должен проводиться в производственной системе, а на протяжении всего эксперимента вы должны быть единственным пользователем системы.
2. Запустите программу Perfmon и щелкните на кнопке Add Counters.
3. В диалоговом окне Add Counters измените значение параметра Performance Object на Process.
4. В списке экземпляров сервера найдите обозначение процесса `sqlservr` и щелкните на нем. (Этому экземпляру может быть присвоено обозначение `#1`, `#2` или какой-то другой номер; если на компьютере работает несколько экземпляров программы SQL Server, обязательно проверьте, правильно лжи выбран вами используемый экземпляр.)
5. Выберите значение Thread Count из списка счетчиков.
6. Щелкните на кнопке Add. В программе Perfmon должна появиться линейная диаграмма с указанием текущего количества потоков, функционирующих в рамках процесса SQL Server. Сделайте отметку о том, какое количество

потоков работает в настоящее время (в поле Last должно быть указано точное количество потоков за последний интервал выборки). На рис. 3.2 показано, как добавить указанный счетчик.

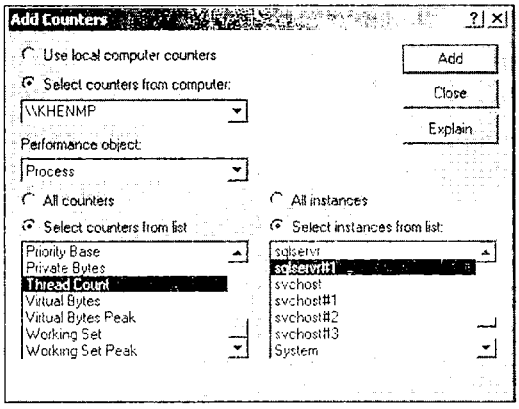


Рис. 3.2. Добавление счетчика количества потоков в сеансе Perfmon

7. Откройте командное окно, перейдите в каталог, в котором вы можете копировать файлы и вызывать на выполнение те или иные тесты, после чего скопируйте в этот каталог файлы `STRESS.CMD` и `STRESS.SQL` из каталога `CH03` компакт-диска, прилагаемого к этой книге. Пакетный файл `STRESS.CMD` вызывает на выполнение указанный сценарий (или сценарии) T-SQL с использованием указанного количества соединений (в нем вызывается программа `osql.exe`). Этот пакетный файл можно использовать для моделирования той ситуации, в которой к серверу подключаются многочисленные пользователи и выполняют заданный запрос или несколько запросов. С помощью вызова пакетного файла `STRESS.CMD` без параметров ознакомьтесь со справкой по его использованию. Файл сценария `STRESS.SQL` просто выводит на внешнее устройство содержимое таблицы `pubs..authors`, а затем устанавливает паузу на 15 секунд с помощью команды `WAITFOR DELAY` языка T-SQL. Запустите пакетный файл `STRESS.CMD` с помощью следующей командной строки:

```
STRESS STRESS.SQL 15 N normal Y YourServerName\Instance
```

Укажите вместо параметров *`YourServerName\Instance`* имя вашего компьютера и экземпляр процесса SQL.

Должно появиться 15 открытых командных окон, в каждом из которых выполняется файл сценария `STRESS.SQL`.

8. После этого снова переключитесь на программу Perfmon. Вы должны обнаружить заметное увеличение количества потоков в процессе SQL Server. Это увеличение обусловлено тем, что новые моделируемые соединения вынуждают сервер создавать новые рабочие потоки для их обслуживания. Обратите внимание на то, что строгой зависимости между количествами рабочих потоков и соединений не существует. Программа SQL Server часто проявляет способность эффективно обслуживать тысячи соединений с помощью всего лишь нескольких сотен рабочих потоков.

9. Примерно через 15 секунд после запуска пакетного файла `STRESS.CMD` вы должны обнаружить, что открытые командные окна автоматически закрываются. Тем не менее количество рабочих потоков процесса SQL Server уменьшиться не должно. Сервер не уничтожает вновь созданные рабочие потоки сразу после того, как они перестают функционировать, даже если они в настоящее время простаивают, поскольку эти потоки могут потребоваться для обслуживания очередной волны запросов, поступающих один за другим. Такое кэширование простаивающих рабочих потоков позволяет программе SQL Server обеспечивать стабильную производительность в той среде, где количество соединений и запросов, передаваемых на сервер, в значительной степени изменяется во времени. Вместе с тем, сервер со временем прекращает использование ненужных системных ресурсов; примерно через 15 минут процесс SQL Server обнаруживает с помощью тайм-аута простаивающий рабочий поток и уничтожает его.

Следует отметить, что количество потоков процесса можно также определить с помощью многих других инструментальных средств; для этого не обязательно использовать Perfmon. Автору особенно нравится программа Pview (из состава инструментальных средств комплекта Platform SDK). На рис. 3.3 показано, как узнать количество потоков в процессе SQL Server с помощью программы Pview.

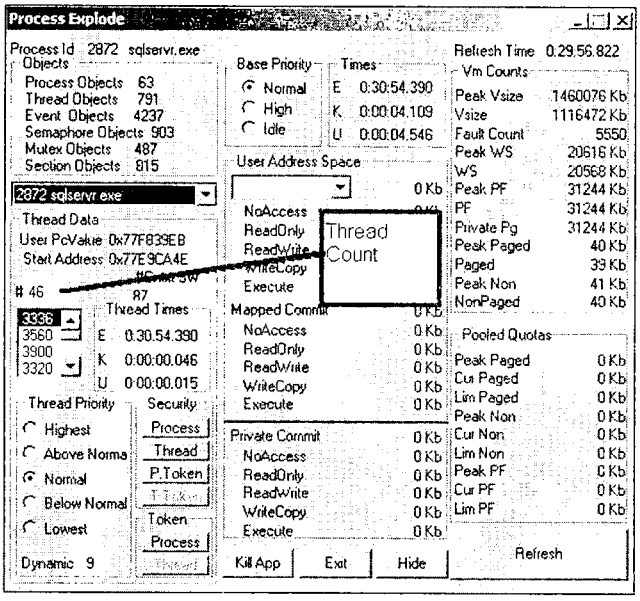


Рис. 3.3. Определение количества потоков с помощью программы Pview

Упражнение 3.3. Ознакомление со списками модулей и процессов в программе SQL Server

В этом последнем упражнении для составления списков загруженных в настоящее время модулей и процессов программы SQL Server используется специализированная расширенная процедура `xp_modlist`. Вы ознакомитесь со всеми библиотеками DLL,

находящимися в пространстве процессов SQL Server, а также со всеми исполняемыми файлами, вызванными на выполнение в этой программе. В системе Windows не предусмотрен официально рекомендованный механизм для установления родительско-дочерних связей между процессами, но существуют недокументированные функции, которые могут использоваться в любой процедуре для получения информации о таких связях. Для вызова на выполнение процедуры `xp_modlist` в программе SQL Server выполните перечисленные ниже пункты.

1. Запустите программу Query Analyzer и подключитесь к экземпляру SQL Server, предназначенному для разработки или экспериментирования. Используемая система не должна быть производственной системой, и в идеальной ситуации вы должны быть ее единственным пользователем.
2. Скопируйте файл `xp_sysinfo.dll` из подкаталога `CH03\xp_sysinfo\release` компакт-диска, прилагаемого к этой книге, в подкаталог `binn` корневого инсталляционного каталога SQL Server.
3. Установите библиотеку `xp_modlist`, выполнив следующую команду в программе Query Analyzer:

   ```
   sp_addextendedproc 'xp_modlist','xp_sysinfo.dll'
   ```

4. Выполните следующую команду, чтобы создать экземпляр дочернего процесса в программе SQL Server:

   ```
   xp_cmdshell 'notepad'
   ```

   Примечание. Вы не увидите, как будет запущена программа Notepad, если только вы не работаете с программой SQL Server как с терминальным приложением. Не беспокойтесь: запуск этой программы производится лишь для того, чтобы создать подпроцесс, который никогда не завершается.
5. Создается впечатление, что в выполненном вызове `xp_cmdshell` произошло зависание. Откройте новое окно Query Analyzer и выполните следующую команду:

   ```
   xp_modlist
   ```

   Вы должны увидеть список процессов, который выглядит примерно так, как показано в листинге 3.1.

Листинг 3.1. Список процессов

```
ParentProcessID ProcessID Handle     ModuleName
--------------- --------- ---------- ----------------------------
264             3544      0x00400000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x77F80000 C:\WINNT\System32\ntdll.dll
264             3544      0x77E80000 C:\WINNT\system32\KERNEL32.DLL
264             3544      0x77DB0000 C:\WINNT\system32\ADVAPI32.DLL
264             3544      0x77D30000 C:\WINNT\system32\RPCRT4.DLL
264             3544      0x77E10000 C:\WINNT\system32\USER32.DLL
264             3544      0x77F40000 C:\WINNT\system32\GDI32.dll
264             3544      0x41060000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x41070000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x42AE0000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x41080000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x25900000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x410D0000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x26A70000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x26B10000 C:\PROGRA~1\MICROS~3\MSSQL$~2
```

```
264             3544      0x10000000 C:\PROGRA~1\MICROS~3\MSSQL$~2
264             3544      0x10010000 C:\WINNT\system32\PSAPI.DLL
3544            652       0x4AD00000 C:\WINNT\system32\cmd.exe
3544            652       0x77F80000 C:\WINNT\System32\ntdll.dll
3544            652       0x77E80000 C:\WINNT\system32\KERNEL32.dll
3544            652       0x77E10000 C:\WINNT\system32\USER32.dll
3544            652       0x77F40000 C:\WINNT\system32\GDI32.dll
652             3468      0x01000000 C:\WINNT\system32\notepad.exe
652             3468      0x77F80000 C:\WINNT\System32\ntdll.dll
652             3468      0x77E80000 C:\WINNT\system32\KERNEL32.dll
652             3468      0x77F40000 C:\WINNT\system32\GDI32.dll
652             3468      0x77E10000 C:\WINNT\system32\USER32.dll
652             3468      0x77DB0000 C:\WINNT\system32\ADVAPI32.dll
652             3468      0x76620000 C:\WINNT\system32\MPR.DLL
```

Обратите внимание на то, что в этом списке используются три разных значения `ParentProcessId`. После запуска команды `xp_cmdshell` вызывается командный интерпретатор `Cmd.exe`, который, в свою очередь, вызывает программу `Notepad.exe`, поскольку ее имя было передано команде `xp_cmdshell`. В команде `xp_cmdshell` выполняется вызов процессов с помощью командного интерпретатора `cmd.exe`, поэтому она позволяет использовать такие службы командного интерпретатора, как конвейерный ввод-вывод и перенаправление.

6. Если программа SQL Server эксплуатируется как служба, то может потребоваться выполнить перезагрузку серверного компьютера, чтобы избавиться от только что запущенного экземпляра Notepad, в зависимости от того, в какой учетной записи пользователя вы зарегистрировались и какими правами обладает эта учетная запись.

Процессы. Резюме

Хотя обычно принято считать процессы такими объектами, которые выполняют в приложении всю работу, в действительности процесс представляет собой лишь среду предоставления контекста для потоков. В системе Windows всю работу в приложении выполняют потоки, а процессы предоставляют среду, в которой осуществляется эта работа.

SQL Server — это процесс, который функционирует либо в качестве службы, либо в качестве приложения терминального режима. Как и все процессы Windows, он состоит из потоков, пространства виртуальных адресов, объектов привилегированного режима и т.д.

Процессы. Вопросы для самопроверки

1. Какая процедура API-интерфейса Windows используется для создания нового объекта процесса?
2. Какой метод завершения процесса предпочтительнее?
3. Возможно ли, что в приложении может обнаружиться утечка ресурсов, связанных с дескрипторами, после его завершения?

4. Какой результат возвращает функция `GetExitCodeProcess` после ее вызова с указанием процесса, который все еще выполняется?
5. В какой из двух основных секций пространства адресов процесса (непривилегированного или привилегированного режима) хранится блок среды процесса (Process Environment Block — PEB)?
6. Какой объем из общего объема 32-битового пространства адресов процесса, равного 4 Гбайт, зарезервирован по умолчанию для пространств привилегированного и непривилегированного режима?
7. Подтвердите или опровергните следующее утверждение. Существует возможность создать новый процесс, который не будет иметь главного потока, но это не рекомендуется, поскольку в соответствующем приложении нельзя будет выполнить какую-либо работу.
8. Какой наибольший объем пространства непривилегированного режима может быть отведен приложению в 32-битовой операционной системе Windows?
9. В каком файле содержится часть подсистемы Win32, относящаяся к привилегированному режиму?
10. Подтвердите или опровергните следующее утверждение. Функцию `SetErrorMode` можно безо всяких опасений вызывать из расширенной хранимой процедуры при условии, что вызов этой функции будет встроен в блок обработки исключительных ситуаций.
11. Каково состояние отмеченности в качестве сигнального такого процесса, который завершен с использованием программы Task Manager?
12. Какой параметр файла `BOOT.INI` используется для настройки конфигурации ОС Windows, чтобы предоставить приложениям дополнительное пространство адресов непривилегированного режима?
13. Подтвердите или опровергните следующее утверждение. Средства, применяемые в операционной системе Windows для защиты процессов друг от друга, исключают возможность того, чтобы один процесс вносил изменения в память, распределенную в пределах пространства адресов другого процесса.
14. Подтвердите или опровергните следующее утверждение. Функция `CreateProcess` может возвратить значение `TRUE`, даже если новый процесс нельзя было создать из-за того, что не удалось найти библиотеку DLL, от которой зависит этот процесс.

Потоки

Поток — это средство, с помощью которого в системе Windows выполняется код программы. Поток — это единственный объект Windows, способный выполнять код. Поток всегда создается в контексте процесса и на протяжении всего своего существования функционирует в пределах этого процесса.

Поток — это единственное средство, с помощью которого в процессе выполняется какая-либо работа.

Потоки. Основные термины и определения

- **Функция точки входа.** Адрес функции, с которого начинается выполнение потока. Для основного потока этот адрес представляет собой точку входа приложения (часто это — функция, имеющая имя `main()`, или аналогичная ей); для всех других потоков функция точки входа задается при создании потока и, по сути, представляет собой простую процедуру обратного вызова.
- **Поток.** Объект Windows, с помощью которого выполняется код приложения. Каждый поток функционирует в контексте приложения или операционной системы.
- **Переключение контекста.** Событие, которое происходит, когда операционная система Windows сохраняет в памяти контекстную информацию, относящуюся к одному потоку, и загружает информацию, относящуюся к другому потоку, для того, чтобы этот поток мог работать. Выполняемые при этом действия включают сохранение или загрузку значений временных регистров и других элементов, относящихся к среде прогона данного потока.
- **Микропоток.** Облегченная, подобная потоку конструкция непривилегированного режима, которая функционирует в контексте потока. В режиме микропотоков на работу программы SQL Server налагается целый ряд ограничений, которые могут оказаться неприемлемыми (например, при работе в режиме микропотоков не может использоваться модуль SQLXML). По этой причине режим микропотоков обычно не применяется, но иногда позволяет добиться более высокой масштабируемости благодаря снижению нагрузки по переключению контекста.
- **Операции SEH.** Операции структурированной обработки исключительных ситуаций (Structured Exception Handling — SEH), особенно тех, которые предусмотрены в системе Windows. Конструкции SEH предоставляют механизмы, необходимые для обеспечения возможности логически отделять в приложении задачи, которые должны быть в нем выполнены, от действий, которые должны быть предприняты, если выполнение одной из этих задач окончится неудачей по той или иной причине.
- **Структура TLS.** Структура TLS (Thread Local Storage — локальная память потока) — это механизм, с помощью которого в потоках можно сохранять в памяти данные, уникальные для каждого потока. Доступ к значениям TLS осуществляется с помощью индекса. Для распределения индекса TLS используется функция `TlsAlloc` API-интерфейса Win32, а для сохранения и получения значений служат функции `TlsSetValue` и `TlsGetValue`.

Потоки. Основные функции API-интерфейсов

Основные функции API-интерфейса Win32, относящиеся к потокам, приведены в табл. 3.5.

Таблица 3.5. Основные функции API-интерфейса Win32, относящиеся к потокам

Функция	Описание
`CreateThread`	Создать новый объект потока
`ExitThread`	Выйти из текущего потока
`TerminateThread`	Завершить работу другого потока
`GetExitCodeThread`	Получить код возврата функции точки входа потока

СОВЕТ. Обратите внимание на то, что в любую функцию API-интерфейса Win32, для которой требуется дескриптор процесса или дескриптор потока, можно передать псевдодескриптор. Это вынуждает рассматриваемую функцию выполнить свое действие в вызывающем процессе или вызывающем потоке.

Основные инструментальные средства, относящиеся к потокам

Как было указано в главе 2, функциональные возможности различных системных диагностических инструментальных средств в значительной степени перекрываются; диагностические инструментальные средства, относящиеся к потокам, не составляют исключения из этого правила. В табл. 3.6 приведена итоговая информация о способах получения основной информации о потоках.

Таблица 3.6. Основные диагностические инструментальные средства, относящиеся к потокам, и информация, которую они возвращают

	Идентификатор потока	Начальный адрес	Процентная доля процессорного времени	Количество переключений контекста	Состояние потока	Процентная доля времени непривилегированного режима	Процентная доля времени привилегированного режима	Причина последнего перехода в состояние ожидания	Последняя ошибка
Perfmon	+	+	+	+	+	+	+	+	
Pstat	+	+		+					
Pviewer	+	+		+	+	+	+	+	
Qslice			+			+	+		
TList	+				+			+	+

Основные счетчики программы Perfmon

Программа Perfmon является бесценным инструментальным средством текущего контроля за использованием потоков в приложении и во всей системе. В табл. 3.7 перечислены некоторые наиболее полезные счетчики программы Perfmon, связанные с потоком.

Таблица 3.7. Полезные счетчики производительности, относящиеся к потокам

Счетчик	Описание
`Process:Priority Base`	Основной приоритет выполнения процесса, к которому относится поток
`Thread:% Privileged Time`	Процентная доля времени, затраченного потоком в привилегированном режиме
`Thread:% User Time`	Процентная доля времени, затраченного потоком в непривилегированном режиме
`Thread:% Processor Time`	Процентная доля времени, в течение которого поток использовал процессор; должна быть равна сумме значений счетчиков `% Privileged Time` и `% User Time`
`Thread:Context Switches/sec`	Количество переключений контекста в секунду
`Thread:Elapsed Time`	Общее время, истекшее с момента запуска потока
`Thread:Id Process`	Внутренний идентификатор процесса, к которому относится поток
`Thread:Id Thread`	Внутренний идентификатор потока
`Thread:Priority Base`	База приоритета потока
`Thread:Priority Current`	Текущий приоритет потока
`Thread:Start Address`	Начальный адрес потока
`Thread:Thread State`	Текущее состояние потока — целое число от 0 до 7 (см. табл. 3.11 на стр. 110)
`Thread:Thread Wait Reason`	Если поток находится в состоянии ожидания, то причина перехода в это состояние

Внутреннее устройство потока

Поток Win32 включает перечисленные ниже компоненты.

- Набор временных регистров, которые представляют состояние процессора.
- Стек для выполнения кода в привилегированном режиме.
- Стек для выполнения кода в непривилегированном режиме.
- Область TLS.
- Уникальный идентификатор, известный под названием *идентификатора потока*. (В составе отдельного потока этот идентификатор называют также *идентификатором клиента*; идентификаторы процесса и потока никогда не перекрываются, поскольку они создаются в одном и том же пространстве имен.)
- Необязательный контекст защиты. (По умолчанию поток наследует контекст защиты от своего родительского процесса. Многопоточные приложения иногда получают отдельную лексему доступа для каждого отдельного потока, чтобы можно было обеспечить анонимность контекста защиты обслуживаемых ими клиентов.)

Область TLS, регистры и стеки потока известны под общим названием *контекста потока*. Данные об этих компонентах хранятся в структуре CONTEXT потока. Структура CONTEXT – это единственная зависимая от процессора структура в API-интерфейсе Win32. Сама эта структура находится в объекте привилегированного режима потока.

Двумя наиболее важными элементами, хранящимися в структуре CONTEXT, являются указатель команд и регистры указателя стека потока. При инициализации объекта привилегированного режима потока указатель стека устанавливается на адрес местонахождения функции точки входа потока в стеке потока, а указатель команд устанавливается на недокументированную функцию `BaseThreadStart`.

Все потоки в процессе совместно используют пространство адресов виртуальной памяти и таблицу дескрипторов процесса. По умолчанию потоки наследуют также лексему защиты доступа процесса, хотя, если это потребуется, могут получить и собственную лексему защиты для обеспечения анонимности клиента (как было указано выше). К тому же, как было сказано в разделе "Процессы", система предотвращает доступ потоков к пространству адресов других процессов без секции совместно используемой памяти, а также предотвращает использование функций `ReadProcessMemory` или `WriteProcessMemory` API-интерфейса.

На уровне системы любой поток представлен с помощью блока ETHREAD. Аналогично блоку процесса EPROCESS блок ETHREAD и относящиеся к нему структуры находятся в пространстве адресов системы. Единственным исключением является блок среды потока (Thread Environment Block – TEB), который находится в пространстве адресов непривилегированного режима. Блок TEB хранит информацию контекста для загрузчика образа и различных библиотек DLL Win32, но находится в пространстве процесса, поскольку для загрузчика образа и библиотек DLL требуется структура, в которую может осуществляться запись в непривилегированном режиме.

В подсистеме Win32 (`Csrss`) существует параллельная структура для каждого потока, создаваемая в любом процессе Win32. А для потоков, в которых вызвана функция API-интерфейса USER или GDI подсистемы Win32, та часть подсистемы Win32, которая относится к привилегированному режиму (а именно `Win32k.sys`), сопровождает параллельную структуру данных (структуру `W32THREAD`), на которую ссылается блок ETHREAD.

Первичный поток

При создании функцией `CreateProcess` нового процесса система обеспечивает автоматическое создание первого потока этого процесса. Такой поток обычно называют *первичным* или *главным потоком*. В однопотоковых приложениях он является единственным потоком, содержащимся в процессе. А в многопотоковых приложениях первичный поток либо порождает другие потоки в процессе, либо взаимодействует с ними; эти потоки обычно называют *рабочими* или *фоновыми потоками*.

Процесс, завершивший свою работу и готовый к закрытию, должен передать всем созданным им рабочим потокам сигнал, что они должны выполнить возврат из своих функций точки входа, а после этого просто выполняется возврат из точки

входа в главный поток. Возврат из функции точки входа первичного потока обеспечивает перечисленные ниже преимущества.

- Для любых объектов, созданных потоком, вызываются их деструкторы, чтобы уничтожение этих объектов было выполнено должным образом.
- Система Windows немедленно освобождает память, используемую стеком потока.
- Код завершения процесса устанавливается равным значению, возвращаемому функцией точки входа.
- Система уменьшает значение счетчика использования объекта привилегированного режима процесса для процесса, владеющего потоком.

Сравнение процессов и потоков

Потоки используют меньше системных ресурсов, чем процессы, поэтому всегда по возможности следует стремиться решать проблемы, возникающие в процессе программирования, с помощью потоков, а не процессов. Издержки, связанные с управлением пространством виртуальных адресов, являются незначительными, тогда как в случае создания на компьютере слишком большого количества процессов может быть исчерпана вся виртуальная память системы, а производительность снизится почти до нуля.

Из сказанного выше не следует, что любую задачу лучше решать с использованием многочисленных потоков, а не процессов. Некоторые проекты можно более успешно реализовать с помощью отдельных процессов. Автор рекомендует изучить все компромиссные варианты, относящиеся к каждому подходу, и сопоставить их друг с другом, прежде чем принимать решение о дальнейших действиях.

Тем не менее многопотоковая организация приложения позволяет не только значительно сэкономить ресурсы системы, но и упростить интерфейс приложения. Если некоторые задачи, активизируемые при обычных условиях с помощью щелчков на элементах меню или на кнопках, можно выполнять автоматически в фоновом режиме с помощью отдельных потоков, то появляется возможность полностью удалить относящиеся к ним элементы пользовательского интерфейса. Но следует учитывать, что в большинстве приложений для проведения всех обновлений в пользовательском интерфейсе должен применяться единственный поток. Это связано с тем, что, в отличие от объектов других типов, дескрипторы окон, принадлежащие к таким компонентам пользовательского интерфейса, как кнопки и текстовые поля, фактически принадлежат отдельным потокам, а не родительскому процессу. Поэтому подход, при котором приходится синхронизировать многочисленные потоки пользовательского интерфейса для того, чтобы приложение отображало информацию и работало правильно, обычно становится неоправданно трудоемким.

Многопотоковая организация обеспечивает также масштабирование приложений. Если на используемом компьютере имеется несколько процессоров, то выполнение многочисленных задач можно обеспечить в полном смысле этого слова одновременно, создав несколько потоков, каждый из которых получит доступ к отдельному процессору.

Следует учитывать, что многопотоковая организация работы, несмотря на все ее преимущества, не может служить наилучшим способом решения любых проблем. Попытка решения задач с помощью многочисленных потоков приводит к такому усложнению приложения, которое не возникло бы с использованием другого подхода, поэтому всегда приходится учитывать возможные компромиссы. Некоторые разработчики считают, что, сталкиваясь со сложной задачей, необходимо в первую очередь распределить реализуемые функции по нескольким потокам. Но это — сомнительная практика проектирования, которая может создать лишние трудности для того разработчика приложений, который ей безоговорочно следует.

Создание и уничтожение потоков

После вызова функции `CreateThread` система создает объект привилегированного режима потока для управления этим потоком. Первоначально в счетчике использования этого объекта находится значение 2. Поэтому объект привилегированного режима потока не уничтожается до тех пор, пока не завершается работа потока и не закрывается дескриптор, возвращенный функцией `CreateThread`.

Создание потока приводит к инициализации его стека. Память для этого стека распределяется из пространства виртуальных адресов процесса, поскольку потоки не имеют собственного пространства адресов. Как только будет выполнено распределение памяти для стека, система записывает в верхнюю область этого стека два значения (стеки потоков всегда наращиваются от верхних адресов памяти к нижним): адрес функции точки входа, который был передан функции `CreateThread`, и значение определяемого пользователем параметра, который был передан вместе с этим адресом.

Выбор процедуры, определяемой значением указателя команд, который применяется непосредственно после инициализации потока, зависит от типа создаваемого потока. Если инициализируемым потоком является главный поток процесса, то указатель команд ссылается на недокументированную (и неэкспортируемую) функцию `BaseProcessStart`. Если рассматриваемым потоком является тот поток, который создается приложением (рабочий, или фоновый, поток), то указатель команд ссылается на функцию `BaseThreadStart` (также недокументированную и неэкспортируемую).

ПРИМЕЧАНИЕ. Обычно разработчиков не интересует идентификатор вновь созданного потока, поскольку для взаимодействия с потоком им достаточно получить дескриптор этого объекта. Поэтому в функции `CreateThread` допускается передавать `NULL` в качестве значения ее последнего параметра, `lpThreadId`, и такая практика находит широкое применение среди опытных разработчиков NT. Подобный подход вполне применим в семействе операционных систем Windows NT (которое включает Windows NT, Windows 2000 и все последующие версии Windows), но может вызвать нарушение доступа в Windows 9x. Если необходимо, чтобы разрабатываемый код работал в системе Win9x, требуется передать в качестве этого параметра некоторое значение, независимо от того, действительно вы намереваетесь выполнять что-либо с его помощью или нет.

Завершение работы потока

Завершить работу потока можно с помощью одного из описанных ниже четырех методов.

1. Выполнить возврат из функции точки входа потока. Это — самый лучший способ закрыть поток. Он гарантирует вызов деструкторов объектов C++, выполнение кода очистки RTL, сброс буферов на диск и т.д.
2. Предусмотреть в потоке самоуничтожение путем вызова функции `ExitThread`. Такой подход исключает возможность вызывать деструкторы объектов C++ и выполнять код очистки RTL, поэтому его следует избегать.
3. Вызвать функцию `TerminateThread` из того же или другого процесса. Но функцию `TerminateThread` не следует использовать для уничтожения потока по трем перечисленным ниже причинам.
   - 3.1. поток не получает никаких извещений о том, что происходит его уничтожение. Вполне естественно, что при этом могут возникать проблемы, связанные с выполнением кода очистки;
   - 3.2. если применяется функция `TerminateThread`, то код, связанный с извещением `DLL_THREAD_DETACH`, не выполняется. Опять-таки, из-за этого могут возникнуть проблемы с очисткой. Следует также отметить, что функция `TerminateThread` является асинхронной. Она может выполнить возврат еще до того, как поток будет отмечен как сигнальный (т.е. до того, как он будет завершен). Чтобы приостановить выполнение вызывающего потока до тех пор, пока останавливаемый поток не будет фактически отмечен как сигнальный, можно применить функцию ожидания;
   - 3.3. память, в которой хранится стек потока, не освобождается до завершения процесса. Функция `TerminateThread` была реализована таким образом, чтобы другие потоки, которые могут все еще оставаться в рабочем состоянии и обращаться к переменным, имеющимся в стеке завершенного потока, могли продолжать работать беспрепятственно.
4. Завершить процесс, в котором содержится рассматриваемый поток. При этом приходится учитывать предостережения, которые касаются использования функции `TerminateThread`, поскольку фактически этот метод сводится к вызову функции `TerminateThread` для каждого потока в процессе.

При завершении работы потока выполняются действия, описанные ниже.

- Объект привилегированного режима потока отмечается как сигнальный.
- Система Windows освобождает все дескрипторы пользовательских объектов, принадлежащие потоку. Как было указано выше, обычно объекты, созданные потоками, принадлежат к процессу, к которому относятся эти потоки. Но из этого правила есть два исключения — окна и обработчики прерываний. После завершения работы потока освобождаются все открытые им дескрипторы окон и обработчиков прерываний.

- Система Windows изменяет значение кода завершения потока, задавая вместо `STILL_ACTIVE` возвращаемое значение функции точки входа или значение, переданное функции `ExitThread` или `TerminateThread`.
- Система Windows уменьшает значение счетчика использования объекта привилегированного режима потока на 1.
- Система Windows переводит процесс в разряд завершившихся, если данный поток является последним активным потоком в процессе.

ПРИМЕЧАНИЕ. По очевидным причинам многие из относящихся к потокам функции API-интерфейса Win32 недоступны из программы Visual C++, если она была связана с однопотоковой библиотекой этапа прогона. При попытке вызвать эти функции после связывания с однопотоковой библиотекой RTL возникают ошибки типа "unresolved external", относящиеся к таким функциям, как `CreateThread` и `_beginthreadex`. При создании приложения, отличного от MFC (Microsoft Foundation Classes — библиотека базовых классов Microsoft), по умолчанию применяется однопотоковая версия библиотеки RTL, а при создании приложения MFC или использовании программы-мастера расширенных хранимых процедур для построения расширенной процедуры SQL Server по умолчанию применяется многопотоковая библиотека RTL. Разрабатывая код, предназначенный для эксплуатации в многопотоковой среде, следует предусмотреть его связывание с многопотоковой версией библиотеки RTL, поскольку в однопотоковой библиотеке RTL не только отсутствуют функции, относящиеся к многопотоковой поддержке, но и не являются потокобезопасными некоторые базовые функции RTL. К примерам таких функций относятся `errno`, `_strerror`, `tmpnam`, `strtok` и многие, многие другие. А в тех случаях, когда неизвестно, к какому типу относится данная библиотека, необходимо учитывать, что для языка C одно- и многопотоковые библиотеки RTL поставляются отдельно, поскольку самые первые версии библиотек RTL создавались примерно в 1970 году, еще до того, как в операционных системах появилась поддержка потоков.

Функции `_beginthreadex` и `_endthreadex`

Даже несмотря на то, что для создания и завершения работы потоков предусмотрены стандартные функции `CreateThread` и `ExitThread` API-интерфейса Win32, разработчики программ Visual C++ должны вместо них предусмотреть использование функций `_beginthreadex` и `_endthreadex`. Ниже описаны три основные причины для такого решения.

1. Функция `_beginthreadex` распределяет структуру (известную как блок `tiddata`), которая позволяет некоторым функциям библиотек RTL для языков C/C++ (в частности, тем, которые упоминались в приведенном выше примечании) работать правильно при одновременном доступе к этим функциям со стороны многочисленных потоков.
2. В функции `_beginthreadex` используется функция точки входа, которая заключает применяемую пользователем функцию точки входа потока в рамку SEH. Эта рамка позволяет учитывать многие условия и ошибки,

которые не были бы перехвачены в случае создания потока непосредственно с помощью функции `CreateThread`.

3. Если для завершения работы потока вместо функции `ExitThread` используется функция `_endthreadex`, освобождается блок `tiddata` потока (а если применяется функция `ExitThread`, то память, занятая этим блоком, остается недоступной до завершения процесса).

А поскольку вызов функции `CreateThread` является единственным способом создания нового потока в системе Windows, то функция `_beginthreadex` в конечном итоге все равно ее вызывает. Тем не менее функция `_beginthreadex` вносит в ходе этого несколько изменений, благодаря которым код пользователя становится более надежным и защищенным от проблем потокобезопасности, которые могут возникать при использовании конкретной библиотеки RTL. Данная функция достигает указанной цели, подставляя свою собственную функцию точки входа потока вместо той, что предусмотрена пользователем, и сохраняя адрес пользовательской функции наряду с параметром, первоначально предоставленным пользователем, в структуре `tiddata`, распределенной из области динамической памяти RTL для пользовательского потока. Затем функция `_beginthreadex` вызывает функцию `CreateThread` и передает ей функцию точки входа потока (`_threadstartex`) в качестве функции точки входа и адрес блока `tiddata` в качестве параметра, определяемого пользователем. Сама функция `_threadstartex` устанавливает упомянутую выше рамку SEH, выполняет некоторую другую работу по инициализации, затем извлекает адрес пользовательской функции точки входа из переданного ей блока `tiddata` и вызывает функцию, передавая ей определяемый пользователем параметр, который был первоначально передан пользователем в функцию `_beginthreadex`. В конечном итоге механизм создания потока становится гораздо более безопасным и надежным. В программе SQL Server для создания экземпляров новых потоков используются функции `_beginthreadex` и `_beginthread`.

Списки параметров функций `CreateThread`/`_beginthreadex` и `ExitThread`/`_endthreadex` имеют небольшие различия, поэтому может потребоваться выполнить определенные операции приведения типов, чтобы успешно осуществить этап обработки программы компилятором, но эти функции весьма подобны, поэтому указанная задача не должна быть слишком сложной.

Функции потока

Функция потока представляет собой точку входа для потока. Именно с нее начинается выполнение при запуске потока. Ниже перечислены некоторые замечания, касающиеся функций этого типа.

- Функции потока могут иметь любые имена.
- Не должны возникать проблемы с использованием кодировок ANSI/Unicode, с которыми приходилось сталкиваться, имея дело с основным потоком и его точками входа `main`/`wmain` и `WinMain`/`wWinMain`.
- Функция потока должна возвращать значение.

- Необходимо максимально использовать параметры функции и локальные переменные; в результате применения статических и глобальных переменных код неизбежно теряет свойство потокобезопасности.

Системный код очистки следует разрабатывать таким образом, чтобы в нем осуществлялся возврат из функции точки входа, а не вызов функции `ExitThread/_endthreadex`. Возврат, выполняемый должным образом из функции потока, гарантирует достижение перечисленных ниже преимуществ.

- Вызываются деструкторы объектов C++, что позволяет правильно удалять эти объекты.
- Система Windows немедленно освобождает память, используемую стеком потока (а не ожидает завершения процесса).
- В качестве кода завершения потока (хранящегося в объекте привилегированного режима потока) устанавливается возвращаемое значение его функции.
- Уменьшается значение счетчика использования объекта привилегированного режима потока.

Потоки и обработка исключительных ситуаций

В системе Windows предусмотрена встроенная поддержка для операций структурированной обработки исключительных ситуаций (Structured Exception Handling – SEH). Это позволяет разработчику создавать код, в котором выполняемая задача отделена от тех действий, которые должны осуществляться в случае возникновения какой-либо ошибки.

Отличием операций SEH от языковых исключительных ситуаций (реализуемых с помощью таких ключевых слов, как `throw` и `catch`) является то, что средства, обеспечивающие выполнение этих операций, предоставляются операционной системой, а не библиотекой RTL. Безусловно, вы можете использовать в своем коде исключительные ситуации обоих типов; в действительности языковые исключительные ситуации в Visual C++ незаметно для пользователя реализованы с помощью средств SEH системы Windows.

При возникновении в операционной системе исключительной ситуации, которая не обрабатывается потоком, активизируется применяемый по умолчанию системный обработчик (если свой собственный обработчик в приложении не установлен), отображается диалоговое окно и процесс завершается. (Как было упомянуто выше, действия, фактически происходящие при возникновении необработанной исключительной ситуации, можно модифицировать с помощью программы `SetErrorMode`.)

Безусловно, такой порядок обработки исключительных ситуаций означает, что к разрабатываемому коду, который должен выполняться в рамках процесса SQL Server, предъявляются важные требования. Что произойдет, если пользователь вызовет какую-либо расширенную процедуру из сценария T-SQL, а эта расширенная процедура вызовет активизацию какой-то исключительной ситуации? Ответ на этот вопрос зависит от того, была ли эта исключительная ситуация активизирована

в контексте рабочего потока SQL Server или в контексте потока, созданного расширенной процедурой.

Если исключительная ситуация была активизирована рабочим потоком SQL Server, эксплуатирующим расширенную процедуру, то предусмотренный по умолчанию обработчик этого потока перехватит исключительную ситуацию и закроет соединение, ставшее причиной ее возникновения. Если же исключительная ситуация была вызвана потоком, созданным расширенной процедурой (и этот поток не имеет собственного кода обработки исключительных ситуаций), то программа SQL Server создает диагностический файл дампа, затем записывает в журнал ошибок сообщение, указывающее, что она заканчивает свою работу и выполняет выход из системы. Это означает, что любая необработанная исключительная ситуация в любом потоке, созданном любой расширенной процедурой, может вызвать аварийное прекращение работы сервера, а такой вариант чреват крупными неприятностями.

Как защититься от этой опасности? Следует всегда заключать функцию точки входа потока для любого потока, создаваемого в расширенной процедуре (или во внутрипроцессном COM-объекте), в код SEH. Пример такой структуры программы приведен в листинге 3.2.

Листинг 3.2. Функция потока, в которой предусмотрен код SEH

```
DWORD WINAPI StartThrd(LPVOID lpParameter)
{
  __try
  {
  CHAR *pCh=NULL;
  // Ссылка на NULL-указатель; активизирует исключительную ситуацию
  *pCh='x';
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    MessageBeep(0);
  }
  return 1;
}
```

В данном примере намеренно предпринята попытка разадресации указателя со значением NULL, чтобы вызвать исключительную ситуацию. А когда возникает исключительная ситуация, просто выполняется функция MessageBeep API-интерфейса Win32. Безусловно, в реальном приложении в блоке обработки исключительных ситуаций выполняются какие-то более существенные действия, но смысл остается тем же.

Функция GetLastError

Результат последнего вызова функции API-интерфейса Win32, который принято называть *последней ошибкой*, сохраняется в памяти каждым потоком. Хотя нет никакой гарантии, что некоторая функция Win32 не установит в качестве значения последней ошибки какое-то ошибочное условие завершения, ситуации, в которых

это происходит, возникают очень редко. Значение последней ошибки можно определить с использованием функции `GetLastError` API-интерфейса. Некоторые функции не только определяют значение последней ошибки при возникновении некоторого условия ошибки, но и устанавливают это значение, даже если не обнаруживаются какие-либо ошибки, иными словами, некоторые функции всегда переустанавливают это значение. Пользователь может установить значение последней ошибки с помощью функции `SetLastError` API-интерфейса.

Функцию `GetLastError` следует вызывать сразу после функций API-интерфейса, из которых желательно получить информацию об ошибках, поскольку, как уже было указано выше, некоторые функции всегда переустанавливают это значение, даже завершая свою работу без ошибок. После того как значение последней ошибки было потеряно в связи с тем, что его переустановила некоторая функция, не остается никакой возможности восстановить потерянное значение.

Для выборки важной информации из значений последней ошибки можно использовать операторы побитовой обработки. Например, бит 29 указывает, что данная ошибка представляет собой ошибку, определяемую пользователем. Данный бит не может быть установлен ни одной системной ошибкой. Определяя свои собственные коды ошибок, обязательно устанавливайте этот бит, чтобы избежать конфликта с кодом ошибки, определенным в системе.

Точный формат кодов ошибок Win32 описан в файле `winerror.h`, входящем в состав документации комплекта Platform SDK; кроме того, в этом файле находятся операторы `#define` для наиболее распространенных кодов ошибок Win32. К тому же, для ознакомления с кратким описанием наиболее распространенных кодов ошибок Win32 можно ввести команду `NET HELPMSG` *nnnn* в приглашении командной строки. Например, после ввода следующей команды в приглашении командной строки:

```
NET HELPMSG 5
```

будет получено такое описание ошибки:

```
Access is denied.
```

Это означает, что с помощью кода ошибки 5 подсистемы Win32 обозначено такое состояние ошибки, в котором запрещен доступ. Функция `FormatMessage` API-интерфейса Win32 позволяет преобразовать любой код ошибки в его текстовое описание. Эту функцию можно использовать для вывода сообщения при возникновении ошибки или для возврата строки с описанием ошибки в вызывающую функцию. Например, функцию `FormatMessage` можно использовать для преобразования кода ошибки Win32, встретившегося при выполнении расширенной процедуры, в строку, которую можно возвратить в программу SQL Server. Затем программа SQL Server передаст это сообщение клиенту.

Микропотоки

Микропотоки часто рассматриваются как упрощенные потоки. Они были введены в систему Windows с целью облегчить задачи переноса на эту платформу серверных приложений UNIX. В системе UNIX многопотоковая поддержка реализована иначе, чем в Windows. Если рассматривать приложения UNIX с точки зрения модели многопотоковой поддержки Windows, то они выглядят как

однопотоковые, но способны обрабатывать запросы многочисленных клиентов. Это означает, что разработчики UNIX создали свою собственную библиотеку многопотоковой поддержки, которую они используют для моделирования чистой многопотоковой поддержки того типа, который предлагается в системе Windows. Этот пакет UNIX позволяет выполнять во многом такие же операции, которые осуществляются в системе Windows при управлении потоками и контекстом потоков: сохраняются (восстанавливаются) некоторые регистры процессора, поддерживаются многочисленные стеки и осуществляется переключение между контекстами потоков в целях обслуживания запросов клиентов.

Основное различие между потоками и микропотоками в операционной системе Windows состоит в том, что потоки являются объектами привилегированного, а микропотоки — непривилегированного режима. Эта операционная система непосредственно рассчитана на поддержку потоков, и поэтому в ней предусмотрены весьма усовершенствованные и легко настраиваемые методы планирования, синхронизации потоков и управления ими в целях максимизации производительности системы и распараллеливания ее работы. В отличие от потоков, микропотоки всегда остаются невидимыми для ядра операционной системы. Они реализуются в коде непривилегированного режима, о работе которого ядро не получает никакой информации.

Настройка конфигурации программы SQL Server может быть выполнена таким образом, чтобы в ней использовались API-интерфейсы микропотоков Windows. Но, как правило, компания Microsoft не рекомендует применять эти интерфейсы, поэтому обычно к использованию микропотоков следует прибегать только после получения прямого указания от компании Microsoft или от одного из ее партнеров. В программе SQL Server имеются многочисленные средства, которые вообще не работают либо не могут работать правильно, если система находится в режиме микропотоков. Например, если система находится в режиме микропотоков, то в программе SQL Server не доступны средства SQLXML (в том числе процедура `sp_xml_preparedocument`). Аналогичное замечание относится к средствам SQLMAIL — их нельзя использовать, находясь в режиме микропотоков. Даже такие с виду безопасные действия, как инициализация COM-объекта из расширенной процедуры, которые в ходе работы в режиме потоков могли бы в принципе повлиять только на одно рабочее соединение SQL Server, при эксплуатации сервера в режиме микропотоков вполне могут нарушить работу сразу многих рабочих соединений, поскольку в одном потоке может быть создано множество собственных микропотоков. Итак, достаточно придерживаться следующей важной рекомендации — избегайте использования режима микропотоков, если это возможно.

Исходя из сказанного выше, необходимо продумать вопрос о том, может ли существовать сценарий, в котором было бы целесообразно предусмотреть использование режима микропотоков в программе SQL Server? Еще раз отметим, что такое решение следует принимать, получив на это указание от компании Microsoft, но существует один распространенный сценарий, в котором режим микропотоков может помочь выйти из положения в той ситуации, которая наблюдается, когда происходит слишком много переключений контекста между рабочими потоками SQL Server. Операции переключения контекста являются дорогостоящими, поэтому слишком

частое их выполнение снижает производительность системы почти до нуля. Кроме того, поскольку потоки являются объектами привилегированного режима, переход в состояние ожидания завершения связанных с ними действий (что чаще всего осуществляется в программе SQL Server) вызывает переключение потока из непривилегированного режима в привилегированный, а эта операция также является дорогостоящей (на ее выполнение затрачивается примерно 1 тысяча тактов процессора). Поскольку микропотоки представляют собой пользовательские объекты (реализованные в библиотеке `Kernel32.DLL`), режим микропотоков можно использовать для повышения производительности программы SQL Server в тех крайне неблагоприятных ситуациях, когда операции переключения контекста жестко конкурируют за ресурсы процессора с операциями, которые связаны с выполнением сервером базы данных реальной работы. В подобных сценариях может оказаться, что многочисленные недостатки режима микропотоков в данной конкретной ситуации компенсируются его преимуществами.

Упражнения

В этих упражнениях исследуется способность программы SQL Server обрабатывать исключительные ситуации, а также проводятся некоторые общие эксперименты, позволяющие узнать, как в этой программе организовано управление потоками. В упражнении 3.4 мы начнем с создания расширенной процедуры, все назначение которой состоит в активизации исключительных ситуаций различных типов в процессе SQL Server для определения того, как обрабатывает эти ситуации программа SQL Server. Изучение данного упражнения позволит читателю лучше понять методы, которые обычно применяются для устранения критических ошибок, предусмотренные в процессах Win32 в целом (и в процессе SQL Server в частности).

В упражнении 3.5 предусматривается подключение к программе SQL Server с помощью отладчика, что позволяет получить определенное представление о том, что происходит в этой программе, и узнать, как можно ознакомиться с информацией о текущем процессе, относящейся к потокам. Для подключения к серверу SQL Server, ознакомления с содержимым некоторых стеков потока и выполнения других основных задач, касающихся поддержки многопотоковой обработки, будет использоваться стандартный отладчик WinDbg компании Microsoft.

Упражнение 3.4. Исключительные ситуации в расширенных процедурах

В этом упражнении показано, что может произойти, если расширенная процедура вызывает некоторую исключительную ситуацию в процессе SQL Server. Выполнение данного упражнения приведет к останову программы SQL Server, поэтому его следует проводить только на сервере, предназначенном для экспериментов или разработки. В идеальном случае вы должны быть единственным пользователем этого сервера. Выполнение этого упражнения начинается с установки расширенной процедуры с именем `xp_exception` в ведущей базе данных путем выполнения описанных ниже шагов.

1. Найдите файл `xp_exception.dll` в подкаталоге `CH03` на компакт-диске, прилагаемом к данной книге.

2. Скопируйте его в каталог `binn` главного каталога программы SQL Server.
3. Запустите программу Query Analyzer и введите расширенную процедуру в главную базу данных с использованием команды `sp_addextendedproc`, как показано ниже.

   ```
   exec sp_addextendedproc 'xp_exception','xp_exception.dll'
   ```

   Для любознательных читателей в листинге 3.3 приведена главная процедура в расширенной процедуре `xp_exception`.

Листинг 3.3. Главный модуль процедуры `xp_exception`

```
RETCODE __declspec(dllexport) xp_exception(SRV_PROC *srvproc)
{

int iParams=srv_rpcparams(srvproc);
BYTE bType;
ULONG cbMaxLen;
ULONG cbActualLen;
BYTE bCrashType;
BOOL bNull;
DWORD dwThreadID;

if (0==iParams) bCrashType=0;
else srv_paraminfo(srvproc,1,&bType,&cbMaxLen,&cbActualLen,
    &bCrashType,&bNull);
switch (bCrashType)
{
case 0: {  // Аварийное завершение работы рабочего потока
     srv_sendmsg(srvproc,SRV_MSG_INFO,0,(DBTINYINT)0,(DBTINYINT)0,
         NULL,0,0,
    "Generating an access violation on the worker thread",
         SRV_NULLTERM);

    CHAR *pCh=NULL;
    // Ссылка на NULL-указатель
    *pCh='x';
     break;
     }
case 1: {  // Аварийное завершение работы нового потока в условиях
           // применения средств обработки исключительных ситуаций
     srv_sendmsg(srvproc,SRV_MSG_INFO,0,(DBTINYINT)0,(DBTINYINT)0,
         NULL,0,0,
    "Generating an access violation on a new thread with exception
       handling",SRV_NULLTERM);

     CreateThread(NULL,
     0,
    (LPTHREAD_START_ROUTINE)StartThrdHandled,
     NULL,
     0,
     (LPDWORD)&dwThreadID);
     break;
     }
case 2: {  // Активизация языковой исключительной ситуации
     srv_sendmsg(srvproc,SRV_MSG_INFO,0,(DBTINYINT)0,(DBTINYINT)0,
         NULL,0,0,
```

```
    "Generating a language exception on a new thread without
       exception handling",SRV_NULLTERM);

     CreateThread(NULL,
     0,
    (LPTHREAD_START_ROUTINE)StartThrdLanguageException,
     NULL,
     0,
     (LPDWORD)&dwThreadID);
     break;
     }
case 3: {  // Аварийное завершение работы нового потока в условиях
           // ОТСУТСТВИЯ средств обработки исключительных ситуаций
     srv_sendmsg(srvproc,SRV_MSG_INFO,0,(DBTINYINT)0,(DBTINYINT)0,
         NULL,0,0,
"Generating an access violation on a new thread WITHOUT
       exception handling -- your server should crash",
        SRV_NULLTERM);

     CreateThread(NULL,
     0,
    (LPTHREAD_START_ROUTINE)StartThrdUnhandled,
    NULL,
     0,
    (LPDWORD)&dwThreadID);
    break;
    }
}
  return XP_NOERROR ;
}
```

Вполне очевидно, что эта расширенная процедура принимает единственный параметр (целое число), который указывает, как должен произойти аварийный отказ в зависимости от переданного параметра. В табл. 3.8 перечислены поддерживаемые значения параметров и указано, с какими ситуациями связано применение каждого из этих параметров.

Таблица 3.8. Значения параметра `xp_exception` и ситуации, возникающие при их применении

Значение	Действие
0	Активизирует исключительную ситуацию в вызывающем рабочем потоке SQL Server
1	Активизирует исключительную ситуацию в новом потоке, который включает оболочку SEH
2	Активизирует языковую исключительную ситуацию
3	Активизирует исключительную ситуацию в новом потоке, который не включает оболочку SEH

4. Вызовем эту расширенную процедуру на выполнение. Вначале передадим этой процедуре значение параметра 0:

```
exec xp_exception 0
```

В программе Query Analyzer должны появиться примерно такие результаты:

```
ODBC: Msg 0, Level 20, State 1
Stored function 'xp_exception' in the library
'xp_exception.dll'
generated an access violation. SQL Server is terminating
process 51.
Generating an access violation on the worker thread

Connection Broken
```

Блок SEH, устанавливаемый программой SQL Server для ее рабочих потоков, перехватил исключительную ситуацию, созданную расширенной процедурой, и уничтожил соответствующее соединение.

5. Рассмотрим, что произойдет после возникновения исключительной ситуации в новом потоке, который имеет оболочку SEH вокруг его функции точки входа:

```
exec xp_exception 1
```

Все, что должно быть обнаружено в программе Query Analyzer, сводится к сообщению, возвращаемому расширенной процедурой; программа SQL Server не испытывает отрицательного воздействия этой исключительной ситуации благодаря применяемому коду SEH:

```
Generating an access violation on a new thread with
exception handling
```

6. Рассмотрим, что произойдет, если будет принудительно сформирована языковая исключительная ситуация, возникающая в новом потоке без оболочки SEH:

```
exec xp_exception 2
```

Все, что будет обнаружено в программе Query Analyzer, сведется к сообщению, возвращаемому самой расширенной процедурой, поскольку работа программы SQL Server не испытывает отрицательного воздействия этой языковой исключительной ситуации:

```
Generating a language exception on a new thread without
exception handling
```

Как было сказано выше в данной главе, исключительные ситуации языковые и Win32 — это два совершенно разных события. Языковая исключительная ситуация не может привести к останову сервера, а исключительная ситуация Win32, не будучи обработанной должным образом, может вызвать такую ситуацию.

7. Попытаемся нарушить доступ в новом потоке, не имеющем оболочки SEH вокруг его функции точки входа. (*Предостережение*. Это приведет к останову сервера.)

```
exec xp_exception 3
```

Ниже показано то, что будет выведено в программе Query Analyzer.

```
Generating an access violation on a new thread WITHOUT
exception handling -- your server should crash
```

Проведенная после этого проверка состояния сервера с помощью программы SQL Server Service Manager должна показать, что сервер был остановлен. Необработанная исключительная ситуация, активизированная процедурой `xp_exception`, вызывает аварийный останов процесса сервера. Такой ход событий должен послужить предостережением и показать, насколько важно предусмотреть обработку любых исключительных ситуаций, которые могут возникать

в пользовательском коде, особенно в многопотоковых расширенных процедурах. Не учитывая этого требования, разработчик может быстро вызвать аварийный останов сервера.

Следует отметить, что автор не рекомендует обязательно обрабатывать все исключительные ситуации в главном потоке расширенной процедуры (в рабочем потоке SQL Server, из которого был вызван этот главный поток). Если разработчик желает лишь обеспечить закрытие вызывающего соединения в случае катастрофического отказа в создаваемой им расширенной процедуре, достаточно разрешить, чтобы об этом позаботились применяемые по умолчанию средства обработки SEH программы SQL Server.

Но, несмотря на это, разработчик должен обязательно обеспечить обработку всех исключительных ситуаций в любых новых потоках, создаваемых в расширенной процедуре. Несоблюдение этого требования приведет к останову сервера в любой исключительной ситуации.

После проверки каталога `LOG`, находящегося под главным каталогом инсталляции SQL Server, должны быть обнаружены два новых файла: файл журнала исключительных ситуаций и файл дампа стека. Оба они являются текстовыми файлами, которые можно просматривать с помощью программы Notepad. Обязательно откройте каждый из них и просмотрите, чтобы ознакомиться с дополнительными данными об активизированных исключительных ситуациях. Особенно интересные сведения о библиотеках DLL, загруженных в пространство адресов процесса SQL Server, и о содержимом стеков вызова рабочих потоков ко времени активизации исключительных ситуаций можно найти в файле дампа.

Упражнение 3.5. Ознакомление с информацией о потоке с помощью отладчика

В этом очередном упражнении показано, как подключиться к программе SQL Server с помощью отладчика WinDbg и ознакомиться с некоторой информацией, касающейся потоков, отображаемой этим отладчиком. Для того чтобы изучить, как работает программа SQL Server, выполните следующее.

1. Если сервер все еще остановлен после выполнения предыдущего упражнения, перезапустите его. Как и перед этим, это должен быть сервер, применяемый для экспериментов или разработки, и в идеальном случае вы должны быть единственным пользователем сервера (поскольку временный останов программы SQL Server происходит даже при подключении к ней с помощью отладчика).
2. Запустите отладчик WinDbg. Как было указано в главе 2, обязательно проверьте, чтобы путь к файлам отладочных символов был задан правильно.
3. Нажмите клавишу <F6>, чтобы подключиться к программе SQL Server. Найдите экземпляр `sqlservr.exe` в списке процессов и выберите его. Если на компьютере работает больше одного экземпляра этой программы, то в списке процессов может находиться несколько строк `sqlservr.exe`. В таком случае разверните узел дерева, относящийся к каждому процессу `sqlservr.exe`, чтобы ознакомиться с его командной строкой, поскольку узнать, каковым является каждый экземпляр, можно с помощью его командной строки. А если программа SQL Server была только что перезапущена, то нужный экземпляр должен находиться в нижней части списка процессов.
4. После подключения к программе должно автоматически открыться окно Disassembly. Закройте его. А если окно Disassembly снова откроется на любом этапе

выполнения данного упражнения, еще раз его закройте; в настоящее время оно не понадобится.

5. Найдите командное окно и сформируйте дамп стеков потока с помощью следующей команды:

   ```
   ~*kv
   ```

   На экране должно появиться содержимое стеков вызовов каждого потока в процессе SQL Server. Вы обнаружите, что в большинстве рассматриваемых потоков выполняется функция `WaitForSingleObject` API-интерфейса Win32. Это связано с тем, что потоки ожидают, пока не станет сигнальным синхронизационный объект определенного типа; дополнительная информация о функции `WaitForSingleObject` и синхронизационных объектах приведена ниже в данной главе.

6. Теперь выполним следующую команду:

   ```
   !teb
   ```

   Эта команда выведет на экран содержимое блока TEB для текущего потока. Текущий поток обозначается приглашением в левой части поля редактирования в командном окне. (Если не был выполнен переход на другой блок, текущий блок должен относиться к последнему рабочему потоку в пространстве процесса SQL Server.) Содержимое блока TEB должно выглядеть примерно так, как показано в листинге 3.4.

Листинг 3.4. Содержимое блока TEB

```
TEB at 7FF99000
    ExceptionList:      26caffdc
    Stack Base:         26cb0000
    Stack Limit:        26cae000
    SubSystemTib:       0
    FiberData:          1e00
    ArbitraryUser:      0
    Self:               7ff99000
    EnvironmentPtr:     0
    ClientId:           bcc.cf8
    Real ClientId:      bcc.cf8
    RpcHandle:          0
    Tls Storage:        f3b38
    PEB Address:        7ffdf000
    LastErrorValue:     0
    LastStatusValue:    0
    Count Owned Locks:  0
    HardErrorsMode:     0
```

Обратите внимание на то, что в этом выводе присутствует значение `LastErrorValue`. Оно представляет собой значение, которое было бы обнаружено в этот момент после вызова в потоке функции `GetLastError` API-интерфейса Win32.

7. Выборку значения последней ошибки для текущего потока можно также осуществить с помощью следующей команды:

   ```
   !gle
   ```

Эта команда выводит значение последней ошибки, а также последнее значение состояния для потока:

```
LastErrorValue: (Win32) 0 (0) - The operation completed
successfully.
LastStatusValue: (NTSTATUS) 0 - STATUS_WAIT_0
```

8. В завершение этого упражнения выполним вывод на экран содержимого блока среды процесса (Process Environment Block — PEB) программы SQL Server:

   ```
   !peb
   ```

   Эта команда выводит большой объем информации, включая список всех модулей, загруженных в настоящее время в пространство процесса SQL Server, командную строку, заданную для этого процесса, путь к библиотекам DLL программы SQL Server и много других полезных сведений.

9. Теперь можно выйти из программы отладчика. Если вы работаете в версии Windows, предшествующей Windows XP, то вам придется перезапустить программу SQL Server, поскольку после выхода из отладчика она остается остановленной.

Потоки. Резюме

Поток представляет собой механизм, с помощью которого в операционной системе Windows выполняется код. Ни в одном приложении не может быть выполнено ни одной команды, если для этого не применяется поток. Потоки функционируют в контексте владеющего ими процесса и завершаются после него. Между микропотоками и потоками существует связь, аналогичная связи между потоками и процессами, но обычно лучше всего использовать потоки и по мере возможности избегать применения режима микропотоков не только в разрабатываемых, но и в эксплуатируемых приложениях, таких как SQL Server. Операционная система Windows с самого начала была спроектирована с учетом использования в ней потоков и оптимизирована для работы с ними. В очень многих ситуациях функциональные возможности микропотоков Win32 являются гораздо более ограниченными по сравнению с теми, что поддерживаются потоками, основанными на использовании ядра.

Потоки. Вопросы для самопроверки

1. Что произойдет, если в приложении запускается новый рабочий поток, который завершает свою работу из-за возникшей исключительной ситуации, а функция точки входа потока не имеет оболочки SEH?
2. Каково имя единственной зависящей от процессора структуры во всем API-интерфейсе Win32 и почему она зависит от процессора?
3. В какой части пространства адресов процесса (в пространстве привилегированного режима или в пространстве непривилегированного режима) хранится блок TEB?
4. Подтвердите или опровергните следующее утверждение. Утилита TList может использоваться для определения процентной доли нагрузки процессора для заданного процесса.

5. Подтвердите или опровергните следующее утверждение. После создания нового процесса система автоматически создает его первый поток независимо от того, какие параметры переданы в функцию `CreateProcess`.
6. Что происходит при попытке откомпилировать и отредактировать связи программы Visual C++, которая вызывает функцию `CreateThread` API-интерфейса Win32, если связывание происходит с однопотоковой версией библиотеки этапа прогона?
7. В чем состоит предпочтительный метод завершения работы потока?
8. Подтвердите или опровергните следующее утверждение. При проектировании большинства сложных приложений разработчик должен прежде всего разделить объем работы, выполняемой приложением, на задачи, распределенные по нескольким потокам, и спроектировать потоки для выполнения этих задач.
9. Назовите основную причину, по которой создание процесса происходит медленнее и требует больше ресурсов по сравнению с созданием потока.
10. Какая команда отладчика WinDbg используется для вывода на экран содержимого блока PEB рассматриваемого потока?
11. Опишите назначение функции `ReadProcessMemory` API-интерфейса Win32.
12. Какая функция API-интерфейса Win32 используется для создания потока?
13. Какую команду WinDbg можно использовать для вывода на экран только значения последней ошибки и последнего по времени значения состояния текущего потока?
14. Подтвердите или опровергните следующее утверждение. Идентификаторы процесса и потока могут совпасть, поскольку они формируются в разных пространствах имен.
15. Какая функция API-интерфейса Win32 используется при распределении памяти для индекса TLS?
16. Что является источником ошибки с кодом `0x80000004` — операционная система Windows или пользовательское приложение?
17. Получено значение последней ошибки, равное 6. Какая последовательность командных строк может использоваться для вывода на экран текстового описания этого кода ошибки?
18. Что показывает счетчик `Thread:% Privileged Time` программы Perfmon?
19. Какая команда отладчика WinDbg используется для вывода на экран только одного блока TEB для потока?
20. Подтвердите или опровергните следующее утверждение. Поле `LastErrorValue` в выводе содержимого блока TEB, полученном с помощью отладчика WinDbg, содержит ту же информацию, которую возвращает функция `GetLastError` API-интерфейса Win32.

Планирование потоков

В вытесняющей, многозадачной операционной системе используется определенный подход для принятия решения о том, когда и какие потоки должны выполняться. Алгоритмы, используемые для этого в системе Windows, не всегда полностью документированы, но компания Microsoft спроектировала их таким образом, чтобы они обеспечивали в максимальной степени равноправные условия и были в равной степени применимыми для всех приложений. Ниже описано, как действуют некоторые из этих алгоритмов, но следует учитывать, что в последующих версиях Windows они могут существенно измениться.

Перед переходом к дальнейшему изложению автор должен сделать паузу и указать, что в программе SQL Server, вопреки ожиданиям, не используются планировщик Windows и API-интерфейсы планирования. Это связано с тем, что программа SQL Server обеспечивает основную часть своих потребностей в планировании потоков самостоятельно с помощью своего компонента UMS. В результате операционная система обычно обнаруживает, что в любой момент времени на каждом процессоре функционирует только один поток SQL Server. Поэтому даже несмотря на то, что в любое время в сервере могут применяться сотни рабочих потоков, операционная система Windows фактически имеет информацию лишь о том, что определенную работу выполняет один из этих потоков в расчете на каждый процессор.

Причины использования такого подхода будут подробно описаны в главе 10, но на данный момент достаточно понять, что в программе SQL Server планировщик Windows и API-интерфейсы планирования используются иначе, чем может показаться на первый взгляд. Ниже показана вся глубина различий между ожидаемым и действительным в данном случае, а также описаны причины, лежащие в основе этого подхода.

Планирование потоков. Основные термины и определения

- **Переключение контекста.** Действия, которые происходят, когда операционная система Windows записывает в память информацию контекста одного потока и загружает из нее информацию другого, чтобы в дальнейшем функционировал другой поток. Эти действия состоят из сохранения и загрузки значений временных регистров и других элементов данных, относящихся к среде этапа прогона рассматриваемого потока.
- **Квант времени.** Интервал времени, предоставляемый потоку для его выполнения планировщиком Windows.
- **Вытеснение.** Действия, которые происходят, когда операционная система Windows останавливает один поток до истечения назначенного ему кванта времени, чтобы можно было приступить к выполнению другого потока.
- **Тактовый интервал.** Частота, с которой активизируются прерывания часов процессора.

- **Состояние потока.** Текущее состояние выполнения потока, представленное целочисленным значением от 0 до 7.
- **Приоритет процесса.** Приоритет выполнения процесса, который может принимать значение от `Idle` (Простаивающий) до `Real-Time` (Работающий в реальном времени).
- **Приоритет потока.** Приоритет выполнения определенного потока, устанавливаемый с учетом соответствующего ему процесса, который может принимать значение от `Idle` до `Time-Critical` (Зависящий от времени).
- **Родственность процессора.** Соотношение, показывающее, на каких процессорах может выполняться тот или иной процесс или поток.
- **Идеальный процессор.** Процессор, который считается наиболее приемлемым для конкретного потока. Операционная система Windows при планировании потока на выполнение отдает предпочтение идеальному процессору, но допускает возможность выполнения потока на другом процессоре, если идеальный занят.
- **Исчерпание ресурсов потока.** Ситуация, которая возникает, если поток с низким приоритетом постоянно вытесняется потоками с более высокими приоритетами и не получает возможности продолжить выполнение.

Краткий обзор

В операционной системе Windows планирование работы осуществляется с использованием системы планирования, управляемой приоритетами и основанной на принципе вытеснения. Это означает, что поток с наивысшим приоритетом, который готов для выполнения (работоспособный), вытесняет потоки с более низкими приоритетами (с учетом родственности процессора; дополнительная информация по этой теме приведена ниже). При вытеснении потока с более низким приоритетом его работа прерывается (возможно, лишь на мгновение), а поток с более высоким приоритетом получает возможность выполнять свою работу. Если поток постоянно вытесняется и не получает возможности функционировать в течение продолжительного периода времени, возникает ситуация исчерпания ресурсов этого потока.

Отметим, что в операционной системе Windows работа планируется с учетом распределения задач по потокам. Поскольку сами процессы не функционируют, а лишь предоставляют среду, в которой могут функционировать потоки, такой подход является вполне оправданным.

Операционная система Windows создает иллюзию, будто все потоки выполняются одновременно, разделяя время, в течение которого разрешено выполнение каждого потока, на временные интервалы, называемые *квантами времени*. В этой операционной системе кванты времени передаются потокам по принципу циклической очереди.

В операционной системе Windows не существует какой-то единой процедуры, которая выполняла бы все задачи планирования. Напротив, в ядре Windows имеется множество процедур и модулей, занятых планированием работы. Вся эта совокупность программного кода известна под общим названием *диспетчера*, или *планировщика ядра*.

Планирование потоков. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к планированию потоков, приведены в табл. 3.9.

Таблица 3.9. Основные функции API-интерфейсов, относящиеся к планированию потоков

Функция	Описание
`Suspend/ResumeThread`	Приостановить (возобновить) выполнение потока
`Sleep/SleepEx`	Приостановить выполнение текущего потока на указанное время
`Get/SetPriorityClass`	Получить (задать) базовый приоритет для указанного процесса
`Get/SetThreadPriority`	Получить (задать) для указанного потока приоритет по отношению к процессу
`Get/SetProcessAffinityMask`	Получить (задать) перечень процессоров, на которых разрешено выполняться данному процессу
`SetThreadAffinityMask`	Задать перечень процессоров, на которых разрешено выполняться данному потоку
`Get/SetThreadPriorityBoost`	Получить (задать) параметр, определяющий свойство системы увеличивать на время приоритет потока
`SetThreadIdealProcessor`	Задать для потока номер идеального процессора, на котором он должен работать
`Get/SetProcessPriorityBoost`	Получить (задать) параметр, определяющий свойство системы увеличивать на время приоритет процесса
`SwitchToThread`	Предоставить потоку право отказываться от оставшейся части выделенного ему кванта времени, чтобы дать возможность работать другим потокам

Планирование потоков. Основные инструментальные средства

Программа Perfmon остается основным диагностическим инструментальным средством и в области планирования потоков. Для текущего контроля над тем, что происходит в конкретных процессах, очень ценными являются также программы Pview и Pviewer. С учетом того, что основная часть кода планирования фактически находится в ядре, максимальный объем полученной информации ограничивается возможностями инструментальных средств непривилегированного режима (табл. 3.10).

Внутренняя организация планирования потоков

Планирование потоков осуществляется ядром Windows и остается прозрачным для приложений. В отличие от архитектуры с кооперативной многозадачностью, применявшейся в 16-разрядных версиях Windows, в каждой версии Windows,

начиная с Windows NT 3.1 (с первой версии семейства продуктов Windows NT, представленного в настоящее время операционными системами Windows 2000, Windows XP и Windows Server 2003), реализована система планирования, которая не требует от приложений, чтобы они осуществляли какие-либо специальные действия, обеспечивающие бесперебойную эксплуатацию других приложений или одновременное выполнение сразу нескольких задач. Ни один из потоков или процессов не должен ожидать полного завершения работы другого потока или процесса для того, чтобы работа операционной системы или его самого продолжалась без всяких препятствий.

Таблица 3.10. Основные инструментальные средства, связанные с планированием потоков, и предоставляемая ими информация

	Базовый приоритет процесса	Класс приоритета процесса	Базовый приоритет потока	Текущий приоритет потока
TaskMgr		+		
Perfmon	+		+	+
Pstat	+			+
Pviewer	+			+

Кванты времени

Как было указано выше, *квантом времени* называется интервал времени, в течение которого разрешается выполнение кода некоторого потока до того, как система прервет его работу. Значение кванта времени фактически не измеряется единицами времени; оно представляет собой целочисленное значение, выраженное в единицах, которые принято называть *единицами измерения квантов времени*. Каждый раз, когда какой-то поток планируется на выполнение, его работа начинается с определенного количества единиц измерения квантов времени, а после активизации каждого прерывания от таймера процессора из этой величины вычитается определенное количество указанных единиц измерения. После того как количество единиц измерения квантов времени достигает 0, работа потока прерывается, и право на выполнение своего кода передается другому потоку.

Безусловно, предпосылкой нормального функционирования такой системы является то, что ни один поток с более высоким приоритетом не ожидает освобождения процессора, на котором выполняется рассматриваемый поток. Если возникает необходимость передать право на выполнение потоку с более высоким приоритетом, а процессор, на котором в данный момент выполняется поток с более низким приоритетом, обладает свойством родственности по отношению к высокоприоритетному потоку, то система снимает с себя все обязательства — поток с более высоким приоритетом вытесняет поток с более низким приоритетом и приступает к выполнению своего кода независимо от того, закончился ли квант времени последнего потока.

Частота тактового интервала процессора зависит от платформы. Так, частота тактового интервала определяется на уровне абстракции аппаратных средств (Hardware Abstraction Layer — HAL) операционной системы Windows, а не на уровне

ядра. В большинстве однопроцессорных систем x86 тактовый интервал равен 10 миллисекундам (мс), а в большинстве многопроцессорных систем x86 составляет 15 мс.

Количество единиц измерения квантов времени, вычитаемое после каждого тактового интервала, равно 3. В операционных системах Windows 2000 Professional и Windows XP по умолчанию количество квантов времени, выделяемых для потока, равно 6. В операционных системах Windows 2000 Server и Windows Server 2003 это количество равно 36. С учетом того, что тактовый интервал на однопроцессорном компьютере равен 10 мс, можно сделать вывод, что квант времени потока может охватывать 2 прерывания от таймера (приблизительно 20 мс) в операционной системе Windows XP или 12 прерываний от таймера (120 мс) в операционной системе Windows Server 2003. А на многопроцессорном компьютере с тактовым интервалом, равным 15 мс, квант времени потока может продолжаться не больше 30 мс в операционной системе Windows XP и 180 мс в операционной системе Windows Server 2003.

Как уже было сказано, может оказаться, что какой-то поток не исчерпает до конца свой текущий квант времени. Приведенные выше цифры являются максимальными; если какой-то поток с более высоким приоритетом перейдет в такое состояние, что его можно будет запланировать для выполнения на конкретном процессоре, то высокоприоритетный поток вытеснит работающий поток с более низким приоритетом.

У читателя может возникнуть вопрос, почему квант времени выражается в виде кратного трем числа в расчете на один такт системного таймера, а не как более простое отношение 1:1. Причина состоит в том, что приходится предусматривать частичное расходование кванта времени при выходе некоторых потоков из состояния ожидания. Если в потоке, базовый приоритет которого меньше 14, выполняется функция ожидания (например, `WaitForSingleObject`), продолжительность его кванта времени уменьшается на 1. (А при выходе из состояния ожидания потоков с базовым приоритетом, равным 14 или более высоким, значение отведенного им кванта времени восстанавливается.) Таким образом, при выходе этого потока из состояния ожидания он получает максимум 5 оставшихся единиц измерения квантов времени, вместо того оставшегося количества, равного 6 единицам измерения квантов времени, которое мог бы иметь поток. Такой подход позволяет исключить проблему, которая могла бы возникнуть при использовании соотношения 1:1 в той ситуации, когда поток постоянно вызывается на выполнение и переходит в состояние ожидания между тактами системного таймера. Если бы в системе не было предусмотрено расходование кванта времени потока каждый раз, когда поток выходит из состояния ожидания, то оказалось бы, что квант времени потока длится "бесконечно" (разумеется, при том условии, что при любом прерывании от таймера поток находится в состоянии ожидания).

Состояния потока

После запуска программы Perfmon и вывода на экран пояснения, касающегося назначения счетчика Thread State для объекта `Thread`, вы обнаружите, что существует восемь потенциальных состояний потока (табл. 3.11).

Таблица 3.11. Состояния потока и их значения

Значение	Состояние
0	Инициализированный
1	Готовый (ожидающий предоставления ему процессора)
2	Работающий (в настоящее время использующий процессор)
3	Находящийся в "горячем" резерве (подготавливаемый для выполнения на процессоре)
4	Завершившийся (остановленный)
5	Находящийся в состоянии ожидания (ожидающий завершения операции на периферийном устройстве или освобождения ресурса)
6	Находящийся в переходном состоянии (ожидающий освобождения ресурса, чтобы приступить к выполнению)
7	Находящийся в неизвестном состоянии

У читателя, возможно, вызовет удивление тот факт, что большинство потоков проводят основную часть своего времени в состоянии ожидания. Почти в любом конкретном приложении потоки главным образом ожидают какого-то события (ввода с клавиатуры, щелчка кнопкой мыши, операции ввода-вывода и т.д.), которые должны произойти или завершиться. Безусловно, такое же утверждение относится и к программе SQL Server, в чем можно было убедиться, изучив приведенные выше результаты вывода на экран содержимого стеков потока SQL Server с помощью отладчика WinDbg.

Приоритеты потока

Операционная система Windows поддерживает 32 уровня приоритетов, начиная с 0 и заканчивая 31, из которых наивысшим является 31. Потоки сразу после своего создания наследуют базовый приоритет своих процессов. Такой приоритет может быть задан непосредственно во время создания процесса с помощью функции `CreateProcess`, а впоследствии изменен в самом потоке (с помощью функции `SetPriorityClass`) или вне его (с использованием программы Task Manager или аналогичного инструментального средства). В табл. 3.12 приведена итоговая информация о приоритетах процессов, поддерживаемых операционной системой Windows.

Таблица 3.12. Уровни приоритетов процесса

Приоритет	Описание
`Idle`	Потоки в этом процессе работают только в то время, когда система простаивает
`Below normal`	В Windows 2000 и последующих версиях потоки процесса с приоритетом `Below normal` работают с более низком приоритетом, чем `Normal`, но с более высоким, чем `Idle`
`Normal`	Этот процесс не предъявляет каких-либо особых требований к планированию

Окончание табл. 3.12

Приоритет	Описание
Above normal	В Windows 2000 и последующих версиях потоки процесса с приоритетом Above normal работают с более высоким приоритетом, чем Normal, но с более низким, чем High
High	Этот приоритет используется для задач, зависящих от времени. С этим приоритетом работает программа Task Manager, чтобы с ее помощью можно было уничтожать процессы, создающие большую нагрузку на процессор
Real-Time	Этот приоритет также используется для задач, зависящих от времени. Потоки в процессе, зависящем от времени, должны немедленно реагировать на события. Следует отметить, что задачи с базовым приоритетом Real-Time конкурируют за процессорное время с операционной системой и могут неблагоприятно повлиять на общую производительность системы. Это значение приоритета следует использовать лишь в случае крайней необходимости

Приоритет потока после его создания можно изменить с помощью функции SetThreadPriority. Для задания приоритета потока никогда не следует использовать его числовое значение, а можно указывать лишь одну из заранее определенных констант, предусмотренных в API-интерфейсе Win32 (например, THREAD_PRIORITY_NORMAL, THREAD_PRIORITY_ABOVE_NORMAL и т.д.). Точные числовые значения этих констант могут изменяться при переходе от одного выпуска Windows к другому (и такие изменения действительно уже происходили). Уровни приоритета потоков, поддерживаемые в настоящее время, приведены в табл. 3.13.

Таблица 3.13. Уровни приоритета потоков

Приоритет	Описание
Idle	Поток выполняется на уровне приоритета 1 в процессах с приоритетом Above normal, Below normal, High, Idle и Normal и на уровне приоритета 16 в процессах с приоритетом Real-Time
Lowest	Поток выполняется на уровне, который ниже на 2 пункта уровня приоритета Normal для базового класса приоритета
Below normal	Поток выполняется на уровне, который ниже на 1 пункт уровня приоритета Normal для базового класса приоритета
Normal	Поток выполняется на уровне приоритета Normal для данного класса приоритета
Above normal	Поток выполняется на уровне, который выше на 1 пункт уровня приоритета Normal для базового класса приоритета
Highest	Поток выполняется на уровне, который выше на 2 пункта уровня приоритета Normal для базового класса приоритета
Time-critical	Поток выполняется на базовом уровне приоритета 15 в процессах с приоритетом Above normal, Below normal, High, Idle и Normal и на базовом уровне приоритета 31 в процессах с приоритетом Real-Time

Как было указано выше, потоки с более высокими приоритетами вытесняют потоки с более низкими. Это означает, что при всех прочих равных условиях, как только становится работоспособным поток с более высоким приоритетом, потоки с более низкими приоритетами перестают получать время на используемом процессоре – они будут до бесконечности испытывать нехватку ресурсов, пока поток с более высоким приоритетом либо не завершится, либо не перейдет в состояние ожидания.

Не все приоритеты являются в равной степени доступными для приложений. Например, для приложений с непривилегированным режимом недоступны приоритеты 17–21 и 27–30 (но они могут применяться для драйверов устройств). Кроме того, с приоритетом 0 в системе может выполняться только один поток: поток обнуления страниц Windows. Назначение потока обнуления страниц состоит в том, что он заполняет нулями свободные страницы в оперативной памяти, пока нет других работоспособных потоков. Он работает только при том условии, что система полностью простаивает, не имея какой-то другой работы. Система всегда предпринимает попытки поддерживать процессор в загруженном состоянии и начинает простаивать, только если у нее нет для выполнения абсолютно никакой работы.

Планировщик, обнаружив поток, который использовал свой квант времени, снова планирует данный поток на выполнение в течение еще одного кванта времени, если другие потоки с тем же приоритетом отсутствуют. Именно поэтому потоки с более низкими приоритетами могут так легко оказаться в ситуации исчерпания ресурсов – тот факт, что поток только что завершил использование своего кванта времени, не имеет никакого значения при принятии решения о том, будет ли ему разрешено продолжить свое выполнение.

Это не означает, что положение является безвыходным; обнаружив, что некоторый поток находился в состоянии исчерпания ресурсов на протяжении около 3–4 секунд, планировщик увеличивает приоритет этого потока в расчете на то, что это позволит ему продолжить выполнение. Кроме того, планировщик удваивает продолжительность кванта времени рассматриваемого потока. После выполнения кода потока на данном этапе его приоритет переустанавливается на прежний уровень, и восстанавливается применяемая по умолчанию продолжительность его кванта времени.

СОВЕТ. Пользователь имеет возможность изменить базовый приоритет любого интерактивного (не относящегося к системным службам) приложения с помощью программы Task Manager. Достаточно просто щелкнуть правой кнопкой мыши на имени процесса в списке Processes и выбрать из контекстного меню команду Set Priority. Кроме того, можно запустить процесс, указав значение приоритета, отличное от применяемого по умолчанию, с использованием команды `start` операционной системы Windows и одной из опций командной строки с таким приоритетом, как `/abovenormal`, `/high` или `/realtime`.

Корректировка параметров процесса переднего плана

Чтобы вынудить операционную систему Windows повысить производительность приложений переднего плана, можно воспользоваться диалоговым окном Performance Options операционной системы Windows (в версии Windows 2000 для этого

можно выполнить переход по элементам рабочего стола My Computer | Properties | Advanced). В этом диалоговом окне представлены два варианта выбора приоритета приложения: Applications и Background services. При выборе варианта Background services повышение производительности не происходит. А если будет выбран вариант Applications, то операционная система Windows увеличивает продолжительность кванта времени любого процесса, который в настоящее время находится на переднем плане, независимо от того, какое приложение выходит на передний план. Такое действие в целом приводит к улучшению отклика того приложения, которое в данный момент находится в фокусе, но это происходит за счет таких фоновых служб, как SQL Server. В версиях Windows 2000 Professional и Windows XP по умолчанию применяется вариант Applications, а в версиях Windows 2000 Server и Windows Server 2003 по умолчанию предусмотрен вариант Background services.

Потоки реального времени

Потоком реального времени называется поток, приоритет которого равен 16 или выше. Хотя пользователь имеет возможность создать процесс с базовым приоритетом реального времени, необходимо учитывать, что потоки реального времени могут блокировать потоки системы, а это может стать причиной ошибочного поведения операционной системы Windows. В результате применения потоков реального времени будет задерживаться выполнение таких важных функций системы, как запись на диск и распределение памяти. Режим, в котором работает поток (непривилегированный или привилегированный режим), не имеет никакого отношения к вытеснению — поток, работающий в непривилегированном режиме, может вытеснить поток, работающий в привилегированном, и наоборот. Безусловно, это означает, что пока на данном конкретном процессоре продолжает выполняться запланированный для него поток с приоритетом 31, никакой другой поток (работающий в привилегированном или непривилегированном режиме) не может работать на этом процессоре.

Важное различие между потоками реального времени и потоками с другими классами приоритетов состоит в том, что значения продолжительности квантов времени потоков реального времени переустанавливаются при их вытеснении. Таким образом, обычный поток оставляет за собой все, что не было израсходовано от его кванта времени к моменту его вытеснения. После повторного планирования он выполняется в течение остатка этого кванта времени или до момента следующего вытеснения, а поток реального времени после его повторного планирования вслед за вытеснением получает полностью новый квант времени. Это дает потокам реального времени еще одно преимущество над обычными потоками с точки зрения планирования, и в целом означает, что, вообще говоря, не следует применять при планировании потоки с высокими приоритетами, если необходимо добиться равномерного распределения пропускной способности системы.

Обратите внимание на то, что выражение "поток реального времени" в мире Windows не означает, что этот поток действительно будет выполняться в *реальном времени*, в традиционном смысле этого термина. В операционной системе Windows не предусмотрены такие обычные средства операционной системы,

работающей в реальном времени, как гарантированная низкая продолжительность времени от момента возникновения прерывания до момента его обнаружения или возможность для потока получить гарантированное время выполнения. В Windows могут даже вытесняться потоки реального времени другими потоками реального времени с более высоким приоритетом.

Планирование очередей

При описании организации планирования очередей необходимо прежде всего отметить, что в действительности в системе применяется целый ряд очередей, по одной для каждого приоритета планирования. Каждый из 32 уровней приоритета потока имеет свою собственную очередь планирования. После того как операционная система Windows приступает к поиску потока для выполнения, этот поиск просто начинается с очереди потоков с наивысшим приоритетом (31) и продолжается в направлении к более низким уровням через другие очереди, пока не будет обнаружен готовый к выполнению поток.

Чтобы предотвратить необходимость физически проходить по всем 32 очередям потоков каждый раз, когда требуется найти готовый к выполнению поток, в операционной системе Windows поддерживается 32-битовая структура отображения, называемая *структурой с итоговыми данными* о готовых к выполнению потоках. Каждый бит в этой структуре отображения указывает, что один или несколько потоков на соответствующем уровне приоритета готовы к выполнению. Это позволяет операционной системе Windows быстро обнаруживать, в каких очередях потоков следует искать готовые потоки, не перебирая все эти очереди.

Кроме того, в операционной системе Windows поддерживается структура отображения, которая показывает, какие процессоры простаивают. Это позволяет быстро найти простаивающий процессор, когда возникает необходимость запланировать какой-то поток на выполнение.

Приблизительно через каждые 20 мс в операционной системе Windows проверяются все созданные объекты привилегированного режима потока. Обычно большинство таких объектов не являются работоспособными, поскольку ожидают, что должно произойти какое-то событие, операция ввода-вывода и т.д. Но некоторые из этих объектов могут быть запланированы на выполнение, поэтому Windows выбирает один из них, загружает в регистры процессора те значения, которые были в последний раз сохранены в структуре CONTEXT потока, и планирует поток на выполнение. Такая последовательность действий называется *переключением контекста*.

В потоке, запланированном на выполнение, реализуется код и осуществляются манипуляции с данными в пространстве адресов процесса до тех пор, пока этот поток не будет вытеснен или пока не истечет его квант времени. При переключении контекста на другой поток система снова сохраняет содержимое регистров процессора в структуре CONTEXT работающего в настоящее время потока, восстанавливает значения регистров для того потока, который должен быть запланирован, и планирует новый поток на выполнение. Все подобные действия по переключению контекста между потоками начинаются с запуска системы и продолжаются до завершения ее работы. В определенном смысле все эти действия можно рассматривать как главный цикл программы, называемой *операционной системой Windows*.

Переключение контекста

Как было указано выше, переключение контекста сводится к обмену значениями между регистрами процессора и временными регистрами, распределенными в структуре CONTEXT каждого потока. Переключение контекста вызывает сохранение (загрузку) описанных ниже фрагментов данных.

- Регистр состояния процесса.
- Указатели стека непривилегированного и привилегированного режимов.
- Содержимое других регистров.
- Счетчик команд программы.
- Указатель на пространство адресов, в котором выполняется рассматриваемый поток.

В действительности операционная система Windows следит за тем, сколько раз поток планируется на выполнение, и собирает об этом данные, с которыми можно ознакомиться, проверив данный поток с помощью программы Spy++ или определив содержимое счетчика `Thread:Context Switches/sec` программы Perfmon.

Родственность процессора для потоков

По умолчанию предусмотрено, что потоки процесса могут выполняться на любом из процессоров хост-компьютера. Если же процесс имеет несколько потоков, а на хост-компьютере установлено несколько процессоров, то операционная система Windows предпринимает попытки распределять потоки по процессорам таким образом, чтобы загрузка процессоров была максимально возможной.

Обычно рекомендуется дать операционной системе Windows возможность самой определить, на каком процессоре может выполняться тот или иной поток, а Windows всегда стремится поддерживать процессоры на хост-компьютере в состоянии максимальной активности и очень много делает для того, чтобы процедура выбора и планирования процессоров для потоков оставалась справедливой.

Но в некоторых ситуациях желательно ограничить перечень процессоров, на которых может выполняться тот или иной поток. Например, может существовать какая-то задача с высоким приоритетом, для которой желательно выделить отдельный процессор. В таком случае достаточно определить условие родственности одного процессора для данного потока таким образом, чтобы остальные потоки процесса не планировались для выполнения на данном процессоре. Это позволяет зарезервировать определенный процессор для использования в рассматриваемой задаче с высоким приоритетом (а также в других процессах, для потоков которых можно также в дальнейшем определить родственность процессора). Такой метод структуризации приложения, при использовании которого определенные потоки могут выполняться только на указанных процессорах, широко применяется при разработке высококачественного и высокопроизводительного программного обеспечения.

Но иногда наиболее удобным становится сочетание двух подходов, при котором разработчик определяет родственность процессоров для некоторых потоков и позволяет операционной системе Windows решать, на каких процессорах должны работать остальные потоки. Например, в описанном выше сценарии лучше оставить поток задачи с высоким приоритетом в состоянии, при котором для него не определяется процессор по принципу родственности. Тогда при всех прочих равных условиях операционная система Windows не будет предпринимать попыток запланировать этот поток для выполнения на процессорах, которые заняты другими действиями, если в системе имеется простаивающий процессор. Более того, поскольку в операционной системе Windows по умолчанию предусмотрено, что поток возобновляется на том процессоре, на котором он выполнялся перед этим, то после планирования потока для выделенного процессора, поток, по всей видимости, будет оставаться привязанным к процессору и в дальнейшем (поскольку другие потоки нельзя будет запланировать для выполнения на данном процессоре из-за того, что у них установлены неподходящие маски родственности процессора). В этом конкретном сценарии в целом является более масштабируемым гибридный подход, поскольку он не исключает возможность появления в какой-то момент еще одного работающего параллельно потока задачи с высоким приоритетом. В таком случае вполне можно предусмотреть выполнение и нового потока без определения для него приемлемого процессора. И даже если остается зарезервированным единственный процессор, на котором могут выполняться эти два потока с высоким приоритетом, то, не определяя для них процессор по принципу родственности, вы учитываете возможность, что может стать доступным еще один процессор, а это позволит обеспечить выполнение второго потока с высоким приоритетом, которому не нужно будет ждать, пока наступит перерыв в выполнении первой задачи с высоким приоритетом. Иными словами, даже если для обслуживания потребностей потоков задач с высоким приоритетом будет отведен лишь один процессор, одному из этих потоков будет разрешено работать на другом процессоре, если выделенный процессор занят. А поскольку другие потоки не могут использовать этот процессор, подобная ситуация будет возникать лишь тогда, когда выделенный процессор выполняет одну из задач с высоким приоритетом и возникает необходимость в выполнении другой высокоприоритетной задачи. Не определяя для этих задач процессор, заданный по условию родственности, можно избежать бесполезного расходования неиспользованных ресурсов процессора на компьютере и исключить необходимость назначать выделенный процессор для каждой задачи с высоким приоритетом.

Безусловно, операция переноса потока с одного процессора на другой является дорогостоящей, поскольку при ее выполнении вероятность оптимального использования вторичного кэша каждого процессора намного ниже по сравнению с ситуацией, в которой каждый поток всегда выполняется на одном и том же процессоре. Поэтому, безусловно, существуют обстоятельства, в которых имеет смысл установить строгие условия родственности процессоров для потоков некоторого процесса, а не предоставлять возможность операционной системе Windows принимать решение о том, на каком процессоре может выполняться каждый поток. Таким образом, единого подхода для всех приложений не существует.

Автор рекомендует дать возможность операционной системе Windows самой устанавливать условия родственности процессоров для потоков лишь в том случае, если просто не существует других способов добиться требуемых показателей производительности и масштабируемости. Если в ходе настройки приложения придется заняться определением выделенных процессоров для некоторых задач, то наилучшим подходом становится метод проб и ошибок; вам придется немного поэкспериментировать, и вашим лучшим учителем станет приобретенный при этом опыт.

Типы родственности

Для потоков в процессе Windows может быть определен один из трех основных типов родственности процессора: родственность последнего по времени процессора (или "мягкая" родственность), идеальная и "жесткая" родственность. "Жесткая" родственность представляет собой именно тот тип родственности, который только что рассматривался, причем именно этот тип обычно подразумевается, когда речь идет о родственности процессора — при использовании этого типа для потока назначается определенный набор процессоров, и поток не может работать на других процессорах. После того как для потока определена родственность конкретного процессора или процессоров, операционная система Windows гарантирует, что поток будет выполняться только на этих процессорах. Система будет принудительно переводить рассматриваемый поток в состояние ожидания, в котором поток будет ожидать освобождения назначенного ему процессора (процессоров), даже если окажется, что другие процессоры простаивают. Определить условие "жесткой" родственности для отдельного потока можно, вызвав функцию `SetThreadAffinityMask` API-интерфейса Win32.

Условие родственности процессора можно также определить в заголовке исполняемого модуля, но для этой цели не предусмотрено ни одного параметра редактора связей. Тем не менее, чтобы изменить условия родственности процессора для некоторого процесса, можно отредактировать заголовок исполняемого файла с помощью программы `ImageCfg.exe`.

СОВЕТ. Программа Task Manager операционной системы Windows позволяет задавать родственность процессора для определенного процесса с помощью опции меню Set Affinity. Найдите нужный процесс в списке Processes, щелкните на нем правой кнопкой мыши и выберите команду Set Affinity из контекстного меню. Обратите внимание на то, что такая опция доступна только на многопроцессорных компьютерах и только в семействе Windows NT (в версиях Windows 9x не предусмотрена специальная поддержка для многопроцессорных компьютеров).

Как уже было указано выше, подход, в котором предусматривается определение жестких условий родственности, часто бывает не самым оптимальным с точки зрения производительности. Применение жестких условий родственности может привести к тому, что поток останется закрепленным за конкретным процессором, который будет интенсивно использоваться, тогда как другие процессоры компьютера будут простаивать. В этом сценарии лучшее решение может заключаться в использовании условия идеальной родственности. Определение условия идеальной родственности для потока равносильно тому, что системе

передается информация о том, какой процессор является предпочтительным для выполнения данного потока, но предоставляется также возможность запланировать этот поток для выполнения на других процессорах, если предпочтительный процессор занят. Чтобы определить идеальный процессор для потока, можно вызвать функцию `SetThreadIdealProcessor` API-интерфейса Win32.

Модель определения условия родственности процессора для потока третьего типа, поддерживаемая операционной системой Windows, — это родственность последнего по времени процессора. По умолчанию в операционной системе Windows используется условие родственности именно этого типа; при всех прочих равных условиях в Windows предпринимаются попытки запланировать поток для выполнения на том же процессоре, который использовался потоком в последнюю очередь. Это позволяет в максимальной степени использовать вторичный кэш процессора (в надежде на то, что часть данных потока все еще остается в кэше к тому времени, как поток получит новый квант времени).

Операционная система Windows способна определить, на каком процессоре поток выполнялся в последний раз, поскольку отслеживает эту информацию в блоке привилегированного режима потока. В действительности в ОС Windows сопровождаются два номера процессора для каждого потока — последний по времени процессор, на котором работал поток, и его идеальный процессор.

После того как поток переходит в состояние готовности к планированию на выполнение, операционная система Windows вначале проверяет поток, чтобы определить, задан ли идеальный процессор. Если для потока предусмотрен идеальный процессор, Windows предпринимает попытку запланировать данный поток для выполнения на этом процессоре. Если идеальный процессор занят или для потока не предусмотрен идеальный процессор, ОС Windows предпринимает попытку запланировать поток для выполнения на функционирующем в настоящее время процессоре, т.е. на том процессоре, на котором в данный момент работает сам планировщик. А если этот процессор не отмечен как простаивающий, то система выбирает первый же простаивающий процессор, находя его путем просмотра маски простаивающих процессоров в направлении от наибольших номеров к наименьшим.

Если ни один процессор не обозначен сигналом как простаивающий, то ОС Windows сравнивает приоритет данного потока с приоритетом другого потока, выполняющегося в настоящее время на идеальном процессоре. Если данный поток имеет более высокий приоритет по сравнению с другим потоком, который работает на его идеальном процессоре, Windows вытесняет другой поток и планирует рассматриваемый поток для выполнения на данном процессоре.

Если же рассматриваемый поток имеет более низкий приоритет по сравнению с потоком, работающим на его идеальном процессоре (или для него не определен идеальный процессор), то Windows проверяет процессор, на котором этот поток выполнялся последний раз. Если рассматриваемый поток имеет более высокий приоритет по сравнению с потоком, работающим в настоящее время на данном процессоре, Windows вытесняет этот поток и планирует рассматриваемый поток для выполнения на данном процессоре.

Если же данный поток не может вытеснить ни поток, работающий на его идеальном процессоре, ни поток, работающий на его последнем по времени процессоре,

система проверяет другие процессоры в системе, чтобы определить, может ли она вытеснить какой-либо из работающих на них потоков. Проверка процессоров осуществляется в направлении от процессора с наибольшим номером в активной маске родственности к процессору с наименьшим номером. Если будет обнаружен поток с более низким приоритетом, выполняемый на одном из этих процессоров, то система вытеснит его в пользу готового к выполнению потока.

Очевидно, что если с потоком связана "жесткая" маска родственности процессоров, то перечень процессоров, которые могут рассматриваться в описанном выше процессе поиска, становится ограниченным. Например, даже если для некоторого потока будет определен идеальный процессор, поток так и не сможет работать на нем, если этот процессор не определен в указанной "жесткой" маске родственности процессоров.

Если не простаивает ни один из процессоров, а поток не может вытеснить ни один из прочих работающих потоков, то рассматриваемый поток помещается в очередь готовых потоков и снова рассматривается как кандидат для планирования лишь после того, как до него еще раз дойдет очередь. Следует отметить, что операционная система Windows никогда не перемещает потоки, чтобы освободить место для тех потоков, для которых определены условия родственности. Если для потока *Thread A* определен по условию родственности процессор *CPU 0*, и *CPU 0* занят, а для потока *Thread B* определены по условию родственности и процессор *CPU 0*, и процессор *CPU 1* (который простаивает), то система не перемещает поток *Thread B* с процессора *CPU 0* на процессор *CPU 1*, чтобы дать возможность выполнять поток *Thread A*. Последнему просто приходится ожидать, когда наступит его очередь на использование процессора *CPU 0*.

Выбор потока

Если в планировщике возникает необходимость найти новый поток для выполнения на процессоре, который в настоящее время выполняет какую-то работу (например, после того как выполняемый в данный момент поток переходит в состояние ожидания, понижает свой приоритет, изменяет заданный для него по условию родственности процессор и т.д.), в операционной системе Windows используется простой алгоритм для определения того, какой поток должен начать работу. На однопроцессорном компьютере операционная система Windows выбирает первый готовый поток из списка очередей готовых потоков, начиная с наивысших приоритетов и постепенно переходя к более низким. А на многопроцессорном компьютере операционная система выбирает поток, который соответствует одному из перечисленных ниже условий.

- Поток выполнялся на данном процессоре последним.
- Для потока рассматриваемый процессор задан в качестве идеального процессора.
- Приоритет потока больше или равен 24.
- Поток был готов к выполнению дольше, чем в течение 2 квантов времени.

Приостановка и возобновление работы потоков

Операционная система Windows позволяет приостанавливать и возобновлять работу потоков с помощью вызовов функций SuspendThread и ResumeThread API-интерфейса. Приостановленный поток не использует процессорное время и не планируется для выполнения до тех пор, пока система не возобновит его работу.

Система поддерживает счетчик приостановок для каждого потока в объекте привилегированного режима потока. Счетчик приостановок для потока показывает, сколько раз этот поток был приостановлен. Поток планируется для выполнения только при том условии, что значение счетчика приостановок не равно 0. А после того как значение счетчика приостановок потока достигает 0, поток становится применимым для планирования, если не ожидает какого-то другого события (например, завершения операции ввода-вывода).

При первоначальном создании потока в приложении операционная система Windows устанавливает значение его счетчика приостановок, равное 1, чтобы исключить возможность того, что этот поток будет запланирован для выполнения в ходе своей инициализации. А после инициализации потока система проводит проверку, чтобы определить, не был ли этот поток создан с условием CREATE_SUSPENDED (создать как приостановленный). Если поток действительно создан с указанным флажком, то счетчик приостановок остается неизменным; в противном случае счетчик сбрасывается в нулевое значение. Если поток создается, чтобы начать работу в приостановленном состоянии, то в приложении необходимо вызвать функцию ResumeThread, чтобы разрешить выполнение этого потока.

Если вызов функции ResumeThread выполняется успешно, функция возвращает предыдущее значение счетчика приостановок потока (в противном случае возвращает значение 0xFFFFFFFF (-1)). Это — полезная информация, учитывая то, что поток может стать применимым для планирования лишь после того, как его выполнение будет возобновлено столько же раз, сколько он был приостановлен. Например, значение, возвращаемое функцией ResumeThread, может использоваться в качестве переменной управления циклом, чтобы можно было обеспечить присваивание счетчику приостановок потока значения, равного 0, после того как возникнет необходимость перевести поток в рабочее состояние.

Итак, очевидно, что поток может самостоятельно приостановить свою работу, но не может так же ее возобновить. После приостановки работы потока возобновить его работу должен какой-то другой поток, чтобы первый поток мог продолжить свое выполнение.

Средствами приостановки и возобновления работы потоков необходимо пользоваться с осторожностью, поскольку разработчик не всегда имеет полное представление о том, какие действия осуществляются в потоке в тот момент, когда он приостанавливается. Обычно лучше всего реализовывать в коде функции потока таким образом, чтобы он переходил в состояние ожидания (например, предусматривая в нем ожидание освобождения синхронизационного объекта), когда требуется на время прекратить его выполнение, а не устанавливать для потока вынужденную паузу, вызывая функцию SuspendThread, если точно не известно, какие действия осуществляются в потоке в момент его приостановки.

Например, если приостанавливается работа потока, который открыл мьютекс или критическую секцию, это может вызвать непреднамеренную блокировку выполнения других потоков, которые ожидают освобождения объекта, открытого приостановленным потоком.

Перевод потока в состояние ожидания

Операционная система Windows позволяет не только приостановить работу потока на неопределенно долгое время (до тех пор пока работа не будет возобновлена из другого потока), но и прекратить работу потока на заданный период времени, переведя его в состояние ожидания. Чтобы перевести поток в состояние ожидания на указанное количество миллисекунд, можно использовать функции `Sleep` или `SleepEx`.

Как и при использовании функций ожидания подсистемы Win32, функции `Sleep/SleepEx` можно передать константу `INFINITE`, чтобы вынудить поток перейти в состояние ожидания на неопределенно долгое время, но автор не может представить себе, как такой вариант вызова функции может применяться в реальном приложении. Если при этом используется функция `Sleep`, то нельзя прервать ожидание потока, чтобы обеспечить его дальнейшее применение. Поэтому вместо перевода потока в бесконечно долгое состояние ожидания лучше его уничтожить, чтобы освободить связанные с ним ресурсы системы. А если используется функция `SleepEx`, то бесконечное ожидание `INFINITE` может быть прервано только с помощью функции обратного вызова, связанной с завершением операции ввода-вывода, или с помощью асинхронного вызова процедуры (Asynchronous Procedure Call — APC).

Следует отметить, что в функцию `Sleep/SleepEx` можно передать значение задержки, равное 0, чтобы вынудить данный поток предоставить возможность начать выполнение другим потокам, имеющим по меньшей мере такой же приоритет. В версиях Windows, предшествующих Windows Server 2003, в том случае, если нужно было предоставить возможность приступить к выполнению потокам с более низким приоритетом, необходимо было перевести рассматриваемый поток в состояние ожидания на более продолжительное время, чем нулевое (потоки с более низким приоритетом получают возможность приступить к выполнению, даже если текущий поток переходит в режим ожидания на одну миллисекунду). В версиях, предшествующих Windows Server 2003, переход в состояние ожидания с продолжительностью 0 вынуждает систему сразу же снова запланировать вызывающий поток для выполнения, даже если исчерпаны ресурсы потоков с более низким приоритетом.

Вместо функции `Sleep` можно применить еще один интересный вариант — функцию `SwitchToThread`. Функция `SwitchToThread` предназначена именно для того, чтобы дать возможность какому-то потоку отказаться от использования оставшегося у него временного интервала и позволить приступить к работе другим потокам, независимо от того, имеют они такой же или более низкий приоритет по сравнению с текущим потоком.

Упражнения

В этих упражнениях показано, как провести эксперименты с приоритетами программы SQL Server, а также с приоритетами потоков/процессов. В них приведена информация о том, как выполнить запуск программы SQL Server в качестве процесса реального времени и определить приоритет этого процесса с помощью программы Task Manager. Кроме того, в них показано, как программа SQL Server переходит в состояние ожидания на то время, когда ей приходится ожидать завершения выполнения таких команд T-SQL, как `WAITFOR DELAY`. В этих упражнениях используется также расширенная процедура, позволяющая немного глубже ознакомиться с работой программы и узнать о приоритетах потоков, масках родственности, а также получить другую полезную информацию о рабочих потоках SQL Server.

Упражнение 3.6. Эксплуатация программы SQL Server с приоритетом процесса реального времени

Существует возможность применить настройку конфигурации SQL Server для эксплуатации этой программы с приоритетом процесса `High` (высокий). Такое значение приоритета можно установить в программе Enterprise Manager с помощью вкладки Processor диалогового окна Properties, относящегося к рассматриваемому серверу.

После изменения значения этого параметра настройки и перезапуска сервера программа Task Manager покажет, что теперь программа SQL Server работает с приоритетом `High`. Компания Microsoft обычно не рекомендует своим заказчикам изменять значение данного параметра, но нередко приходится встречаться с ситуациями, когда заказчики применяют такую настройку по своему усмотрению.

По этому пути можно пройти еще немного дальше и действительно обеспечить эксплуатацию SQL Server как процесса реального времени. Такой способ эксплуатации указанной программы также не рекомендуется, но с формальной точки зрения он возможен. Как уже было сказано, эксплуатация любого процесса с приоритетом `Real-Time` приводит к тому, что потоки этого процесса начинают конкурировать с потоками операционной системы за процессорное время, что в принципе может привести к замедлению или даже блокировке таких ключевых функций операционной системы, как дисковый ввод-вывод и распределение памяти. Поэтому автор предлагает читателю провести описанные ниже действия исключительно в качестве эксперимента. Вам не следует эксплуатировать программу SQL Server с приоритетом `Real-Time` на производстве, а также проводить описанные ниже испытания на любом компьютере, отличном от используемого для экспериментирования или разработки, поскольку при этом может стать неработоспособной даже сама операционная система.

Для того чтобы запустить программу SQL Server в режиме `Real-Time`, выполните описанные ниже действия.

1. Остановите работающий экземпляр SQL Server с помощью программы SQL Server Service Manager. Еще раз подчеркнем, что это должен быть экземпляр, предназначенный для экспериментов или разработки, и в идеальном случае вы должны быть его единственным пользователем.
2. Откройте окно с приглашением командной строки и перейдите в каталог, который содержит файл `sqlservr.exe` — исполняемый файл SQL Server. Этот файл должен находиться в подкаталоге `binn` главного инсталляционного каталога программы SQL Server.

3. Запустите программу `sqlservr.exe` в терминальном режиме с помощью следующей команды:

   ```
   start /realtime sqlservr.exe -c -sYourInstanceName
   ```

   Укажите вместо параметра *YourInstanceName* имя запускаемого вами экземпляра. Полностью исключите параметр `-s`, если вы запускаете экземпляр, применяемый по умолчанию.

4. Вы должны видеть, как происходит запуск программы SQL Server в качестве терминального приложения в отдельном окне. Переключитесь на программу Task Manager и проверьте столбец базового приоритета процесса, `Base Pri`. (Если в окне Task Manager столбец `Base Pri` не показан, выберите опцию меню View | Select Columns и отметьте указанный столбец в списке доступных столбцов.) Теперь в качестве базового приоритета процесса должно быть указано `Real-Time`.

5. Проверим, позволяет ли использование приоритета `Real-Time` быстрее доводить до конца выполнение запросов, требующих больших затрат процессорного времени. Подключитесь к экземпляру SQL Server с помощью программы Query Analyzer и выполните следующий запрос:

   ```
   declare @var int
   set @var=1
   while @var<100000 begin
     set @var=@var+1
   end
   ```

6. Запишите, сколько времени потребовалось на его выполнение. Это время зависит от частоты процессора, но не должно превышать нескольких секунд.

7. Теперь остановите программу SQL Server (перейдите в окно терминала и нажмите клавиши <Ctrl+C>) и снова запустите с помощью программы SQL Server Service Manager. В результате этого программа SQL Server должна снова возвратиться к предусмотренному в ней по умолчанию приоритету процесса.

8. Теперь снова выполните запрос с помощью программы Query Analyzer и проверьте, сколько времени потребовалось на его выполнение.

В системе автора продолжительность выполнения обоих прогонов была одинаковой; это означает, что вызов на выполнение программы SQL Server с приоритетом `Real-Time` не привел к ускорению применяемых автором запросов, которые создают большую нагрузку на процессор. При обычных условиях не следует рассчитывать на то, что эксплуатация программы SQL Server с более высоким приоритетом приведет к каким-то значительным изменениям; это может произойти лишь в том случае, если на компьютере одновременно эксплуатируются другие программы, имеющие достаточно высокий базовый приоритет, позволяющий им часто вытеснять программу SQL Server.

Общий вывод, к которому пришел автор, состоит в следующем: не следует рассчитывать на то, что эксплуатация программы SQL Server с более высоким приоритетом сама по себе автоматически ускорит работу системы. Изменив приоритеты процессов, можно лишь определить победителя в споре за ресурсы процессора; сами изменения непосредственно не приводят к автоматическому ускорению работы каких-либо программ. По этой причине (а также потому, что программа SQL Server не была сертифицирована как обеспечивающая безопасную эксплуатацию с приоритетом `Real-Time`) автор рекомендует оставить неизменным заданный по умолчанию приоритет программы SQL Server.

Упражнение 3.7. Ознакомление с тем, как программа SQL Server переходит в состояние ожидания

В этом упражнении показано, какие действия выполняются в программе SQL Server после того как программа получает указание перевести какое-то соединение в состояние ожидания с помощью команды `WAITFOR DELAY` языка Transact-SQL. Поскольку выше были описаны функции `Sleep` и `SleepEx` API-интерфейса Win32, то может показаться очевидным, что при выполнении команды `WAITFOR DELAY` в сценарии на языке Transact-SQL в программе SQL Server вызывается одна из этих функций для перевода потока в состояние ожидания. Поэтому необходимо понять, как в сервере фактически осуществляются эти действия и как обрабатываются языковые события, в отличие от событий дистанционного вызова процедур (Remote Procedure Call — RPC), чтобы получить определенное представление о том, как действительно работает сервер. Для того чтобы ознакомиться с внутренней организацией работы сервера, выполните описанные ниже действия.

1. Запустите экземпляр SQL Server, к которому вы можете подключиться с помощью отладчика. Компьютер, на котором проводятся эти эксперименты, не должен представлять собой компьютер производственного назначения, и в идеальном случае вы должны быть его единственным пользователем. Вы обнаружите, что лучше выполнить запуск сервера в качестве терминального приложения, чем в качестве службы, поскольку в последнем случае будет исключена возможность вызвать на выполнение программу SQL Server Agent, зависящую от этой службы.
2. Запустите отладчик WinDbg и убедитесь в том, что путь к файлам с отладочной информацией задан правильно, как описано в главе 2.
3. Подключитесь с помощью отладчика к программе SQL Server (нажмите клавишу <F6>). Найдите строку `sqlservr.exe` в списке работающих процессов отладчика WinDbg и дважды щелкните на ней. Если запуск экземпляра SQL Server был выполнен только что, то строка `sqlservr.exe` должна находиться в нижней части списка.
4. Если откроется окно Disassembly, закройте его, поскольку оно не требуется для данного упражнения. А если это окно будет снова открываться на любом этапе выполнения данного упражнения, также закрывайте его без колебаний.
5. Введите букву `g` в окне с приглашением командной строки и нажмите клавишу <Enter>. Это приведет к тому, что процесс SQL Server продолжит свою работу.
6. Откройте соединение программы Query Analyzer с сервером и выполните следующую команду:

   ```
   WAITFOR DELAY '00:00:30'
   ```

   Выполнение этой команды вынудит сервер установить паузу в данном соединении на 30 с.
7. Теперь вернитесь к отладчику и нажмите клавиши <Ctrl+Break>, чтобы остановить процесс SQL Server. Введите следующую команду для ознакомления с тем, какие действия выполняются в каждом потоке:

   ```
   ~*kv
   ```

   Эта команда выведет на экран стек вызовов каждого потока, а мы получим возможность определить, какое следующее действие должно быть выполнено в каждом потоке во время действия вызова `WAITFOR`, изучив эти стеки.

8. Внимательно рассмотрев каждый стек вызовов (при этом обязательным условием является то, что вы должны быть единственным пользователем сервера), вы обнаружите, что почти во всех потоках выполняется идентичная работа, за исключением одного потока. Пропустите первые несколько потоков, поскольку они представляют собой системные потоки, которые не соответствуют пользовательским соединениям. Перейдите в нижнюю часть списка стеков вызовов, после чего вы должны обнаружить один поток со списком вызовов функций, который заметно отличается от других. Особенность содержимого этого стека заключается в том, что в нем представлен вызов функции с именем `language_exec`. Проверьте, сможете ли вы найти этот стек. Следует отметить, что самой верхней функцией в этом стеке вызовов не является `Sleep` или `SleepEx`. Если это — поток, обслуживающий введенную раньше команду `WAITFOR DELAY`, то в нем, безусловно, для этой цели не используется функция `Sleep` или `SleepEx`. В применяемом автором экземпляре WinDbg стек, содержащий вызов функции `language_exec` (обозначена полужирным шрифтом), выглядит, как показано в листинге 3.5.

Листинг 3.5. Содержимое стека

```
21  Id: 8f8.65c Suspend: 1 Teb: 7ffa6000 Unfrozen
ChildEBP RetAddr  Args to Child
2619f674 77e8780f 000002fc 00000001 00000000 ntdll!
NtWaitForSingleObject+0xb (FPO: [3,0,0])
2619f69c 4107149d 000002fc ffffffff 00000001 KERNEL32!
WaitForSingleObjectEx+0x71 (FPO: [Non-Fpo])
2619f6b8 4107173f 251c8bd8 251c86e0 00bc66d8 UMS!
UmsThreadScheduler::Switch+0x58
2619f6dc 410717ff 00bc66d8 251c8bd8 42dd36fc UMS!
UmsScheduler::IdleLoop+0x11f
2619f6f4 41071918 00007530 00000001 42dd36ec UMS!
UmsScheduler::Suspend+0x7e
2619f710 0040129c 00007530 00000000 00000000 UMS!
UmsEvent::Wait+0x95
2619f754 00637256 00007530 00a6997c 00000000
sqlservr!ExecutionContext::WaitForSignal+0x1b5
2619f7a8 004160db 42dc8060 42dd3240 42dc8060 sqlservr!
CStmtWait::XretExecute+0x128
2619f814 00415765 42dd3550 00000000 2619f8d4
sqlservr!CMsqlExecContext::ExecuteStmts+0x27e
2619f858 00415410 00000000 00000000 42dd3240 sqlservr!
CMsqlExecContext::Execute+0x1c7
2619f8a4 00459a54 00000000 4200e700 42dd4038 sqlservr!
CSQLSource::Execute+0x343
2619fa64 004175d8 42dd6090 0024004c 251c43c0
sqlservr!language_exec+0x3c8
2619fefc 410735d0 42dd6090 2619fe90 00000000 sqlservr!
process_commands+0xe0
2619ff68 4107382c 00bc6770 00bc6770 00bc66d8 UMS!
ProcessWorkRequests+0x264
2619ff80 7800c9eb 251c21b0 0024004c 00530053 UMS!
ThreadStartRoutine+0xbd
2619ffb4 77e887dd 251c43c0 0024004c 00530053 MSVCRT!
_beginthread+0xce
2619ffec 00000000 7800c994 251c43c0 00000000 KERNEL32!
BaseThreadStart+0x52 (FPO: [Non-Fpo])
```

Самым верхним вызовом функции в этом стеке является вызов функции NtWaitForSingleObject; она представляет собой функцию собственного API-интерфейса системы, которая выполняет вызов к ядру Windows для фактического выполнения операции ожидания затребованной функции WaitForSingleObject. Поэтому, как уже было сказано выше, если в данном потоке действительно выполняется команда WAITFOR DELAY (а на вопрос, так ли это на самом деле, вскоре будет дан ответ), то можно утверждать со всей определенностью, что выполнение команды WAITFOR DELAY не приводит к вызову функции Sleep подсистемы Win32. (Поразмыслив над этим вопросом, можно понять, что дела обстоят именно так, даже не зная ничего о системе UMS, ведь команда WAITFOR поддерживает также опцию ожидания до наступления абсолютной даты/времени и поддерживает возможность ее отмены, тогда как очевидно, что функция Sleep не позволяет этого сделать.)

9. На основании того, что применяемая при этом функция имеет имя language_exec, можно сделать вывод, что именно она вызывается на выполнение при передаче на сервер пакета команд на языке T-SQL. Чтобы проверить это предположение, установим на данной функции точку останова и рассмотрим, что произойдет при повторном выполнении команды WAITFOR. Введите следующую команду в командном окне:

   ```
   bp language_exec
   ```

   Чтобы убедиться в том, что точка останова установлена, можно ввести bl в командном окне. После ввода команды bl в окне должна появиться примерно такая строка:

   ```
   0 e 004597ef     0001 (0001)  0:*** sqlservr!language_exec
   ```

   Важно отметить, что во втором столбце этой выходной строки находится буква e (сокращение от enabled), показывающая, что точка останова разрешена.

10. Введите g в окне с приглашением командной строки и нажмите <Enter>. Это приведет к тому, что процесс SQL Server продолжит свою работу.

11. Возвратитесь в программу Query Analyzer и щелкните на кнопке останова, чтобы отменить введенный вами запрос, затем снова вызовите его на выполнение.

12. Возвратитесь в отладчик WinDbg, после чего вы должны обнаружить, что отладчик остановился на заданной вами точке останова. Полученный при этом вывод должен выглядеть примерно таким образом:

    ```
    Breakpoint 0 hit
    eax=00000000 ebx=0097fb00 ecx=0000003c edx=00000000
    esi=42d9a090 edi=00000001
    eip=004597ef esp=260afa68 ebp=260afefc iopl=0
    nv up ei pl zr na po nc
    cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000
    efl=00000246
    sqlservr!language_exec:
    004597ef b8ff619300      mov     eax,0x9361ff
    ```

    Эти данные сообщают, что действительно достигнута точка останова, установленная перед этим, но кроме того логического заключения, к которому мы пришли на основании анализа имени функции language_exec, как убедиться в том, что именно эта функция вызывается при получении сервером информации о событии, связанном с обработкой сценария на языке T-SQL? Рассмотрим, что происходит при передаче на сервер информации о событии, связанном

с выполнением вызова RPC, чтобы определить, возникнет ли точка останова, установленная на функции `language_exec`, и в этой ситуации.

13. Введите `bd 0` в командном окне и нажмите <Enter>. Эта команда удалит точку останова, установленную перед этим. Это нужно для того, чтобы можно было создать процедуру для использования в подготавливаемом событии RPC.
14. Введите `g` в окне с приглашением командной строки и нажмите <Enter>. Это приведет к тому, что процесс SQL Server продолжит свою работу.
15. Возвратитесь в программу Query Analyzer и остановите выполнение введенного ранее запроса, если оно еще продолжается. Откройте новое окно Query Analyzer и создайте новую процедуру в базе данных `pubs` с помощью следующих команд:

```
USE pubs
GO
CREATE PROC waiter as WAITFOR DELAY '00:00:30'
```

Выполните этот пакет команд, чтобы создать новую процедуру.

16. Теперь откройте новое окно Query Analyzer и введите в нем следующую команду:

```
{CALL waiter}
```

Эта синтаксическая конструкция позволяет передать в хранимую процедуру `waiter` вызов ее как события RPC. Но после ввода этой команды еще не запускайте ее на выполнение.

17. Возвратитесь в отладчик WinDbg и нажмите клавиши <Ctrl+Break>, чтобы остановить процесс SQL Server.
18. Теперь снова разрешите использование установленной ранее точки останова, введя следующую команду в командном окне и нажав <Enter>:

```
be 0
```

19. Введите `g` в окне с приглашением командной строки и нажмите <Enter>. Это приведет к тому, что процесс SQL Server продолжит свою работу.
20. Возвратитесь в программу Query Analyzer и выполните процедуру `waiter` с помощью синтаксической конструкции RPC, введенной перед этим.
21. Снова переключитесь на отладчик WinDbg и узнайте, была ли достигнута установленная точка останова. Этого не должно произойти. Итак, мы получили достаточно надежные свидетельства того, что `language_exec` — это внутренняя функция, используемая в программе SQL Server при обработке событий, связанных с использованием языка T-SQL. Автор рекомендует попытаться выполнить некоторые другие запросы, чтобы узнать, приводят ли они к активизации точки останова, установленной на функции `language_exec`. Если такие запросы не будут передаваться на сервер с помощью синтаксической конструкции RPC, описанной выше, то каждый пакет, переданный на сервер, должен вызывать активизацию точки останова по меньшей мере один раз. А запросы со встроенными командами `GO` должны приводить к возникновению точки останова несколько раз, поскольку они передаются на сервер отдельно с помощью программы Query Analyzer.
22. У читателя может возникнуть вопрос: какая внутренняя функция вызывается под действием события, обусловленного выполнением команд с помощью RPC? Определим имя этой функции. Нажмите клавиши <Ctrl+Break>, чтобы остановить процесс SQL Server, затем введите `~*kv`, чтобы вывести на экран список содержимого стеков вызовов, относящихся к потокам процесса.

23. Как и прежде, стеки вызовов большинства потоков, отличных от системных, практически одинаковы, за исключением одного стека, характерной особенностью которого является наличие в нем вызова процедуры с именем `execute_rpc`. Установим точку останова на функции `execute_rpc`, чтобы определить, будет ли эта точка останова активизироваться при передаче на сервер информации о событии, связанном с выполнением конструкции RPC. Введите следующую команду в командном окне WinDbg:

    ```
    bp execute_rpc
    ```

24. Введите `g` в окне с приглашением командной строки и нажмите <Enter>. Это приведет к тому, что процесс SQL Server продолжит свою работу.

25. Возвратитесь в программу Query Analyzer и остановите выполнение вызова процедуры, если оно все еще не закончилось, затем снова вызовите эту процедуру.

26. Возвратитесь к отладчику WinDbg. Вы должны обнаружить, что установленная ранее точка останова достигнута. Результаты, выведенные на экран, должны выглядеть примерно так:

    ```
    Breakpoint 1 hit
    eax=00000000 ebx=0097fb00 ecx=00000018 edx=42dd30a8
    esi=42dd6090 edi=00000003
    eip=0043b7d2 esp=2619fa68 ebp=2619fefc iopl=0
    nv up ei pl zr na po nc
    cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000
    efl=00000246
    sqlservr!execute_rpc:
    0043b7d2 55              push    ebp
    ```

    Итак, мы можем на полном основании заключить, что при поступлении на сервер информации о событии, связанном с выполнением команд на некотором языке, вызывается функция `language_exec`, а при получении информации о событии RPC — функция `execute_rpc`. Автор рекомендует провести эксперименты с другими событиями RPC, которые связаны с выполнением команд с помощью средств языка и средств RPC, чтобы полностью разобраться в том, как это происходит.

27. Введите `q`, чтобы выйти из программы отладчика. После этого необходимо выполнить перезапуск экземпляра SQL Server с помощью программы SQL Server Service Manager.

Упражнение 3.8. Просмотр информации о приоритетах потоков, родственности процессоров и другой полезной информации

В этом последнем упражнении показано, как выполнить расширенную процедуру, позволяющую переходить от одного активного в настоящее время рабочего потока SQL Server к другому такому же потоку и выводить на экран важную информацию о каждом из них. Выполните перечисленные ниже шаги.

1. Скопируйте файл `xp_sysinfo.dll` из подкаталога `CH03\xp_sysinfo\release` компакт-диска, прилагаемого к этой книге, в подкаталог `binn` инсталляционного каталога программы SQL Server. (Может оказаться, что эта библиотека DLL уже была установлена, если перед этим читатель выполнил предыдущее упражнение данной главы.)

2. Установите процедуру xp_threadlist, выполнив следующую команду в программе Query Analyzer:

   ```
   sp_addextendedproc 'xp_threadlist','xp_sysinfo.dll'
   ```

3. Откройте сценарий xp_threadlist.sql на языке T-SQL из подкаталога CH03\xp_sysinfo в программе Query Analyzer. Но пока не вызывайте его на выполнение.

4. Создайте любой временный каталог на жестком диске используемого компьютера и скопируйте в него файлы STRESS.CMD и STRESS.SQL из подкаталога CH03 компакт-диска.

5. Вызовите на выполнение пакетный файл STRESS.CMD с помощью командной строки, подобной приведенной ниже, заменив параметр *YourServerName* именем применяемого сервера:

   ```
   stress stress.sql 15 N normal Y YourServerName
   ```

6. На рабочем столе компьютера должны открыться 15 окон с программой osql.exe для выполнения операции выборки на одной из таблиц в базе данных pubs. Введите команду WAITFOR DELAY.

7. В ходе выполнения этих запросов переключитесь в программу Query Analyzer и выполните сценарий, загруженный вами ранее. Должны быть получены результаты, аналогичные приведенным в листинге 3.6.

Листинг 3.6. Пример полученных результатов

```
ThreadID  Impersona-  HasAccess-  TebBase-      Affini-  BasePrio-  SID
          tionLevel   Token       Address       tyMask   rity
--------  ----------  ----------  ------------  -------  ---------  ----
556       N/A         0           0x7FF9A000    3        9          S-1-
1784      N/A         0           0x7FFDC000    3        9          S-1-
2784      N/A         0           0x7FF98000    3        9          S-1-
2808      N/A         0           0x7FF9B000    3        9          S-1-
2908      N/A         0           0x7FF91000    3        9          S-1-
2912      N/A         0           0x7FF9C000    3        9          S-1-
2920      N/A         0           0x7FF95000    3        9          S-1-
3068      N/A         0           0x7FF96000    3        8          S-1-
3232      N/A         0           0x7FF90000    3        9          S-1-
3280      N/A         0           0x7FF8D000    3        9          S-1-
3292      N/A         0           0x7FF92000    3        9          S-1-
3372      N/A         0           0x7FF97000    3        9          S-1-
3380      N/A         0           0x7FF93000    3        9          S-1-
3404      N/A         0           0x7FFA1000    3        9          S-1-
3452      N/A         0           0x7FFDD000    3        9          S-1-
3640      N/A         0           0x7FFA4000    3        9          S-1-
3812      N/A         0           0x7FF94000    3        9          S-1-
```

8. На многопроцессорном компьютере можно провести эксперименты по изменению значений родственности процессора SQL Server, чтобы узнать, какие различные значения появляются в столбце AffinityMask. В приведенном выше примере для каждого потока определены в качестве родственных процессоры 1 и 2 данного компьютера, что привело к получению битовой маски 3.

   Можно также использовать связанные серверные запросы, чтобы обеспечить анонимность конкретного рабочего потока. Все потоки в примере, приведенном выше,

имели унаследованный контекст защиты SQL Server и для них не были предусмотрены собственные лексемы защиты.

Следует отметить, что в столбце `BasePriority` могут обнаруживаться различные значения в зависимости от того, когда была вызвана на выполнение рассматриваемая расширенная процедура и какие действия выполняются в системе. Сам автор наблюдал существенную изменчивость этих значений.

Следует также отметить, что потоки, перечисленные в списке, представляют собой рабочие потоки, активные в настоящее время; в этот список не попадают неактивные или простаивающие потоки. Если требуется ознакомиться со всеми потоками, экземпляры которых в настоящее время применяются в процессе SQL Server, воспользуйтесь инструментальным средством наподобие Perfmon или Pview.

9. Все 15 окон должны закрыться без постороннего вмешательства, после чего система снова возвращается к нормальной работе.

Планирование потоков. Резюме

В операционной системе Windows реализован вытесняющий планировщик, действующий с учетом приоритетов, который предназначен для планирования и выполнения кода приложений с помощью потоков. Эта операционная система спроектирована таким образом, чтобы в ней исключалась возможность для отдельного приложения получить контроль над ресурсами всей системы и обеспечивалась равномерная производительность во всей системе.

Существует 32 уровня приоритетов, на которых может работать тот или иной поток. Потоки с более высокими приоритетами могут вытеснять потоки с более низкими.

Каждый поток получает запланированный интервал времени, называемый *квантом времени*, на протяжении которого он может выполняться. Точная длина кванта времени зависит от версии Windows, а также от того, является ли используемый компьютер многопроцессорным или однопроцессорным. Возможность полного использования потоком кванта времени не гарантируется, поскольку поток может быть вытеснен другим потоком с более высоким приоритетом.

В программе SQL Server, в отличие от большинства других многопотоковых приложений, почти не используются планировщик Windows и API-интерфейсы планирования. Это связано с тем, что программа SQL Server обеспечивает большую часть своих потребностей в планировании с использованием собственного средства UMS.

Планирование потоков. Вопросы для самопроверки

1. Что такое квант времени?
2. Какое основное действие осуществляется во время переключения контекста?
3. С помощью какой функции API-интерфейс Win32 можно задать идеальный процессор для потока?
4. Подтвердите или опровергните следующее утверждение. Одним из восьми состояний потока (обозначенных цифрами от 0 до 7) является приостанов-

ленное состояние, которое указывает, что поток был приостановлен с помощью вызова функции SuspendThread.

5. Подтвердите или опровергните следующее утверждение. Программа Task Manager не позволяет выводить на экран значения базового приоритета для отдельных потоков в процессе.
6. Какая часть архитектуры Windows определяет частоту прерываний от таймера?
7. Каковым является самый высокий приоритет процесса, доступный в операционной системе Windows?
8. Подтвердите или опровергните следующее утверждение. Ни один поток в системе не работает с приоритетом потока 0.
9. Каким термином обозначается ситуация, в которой поток с более низким приоритетом постоянно вытесняется потоками с более высокими приоритетами и не получает возможности работать?
10. Какие действия осуществляются в операционной системе Windows после обнаружения того, что поток не получает возможности работать в течение 3–4 с? Что происходит после того, как подобный поток начинает выполняться?
11. Подтвердите или опровергните следующее утверждение. В ОС Windows реализована система кооперативной многозадачности, по условиям функционирования которой процессы обязательно должны своевременно уступать право доступа к ресурсам процессора другим процессам, чтобы можно было обеспечить бесперебойное функционирование системы.
12. Какая функция API-интерфейса Win32 позволяет задать условие родственности процессора и отдельного потока?
13. Чему равен тактовый интервал на большинстве однопроцессорных компьютеров с процессором x86?
14. Какое количество единиц измерения квантов времени вычитается из кванта времени потока при каждом прерывании от таймера?
15. Подтвердите или опровергните следующее утверждение. В операционной системе Windows при принятии решения о том, какой поток должен быть запланирован для выполнения в следующую очередь, автоматически учитывается тот факт, что для потока (в том числе высокоприоритетного) только что закончился квант времени, поэтому ОС Windows автоматически разрешает вначале приступить к выполнению потокам с более низким приоритетом, и только после этого позволяет потоку с высоким приоритетом возобновить свою работу.
16. Каким термином обозначается условие, с помощью которого устанавливается связь между потоком или процессом и определенным множеством процессоров?
17. Подтвердите или опровергните следующее утверждение. После того как в потоке со значением приоритета меньше 14 завершается успехом ожидание освобождения объекта привилегированного режима, установленное с помощью функции WaitForSingleObject, система автоматически вычитает одну единицу измерения квантов времени из ее кванта времени.

18. Подтвердите или опровергните следующее утверждение. Функция, предназначенная для выполнения основного объема работы по планированию в ядре Windows, носит имя `ScheduleThread`.
19. Действительно ли вызов функции `Sleep(0)` в потоке приводит к тому, что получают разрешение на выполнение потоки с более низким приоритетом?
20. Подтвердите или опровергните следующее утверждение. В операционной системе Windows ведется отдельный список готовых потоков для каждого приоритета потока и сопровождается структура отображения, позволяющая ускорить доступ к этому списку.
21. Что происходит с квантом времени потока, имеющего приоритет 14 или выше, в котором только что успешно завершилось ожидание освобождения объекта привилегированного режима, организованное с помощью функции `WaitForSingleObject`?
22. Какая внутренняя функция программы SQL Server предназначена для обработки языковых событий? А что можно сказать о событиях RPC?
23. Подтвердите или опровергните следующее утверждение. В программе SQL Server для обслуживания команды Transact-SQL `WAITFOR DELAY` не используется функция `Sleep` или `SleepEx` API-интерфейса Win32.
24. Подтвердите или опровергните следующее утверждение. Эксплуатация программы SQL Server с приоритетом процесса `Real-Time` в системе, не загруженной ничем иным, способствует ускоренному выполнению запросов, требующих больших затрат процессорного времени.
25. Подтвердите или опровергните следующее утверждение. Базовый приоритет процесса можно изменить с использованием программы Windows Task Manager.
26. Подтвердите или опровергните следующее утверждение. Чтобы вывести на экран информацию о количестве переключений контекста для данного потока, можно воспользоваться утилитой Spy++, включенной в состав комплекта Platform SDK и последних версий Visual Studio.
27. Какова применяемая по умолчанию продолжительность кванта времени в версиях Windows 2000 Professional и Windows XP?
28. Опишите функциональные возможности функции `SwitchToThread` API-интерфейса.
29. Назовите числовое обозначение уровня приоритета, на котором поток рассматривается как поток реального времени.
30. Подтвердите или опровергните следующее утверждение. Даже если идеальный процессор потока не задан в маске родственности потока, все равно рассматриваемый поток может быть запланирован для выполнения на этом процессоре, поскольку в операционной системе Windows маска родственности игнорируется, если она конфликтует с условием определения идеального процессора.

Синхронизация потоков

Когда программист выходит за рамки простых однопотоковых приложений и начинает знакомиться с миром многопотокового программирования, он прежде всего обнаруживает, что в разрабатываемом приложении теперь приходится синхронизировать действия, выполняемые в отдельных потоках. В частности, позволив одному потоку модифицировать глобальную переменную, тогда как другой поток использует ее для управления ходом выполнения программы, программист невольно создаст предпосылки для ошибочного поведения приложения и, возможно, даже для нарушения доступа. А тот факт, что на многопроцессорных компьютерах многочисленные потоки могут выполняться в буквальном смысле этого слова одновременно, не означает, что действительно следует стремиться к тому, чтобы все эти потоки постоянно работали, не останавливаясь. Иногда возникает необходимость, чтобы один поток подождал завершения других потоков, прежде чем продолжить выполнение своей работы. Именно с этим связана необходимость в использовании средств синхронизации.

Одним из наиболее важных элементов синхронизации потоков является неразрывный доступ — обеспечение того, чтобы некоторый поток мог иметь доступ к ресурсу в таком режиме, который гарантирует, что больше ни один поток не получит доступ к тому же ресурсу одновременно с этим потоком. В операционной системе Windows предусмотрен целый ряд объектов и функций API-интерфейсов, позволяющих упростить задачу обеспечения неразрывного доступа. В этом разделе рассматривается каждое из этих средств и отдельно исследуется их функционирование.

Синхронизация потоков. Основные термины и определения

- **Синхронизация.** Задача обеспечения того, чтобы доступ потоков к ресурсам осуществлялся в режиме, позволяющем потокам безопасно использовать эти ресурсы, а пользователям — доверять данным, обрабатываемым в потоках.
- **Взаимоблокировка.** Ситуация, которая возникает, если два или несколько потоков переходят в состояние ожидания освобождения ресурсов, принадлежащих друг другу, на неопределенно долгое время.
- **Функция ожидания.** Одна из функций подсистемы Win32, предназначенная для перевода потока в состояние ожидания до тех пор, пока не станет доступным некоторый ресурс или не истечет установленный тайм-аут.
- **Сигнальный.** Состояние любого объекта, в котором объект доступен для использования потоком или в котором функция ожидания Win32 не должна ожидать освобождения объекта.
- **Несигнальный.** Состояние любого объекта, в котором объект недоступен для использования потоком или в котором функция ожидания Win32 должна ожидать освобождения объекта.

- **Спин-блокировка.** Конструкция непривилегированного режима, позволяющая непрерывно опрашивать ресурс для проверки его доступности. В спин-блокировках часто используется семейство функций комплексной блокировки.
- **Функция комплексной блокировки.** Функция из семейства функций Win32, которые обеспечивают выполнение простых, неразрывных операций обновления переменных.
- **Синхронизационный объект привилегированного режима.** Один из нескольких различных типов объектов привилегированного режима, которые могут использоваться для синхронизации доступа к ресурсам. К примерам синхронизационных объектов привилегированного режима относятся мьютексы, семафоры, события и многие другие объекты.
- **Потокобезопасный код.** Код, спроектированный таким образом, что в многочисленных потоках, получающих доступ к одним и тем же ресурсам, операции доступа выполняются в безопасной и предсказуемой форме.
- **Неразрывный доступ.** Обеспечение того, что в любом потоке выборка, модификация и возврат значения или ресурса осуществляется в виде одной операции, чтобы при этом не приходилось беспокоиться о том, что одновременно с этим модификация значения или ресурса может также осуществляться другим потоком.

Синхронизация потоков. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящихся к синхронизации потоков, приведены в табл. 3.14.

Таблица 3.14. Основные функции API-интерфейсов, относящихся к синхронизации потоков

Функция	Описание
`EnterCriticalSection`	Обозначить секцию кода, к которой в один момент времени может получить доступ только один поток
`LeaveCriticalSection`	Выйти из секции кода, которая предназначена для однопотокового доступа
`InterlockedExchange`	Выполнить операцию присваивания значения одной переменной другой в режиме неразрывного доступа
`InterlockedExchangeAdd`	Сложить значение одной переменной со значением другой переменной в режиме неразрывного доступа
`CreateEvent`	Создать объект события привилегированного режима
`SetEvent/ResetEvent`	Обозначить объект события как сигнальный/не сигнальный
`CreateSemaphore`	Создать объект семафора
`ReleaseSemaphore`	Освободить ссылку на семафор
`CreateWaitableTimer`	Создать объект таймера ожидания

Окончание табл. 3.14

Функция	Описание
`SetWaitableTimer`	Выполнить настройку конфигурации таймера ожидания
`CreateMutex`	Создать объект мьютекса (взаимоисключающей блокировки)
`ReleaseMutex`	Освободить мьютекс, принадлежащий потоку
`WaitForSingleObject`	Ожидать, пока не станет сигнальным объект привилегированного режима
`WaitForMultipleObjects`	Ожидать, пока не станут сигнальными несколько объектов привилегированного режима

Синхронизация потоков. Основные инструментальные средства

Большинство синхронизационных объектов представляют собой объекты привилегированного режима, поэтому количество информации о синхронизации потоков, которая может быть предоставлена любым инструментальным средством непривилегированного режима, весьма ограничена. К сожалению, в этих обстоятельствах наилучший вариант состоит в использовании отладчика привилегированного режима, который нам недоступен. Несмотря на сказанное выше, инструментальные средства, приведенные в табл. 3.15, предоставляют некоторые полезные фрагменты данных, относящихся к синхронизации.

Таблица 3.15. Диагностические инструментальные средства синхронизации потоков

	Количество дескрипторов	Количество переключений контекста	Контекст защиты потока	Распределение количества привилегированных объектов по типам	Приоритет потока	Состояние потока	Затраты процессорного времени
Perfmon	+	+		+		+	+
Pview	+	+	+	+	+	+	+
Spy++	+	+			+	+	+

Синхронизация с использованием конструкций непривилегированного режима

В операционной системе Windows предусмотрены два типа синхронизации потоков — непривилегированного и привилегированного режима. Синхронизация непривилегированного режима реализуется с помощью функций из библиотеки `Kernel32.DLL` и, как указывает само название синхронизации этого типа, не требует переключения потока в привилегированный режим. Поэтому средства синхронизации непривилегированного режима являются более быстродействующими по сравнению со средствами синхронизации потоков, в которых используются

объекты привилегированного режима. С другой стороны, объекты непривилегированного режима не могут применяться для синхронизации потоков из нескольких процессов, а также не могут использоваться в сочетании с функциями ожидания Win32, позволяющими обеспечить переход потока в состояние ожидания ресурса, в котором не потребляются какие-либо ресурсы процессора, и ожидающему потоку предоставляется возможность завершить ожидание по тайм-ауту. К примерам синхронизационных объектов/конструкций непривилегированного режима относятся спин-блокировки и критические секции.

Спин-блокировки

Проще всего спин-блокировку можно определить как цикл, который повторяется до тех пор, пока не станет доступным некоторый ресурс. В листинге 3.7 приведен простой пример на языке C++.

Листинг 3.7. Простая реализация спин-блокировки

```
void CSpinLock::GetLock() {
  while (TRUE == InterlockedExchange(&g_bLocked, TRUE))
    SwitchToThread();

  // Использовать ресурс.
  // "Разблокировать" ресурс
  InterlockedExchange(&g_bLocked, FALSE);

}
```

Функция `InterlockedExchange`, дополнительная информация о которой приведена чуть ниже, присваивает значение своего второго параметра своему первому параметру в режиме неразрывного доступа и возвращает первоначальное значение своего первого параметра. В приведенном выше коде спин-блокировки просто повторяется цикл до тех пор, пока первоначальное значение переменной `g_bLocked` остается равным `TRUE`; иными словами, пока рассматриваемый глобальный ресурс остается заблокированным. В этом цикле переменной `g_bLocked` значение `TRUE` присваивается до тех пор, пока функция `InterlockedExchange` больше не будет возвращать `TRUE`. А после того как функция `InterlockedExchange` возвратит `FALSE` (а это означает, что ресурс не был заблокирован к моменту вызова этой функции), происходит выход из цикла. Поскольку функция `InterlockedExchange` уже установила значение `g_bLocked`, равное `TRUE`, то другие потоки, вызывающие метод `GetLock`, останавливаются на этом цикле `while` до тех пор, пока переменная `g_bLocked` не примет значение `FALSE` в том потоке, который только что приобрел эту спин-блокировку.

Вполне очевидно, что спин-блокировка не представляет собой объект непривилегированного режима отдельного типа; скорее всего, ее можно считать объектом, реализованным с использованием объектов и кода непривилегированного режима, в данном случае с помощью глобальной переменной и одной из функций комплексной блокировки. Хотя спин-блокировки часто оформляются в виде класса определенного типа в таких объектно-ориентированных языках, как C++ (по

крайней мере, если речь идет об операционной системе Windows), фактически спин-блокировки в большей степени представляют собой программную конструкцию, чем отдельный тип синхронизационного объекта.

Спин-блокировки и загрузка процессора

Даже несмотря на то, что в приведенном выше примере предприняты попытки добиться максимально эффективного использования процессора за счет вызова в цикле спин-блокировки функции `SwitchToThread`, все равно эта конструкция потребляет некоторое количество процессорного времени на протяжении того, как с ее помощью происходит ожидание освобождения ресурсов. К сожалению, этот недостаток невозможно устранить без помощи одного из средств планирования (например, предусмотренных в Windows), которое позволяло бы сопровождать списки ожидающих потоков и необходимых для них ресурсов, независимо от самих потоков, по сути, переводя потоки в состояние ожидания до тех пор, пока не станут доступными требуемые для них ресурсы.

В этом состоит основная причина того, что объекты привилегированного режима, такие как мьютексы и семафоры, обладают существенным преимуществом перед спин-блокировками с точки зрения загрузки процессора. Поскольку операционная система Windows может действовать от имени ожидающего потока и позволить этому потоку не потреблять никаких дополнительных ресурсов, пока ресурсы, освобождения которых он ожидает, остаются недоступными, то средства поддержки режима ожидания, основанные на использовании объектов привилегированного режима, часто являются более эффективными с точки зрения применения ресурсов процессора по сравнению со средствами, в которых для реализации режима ожидания освобождения ресурса используются спин-блокировки, даже несмотря на то, что при применении средств первого типа в потоке необходимо выполнить переход из непривилегированного режима в привилегированный, чтобы организовать ожидание с помощью объекта привилегированного режима.

Кроме того, поскольку работа спин-блокировок, подобных приведенной выше, не координируется операционной системой, разработчик, использующий их в своем приложении, должен исходить из описанных ниже условий.

1. В спин-блокировках не учитывается приоритет потоков, т.е. предполагается, что все потоки функционируют на одном и том же уровне приоритета потоков (именно по этой причине попытка увеличивать в условиях использования взаимоблокировок приоритет потоков лишена смысла).
2. При использовании спин-блокировок необходимо предусмотреть, чтобы переменная блокировки и данные, к которым соответствующая блокировка предоставляет доступ, сопровождались в областях памяти, к которым обращаются разные процессоры. Если же они находятся в области памяти, к которой обращается один и тот же процессор, то возникает конкуренция между процессором, использующим ресурс, и всеми прочими процессорами, ожидающими освобождения ресурса. Иными словами, непрерывно выполняемые другими процессорами операции присваивания значения управляющей переменной будут конкурировать с кодом, который получает доступ к защищенному ресурсу.

3. По этой причине применение спин-блокировок целесообразно лишь на многопроцессорном компьютере. Если же потоки, пытающиеся получить доступ к ресурсу, и тот поток, который в настоящее время его заблокировал, совместно используют один процессор, то возникает значительная конкуренция между потоком, использующим ресурс, и ожидающими потоками, поскольку последние постоянно выполняют операции присваивания значения управляющей переменной.

Вообще говоря, следует избегать использования методов и элементов проекта, в которых предусматривается непрерывное проведение опроса для определения доступности ресурса. В операционной системе Windows предусмотрен богатый набор инструментов, позволяющих организовать ожидание освобождения ресурсов с минимальной загрузкой процессора. Поэтому имеет смысл применять именно те средства, которые не создают лишней нагрузки на компьютер с операционной системой Windows.

Часто возникает также ситуация, в которой наиболее приемлемым для осуществления конкретного проекта становится гибридный подход — применение спин-блокировок и критических секций для защиты ресурсов одних типов, а объектов привилегированного режима — для защиты других. Еще один вариант состоит в том, что спин-блокировка используется для обеспечения ожидания на фиксированное количество итераций, а вслед за этим происходит переход к применению объекта привилегированного режима, если становится очевидно, что для данного потока необходимо обеспечить ожидание в течение более продолжительного периода времени. Фактически именно по этому принципу реализованы сами критические секции. Критическая секция начинает свою работу с использования спин-блокировки, которая выполняет заданное количество итераций, а затем в критической секции осуществляется переход в привилегированный режим, и в этом режиме происходит ожидание освобождения требуемого ресурса.

Функции комплексной блокировки

В операционной системе Windows предусмотрено семейство функций API-интерфейса, известных под общим названием функций комплексной блокировки. Выше уже был приведен простой пример использования этих функций. Функции комплексной блокировки обеспечивают простой, удобный метод синхронизации потоков, в котором не используются объекты привилегированного режима. Общие сведения об этих функциях приведены в табл. 3.16.

Заслуживает внимания то, что отсутствуют функции комплексной блокировки, предназначенные для чтения некоторого значения. Это связано с тем, что подобная функция не нужна. Если бы в потоке предпринимались попытки чтения значения, которое постоянно модифицируется с использованием функции комплексной блокировки, то получение правильного значения могло бы происходить лишь в определенных обстоятельствах. Таким образом, вполне можно предположить, что будет считано либо то значение переменной, каковым оно было до его изменения, либо то значение, каким оно стало после изменения, а система гарантирует, что считанное значение будет либо тем, либо другим, а не каким-либо промежуточным.

Таблица 3.16. Семейство функций комплексной блокировки

Функция	Описание
`InterlockedIncrement`	Увеличить значение переменной и проверить ее значение в одной неразрывной операции
`InterlockedDecrement`	Уменьшить значение переменной и проверить ее значение в одной неразрывной операции
`InterlockedExchangePointer`	Заменить значение, на которое указывает первый параметр, значением, переданным во втором параметре, в одной неразрывной операции
`InterlockedExchangeAdd`	Сложить значение, на которое указывает первый параметр, со значением, переданным во втором параметре, в одной неразрывной операции
`InterlockedCompareExchangePointer`	Сравнить два значения и заменить первое значение третьим с учетом результатов сравнения в одной неразрывной операции

Критические секции

Критическая секция — это объект непривилегированного режима, который может использоваться для синхронизации потоков и упорядочения доступа к совместно применяемым ресурсам. В качестве критической секции обозначается фрагмент кода, для выполнения которого в один момент времени может использоваться лишь один поток. Организация работы программы, в которой применяется критическая секция, осуществляется в основном, как описано ниже.

1. Инициализация критической секции с помощью вызова функции `InitializeCriticalSection`. Эта операция часто выполняется при запуске программы и в основном используется для настройки критической секции, структура которой хранится в глобальной переменной.
2. На входе в такую процедуру, для выполнения которой необходимо использовать в один момент времени только один поток, вызывается функция `EnterCriticalSection`, и ей передается инициализированная ранее структура критической секции. После выполнения этой операции все другие потоки, предпринимающие попытки войти в данную процедуру, переводятся в состояние ожидания до того момента, как текущий поток выйдет из процедуры.
3. После выхода из процедуры текущий поток вызывает функцию `LeaveCriticalSection`.
4. Во время завершения работы программы (или после наступления какого-то другого аналогичного события завершения, которое происходит после того, как критическая секция становится больше не нужной) вызывается функция `DeleteCriticalSection` для освобождения ресурсов системы, используемых этим объектом.

Поскольку при реализации описанного выше подхода невозможно задать количество времени, в течение которого поток должен находиться в состоянии

ожидания, вполне возможно, что потоку придется ожидать освобождения этого ресурса неопределенно долгое время. Такое положение складывается, например, после того, как критическая секция становится никому не принадлежащей из-за аварийного прекращения работы владеющего ею потока в результате исключительной ситуации. Ожидающие потоки не имеют возможности определить, что поток, который перед этим получил доступ к критической секции, больше никогда его не освободит, поэтому вынуждены в течение неопределенно долгого времени ожидать освобождения ресурса, который так и не станет доступным.

Один из способов предотвращения такой ситуации, в которой проявляется принцип "все или ничего", состоит в использовании функции `TryEnterCriticalSection`, а не `EnterCriticalSection`. Функция `TryEnterCriticalSection` никогда не позволяет переводить какой-либо поток в состояние ожидания. Она либо приобретает контроль над критической секцией, либо немедленно возвращает `FALSE`. Тот факт, что функция `TryEnterCriticalSection` имеет возвращаемое значение, а функция `EnterCriticalSection` его не имеет, обусловлен именно способностью первой немедленно выполнять возврат.

Если в потоке вызывается функция `EnterCriticalSection` для входа в объект критической секции, который уже принадлежит другому потоку, операционная система Windows переводит первый поток в состояние ожидания. Это означает, что должен быть выполнен переход из непривилегированного режима в привилегированный, а для выполнения этой операции требуется примерно 1 тыс. циклов процессора. Но такой переход обычно требует меньше процессорного времени по сравнению с использованием спин-блокировки того или иного типа, которая непрерывно опрашивает ресурс, чтобы определить, стал ли он доступным.

Можно объединить подходы, основанные на использовании спин-блокировки и критической секции, с помощью функции `InitializeCriticalSectionAndSpinCount`. Эта функция позволяет указать количество итераций спин-блокировки, которые выполняются, если не удается сразу же войти в критическую секцию. Если критическая секция недоступна, то функция выполняет указанное количество повторных попыток войти в критическую секцию, и только после этого переходит в состояние ожидания. Если обычно продолжительность ожидания невелика, то применение указанного подхода позволяет исключить издержки, связанные с ненужным переходом в привилегированный режим.

Следует отметить, что указывать количество итераций спин-блокировки имеет смысл только на многопроцессорном компьютере. Дело в том, что на однопроцессорном компьютере поток, которому принадлежит критическая секция, не может ее освободить, пока другой поток с помощью спин-блокировки пытается проверить, освободилась ли критическая секция, поэтому на таком компьютере в функции `InitializeCriticalSectionAndSpinCount` игнорируется ненулевое значение параметра количества итераций спин-блокировки и, если критическая секция недоступна, немедленно осуществляется переход в состояние ожидания.

Чтобы установить количество итераций спин-блокировки для конкретной критической секции, можно воспользоваться функцией `SetCriticalSectionSpinCount` API-интерфейса Win32. Оптимальное значение этого количества зависит от ситуации, но в качестве приемлемого критерия может служить тот факт,

что для критической секции, применяемой для синхронизации доступа к области динамической памяти процесса, предусмотрено количество итераций спин-блокировки, равное 4000.

Как правило, для каждого совместно используемого ресурса необходимо определить отдельную переменную критической секции. Не следует пытаться экономить ресурсы системы, предусматривая совместное применение критических секций для разных ресурсов. Во-первых, критические секции не расходуют слишком большой объем памяти, а, во-вторых, непродуманная попытка обеспечить совместное использование критических секций может привести к усложнению кода и созданию предпосылок возникновения взаимоблокировок.

Потоки и состояния ожидания

Как было указано выше, пока поток ожидает освобождения ресурса, система действует как представитель потока, защищающий его интересы. Сам же поток, находясь в состоянии ожидания, не потребляет ресурсы процессора. Система переводит поток в состояние ожидания, а затем автоматически активизирует, после того как становится доступным ресурс (ресурсы), ожидаемый потоком.

Если будет выполнена проверка состояний потоков по всем процессам в системе, то обнаружится, что большинство из них основную часть времени находятся в состоянии ожидания того или иного типа. То, что большинство потоков любого процесса проводит основную часть своего времени в ожидании какого-то события (например, ввода данных с помощью клавиатуры или мыши), является вполне нормальным. Именно поэтому особенно важно, чтобы в операционной системе были предусмотрены механизмы ожидания освобождения ресурсов, позволяющие максимально эффективно использовать ресурсы процессора.

Взаимоблокировки между потоками

Взаимоблокировка потоков возникает в том случае, если каждый из двух потоков ожидает освобождения ресурсов, заблокированных другим потоком. Если каждый из этих потоков настроен на ожидание в течение неопределенно долгого времени, то потоки становятся в буквальном смысле слова заблокированными и никогда не смогут быть запланированы на дальнейшее выполнение. В отличие от программы SQL Server, операционная система Windows не обнаруживает взаимоблокировки автоматически. После того как потоки заключают друг друга в "смертельные объятия", единственным выходом из положения становится прекращение работы взаимно заблокированных потоков. В листинге 3.8 приведен код C++, который иллюстрирует широко распространенный сценарий взаимоблокировки.

Листинг 3.8. Классический сценарий взаимоблокировки

```
// deadlock.cpp
//

#include "stdafx.h"
#include "windows.h"
```

```
CRITICAL_SECTION g_CS1;
CRITICAL_SECTION g_CS2;

int g_HiTemps[100];
int g_LoTemps[100];

DWORD WINAPI ThreadFunc1(PVOID pvParam)
{
EnterCriticalSection(&g_CS1);
  Sleep(5000);
  EnterCriticalSection(&g_CS2);

  for (int i=0; i<100; i++)
     g_HiTemps[i]=g_LoTemps[i];

  printf("Exiting ThreadFunc1\n");

  LeaveCriticalSection(&g_CS1);
  LeaveCriticalSection(&g_CS2);

  return(0);
}

DWORD WINAPI ThreadFunc2(PVOID pvParam)
{
  EnterCriticalSection(&g_CS2);
  EnterCriticalSection(&g_CS1);

  for (int i=0; i<100; i++)
     g_HiTemps[i]=g_LoTemps[i];

  printf("Exiting ThreadFunc2\n");

  LeaveCriticalSection(&g_CS2);
  LeaveCriticalSection(&g_CS1);

  return(0);
}

int main(int argc, char* argv[])
{
  DWORD dwThreadId;
  HANDLE hThreads[2];

  InitializeCriticalSection(&g_CS1);
  InitializeCriticalSection(&g_CS2);

  hThreads[0]=CreateThread(NULL,0,ThreadFunc1,NULL,0,&dwThreadId);
  hThreads[1]=CreateThread(NULL,0,ThreadFunc2,NULL,0,&dwThreadId);

  WaitForMultipleObjects(2,hThreads,TRUE,INFINITE);

  DeleteCriticalSection(&g_CS1);
  DeleteCriticalSection(&g_CS2);

  return 0;
}
```

Проблема, возникающая в процессе эксплуатации данного кода, состоит в том, что два рабочих потока обращаются к ресурсам в разных последовательностях: функция ThreadFunc1 входит в критические секции в порядке номеров этих секций, а в функции ThreadFunc2 применяется другой принцип. Автор помес-

тил вызов функции Sleep после первой критической секции, в которую входит функция первого потока ThreadFunc1, чтобы дать возможность функции второго потока ThreadFunc2 начать работу и распределить вторую критическую секцию прежде, чем в нее сможет войти функция ThreadFunc1. После истечения тайм-аута функции Sleep функция ThreadFunc1 пытается войти во вторую критическую секцию, но блокируется функцией ThreadFunc2, а функция ThreadFunc2, с другой стороны, ожидает освобождения критической секции 1, которая уже принадлежит функции ThreadFunc1. Поэтому каждый поток переходит на неопределенно долгое время в состояние ожидания освобождения ресурсов, заблокированных другой функцией, что представляет собой классический сценарий взаимоблокировки.

На основании этого примера можно сделать следующий вывод: в многопотоковых приложениях необходимо всегда соблюдать согласованный порядок выдачи запросов на доступ к ресурсам. В этом состоит один из принципов качественного проектирования, который должен соблюдаться независимо от того, предназначена ли разрабатываемая программа для работы с объектами непривилегированного режима или с объектами привилегированного режима.

Синхронизация с использованием объектов привилегированного режима

Как было указано выше, методы синхронизации потоков с помощью объектов непривилегированного режима являются более быстродействующими по сравнению с методами синхронизации, в которых используются объекты привилегированного режима, но необходимо учитывать и другие условия. Объекты непривилегированного режима не могут служить для синхронизации многочисленных процессов, а также не позволяют воспользоваться функциями ожидания Win32 для организации ожидания их освобождения. Поскольку при переходе в состояние ожидания освобождения таких объектов непривилегированного режима, как критические секции, невозможно задать значение тайм-аута, то вполне может произойти блокировка других потоков и возникнуть ситуация взаимоблокировки. С другой стороны, для перехода в привилегированный режим приходится затрачивать около тысячи тактов процессора x86 (причем в это количество времени не входит фактическая продолжительность выполнения кода привилегированного режима, в котором реализована вызываемая функция), поэтому, безусловно, приходится учитывать требования производительности, принимая решение о том, следует ли обеспечивать синхронизацию потоков с помощью объектов привилегированного режима. Как и во многих других случаях, ключом к успеху является выбор для работы правильного инструмента.

Использование сигнального и несигнального состояний

Объекты привилегированного режима могут, как правило, находиться в одном из двух состояний: сигнальное или несигнальное. Применение способа обозначения объекта с помощью сигнального и несигнального состояний можно рассматри-

вать как использование флажка, который поднимается, если объект сигнализирован, и опускается в противном случае. Метод обозначения объекта с помощью сигнального и несигнального состояний позволяет передавать другим потокам (в текущем процессе или вне его) извещения о том, что они могут приступить к выполнению определенной работы или что закончено выполнение некоторой задачи. Например, можно предусмотреть, чтобы фоновый поток обозначал как сигнальный некоторый объект, после того как этот поток завершит создание резервной копии файла, редактируемого в настоящее время в специализированной программе редактора, применяемой программистом. Фоновый поток сохраняет файл, а затем "поднимает флажок" (отмечая какой-то объект как сигнальный), чтобы передать потоку переднего плана информацию о том, что копирование выполнено.

Для синхронизации потоков и защиты ресурсов по принципу обозначения в качестве сигнальных и несигнальных могут использоваться такие объекты привилегированного режима, как события, мьютексы, таймеры ожидания и семафоры. Эти объекты, отдельно взятые, не нужны. Они предназначены исключительно для упрощения задачи защиты и управления ресурсами (особенно в многопотоковых приложениях) за счет того, что для них применяются различные способы обозначения сигнального и несигнального состояний. Сигнальными или несигнальными могут быть процессы, потоки, задания, события, семафоры, мьютексы, таймеры ожидания, файлы, результаты ввода с терминала и извещения об изменении файла.

Некоторые объекты привилегированного режима, такие как объекты событий, могут снова переустанавливаться в несигнальное состояние после того, как они были сигнальными, а к другим объектам такая операция неприменима. Например, ни объект процесса, ни объект потока нельзя перевести в несигнальное состояние после сигнального. Это связано с тем, что после обозначения как сигнального любого из указанных объектов выполнение такого объекта должно быть завершено, а возобновить выполнение завершенного процесса или потока невозможно — можно лишь создать новый процесс или поток.

Функции ожидания

В некоторых из приведенных выше примеров уже использовались функции ожидания подсистемы Win32, а в данном разделе приведено немного более подробное их описание. Функции ожидания позволяют любому потоку приостановить свое выполнение до тех пор, пока другой объект (или объекты) не станет сигнальным. Наиболее широко применяемой функцией ожидания Win32 является `WaitForSingleObject`. Эта функция принимает два параметра: дескриптор объекта, освобождения которого должна ожидать эта функция, и количество миллисекунд ожидания. Для перехода в состояние ожидания на неопределенно долгое время можно передать в эту функцию константу `INFINITE` (`0xFFFFFFFF`, или `-1`).

С другой стороны, функция `WaitForMultipleObjects`, как показывает само ее имя, позволяет обеспечить ожидание того, как будет сигнализировано несколько объектов (вплоть до 64). Вместо передачи в качестве первого параметра дескриптора одного объекта ей передается массив, содержащий параметры всех дескрипторов объектов, для которых нужно обеспечить ожидание освобождения. Функция `WaitForMultipleObjects` позволяет организовать ожидание того, что будут сиг-

нализированы все объекты или только один из них. Если возврат из функции WaitForMultipleObjects вызван одним объектом, то возвращаемое значение функции показывает, какой объект был сигнализирован. Это значение может находиться в пределах от WAIT_OBJECT_0 до WAIT_OBJECT_0 + NumberOfHandles - 1. Если же требуется снова вызвать эту функцию с тем же массивом дескрипторов, то вначале необходимо удалить сигнальный объект, так как в противном случае функция немедленно выполнит возврат, не переходя в состояние ожидания.

Если данная функция выходит по тайм-ауту в ходе ожидания освобождения некоторого объекта, она возвращает значение WAIT_TIMEOUT. Такое свойство функции можно использовать для реализации своего рода спин-блокировки, которая ожидает в течение короткого периода времени, выходит по тайм-ауту, выполняет определенную работу, а затем снова переходит к ожиданию освобождения желаемого ресурса. Это позволяет предотвратить ситуацию, при которой поток становится полностью непригодным для планирования на выполнение в то время, как он ожидает освобождения требуемого ресурса, а также дает возможность программисту использовать более тонкую степень детализации управления блокирующим поведением потоков.

Есть также несколько других функций ожидания (например, MsgWaitForSingleObject, MsgWaitForMultipleObjects, MsgWaitForMultipleObjectsEx, WaitForMultipleObjectsEx, WaitForSingleObjectEx, SignalObjectAndWait и т.д.), которые в этом разделе не рассматриваются. Подробные сведения об этих функциях приведены в справочнике комплекта Windows Platform SDK. Все перечисленные выше функции в основном представляют собой те или иные варианты функции WaitForSingleObject или WaitForMultipleObjects.

События

Объекты событий представляют собой именно то, о чем говорит их название — они свидетельствуют об изменениях. Поток переводит объект события в сигнальное состояние, чтобы другие потоки могли узнать о том, что произошло какое-то изменение. Объекты событий обычно используются для организации поэтапного выполнения работы. Например, один поток может выполнить первые несколько этапов некоторого задания, после чего отметить некоторое событие как сигнальное и сообщить тем самым другому потоку, что теперь именно он должен продолжить работу.

События можно отключать вручную или автоматически. Чтобы отметить некоторое событие как сигнальное, нужно вызвать функцию SetEvent, а чтобы отметить его как несигнальное — функцию ResetEvent. Сигнальное событие с автоматическим отключением становится несигнальным, как только один из потоков обнаруживает, что это событие стало сигнальным, и тем самым успешно завершает период ожидания, организованный с помощью этого события, а переустановка в неотмеченное состояние события с отключением вручную должна быть выполнена с помощью вызова функции ResetEvent. Если несколько потоков ожидают автоматически отключаемое событие, то после отметки этого события как сигнального на выполнение планируется только один из этих потоков,

а само событие автоматически становится несигнальным. А если несколько потоков ожидают, когда будет переведено в сигнальное состояние событие, отключаемое вручную, то после перевода этого события в сигнальное состояние могут быть запланированы на выполнение все ожидающие потоки.

Для создания нового события можно вызвать функцию `CreateEvent` API-интерфейса. Другие потоки могут получить доступ к этому событию, вызывая функции `CreateEvent`, `DuplicateHandle` или `OpenEvent`.

Таймеры ожидания

Таймеры ожидания представляют собой объекты, которые отмечают сами себя как сигнальные в указанное время или через регулярные интервалы. Таймер ожидания создается с помощью функции `CreateWaitableTimer`, а настройка его конфигурации выполняется с помощью функции `SetWaitableTimer`. Для таймера ожидания может устанавливаться абсолютное или относительное значение времени (для указания относительного времени в будущем используется отрицательное значение параметра `DueDate`) или устанавливается интервал, в котором предполагается осуществление таймером перехода в сигнальное состояние. После истечения интервала или значения времени таймер переходит в сигнальное состояние и в качестве необязательного действия ставит в очередь некоторую процедуру APC. Если таймер создан как предназначенный для отключения вручную, то остается сигнальным до тех пор, пока не будет снова вызвана функция `SetWaitableTimer`. Если же он создан как таймер с автоматическим отключением, то сразу же переходит в несигнальное состояние, как только для некоторого потока успешно завершается ожидание перехода этого таймера в сигнальное состояние.

Как показывает само название функции `CancelWaitableTimer`, она используется для отмены таймера ожидания. Но после того как становится сигнальным таймер с отключением вручную, его не нужно отменять; достаточно просто закрыть его дескриптор.

Как было сказано выше, с помощью таймера ожидания может быть выполнено необязательное действие: постановка в очередь некоторой процедуры APC. Но обычно не рекомендуется широко пользоваться этим средством, поскольку всегда можно просто подождать, пока данный таймер не станет сигнальным, после чего выполнить любой необходимый код. Решив использовать процедуру APC, разработчик должен учитывать, что бессмысленно применять поток для ожидания освобождения дескриптора таймера и одновременно с этим ожидать предупреждающего сообщения об освобождении таймера. Суть дела заключается в следующем — как только таймер становится сигнальным, поток активизируется (в результате чего поток выходит из состояния, в котором мог бы принимать предупреждающие сообщения), а это приводит к тому, что процедура APC так и не будет вызвана.

Разработчики, создавшие много приложений для операционной системы Windows, обычно знакомы с объектом таймера непривилегированного режима Windows. Его не следует путать с таймером ожидания привилегированного режима. Самое важное различие между ними состоит в том, что при работе с таймерами непривилегированного режима необходимо предусмотреть в приложении средства пользовательского интерфейса, поэтому применение таких таймеров связано с довольно значи-

тельным потреблением ресурсов. Кроме того, как и другие рассматриваемые здесь объекты привилегированного режима, таймеры ожидания могут совместно использоваться многочисленными потоками и могут быть защищены.

Объекты таймера непривилегированного режима Windows вырабатывают сообщения `WM_TIMER`, возвращаемые либо в поток, в котором вызвана функция `SetTimer`, либо в поток, в котором создано текущее окно. Это означает, что по истечении установки таймера извещение об этом передается только в один поток. В отличие от этого, таймеры ожидания могут становиться сигнальными одновременно для нескольких потоков, которые могут даже принадлежать к разным процессам.

Решение о том, следует ли использовать объект таймера ожидания или таймер непривилегированного режима, должно быть основано на анализе того, насколько большой объем манипуляций в пользовательском интерфейсе выполняется в ответ на сигнал, поступающий от таймера. Если объем этих манипуляций значителен, то наилучший вариант состоит в использовании таймера непривилегированного режима, поскольку в противном случае, при применении таймера ожидания, приходится ожидать появления и сообщений от объекта этого таймера, и сообщений от окна. Если же в конечном итоге в приложении с графическим интерфейсом пользователя необходимо использовать таймер ожидания и требуется поток, который ожидает сигнала от таймера, чтобы отвечать на сообщения в ходе этого ожидания, можно применить функцию `MsgWaitForMultipleObjects`, чтобы одновременно ожидать и сигналов от объекта, и сообщений.

Семафоры

Как правило, объект семафора привилегированного режима используется для ограничения количества потоков, которые могут одновременно получать доступ к какому-то ресурсу. Мьютекс по определению позволяет предоставлять доступ к защищенному ресурсу одновременно только одному потоку, а семафор может применяться и для обеспечения одновременного доступа к ресурсу многочисленных потоков, и для регламентации максимального количества потоков, одновременно осуществляющих доступ. Максимальное количество потоков, одновременно осуществляющих доступ, задается при создании семафора, а операционная система Windows гарантирует соблюдение указанного максимального предела.

При первоначальном создании объекта семафора задается не только максимальное значение для семафора, но и его начальное значение. При условии, что это значение остается больше 0, семафор является сигнальным, и любой поток, предпринимающий попытку обратиться к ресурсу с его помощью, немедленно выполняет возврат, уменьшая при этом значение семафора. Допустим, в частности, что необходимо обеспечить одновременный доступ к конкретному ресурсу для потоков, количество которых не должно превышать пяти (а сами потоки принадлежат к пулу потоков, включающему десять потоков). Для этого создается семафор с максимальным значением 5, после чего каждый поток, в котором требуется доступ к ресурсу, вызывает одну из функций ожидания, позволяющую воспользоваться семафором. В первых пяти потоках будет немедленно выполнен возврат из функции ожидания, и при каждой очередной успешной попытке вызвать на выполнение

функцию ожидания значение семафора будет уменьшаться на 1. А если попытку воспользоваться семафором предпримет шестой поток, то будет заблокирован до тех пор, пока один из первых пяти потоков не освободит семафор. Если после этого, скажем, в потоке *Thread 5* будет вызвана функция `ReleaseSemaphore`, то поток *Thread 6* немедленно возвратится из состояния ожидания. Между тем, количество потоков, одновременно получающих доступ к ресурсу, никогда не превысит пяти.

Мьютексы

Мьютексы относятся к числу наиболее полезных и широко используемых объектов привилегированного режима Windows. Они имеют много важных практических назначений (начиная от упорядочения доступа к критическим ресурсам и заканчивая обеспечением потокобезопасности кода) и часто встречаются в несметном количестве в сложных многопотоковых приложениях Windows.

Мьютексы весьма напоминают критические секции, разумеется, за исключением того, что они представляют собой объекты привилегированного режима. Это означает, что доступ к мьютексам может потребовать больше времени, но в целом эти объекты обладают более широкими функциональными возможностями, чем критические секции, поскольку могут совместно использоваться различными процессами, а для доступа к ресурсам с помощью мьютексов могут применяться функции ожидания с заданным тайм-аутом.

Мьютексы обладают одним отличительным свойством по сравнению с другими объектами привилегированного режима, которое заключается в том, что мьютексы поддерживают понятие принадлежности к потоку. В каждом объекте мьютекса привилегированного режима имеется поле, содержащее идентификатор потока, которому принадлежит этот объект. Если идентификатор потока равен 0, то мьютекс никому не принадлежит и является сигнальным. Если же идентификатор потока отличен от 0, это означает, что мьютекс принадлежит некоторому потоку и является несигнальным.

В отличие от других объектов привилегированного режима, мьютекс допускает, чтобы любой поток мог обращаться к ресурсу с его помощью несколько раз, не освобождая сам мьютекс и не ожидая его освобождения. Каждый раз, когда поток успешно получает доступ к ресурсу с помощью мьютекса, он становится владельцем мьютекса (идентификатор этого потока сохраняется в объекте мьютекса привилегированного режима), а счетчик рекурсии данного объекта увеличивается. Это означает, что необходимо выполнить операцию освобождения мьютекса (с помощью функции `ReleaseMutex`) столько же раз, сколько было выполнено с его помощью операций доступа, чтобы мьютекс стал сигнальным, и другие потоки получили возможность стать его владельцем.

Операционная система Windows контролирует, чтобы мьютекс не оставался заброшенным из-за аварийного завершения владеющего им потока, что может привести к блокировке других потоков на неопределенно долгое время. Если поток, владеющий мьютексом, завершает свою работу, не освобождая его, то операционная система Windows автоматически переустанавливает идентификатор потока и счетчик рекурсии в объекте мьютекса в 0 (а это приводит к тому, что объект мьютекса отмечается как сигнальный). Если другой поток ожидает получения доступа

к ресурсу с помощью данного мьютекса, то система передает ему рассматриваемый мьютекс во владение, установив идентификатор потока мьютекса таким образом, чтобы он ссылался на ожидающий поток, и присвоив его счетчику рекурсии значение 0. Сама же функция ожидания возвращает значение WAIT_ABANDONED, чтобы ожидающий поток мог определить, благодаря чему он стал владельцем этого мьютекса.

Как было сказано выше, приостановка или аварийное завершение потока может стать причиной того, что мьютекс останется во владении этого потока на неопределенно долгое время или даже окажется заброшенным. Поэтому всегда лучше предусмотреть по возможности нормальный возврат из функции точки входа потока.

Порты завершения ввода-вывода

Порт завершения ввода-вывода позволяет синхронизировать многочисленные потоки, выполняющие асинхронные операции ввода-вывода с помощью одного объекта. Дескрипторы файлов можно связывать с портом завершения ввода-вывода с помощью функции CreateIOCompletionPort API-интерфейса Win32. После завершения асинхронной операции ввода-вывода, начатой в файле, связанном с портом завершения ввода-вывода, в очередь этого порта передается пакет завершения ввода-вывода.

В потоке можно обеспечить доступ к ресурсу с помощью порта завершения ввода-вывода, вызвав функцию GetQueuedCompletionStatus. Если пакет завершения ввода-вывода не готов, поток переходит в состояние ожидания. Если же в очередь порта завершения ввода-вывода направлен пакет, указанная функция немедленно выполняет возврат.

Следует отметить, что с помощью портов завершения ввода-вывода можно синхронизировать не только ввод-вывод, но и другие операции. Используя функцию PostQueuedCompletionStatus API-интерфейса в сочетании с функцией GetQueuedCompletionStatus, можно создать многопотоковый механизм обработки сигналов, более масштабируемый по сравнению с вариантом, в котором используются функции SetEvent/WaitForMultipleObjects. Это связано с тем фактом, что возможности функции WaitForMultipleObjects по обеспечению доступа к ресурсу ограничиваются максимальным количеством рабочих потоков, равным 64. А с применением порта завершения ввода-вывода можно создать систему синхронизации, позволяющую обеспечить доступ к ресурсу для любого количества потоков, которое может быть создано в процессе. Ниже приведен пример того, как можно реализовать механизм, способный обеспечить доступ к ресурсу больше чем для 64 потоков одновременно.

1. В главном потоке процесса создается порт завершения ввода-вывода, который не связан с конкретным файлом (или файлами), поскольку в функцию CreateIOCompletionPort в качестве параметра FileHandle передано значение NULL.
2. Главным потоком создается такое количество потоков, которое требуется для рассматриваемого процесса; предположим, что работа процесса начинается с создания пула из 100 рабочих потоков.

3. После создания всех потоков в главном потоке вызывается функция `GetQueuedCompletionStatus` для обеспечения доступа к ресурсу с помощью порта завершения ввода-вывода.
4. После того как очередной рабочий поток завершает свою работу, в связи с чем возникает необходимость передать сигнал главному потоку, поток вызывает функцию `PostQueuedCompletionStatus`, чтобы отправить пакет завершения ввода-вывода в порт. Например, поток может сделать это до возврата из функции точки входа или до перехода в режим ожидания, допустим, глобального события, связанного с главным потоком. Для того чтобы в главный поток поступила информация о том, какой рабочий поток завершил свою работу, этот рабочий поток может передать в функцию `PostQueuedCompletionStatus` свой идентификатор потока.
5. Главный поток выполняет возврат из вызова функции `GetQueued-CompletionStatus` после обнаружения данного пакета.
6. Поскольку в главном потоке есть информация о том, сколько создано рабочих потоков, в нем снова вызывается функция `GetQueuedCompletionStatus`, и такая операция выполняется в цикле до тех пор, пока все рабочие потоки не сообщат, что они закончили свою работу. Поскольку не нужно больше завершать работу потока, чтобы перевести его в сигнальное состояние, а также поскольку в главном потоке (или в любом другом потоке) можно обеспечить доступ к ресурсу с помощью такого количества потоков, какое потребуется в конкретном приложении, порт завершения ввода-вывода может стать удобной масштабируемой альтернативой по сравнению с подходом, в котором доступ к ресурсу для многочисленных потоков обеспечивается с помощью вызова функции `WaitForMultipleObjects` или аналогичной функции API-интерфейса.

Упражнения

В следующем упражнении проводятся эксперименты с приложением, в котором намеренно нарушены требования потокобезопасности. Прежде всего это упражнение позволяет показать, что происходит, если в приложении не синхронизируется доступ к совместно используемым ресурсам. Это позволяет лучше оценить тот огромный объем работы, который был проделан для того, чтобы в программе SQL Server обеспечивался доступ рабочих потоков к совместно используемым ресурсам с помощью таких методов, которые являются одновременно и быстрыми, и безопасными.

В последнем упражнении данной главы показано, как реализовать спин-блокировку, основанную на основе объекта мьютекса привилегированного режима. В программе SQL Server спин-блокировки используются для защиты доступа к совместно используемым ресурсам внутри сервера; понимание того, как они работают, позволяет получить более полное представление о функционировании программы SQL Server.

Упражнение 3.9. Последствия нарушения синхронизации потоков

В приведенном ниже приложении C++ демонстрируются три метода доступа к совместно используемому ресурсу (в данном случае к глобальной переменной) из многочисленных потоков. Код этого приложения можно найти в подкаталоге

CH03\thread_sync на компакт-диске, прилагаемом к данной книге. Загрузите это приложение в среду разработки Visual C++, затем откомпилируйте и вызовите на выполнение, чтобы проработать данное упражнение.

Приложение создает 50 потоков, которые проверяют значение глобальной переменной и, если полученное значение меньше 50, увеличивает его. Конечным результатом выполнения таких действий должно стать наличие значения 50 в глобальной переменной после завершения работы всех потоков, но такое происходит не всегда. Рассмотрим код данного приложения (листинг 3.9).

Листинг 3.9. Применение средств синхронизации потоков

```
// thread_sync.cpp. Определяет точку входа для терминального приложения
//

#include "stdafx.h"
#include "windows.h"

#define MAXTHREADS 50
//#define THREADSAFE
//#define CRITSEC
//#define MUTEX

long g_ifoo=0;
HANDLE g_hStartEvent;

#ifdef CRITSEC
CRITICAL_SECTION g_cs;
#endif

#ifdef MUTEX
HANDLE hMutex;
#endif

DWORD WINAPI StartThrd(LPVOID lpParameter)
{
  WaitForSingleObject(g_hStartEvent, INFINITE);
#ifdef CRITSEC
  EnterCriticalSection(&g_cs);
#endif
#ifdef MUTEX
  WaitForSingleObject(hMutex,INFINITE);
#endif
#ifdef THREADSAFE
  if (g_ifoo<50) InterlockedIncrement(&g_ifoo);
#else
  if (g_ifoo<50) g_ifoo++;
#endif
#ifdef CRITSEC
  LeaveCriticalSection(&g_cs);
#endif
#ifdef MUTEX
  ReleaseMutex(hMutex);
#endif
  return 0;
}
```

```
int main(int argc, char* argv[])
{
  DWORD dwThreadID;
  HANDLE hThreads[MAXTHREADS];
#ifdef CRITSEC
  InitializeCriticalSection(&g_cs);
#endif
#ifdef MUTEX
  hMutex=CreateMutex(NULL,FALSE,NULL);
#endif
  g_hStartEvent=CreateEvent(NULL,true,false,NULL);
  for (int i=0; i<MAXTHREADS; i++) {
    hThreads[i]=CreateThread(NULL,
    0,
    (LPTHREAD_START_ROUTINE)StartThrd,
    0,
    0,
    (LPDWORD)&dwThreadID);
  };
  SetEvent(g_hStartEvent);
  WaitForMultipleObjects(i,hThreads,true,INFINITE);

  printf("g_ifoo=%d\n",g_ifoo);
#ifdef CRITSEC
  DeleteCriticalSection(&g_cs);
#endif
#ifdef MUTEX
  CloseHandle(hMutex);
#endif
  return 0;
}
```

Для контроля над тем, как синхронизируется в программе доступ к глобальной переменной (и синхронизируется ли он вообще), применяются три константы `#define`. По умолчанию константы `THREADSAFE`, `CRITSEC` и `MUTEX` не определены, поэтому доступ к глобальной переменной не синхронизирован. После вызова на выполнение этой программы на многопроцессорном компьютере достаточное количество раз можно в конечном итоге обнаружить такую ситуацию, в которой переменная `g_ifoo` не принимает окончательного значения, равного 50. Это связано с тем, что доступ к этой переменной не был синхронизирован и возникали перекрытия между теми интервалами времени, когда один поток осуществлял выборку значения, а другой — его наращивание, как показано в сценарии, приведенном в табл. 3.17.

Таблица 3.17. Пример несинхронизированного доступа к ресурсу со стороны нескольких потоков

Шаг	Действие
1	Поток `Thread10`. Проверить условие `g_ifoo < 50`. Результат — положительный
2	Поток `Thread11`. Проверить условие `g_ifoo < 50`. Результат — положительный
3	Поток `Thread10`. Получить значение переменной `g_ifoo`. В настоящее время оно равно 10

Окончание табл. 3.17

Шаг	Действие
4	Поток `Thread11`. Получить значение переменной `g_ifoo`. В настоящее время оно равно 10
5	Поток `Thread10`. Увеличить его (до 11)
6	Поток `Thread11`. Увеличить его (до 11)
7	Поток `Thread10`. Присвоить новое значение переменной `g_ifoo`
8	Поток `Thread11`. Присвоить новое значение переменной `g_ifoo`

Из-за такого перекрытия два потока присваивают переменной `g_ifoo` одинаковое значение, поэтому окончательное значение `g_ifoo` остается меньше 50, тогда как должно быть равным 50, т.е. количеству рабочих потоков.

Если затем будет раскомментирована строка `//#define THREADSAFE` и выполнена повторная компиляция приложения, то указанное перекрытие должно быть исключено. Это связано с тем, что в коде приложения начинает применяться функция `InterlockedIncrement` для обеспечения неразрывности операции приращения. Применительно к сценарию, показанному в табл. 3.17, это означает, что шаги 3, 5 и 7 выполняются в виде одной операции, как и шаги 4, 6 и 8. Поток `Thread10` завершает свою операцию приращения до того, как поток `Thread11` получает разрешение сделать то же самое; таким образом, поток `Thread11` в качестве текущего значения `g_ifoo`, к которому он применяет операцию приращения, обнаруживает 11, а не 10.

На следующем этапе выполнения данного упражнения можно закомментировать строку `#define`, касающуюся константы `THREADSAFE`, и раскомментировать строку, касающуюся константы `CRITSEC`. После этого доступ к глобальной переменной синхронизируется с помощью критической секции.

Для использования последнего из рассматриваемых методов многопотоковой синхронизации можно закомментировать строку с константой `CRITSEC` и раскомментировать строку с константой `MUTEX`. После этого в коде будет использоваться объект мьютекса привилегированного режима для упорядочения доступа к глобальной переменной.

Проведите эксперименты со всеми четырьмя методами и изучите полученные вами результаты. Вообще говоря, при создании приложений выбор среди основных методов синхронизации следует осуществлять в том порядке, в каком они представлены в данном разделе. В частности, если в приложении не требуется синхронизация потоков, то не нужно предусматривать код, предназначенный для этой цели. Если же такая потребность возникает, попробуйте вначале воспользоваться функциями комплексной блокировки. Если они не соответствуют вашим требованиям, то, возможно, подойдет способ с использованием критической секции. А если нельзя применить и критическую секцию (возможно, потому, что нужно предусмотреть ожидание по тайм-ауту или требуется синхронизация нескольких процессов), то перейдите к использованию объекта привилегированного режима, такого как мьютекс.

Упражнение 3.10. Реализация спин-блокировки привилегированного режима с помощью мьютекса

Выше в данной главе был приведен пример того, как обычно в приложениях реализуется спин-блокировка, — конструкция непривилегированного режима, в которой используется одна из функций комплексной блокировки для обеспечения неразрывного

доступа к переменной блокировки. Кроме того, могут быть созданы спин-блокировки, которые основаны на применении объектов привилегированного режима. Такой подход на первый взгляд может показаться странным, но одним из вполне оправданных областей использования спин-блокировок привилегированного режима является выполнение другого кода в потоке, в то время как происходит ожидание освобождения какого-то объекта привилегированного режима. При этом, по сути, код спин-блокировки разрабатывается таким образом, чтобы в спин-блокировке устанавливался тайм-аут на довольно короткий интервал, выполнялся любой код, который требуется выполнить в период ожидания, после чего происходил возврат в цикл ожидания. В результате этого поток в период ожидания доступа к ресурсу остается частично занятым выполнением работы, а такой вариант может оказаться более предпочтительным по сравнению с тем, чтобы просто перевести поток в состояние ожидания до тех пор, пока ресурс не станет доступным.

Пример спин-блокировки, основанной на использовании объекта привилегированного режима, приведен ниже. Код этой программы можно найти в подкаталоге `CH03\kernel_spinlock` на компакт-диске, прилагаемом к данной книге. Загрузите эту программу в среду разработки Visual C++, затем откомпилируйте и вызовите на выполнение. Код программы приведен в листинге 3.10.

Листинг 3.10. Реализация спин-блокировки на основе объекта привилегированного режима

```
// kernel_spinlock.cpp. Определяет точку входа для
//                      терминального приложения
//

#include "stdafx.h"
#include "windows.h"

#define MAXTHREADS 2
#define SPINWAIT 1000

HANDLE g_hWorkEvent;

class CSpinLock {
public:
  static void GetLock(HANDLE hEvent);
};

void CSpinLock::GetLock(HANDLE hEvent) {
  int i=0;
  while (WAIT_TIMEOUT==WaitForSingleObject(hEvent, SPINWAIT)) {
printf("Spinning count=%d for thread 0x%08x\n",++i,
       GetCurrentThreadId());
    // Ввести здесь другой код, который будет выполняться, пока
    // происходит ожидание освобождения ресурсов
  }
}

DWORD WINAPI StartThrd(LPVOID lpParameter)
{
  printf("Inside thread function for thread 0x%08x\n",
       GetCurrentThreadId());
  CSpinLock::GetLock(g_hWorkEvent);
  printf("Acquired spinlock for thread 0x%08x\n",
```

```
      GetCurrentThreadId());
  Sleep(5000);
  SetEvent(g_hWorkEvent);
  return 0;
}

int main(int argc, char* argv[])
{
  DWORD dwThreadID;
  HANDLE hThreads[MAXTHREADS];

  g_hWorkEvent=CreateEvent(NULL,false,true,NULL);

  for (int i=0; i<MAXTHREADS; i++) {
    hThreads[i]=CreateThread(NULL,
     0,
    (LPTHREAD_START_ROUTINE)StartThrd,
     0,
     0,
    (LPDWORD)&dwThreadID);
  };
  WaitForMultipleObjects(i,hThreads,true,INFINITE);

  return 0;
}
```

В данном примере спин-блокировка реализована в своем собственном классе, а доступ к ней предоставлен с помощью статического метода. (Статический метод позволяет избежать необходимости создавать экземпляр класса `CSpinLock` для того, чтобы его можно было использовать в программе.) В спин-блокировку можно передать дескриптор любого объекта привилегированного режима; в этом примере используется объект события.

В главной функции создаются два потока, которые приобретают спин-блокировку, переходят в состояние ожидания на пять секунд, затем освобождают спин-блокировку, отметив как сигнальное данное событие. Поскольку в любое время приобрести спин-блокировку может только один из потоков, второй поток должен перейти в состояние ожидания завершения работы первого потока, устанавливая период ожидания, равный одной секунде, столько раз, сколько потребуется, чтобы дождаться освобождения спин-блокировки (т.е. дождаться того момента, когда событие станет сигнальным).

Применяемое событие, представляющее собой событие с автоматическим отключением, немедленно переустанавливается в несигнальное состояние, как только ожидание освобождения этого события каким-то потоком завершается успешно. Поэтому после того, как состояние ожидания освобождения объекта события, в которое переходит первый поток с помощью функции `GetLock`, завершается успешно, это событие немедленно переустанавливается в несигнальное состояние, а второй поток должен ожидать того, что первый поток передаст ему сигнал, и только после этого приступать к дальнейшему выполнению. С учетом того, что в данном примере функция потока переходит в состояние ожидания на пять секунд, такая ситуация может возникнуть не раньше, чем через пять секунд после того, как первый поток приобретет спин-блокировку.

Откомпилируйте и вызовите на выполнение этот код, после чего проведите эксперименты с разными значениями интервалов времени ожидания для функции потока и с разными количествами потоков. После вызова на выполнение этого приложения должны быть получены примерно такие результаты, как показано ниже.

```
Inside thread function for thread 0x00000d00
Acquired spinlock for thread 0x00000d00
Inside thread function for thread 0x00000f20
Spinning count=1 for thread 0x00000f20
Spinning count=2 for thread 0x00000f20
Spinning count=3 for thread 0x00000f20
Spinning count=4 for thread 0x00000f20
Spinning count=5 for thread 0x00000f20
Acquired spinlock for thread 0x00000f20
```

Синхронизация потоков. Резюме

В операционной системе Windows предусмотрен богатый набор функций синхронизации потоков. Средства синхронизации потоков подразделяются на две основные разновидности: относящиеся к непривилегированному и привилегированному режиму. Как средства синхронизации непривилегированного режима обычно рассматриваются спин-блокировки, критические секции и функции комплексной блокировки. К средствам синхронизации привилегированного режима относятся такие объекты привилегированного режима, как мьютексы, семафоры, события, потоки, процессы и таймеры ожидания.

В программе SQL Server используются средства синхронизации обоих типов. Компонент UMS этой программы затрачивает довольно большой объем времени на ожидание доступа к синхронизационным объектам привилегированного режима, но в нем также реализовано множество разновидностей спин-блокировок и сделано все возможное для предотвращения переключения любого потока в привилегированный режим без абсолютной необходимости.

Ключом к обеспечению успешной синхронизации потоков является получение гарантий неразрывного доступа к ресурсам. Создание предпосылок одновременной модификации одного и того же ресурса со стороны нескольких потоков неизбежно ведет к нарушению работы. Такой исход событий позволяют предотвратить эффективные средства синхронизации потоков.

Синхронизация потоков. Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Одной из наиболее важных составляющих синхронизации потоков является обеспечение неразрывного доступа к ресурсам.
2. Приведите два примера синхронизационных объектов или конструкций непривилегированного режима.
3. Что происходит после того, как в потоке успешно завершается ожидание перехода в сигнальное состояние события с автоматическим отключением?
4. Что происходит после того, как в потоке успешно завершается ожидание перехода в сигнальное состояние семафора?

5. Что происходит после того, как в потоке успешно завершается ожидание перехода в сигнальное состояние мьютекса?
6. Назовите единственный объект привилегированного режима, который поддерживает понятие "владеющего им потока"?
7. Какой синхронизационный объект следует использовать, если требуется защитить одну из процедур (библиотеки DLL), которая может совместно использоваться несколькими процессами, чтобы она не могла быть вызвана на выполнение больше чем из одного процесса в один и тот же момент времени?
8. Следует ли использовать для создания оконного графического интерфейса пользователя (Graphical User Interface – GUI) объект таймера ожидания или таймер непривилегированного режима Windows, если требуется модифицировать графический интерфейс пользователя через некоторые регулярные интервалы времени?
9. Какая функция API-интерфейса используется для задания частоты перехода объекта таймера ожидания в сигнальное состояние?
10. Подтвердите или опровергните следующее утверждение. Спин-блокировка – это объект привилегированного режима, который выполняет итерации (повторяет свои действия в цикле) до тех пор, пока не установит блокировку на некотором ресурсе.
11. Объясните, для чего предназначена функция `InterlockedExchange` API-интерфейса.
12. Подтвердите или опровергните следующее утверждение. Организовывая доступ к ресурсу с помощью объекта критической секции, нельзя задать значение тайм-аута.
13. Какие действия предпринимает система после завершения работы потока, владеющего каким-то мьютексом?
14. Каково максимальное число объектов, доступ к которым может быть организован одновременно с помощью функции `WaitForMultipleObjects`?
15. Какое рода сообщение вырабатывает объект таймера непривилегированного режима Windows?
16. Подтвердите или опровергните следующее утверждение. Следует избегать использования методов и элементов проекта, которые предусматривают проведение непрерывного опроса для определения доступности ресурса.
17. Подтвердите или опровергните следующее утверждение. Операционная система Windows обнаруживает взаимоблокировки потоков, выбирает один из участвующих в них потоков в качестве жертвы взаимоблокировки и принудительно завершает его выполнение.
18. Какая функция API-интерфейса используется в потоке для приобретения объекта критической секции?
19. Какая функция API-интерфейса используется в потоке для освобождения объекта критической секции?

20. Назовите механизм, описанный в настоящей главе, который позволяет обеспечить ожидание перехода в сигнальное состояние большего количества объектов по сравнению с максимальным количеством, поддерживаемым функцией `WaitForMultipleObjects`.
21. Подтвердите или опровергните следующее утверждение. Спин-блокировка вообще не потребляет ресурсов процессора в то время, как с ее помощью осуществляется ожидание освобождения ресурса.
22. Подтвердите или опровергните следующее утверждение. Объект процесса — единственный тип объекта привилегированного режима, который не может быть переведен в сигнальное состояние.
23. Подтвердите или опровергните следующее утверждение. Порядок, в котором происходит доступ к ресурсам привилегированного режима, не может стать причиной возникновения взаимоблокировки потоков, поскольку потоки не могут захватывать ресурсы привилегированного режима.
24. Подтвердите или опровергните следующее утверждение. Синхронизация потоков с использованием объектов привилегированного режима обычно происходит быстрее по сравнению с их синхронизацией с помощью объектов непривилегированного режима.
25. В какое состояние, сигнальное или несигнальное, переходит семафор после того, как его значение достигает 0?

ГЛАВА 4

Основные принципы организации памяти

Понимание архитектуры памяти операционной системы — это, по-видимому, одна из наиболее важных предпосылок для понимания функционирования самой операционной системы. Как и во всех других операционных системах, в ОС Windows применяются собственные методы управления ресурсами памяти и предоставления приложениям доступа к службам, относящимся к памяти. В этой главе приведено подробное описание того, как в ОС Windows организовано управление памятью и как обычно в приложениях используются средства управления памятью Windows. В ней также рассматриваются различные типы памяти Windows: виртуальная память, динамические области памяти и совместно используемая память.

Основы организации памяти

Средства доступа к памяти являются такой неотъемлемой частью архитектуры приложения и важным фактором обеспечения высокой производительности, что для управления памятью и предоставления доступа к ней приложениям посвящена значительная часть инфраструктуры Windows. Эффективное управление памятью является ключом к достижению таких показателей производительности приложения, которые были бы и приемлемыми, и устойчивыми. Несмотря на то что современные модули оперативной памяти отличаются сравнительно низкой ценой, память все еще остается ограниченным ресурсом, а также одним из наиболее важных факторов, от которых зависит производительность приложения и общая пропускная способность системы. Во многих случаях при установке на компьютере дополнительного объема оперативной памяти производительность повышается значительнее, чем в результате модернизации компьютера за счет установки более быстродействующего процессора.

Основы организации памяти. Основные термины и определения

- **Пространство адресов процесса.** Пространство адресов для приложения, имеющее объем 4 Гбайт. Объем адресуемой памяти в приложениях Win32 ограничивается величиной 4 Гбайт, поскольку 4 294 967 296 (2^{32}) — это наибольшее целочисленное значение, которое может храниться в 32-битовом

указателе. Из этих 4 Гбайт памяти 2 зарезервировано по умолчанию для доступа в привилегированном, а 2 — в непривилегированном режиме. В некоторых версиях Windows пространство адресов непривилегированного режима может быть увеличено до 3 Гбайт (за счет пространства привилегированного режима) с использованием параметра `/3GB` файла `BOOT.INI`. Этот вариант может применяться для приложений, настройка конфигурации которых была выполнена таким образом, чтобы они могли этим воспользоваться. Вся память, распределяемая приложением, берется из этого пространства.

- **Виртуальная память.** Средство, с помощью которого диспетчер памяти предоставляет приложению больше памяти, чем физически существует на компьютере. Диспетчер виртуальной памяти Windows позволяет разработчикам приложений действовать так, как если бы на компьютере было 4 Гбайт памяти, независимо от того, какой объем физической памяти фактически установлен на компьютере. Виртуальная память Windows реализуется в основном с использованием системного файла подкачки.
- **Размер страницы.** Размер страницы памяти, который требуется для архитектуры используемого процессора. Для процессора x86 он равен 4 Кбайт. Во всех операциях распределения памяти Windows должны использоваться величины, кратные размеру страницы системы.
- **Степень детализации распределения.** Границы, по которым должны выполняться операции резервирования виртуальной памяти под управлением ОС Windows. Во всех современных версиях Windows степень детализации распределения равна 64 Кбайт, поэтому операции резервирования виртуальной памяти непривилегированного режима должны выполняться на границах секций с объемом 64 Кбайт в пространстве адресов процесса.
- **Системный файл подкачки.** Файл (или файлы), используемый в операционной системе Windows для предоставления реальной памяти, соответствующей виртуальной памяти. В ОС Windows файл подкачки применяется для обмена страницами физической памяти между оперативной памятью и жестким диском в форме, прозрачной для приложения. На каждом конкретном компьютере общий объем хранения физической памяти равен сумме объема физической памяти и объемов всех файлов подкачки, вместе взятых.
- **Преобразование адресов.** Операция преобразования адресов виртуальной памяти в адреса физической памяти.
- **Ситуация отсутствия страницы.** Условие, активизируемое модулем управления памятью (Memory Management Unit — MMU) процессора, которое вынуждает соответствующую процедуру обработки ошибки Windows загрузить страницу из системного файла подкачки в физическую память, если ее удается найти.
- **Пробуксовка.** Условие, которое возникает, если система обнаруживает нехватку физической памяти и в связи с этим непрерывно осуществляет обмен страницами между физической памятью и системным файлом подкачки, что обычно приводит к существенному замедлению работы приложений.

- **Секция присваивания NULL-указателя.** Первые 64 Кбайт пространства адресов непривилегированного режима; эта секция обозначается как неприменимая в операциях распределения памяти для того, чтобы можно было проще обнаруживать ссылки на NULL-указатель.
- **Приложение, приспособленное для работы в большом пространстве адресов.** Приложение, в заголовке исполняемого файла которого установлен флажок `IMAGE_FILE_LARGE_ADDRESS_AWARE`. Приложение, приспособленное для работы в большом пространстве адресов, получает пространство адресов непривилегированного режима с объемом 3 Гбайт, если эксплуатируется в подходящей для этого версии Windows, которая была загружена с опцией `/3GB`.
- **AWE (Address Windowing Extensions — адресные расширения для работы с окнами).** Средство, предусмотренное в ОС Windows для адресации физической памяти, превышающей по объему 4 Гбайт.
- **Настройка памяти приложения.** Средство, с помощью которого приложение, приспособленное для работы в большом пространстве адресов, может использовать вплоть до 3 Гбайт пространства адресов процесса.

Основы организации памяти. Основные функции API-интерфейса Win32

Основные функции API-интерфейса Win32, связанные с памятью, приведены в табл. 4.1.

Таблица 4.1. Основные функции API-интерфейса Win32, связанные с памятью

Функция	Описание
`GetSystemInfo`	Получить информацию системного уровня о таких ресурсах компьютера, как процессоры и память
`VirtualAlloc`	Зарезервировать, закрепить за запоминающим устройством и освободить виртуальную память
`AllocateUserPhysicalPages`	Распределить физическую память для использования со средствами AWE операционной системы Windows
`MapUserPhysicalPages`	Отобразить часть физической памяти AWE на буфер виртуальной памяти, выделенный с помощью функции `VirtualAlloc`
`ReadProcessMemory`	Разрешить одному процессу выполнять операции чтения в памяти, принадлежащей другому процессу
`WriteProcessMemory`	Разрешить одному процессу выполнять операции записи в памяти, принадлежащей другому процессу

Основы организации памяти. Основные инструментальные средства

Наилучшим универсальным инструментальным средством для текущего контроля над статистическими данными об использовании памяти и производительности Windows является программа Perfmon. Удивительно полезной оказалась также программа Task Manager. Следует учитывать, что в столбце `Mem Usage` программы Task Manager указан размер рабочего набора каждого процесса, а не показатель общего использования виртуальной памяти этим процессом. Поскольку упомянутый столбец включает данные о совместно используемых страницах, содержащиеся в нем цифры нельзя просуммировать, чтобы определить общий объем физической памяти, применяемой всеми процессами. Кроме того, в столбце `VM Size` программы Task Manager фактически указан объем закрытой памяти процесса в байтах (количество закрытых зафиксированных страниц процесса), а не суммарный объем виртуальной памяти процесса. Основные инструментальные средства текущего контроля над использованием памяти перечислены в табл. 4.2.

Таблица 4.2. Основные инструментальные средства текущего контроля над использованием памяти

	Зарезервированная виртуальная память	Объем файла подкачки	Количество ситуаций отсутствия страницы	Размер рабочего набора	Нерезидентный пул	Резидентный пул
Perfmon	+	+	+	+	+	+
Pstat	+	+	+	+	+	+
Pview	+		+	+	+	+
pmon	+		+		+	+
TaskMgr	+		+	+	+	+
TList	+			+		

Основные счетчики программы Perfmon

Основные счетчики программы Perfmon, относящиеся к памяти, приведены в табл. 4.3.

Таблица 4.3. Основные счетчики программы Perfmon, относящиеся к памяти

Счетчик	Описание
`Memory:Committed Bytes`	Переданное закрытое пространство адресов (и в файле подкачки, и в физической памяти)
`Memory:Commit Limit`	Объем памяти, который можно закрепить за системным файлом подкачки, не вынуждая систему увеличивать его объем

Окончание табл. 4.3

Счетчик	Описание
`Memory:% Committed Bytes In Use`	Значение `Memory:Committed Bytes`, деленное на значение `Memory:Commit Limit`
`Process:Virtual Bytes`	Суммарный размер пространства адресов процесса (совместно используемые и закрытые страницы)
`Process:Private Bytes`	Размер закрепленного пространства адресов, отличного от совместно используемого
`Process:Page File Bytes`	То же, что и счетчик `Process:Private Bytes`
`Process:Page File Peak`	Пиковое значение счетчика `Process:Page File Bytes`

Адреса

Поскольку Windows — это 32-битовая операционная система, всем процессам непривилегированного режима предоставляется линейное пространство адресов с объемом 4 Гбайт. Объем этого пространства ограничен величиной 4 Гбайт, поскольку 32-битовый указатель может принимать одно из 4 294 967 296 (2^{32}) значений. Таким образом, значения указателя в приложениях Windows могут находиться в пределах от `0x00000000` до `0xFFFFFFFF`.

В 64-битовых версиях Windows процессам предоставляется линейное пространство адресов с объемом 16 Эбайт (эксабайтов), поскольку 64-битовый указатель может принимать одно из 18 446 744 073 709 551 616 (2^{64}) значений в пределах от `0x0000000000000000` до `0xFFFFFFFFFFFFFFFF`.

Тот факт, что в 32-битовых версиях Windows процессам непривилегированного режима предоставляется пространство адресов, ограниченное объемом 4 Гбайт, не означает, что приложения не могут получать доступ больше чем к 4 Гбайт физической памяти. Как может быть известно читателю, на серверных компьютерах нередко бывает установлена оперативная память с объемом больше 4 Гбайт. Средство AWE операционной системы Windows позволяет приложениям полностью использовать физическую память, имеющуюся на применяемых для них хост-компьютерах. Дополнительные сведения о средстве AWE приведены ниже в данной главе. А пока достаточно отметить, что это средство позволяет приложению получать доступ к физической памяти, превышающей по объему 4 Гбайт. Версии Windows 2000 Professional и Windows 2000 Server поддерживают вплоть до 4 Гбайт физической памяти. Версия Windows 2000 Advanced Server поддерживает до 8 Гбайт, а версия Windows 2000 Data Center — до 64 Гбайт. Благодаря наличию средства AWE в приложении может использоваться такой объем физической памяти, который поддерживается операционной системой.

Необходимо помнить, что объем памяти 4 Гбайт, с которым должен работать 32-битовый процесс, представляет собой пространство виртуальных адресов, а не реальную память. Термин *виртуальный* означает, что данное пространство адресов представляет собой просто ряд адресов памяти. Прежде чем в приложении можно будет использовать это пространство адресов, не вызывая нарушения доступа, на данное пространство должна быть отображена память физического запоминающего устройства.

Основные службы управления памятью

На самом низком уровне средства управления памятью Windows состоят из средств реализации виртуальной памяти и управления обменом между виртуальной и физической памятью. Реализация этих средств связана с выполнением ряда описанных ниже фундаментальных задач.

1. Отображение пространства виртуальной памяти, отведенного процессу, на физическую память.
2. Подкачка, выполняемая путем обмена страницами в прямом и обратном направлениях между памятью и жестким диском, когда потоки процесса пытаются использовать больше физической памяти, чем имеется в настоящее время.

Диспетчер памяти не только предоставляет услуги по управлению виртуальной памятью, но и поддерживает основные службы подсистем среды Windows. К этим службам относятся перечисленные ниже.

- Отображаемые на память файлы.
- Поддержка приложений, в которых используются пространства адресов с разреженным заполнением памяти.
- Память с копированием при записи.

Значения степени детализации

Во всех микросхемах процессоров определен постоянный размер страницы для работы с памятью. В семействе процессоров x86 размер страницы равен 4 Кбайт. Объем памяти, указанный в любом запросе на распределение памяти, поступающем от приложения, округляется в *б*ольшую сторону до ближайшей границы страницы. Это означает, например, что для выполнения запроса на распределение объема памяти 5 Кбайт фактически требуется 8 Кбайт памяти.

Как и в большинстве операционных систем, в 32-битовых версиях Windows предусмотрена постоянная степень детализации распределения — граница, с учетом которой должны выполняться все операции резервирования памяти приложением. Эта граница всегда должна быть кратной размеру страницы системы. Для 32-битовых версий Windows эта граница равна 64 Кбайт, поэтому, если приложение запрашивает выполнение операции резервирования памяти, это резервирование должно начинаться с границы 64 Кбайт в пространстве адресов процесса. Хотя многие приложения предоставляют операционной системе Windows возможность самой определить точное местонахождение распределяемых ими буферов, в некоторых других приложениях выполняются операции распределения с указанием конкретных адресов. Но даже те приложения, которые сами указывают адреса для распределения, должны передавать в ОС Windows начальный адрес резервирования, который выровнен по границе 64 Кбайт в пространстве адресов процесса. Операционная система Windows округляет в меньшую сторону любой начальный адрес резервирования, который не выровнен правильно с учетом степени детализации распределения.

Если в приложении не учитывается применяемая в системе степень детализации распределения, равная 64 Кбайт, это может привести к бесполезному расходованию пространства адресов. Например, если в приложении зарезервирована область виртуальных адресов меньше, чем 64 Кбайт, то оставшаяся часть области с объемом 64 Кбайт остается не используемой приложением из-за того, что не учтена степень детализации распределения, предусмотренная в системе. Поскольку в приложении нельзя после этого выполнить операцию резервирования, которая позволила бы занять остальную часть этой области без того, чтобы система автоматически округлила указанное значение в меньшую сторону до начала области с объемом 64 Кбайт, это неиспользуемое пространство адресов, по существу, становится недоступным. Поэтому в любом приложении может быть легко исчерпано пространство адресов, отведенное для процесса, тогда как фактически будет зарезервирован или распределен лишь небольшой объем памяти. С дополнительными сведениями о резервировании и передаче памяти можно ознакомиться в приведенном ниже разделе "Виртуальная память".

Для получения сведений о степени детализации распределения памяти в системе и размере страницы в системе можно воспользоваться функцией `GetSystemInfo` API-интерфейса Win32. Несомненно, что оба эти значения в будущих версиях Windows могут измениться, поэтому в код приложения не следует включать текущие буквальные значения этих параметров. Пример того, как можно использовать функцию `GetSystemInfo` в расширенной процедуре SQL Server, приведен в упражнении 4.4 ниже в данной главе.

Защита памяти процесса

Операционная система Windows изолирует процессы друг от друга таким образом, что ни один процесс непривилегированного режима не может получить доступ к пространству адресов другого процесса или самой операционной системы и внести в него несанкционированные изменения. Благодаря этому операционная система Windows становится более надежной и защищает приложения друг от друга. Ниже описаны четыре фундаментальных аспекта такой защиты.

1. Определенная форма защиты памяти на основе аппаратных средств предоставляется всеми микросхемами процессоров, поддерживаемыми операционной системой Windows.
2. Общесистемные структуры данных и области памяти, используемые компонентами привилегированного режима, доступны только в привилегированном режиме, поэтому к ним не может обращаться код, работающий в непривилегированном режиме.
3. Операционная система Windows предоставляет каждому процессу закрытое пространство адресов. К нему запрещен доступ для потоков, принадлежащих к другим процессам.
4. Для секций совместно используемой памяти предусмотрены стандартные списки контроля доступа (Access Control List — ACL), которые проверяются при обращении к ним любых процессов.

Благодаря наличию этих четырех аспектов архитектуры управления памятью операционная система Windows становится гораздо более надежной, чем при использовании любого другого подхода. Указанные средства защиты позволяют исключить возможность внесения преднамеренных и непреднамеренных искажений в пространство адресов одного процесса другим процессом, а также способствуют повышению защищенности самой ОС Windows от катастрофических ошибок в приложениях.

ПРИМЕЧАНИЕ. Как было указано выше, в операционной системе Windows предусмотрены такие функции API-интерфейса, позволяющие одному процессу получать доступ к пространству адресов другого процесса, как `ReadProcessMemory` и `WriteProcessMemory`. Тем не менее для применения этих функций требуются особые права доступа, поэтому возможность случайно прочитать или модифицировать память, принадлежащую другому процессу, исключена. Обычно (но не всегда) эти функции используются в отладчике для доступа к памяти отлаживаемого процесса. Следует также отметить, что по умолчанию, если один процесс порождает другой с помощью вызова функции `CreateProcess`, то родительский процесс имеет права доступа, которые позволяют ему получить доступ к виртуальной памяти дочернего процесса. Эта возможность обычно используется для упрощения отладки.

Секции

В общих чертах пространство адресов процесса объемом 4 Гбайт организовано, как показано в табл. 4.4.

Таблица 4.4. Пространство адресов процесса и его содержимое

Диапазон адресов	Описание
`0x00000000-0x7FFFFFFF`	Код приложения и библиотек DLL, глобальные переменные, стеки потока, иными словами, память непривилегированного режима
`0x80000000-0xBFFFFFFF`	Ядро и исполняющая система, уровень HAL, драйверы начальной загрузки
`0xC0000000-0xC07FFFFF`	Таблицы страниц процесса, гиперпространство
`0xC0800000-0xFFFFFFFF`	Системный кэш, нерезидентный пул, резидентный пул

Если не применяется опция `/3GB`, то часть этого пространства, относящаяся к непривилегированному режиму, занимает первые 2 Гбайт, а часть, относящаяся к привилегированному режиму, занимает оставшиеся 2 Гбайт. Если же применяется опция `/3GB`, то часть, относящаяся к непривилегированному режиму, занимает первые 3 Гбайт (`0x00000000-0xBFFFFFFF`), а часть, относящаяся к привилегированному режиму, сжимается до оставшегося объема 1 Гбайт. Дополнительная информация об использовании указанной опции приведена в подразделе "Настройка памяти приложения". Но в приведенном ниже описании предполагается, что опция `/3GB` не введена в действие.

В части, относящейся к непривилегированному режиму, имеется несколько меньших секций (табл. 4.5). Эти секции кратко описаны ниже в данной главе.

Таблица 4.5. Секции в той части пространства адресов процесса, которая относится к непривилегированному режиму

Диапазон адресов	Размер	Описание
0x00000000-0x0000FFFF	64 Кбайт	Запрещенная область (предотвращает выполнение операций присваивания NULL-указателя)
0x00010000-0x7FFEFFFF	2 Гбайт за вычетом примерно 192 Кбайт	Закрытое пространство адресов процесса
0x7FFDE000-0x7FFDEFFF	4 Кбайт	Блок TEB для главного потока процесса. Блоки TEB для других потоков размещаются в направлении уменьшения адресов, начиная от предыдущей страницы (0x7FFDD000)
0x7FFDF000-0x7FFDFFFF	4 Кбайт	Блок PEB процесса
0x7FFE0000-0x7FFE0FFF	4 Кбайт	Совместно используемая страница данных непривилегированного режима
0x7FFE1000-0x7FFEFFFF	60 Кбайт	Запрещенная область (оставшаяся часть от секции с объемом 64 Кбайт, содержащей совместно используемую страницу данных непривилегированного режима)
0x7FFF0000-0x7FFFFFFF	64 Кбайт	Запрещенная область (предотвращает возможность того, что данные, вышедшие за пределы буфера в результате его случайного переполнения, пересекут границу между областями памяти непривилегированного и привилегированного режимов)

Раздел, в котором происходит присваивание NULL-указателя

Приходилось ли читателю задумываться над тем, почему в приложении нельзя использовать NULL-указатель (указатель на адрес 0x00000000)? В конечном итоге, он ведь ничем не отличается от других адресов в пространстве адресов процесса (и фактически даже является самым первым адресом в этом пространстве)? Тем не менее дело обстоит иначе. И, как описано ниже, причина такого положения дел состоит не в том, что в операционной системе Windows первые 64 Кбайт пространства адресов процесса обозначены как выходящие за допустимые пределы, чтобы помочь программистам перехватывать ошибки, связанные с присваиванием NULL-указателя.

Дело в том, что секция присваивания NULL-указателя является очень простым, но удивительно полезным средством операционной системы, которое позволяет перехватывать в программах ошибки, связанные с неудачным выполнением операций распределения памяти. Например, рассмотрим следующий код на языке C:

```
char *pszLastName = (char *)malloc(LAST_NAME_SIZE);
strcpy(pszLastName,"Smith");
```

В этом коде не предусмотрена проверка ошибок. Если же функция `malloc` оказывается неспособной распределить буфер требуемого размера, то возвращает `NULL`. А поскольку в ОС Windows все первые 64 Кбайт пространства адресов процесса обозначены как выходящие за допустимые пределы (включая

адрес 0x00000000 – адрес NULL-указателя), то любая попытка получить доступ с помощью NULL-указателя приводит к возникновению ошибки нарушения доступа. Если в приведенном выше фрагменте кода вызов функции `malloc` возвратит `NULL`, то выполнение функции `strcpy` вызовет активизацию ошибки нарушения доступа. Такая возможность появляется не в связи с тем, что ОС Windows проверяет каждую ссылку на указатель для получения гарантий того, что он не равен `NULL`, а в связи с тем, что не разрешено использование ни одного адреса в пределах первых 64 Кбайт пространства непривилегированного режима (будь то 0x00000000 или другой адрес из этой секции памяти), и нарушение этого требования очень легко обнаруживается.

Означает ли это, что операционная система теряет 64 Кбайт памяти системы в расчете на каждый процесс? Нет, это не так. Следует учитывать, что пространство адресов процесса является виртуальным, и те секции, которые обозначены операционной системой как выходящие за допустимые пределы, не отображаются на физическую память. Для того чтобы можно было воспользоваться таким полезным средством, как секция присваивания NULL-указателя, приходится расставаться лишь с частью адресов виртуальной памяти, равной 64 Кбайт, а физическая память при этом не расходуется. Почему же секция присваивания NULL-указателя имеет объем 64 Кбайт? Почему бы не обозначить как выходящий за допустимые пределы только адрес NULL-указателя (0x00000000) или, самое большее, одну страницу с объемом 4 Кбайт? В ОС Windows как выходящая за допустимые пределы обозначается вся секция памяти с объемом 64 Кбайт по двум описанным ниже причинам.

1. Операции резервирования памяти приложениями непривилегированного режима должны осуществляться с учетом границ степени детализации распределения (64 Кбайт). Поэтому, даже если бы была обозначена как выходящая за пределы допустимых адресов только первая страница объемом 4 Кбайт, все равно нельзя было бы зарезервировать память в объеме 60 Кбайт пространства адресов, оставшемся от первоначального объема, составляющего 64 Кбайт.
2. Ссылки на NULL-указатель часто скрыты в арифметических операциях с указателями, так что фактически не происходит ссылка на адрес памяти NULL-указателя, а берется сам NULL-указатель и какое-то смещение. Это означает, что наличие ссылки на NULL-указатель может в конечном итоге привести к тому, что в приложении будет сформирована ссылка на адрес в памяти, отличный от 0x00000000. А поскольку вся область с объемом 64 Кбайт обозначена как выходящая за пределы допустимых адресов, то появляется возможность перехватывать многие ошибки, вызванные подобными недостатками в программах.

Описанные выше особенности секции присваивания NULL-указателя лучше всего объяснить на примере. Упражнения 4.1–4.3 посвящены рассмотрению того, как создать несколько экспериментальных приложений, в которых демонстрируются ссылки на NULL-указатель, и как воспользоваться средствами Windows для обнаружения таких ситуаций.

Секция пространства закрытых адресов процесса

Исполняемый файл приложения и библиотеки DLL загружаются именно в пространство закрытых адресов процесса. При выполнении всех операций распределения закрытых адресов память берется из этой области; кроме того, ей ставятся в соответствие файлы с отображением на память. Приложение функционирует именно в этом пространстве.

Секция привилегированного режима

В секции привилегированного режима размещается код поддержки файловой системы, управления потоками, управления памятью, поддержки сетевого взаимодействия и всех драйверов устройств. Весь код, находящийся в секции привилегированного режима, совместно используется всеми потоками.

У читателя может возникнуть вопрос, действительно ли ядру операционной системы требуется вся та верхняя половина пространства адресов процесса, которую оно занимает. К сожалению, ответ на этот вопрос является положительным; это пространство действительно требуется. Ядру необходимо такое пространство для размещения кода операционной системы, буферов кэша устройств ввода-вывода, таблиц страниц процесса, кода драйверов устройств и т.д. Несомненно, что ядро вполне могло бы продуктивно использовать намного больший объем пространства. (В конечном итоге ядро получит все необходимое ему пространство в 64-битовой версии Windows.)

Применительно к пространству привилегированного режима следует учитывать еще одно соображение — если начальная загрузка операционной системы выполняется с опцией `/3GB` (описанной ниже), то пространство привилегированного режима сокращается и становится равным всего лишь 1 Гбайт. Это, в свою очередь, приводит к ограничению размеров некоторых структур данных, которые обычно хранятся в пространстве привилегированного режима. Например, если применяется опция `/3GB`, то в приложении может быть получен доступ лишь к 16 Гбайт общей памяти системы из-за уменьшения размера таблицы страниц процесса, вызванного использованием ограниченного пространства привилегированного режима.

Области PEB и TEB

Области PEB и TEB не относятся к тем областям памяти, которыми приходится часто пользоваться разработчику, но ему полезно знать об их существовании и назначении. Как было указано в главе 3, каждый процесс имеет блок среды процесса (Process Environment Block — PEB), который располагается в пространстве непривилегированного режима. Как показано в табл. 4.5, точный адрес блока PEB процесса равен `0x7FFDF000`. Это означает, что можно вывести на экран содержимое этой области памяти из любого отладчика, чтобы ознакомиться с блоком PEB любого процесса. В отладчике WinDbg предусмотрена специальная команда для выполнения именно этой задачи — `!peb`. Попытайтесь выполнить команду `!peb` после очередного подключения к программе SQL Server с помощью отладчика WinDbg. Вы обнаружите, что результаты выполнения этой команды содержат целый ряд интересных фрагментов данных, включая список модулей,

загруженных в настоящее время в пространство памяти процесса SQL Server, командную строку, переданную в этот процесс, адрес применяемой в нем по умолчанию динамической области памяти и многие другие данные.

Как отмечалось в главе 3, с каждым потоком связан блок среды потока (Thread Environment Block – TEB). В пространстве адресов непривилегированного режима содержится по одному блоку TEB для каждого потока, принадлежащего данному процессу. Как и блок PEB, эти блоки хранятся в пространстве непривилегированного режима для того, чтобы система могла получить к ним доступ, не переключаясь в привилегированный режим.

Как показано в табл. 4.5, адрес блока TEB для главного потока процесса равен `0x7FFDE000`. Чтобы вывести на экран содержимое блока TEB, можно воспользоваться командой `!teb` отладчика WinDbg. Если в вызове команды `!teb` не будут указаны какие-либо параметры, то будет получен блок TEB, относящийся к текущему потоку. А если в вызове команды `!teb` передается адрес, то на экран выводится блок TEB, находящийся по этому адресу, если он действительно там имеется.

Блоки TEB для рабочих потоков в многопотоковом приложении хранятся на странице с адресом `0x7FFDD000` и на страницах, непосредственно предшествующих ей в памяти (например, `0x7FFDC000`, `0x7FFDB000` и т.д.).

Страница совместно используемых данных непривилегированного режима

Страница памяти, находящаяся по адресу `0x7FFE0000`, называется *страницей совместно используемых данных непривилегированного режима*. Она содержит такие глобальные элементы данных, как количество тактов системного таймера, системное время, номер версии и многие другие элементы данных, относящиеся к уровню системы. Она предназначена только для чтения и отображается на страницу памяти, которая фактически находится в пространстве адресов привилегированного режима. Размещение этой страницы в пространстве непривилегированного режима предусмотрено для того, чтобы все процедуры API-интерфейса могли получить доступ к важным системным данным без необходимости переключаться в привилегированный режим.

Граничные секции

Последние две области пространства адресов непривилегированного режима находятся за пределами адресов, доступных для приложений. Первой из этих областей является оставшаяся часть области с объемом 64 Кбайт, в которой находится страница совместно используемых данных непривилегированного режима. Эта область с объемом 60 Кбайт отмечается операционной системой как находящаяся за пределами доступных адресов; любая попытка обратиться к данной области приводит к возникновению ошибки нарушения доступа. Но тот факт, что оставшаяся часть области с объемом 64 Кбайт, содержащей страницу совместно используемых данных непривилегированного режима, отмечается как выходящая за пределы допустимых адресов, фактически не оказывает отрицательного влияния на работу приложений непривилегированного режима, поскольку эта область

так или иначе была бы для них недоступной, если учесть, что операции резервирования памяти в непривилегированном режиме должны начинаться с границы степени детализации распределения.

Второй областью являются последние 64 Кбайт пространства адресов непривилегированного режима. В операционной системе Windows эта область отмечается как выходящая за пределы доступных адресов, чтобы исключить возможность доступа приложения к области виртуальной памяти, которая находится слишком близко от границы между областями памяти непривилегированного и привилегированного режимов. Поскольку проверка работы таких процедур, как `WriteProcessMemory`, фактически проводится с учетом кода привилегированного режима, эти процедуры могут получать доступ к областям памяти, обычно находящимся за пределами допустимых адресов для кода непривилегированного режима. Обозначив последние 64 Кбайт пространства непривилегированного режима как выходящие за пределы допустимых адресов, операционная система Windows устанавливает защиту против операций доступа к памяти, которые начинаются в пространстве непривилегированного режима и распространяются на пространство привилегированного режима.

Системный файл подкачки

Для реализации виртуальной памяти (т.е. для предоставления приложениям возможности получать доступ к большему объему памяти, чем физически существует на компьютере) диспетчер памяти Windows выполняет по мере необходимости операции копирования страниц на диск и с диска прозрачно для приложения. Файл, применяемый для хранения таких страниц, называется *системным файлом подкачки*.

С точки зрения разработчика приложения, системный файл подкачки увеличивает объем памяти, доступной для использования. Благодаря наличию этого файла создается впечатление, что система обладает гораздо бо́льшим объемом физической памяти, чем фактически в ней имеется. Именно поэтому на компьютере, который оснащен физической памятью с объемом 1 Гбайт, могут одновременно эксплуатироваться многочисленные приложения, в каждом из которых имеется пространство адресов процесса с объемом 4 Гбайт, несмотря на то, что лишь для 50% той части этого пространства, которая относится к непривилегированному режиму, предусмотрена поддержка в виде физической памяти.

В принципе, целесообразно рассматривать всю реальную память, на которую отображается виртуальная память, как системный файл подкачки. Даже несмотря на то, что в ходе работы системы постоянно происходит копирование страниц в физическую оперативную память и из нее, львиная доля реальной памяти, на которую отображается виртуальная память в системе, обычно находится в системном файле подкачки.

Хотя и существует возможность обеспечить эксплуатацию операционной системы Windows без файла подкачки, обычно это не рекомендуется. В типичной конфигурации системный файл подкачки значительно превышает по объему физическую память, установленную на компьютере, и предоставляет приложениям

эффективный механизм, позволяющий получить доступ к большему объему памяти, чем фактически имеется на компьютере.

Объем файла подкачки является наиболее важным параметром, от которого зависит то, насколько велика область хранения данных, доступная для приложения. Сама величина реальной памяти, доступной для приложения, очень мало зависит от объема оперативной памяти, но от этого объема, безусловно, в значительной степени зависит производительность. Если объем физической оперативной памяти слишком мал, то система вынуждена непрерывно заниматься копированием страниц данных из оперативной памяти в файл подкачки и из файла подкачки в оперативную память (такое состояние называется *пробуксовкой*), а это, безусловно, приводит к существенному снижению производительности.

Адресные расширения для работы с окнами

В операционной системе Window предусмотрены средства, позволяющие приложениям получать доступ к объему физической памяти больше 4 Гбайт, которые известны под названием адресных расширений для работы с окнами (Address Windowing Extensions — AWE). Как было указано выше, 32-битовый указатель представляет собой целочисленное значение с ограниченными размерами представления в памяти, которое позволяет хранить значения не больше `0xFFFFFFFF`, т.е. обращаться к пространству адресов памяти объемом 4 Гбайт. Средства AWE дают возможность приложению обойти это ограничение и получить доступ ко всей памяти, поддерживаемой операционной системой.

На концептуальном уровне средства AWE не представляют собой что-либо новое, поскольку в операционных системах и приложениях использовались аналогичные механизмы для преодоления ограничений на размер указателей практически с первых дней создания компьютеров. Например, еще в те дни, когда применялась операционная система DOS, находили свое применение 32-битовые расширители (например, Phar Lap, Plink и другие), которые позволяли в 16-битовых приложениях обращаться к памяти, выходящей за пределы их обычного пространства адресов. В то время были распространены диспетчеры и API-интерфейсы специального назначения для поддержки расширенной и дополнительной памяти; читатель может помнить даже такие программные продукты, как программа QEMM-386 компании Quarterdeck, которая в то время широко использовалась для указанных целей.

Как правило, действие механизмов, позволяющих обращаться с помощью указателя к адресам памяти, находящимся за пределами непосредственного доступа (т.е. к адресам, слишком большим, для того чтобы их можно было сохранить в самом указателе), основано на том, что в них предоставляется окно (или область) в пределах доступного пространства адресов, которое используется для переноса в него недоступной области памяти, а затем — обратного переноса. Именно по этому принципу действуют средства AWE: в них предоставляется область в пространстве адресов процесса (окно), которая служит в качестве своего рода промежуточной области для закрепления на ней страниц памяти, выходящих за пределы 4 Гбайт, с последующим переносом в противоположном направлении.

Для того чтобы можно было воспользоваться средствами AWE, в приложении должны быть выполнены описанные ниже действия.

1. Распределить область физической памяти, к которой должен быть получен доступ, с помощью функции `AllocateUserPhysicalPages` API-интерфейса Win32. Для использования этой функции требуется, чтобы вызывающая процедура имела разрешение `Lock Pages in Memory`.
2. Создать в пространстве адресов процесса область, которая будет использоваться в качестве окна для отображения представлений этой физической памяти с помощью функции API-интерфейса `VirtualAlloc`. Описание функции `VirtualAlloc` приведено ниже.
3. Отобразить представление физической памяти в окно виртуальной памяти с помощью функции `MapUserPhysicalPages` или `MapUserPhysicalPagesScatter` API-интерфейса Win32.

Хотя средства AWE предусмотрены во всех версиях Windows 2000 и в более поздних выпусках этой операционной системы и могут даже применяться в системах с объемом физической оперативной памяти меньше 2 Гбайт, чаще всего эти средства используются в системах с объемом памяти 2 Гбайт или больше, поскольку, как было указано выше в данной главе, они представляют собой единственный способ, с помощью которого 32-битовый процесс может обращаться к памяти за пределами 3 Гбайт. Если поддержка AWE в программе SQL Server применяется в системе с объемом физической памяти меньше 3 Гбайт, то система игнорирует опцию AWE и использует вместо нее обычные средства управления виртуальной памятью.

Одной из интересных характерных особенностей памяти AWE является то, что ее содержимое никогда не выводится на жесткий диск по принципу страничного обмена. Заслуживает внимания то, что в процедурах API-интерфейса, относящихся к AWE, ссылки на память, к которой они обращаются, оформлены как ссылки на физическую память. Именно в этом состоит суть самой памяти AWE — это физическая память, выходящая за пределы контроля диспетчера виртуальной памяти Windows.

Для доступа к окну виртуальной памяти, применяемому для отображения области физической памяти, предоставленной средствами AWE, необходимо иметь права чтения-записи. Поэтому единственным атрибутом защиты, который может быть передан в функцию `VirtualAlloc` при определении этого окна, является `PAGE_READWRITE`. Безусловно, это также означает, что для защиты страниц из этой области от модификации или доступа нельзя использовать функцию `VirtualProtect`.

Настройка памяти приложения

Возможность применения опции начальной загрузки `/3GB` предусмотрена в версиях Advanced Server и Data Center операционной системы Windows 2000 (и в более поздних выпусках ОС Windows). Эта опция позволяет расширить пространство адресов непривилегированного режима процесса от 2 до 3 Гбайт за счет пространства адресов привилегированного режима (которое сокращается от 2 до 1 Гбайт). В операционной системе Windows применение данного средства

принято называть настройкой памяти приложения или настройкой 4 Гбайт (4GB Tuning – 4GT). Чтобы ввести в действие средства настройки памяти приложения, достаточно ввести параметр "/3GB" (без кавычек) в соответствующей строке секции [operating systems] файла BOOT.INI. Обычно специалисты службы эксплуатации выполняют настройку конфигурации своих систем так, чтобы можно было выполнять их начальную загрузку с опцией и без опции /3GB, определяя формат записей в секции [operating systems] файла BOOT.INI с учетом возможности выбирать ту или иную опцию при запуске системы.

ПРЕДОСТЕРЕЖЕНИЕ. С опцией /3GB может быть также выполнена начальная загрузка версий Windows 2000 Professional, Windows 2000 Server и Windows XP. Но в этом случае применение указанной опции влечет за собой отрицательные последствия, поскольку пространство привилегированного режима уменьшается до 1 Гбайт, а пространство непривилегированного режима не увеличивается. Иными словами, в обмен на отказ от части пространства привилегированного режима вы ничего не получаете.

ПРИМЕЧАНИЕ. В версии Windows Server 2003 введена новая опция начальной загрузки, предназначенная для настройки пространства адресов процесса непривилегированного режима, /USERVA. Параметр /USERVA можно ввести в файл BOOT.INI по такому же принципу, как и параметр /3GB. Преимущество применения /USERVA вместо /3GB состоит в том, что первая опция предоставляет более точную степень контроля над тем, какая именно часть пространства адресов резервируется для использования в непривилегированном и привилегированном режимах, по сравнению со второй. Например, параметр /USERVA=2560 определяет конфигурацию, в которой 2,5 Гбайт будут отведены для пространства непривилегированного режима, а остальные 1,5 Гбайт — для привилегированного режима. Но остаются в силе те же предостережения, которые относятся к параметру /3GB.

Исполняемые файлы, способные работать с большим пространством адресов

До того как в операционную систему Windows была введена поддержка опций /3GB, ни одно приложение не имело возможности получить доступ к памяти с помощью указателя, в котором установлен старший бит. Приложения непривилегированного режима могли получать доступ только к их адресам, для представления которых достаточно было использовать всего лишь 31 бит из 32-битового указателя. Таким образом, 1 бит оставался неиспользуемым, поэтому некоторые разработчики, которые были изобретательными программистами и не хотели терять целый бит в указателе пространства адресов процесса, стали применять его для других целей (например, для обозначения указателя как ссылающегося на конкретный тип распределения, характерного для данного приложения). Введение опции /3GB вызвало настоящее замешательство в сообществе пользователей программы SQL Server, поскольку в приложениях подобного типа нелегко было различить указатель с допустимым форматом, который просто ссылается на область памяти, выходящую за границы 2 Гбайт, и указатель, который ссылается на

область памяти в пределах 2 Гбайт, но в нем старший бит установлен по другим причинам. По сути, начальная загрузка компьютеров с применением опции /3GB могла привести к нарушению работы таких приложений.

Для того чтобы выйти из сложившегося положения, компания Microsoft дополнительно ввела поддержку для нового битового флажка в поле Characteristics формата переносимого исполняемого файла (Portable Executable – PE) Win32 (этот формат определяет компоновку исполняемых файлов EXE и DLL в операционной системе Windows). Такой битовый флажок указывает, способно ли приложение поддерживать большое пространство адресов непривилегированного режима. Если этот флажок (IMAGE_FILE_LARGE_ADDRESS_AWARE) разрешен, то устанавливается бит 32 в поле Characteristics заголовка исполняемого файла. Устанавливая данный флажок в заголовке исполняемого файла, разработчик приложения сообщает операционной системе Windows, что это приложение способно правильно обрабатывать указатели с установленным старшим битом, т.е. с помощью этого бита в приложении не выполняются какие-либо специальные операции. Если установлен данный флажок и выполнена начальная загрузка подходящей версии Windows с разрешенной опцией /3GB, то система предоставляет процессу пространство закрытых адресов непривилегированного режима с объемом 3 Гбайт. Для проверки того, установлен ли в исполняемом файле этот флажок, можно воспользоваться такими утилитами, как DumpBin и ImageCfg, которые позволяют вывести на внешнее устройство информацию заголовка любого исполняемого файла.

В языке Visual C++ доступ к флажку IMAGE_FILE_LARGE_ADDRESS_AWARE предоставляется с помощью параметра /LARGEADDRESSAWARE редактора связей. (Кроме того, значение этого флажка в существующем исполняемом файле можно изменить с помощью программы ImageCfg.) В программе SQL Server такой флажок разрешен, поэтому после начальной загрузки подходящей версии Windows с опцией /3GB система установит объем пространства закрытых адресов процесса SQL Server, равный 3 Гбайт.

ПРИМЕЧАНИЕ. Флажок IMAGE_FILE_LARGE_ADDRESS_AWARE проверяется при запуске процесса и игнорируется при загрузке библиотеки DLL. Поэтому при разработке библиотеки DLL, в которой могут использоваться указатели с установленным старшим битом, следует всегда учитывать такую возможность и не применять старший бит указателя в непредусмотренных для этого целях.

Сравнение возможностей опции /3GB и средств AWE

Возможность увеличить пространство закрытых адресов процесса на 50%, безусловно, является удобным и полезным усовершенствованием средств управления памятью Windows, но средства AWE операционной системы Windows являются еще более гибкими и масштабируемыми, чем кажется на первый взгляд. Как было указано выше, при увеличении пространства закрытых адресов процесса на гигабайт, этот гигабайт приходится брать из пространства адресов привилегированного режима, которое сужается от 2 до 1 Гбайт. А поскольку при размещении кода привилегированного режима приходится сталкиваться с нехваткой пространства

даже при использовании для него полного объема в 2 Гбайт, сокращение объема этого пространства означает, что приходится также дополнительно сжимать некоторые внутренние структуры ядра. Основной среди этих структур является таблица, применяемая в операционной системе Windows для управления физической памятью на компьютере. После сжатия секции привилегированного режима до 1 Гбайт размер этой таблицы ограничивается до такой степени, что она позволяет управлять максимальным объемом физической памяти, который составляет всего лишь 16 Гбайт. Например, при эксплуатации приложения под управлением версии Windows 2000 Data Center на компьютере с объемом физической памяти 64 Гбайт начальная загрузка с опцией `/3GB` приводит к тому, что обеспечивается возможность управлять только частью оперативной памяти компьютера, составляющей 25%, а оставшиеся 48 Гбайт не будут применимыми для операционной системы или приложений.

Кроме того, средства AWE позволяют получить доступ к большему объему памяти по сравнению с тем, который доступен при использовании опции `/3GB`. Очевидно, что опция `/3GB` позволяет распределить всего лишь один дополнительный гигабайт пространства закрытых адресов процесса. Это дополнительное пространство предоставляется тем приложениям, в которых обеспечивается автоматическая и прозрачная поддержка большого пространства адресов, но оно ограничивается только объемом 1 Гбайт. В отличие от этого, средства AWE позволяют предоставить приложению доступ ко всему объему физической оперативной памяти, доступной для операционной системы, при условии, что это приложение было запрограммировано с учетом функций API-интерфейса AWE подсистемы Win32. Поэтому средства AWE являются намного более удобными и открывают целый ряд дополнительных возможностей, несмотря на то, что при их использовании и доступе к ним приходится преодолевать дополнительные трудности.

Преобразование адресов

Преобразованием адресов называется операция преобразования виртуального адреса в адрес физической оперативной памяти. Такая операция выполняется каждый раз, когда в любом из процессов предпринимается попытка получить доступ к блоку данных с использованием его виртуального адреса. При любой попытке обратиться из процесса к блоку данных по адресу могут возникать три описанные ниже ситуации.

1. Адрес является действительным, и страница с этим адресом уже находится в физической памяти.
2. Адрес является действительным, и страница с этим адресом хранится в системном файле подкачки. В этом случае страница с данными передается в физическую память для того, чтобы к ней можно было получить доступ. Такая ситуация называется *ошибкой отсутствия страницы*. (За ошибками отсутствия страницы, возникающими в определенном процессе, можно следить с помощью счетчика `Process:Page Faults/sec` программы Perfmon и столбца `Page Faults` программы Task Manager.)
3. Адрес является недействительным; в этом случае система активизирует исключительную ситуацию, связанную с нарушением доступа (непривилеги-

рованный режим), или происходит ее останов — появляется так называемый *синий экран* (привилегированный режим).

Виртуальные адреса не отображаются на физические адреса непосредственно. Вместо этого каждый виртуальный адрес компонуется из трех элементов: индекс каталога страниц, индекс таблицы страниц и индекс байта. Эти элементы определяют отображение между виртуальным адресом и адресом физической оперативной памяти, на который он ссылается. Для каждого процесса диспетчер памяти Windows создает каталог страниц, который используется для хранения в нем информации обо всех таблицах страниц этого процесса. В операционной системе Windows физический адрес каталога страниц каждого процесса хранится в блоке KPROCESS (это — блок процесса привилегированного режима, хранящийся в блоке EPROCESS, который упоминался в главе 3), и ему присваивается адрес `0xC0300000` в пространстве адресов процесса.

В процессоре для отслеживания адресов в таблицах каталога страниц процессов предусмотрен специальный регистр (CR3, или Control Register 3, — регистр управления 3, в процессоре x86; PDR, или Page Directory Register, — регистр каталога страниц, в процессоре Alpha). После каждого переключения контекста, при котором для выполнения на процессоре планируется поток из другого процесса, содержимое этого регистра загружается из блока KPROCESS в процессор для того, чтобы модуль MMU процессора мог определить, где находится таблица каталога страниц. При переключении контекста между потоками в одном и том же процессе перезагрузка этого регистра не требуется, поскольку все потоки процесса совместно используют одно и то же пространство адресов.

Содержимое этого специального регистра служит в качестве точки отсчета для всех средств управления памятью системы. Без него нельзя было бы найти каталог страниц процесса, а без каталога страниц невозможно получить доступ к самому пространству адресов процесса. Данный регистр предоставляет точку входа для аппаратных средств процессора, обеспечивающих управление памятью, что позволяет получить доступ к пространству адресов отдельного процесса.

Каждый каталог страниц состоит из последовательности записей каталога страниц. В первых 10 бит 32-битового виртуального адреса хранится индекс записи каталога страниц (Page Directory Entry — PDE), который сообщает операционной системе Windows, какая таблица страниц должна использоваться для поиска физической памяти, связанной с этим адресом.

Каждая таблица страниц состоит из последовательности записей таблицы страниц. Вторые 10 бит 32-битового виртуального адреса содержат индекс записи в этой таблице и указывают, какая запись таблицы страниц (Page Table Entry — PTE) содержит адрес страницы в физической памяти, на которую отображен виртуальный адрес.

На компьютерах с процессорами x86 последние 12 бит 32-битового виртуального адреса содержат смещение в байтах на странице физической памяти, к которому относится этот виртуальный адрес. Количество битов, требуемых для хранения данного смещения, зависит от размера системной страницы. Поскольку размер системной страницы на компьютерах с процессорами x86 равен 4 Кбайт, для хранения смещения страницы требуется 12 бит ($4096 = 2^{12}$).

При преобразовании адреса происходят описанные ниже события.

1. Модуль MMU процессора находит каталог страниц для рассматриваемого процесса с использованием упомянутого выше специального регистра.
2. Для поиска записи PDE, которая указывает на таблицу страниц, применяемую при преобразовании виртуального адреса в физический, используется индекс каталога страниц (он берется из первых 10 бит виртуального адреса).
3. Для поиска записи PTE, которая отображает физическое местонахождение страницы виртуальной памяти, указанной с помощью этого адреса, используется индекс таблицы страниц (он берется из вторых 10 бит виртуального адреса).
4. Для поиска физической страницы применяется запись PTE. Если виртуальная страница отображается на страницу, которая уже находится в физической памяти, то запись PTE будет содержать значение PFN (Page Frame Number — номер фрейма страницы) для той страницы в физической памяти, которая содержит искомые данные. (Процессоры обращаются к страницам памяти, указывая их номера PFN.) Если же искомая страница не находится в физической памяти, то модуль MMU активизирует ошибку отсутствия страницы, после чего в коде обработки ошибки отсутствия страницы операционной системы Windows предпринимаются попытки найти эту страницу в системном файле подкачки. Если страницу удается найти, она загружается в физическую память, и обновляется запись PTE, которая показывает ее местонахождение. Если же страницу не удается найти, а преобразование адресов осуществляется в непривилегированном режиме, то происходит нарушение доступа, поскольку данный виртуальный адрес ссылается на недействительный физический адрес. Если же страницу не удается найти, а преобразование адресов происходит в привилегированном режиме, то на экран выводится сообщение о том, что необходимо проверить ошибку в коде системы (поскольку это сообщение выводится на экран с синим фоном, такую ситуацию называют также "появлением синего экрана").

На первый взгляд этот четырехэтапный процесс, который требуется для получения физического адреса, соответствующего виртуальному адресу, может показаться неэффективным. Возникает впечатление, что было бы гораздо проще и эффективнее компоновать виртуальный адрес из двух основных элементов: во-первых, запись PTE, хранящая ссылку на страницу в физической памяти, на которую отображается виртуальный адрес, и, во-вторых, смещение страницы, которое позволяет точно указать местонахождение блока данных, обозначенного этим адресом. Тем не менее в процессорах x86 и Alpha принят именно такой четырехэтапный подход из соображений экономии памяти. Попытка упростить описанный выше процесс преобразования адресов и свести его к примитивному одноэтапному преобразованию, в котором каждый виртуальный адрес компонуется только из двух элементов, как указано в данном абзаце, привела бы к значительному возрастанию потребности в памяти для управления упомянутой таблицей по сравнению с четырехэтапным процессом, особенно в системах, в которых основная часть пространства адресов остается нераспределенной. В частности, для

преобразования пространства адресов с объемом 4 Гбайт потребовалось бы 1 048 576 записей PTE (4 Гбайт × размер страницы 4 Кбайт = 1 048 576). А поскольку для каждой записи PTE требуется 32-битовый указатель, то для преобразования в пространстве адресов каждого процесса нужно было бы расходовать 4 Мбайт физической памяти (1 048 576 × 4 байта = 4 Мбайт). С другой стороны, при использовании четырехэтапной процедуры преобразования, которая предусмотрена в процессорах x86 и Alpha, существует необходимость полностью определять только каталог страниц, а память для каталога страниц можно распределять по мере того, как в этом будет возникать необходимость. В связи с тем, что пространство адресов в большинстве процессов остается нераспределенным, такой подход позволяет экономить значительный объем физической памяти.

Но из сказанного выше следует, что производительность преобразования адресов при обслуживании таких процессов, для которых требуется доступ ко всему объему памяти, будет очень низкой, поэтому в процессорах x86 и Alpha пары адресов, преобразованных из виртуальных в физические, записываются в кэш. Кэш-память, предназначенная для хранения указанных пар, известна под названием *буфера преобразования* (Translation Buffer — TB) или *буфера быстрого преобразования адреса* (Translation Look-aside Buffer — TLB). При поступлении в модуль MMU виртуального адреса берется номер виртуальной страницы, который сравнивается с номерами виртуальных страниц каждой записи в кэше. Если обнаруживается совпадение, то вся четырехэтапная процедура преобразования исключается и просто осуществляется поиск номера PFN в физической памяти, адрес которой находится в записи кэша. Недостатком процедуры, применяемой в планировщике Windows для переключения с одного процесса на другой, является то, что записи, связанные с процессом, который уходит из-под управления планировщиком, должны быть уничтожены. После этого с помощью четырехэтапной процедуры кэш заполняется записями, относящимися к новому процессу.

Расширение физического адреса

В процессорах Intel, начиная с Pentium Pro и включая последующие модели, предусмотрена поддержка модели преобразования адресов памяти, называемой *расширением физического адреса* (Physical Address Extension — PAE). Модель PAE позволяет предоставить доступ к объему физической памяти, достигающему 64 Гбайт. В режиме PAE модуль MMU все еще поддерживает каталоги страниц и таблицы страниц, но между ними вводится новый уровень — таблица указателей каталога страниц. Кроме того, в режиме PAE записи PDE и PTE имеют длину 64 бита (а не стандартные 32 бита). Система обладает способностью адресовать больше памяти, чем при стандартном преобразовании, в связи с тем, что записи PDE и PTE имеют длину, вдвое превышающую стандартную, а не благодаря введению таблицы указателей каталога страницы. А сама таблица указателей каталога страниц требуется для управления этими таблицами с высокой емкостью и введения в них индексов.

Для использования режима PAE требуется специальная версия ядра Windows. Это ядро поставляется с каждой версией операционной системы Windows, начиная от Windows 2000 и включая последующие версии; оно находится в файле `Ntkrnlpa.exe`, предназначенном для однопроцессорных компьютеров,

и в файле Ntkrnlpamp, предназначенном для многопроцессорных компьютеров. Чтобы разрешить использование режима PAE, нужно ввести параметр /PAE в файл BOOT.INI по такому же принципу, как и параметр /3GB или /USERVA.

Упражнения

Выше в этой главе рассматривались ссылки на NULL-указатели и способы, применяемые в ОС Windows для упрощения их обнаружения в приложениях (тем не менее, эти способы не позволяют полностью предотвратить их появление). В следующих трех упражнениях приведены некоторые примеры кода, позволяющие ознакомиться с различными типами ссылок на NULL-указатели и демонстрирующие способы обработки в операционной системе Windows ссылок каждого из этих типов.

Упражнение 4.1. Ссылки на NULL-указатель

1. Создайте терминальное приложение на основе листинга 4.1, загрузив и откомпилировав проект Visual Studio, находящийся в подкаталоге CH04\memexamp00 компакт-диска, прилагаемого к данной книге. Предполагается, что для выполнения описанных ниже этапов используется программа Visual Studio C++ (VC++) 6.0 или последующей версии.

Листинг 4.1. Ссылка на NULL-указатель

```
// memexamp00.cpp. Пример формирования ссылки на NULL-указатель
//

#include "stdafx.h"
#include "stdlib.h"
#include "string.h"

#define LAST_NAME_SIZE 2147483647

int main(int argc, char* argv[])
{
  char *pszLastName = (char *)malloc(LAST_NAME_SIZE);
  strcpy(pszLastName,"Smith");
  return 0;
}
```

2. Установите точку останова на строке с вызовом функции strcpy и вызовите приложение на выполнение.
3. После того как приложение остановится на строке с вызовом функции strcpy, подведите указатель мыши к элементу pszLastName в окне редактора VC++. На экране должна появиться всплывающая подсказка, которая показывает, что переменная pszLastName имеет значение 0x00000000. С чем это связано? Данный указатель равен NULL, поскольку в программе затребовано распределение большего объема памяти (2 Гбайт), чем может предоставить Windows.

   Поскольку в данном коде не предусмотрен контроль ошибок, в функции strcpy предпринимается попытка скопировать строку "Smith" по этому недопустимому адресу.

4. Нажмите клавишу <F10>, чтобы выполнить код строки с функцией `strcpy`. Теперь должно быть обнаружено нарушение доступа. ОС Windows перехватывает попытку доступа к указанному адресу памяти `0x00000000` (`NULL`) и активизирует наблюдаемую ошибку. Нажмите клавиши <Shift+F5>, чтобы прекратить отладку.

Упражнение 4.2. Скрытая ссылка на NULL-указатель

Теперь модифицируем рассматриваемое приложение, чтобы вызвать появление ссылки на NULL-указатель, которая не является такой очевидной.

1. Откорректируйте используемый код, чтобы он выглядел так, как в листинге 4.2 (или загрузите файл из каталога `memexamp01` компакт-диска).

Листинг 4.2. Менее очевидная ссылка на NULL-указатель

```
// memexamp01.cpp. Пример формирования ссылки на NULL-указатель
//
#include "stdafx.h"
#include "stdlib.h"
#include "string.h"

#define LAST_NAME_SIZE 2147483647
char szLastName[]="Smith";

int main(int argc, char* argv[])
{
  char *pszLastName = (char *)malloc(LAST_NAME_SIZE);
  *(pszLastName+strlen(szLastName)+1)='\0';
  strncpy(pszLastName,szLastName,strlen(szLastName));
  return 0;
}
```

2. В этом коде для заполнения данными адреса, на который ссылается переменная `pszLastName`, используется функция `strncpy`, а не `strcpy`. Функция `strncpy` часто оказывается более предпочтительной, чем `strcpy`, поскольку позволяет предотвращать переполнение буфера. При ее использовании можно управлять количеством копируемых символов. Но в связи с применением функции `strncpy` необходимо позаботиться о правильном завершении строки, на которую ссылается переменная `pszLastName`, поэтому работа приложения начинается с размещения символа ASCII 0 в конце целевого буфера для строки `szLastName`. Чтобы вычислить адрес назначения для символа завершения строки, достаточно взять длину строки `szLastName`, сложить ее с адресом, содержащимся в переменной `pszLastName`, и увеличить на 1.
3. К сожалению, этот код также основан на предположении, что вызов функции `malloc` не может окончиться неудачей. Но после неудачного завершения функция `malloc` возвращает значение `NULL`, которое сохраняется в переменной `pszLastName`. Затем этот адрес используется при вычислении адреса того байта, в который должен быть записан символ завершения строки. А поскольку этот адрес равен 0, то фактически предпринимается попытка поместить символ ASCII 0 по тому адресу памяти, равному длине строки, на которую указывает

переменная szLastName, плюс 1. Поэтому вместо простой ссылки на NULL создается ссылка на адрес 0x00000006, равный 5 (длина строки "Smith") + 1.

4. В этом можно легко убедиться, рассмотрев приведенный ниже результат преобразования кода приложения в код ассемблера (листинг 4.3).

Листинг 4.3. Результат преобразования кода приложения в код ассемблера

```
13:       char *pszLastName = (char *)malloc(LAST_NAME_SIZE);
00401028   push         7FFFFFFFh
0040102D   call         malloc (00401220)
00401032   add          esp,4
00401035   mov          dword ptr [ebp-4],eax
14:       *(pszLastName+strlen(szLastName)+1)='\0';
00401038   push         offset szLastName (00421a30)
0040103D   call         strlen (004011a0)
00401042   add          esp,4
00401045   mov          ecx,dword ptr [ebp-4]
00401048   mov          byte ptr [ecx+eax+1],0
```

а. Вызов функции malloc (строка 13) начинается с того, что в стек задвигается значение 0x7FFFFFFF. Оно представляет собой значение объявленной в программе константы LAST_NAME_SIZE: 2 147 483 647, или 2 Гбайт минус 1.

б. Значение, возвращаемое из функции malloc, содержится в регистре eax. Поскольку известно, что этот вызов должен окончиться неудачей, известно также, что это значение равно NULL, или 0x00000000. Указанное значение передается в переменную pszLastName непосредственно перед осуществлением попытки записать в память символ завершения строки.

в. Длина строки szLastName вычисляется в строке 14, это значение складывается с предыдущим значением, хранящимся в переменной pszLastName, плюс 1, а затем предпринимается попытка использовать новое значение как адрес (выполнить его разадресацию), чтобы с его помощью ввести в память символ завершения строки. Фактическая операция разадресации (являющаяся причиной последующей активизации ошибки нарушения доступа) обозначена в листинге 4.3 полужирным шрифтом.

5. Поскольку адрес 0x00000006 находится в пределах первых 64 Кбайт пространства адресов процесса, то после попытки выполнить его разадресацию активизируется ошибка нарушения доступа. В следующем упражнении показан случай, в котором ссылка на NULL-указатель появляется в результате перезаписи значения указателя. Подобная проблема часто возникает в приложениях, особенно на тех языках, в которых предусмотрены явно заданные операции с указателями (таких как C и C++).

Упражнение 4.3. Возникновение ссылки на NULL-указатель из-за перезаписи в области памяти

Ниже приведен довольно надуманный пример, который, тем не менее, позволяет еще раз продемонстрировать важность использования секции доступа к NULL-указателю.

1. Загрузите приложение, приведенное в листинге 4.4, с компакт-диска (см. каталог CH04\memexamp02) и откомпилируйте его.

Листинг 4.4. Появление ссылки на NULL-указатель, вызванное искажением содержимого области памяти, в которой находится указатель

```
// memexamp02.cpp. Пример формирования ссылки на NULL-указатель из-за
//                 искажения содержимого области памяти, в которой
//                 находится указатель
//

#include "stdafx.h"
#include "stdlib.h"
#include "string.h"

#define MAX_FIRST_NAME_SIZE 10
#define MAX_LAST_NAME_SIZE 30

#pragma pack(1)

struct NAME
{
  char szFirstName[MAX_FIRST_NAME_SIZE];
  char *pszLastName;
} nmEmployee;

int main(int argc, char* argv[])
{
  int dwFirstNameLen=__min(strlen(argv[1]),MAX_FIRST_NAME_SIZE);
  int dwLastNameLen=__min(strlen(argv[2]),MAX_LAST_NAME_SIZE);

  nmEmployee.pszLastName=(char *)malloc(dwLastNameLen);

  strncpy(nmEmployee.szFirstName,argv[1],dwFirstNameLen);
  strncpy(nmEmployee.pszLastName,argv[2],dwLastNameLen);

  nmEmployee.szFirstName[dwFirstNameLen+2]='\0';
  nmEmployee.pszLastName[dwLastNameLen+2]='\0';

  strupr(nmEmployee.pszLastName);

  printf("First Name=%s Last
    Name=%s\n",nmEmployee.szFirstName,nmEmployee.pszLastName);
  return 0;
}
```

Эта программа работает безукоризненно при условии, что передаваемый в нее первый параметр имеет длину 8 символов или меньше. Поскольку во всем этом приложении используются операции с указателями, подверженные ошибкам, и особенно потому, что строки с именами оканчиваются завершающим символом, применение параметра с указанием фамилии, имеющей длину больше 8 символов, приводит к перекрытию части указателя pszLastName символом ASCII 0.

2. Для того чтобы определить, что при этом происходит, задайте в качестве параметров командной строки (с помощью опций меню Alt+F7 | Debug | Program arguments) значение "Wolfgangus Mozart" (без кавычек).
3. Установите точку останова на строке, в которой присваивается символ завершения строки для переменной szFirstName:

```
nmEmployee.szFirstName[dwFirstNameLen+2]='\0';
```

4. Теперь вызовите это приложение на выполнение из интегрированной среды разработки (Integrated Development Environment — IDE) VC++. После того как отладчик остановится на указанной точке, добавьте структуру nmEmployee к переменным в окне Watch, затем разверните это окно таким образом, чтобы можно было видеть элементы структуры во время пошагового выполнения кода.
5. Нажмите клавишу <F10>, чтобы приступить к выполнению строки с точкой останова. В окне Watch вы должны обнаружить, что в результате выполнения данной строки изменилось не только значение szFirstName, но и значение pszLastName (оба элемента этой структуры должны быть обозначены в окне Watch красным цветом). Это связано с тем, что символ ASCII 0, предназначенный для записи в конце строки szFirstName, фактически был записан на 3 байта дальше конца этой строки. Поскольку строка szFirstName имеет длину 10 символов, а индексы массивов в C++ всегда начинаются с нуля, то допустимыми индексами для строки szFirstName являются 0–9. Однако значение dwFirstNameLen равно 10. Присваивание значения ASCII 0 элементу массива szFirstName[dwFirstNameLen] должно было бы также перекрыть часть значения pszLastName, но фактически перекрывает только первый байт этого четырехбайтового указателя. А после добавления 2 к этому смещению происходит сдвиг на третий байт указателя pszLastName. Зануление указанного байта приводит к тому, что адрес приобретает такое значение, которое соответствует первым 64 Кбайт пространства адресов процесса.
6. Теперь сделаем попытку перейти в режиме пошагового выполнения на следующую строку. Поскольку выполнение предыдущей строки привело к искажению значения указателя pszLastName, должна быть обнаружена ситуация нарушения доступа. Конкретная причина этого нарушения доступа состоит в том, что происходит ссылка на адрес в пределах первых 64 Кбайт памяти, а средства защиты секции доступа по NULL-указателю ОС Windows перехватывают эту недействительную ссылку.

Выше в этой главе было указано, что для получения сведений о размере страницы системы и степени детализации распределения можно воспользоваться вызовом функции GetSystemInfo API-интерфейса Win32. В следующем упражнении показано, как создать и вызвать на выполнение одну из расширенных процедур SQL Server, которая возвращает такую же информацию.

Упражнение 4.4. Расширенная хранимая процедура GetSystemInfo

1. Скопируйте проект xp_sysinfo из подкаталога CH04\xp_sysinfo компакт-диска, прилагаемого к данной книге, на жесткий диск и загрузите его в среду Visual C++. Для любознательных читателей в листинге 4.5 приведен полный исходный код расширенной процедуры xp_sysinfo.

Листинг 4.5. Расширенная процедура, которая возвращает информацию о памяти системы

```
RETCODE __declspec(dllexport) xp_sysinfo(SRV_PROC *srvproc)
{

  DBCHAR colname[MAXCOLNAME];
  DBCHAR szProcType[MAX_PATH];
  DBCHAR szMinAddress[MAXCOLNAME];
  DBCHAR szMaxAddress[MAXCOLNAME];
  DBCHAR szAffinityMask[MAXCOLNAME];
  SYSTEM_INFO si;
GetSystemInfo(&si);

  // Определить имена столбцов
  wsprintf(colname, "PageSize");
  srv_describe(srvproc, 1, colname, SRV_NULLTERM, SRVINT4,
      sizeof(DBINT), SRVINT4, sizeof(DBINT), &si.dwPageSize);

  wsprintf(colname, "AllocationGranularity");
  srv_describe(srvproc, 2, colname, SRV_NULLTERM, SRVINT4,
      sizeof(DBINT), SRVINT4, sizeof(DBINT),
       &si.dwAllocationGranularity);

  wsprintf(colname, "NumberOfProcessors");
  srv_describe(srvproc, 3, colname, SRV_NULLTERM, SRVINT4,
      sizeof(DBINT), SRVINT4, sizeof(DBINT),
      &si.dwNumberOfProcessors);

  wsprintf(colname, "ProcessorType");
  switch (si.wProcessorArchitecture)
  {
    case PROCESSOR_ARCHITECTURE_INTEL :
    {
      strcpy(szProcType,"Intel ");
      switch (si.wProcessorLevel)
      {
      case 3 :
         {
           strcat(szProcType,"386");
           break;
         }
       case 4 :
         {
           strcat(szProcType,"486");
           break;
         }
       case 5 :
         {
          strcat(szProcType,"Pentium");
           break;
         }
       case 6 :
         {
          strcat(szProcType,"Pentium II or Pentium Pro or later");
           break;
         }
```

```
case 7 :
  {
   strcat(szProcType,"Pentium III");
    break;
  }
case 8 :
  {
   strcat(szProcType,"Pentium 4");
    break;
  }
default :
  {
    strcat(szProcType,"Unknown");
    break;
  }

}
break;
}
case PROCESSOR_ARCHITECTURE_MIPS :
{
strcpy(szProcType,"MIPS ");
switch (si.wProcessorLevel)
{
case 4:
  {
   strcat(szProcType,"R4000");
    break;
  }
default:
  {
    strcat(szProcType,"Unknown");
    break;
  }
}
break;
}
case PROCESSOR_ARCHITECTURE_ALPHA :
{
strcpy(szProcType,"Alpha ");
 switch (si.wProcessorLevel)
 {
case 21064:
  {
    strcat(szProcType,"21064");
    break;
  }
 case 21066:
  {
    strcat(szProcType,"21066");
    break;
  }
 case 21164:
  {
    strcat(szProcType,"21164");
    break;
```

```
      }
    default:
      {
       strcat(szProcType,"Unknown");
        break;
      }
    }
   break;
    }
  case PROCESSOR_ARCHITECTURE_PPC :
    {
   strcpy(szProcType,"PPC ");
   switch (si.wProcessorLevel)
    {
    case 1:
      {
        strcpy(szProcType, "601");
        break;
      }
    case 3:
      {
        strcpy(szProcType, "603");
        break;
      }
    case 4:
      {
        strcpy(szProcType, "604");
        break;
      }
    case 6:
      {
        strcpy(szProcType, "603+");
        break;
      }
    case 9:
      {
        strcpy(szProcType, "604+");
        break;
      }
    case 20:
      {
        strcpy(szProcType, "620");
        break;
      }
    default:
      {
       strcat(szProcType,"Unknown");
        break;
      }
    }
   break;
   }
 default :
   {
   strcpy(szProcType,"Unknown ");
   break;
```

```
        }
    }
    srv_describe(srvproc, 4, colname, SRV_NULLTERM, SRVCHAR,
      strlen(szProcType), SRVCHAR, strlen(szProcType),
      &szProcType);

    wsprintf(colname, "ProcessorAffinityMask");
    wsprintf(szAffinityMask,"0x%08X",si.dwActiveProcessorMask);
    srv_describe(srvproc, 5, colname, SRV_NULLTERM, SRVCHAR,
      strlen(szAffinityMask), SRVCHAR, strlen(szAffinityMask),
      &szAffinityMask);

    wsprintf(colname, "MinimumAppAddress");
    wsprintf(szMinAddress,"0x%08X",si.lpMinimumApplicationAddress);
    srv_describe(srvproc, 6, colname, SRV_NULLTERM, SRVCHAR,
      strlen(szMinAddress), SRVCHAR, strlen(szMinAddress),
      &szMinAddress);
  wsprintf(colname, "MaximumAppAddress");
  wsprintf(szMaxAddress,"0x%08X",si.lpMaximumApplicationAddress);
  srv_describe(srvproc, 7, colname, SRV_NULLTERM, SRVCHAR,
      strlen(szMaxAddress), SRVCHAR, strlen(szMaxAddress),
      &szMaxAddress);

  wsprintf(colname, "UserModeAddressSpace");
  DWORD dwUserModeSpace = ((DWORD)si.lpMaximumApplicationAddress -
      (DWORD)si.lpMinimumApplicationAddress);
  srv_describe(srvproc, 8, colname, SRV_NULLTERM, SRVINT4,
      sizeof(DBINT), SRVINT4, sizeof(DBINT), &dwUserModeSpace);

  srv_sendrow(srvproc);

  // Теперь возвратить количество обработанных строк
  srv_senddone(srvproc, SRV_DONE_MORE | SRV_DONE_COUNT,
      (DBUSMALLINT)0, 1);

    return XP_NOERROR;

}
```

2. Откомпилируйте данный проект. Выполнение указанного действия должно привести к получению библиотеки DLL с именем `xp_sysinfo.dll`, находящейся в подкаталоге `Release` корневого каталога `xp_sysinfo`.
3. Скопируйте файл `xp_sysinfo.dll` в подкаталог `binn` корневого инсталляционного каталога SQL Server. Если вами были выполнены упражнения из предыдущих глав, то на экране может появиться вопрос, нужно ли перезаписать существующий файл `xp_sysinfo`. Ответьте `Yes` на этот вопрос.
4. Добавьте эту расширенную процедуру к основной базе данных с помощью следующей команды:

   ```
   sp_addextendedproc 'xp_sysinfo','xp_sysinfo.dll'
   ```

5. Вызовите исполняемый файл `xp_sysinfo` на выполнение из программы Query Analyzer. При этом должен быть получен примерно такой вывод (часть результатов исключена):

```
PageSize  AllocGranularity Processors ProcessorType     AffinityM
--------- ----------------- ---------- ---------------- ---------
4096      65536             2           Intel Pentium... 0x0000000
```

Вполне очевидно, что размер страницы системы равен 4 Кбайт, а степень детализации распределения — 64 Кбайт. Обратите внимание то, что на других процессорах или в будущих версиях Windows эти цифры могут измениться.

Заслуживает также внимания столбец `UserModeSpace` выше (не показан). Предположим, что на рассматриваемом компьютере указанный в этом столбце максимальный объем пространства непривилегированного режима приблизительно равен 2 Гбайт. Это означает, что попытка разрешить применение опции начальной загрузки `/3GB` либо не предпринималась, либо не была выполнена успешно. Программа SQL Server представляет собой приложение с поддержкой большого пространства адресов, поэтому полученные результаты показали бы, что объем пространства адресов непривилегированного режима примерно равен 3 Гбайт, только если бы этот сервер эксплуатировался в соответствующей версии Windows, а начальная загрузка системы была выполнена с опцией `/3GB`.

Основы функционирования памяти. Резюме

В операционной системе Windows предусмотрен богатый набор средств, упрощающий решение задачи предоставления памяти приложениям. Даже если на компьютере установлен относительно небольшой объем физической оперативной памяти, ОС Windows предоставляет каждому процессу пространство виртуальных адресов с объемом 4 Гбайт, в котором выполняется это приложение, и обеспечивает закрепление страниц физической памяти по мере необходимости за жестким диском и отмену закрепления прозрачно для приложения.

В семействе процессоров x86 применяется размер страницы памяти, равный 4 Кбайт. Это означает, что все операции распределения памяти в ОС Windows фактически осуществляются с учетом значений, кратных 4 Кбайт. Например, для выполнения запроса о распределении 5 Кбайт фактически требуется 8 Кбайт памяти.

Средства AWE и опция `/3GB` предоставляют приложениям механизмы доступа к памяти, выходящей за пределы стандартной секции непривилегированного режима с объемом 2 Гбайт. Опция `/3GB` фактически ограничивает суммарный объем физической памяти, которой может управлять операционная система Windows, поэтому ее применение, как правило, не рекомендуется. Средства AWE представляют собой наиболее гибкий подход из этих двух подходов к управлению памятью и позволяют предоставить приложениям доступ ко всей физической памяти, которая находится в распоряжении операционной системы.

Основы функционирования памяти. Вопросы для самопроверки

1. Чему равен размер страницы системы в семействе процессоров x86?
2. Какой величиной определяется степень детализации распределения в 32-битовых версиях Windows?

3. Подтвердите или опровергните следующее утверждение. Ошибка отсутствия страницы приводит к активизации исключительной ситуации, которая вызывает аварийное завершение приложения, если эта исключительная ситуация не перехватывается в приложении с помощью кода структурированной обработки исключительных ситуаций (Structured Exception Handling – SEH).
4. Если опция `/3GB` будет разрешена в версии Windows 2000 Professional, то какой объем пространства адресов непривилегированного режима будет распределен в процессе SQL Server во время его запуска?
5. Подтвердите или опровергните следующее утверждение. Операцией преобразования адресов называется двухэтапная операция, в который для преобразования виртуального адреса в физический используются два компонента виртуального адреса – индекс таблицы страниц и смещение страницы.
6. Подтвердите или опровергните следующее утверждение. Пробуксовкой называется ситуация, в которой непрерывно происходит обмен страницами в прямом и обратном направлениях между физической памятью и системным файлом подкачки, что часто влечет за собой существенное замедление работы приложений.
7. Какая область адресов резервируется в операционной системе Windows для того, чтобы в приложениях можно было проще обнаруживать присваивания NULL-указателей?
8. Каков объем применяемого по умолчанию пространства непривилегированного режима в процессе, поддерживаемом в 32-битовой версии Windows?
9. Каким общим объемом физической памяти способна управлять версия Windows 2000 Data Center?
10. Подтвердите или опровергните следующее утверждение. Использование функций AWE приводит к тому, что пространство привилегированного режима сокращается до такой степени, что операционная система Windows имеет возможность управлять лишь общим объемом физической памяти, равным 16 Гбайт.
11. Какой параметр редактора связей VC++ позволяет создавать приложения, способные воспользоваться большим пространством адресов?
12. Какое количество битов виртуального адреса можно было использовать в приложении непривилегированного режима для формирования непосредственных ссылок на виртуальную память до того, как в ОС Windows появились средства поддержки опции начальной загрузки `/3GB`?
13. Подтвердите или опровергните следующее утверждение. Системный файл подкачки может фактически состоять из нескольких физических файлов, которые разрешается размещать на разных жестких дисках.
14. Какая информация о процессе находится в столбце `Mem Usage` программы Task Manager?
15. Что происходит при попытке выполнить в непривилегированном режиме операцию преобразования адресов с недействительным адресом?

16. Какая функция API-интерфейса Windows, описанная в этой главе, позволяет узнать и размер страницы системы, и степень детализации распределения системы?
17. Подтвердите или опровергните следующее утверждение. Для блока PEB не распределяется память по конкретному адресу в пространстве виртуальных адресов процесса, и в различных процессах он почти всегда находится в разных местах.
18. Подтвердите или опровергните следующее утверждение. Во всех микросхемах процессоров, поддерживаемых операционной системой Windows, предусмотрены встроенные средства защиты памяти того или иного типа.
19. Чем отличается содержимое счетчиков `Process:Private Bytes` и `Process:Page File Bytes` программы Perfmon?
20. Подтвердите или опровергните следующее утверждение. Поскольку Windows представляет собой 32-битовую операционную систему, всем процессам непривилегированного режима предоставляется линейное пространство адресов с объемом 4 Гбайт.
21. Какая команда отладчика WinDbg применяется для вывода на экран содержимого блока PEB процесса?
22. Что показывает столбец `VM Size` программы Task Manager, который относится к некоторому процессу?
23. Какая функция API-интерфейса Win32, описанная в этой главе, позволяет определить, не требуется ли процессу больший объем пространства адресов непривилегированного режима?
24. Подтвердите или опровергните следующее утверждение. Поскольку наибольшее целое число, которое может храниться в 32-битовом указателе, равно 2^{32}, то максимальный объем памяти, к которому может получить доступ приложение непривилегированного режима, равен 4 Гбайт.
25. Подтвердите или опровергните следующее утверждение. Основной объем памяти физического запоминающего устройства, используемый для реализации виртуальной памяти, принадлежит к физической оперативной памяти, установленной на компьютере.
26. Какой регистр специального назначения используется для хранения информации о местонахождении каталога страниц в процессорах x86?
27. Подтвердите или опровергните следующее утверждение. Если для процесса требуется больший объем памяти по сравнению со стандартным объемом пространства виртуальной памяти, равным 2 Гбайт, обычно лучше воспользоваться средствами AWE, а не опцией `/3GB`.
28. Что представляет собой буфер быстрого преобразования адреса (Translation Look-aside Buffer – TLB)?
29. Подтвердите или опровергните следующее утверждение. Совместно используемая страница данных непривилегированного режима фактически представляет собой копию одной из страниц, находящихся в области памяти привилегированного режима.

30. Подтвердите или опровергните следующее утверждение. В приложении можно указать размеры распределения памяти, которое должно быть в нем выполнено, но нельзя указать точное местонахождение в памяти для этого распределения.

Виртуальная память

В приложениях, эксплуатируемых в операционной системе Windows, могут использоваться три различных типа памяти: виртуальная память, динамические области памяти и совместно используемая память. Виртуальная память в наибольшей степени приспособлена для управления большими массивами или коллекциями объектов и структур с различными размерами. Кроме того, виртуальная память представляет собой основной механизм, с помощью которого программа SQL Server распределяет память; описанию этого механизма в основном посвящен данный раздел.

Для распределения виртуальной памяти используются функции `VirtualAllocEx` и `VirtualAlloc` API-интерфейса. Функция `VirtualAlloc` распределяет память только в пространстве адресов вызывающего процесса, а функция `VirtualAllocEx` позволяет распределять память в пространстве адресов другого процесса. Из этих двух функций значительно более широкое распространение находит функция `VirtualAlloc` и именно эта функция будет применяться во всей данной главе.

Страницы в виртуальной памяти всегда находятся в одном из трех состояний: свободная, зарезервированная или закрепленная (отображенная на физическую память). Для резервирования и/или закрепления страниц виртуальной памяти используется функция `VirtualAlloc`, а для отмены резервирования и/или закрепления страниц распределенной памяти — функция `VirtualFree`. Освобожденная память не является зарезервированной или закрепленной — она свободна.

Виртуальная память. Основные термины и определения

- **Размер страницы.** Размер страницы памяти, который поддерживается архитектурой данного конкретного процессора. На процессоре x86 он равен 4 Кбайт. Все операции распределения памяти в ОС Windows должны осуществляться с учетом значений, кратных размеру страницы системы.
- **Степень детализации распределения.** Граница, по которой должны выполняться операции резервирования виртуальной памяти в ОС Windows. Во всех современных версиях Windows эта величина равна 64 Кбайт, поэтому операции резервирования виртуальной памяти в непривилегированном режиме должны осуществляться с учетом границ 64 Кбайт в пространстве адресов процесса.
- **Зарезервированная память.** Область адресов виртуальной памяти, которая отведена для использования в процессе. Для зарезервированной области

не требуется реальная память. Операции резервирования памяти всегда должны выполняться с учетом границ, определяющих степень детализации распределения. Доступ к зарезервированной памяти нельзя получить до тех пор, пока она не будет закреплена.

- **Закрепленная память.** Область виртуальной памяти, которой поставлена в соответствие реальная память.
- **Копирование при записи.** Средство Windows, при использовании которого попытки внесения изменений на какой-то странице вызывают дублирование этой страницы и внесение этих изменений на новой, а не на старой странице. Этот механизм применяется, например, если эксплуатируются несколько экземпляров приложения и один из них вносит изменения в одну из страниц данных (например, изменяет значение глобальной переменной).
- **Защищенная страница.** Страница, которая была обозначена атрибутом защиты страницы `PAGE_GUARD`. При первой попытке со стороны какого-то процесса получить доступ к защищенной странице в операционной системе Windows эта операция доступа оканчивается неудачей и происходит либо активизация исключительной ситуации `STATUS_GUARD_PAGE`, либо возврат последнего кода ошибки `STATUS_GUARD_PAGE_VIOLATION`. Кроме того, при этом сбрасывается состояние защиты страницы, поэтому следующая попытка получить к ней доступ завершается успешно.

Виртуальная память. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к виртуальной памяти, приведены в табл. 4.6.

Таблица 4.6. Основные функции API-интерфейсов, относящиеся к виртуальной памяти

Функция	Описание
`VirtualAlloc`	Зарезервировать, закрепить (отобразить на реальную память) и переопределить виртуальную память
`VirtualFree`	Отменить закрепление и освободить виртуальную память
`VirtualProtect`	Изменить атрибуты защиты страницы для ряда страниц виртуальной памяти
`VirtualLock/VirtualUnlock`	Заблокировать/разблокировать страницы виртуальной памяти в физической памяти
`VirtualQuery(Ex)`	Возвратить информацию системного уровня об области виртуальной памяти
`SetWorkingSetSize`	Задать количество страниц виртуальной памяти, которые могут быть заблокированы процессом в физической памяти
`GetSystemInfo`	Получить информацию системного уровня о таких ресурсах компьютера, как количество процессоров и объем памяти

Атрибуты защиты страницы

При выполнении в приложении операции распределения виртуальной памяти с помощью функции `VirtualAlloc` операционная система Windows позволяет распределяющему процессу задавать атрибуты защиты для ряда распределенных страниц. После этого указанные атрибуты передаются в аппаратные средства управления памятью системы, а эти средства позволяют их реализовать. В языке Visual C++ указанные атрибуты задаются путем применения различных комбинаций констант `PAGE_`. В табл. 4.7 перечислены все константы атрибутов и указано их назначение.

Таблица 4.7. Атрибуты защиты страницы

Атрибут защиты	Назначение
`PAGE_GUARD`	Обеспечивает то, что попытка доступа к странице приводит к активизации исключительной ситуации `STATUS_GUARD_PAGE` (или к возврату ошибки `STATUS_GUARD_PAGE_VIOLATION`) и сбросу защиты защищенной страницы. Этот атрибут не может применяться в сочетании с атрибутом `PAGE_NOACCESS`
`PAGE_EXECUTE`	Предотвращает попытки выполнить запись на странице
`PAGE_EXECUTE_READ`	Предотвращает попытки выполнить запись на странице
`PAGE_EXECUTE_READ_WRITE`	Позволяет успешно осуществить любые попытки доступа
`PAGE_EXECUTE_WRITECOPY`	Вынуждает систему при попытке выполнить операцию записи на странице скопировать страницу и предоставить процессу новую копию. Выполнение кода, находящегося на странице, разрешается
`PAGE_NOACCESS`	Предотвращает любые попытки доступа к странице
`PAGE_NOCACHE`	Предотвращает возможность кэширования страницы. Этот атрибут не рекомендуется для общего использования. Предназначен главным образом для драйверов устройств
`PAGE_READONLY`	Предотвращает попытки выполнить запись на странице
`PAGE_READWRITE`	Позволяет успешно осуществить любые попытки доступа
`PAGE_WRITECOMBINE`	Вынуждает систему объединять несколько операций записи на одном устройстве в одну операцию в целях повышения производительности. Используется главным образом в драйверах устройств
`PAGE_WRITECOPY`	Вынуждает систему при попытке выполнить операцию записи на странице скопировать страницу и предоставить процессу новую копию. Выполнение кода, находящегося на странице, не разрешается

ПРИМЕЧАНИЕ. Доступ только для выполнения не поддерживается семейством процессоров x86. Если речь идет о семействе процессоров x86, то страница, предназначенная для чтения, является также предназначенной для выполнения. Это означает, например, что атрибуты защиты `PAGE_EXECUTE` и `PAGE_READONLY` являются функционально эквивалентными, если операционная система Windows эксплуатируется на компьютере с процессором x86.

Нередко в процедурах распределения памяти атрибут защиты PAGE_GUARD используется для обнаружения ситуаций переполнения буфера. Применяемый при этом способ обнаружения указанных ситуаций состоит в том, что после выполнения каждой операции распределения процедура распределения памяти выделяет еще одну страницу с установленным атрибутом PAGE_GUARD (т.е. защищенную страницу) сразу вслед за вновь распределенной областью. Если после этого в операциях доступа к памяти предпринимаются попытки выполнить запись за пределами конца распределенной области, то активизируется исключительная ситуация STATUS_GUARD_PAGE, и попытка доступа оканчивается неудачей.

Во время запуска процесса операционная система Windows устанавливает защиту страниц с кодом исполняемого модуля этого процесса с помощью атрибута PAGE_EXECUTE_READ. Это позволяет совместно использовать одну и ту же область реальной памяти для многочисленных копий одного и того же исполняемого модуля. Например, если на компьютере эксплуатируется несколько экземпляров программы Explorer, то на память физически отображается лишь одна копия исполняемого файла `explorer.exe`.

Копирование при записи

Как было указано выше, в нескольких экземплярах приложения для страниц кода исполняемого модуля этого приложения совместно применяется одна и та же реальная память, но хранение страниц данных для этих экземпляров организовано иначе. Дело в том, что если страницы данных для всех экземпляров исполняемого приложения отображаются на одну и ту же реальную память, то в одном экземпляре приложения невозможно изменить, скажем, глобальную переменную без того, чтобы это изменение не отразилось на работе других экземпляров. Поэтому необходимо найти способ разделения страниц данных разных экземпляров. Для этого применяется атрибут защиты страницы PAGE_WRITECOPY. После запуска процесса операционная система Windows защищает его страницы данных с помощью атрибута PAGE_WRITECOPY, а при модификации в процессе одной из его страниц данных процесс получает собственную закрытую копию этой страницы. Такое функциональное средство называется *копированием при записи*. Оно позволяет уменьшить потребность в реальной памяти и вносить изменения в глобальные данные в отдельных экземплярах приложения, не оказывая влияния на другие экземпляры.

Функциональные средства копирования при записи всегда поддерживались в семействе операционных систем Windows NT (Windows NT, Windows 2000, Windows XP и Windows Server 2003). Определенные типы функциональных средств копирования при записи поддерживаются также в большинстве разновидностей UNIX. Тем не менее в некоторых операционных системах (таких как Windows 9x и OpenVMS) этого не предусмотрено. Если в операционной системе не предусмотрены функциональные средства копирования при записи, то обычно принято создавать закрытую копию всех страниц данных приложения при первоначальном запуске процесса. Очевидно, что подход, принятый в семействе Windows NT, является гораздо более эффективным.

В семействе Windows NT система всегда распределяет пространство в системном файле подкачки для размещения страниц данных любого приложения. Но место в реальной памяти, зарезервированное для каждой страницы данных, фактически не используется до тех пор, пока не будет выполнена запись на данную страницу. Это позволяет экономить место в реальной памяти и вместе с тем гарантировать, что приложение всегда будет иметь возможность выполнять запись данных на своих страницах, когда это потребуется.

ПРЕДОСТЕРЕЖЕНИЕ. Не передавайте атрибут `PAGE_WRITECOPY` или `PAGE_EXECUTE_WRITECOPY` при вызове функции `VirtualAlloc` для резервирования или закрепления памяти. Если это условие не соблюдается, то операция распределения завершается с ошибкой `ERROR_INVALID_PARAMETER`. Указанные атрибуты защиты страницы зарезервированы для использования только системой.

Резервирование памяти

В операционной системе Windows разрешается резервировать пространство адресов памяти в любом процессе, поскольку фактически выполнение таких операций не приводит к расходованию каких-либо закрепленных страниц и не уменьшает квоту в файле подкачки для рассматриваемого процесса (эта квота не обязательно выражается как какая-то часть файла подкачки, а, скорее, обозначает максимальное количество закрепленных страниц, которое может быть израсходовано в процессе). Выполнение операции резервирования виртуальной памяти приводит лишь к тому, что приложению передается непрерывный блок пространства адресов процесса; при этом фактически данному приложению не предоставляется доступ хоть к какой-либо части вновь распределенной памяти. Прежде чем в приложении можно будет использовать память, ее необходимо закрепить за реальной памятью.

Большинство разработчиков не задумывается над тем, что они в своих приложениях могут не только предусмотреть распределение памяти, но и указать точный адрес, по которому будет зарезервирована распределяемая область памяти. Вообще говоря, такие средства распределения памяти, как функция `malloc` и оператор `new` языка C++, не позволяют указывать в приложении, где должна быть распределена память; они дают возможность задавать в приложении лишь размер распределения. Но операционная система Windows позволяет разработчику получить контроль над обоими аспектами распределения, что может повлечь за собой важные последствия с точки зрения того, как должен разрабатываться код приложения. Это утверждение будет вскоре подтверждено в данной главе на практическом примере.

Если речь идет об операционной системе Windows, то операции резервирования памяти являются не очень дорогостоящими, поскольку при резервировании памяти обновляются лишь относительно небольшие дескрипторы виртуального адреса (Virtual Address Descriptor – VAD), предназначенные для рассматриваемого процесса. Эта операция обычно выполняется очень быстро, поскольку фактически процессу не передается физическая память, а квота страниц процесса не изменяется.

Предусмотренный в ОС Windows двухэтапный подход к распределению виртуальной памяти используется в самой этой операционной системе. В качестве одного из самых ярких примеров можно указать тот способ, с помощью которого распределяется пространство стека для потока. При создании потока операционная система Windows резервирует область виртуальной памяти для хранения стека потока. Эта область по умолчанию равна 1 Мбайт; данное значение можно переопределить для отдельного потока, указав иное значение размера стека в вызове функции `CreateThread`, или для всех потоков, с помощью параметра `/STACK` редактора связей (параметр, применяемый в компиляторе Visual C++; в большинстве других компиляторов поддерживается аналогичная опция) или с использованием таких инструментальных средств, как ImageCfg, которые позволяют редактировать заголовок исполняемого файла.

Даже после того как операционная система Windows резервирует полное пространство стека для рассматриваемого потока, эта ОС фактически ожидает запросов на закрепление страниц из данной области, после того как они действительно потребуются. А после поступления такого запроса ОС Windows вначале передает лишь одну страницу из зарезервированной области и отмечает страницу, которая непосредственно следует за ней, как защищенную. При попытке приложения распространить область стека на защищенную страницу операционная система Windows перехватывает возникающую при этом исключительную ситуацию `STATUS_GUARD_PAGE` и расширяет область стека, передавая защищенную страницу (а само состояние защиты страницы сбрасывается в результате попытки доступа). После этого Windows отмечает страницу, расположенную вслед за вновь закрепленной страницей, как защищенную и предоставляет потоку возможность продолжить свое выполнение. Такие действия продолжаются до тех пор, пока не достигается конец области, первоначально зарезервированной для стека потока. Это позволяет гарантировать, что область адресов, используемая стеком потока, остается непрерывной, а реальная память не применяется до тех пор, пока в этом действительно не возникает необходимость.

Как было указано выше, степень детализации распределения в 32-битовых версиях Windows равна 64 Кбайт. В приложении всегда следует резервировать память в виде фрагментов, кратных величине степени детализации распределения, поскольку незарезервированное пространство адресов, оставшееся после резервирования только части области с объемом 64 Кбайт, остается недоступным для запросов на распределение в непривилегированном режиме. Учитывая то, что начальный адрес любого переданного системе запроса на резервирование округляется в меньшую сторону до ближайшей границы распределения, в приложении не существует способа вынудить операционную систему Windows зарезервировать оставшееся неиспользуемым пространство адресов. Со временем это может привести к исчерпанию пространства адресов процесса, даже несмотря на наличие огромного объема доступной реальной памяти, что может стать причиной возникновения катастрофических проблем для приложения. Например, если объем непрерывного пространства адресов становится меньше 1 Мбайт, то в большинстве приложений исключается возможность создавать новые потоки, поскольку по умолчанию размер стека потока равен 1 Мбайт. (Для программы

SQL Server применяемый по умолчанию размер стека потока уменьшен до 0,5 Мбайт, но все еще существует вероятность того, что пространство адресов виртуальной памяти будет израсходовано до такой степени, что создание новых рабочих потоков станет невозможным.)

Даже несмотря на то, что в операционной системе Windows требуется, чтобы запросы на распределение памяти в непривилегированном режиме начинались на границе степени детализации распределения, к операциям распределения в привилегированном режиме такие требования не предъявляются. В системных операциях распределения обычной и нормальной является такая ситуация, что область, предназначенная для хранения блока PEB и блоков TEB процесса, распределяется не на границе области с объемом 64 Кбайт.

Закрепление виртуальной памяти

Прежде чем появится возможность использовать виртуальную память, ее нужно закрепить (отобразить на реальную память). Попытка доступа к памяти, которая была только зарезервирована, но не закреплена, вызывает нарушение доступа. Для закрепления области виртуальной памяти достаточно вызвать функцию `VirtualAlloc` с параметром `MEM_COMMIT`. Закрепление памяти может осуществляться одновременно с ее резервированием; для этой цели можно также использовать отдельный вызов функции `VirtualAlloc`. Если необходимо одновременно и зарезервировать, и закрепить область памяти, то в вызове функции `VirtualAlloc` применяется побитовый оператор `OR` для объединения констант `MEM_RESERVE` и `MEM_COMMIT` примерно следующим образом:

```
pBuf=(Buf *)VirtualAlloc(NULL, 65536, MEM_RESERVE|MEM_COMMIT,
    PAGE_READWRITE);
```

А если закрепление страниц из зарезервированной области выполняется в виде отдельной операции, то нет необходимости закреплять все эти страницы одновременно. Для закрепления можно выбирать отдельные страницы из зарезервированной области. Это позволяет легко внедрять в свои приложения разреженно заполняемые структуры данных, которые объединяют в себе преимущества блоков непрерывных адресов с эффективностью распределения памяти на физическом устройстве хранения только по мере необходимости. Наглядным примером разреженной структуры данных такого типа является пул буферов программы SQL Server. Все необходимое пространство для этого пула резервируется при запуске процесса, а отдельные страницы передаются по мере того, как в этом возникает необходимость.

Размер области в любом запросе на закрепление округляется в *б*ольшую сторону до ближайшей границы страницы. Например, при попытке закрепить область с объемом 10 Кбайт параметр запроса округляется в *б*ольшую сторону до 12 Кбайт. Ниже приведен пример запроса на закрепление виртуальной памяти.

```
// Начать с резервирования буфера с объемом 64 Кбайт
pBuf=(Buf *)VirtualAlloc(NULL, 65536, MEM_RESERVE,
     PAGE_READWRITE);

...
```

```
// Закрепить вторую страницу ранее зарезервированного буфера
VirtualAlloc((void *)(pBuf+4096), 4096, MEM_COMMIT,
    PAGE_READWRITE);
```

Инициализация и модификация страниц

Если закрепленная страница является закрытой и к ней еще не осуществлялся доступ, она создается при первом доступе к ней как страница, инициализированная нулями (называемая также *страницей, обнуляемой по требованию*). Это означает, что каждая страница виртуальной памяти поступает в распоряжение приложения, будучи заполненной нулями.

Операционная система Windows автоматически записывает закрытую закрепленную страницу, которая была модифицирована приложением, на жесткий диск, как только в этом возникает необходимость в связи с увеличением потребностей в ресурсах системной памяти. В операционной системе Windows закрепленные страницы записываются на жесткий диск с использованием обычного процесса записи модифицированных страниц, в котором страницы передаются из рабочего набора системы в список модифицированных страниц, а затем на диск. В приложении можно также предусмотреть немедленную запись отображенных на файл страниц на жесткий диск с помощью вызова функции `FlushViewOfFile` API-интерфейса Win32. Безусловно, операционная система Windows автоматически загружает страницы с жесткого диска в физическую память по мере необходимости.

Если страница в приложении модифицируется, но требуется, чтобы операционная система Windows рассматривала ее как немодифицируемую, и она не записывалась по принципу страничного обмена на диск, когда в этом возникнет необходимость в связи с увеличением потребности в памяти системы, можно вызвать функцию `VirtualAlloc` с параметром `MEM_RESET`. Такой вызов сообщает ОС Windows, что не нужно сохранять данные, находящиеся на этой странице, после того, как система определит, что физическая память, занимаемая страницей, требуется для удовлетворения других запросов к памяти. Если же сбрасывается содержимое страницы, которая в настоящее время находится в системном файле подкачки, эта страница уничтожается. Кроме того, если страница находится в физической памяти, она отмечается как немодифицированная, чтобы система могла просто ее перезаписать после того, как потребуется данная конкретная страница памяти. А при выполнении следующей операции доступа к странице она заполняется нулями, как будто доступ к ней происходит впервые. В коде, структурированном должным образом, можно использовать информацию о том, что страница после сброса заполняется нулями, чтобы определить, нужно ли снова заполнить эту страницу данными.

Применение подхода, в котором ненужные страницы сбрасываются, позволяет повысить производительность приложения, поскольку при этом исключается необходимость записывать модифицированные страницы в системный файл подкачки, тогда как этого в действительности не требуется. Безусловно, можно также просто отменять закрепление одной или нескольких страниц, что приведет к тому же эффекту. Но применение метода, в котором предусматривается сброс, а не отмена закрепления, позволяет исключить издержки, связанные с пе-

редачей новой страницы. Страница остается закрепленной, но заполняется нулями при выполнении следующей операции доступа к ней.

Следует отметить, что в функции VirtualAlloc округление параметров базового адреса и размера распределения при передаче параметра MEM_RESET происходит иначе. Обычно в этой функции округление базового адреса осуществляется в меньшую сторону до ближайшей границы страницы или границы степени детализации распределения, а размер распределения округляется в большую сторону до ближайшего целого количества страниц. А если в функцию VirtualAlloc передается параметр MEM_RESET, то округление базового адреса осуществляется в большую сторону до ближайшей границы страницы, а размер распределения округляется в меньшую сторону до ближайшей границы страницы. Такая организация выполнения функции позволяет исключить возможность случайно сбросить страницу. Если сбрасывать страницу нужно, то она должна быть полностью охвачена той областью, которая передается в функцию VirtualAlloc; страницы, не охваченные полностью этой областью, не сбрасываются.

Освобождение памяти

Закрепленные страницы являются либо закрытыми (не предназначенными для совместного доступа), либо отображенными на совместно используемую память. После того как страница была закреплена, приложение получает полное право осуществлять к ней доступ. В приложении можно вызывать функцию VirtualProtect для изменения атрибутов защиты страницы, VirtualLock для блокировки страницы в физической памяти и VirtualFree для освобождения страницы.

Функцию VirtualFree можно вызывать для освобождения области на устройстве памяти, которая была закреплена для некоторого блока адресов, не отменяя резервирование этого блока адресов. Такая операция называется *отменой закрепления*. При отмене закрепления можно указать, какая часть закрепленной области должна быть освобождена. Можно также освободить ряд адресов, связанных с распределением. Такая операция называется *освобождением*. При освобождении блока адресов виртуальной памяти нельзя указывать только часть освобождаемой области — освобождается либо вся область, либо она вообще не освобождается. Ниже приведены некоторые примеры.

```
// Начать с резервирования буфера с объемом 64 Кбайт
pBuf=(Buf *)VirtualAlloc(NULL, 65536, MEM_RESERVE,
     PAGE_READWRITE);

...

// Закрепить вторую страницу ранее зарезервированного буфера
VirtualAlloc((void *)(pBuf+4096), 4096, MEM_COMMIT,
     PAGE_READWRITE);

...

// Отменить закрепление только что закрепленной страницы
VirtualFree((void *)(pBuf+4096), 4096, MEM_DECOMMIT);
```

```
...

// Освободить всю ранее зарезервированную память
VirtualFree((void *)(pBuf),0,MEM_RELEASE);
```

При отмене закрепления области памяти можно не задумываться над тем, какие страницы этой области фактически были закреплены, а какие остались лишь зарезервированными. Отмена закрепления незакрепленной страницы не вызывает ошибки.

Блокировка страниц в памяти

По умолчанию в процессе устанавливается предел, согласно которому он может заблокировать в памяти не больше 30 страниц. Если же в приложении необходимо заблокировать в физической памяти большее количество страниц, то необходимо вначале вызвать функцию `SetWorkingSetSize` API-интерфейса для увеличения размера рабочего набора процесса. Именно эта функция API-интерфейса вызывается в программе SQL Server, если разрешена дополнительная опция конфигурации, определяющая размер рабочего набора. Для разблокирования страницы, заблокированной в физической памяти, в приложении вызывается функция `VirtualUnlock`.

Следует отметить, что страницы, заблокированные в физической памяти с помощью функции `VirtualLock`, все еще могут быть переписаны на жесткий диск по методу страничного обмена при некоторых обстоятельствах. Если все потоки в процессе находятся в состоянии ожидания из-за нехватки памяти, то операционная система Windows получает право исключать подобные страницы из рабочего набора системы, что в конечном итоге приводит к записи этих страниц на жесткий диск, если они были модифицированы.

Упражнения

В следующем упражнении показано, что происходит при резервировании виртуальной памяти в приложении. В ходе его выполнения вы узнаете, каким образом в функции `VirtualAlloc` происходит округление параметров запросов на распределение до границ страниц, а также ознакомитесь с тем, как значение степени детализации распределения, предусмотренное в системе, влияет на выполнение операций резервирования виртуальной памяти.

Упражнение 4.5. Исследование процесса резервирования виртуальной памяти

1. Скопируйте проект `vm_reserve` из подкаталога `CH04\vm_reserve` компакт-диска, прилагаемого к данной книге, на локальный жесткий диск и загрузите его в среду разработки VC++. Еще один вариант состоит в том, что можно скопировать исполняемый файл из подкаталога `Release` и выполнить его отдельно, если вы не заинтересованы в том, чтобы создать это приложение в среде VC++.
2. Начнем с анализа исходного кода программы `vm_reserve` (листинг 4.6).

Листинг 4.6. Простое приложение, в котором резервируется область памяти

```
int main(int argc, char* argv[])
{

  void *pv=VirtualAlloc(
    (void *)0x7FF01000,
    4096,
    MEM_RESERVE,
    PAGE_READWRITE);

  if (pv)  {
    MEMORY_BASIC_INFORMATION mbi;
    DWORD dwLen=sizeof(MEMORY_BASIC_INFORMATION);

  VirtualQueryEx(GetCurrentProcess(),pv,&mbi,dwLen);
    printf("%d bytes reserved at 0x%08X.\n",
      mbi.RegionSize,pv);
  }
  else printf("Error reserving mem %d.\n",
      GetLastError());

  return 0;
}
```

3. Это приложение начинается с вызова функции `VirtualAlloc` для резервирования буфера с объемом 4 Кбайт по конкретному адресу в памяти. Подход, в котором предусмотрено резервирование или закрепление памяти по конкретному адресу, непривычен для большинства разработчиков Windows, но относится к числу тех средств, которые всегда поддерживались в подсистеме Win32. Если затребованный адрес не находится на границе страницы (4 Кбайт на процессоре x86) или на границе степени детализации распределения (64 Кбайт в 32-битовой версии Windows), то система округляет этот адрес, чтобы обеспечить его правильное выравнивание.

4. Затем в приложении вызывается функция `VirtualQueryEx` для определения данных о размере вновь зарезервированной области, которые после этого отображаются на экране. В рассматриваемом приложении используется функция `VirtualQueryEx`, а не функция `VirtualQuery`, поскольку в некоторых ситуациях функция `VirtualQuery` может возвращать неточную информацию в системах с огромными объемами памяти.

5. Выполните это приложение и сравните его вывод со значением параметра, заданным в первоначальном запросе `VirtualAlloc`. Отличия должны наблюдаться в двух элементах этого вывода. Полученные данные должны выглядеть примерно так, как показано ниже.

   ```
   8192 bytes reserved at 0x7FF00000.
   ```

6. Прежде всего заслуживает внимания то, что зарезервировано 8192 байта, а не 4096. Почему так произошло? Это связано с тем, что заданные значения начального адреса и размера распределения вынудили операционную систему применить операцию резервирования, которая охватывает две страницы в пространстве адресов виртуальной памяти. Как было указано выше, заданные

значения начального адреса для резервирования всегда округляются в меньшую сторону до ближайшей границы степени детализации распределения. А если размер резервирования требует, чтобы эта операция распространилась еще на одну страницу, то размер резервирования округляется в большую сторону, до границы следующей страницы. Поэтому в данном случае в конечном итоге было зарезервировано две страницы, а не одна.

7. Кроме того, важно отметить, чему равен начальный адрес резервирования. Он отличается от заданного нами начального адреса. Как было указано выше, диспетчер памяти Windows округляет заданный начальный адрес в меньшую сторону для того, чтобы он был выровнен правильно с учетом размера страницы системы и степени детализации распределения. Поскольку в данном случае выполняется операция резервирования, то начальный адрес округляется в меньшую сторону до ближайшего значения степени детализации распределения. Наконец, необходимо обратить внимание на то, какой размер области возвращен функцией. Если бы вместо этого был выполнен запрос на закрепление, то округление происходило бы в меньшую сторону до ближайшей границы страницы.
8. Наконец, следует отметить, какой размер области возвращен функцией `VirtualQueryEx`. Даже несмотря на то, что степень детализации распределения в системе равна 64 Кбайт, в этой программе зарезервировано только 8 Кбайт. Это означает, что оставшиеся адреса виртуальной памяти между концом резервирования и следующей границей степени детализации распределения (56 Кбайт пространства адресов между `0x7FF02000` и `0x7FF0FFFF`) становятся неприменимыми. Эти адреса недоступны, поскольку при всех попытках резервирования в этой области будет происходить округление в меньшую сторону, к начальному адресу `0x7FF00000`. Как правило, никогда не следует резервировать пространство виртуальной памяти с объемом меньше 64 Кбайт. Это приводит к бесполезному расходованию пространства адресов, не давая при этом никаких реальных преимуществ. В частности, это не позволяет добиться экономии памяти, поскольку память при ее резервировании фактически не распределяется — при этом просто обозначается ряд адресов как предназначенных для использования в приложении. Всегда можно закреплять за физическим устройством хранения данных лишь отдельные страницы из зарезервированного диапазона, поэтому не следует думать, что, зарезервировав 64 Кбайт, вы при этом занимаете также 64 Кбайт физической памяти. К тому же, следует учитывать, что после исчерпания в процессе пространства адресов (независимо от объема доступной физической памяти) приложение, скорее всего, завершится аварийно, поскольку любые новые запросы на резервирование, связанные с рассматриваемым процессом (включая операции резервирования, инициируемые системой, например, которые нужны для резервирования памяти под новый стек потока), будут оканчиваться неудачей.

В следующем упражнении описано резервирование области памяти с последующей передачей и освобождением отдельных страниц в этой области. После каждого шага на экран выводится состояние каждой страницы в области для проверки того, действительно ли дела обстоят так, как предполагается.

Упражнение 4.6. Резервирование, закрепление и освобождение виртуальной памяти

1. Скопируйте пример проекта vm_release из подкаталога CH04\vm_release компакт-диска, прилагаемого к данной книге, на жесткий диск и загрузите его в среду разработки VC++. Затем откомпилируйте этот проект и вызовите на выполнение. Еще один вариант состоит в том, что можно скопировать исполняемый файл из подкаталога Release и выполнить его вне среды Visual Studio, если вы не заинтересованы в том, чтобы вначале проводить его компиляцию.
2. Работа приложения начинается с распределения области пространства адресов с объемом 64 Кбайт. После этого происходит закрепление второй страницы из этого пространства. Затем закрепление данной страницы отменяется, и работа приложения заканчивается освобождением всей области с объемом 64 Кбайт.
3. После каждого шага вызывается процедура DumpRegionMemoryStatus, которая проходит по циклу через страницы области и выводит информацию о состоянии каждой из них (зарезервированная, закрепленная или свободная). Исходный код программы vm_release приведен в листинге 4.7.

Листинг 4.7. Пример приложения, в котором показаны операции работы с виртуальной памятью

```
// vm_release.cpp. Пример приложения, в котором осуществляются операции
//                 резервирования, закрепления, отмены закрепления и
//                 освобождения памяти
//

#include "stdafx.h"
#include "conio.h"
#include "windows.h"

void DumpRegionMemoryStatus(char *szMsg, char * pV, DWORD
    dwRegionSize)
{

// Вывести вступительное сообщение
  printf("\n%s\n",szMsg);

  // Определить размер страницы системы
  SYSTEM_INFO si;
  GetSystemInfo(&si);

  MEMORY_BASIC_INFORMATION mbi;
  DWORD dwLen=sizeof(mbi);

  char * pCur=pV;
  while ((DWORD)pCur < ((DWORD)pV + dwRegionSize)) {
    VirtualQueryEx(GetCurrentProcess(),pCur,&mbi,dwLen);

    printf("Page at 0x%08x is %s\n",pCur,
       MEM_COMMIT==mbi.State?"Committed":
      MEM_RESERVE==mbi.State?"Reserved":"Free");
    pCur+=si.dwPageSize;
  }
```

```
}

#define REGIONSIZE 65536

int main(int argc, char* argv[])
{

  char *pv=(char *)VirtualAlloc((void *)0x7FF00000,
                                REGIONSIZE,
                                MEM_RESERVE,
                                PAGE_READWRITE);

  if (pv)  {
    DumpRegionMemoryStatus("Memory status after reserving the
        region",pv,REGIONSIZE);

     VirtualAlloc((void *)(pv+4096),4096,
        MEM_COMMIT,PAGE_READWRITE);
    DumpRegionMemoryStatus("Memory status after committing a
         page",pv,REGIONSIZE);

    VirtualFree((void *)(pv+4096),4096,MEM_DECOMMIT);
    DumpRegionMemoryStatus("Memory status after decommitting a
         page",pv,REGIONSIZE);

VirtualFree((void *)pv,0,MEM_RELEASE);
    DumpRegionMemoryStatus("Memory status after releasing the
        region",pv,REGIONSIZE);
  }
  else printf("Error reserving mem %d.\n",GetLastError());

  return 0;
}
```

4. Вызовите это приложение на выполнение и изучите его вывод. Полученные результаты должны выглядеть примерно так, как показано в листинге 4.8.

Листинг 4.8. Вывод программы `vm_release`

```
Memory status after reserving the region
Page at 0x7ff00000 is Reserved
Page at 0x7ff01000 is Reserved
Page at 0x7ff02000 is Reserved
Page at 0x7ff03000 is Reserved
Page at 0x7ff04000 is Reserved
Page at 0x7ff05000 is Reserved
Page at 0x7ff06000 is Reserved
Page at 0x7ff07000 is Reserved
Page at 0x7ff08000 is Reserved
Page at 0x7ff09000 is Reserved
Page at 0x7ff0a000 is Reserved
Page at 0x7ff0b000 is Reserved
Page at 0x7ff0c000 is Reserved
Page at 0x7ff0d000 is Reserved
Page at 0x7ff0e000 is Reserved
```

```
Page at 0x7ff0f000 is Reserved

Memory status after committing a page
Page at 0x7ff00000 is Reserved
Page at 0x7ff01000 is Committed
Page at 0x7ff02000 is Reserved
Page at 0x7ff03000 is Reserved
Page at 0x7ff04000 is Reserved
Page at 0x7ff05000 is Reserved
Page at 0x7ff06000 is Reserved
Page at 0x7ff07000 is Reserved
Page at 0x7ff08000 is Reserved
Page at 0x7ff09000 is Reserved
Page at 0x7ff0a000 is Reserved
Page at 0x7ff0b000 is Reserved
Page at 0x7ff0c000 is Reserved
Page at 0x7ff0d000 is Reserved
Page at 0x7ff0e000 is Reserved
Page at 0x7ff0f000 is Reserved

Memory status after decommitting a page
Page at 0x7ff00000 is Reserved
Page at 0x7ff01000 is Reserved
Page at 0x7ff02000 is Reserved
Page at 0x7ff03000 is Reserved
Page at 0x7ff04000 is Reserved
Page at 0x7ff05000 is Reserved
Page at 0x7ff06000 is Reserved
Page at 0x7ff07000 is Reserved
Page at 0x7ff08000 is Reserved
Page at 0x7ff09000 is Reserved
Page at 0x7ff0a000 is Reserved
Page at 0x7ff0b000 is Reserved
Page at 0x7ff0c000 is Reserved
Page at 0x7ff0d000 is Reserved
Page at 0x7ff0e000 is Reserved
Page at 0x7ff0f000 is Reserved

Memory status after releasing the region
Page at 0x7ff00000 is Free
Page at 0x7ff01000 is Free
Page at 0x7ff02000 is Free
Page at 0x7ff03000 is Free
Page at 0x7ff04000 is Free
Page at 0x7ff05000 is Free
Page at 0x7ff06000 is Free
Page at 0x7ff07000 is Free
Page at 0x7ff08000 is Free
Page at 0x7ff09000 is Free
Page at 0x7ff0a000 is Free
Page at 0x7ff0b000 is Free
Page at 0x7ff0c000 is Free
Page at 0x7ff0d000 is Free
Page at 0x7ff0e000 is Free
Page at 0x7ff0f000 is Free
```

Вполне очевидно, что операция закрепления отдельной страницы из зарезервированной области является весьма несложной. Столь же простой является операция отмены закрепления страницы, как и операция освобождения всей области. В следующем упражнении показано, как применяется атрибут защиты страницы PAGE_GUARD. В нем распределяется буфер в памяти, который первоначально является защищенным, после этого атрибут защиты с одной из страниц снимается с помощью попытки заблокировать эту страницу в памяти.

Упражнение 4.7. Защита памяти с помощью атрибута PAGE_GUARD

1. Начнем с изучения исходного кода программы vm_guard — примера приложения, который используется для изучения принципов действия атрибута PAGE_GUARD. Кратко ознакомьтесь с кодом, приведенным в листинге 4.9, и определите, можете ли вы понять его работу с первого взгляда. Подробное описание этого приложения приведено немного ниже.

Листинг 4.9. Пример приложения, в котором используется атрибут PAGE_GUARD

```
// vm_guard.cpp. Пример, который демонстрирует, как действует
//               атрибут PAGE_GUARD
//

#include "stdafx.h"
#include "conio.h"
#include "windows.h"

#define REGIONSIZE 65536

int main(int argc, char* argv[])
{

  char *pv=(char *)VirtualAlloc(NULL,
                          REGIONSIZE,
                          MEM_RESERVE | MEM_COMMIT,
                          PAGE_READWRITE | PAGE_GUARD);

  if (pv)  {

    // Попытка заблокировать страницу; должна окончиться неудачей,
    // поскольку установлен атрибут PAGE_GUARD
      bool bLocked=VirtualLock((void *)pv,4096);
      if (!bLocked) {
        printf("First VirtualLock failed for 0x%08X, Last error =
            0x%08X\n", pv, GetLastError());
      } else printf("First VirtualLock succeeded for 0x%08X\n",
                    pv);

     // Повторная попытка заблокировать страницу; должна окончиться
     // успешно, поскольку атрибут PAGE_GUARD был сброшен
      bLocked=VirtualLock((void *)pv,4096);
      if (!bLocked) {
         printf("Second VirtualLock failed for 0x%08X, Last error
        = 0x%08X\n", pv, GetLastError());
       } else printf("Second VirtualLock succeeded for 0x%08X\n",
```

```
                    pv);

    VirtualFree((void *)pv,0,MEM_RELEASE);
  }
  else printf("Error reserving/committing memory.  Last error=
              %d.\n",GetLastError());

  return 0;
}
```

2. Загрузите этот код из подкаталога CH04\vm_guard компакт-диска, откомпилируйте его и вызовите на выполнение. Полученные результаты должны выглядеть примерно таким образом:

   ```
   First VirtualLock failed for 0x00440000, Last error = 0x80000001
   Second VirtualLock succeeded for 0x00440000
   ```

3. Выполнение этого кода начинается с распределения блока виртуальной памяти, для которого в этой программе устанавливается защита с помощью атрибута PAGE_GUARD. Весь этот блок становится выходящим за пределы доступа из-за наличия атрибута PAGE_GUARD.

4. После этого предпринимается попытка заблокировать первую страницу блока в физической памяти с помощью функции VirtualLock. (Автор ввел непосредственно в код данные о размере страницы в целях упрощения; на практике в коде для выборки информации о размере страницы системы следует всегда использовать функцию GetSystemInfo.)

5. Этот первый вызов функции VirtualLock приводит к получению двух результатов: он оканчивается неудачей и сбрасывает атрибут PAGE_GUARD для первой страницы в области.

6. Следует отметить, что вывод функции GetLastError, полученный после этого вывода, окончившегося неудачей, равен 0x80000001, а это значение эквивалентно коду возврата STATUS_GUARD_PAGE_VIOLATION, который был упомянут выше в данной главе.

7. Поскольку состояние PAGE_GUARD для первой страницы области было сброшено, вторая попытка заблокировать эту страницу завершается успехом. Атрибут PAGE_GUARD действует именно по этому принципу: он предоставляет единоразовый механизм отказа, который позволяет обнаруживать недопустимые попытки доступа к странице.

ПРИМЕЧАНИЕ. Следует отметить, что функция VirtualQuery(Ex) всегда сообщает атрибуты защиты страницы на тот момент, как она была распределена первоначально; на выводе функции VirtualQuery(Ex) не отражаются ни изменения, внесенные с помощью функции VirtualProtect, ни изменения, происшедшие в результате сброса атрибута PAGE_GUARD. Обнаружив это впервые, автор был удивлен, но такое действие указанной функции соответствует документации комплекта Platform SDK. Еще одно интересное направление использования функции VirtualQuery состоит в проверке памяти процесса SQL Server. Поскольку пользователь программы SQL Server имеет возможность создавать и выполнять расширенные процедуры, то ему предоставляется право загрузить библиотеку DLL

в пространство процесса SQL Server и вызвать на выполнение содержащийся в ней код так, как если бы он был частью самой программы SQL Server. Эта возможность часто используется для проверки различных внутренних структур сервера, включая собственные распределения памяти сервера. В следующем упражнении показано, как создать расширенную процедуру, позволяющую подробно изучать операции распределения виртуальной памяти процесса SQL Server.

Упражнение 4.8. Проверка распределений памяти процесса SQL Server с помощью функции `VirtualQuery`

Начнем с изучения исходного кода рассматриваемой расширенной процедуры. На базе тех знаний о виртуальной памяти, которые были получены вами при изучении предыдущей части этой главы, кратко ознакомьтесь с кодом, приведенным в листинге 4.10, и определите, можете ли вы понять, как он работает. Подробное описание работы данного кода приведено ниже.

Листинг 4.10. Исходный код программы `xp_vmquery`

```
#include <stdafx.h>

#define XP_NOERROR              0
#define XP_ERROR                1
#define MAXADDRLEN             12
#define MAXSIZELEN             12
#define MAXPROTLEN            128
#define MAXSTATELEN            20
#define MAXTYPELEN             20

#ifdef __cplusplus
extern "C" {
#endif

RETCODE __declspec(dllexport) xp_vmquery(SRV_PROC *srvproc);
#ifdef __cplusplus
}
#endif

RETCODE __declspec(dllexport) xp_vmquery(SRV_PROC *srvproc)
{

  bool bByPage=false;
  DWORD dwParams=srv_rpcparams(srvproc);

  if (1==dwParams) {

    BYTE bType;
   ULONG cbMaxLen;
   ULONG cbActualLen;
   char szByPage[2];
    BOOL fNull;

    srv_paraminfo(srvproc, 1, &bType, &cbMaxLen, &cbActualLen,
        (BYTE *)&szByPage, &fNull);
```

```
    // Разрешить переход в страничный режим, если передан параметр "P"
    bByPage=(!stricmp("P",szByPage));
}

// Определить имена столбцов
char szColName[129];

    wsprintf(szColName, "Address");
    srv_describe(srvproc, 1, szColName, SRV_NULLTERM, SRVCHAR,
        MAXADDRLEN, SRVCHAR, 0, NULL);

   wsprintf(szColName, "Size");
    srv_describe(srvproc, 2, szColName, SRV_NULLTERM, SRVCHAR,
        MAXSIZELEN, SRVCHAR, 0, NULL);

    wsprintf(szColName, "Protection");
    srv_describe(srvproc, 3, szColName, SRV_NULLTERM, SRVCHAR,
        MAXPROTLEN, SRVCHAR, 0, NULL);

   wsprintf(szColName, "State");
    srv_describe(srvproc, 4, szColName, SRV_NULLTERM, SRVCHAR,
        MAXSTATELEN, SRVCHAR, 0, NULL);

   wsprintf(szColName, "Type");
   srv_describe(srvproc, 5, szColName, SRV_NULLTERM, SRVCHAR,
       MAXTYPELEN, SRVCHAR, 0, NULL);

   // Получить информацию об адресах непривилегированного режима
   SYSTEM_INFO si;
   GetSystemInfo(&si);
   char * pszStart=(char *)si.lpMinimumApplicationAddress;

   char szProt[256];
   char szState[256];
   char szType[256];
   char szBase[12];
   char szSize[12];

   // Задать привязки данных столбцов
   srv_setcoldata(srvproc, 1, szBase);
   srv_setcoldata(srvproc, 2, szSize);
   srv_setcoldata(srvproc, 3, szProt);
   srv_setcoldata(srvproc, 4, szState);
   srv_setcoldata(srvproc, 5, szType);

   MEMORY_BASIC_INFORMATION mbi;
   int i=0;
   while ((pszStart) &&
     (pszStart<si.lpMaximumApplicationAddress)) {

   // Получить информацию, относящуюся к текущему блоку памяти
   VirtualQuery(pszStart,&mbi,sizeof(mbi));

   // Определить столбец Address
   wsprintf(szBase,"0x%lp",mbi.BaseAddress);

   // Определить столбец Size
```

```
    wsprintf(szSize,"%010d",mbi.RegionSize);

    // Определить столбец Protection
    szProt[0]='\0';
    if (mbi.Protect & PAGE_READONLY) strcat(szProt,"READONLY ");
    if (mbi.Protect & PAGE_READWRITE) strcat(szProt,"READWRITE ");
    if (mbi.Protect & PAGE_WRITECOPY) strcat(szProt,"WRITECOPY ");
    if (mbi.Protect & PAGE_EXECUTE) strcat(szProt,"EXECUTE ");
    if (mbi.Protect & PAGE_EXECUTE_READ)
      strcat(szProt,"EXECUTE_READ ");
if (mbi.Protect & PAGE_EXECUTE_READWRITE)
    strcat(szProt,"EXECUTE_READWRITE ");
if (mbi.Protect & PAGE_EXECUTE_WRITECOPY)
    strcat(szProt,"EXECUTE_WRITECOPY ");
if (mbi.Protect & PAGE_GUARD) strcat(szProt,"GUARD ");
if (mbi.Protect & PAGE_NOACCESS) strcat(szProt,"NOACCESS ");
if (mbi.Protect & PAGE_NOCACHE) strcat(szProt,"NOCACHE ");

// Удалить заключительный пробел
if (szProt[0]) szProt[strlen(szProt)-1]='\0';
else strcpy(szProt,"UNKNOWN");

// Определить столбец State
szState[0]='\0';
if (mbi.State & MEM_FREE) strcat(szState, "Free ");
else {
    if (mbi.State & MEM_RESERVE) strcat(szState, "Reserved ");
    if (mbi.State & MEM_COMMIT) strcat(szState, "Commit ");
}

// Удалить заключительный пробел
if (szState[0]) szState[strlen(szState)-1]='\0';

// Определить столбец Type
szType[0]='\0';
if (mbi.Type & MEM_IMAGE) strcat(szType,"Image ");
else if (mbi.Type & MEM_MAPPED) strcat(szType,"Mapped ");
else if (mbi.Type & MEM_PRIVATE) strcat(szType,"Private ");

if (szType[0]) szType[strlen(szType)-1]='\0';
else strcpy(szType,"Unknown");

// Задать текущие значения ширины столбцов
srv_setcollen(srvproc, 1, strlen(szBase));
srv_setcollen(srvproc, 2, strlen(szSize));
srv_setcollen(srvproc, 3, strlen(szProt));
srv_setcollen(srvproc, 4, strlen(szState));
srv_setcollen(srvproc, 5, strlen(szType));

// Передать строку клиенту
    srv_sendrow(srvproc);
i++;

// Перейти к следующей странице или области
if (bByPage) pszStart+=si.dwPageSize;
    else pszStart+=mbi.RegionSize;
  }
```

```
  return XP_NOERROR ;
}
```

1. Скопируйте исполняемый файл этой расширенной процедуры из подкаталога `CH04\xp_vmquery` компакт-диска в подкаталог `binn` инсталляционного каталога SQL Server.
2. Установите необходимую библиотеку в главную базу данных, выполнив следующую команду в программе Query Analyzer:
   ```
   sp_addextendedproc 'xp_vmquery','xp_vmquery.dll'
   ```
3. Вызовите исполняемый файл на выполнение из программы Query Analyzer следующим образом:
   ```
   xp_vmquery
   ```
4. Должны быть получены примерно такие результаты, как показано в листинге 4.11.

Листинг 4.11. Вывод расширенной процедуры `xp_vmquery`

```
Address    Size       Protection      State       Type
---------- ---------- --------------- ----------- --------
0x00010000 0000004096 READWRITE       Committed   Private
0x00011000 0000061440 NOACCESS        Free        Unknown
0x00020000 0000004096 READWRITE       Committed   Private
0x00021000 0000061440 NOACCESS        Free        Unknown
0x00030000 0000454656 UNKNOWN         Reserved    Private
0x0009F000 0000004096 READWRITE GUARD Committed   Private
0x000A0000 0000065536 READWRITE       Committed   Private
0x000B0000 0000282624 READWRITE       Committed   Private
0x000F5000 0000061440 UNKNOWN         Reserved    Private
0x00104000 0000004096 READWRITE       Committed   Private
0x00105000 0000700416 UNKNOWN         Reserved    Private
0x001B0000 0000004096 READWRITE       Committed   Mapped
0x001B1000 0000061440 UNKNOWN         Reserved    Mapped
0x001C0000 0000090112 READONLY        Committed   Mapped
0x001D6000 0000040960 NOACCESS        Free        Unknown
0x001E0000 0000192512 READONLY        Committed   Mapped
0x0020F000 0000004096 NOACCESS        Free        Unknown
```

5. Как и в примере с программой `vm_release`, в этой расширенной процедуре используется функция `VirtualQuery` для последовательного прохождения по всему пространству процесса SQL Server и формирования отчета о содержимом каждой области распределенной памяти. С помощью данной процедуры можно быстро получить итоговые сведения о том, какой объем зарезервированной и закрепленной памяти распределен в указанном процессе и какой объем виртуальной памяти остается неиспользуемым (свободным). С ее помощью можно узнать, какие страницы являются закрытыми страницами (обычные распределения), отображенными (страницами с файлами, отображаемыми на память) и страницами двоичных образов (принадлежащими к файлам EXE и DLL).
6. Воспользуйтесь программой OSQL для вызова процедуры `xp_vmquery` с предусмотренной в ней опцией `'P'` (режим страниц), чтобы просмотреть информацию о распределении для каждой страницы в пространстве процесса SQL Server

(а не для каждой области, как в п. 4). Автор рекомендует вызвать эту процедуру с помощью программы OSQL, чтобы предотвратить возможность выхода за пределы виртуальной памяти в программе Query Analyzer, поскольку процедура `xp_vmquery` возвращает сотни тысяч строк при выполнении в режиме страниц.

Виртуальная память. Резюме

Средства управления виртуальной памятью операционной системы Windows принадлежат к числу наиболее мощных средств этой ОС. Предоставляя приложению доступ к огромному пространству адресов процесса, которое может передаваться в реальную память в виде отдельных фрагментов, эта операционная система позволяет пользоваться простыми средствами непрерывной адресации в сочетании с эффективными и экономичными средствами разреженного потребления ресурсов.

Каждая страница в виртуальной памяти находится в одном из трех состояний: зарезервированная, закрепленная или свободная. В приложении можно распределять конкретные страницы памяти или позволить операционной системе Windows определить точное местонахождение распределяемых страниц для выполнения запроса на распределение.

В приложении операции резервирования и закрепления памяти могут выполняться последовательно или одновременно, кроме того, может осуществляться закрепление отдельных зарезервированных страниц. Страницы в зарезервированной области могут иметь и часто имеют разные атрибуты защиты. Такие атрибуты защиты могут быть присвоены либо непосредственно во время выполнения операции резервирования (или закрепления), либо путем последующего вызова функции `VirtualProtect`. Для отмены закрепления закрепленных страниц, а также для освобождения зарезервированных страниц может применяться функция `VirtualFree`. Функция `VirtualLock` предназначена для блокирования страниц в физической памяти, а функция `VirtualUnlock` – для их разблокирования.

В качестве физической памяти, на которую отображается виртуальная память, часто используется системный файл подкачки, хотя, безусловно, некоторые страницы могут отображаться непосредственно на физическую память. Кроме того, виртуальная память может отображаться на какой-то файл на диске. В операционной системе Windows такая возможность применяется для предоставления совместного доступа к исполняемому коду приложения нескольким экземплярам этого приложения. В таком случае физической памятью, на которую отображается виртуальная память, зарезервированная в каждом процессе для хранения кода и данных приложения, становится сам файл EXE или DLL. При попытке в приложении изменить одну из его страниц данных средство копирования при записи создает копию этой страницы и дает модифицирующему ее процессу указание о том, что эта копия должна использоваться вместо оригинала. Это позволяет обеспечить максимально возможную продолжительность совместного доступа многочисленных экземпляров к максимально возможному объему кода и данных исполняемого файла, из которого состоит эксплуатируемое приложение.

Виртуальная память. Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Даже если в программе выполняется операция резервирования диапазона адресов виртуальной памяти, составляющего только 32 Кбайт, ОС Windows все равно резервирует диапазон, равный 64 Кбайт, поскольку в системе применяется именно такая степень детализации распределения.
2. Какой атрибут защиты страницы используется в операционной системе Windows для обозначения конца диапазона адресов закрепленной памяти стека потока?
3. Подтвердите или опровергните следующее утверждение. Даже если страница была заблокирована в физической памяти с помощью вызова функции `VirtualLock`, она все еще может быть выведена на диск в результате страничного обмена, если в этом возникнет необходимость в связи с увеличением потребности в памяти.
4. Что происходит при попытке внести в каком-либо процессе изменения в страницу, которая была обозначена атрибутом защиты `PAGE_WRITECOPY`?
5. Подтвердите или опровергните следующее утверждение. В программе SQL Server основной объем распределяемой памяти берется из динамической области памяти системы.
6. Чему равен заданный по умолчанию размер стека потока в программе SQL Server?
7. Можно ли резервировать и закреплять память в одном вызове функции `VirtualAlloc`, или в приложении необходимо применять отдельные вызовы этой функции для резервирования и закрепления памяти?
8. Подтвердите или опровергните следующее утверждение. Во всех запросах к виртуальной памяти (независимо от того, являются они запросами непривилегированного или привилегированного режима) необходимо учитывать степень детализации распределения в системе.
9. В приложениях какого типа атрибут защиты страницы `PAGE_WRITECOMBINE` используется чаще всего?
10. Подтвердите или опровергните следующее утверждение. В приложении атрибут защиты `PAGE_EXECUTE_WRITECOPY` можно использовать для реализации функциональных возможностей копирования при записи для закрепленных страниц в пространстве адресов непривилегированного режима.
11. Какая функция API-интерфейса Win32 используется для освобождения зарезервированной области адресов виртуальной памяти?
12. Подтвердите или опровергните следующее утверждение. Атрибуты `PAGE_EXECUTE` и `PAGE_READONLY` в операционной системе, эксплуатируемой на компьютере с процессором x86, функционально эквивалентны.
13. Подтвердите или опровергните следующее утверждение. В приложении можно указывать размеры распределения виртуальной памяти, но не точное

местонахождение; решение о том, каким будет точное местонахождение области распределения, принимает диспетчер памяти Windows.

14. Какой параметр можно передать в функцию `VirtualAlloc`, чтобы сообщить ей, что некоторую область закрепленной памяти необходимо сбрасывать (очищать) в целях предотвращения вывода содержимого этой области на диск с помощью страничного обмена, если нужно будет освободить эту область из-за увеличения потребностей в памяти?
15. Подтвердите или опровергните следующее утверждение. Для резервирования диапазона адресов виртуальной памяти, которые пока еще не нужно закреплять, используется функция `VirtualReserve`.
16. Подтвердите или опровергните следующее утверждение. Функция `VirtualRelease` позволяет освободить область зарезервированной виртуальной памяти без предварительной отмены ее закрепления.
17. Какой параметр редактора связей VC++ может использоваться разработчиком приложений для корректировки заданного по умолчанию размера стека?
18. Подтвердите или опровергните следующее утверждение. С помощью вызова функции `FlushViewOfFile` API-интерфейса Win32 можно вынудить систему немедленно выполнить запись на диск отображаемых на память страниц файла.
19. Подтвердите или опровергните следующее утверждение. Попытка доступа к странице с атрибутом защиты PAGE_GUARD вынуждает систему активизировать исключительную ситуацию STATUS_GUARD_PAGE и сбросить защиту защищенной страницы.
20. Подтвердите или опровергните следующее утверждение. Виртуальная память, для которой была выполнена отмена закрепления, все еще остается зарезервированной до тех пор, пока не будет освобождена.
21. Какая функция API-интерфейса Windows вызывается программой SQL Server, если разрешена опция установки размера рабочего набора?
22. Существует ли возможность изменить объем виртуальной памяти, зарезервированной для стека нового потока с помощью вызова функции `CreateThread`?
23. Какое количество страниц виртуальной памяти разрешено по умолчанию заблокировать в процессе в физической памяти?
24. Подтвердите или опровергните следующее утверждение. Для отмены закрепления ранее закрепленных страниц виртуальной памяти, которые охватывают границы степени детализации распределения, может использоваться функция `VirtualDecommit` API-интерфейса.
25. Подтвердите или опровергните следующее утверждение. При передаче области зарезервированной памяти операционная система Windows округляет запрос на закрепление до ближайшей границы степени детализации распределения.

Динамические области памяти

Динамической областью памяти называется область памяти, состоящая из одной или нескольких страниц зарезервированного пространства, которая может дополнительно распределяться с разделением на меньшие части с помощью диспетчера динамической области памяти. Динамические области памяти наиболее часто применяются для распределения большого количества относительно небольших объектов или структур, ненамного различающихся по своим размерам. Динамические области памяти не следует использовать для блоков с объемом 1 Мбайт или больше; для подобных крупных распределений следует применять функцию `VirtualAlloc` и подобные ей функции.

Одним из преимуществ динамических областей памяти является то, что они позволяют не учитывать границы, определяющие степень детализации распределения системы и размеры страниц системы. А недостатком динамических областей памяти является то, что доступ с их помощью происходит немного медленнее и они предоставляют менее развитый уровень контроля по сравнению с API-интерфейсами виртуальной памяти. Например, нельзя зарезервировать область памяти, не выполнив также ее закрепление; функция `VirtualAlloc` является единственной функцией распределения Win32, в которой эти две операции разделены.

Точные алгоритмы, используемые диспетчером динамической области памяти для закрепления и отмены закрепления областей памяти на физическое устройство хранения, не документированы. Кроме того, они изменялись от одного выпуска Windows к другому. Если весь проект приложения зависит от наличия точной информации о том, какие операции используются в диспетчере динамической области памяти для управления реальной памятью, на которую отображаются динамические области памяти, и/или необходимо обеспечить жесткий контроль над этими операциями, то автор рекомендует сразу же отказаться от использования динамических областей памяти и вместо них применять виртуальную память.

Динамические области памяти. Основные термины и определения

- **Применяемая по умолчанию динамическая область памяти.** Встроенная динамическая область памяти, предоставляемая по умолчанию операционной системой Windows каждому процессу. Применяемая по умолчанию динамическая область памяти процесса имеет базовый размер 1 Мбайт, но это значение может быть изменено с помощью одного из параметров редактора связей.
- **Закрытая (или пользовательская) динамическая область памяти.** Динамическая область памяти, создаваемая в процессе для собственного использования отдельно от применяемой по умолчанию динамической области памяти процесса.

- **Упорядочение доступа к динамической области памяти.** Средство, с помощью которого диспетчер динамической области памяти Windows обеспечивает то, что многочисленные потоки не искажают динамическую область памяти в ходе одновременного доступа.

Динамические области памяти. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к динамической области памяти, приведены в табл. 4.8.

Таблица 4.8. Основные функции API-интерфейсов, относящиеся к динамической области памяти

Функция	Описание
`HeapCreate`	Создать закрытую динамическую область памяти
`HeapAlloc`	Распределить память из динамической области памяти
`HeapFree`	Освободить блок памяти, распределенный из динамической области памяти
`HeapDestroy`	Уничтожить (освободить) закрытую динамическую область памяти
`GetProcessHeap`	Возвратить дескриптор заданной по умолчанию динамической области памяти процесса

Применяемая по умолчанию динамическая область памяти

В операционной системе Windows каждому процессу предоставляется применяемая по умолчанию динамическая область памяти. В приложениях применяемая по умолчанию динамическая область памяти служит для поддержки таких средств распределения памяти, как функция `malloc` и оператор `new` языка C++. Кроме того, применяемая по умолчанию динамическая область памяти используется в некоторых функциях API-интерфейса Win32, включая старые 16-битовые функции `LocalAlloc` и `GlobalAlloc`.

Динамическая область памяти процесса по умолчанию имеет размер 1 Мбайт, но это значение может быть изменено с помощью параметра `/HEAP` редактора связей (это — параметр компилятора Visual C++; в большинстве других компиляторов предусмотрена аналогичная опция) или с помощью утилит, которые позволяют редактировать заголовок исполняемого файла. Указанное значение невелико, и именно поэтому динамические области памяти не подходят для крупных распределений памяти; для этой цели следует применять виртуальную память.

По умолчанию операционная система Windows упорядочивает доступ к динамической области памяти процесса. Это означает, что к применяемой по умолчанию динамической области памяти может одновременно иметь доступ только один поток. Такой подход позволяет предотвратить возникновение ошибок синхронизации доступа к динамической области памяти в многопотоковых

приложениях и защитить динамическую область памяти от искажения. Следует отметить, что существует возможность отменить такую синхронизацию для отдельных операций распределения памяти из динамической области памяти, но подобный подход обычно не рекомендуется.

С помощью вызова функции HeapDestroy нельзя уничтожить применяемую по умолчанию динамическую область памяти процесса. Если в функцию HeapDestroy передается дескриптор применяемой по умолчанию динамической области памяти, система игнорирует этот вызов. Если требуется ограничить физические размеры применяемой по умолчанию динамической области памяти, то для этого следует использовать опцию /HEAP редактора связей.

Распределение памяти из динамической области памяти

Для распределения и отмены распределения памяти из динамической области памяти используются процедуры HeapAlloc и HeapFree. Для обеих этих функций требуется указать дескриптор динамической области памяти, из которой необходимо распределить память или отменить распределение. Этот дескриптор может быть возвращен из функции HeapCreate или из функции GetProcessHeap (если для работы требуется память, полученная из применяемой по умолчанию динамической области памяти).

Для того чтобы распределить память из динамической области памяти, функция HeapAlloc должна выполнить описанные ниже шаги.

1. Просмотреть связный список распределенных и освобожденных блоков, чтобы найти первый свободный блок, достаточно большой для выполнения данного запроса.
2. Распределить этот свободный блок, отметив его как распределенный.
3. Добавить новый блок к связному списку, которым управляет диспетчер динамической области памяти.

Функция HeapAlloc поддерживает три параметра: HEAP_ZERO_MEMORY, HEAP_GENERATE_EXCEPTIONS и HEAP_NO_SERIALIZE. Эти константы можно комбинировать с помощью побитового оператора OR для передачи их в функцию HeapAlloc в качестве второго параметра.

Как указывает само его имя, параметр HEAP_ZERO_MEMORY вызывает заполнение каждого блока распределенной памяти нулями по такому же принципу, как заполняются нулями страницы виртуальной памяти при первом доступе к ним. Такая возможность упрощает задачу контроля над ошибками, связанными с применением неинициализированного буфера.

Параметр HEAP_GENERATE_EXCEPTIONS вынуждает функцию HeapAlloc активизировать при возникновении ошибок исключительные ситуации, а не возвращать NULL. При использовании этого параметра может активизироваться одна из двух исключительных ситуаций: STATUS_NO_MEMORY (которая указывает на наличие ситуации нехватки памяти) или STATUS_ACCESS_VIOLATION (которая

указывает на то, что возникло искажение динамической области памяти или применяются недопустимые параметры функции).

Параметр HEAP_NO_SERIALIZE отменяет все средства синхронизации потоков, которые должны использоваться при обычных обстоятельствах во время доступа к динамической области памяти. Как уже было сказано, применяемая по умолчанию динамическая область памяти процесса всегда создается с разрешенными по умолчанию средствами упорядочения доступа. Кроме того, с разрешенными средствами упорядочения доступа могут создаваться и пользовательские динамические области памяти. Применение этих средств упорядочения доступа можно отменять для конкретной операции распределения, передавая в функцию HeapAlloc параметр HEAP_NO_SERIALIZE.

Функция HeapRealloc позволяет изменить размеры блока. Но, выполняя дополнительные распределения внутри полученного блока, необходимо соблюдать осторожность, поскольку увеличение размера блока может привести к его перемещению в динамической области памяти. А после перемещения блока необходимо также откорректировать все указатели, которые на него ссылаются. Для предотвращения перемещения блока в динамической области памяти можно передать параметр HEAP_REALLOC_IN_PLACE_ONLY. Применение такого параметра приводит к тому, что операция перераспределения оканчивается неудачей, если блок должен увеличиться в размерах, а для этого его необходимо переместить в больший по величине свободный блок.

Пользовательские динамические области памяти

В приложении пользовательская динамическая область памяти может быть создана путем вызова функции HeapCreate. Для создания собственной пользовательской динамической области памяти существует несколько важных причин, включая перечисленные ниже.

- Отделение компонентов друг от друга. Помещая разные компоненты в отдельные динамические области памяти, можно исключить ситуацию, в которой ошибочные модификации одного компонента могут привести к искажению других компонентов.
- Эффективное управление памятью. Распределив свою собственную динамическую область памяти, можно установить ее размеры таким образом, чтобы в ней можно было хранить требуемое количество объектов с одинаковыми размерами максимально эффективно.
- Повышение плотности распределения. Распределяя объекты в памяти близко друг к другу, можно уменьшить вероятность того, что будет возникать пробуксовка системы во время обработки в цикле списка объектов, хранящихся в памяти.
- Исключение издержек, связанных с синхронизацией потоков. Если известно, что синхронизация потоков не требуется (например, в случае применения однопотокового приложения), то можно исключить издержки синхронизации доступа к динамической области памяти, создав собственную динамическую область памяти.

- Быстрое и простое выполнение операции отмены распределения. Независимо от того, какое количество отдельных распределений памяти создано в пользовательской динамической области памяти с помощью функции HeapAlloc, все эти операции распределения можно отменить за один раз, уничтожив динамическую область памяти с помощью вызова функции HeapDestroy.

При создании пользовательской динамической области памяти можно задать параметры HEAP_NO_SERIALIZE или HEAP_GENERATE_EXCEPTIONS либо комбинацию этих двух параметров. Как было указано выше, параметр HEAP_NO_SERIALIZE отменяет упорядочение доступа к динамической области памяти. Упорядочение доступа потоков разрешено по умолчанию. Эта тема будет более подробно описана ниже, но, вообще говоря, эту опцию не следует использовать, за исключением тех случаев, когда вы абсолютно уверены в том, что синхронизации потоков во время доступа к динамической области памяти не потребуется.

Кроме того, как и при использовании функции HeapAlloc, передача в функцию HeapCreate параметра HEAP_GENERATE_EXCEPTIONS вынуждает систему активизировать исключительную ситуацию, когда попытка распределить (или перераспределить) блок памяти из динамической области памяти оканчивается неудачей. При обычных обстоятельствах неудачные попытки распределения приводят к возврату NULL-указателя. Этот параметр можно использовать для передачи диспетчеру динамической области памяти указания о том, что вместо этого следует активизировать исключительную ситуацию.

Во втором параметре функции HeapCreate задается количество байтов, первоначально закрепленных для динамической области памяти. В случае необходимости в функции HeapCreate это значение округляется в большую сторону до величины, кратной размеру страницы процессора.

Третий параметр функции HeapCreate определяет максимальный размер динамической области памяти. Если необходимо создать динамическую область памяти без фиксированного предела размера, в качестве этого параметра нужно указать 0. Для уничтожения пользовательской динамической области памяти можно использовать функцию HeapDestroy. Если попытка уничтожить пользовательскую динамическую область памяти окончится неудачей, то данная область останется в памяти до завершения работы процесса.

Упорядочение доступа к динамической области памяти

При создании пользовательской динамической области памяти может применяться параметр HEAP_NO_SERIALIZE, от которого зависит, будет ли автоматически упорядочен доступ к динамической области памяти. Если в приложении имеется несколько потоков, получающих доступ к динамической области памяти, для которой отменена синхронизация доступа, то несколько потоков могут одновременно захватывать один и тот же блок памяти, в результате чего в приложение будет заложена настоящая бомба замедленного действия, готовая взорваться в самый неподходящий момент. Тот факт, что в программе допущена такая серьезная ошибка, может остаться незаметным на первый взгляд. Например, окажется так, что проблема не будет обнаружена до тех пор, пока приложение не будет

вызвано на выполнение на многопроцессорном компьютере или на компьютере, оборудованном гораздо более быстродействующим процессором по сравнению с тем компьютером, который применялся при разработке данной программы. Как известно, ошибки синхронизации потоков отслеживать сложнее всего, поскольку они почти всегда обусловлены временными соотношениями. Даже сам тот факт, что код, содержащий эти ошибки, выполняется под управлением отладчика в пошаговом режиме, может привести к тому, что они покажутся несуществующими.

Ниже перечислены некоторые потенциальные проблемы, связанные с синхронизацией доступа к динамической области памяти в многопотоковых приложениях.

- Искажается связный список блоков в динамической области памяти.
- Возникает такая ситуация, при которой многочисленные потоки совместно используют один и тот же блок памяти.
- Один из потоков освобождает блок, который все еще используется другими потоками. В дальнейшем эти потоки перезаписывают данные в нераспределенной памяти, что, в свою очередь, приводит к искажению динамической области памяти.

Вообще говоря, в действительности не следует использовать параметр `HEAP_NO_SERIALIZE`, за исключением тех случаев, когда вы абсолютно уверены, что упорядочивать доступ к динамической области памяти не нужно. В частности, этот параметр может применяться, если только соблюдается одно из описанных ниже условий.

- Рассматриваемый процесс является однопотоковым.
- В процессе имеется несколько потоков, но доступ к динамической области памяти получает только один из этих потоков.
- Рассматриваемый процесс является многопотоковым, и доступ к динамической области памяти осуществляется многими потоками, но в приложении предусмотрены собственные средства упорядочения доступа к применяемой динамической области памяти.

Упражнения

В программе можно выполнить перегрузку операторов `new` и `delete` языка C++ для того, чтобы с их помощью распределять память для объектов из пользовательских динамических областей памяти. При обнаружении вызова оператора `new` в программе компилятор C++ проводит проверку, чтобы определить, не был ли этот оператор перегружен в данном классе. Если дело обстоит именно так, то компилятор формирует вызов этой функции, а не код, распределяющий память для объекта из применяемой по умолчанию динамической области памяти. Перегрузку оператора `new` можно использовать таким образом, чтобы в нем применялись любые средства распределения памяти, выбранные пользователем. Код, приведенный в следующем упражнении, показывает, как выполнить перегрузку операторов `new` и `delete`, чтобы с их помощью распределять память для объектов из пользовательской динамической области памяти.

Упражнение 4.9. Перегрузка операторов new и delete, чтобы с их помощью можно было распределять память из пользовательской динамической области памяти

1. Загрузите код, приведенный в листинге 4.12, из подкаталога CH04\heapnew компакт-диска, прилагаемого к данной книге, или введите его вручную в среде VC++, затем откомпилируйте и вызовите на выполнение.

Листинг 4.12. Перегрузка операторов new и delete для применения определяемого пользователем средства распределения памяти

```
// heapnew.cpp. Перегрузка операторов new и delete для применения
//              определяемой пользователем динамической области памяти
//

#include "stdafx.h"
#include "windows.h"

class CMemObj {
public:
  static HANDLE s_hPrivateHeap;
  static DWORD s_dwBlocks;
  void* operator new (size_t size);
  void  operator delete(void *p);
};
HANDLE CMemObj::s_hPrivateHeap=NULL;
DWORD CMemObj::s_dwBlocks=0;

void* CMemObj::operator new (size_t size) {

  // Создать закрытую динамическую область памяти, если она еще
  // не существует
  if (NULL==s_hPrivateHeap) {
     s_hPrivateHeap=HeapCreate(HEAP_NO_SERIALIZE,0,0);
    if (NULL==s_hPrivateHeap) return NULL;
  }

  // Распределить память
  void* p=HeapAlloc(s_hPrivateHeap,0,size);

  // Увеличить значение количества блоков
  if (p) s_dwBlocks++;
  return p;
}

void CMemObj::operator delete(void *p) {

  // Отменить распределение памяти
  if (HeapFree(s_hPrivateHeap,0,p)) {

    // Уменьшить значение количества блоков
    s_dwBlocks--;

     // Если все блоки освобождены, уничтожить динамическую
     // область памяти
```

```
    if (0==s_dwBlocks)
      if (HeapDestroy(s_hPrivateHeap))
        s_hPrivateHeap=NULL;
  }
}

int main(int argc, char* argv[])
{

  // Распределить память для одного объекта в закрытой динамической
  // области памяти
  CMemObj *pMO = new CMemObj();
  printf("Custom heap after first new: %d block(s),
      handle=0x%08x\n",CMemObj::s_dwBlocks,
      CMemObj::s_hPrivateHeap);

  // Распределить память для второго объекта в закрытой динамической
  // области памяти
  CMemObj *pMO2 = new CMemObj();
  printf("Custom heap after second new: %d block(s),
       handle=0x%08x\n",CMemObj::s_dwBlocks,
       CMemObj::s_hPrivateHeap);

  // Удалить второй объект из динамической области памяти
  delete pMO2;
  printf("Custom heap after first delete: %d block(s),
       handle=0x%08x\n",CMemObj::s_dwBlocks,
       CMemObj::s_hPrivateHeap);

  // Удалить первый объект из динамической области памяти (это вызовет
  // уничтожение динамической области памяти)
  delete pMO;
  printf("Custom heap after second delete: %d block(s),
       handle=0x%08x\n",CMemObj::s_dwBlocks,
       CMemObj::s_hPrivateHeap);

  return 1;
}
```

2. Вызовите это приложение на выполнение и ознакомьтесь с его выводом. Результаты работы приложения должны выглядеть примерно таким образом:

```
Custom heap after first new: 1 block(s), handle=0x00440000
Custom heap after second new: 2 block(s), handle=0x00440000
Custom heap after first delete: 1 block(s), handle=0x00440000
Custom heap after second delete: 0 block(s), handle=0x00000000
```

3. В этом коде автоматически происходит распределение и отмена распределения памяти из закрытой динамической области памяти по мере необходимости. В классе `CMemObj` для хранения указателя на определенную в программе закрытую динамическую область памяти используется статическая переменная `s_hPrivateHeap`. Во время запуска программы происходит инициализация этой переменной и ей присваивается `NULL`. Если выполняется вызов оператора `new` класса `CMemObj`, производится проверка переменной `s_hPrivateHeap` для определения того, имеет ли она значение `NULL`. В случае положительного ответа создается новая динамическая область памяти с использованием функции

HeapCreate, и дескриптор этой области присваивается переменной s_hPrivateHeap. После этого в целях выполнения запроса распределяется блок памяти из динамической области памяти, на которую указывает переменная s_hPrivateHeap. В случае успешного завершения этого действия увеличивается значение еще одной статической переменной, s_dwBlocks, которая предназначена для отслеживания количества блоков, распределенных из динамической области памяти.

4. После вызова оператора delete класса CMemObj, прежде всего, происходит отмена распределения рассматриваемого блока с использованием функции HeapFree. В случае успешного завершения этого действия уменьшается значение переменной s_dwBlocks для указания на то, что теперь в данной динамической области памяти количество распределенных блоков уменьшилось на единицу. После того как значение переменной s_dwBlocks достигает 0, освобождается сама динамическая область памяти путем вызова функции HeapDestroy. Это позволяет исключить расходование ресурсов памяти на сопровождение неиспользуемой динамической области памяти. Переменной s_hPrivateHeap после уничтожения динамической области памяти присваивается NULL, что позволяет успешно вызывать оператор new класса CMemObj из другой строки кода и после уничтожения закрытой динамической области памяти. При обнаружении такого вызова будет просто снова создана закрытая динамическая область памяти.
5. У читателя может возникнуть вопрос, почему в этом коде для хранения дескриптора закрытой динамической области памяти и счетчика блоков используются статические переменные. Причина применения такого подхода состоит в том, что в данной программе все экземпляры класса CMemObj совместно используют одну и ту же закрытую динамическую область памяти. Если бы эти переменные не были объявлены как статические, то каждый экземпляр CMemObj получил бы собственную закрытую динамическую область памяти, а такое решение является не только непроизводительным, но и нелогичным. Вся идея данной программы состоит в том, что различные экземпляры CMemObj должны распределять память из общей закрытой динамической области памяти. А поскольку все объекты будут иметь одинаковый размер и оставаться относительно небольшими, то рассматриваемый подход к использованию динамической области памяти является эффективным и обоснованным.
6. Может оказаться также не совсем понятным, почему в этом коде используется отдельная статическая переменная для отслеживания количества блоков, распределенных из закрытой динамической области памяти. В конечном итоге, разве нельзя получить ту же информацию из функции HeapWalk API-интерфейса, не используя отдельную статическую переменную? Да, такой подход к организации работы действительно возможен, но он был бы чрезвычайно неэффективным. При этом подходе при выполнении каждой операции распределения и отмены распределения приходилось бы проходить по всей динамической области памяти и подсчитывать количество блоков, из которых она состоит, тщательно следя за тем, чтобы не учитывались те распределенные блоки, которые применяются для сопровождения самой динамической области памяти (так называемые *издержки* динамической области памяти). Если в приложении выполняется большое количество операций распределения, такая организация работы отрицательно влияет на производительность и может даже неблагоприятно отразиться на показателях загрузки процессора.

В следующем упражнении показано, как используется расширенная процедура для распределения пользовательских динамических областей памяти из среды программы SQL Server. В этих динамических областях памяти можно сохранить некоторые данные, а затем выполнить выборку этих данных с помощью запроса. Поскольку при выполнении данного упражнения необходимо подключаться к программе SQL Server с помощью отладчика, то его следует выполнять лишь на компьютере, предназначенном для экспериментов или разработки, и в идеальном случае вы должны быть его единственным пользователем.

Упражнение 4.10. Распределение динамических областей памяти из среды программы SQL Server

1. Вначале скопируйте файл `xp_array.dll` из подкаталога `CH04\xp_array\release` компакт-диска, прилагаемого к данной книге, в подкаталог `binn` главного инсталляционного каталога SQL Server.
2. Зарегистрируйте расширенные процедуры, содержащиеся в библиотеке `xp_array`, в программе SQL Server, открыв и выполнив сценарий `xp_array.sql` из подкаталога `CH04\xp_array` компакт-диска.
3. Подключитесь к программе SQL Server с помощью отладчика WinDbg. После того как на экране появится приглашение к вводу команд отладчика WinDbg, наберите команду `!heap` для вывода на экран списка динамических областей памяти, которые распределены в настоящее время в процессе SQL Server. В этом списке, скорее всего, будет довольно много элементов, возможно, даже порядка 20 или 30. Запишите точное количество элементов, после этого введите `g` и нажмите <Enter>, чтобы предоставить программе SQL Server возможность продолжить работу.
4. Теперь загрузите сценарий `arrays.sql` из подкаталога `CH04\xp_array` компакт-диска в программу Query Analyzer и выполните его. Этот сценарий устанавливает целый ряд определяемых пользователем функций, которые значительно упрощают вызов расширенных процедур, установленных в предыдущем пункте.
5. Загрузите и вызовите на выполнение сценарий `leapheap.sql` из подкаталога `CH04\xp_array`. Сценарий `leapheap.sql` создает массив, основанный на динамической области памяти, с использованием установленных ранее расширенных процедур, а затем загружает в этот массив часть столбцов из таблицы `Northwind..Orders`.
6. Возвратитесь к отладчику и нажмите клавиши <Ctrl+Break>, чтобы прекратить выполнение сценария, затем введите команду `!heap` в приглашении к вводу команд, чтобы снова получить список динамических областей памяти, которые были распределены в процессе SQL Server. Этот список должен частично совпадать с тем, который был получен раньше, поскольку по умолчанию в коде процедуры `xp_array` распределения осуществляются из применяемой по умолчанию динамической области памяти процесса; в этой расширенной процедуре не создается закрытая динамическая область памяти.
7. Введите `g` и нажмите <Enter>, чтобы предоставить программе SQL Server возможность продолжить работу.

8. Возвратитесь в программу Query Analyzer и отредактируйте сценарий `leapheap.sql`, внеся изменение в строку

   ```
   SET @hdl=fn_createarray(1000, 0)
   ```

   Теперь она должна выглядеть следующим образом:

   ```
   SET @hdl=fn_createarray(1000, 1)
   ```

   Такое изменение приводит к тому, что при первом вызове функции `fn_createarray` в коде `xp_array` распределяется собственная динамическая область памяти.

   Примечание. Эту опцию не следует использовать при вызове кода `xp_array` из многочисленных соединений. Если существует вероятность того, что процедуру `xp_array` будут одновременно вызывать многочисленные рабочие потоки, то всегда следует использовать предусмотренную по умолчанию динамическую область памяти системы.

9. С помощью мыши выберите весь текст сценария в программе Query Analyzer вплоть до вызова функции `fn_destroyarray` (но не включая его), затем нажмите клавишу <F5> для выполнения выбранного кода. Не включая вызов функции `fn_destroyarray` в состав команд, передаваемых на сервер, мы оставляем на данное время распределенными и сам массив, и его динамическую область памяти. Примите наши поздравления. Вы только что создали свою собственную закрытую динамическую область памяти в пространстве процесса SQL Server!

10. Возвратитесь к отладчику, нажмите клавиши <Ctrl+Break> и снова введите команду `!heap` в командном окне. Вы должны обнаружить, что в полученном списке появилась новая динамическая область памяти.

11. Введите команду `!heap 0`, чтобы вывести на экран информацию о сегментах каждой динамической области памяти. Динамическая область памяти может состоять из отдельных сегментов, количество которых может достигать 64. Каждый раз, когда операционной системе Windows потребуется увеличить объем динамической области памяти, для этой области может быть распределен новый сегмент. А последняя динамическая область памяти в списке (только что созданная вами) должна иметь лишь один сегмент.

12. Чтобы убедиться в том, что это и есть созданная пользователем динамическая область памяти, проведем в ней поиск некоторых данных. В информации о сегменте, отведенном для новой динамической области памяти, должны быть указаны начальный и конечный адреса. Эта информация должна выглядеть примерно таким образом:

    ```
    23:   10010000
       Segment at 10010000 to 10020000 (00000500 bytes committed)
    ```

    Числа, обозначенные полужирным шрифтом, представляют собой начальный и конечный адреса сегмента.

13. Воспользуйтесь данными о начальном и конечном адресах для поиска в массиве записи, соответствующей значению "TOMSP" поля `CustomerId` базы данных `Northwind`, следующим образом:

    ```
    s -a 10010000 10020000 'TOMSP'
    ```

    Команда `s` отладчика WinDbg позволяет выполнить поиск значения данных в области памяти. Параметр `-a` этой команды указывает, что поиск должен

выполняться в формате строки ANSI. Два адреса памяти обозначают начало и конец диапазона поиска, и, безусловно, символьная строка в одинарных кавычках задает искомое значение. После вызова на выполнение этой команды должно быть обнаружено, как на экране отображается адрес памяти, по которому в пользовательской динамической области памяти расположено это значение. Хотя такая проверка не является исчерпывающей, она вполне успешно показывает, что это и есть та динамическая область памяти, которая используется в рассматриваемых расширенных процедурах. Введите `g` и нажмите <Enter>, чтобы предоставить программе SQL Server возможность продолжить работу.

14. Вернитесь в программу Query Analyzer, загрузите сценарий `leakheaps.sql` из подкаталога `CH04\xp_array` компакт-диска и выполните его. Это приведет к созданию 128 новых динамических областей памяти в процессе SQL Server.
15. Возвратитесь в отладчик WinDbg и нажмите клавиши <Ctrl+Break>, чтобы остановить процесс SQL Server. Наберите команду `!heap` в приглашении к вводу команд. В списке динамических областей памяти должны появиться вновь созданные динамические области памяти.
16. Формально максимальное количество динамических областей памяти, которые можно распределить в процессе, не установлено. Автор не может представить себе реальную ситуацию, в которой потребовалось бы распределить такое количество динамических областей памяти, какое было распределено в данном упражнении, но следует помнить, что с формальной точки зрения такая возможность существует. Если нужно будет узнать, какие динамические области памяти распределены в процессе и что они содержат, то можно вполне начать с использования команды `!heap` отладчика WinDbg.
17. Введите `q` и нажмите клавишу <Enter>, чтобы прекратить работу программу SQL Server под управлением отладчика, после этого выйдите из отладчика.
18. В случае необходимости перезапустите программу SQL Server.

Динамические области памяти. Резюме

Динамическая область памяти состоит из одной или нескольких страниц зарезервированного пространства, которое может применяться для вторичного распределения с помощью диспетчера динамической области памяти. Динамическая область памяти наиболее хорошо подходит для использования при распределении памяти для относительно небольших объектов и структур, имеющих примерно одинаковые размеры. Динамическая область памяти не должна служить для распределения блоков с размерами порядка 1 Мбайт или больше, поскольку для этой цели в наибольшей степени подходят такие функции виртуальной памяти, как `VirtualAlloc`.

В приложении пользовательские динамические области памяти могут создаваться по мере необходимости. Одним из наиболее важных решений, которые должны быть приняты при создании пользовательской динамической области памяти, является то, следует ли упорядочивать доступ к динамической области памяти с помощью диспетчера этой области. Если в приложении имеются многочисленные потоки, а в этих потоках осуществляются собственные операции распределения из общей динамической области памяти, то доступ к ней должен быть упорядочен для

предотвращения возможности искажения этой динамической области памяти. Если же приложение является однопотоковым или в нем предусмотрены собственные механизмы синхронизации потоков, то вполне можно без опасений отменить упорядочение доступа к пользовательской динамической области памяти.

Динамические области памяти. Вопросы для самопроверки

1. Какая функция API-интерфейса Win32 применяется для распределения блока из динамической области памяти?
2. Подтвердите или опровергните следующее утверждение. Перед уничтожением в процессе динамической области памяти с помощью функции `HeapDestroy` необходимо вызвать функцию `HeapFree` для освобождения всех блоков памяти, которые были распределены из этой области.
3. Распространяются ли на операции распределения блоков из динамической области памяти ограничения, определяемые размерами страницы системы и степенью детализации распределения, применяемой в системе?
4. Подтвердите или опровергните следующее утверждение. Производительность приложения можно повысить, не применяя параметр `HEAP_NO_SERIALIZE` при создании закрытой динамической области памяти.
5. Подтвердите или опровергните следующее утверждение. При распределении в приложении памяти из динамической области памяти существует возможность сэкономить память за счет того, что она не будет в результате закрепления расходовать реальную память, но для этого нужно правильно указать параметры функции `HeapAlloc`.
6. Чему равен заданный по умолчанию размер применяемой по умолчанию динамической области памяти процесса?
7. Подтвердите или опровергните следующее утверждение. Диспетчер динамической области памяти не перемещает блок динамической области памяти после его распределения с помощью функции `HeapAlloc`, поскольку может оказаться, что в программе уже определены указатели, которые ссылаются на этот блок.
8. Какой максимальный объем должен быть указан в функции `HeapCreate`, если требуется создать динамическую область памяти, объем которой увеличивается автоматически по мере того, как из этой области распределяются блоки?
9. Какая функция API-интерфейса Win32 возвращает дескриптор заданной по умолчанию динамической области памяти процесса?
10. Подтвердите или опровергните следующее утверждение. Заданная по умолчанию динамическая область памяти процесса используется в нескольких функциях API-интерфейса Win32.

11. Какой параметр должен быть передан в функцию HeapAlloc, если требуется, чтобы эта функция заполнила вновь распределенную страницу нулями?
12. Подтвердите или опровергните следующее утверждение. Если попытка уничтожить определяемую пользователем динамическую область памяти оканчивается неудачей, эта область остается в памяти до тех пор, пока не завершится сам процесс.
13. Возможно ли отменить упорядочение доступа для отдельных распределений из динамической области памяти, в которой применяются средства упорядочения доступа?
14. Подтвердите или опровергните следующее утверждение. Попытка уничтожить заданную по умолчанию динамическую область памяти процесса вызывает исключительную ситуацию, связанную с нарушением доступа.
15. Какие типы операций распределения в наилучшей степени подходят для динамических областей памяти?
16. Подтвердите или опровергните следующее утверждение. По умолчанию в операционной системе Windows не применяются средства упорядочения доступа к заданной по умолчанию динамической области памяти процесса, но в приложении можно создать закрытую динамическую область памяти и разрешить применение в ней средств упорядочения доступа, если это потребуется.
17. Как изменяется поведение диспетчера динамической области памяти, если функция HeapAlloc вызывается с параметром HEAP_GENERATE_EXCEPTIONS?
18. Позволяет ли инструментальное средство Perfmon контролировать размер системного файла подкачки?
19. Подтвердите или опровергните следующее утверждение. В некоторых обстоятельствах диспетчер динамической области памяти может активизировать исключительную ситуацию STATUS_NO_MEMORY.

Совместно используемая память

В данном разделе рассматриваются средства совместно используемой памяти операционной системы Windows. *Совместно используемой памятью* называется область памяти, доступная многочисленным процессам или представленная в пространстве виртуальных адресов многочисленных процессов. Это — наиболее удобное средство доступа к памяти в той ситуации, когда нужно обеспечить быстрый обмен данными между многочисленными процессами. Поскольку обмен данными происходит с помощью совместно используемых страниц виртуальной памяти, то в каждом процессе, который получает доступ к совместно используемой памяти, должно быть известно, как интерпретировать полученную информацию и как с ней работать. В отличие от данных, которые поступают через сокет TCP/IP или, скажем, передаются в сообщении Windows, данные, доступ к которым предоставляется с помощью совместно используемой памяти, состоят из страниц распределенных в пространстве адресов процесса, для которых в результате закре-

пления выделена реальная память. Процесс должен иметь определенную информацию о том, что содержится в этих страницах, чтобы иметь возможность использовать данные страниц.

Совместно используемая память. Основные термины и определения

- **Совместно используемая память.** Память, доступ к которой предоставляется многочисленным процессам или которая распределена в пространстве виртуальных адресов многочисленных процессов.
- **Файл, отображаемый на память.** Дисковый файл, который отображен на виртуальную память таким образом, что служит в качестве реальной памяти для рассматриваемой виртуальной.
- **Объект секции памяти.** Объект привилегированного режима, предназначенный для реализации совместно используемой памяти и отображаемых на память файлов.

Совместно используемая память. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к совместно используемой памяти, приведены в табл. 4.9.

Таблица 4.9. Основные функции API-интерфейсов, относящиеся к совместно используемой памяти

Функция	Описание
`CreateFileMapping`	Создать объект отображения файла (объект секции памяти) для применения с совместно используемой памятью или с отображаемым на память файлом
`MapViewOfFile`	Отобразить представление файла на память таким образом, чтобы файл служил в качестве реальной памяти для виртуальной. В качестве отображаемого на память файла может применяться файл на диске или системный файл подкачки
`FlushViewOfFile`	Записать модифицированные страницы в представление отображенного файла на диске

Программа SQL Server и совместно используемая память

Средства совместно используемой памяти находят широкое применение в программе SQL Server. Основным примером такого применения является совместно используемая память Net-Library. Если клиентское приложение эксплуатируется

на том же компьютере, что и SQL Server, то его можно подключить к серверу с помощью совместно используемой памяти Net-Library, например, указав имя сервера в виде . (точки), или (local), или поставив перед именем сервера/экземпляра строку lpc:. Это означает, что для обмена данными между клиентом и сервером вполне можно применять простой, совместно используемый буфер памяти, а не организовывать связь между ними на основе такого набора протоколов, как TCP/IP или Named Pipes, и полного стека сетевых протоколов. Поскольку оба процесса эксплуатируются на одном и том же компьютере, указанное решение является не только более целесообразным, но и гораздо более эффективным по сравнению с решением на основе стека сетевых протоколов.

Заслуживает внимания вопрос о том, как координируется доступ к этой совместно используемой области памяти, иными словами, как исключить возможность того, что сервер прочитает клиентские данные, прежде чем они будут готовы, и наоборот? Такая координация осуществляется с использованием именованного объекта события. Вернемся к обсуждению объектов событий в главе 3. Для того чтобы синхронизировать доступ к области памяти, обозначенной как совместно используемая память Net-Library, клиент и сервер обозначают какой-то объект события как сигнальный, сообщая второму участнику обмена данными, что в этой ситуации операция доступа к буферу стала безопасной. Итак, сервер переходит в состояние ожидания освобождения объекта события, вызвав функцию WaitForSingleObject, а клиент обозначает это событие как сигнальное, подготовив для сервера все условия доступа к данному буферу. После этого сам клиент вызывает функцию WaitForSingleObject и ожидает той ситуации, когда объект события станет сигнальным. После того как сервер заканчивает свою работу с буфером, он отмечает событие как сигнальное, после чего клиент снова приступает к работе с буфером. Такая последовательность действий продолжается до тех пор, пока клиент остается подключенным к серверу.

Объекты секции памяти

Наиболее важным объектом привилегированного режима, применяемым для реализации совместно используемой памяти, является *объект секции памяти*. В терминологии API-интерфейса Win32 объект секции памяти именуется как *объект отображения файла*. Операция отображения файла сводится к тому, что содержимое файла ассоциируется с диапазоном адресов виртуальной памяти, после чего этот файл начинает служить в качестве реальной памяти для данного диапазона. В качестве самого файла может применяться любой файл на диске или системный файл подкачки. Но независимо от того, какая область реальной памяти лежит в основе отображения файла, к поддерживаемой этой областью совместно используемой памяти могут иметь доступ многочисленные процессы.

Поскольку объект секции памяти может быть открыт одним процессом или многими процессами, его не обязательно считать эквивалентным совместно используемой памяти. Несмотря на то что объект секции памяти может применяться для реализации совместно используемой памяти, он может также применяться лишь в одном процессе для отображения любого файла в виртуальную память.

Для организации совместно используемой памяти можно подключить объект секции памяти к открытому файлу на диске, чтобы создать отображенный файл или закрепленную виртуальную память. Объект секции памяти, который подключен к виртуальной памяти, рассматривается как отображенный на файл подкачки, поскольку страницы этого объекта могут записываться в системный файл подкачки в случае необходимости. Следует учитывать, что операционная система Windows может функционировать без файла подкачки, поэтому может оказаться, что указанные страницы отображаются на физическую память. Так же, как и закрытые закрепленные страницы, совместно используемые закрепленные страницы всегда заполняются нулями при первом доступе к ним любого процесса. Совместно используемая страница заполняется нулями только один раз, независимо от количества процессов, получающих к ней доступ.

Файлы, отображаемые на память

В операционной системе Windows предусмотрен ряд функций API-интерфейса, которые поддерживают отображение файла на область виртуальных адресов. В приложении эти функции могут использоваться для упрощения реализации операций ввода-вывода в файл благодаря тому, что этот файл, с точки зрения пользователя пространства виртуальных адресов, выглядит просто как область памяти. Вместо проведения операций ввода-вывода в файл в приложении осуществляются операции чтения и записи в виртуальной памяти, связанной с файлом.

Для организации работы с файлом, отображаемым на память, в приложении необходимо выполнить описанные ниже шаги.

1. Вызвать функцию `CreateFile` для создания локального по отношению к процессу дескриптора файла, находящегося на жестком диске.
2. Вызвать функцию `CreateFileMapping`, чтобы создать объект отображения файла для рассматриваемого файла.
3. Вызвать функцию `MapViewOfFile`, чтобы отобразить файл на пространство адресов процесса. Указатель, возвращенный функцией `MapViewOfFile`, представляет собой начальный адрес области памяти, отображенной на файл.

Если же в приложении не требуется отображать файл на диск в целях осуществления операций ввода-вывода, а требуется лишь создать область совместно используемой памяти, то в приложении достаточно выполнить два описанных ниже обязательных шага.

1. Вызвать функцию `CreateFileMapping`, чтобы создать объект отображения файла. Передать в качестве дескриптора файла значение `INVALID_HANDLE_VALUE`. Это приведет к тому, что область совместно используемой памяти будет отображена на системный файл подкачки. Если область совместно используемой памяти отображается на системный файл подкачки, то функции `CreateFileMapping` необходимо сообщить, насколько велика эта область. (Если для отображения служит файл на диске, то пропуск параметров с размерами файлов приводит к тому, что функция

`CreateFileMapping` создаст объект отображения с учетом физического размера файла.)

2. Вызвать функцию `MapViewOfFile`, чтобы получить указатель на область совместно используемой памяти.

Отображения файлов образов

При запуске процесса операционная система Windows открывает исполняемый файл приложения и определяет размер его кода и данных. Затем ОС Windows резервирует область пространства адресов процесса, достаточно большую для того, чтобы в ней можно было разместить код и данные исполняемого файла, после чего определяет в качестве реальной памяти для этих адресов сам исполняемый файл. Как было указано в разделе "Виртуальная память", страницы исполняемого файла обозначаются атрибутом `PAGE_EXECUTE`, а страницы данных – атрибутом `PAGE_WRITECOPY`. Все экземпляры данного конкретного исполняемого файла совместно используют одну и ту же область реальной памяти – сам исполняемый файл. Если некоторый процесс вносит изменение в одну из страниц данных файла (например, присваивает значение глобальной переменной), операционная система Windows создает копию страницы данных и сообщает процессу, что в нем должна использоваться новая страница вместо старой. Это функциональное средство копирования при записи позволяет свести к минимуму расходование пространства адресов и вместе с тем позволяет приложениям вносить изменения в применяемые ими страницы данных каждый раз, когда в этом возникает необходимость.

Двоичный образ исполняемого файла (файл EXE или DLL), который служит в качестве реальной памяти для области виртуальных адресов, представляет собой своего рода файл, отображаемый на память. Кроме того, по аналогии с тем, как приложение отображает диапазон адресов на файл, находящийся на жестком диске, операционная система Windows использует свои собственные средства отображения файлов для упрощения загрузки и обработки образов исполняемых файлов.

Следует отметить, что при использовании некоторых типов носителей операционная система Windows вынуждена копировать весь образ исполняемого файла в виртуальную память, поскольку не может разрешить ему оставаться на этом носителе. Например, в виртуальную память полностью копируется образ, загружаемый с гибкого диска. Такая операция выполняется для того, чтобы можно было продолжить работу программ начальной установки, загружаемых с гибкого диска, даже после того, как в ходе начальной установки вместо одного гибкого диска будет вставлен другой. Загрузка образа со сменных носителей других типов, таких как компакт- или сетевой диск, не вынуждает операционную систему Windows копировать образ в виртуальную память при условии, что при редактировании связей исполняемого файла программы самой операционной системы Windows не применялся параметр `/SWAPRUN:NET` или `/SWAPRUN:CD`.

Упражнения

В следующем упражнении показано, как применяется совместно используемая память для обеспечения совместного использования данных в нескольких процессах, а также описано, каким образом можно обеспечить с помощью синхронизационного объекта потокобезопасный доступ к этим данным. Будет создано одно приложение, а затем запущены его многочисленные экземпляры для демонстрации того, как может быть организовано совместное использование данных в этих процессах с помощью совместно используемой памяти.

Упражнение 4.11. Применение совместно используемой памяти для обеспечения совместного использования данных в разных процессах

1. Скопируйте пример приложения `sharedmem_client` из подкаталога `CH04\sharedmem_client` компакт-диска, прилагаемого к данной книге, на жесткий диск и загрузите его в среду разработки VC++ (MSDEV). Откомпилируйте и отредактируйте связи этого приложения таким образом, чтобы файл `sharedmem_client.exe` был записан в подкаталог `Release`.

   Еще один вариант состоит в том, что можно просто скопировать этот исполняемый файл с компакт-диска, если вы не заинтересованы в том, чтобы вначале проводить его компиляцию.

2. Запустите программу Explorer и перейдите в подкаталог `Release`, в котором находится файл `sharedmem_client.exe`. Дважды щелкните на имени этого исполняемого файла, чтобы его запустить.

3. Несколько раз введите букву `Y`, чтобы разрешить этому приложению продолжать предусмотренные в нем итерации цикла модификации данных. Вы обнаружите, что в данном приложении осуществляется выборка целого числа из совместно используемой памяти, после чего оно увеличивается и снова записывается в совместно используемую память.

4. После нескольких итераций этого цикла оставьте файл `sharedmem_client.exe` в рабочем состоянии и возвратитесь в программу Explorer. Второй раз щелкните на имени файла `sharedmem_client.exe` в программе Explorer, чтобы запустить второй экземпляр этого исполняемого файла. Вы заметите, что данное приложение на первый взгляд кажется "зависшим".

5. Возвратитесь к первому экземпляру приложения и введите `Y`, чтобы разрешить ему продолжить работу. Теперь будет казаться "зависшим" этот экземпляр.

6. Возвратитесь ко второму экземпляру, и вы увидите, что он наконец-то начал работать. Точнее, в рассматриваемом приложении используется именованный объект события для синхронизации доступа к области совместно используемой памяти. Это означает, что эту область в один момент времени может модифицировать только один экземпляр приложения. Если один экземпляр приложения получает контроль над областью совместно используемой памяти, то второй должен ожидать завершения его работы и только после этого приступать к выполнению. Вы обнаружите, что после каждого выполнения цикла модификации значение целочисленной переменной экземпляра увеличивается на 1, независимо от того, в каком процессе выполнена операция увеличения этого значения.

7. Можно запустить такое количество экземпляров программы sharedmem_client.exe, какое вы захотите, а эффект остается одним и тем же — только одному из них будет разрешено в один момент выполнять чтение или запись в совместно используемой памяти.
8. Введите N в каждом из запущенных экземпляров программы sharedmem_client.exe, для того чтобы выполнить их останов.

Для любознательных читателей в листинге 4.13 приведен код программы sharedmem_client.exe.

Листинг 4.13. Приложение, в котором применяется совместно используемая память, и доступ к этой памяти синхронизируется с помощью объекта события

```
// sharedmem_client.cpp. Приложение, в котором для предоставления
//                       разным процессам совместного доступа к данным
//                       используется общая память, а для синхронизации
//                       доступа к данным служит именованный объект
//                       события
//

#include "stdafx.h"
#include "windows.h"
#include "conio.h"

#define SHARED_MEM_NAME "GGSharedMem"
#define EVENT_NAME "GGSharedMemEvent"

int main(int argc, char* argv[])
{
  // Создать объект события для синхронизации доступа к общей памяти
  HANDLE hEvent=CreateEvent(NULL,false,true,EVENT_NAME);

  if (INVALID_HANDLE_VALUE==hEvent) return 0;

  LPVOID lpSharedMemory;
  DWORD dwValue;

  HANDLE hMapFile;

  // Создать отображение файла на основе системного файла подкачки
  hMapFile = CreateFileMapping(INVALID_HANDLE_VALUE,
                               NULL,
                               PAGE_READWRITE,
                               0,
                               0x1000,
                               SHARED_MEM_NAME);

  if (NULL == hMapFile)
  {
    printf("Could not create file mapping object.
        Last error=%s\n",GetLastError());
    return 1;
  }

  // Получить указатель на совместно используемую память
```

```
  lpSharedMemory = MapViewOfFile
      (hMapFile, FILE_MAP_ALL_ACCESS, 0, 0, 0);

  if (NULL == lpSharedMemory)
  {
    printf("Could not map view of file.
         Last error=%s\n",GetLastError());
     return 1;
  }

  __try
  {
    char ch='N';
do {
      // Ожидать перехода объекта в сигнальное состояние.
      // Поскольку объект представляет собой отключаемое автоматически
      // событие, это также приведет к его переустановке в несигнальное
      // состояние
      WaitForSingleObject(hEvent,INFINITE);

       // Прочитать значение из совместно используемой области памяти
       dwValue = *((LPDWORD) lpSharedMemory);
       printf("\n\ndwValue READ = %d for process 0x%08x\n",
              dwValue,GetCurrentProcessId());

      // Увеличить значение локальной копии переменной
      dwValue++;

       // Переписать содержимое локальной копии переменной обратно
       // в совместно используемую память
       *((LPDWORD) lpSharedMemory) = dwValue;
      printf("dwValue WRITE = %d for process 0x%08x\n",
              dwValue,GetCurrentProcessId());

      printf("Continue? ");
      ch=getche();

      // Перевести событие в сигнальное состояние (теперь доступ
      // может быть предоставлен другому клиенту)
      SetEvent(hEvent);

    } while ('Y'==toupper(ch));
  }

  // Обеспечить выполнение кода очистки
  finally
  {
    // Отменить представление отображения файла
    if (!UnmapViewOfFile(lpSharedMemory))
     {
       printf("Could not unmap view of file.
           Last error=%d\n",GetLastError());
     }

     // Закрыть дескриптор синхронизационного события
    CloseHandle(hEvent);
```

```
  }

  return 0;
}
```

9. В этом коде вначале создается именованный объект события. При каждой попытке создать в процессе именованный объект, который уже существует, процесс получает дескриптор существующего объекта, локальный по отношению к данному процессу. Это означает, что все экземпляры программы `sharedmem_client.exe` применяют один и тот же объект события для синхронизации доступа к области совместно используемой памяти.
10. Затем создается именованный объект отображения файла, который отображается на системный файл подкачки. Как и для объекта события, при предпринимаемой в каждом процессе попытке создать этот именованный объект сам процесс получает дескриптор одного и того же объекта привилегированного режима, если он уже существует. Это означает, что в нескольких экземплярах данного исполняемого файла применяется ссылка на один и тот же объект отображения файла, и поэтому применяется одна и та же область совместно используемой памяти.
11. После создания объекта отображения файла осуществляется настройка самой совместно используемой памяти с помощью вызова функции `MapViewOfFile`. Указатель, возвращенный этой функцией, обозначает начало области совместно используемой памяти.
12. Вызывается функция `WaitForSingleObject` для организации ожидания освобождения памяти с помощью события. Поскольку событие было создано как находящееся в сигнальном состоянии (см. третий параметр функции `CreateEvent`), функция `WaitForSingleObject` немедленно выполняет возврат после ее вызова в первом экземпляре программы `sharedmem_client.exe`. Поскольку это событие было создано как событие с автоматическим отключением (см. второй параметр функции `CreateEvent`), событие немедленно возвращается в несигнальное состояние, как только ожидание его освобождения с помощью функции `WaitForSingleObject` завершается успешно. Благодаря этому исключается возможность одновременного получения доступа к области совместно используемой памяти со стороны многочисленных экземпляров процесса. Каждый из них должен ожидать, пока тот экземпляр, в котором успешно завершилось ожидание освобождения события, не обозначит это событие как сигнальное.
13. После этого осуществляется выборка значения из совместно используемой памяти. Для этого просто производится разадресация и приведение к требуемому типу типа данных первого двойного слова `DWORD` в буфере. Затем в текущем экземпляре увеличивается значение локальной целочисленной переменной, содержащей полученное значение, и результат записывается обратно в буфер совместно используемой памяти.
14. Функционирование кода приложения заканчивается тем, что пользователю поступает запрос, следует ли продолжить выполнение цикла обновления. Если пользователь вводит букву `Y`, то предпринимается попытка выполнить еще одну последовательность операций увеличения целочисленного значения и вывода содержимого первого двойного слова `DWORD` из области совместно используемой памяти. После получения ответа от пользователя данное событие

немедленно отмечается как сигнальное, независимо от ответа пользователя. Если освобождения события ожидает еще один экземпляр приложения, то он получает разрешение на выполнение еще до того, как в текущем экземпляре можно будет выполнить еще одну итерацию цикла. После этого текущий экземпляр находится в состоянии ожидания до тех пор, пока следующий экземпляр не закончит работу с областью совместно используемой памяти и не отметит событие как сигнальное. Указанные действия могут продолжаться до бесконечности, пока не будет завершена работа всех экземпляров программы.

ПРИМЕЧАНИЕ. Автор обычно не рекомендует разрабатывать код приложений таким образом, чтобы они переходили в состояние ожидания освобождения какого-то объекта привилегированного режима на неопределенно долгое время, когда требуется обеспечить получение входных данных от пользователя, поскольку при таком подходе может быть заблокирован доступ других приложений к объекту на бесконечное время. В данном примере указанный подход был применен для того, чтобы было проще следить за развитием ситуации.

В последнем упражнении показано, как создать объект совместно используемой памяти, к которому будет предоставлен доступ целому ряду процессов, а затем изучить работу этого объекта с помощью инструментального средства WinObj из комплекта Platform SDK.

Упражнение 4.12. Использование инструментального средства WinObj для изучения именованных объектов совместно используемой памяти

Для создания объекта совместно используемой памяти, который можно будет изучать с помощью программы WinObj, применяется приложение SuperRecorder. Это приложение было первоначально создано автором примерно десять лет тому и с тех пор несколько раз дорабатывалось. Чтобы понять, как и почему в нем применяется совместно используемая память, позвольте кратко описать его историю и развитие.

Если в числе моих читателей есть ветераны программирования, они могут помнить такую вспомогательную программу Windows 3.x, как Recorder. Программа Recorder позволяла регистрировать события, активизируемые при манипуляциях с мышью и клавиатурой, в виде макрокоманд и воспроизводить эти макрокоманды в любом приложении с помощью некоторой комбинации клавиш. В то время возможность регистрировать события, активизируемые и с помощью клавиатуры, и с помощью мыши, предоставлялась далеко не всякой программой, а поскольку автор всегда много работал с клавиатурой, то очень интенсивно использовал указанное инструментальное средство и определил в своей системе Win 3.x множество макрокоманд.

Приблизительно к 1992 году автор пришел к выводу, что было бы весьма неплохо реализовать функциональные средства программы Recorder в виде компонента какого-то типа, который можно было бы подключать к любому приложению и мгновенно дополнять его возможности средствами создания программируемых макрокоманд для клавиатуры/мыши. Автор изучил всевозможные источники информации о программах, имеющихся в то время, и не нашел для Windows никаких готовых решений, подобных искомому, поэтому приступил к его самостоятельной разработке.

Автор в 1980-х годах уже разработал несколько редакторов для программистов, которые включали функциональные средства поддержки макрокоманд с различными

уровнями сложности (например, Cheetah, TurboEdit, TEdit и т.д.), поэтому имел достаточно полное представление о том, что ему хотелось бы сделать и как можно справиться с техническими проблемами хранения списков входных событий. Вопрос заключался в том, как сделать это в среде Windows.

Автор приступил к исследованию этого вопроса и обнаружил, что средства регистрации и воспроизведения сообщений Windows, на которых он вначале хотел построить свое приложение, действуют недостаточно надежно. Передача сообщения WM_KEYDOWN с кодом клавиши VK_SHIFT, за которым следует код клавиши VK_A, не всегда приводит к тому, что в текущем приложении эмулируется ввод прописной буквы A. Эти эксперименты автор проводил, когда еще не существовали инструментальные средства наподобие функции SendKeys языка VB (и в конечном итоге это был механизм, предназначенный для работы только с клавиатурой), поэтому автор оставил эти попытки и начал искать другие пути.

В конечном итоге автору удалось открыть, на чем основана магическое действие самой программы Recorder — в ней использовалась комбинация глобальных обработчиков системных прерываний (для перехвата событий от клавиатуры и мыши) и функции регистрации/воспроизведения API-интерфейса ведения журнала. Глобальные перехватчики системных прерываний позволяли этой программе перехватывать события от клавиатуры и мыши по принципу, аналогичному процедуре обслуживания прерываний в DOS, которая использовалась в некоторых редакторах для программистов, написанных автором. А функции API-интерфейса ведения журнала позволяли программе Recorder надежно регистрировать и воспроизводить события и от мыши, и от клавиатуры. Автор решил применить такой же подход и в разрабатываемом им компоненте. Он начал с создания библиотеки DLL (для обработчиков системных прерываний требуется библиотека DLL, в которой находились бы их функции обратного вызова), после этого заключил функции, предоставляемые этой библиотекой DLL, в отдельный компонент.

Автор назвал такой пакет, состоящий из библиотеки и компонента, WinMacro и распространял его на коммерческой основе в течение нескольких лет через CompuServe и другие форумы еще до того, как началось действительно крупномасштабное развертывание World Wide Web. Этот пакет позволял разработчику Windows подключать к своему приложению один компонент и мгновенно получать в этом приложении программный доступ ко всем функциональным средствам вспомогательной программы Recorder. В приложении можно было в любое время начать или остановить регистрацию событий, регистрировать события не только от мыши, но и от клавиатуры, а затем снова их воспроизводить после поступления определенного события от мыши/клавиатуры или после вызова определенной функции API-интерфейса. Сам автор использовал это средство в нескольких своих приложениях (например, в приложении DB-Library и в версиях ODBC его редактора Sequin SQL Editor для приложений Windows) и даже прочитал на конференции разработчиков доклад о том, как он создал свою библиотеку и как организовано ее внутреннее функционирование. Жизнь была так прекрасна!

После этого появились 32-битовые версии Windows. Проверив созданную им демонстрационную версию пакета WinMacro, которая получила название SuperRecorder, на первой версии Windows NT (Windows NT 3.1), автор обнаружил, что макрокоманды, зарегистрированные в одном процессе, могли воспроизводиться лишь в этом же процессе, а другие процессы их полностью не воспринимали. Одной из самых приятных особенностей пакета WinMacro в свое время было то, что он позволял зарегистрировать макрокоманду, скажем, в программе Notepad, затем перейти в программу Word и воспроизвести ее там. Это позволяло создавать глобальные, общесистемные

макрокоманды, которые имели одни и те же горячие клавиши и воспроизводились одинаково, независимо от приложения. Такими же функциональными возможностями обладала оригинальная вспомогательная программа Recorder. Но несмотря на все свои усилия, автор, проводивший свои эксперименты на первой полноценной версии Windows NT, не мог заставить свой пакет работать.

Дополнительно исследовав этот вопрос, автор обнаружил, почему макрокоманды WinMacro, зарегистрированные в одном процессе, нельзя было воспроизвести в другом. Средства изоляции процессов подсистемы Win32 исключали возможность предоставления другим процессам доступа к связному списку макрокоманд, зарегистрированных в одном процессе. При проектировании оригинального пакета WinMacro автор опирался на присущий операционной системе Windows 3.x совместный доступ к памяти процесса, который по определению уже не существовал в 32-битовых версиях Windows.

(Автор также обнаружил, что вспомогательная программа Recorder в 32-битовых версиях Windows ушла в небытие — ее невозможно было нигде найти. Скорее всего, причиной этого было применение для организации ее работы такого же принципа, как в пакете SuperRecorder, в связи с чем потребовалась бы полная переработка этой вспомогательной программы для ее эксплуатации в 32-битовых версиях Windows.)

Очевидно, что подход, применявшийся в 16-битовых версиях Windows, не был осуществим в Win32, поэтому примерно в 1994 году автор занялся доработкой пакета WinMacro для подсистемы Win32. Он назвал новую версию *WinMac32*.

Фундаментальная проблема, которую должен был решить автор в связи с разработкой версии WinMac32, состояла в том, чтобы обеспечить доступ другим процессам к связным спискам, которые могли быть распределены в текущем процессе. Автор быстро освоил работу со средствами совместного использования памяти подсистемы Win32 и спроектировал версию WinMac32 таким образом, чтобы в ней применялась совместно используемая память для хранения связного списка макрокоманд и событий клавиатуры/мыши. Эти средства в сочетании с обработчиками системных прерываний и средствами обеспечения доступа к библиотеке DLL позволили автору предусмотреть для приложений Win32 такие же функциональные возможности, которые были первоначально предусмотрены для приложения Win 3.x. Пакет WinMac32 поставлялся на компакт-дисках со многими другими книгами автора, а полный исходный код этого пакета (наряду со старым исходным кода Win 3.x) включен в состав компакт-диска, прилагаемого к данной книге. Процедуры `SendKeys` и `AppActivate` из применяемой автором библиотеки, в которых фактически не используются разработанные автором средства, поставлялись компанией Borland с продуктом Delphi, начиная с версии 4.0.

В ходе выполнения данного упражнения мы перейдем к приложению SuperRecorder, демонстрирующему возможности пакета WinMac32, и запишем макрокоманду, затем перейдем в редактор Notepad и воспроизведем данную макрокоманду. В ходе выполнения этих двух приложений мы будем контролировать именованный объект совместно используемой памяти WinMac32 с помощью инструментального средства WinObj.

1. Вначале вызовите на выполнение программу SuperRecorder. Исполняемый файл этой программы находится в подкаталоге `CH04\WinMac32` компакт-диска, прилагаемом к данной книге, и носит имя `srecorder.exe`. Для настройки конфигурации программы SuperRecorder до начала записи можно использовать меню Options.
2. Перейдите с помощью клавиши табуляции в поле `Scratch Area` (поле черновика), затем нажмите клавиши <Ctrl+Alt+Shift+F11>, чтобы начать запись.

3. Введите любые требуемые вам данные в поле черновика. При этом можно свободно использовать мышь для выбора части вводимого текста, затем для его вырезки, копирования и вставки.
4. Закончив ввод текста, нажмите клавиши <Ctrl+Alt+Shift+F11> второй раз, чтобы остановить запись. Появится приглашение к вводу произвольной комбинации, состоящей из нажатий клавиш/щелчков мышью, которая должна быть связана с этой макрокомандой. Нажмите клавишу обратной одинарной кавычки "`" (на клавиатурах большинства типов символ обратной одинарной кавычки вводится с помощью той же клавиши, что и символ тильды, но без перехода на верхний регистр), чтобы связать данную макрокоманду с клавишей обратной одинарной кавычки. Назовите макрокоманду `Shared Memory Test` и щелкните на кнопке OK.
5. Теперь запустите редактор Notepad и нажмите клавишу обратной одинарной кавычки. Вы должны увидеть, как воспроизводятся события клавиатуры и мыши, происходившие во время записи макрокоманды.
6. Запустите инструментальное средство WinObj (из комплекта Platform SDK) и щелкните на узле в дереве, обозначенном как `BaseNamedObjects`.
7. Выполняйте прокрутку списка именованных объектов вправо до тех пор, пока не обнаружите объект `WinMac32SharedData`. Дважды щелкните на этом объекте, чтобы вывести на экран окно с его свойствами. Вы увидите, что программа WinObj, кроме всего прочего, имеет информацию о том, что это — объект секции памяти (объект совместно используемой памяти). Он представляет собой именно тот объект совместно используемой памяти, который применяется в пакете WinMac32 для хранения его совместно используемых структур данных, а также служит средством, с помощью которого к результатам выполнения операций распределения памяти одним процессом (к записям макрокоманд) могут получить доступ другие процессы.
8. В ходе ознакомления с этими сведениями можно воспользоваться утилитой TList, чтобы узнать, какой метод внедрения DLL используется в пакете WinMac32 для подключения к действиям, осуществляемым в ходе того, как пользователь вводит символы с клавиатуры или работает мышью. Учитывая то, что приложение WinMac32 устанавливает обработчики системных прерываний (с помощью функции `SetWindowsHookEx` API-интерфейса) для перехвата событий от клавиатуры и мыши еще до того, как они поступят в процесс, для которого они предназначены, фактически эта программа вынуждает загружать себя в каждый процесс, в котором осуществляется ввод с клавиатуры или работа с мышью. Пока программы SuperRecorder и Notepad еще работают, вызовите на выполнение из командной строки утилиту TList, чтобы получить список идентификаторов всех процессов.
9. Определите идентификаторы, возвращенные для программ SuperRecorder и Notepad, и передайте каждый из них отдельно в утилиту TList, чтобы получить список модулей, загруженных в каждый процесс. Вы обнаружите, что в пространство процесса SuperRecorder загружена библиотека WinMac32. В этом нет ничего удивительного, поскольку программа SuperRecorder представляет собой демонстрационное приложение для этой библиотеки. Но вы также обнаружите, что библиотека WinMac32 загружена в пространство процесса Notepad. Известно, что в программе Notepad не загружается явно библиотека WinMac32, а также отсутствует ссылка на эту библиотеку через средства неявного импорта.

Так как же оказалось, что библиотека WinMac32 была загружена в пространство этого процесса? С помощью средств внедрения библиотеки DLL. Поскольку обработчики прерываний, установленные библиотекой WinMac32, являются общесистемными, а также поскольку код обратного вызова, предназначенный для обслуживания этих прерываний, находится в самой библиотеке WinMac32.DLL, то операционная система Windows загружает эту библиотеку DLL в пространство каждого процесса, в котором требуется выполнить код обработчика прерываний, иными словами, в пространство каждого процесса, в котором осуществляется ввод с клавиатуры, или используется мышь. В дебе впрочем, применив утилиту Tlist для проверки списка модулей, относящегося к инструментальному средству WinObj, с которым мы только что работали, можно обнаружить, что библиотека WinMac32.DLL загружена и в пространство этого процесса, поскольку в программе WinObj со времени запуска программы SuperRecorder осуществлялся ввод с клавиатуры или использовалась мышь.

10. Если теперь вы закроете приложение SuperRecorder, то увидите, что прекратились действия этой программы (больше не будет происходить внедрение библиотеки WinMac32.DLL в какие либо новые процессы), к тому же, библиотека WinMac32.DLL будет выгружена из пространств всех существующих процессов. Автор разработал данную программу именно так, чтобы пользователь мог легко отключить это средство поддержки макрокоманд, закрыв приложение, предназначенное для их регистрации. Хотя формально можно было бы предусмотреть возможность записи макрокоманд в одном приложении и воспроизведения их в любом другом, автор разработал данное демонстрационное приложение таким образом, что после его закрытия машина поддержки макрокоманд отключается, а связанная с ней библиотека выгружается из пространств всех процессов в системе. Если бы автор не предусмотрел такую возможность, то пользователю пришлось бы закрывать каждый процесс, в пространстве которого была внедрена библиотека WinMac32.DLL, чтобы полностью выгрузить ее из памяти. Средства внедрения библиотек DLL имеют свои преимущества и недостатки. Одним из недостатков является то, что внедренные библиотеки могут распространяться по всей системе, как лесной пожар, и автор думает, что пользователям вряд ли понравилось бы, если бы им пришлось принудительно удалять внедренную библиотеку из всех работающих процессов, чтобы ее отключить.

В пакете WinMac32 используются некоторые функции API-интерфейса Win32, которые заслуживают дополнительного описания. В нем с помощью функции SetWindowsHookEx устанавливаются четыре общесистемных обработчика прерываний (обработчик прерываний от клавиатуры, обработчик прерываний от мыши, обработчик прерываний записи в журнал и обработчик прерываний воспроизведения журнала). Кроме того, в этом пакете устанавливается обработчик сообщений для межпроцессного взаимодействия (с помощью функции CreateWindow) и область совместно используемой памяти, отображенная на системный файл подкачки с помощью функций MapViewOfFile и CreateFileMapping API-интерфейса.

Функции API-интерфейса ведения журнала позволяют надежно перехватывать и воспроизводить события клавиатуры и мыши в процессе. Эти функции обязательно должны взаимодействовать с подсистемами ввода Windows на самом низком уровне, и автору приходилось наблюдать, что при выполнении данных функций в системах со сбойными драйверами клавиатуры или мыши возникали синие экраны аварийного останова.

Автор время от времени только заново компилировал пакет WinMac32, но не обновлял его в течение многих лет, а после появления пакета Service Pack 3 для Windows NT 4 обнаружил, что работа WinMac32 стала несколько нестабильной. У автора не было времени, чтобы заняться исследованием того, что было изменено в этом сервисном пакете, к тому же он не очень стремился сделать это как можно быстрее, поскольку больше не распространял WinMac32 на коммерческой основе. Одним из преимуществ предоставления доступа к программному обеспечению по принципу бесплатного распространения является то, что разработчик сам решает, сколько времени тратить на его поддержку. У автора данной книги в течение многих лет действительно не хватало времени на поддержку какого-либо распространяемого им бесплатно программного обеспечения, и он сомневается, что эта ситуация изменится в ближайшем будущем.

Поскольку пакет WinMac32 относится к категории бесплатного программного обеспечения, автор не рекомендует использовать его в производственной среде. Изучите код этого приложения, что позволит многое узнать об операционной системе Windows и о функциях API-интерфейса Windows, но не возлагайте слишком большие надежды на само инструментальное средство.

Если вы захотите откомпилировать программу WinMac32 самостоятельно, то вам потребуется продукт Delphi 2.0 или последующей версии. Чтобы откомпилировать программу WinMacro (версию для Windows 3.x), вы должны иметь продукт Delphi 1.0. Для поддержки встроенного языка ассемблера вам потребуется ассемблер, совместимый с TASM 1.5 или последующей версией (по-видимому, подойдет MASM). А если удастся воспользоваться более современными версиями продукта Delphi, то можно применить лишь встроенный ассемблер — отдельного ассемблирования не требуется.

Следует также отметить, что объекты секции можно просматривать с помощью команды `!handle` отладчика WinDbg. Для этого выполните описанные ниже действия.

1. Введите команду `!handle` в приглашении к вводу команд отладчика WinDbg, чтобы получить список всех дескрипторов, локальных по отношению к процессу, в таблице дескрипторов процесса. Вызов на выполнение команды `!handle` без параметров приводит к тому, что создается список со всеми дескрипторами (с указанием типов их объектов) из таблицы дескрипторов процесса.
2. Найдите в этом списке дескрипторы, обозначенные как объекты секции. Еще раз вызовите на выполнение команду `!handle`, но на этот раз укажите номер дескриптора для каждого дескриптора объекта секции, который нужно вывести на экран. В команде `!handle` можно также передать маску опции, которая указывает, какая информация должна содержаться в дескрипторе. Например, предположим, что нужно получить информацию о дескрипторе с номером `a3c` — об объекте секции памяти. Это можно сделать примерно таким образом:

```
!handle a3c 4
```

В данном примере `a3c` — это номер дескриптора, а 4 — маска опции. Передавая 4 в команде `!handle`, вы требуете, чтобы было указано имя объекта, если оно имеется.

Совместно используемая память. Резюме

В операционной системе Windows предусмотрен богатый набор средств для распределения совместно используемой памяти и работы с ней. Совместно используемая память и отображаемые на память файлы широко применяются

в самой операционной системе; способы их применения являются не более сложными по сравнению с памятью других типов. Средства организации совместного доступа к данным из разных процессов или отображения файлов на виртуальную память являются довольно несложными и весьма мощными.

Совместно используемая память. Вопросы для самопроверки

1. Какой термин, применяемый при описании API-интерфейса Win32, соответствует термину "объект секции памяти"?
2. Подтвердите или опровергните следующее утверждение. Системный файл подкачки может использоваться в качестве реальной памяти для совместно используемой памяти.
3. Заполняет ли нулями страницы совместно используемой памяти операционная система Windows перед выполнением первой операции доступа к ним по такому же принципу, как заполняются нулями закрытые закрепленные страницы?
4. Какой объект привилегированного режима используется программой SQL Server и клиентской программой для координации доступа к области совместно используемой памяти, если для подключения клиентской программы к программе SQL Server применяется совместно используемая память?
5. Какая функция API-интерфейса Win32 используется для того, чтобы можно было немедленно записать страницы, модифицированные в отображаемом на память файле, на диск?
6. Какое значение необходимо задать при вызове функции `CreateFileMapping` в приложении для настройки совместно используемой памяти, чтобы в качестве реальной памяти для совместно используемой памяти применялся системный файл подкачки?
7. Подтвердите или опровергните следующее утверждение. При загрузке в ОС Windows исполняемой программы с жесткого диска в виртуальную память операционная система использует системный файл подкачки в качестве реальной памяти для кода и данных исполняемой программы.
8. Какая функция API-интерфейса Windows фактически предоставляет указатель на область совместно используемой памяти?
9. Подтвердите или опровергните следующее утверждение. Обмен данными с помощью совместно используемой памяти происходит медленнее, чем с помощью стека сетевых протоколов, но, несмотря на это, совместно используемая память — наилучшее инструментальное средство в той ситуации, когда требуется обеспечить надежный обмен данными.
10. Подтвердите или опровергните следующее утверждение. Хотя объект секции памяти применяется для реализации совместно используемой памяти, его не

следует считать равнозначным последней, поскольку он может также использоваться отдельным процессом для отображения файла на виртуальную память.

11. Подтвердите или опровергните следующее утверждение. Чтобы подготовить область совместно используемой памяти к применению, в приложении необходимо вначале вызвать функцию `AllocateUserPhysicalPages` API-интерфейса Win32, а затем функцию `MapUserPhysicalPages` для отображения представления совместно используемой памяти на виртуальное адресное пространство.
12. Какой объект привилегированного режима служит для реализации совместно используемой памяти и отображаемых на память файлов?
13. Какую команду WinDbg можно использовать для получения информации об объекте привилегированного режима, который соответствует области совместно используемой памяти?
14. Предположим, для подключения к экземпляру сервера SQL Server с именем `khen\ss2k_sp4` используется экземпляр программы Query Analyzer, которая вызвана на выполнение на том же компьютере, что и сервер. Какой четырехсимвольный префикс можно использовать вместе с именем сервера, чтобы вынудить программу Query Analyzer предпринять попытку подключиться с применением совместно используемой памяти Net-Library?

Основные принципы ввода-вывода

Автор всегда считал, что наилучший способ освоения технологии заключается в ее использовании для создания чего-то полезного. Любую применяемую им компьютерную технологию, будь то аппаратные средства, операционные системы, языки программирования, приложения и инструментальные средства, автор изучал, непосредственно осваивая ее на деле. Не может быть никакой замены практическому опыту: нет лучшего способа узнать, как что-либо работает, чем сделать это самому.

Автор еще не забыл, как первый раз он почувствовал восхищение, которое можно испытать, экспериментируя с компьютерной технологией и наблюдая за тем, как ее результаты воплощаются в реальность перед вашими глазами. Это был субботний вечер, и автор сел за терминал компьютера, чтобы впервые приступить к изучению языка программирования. Потратив примерно пять минут на чтение синтаксических диаграмм и всех тех необычайно скучных наставлений из руководства по языку программирования, автор решил просто попытаться самостоятельно создать любое приложение и посмотреть, что из этого получится. Это достижение должно было стать его самой первой программой.

Большинство людей, впервые приступая к изучению нового языка, стараются создать нечто вроде приложения, которое выводит на экран слова "Hello World", но автор не относится к этой категории. Для него самой привлекательной стороной компьютеризации была возможность значительно ускорить выполнение рутинной работы, а не просто формировать какие-то результаты. Люди сами могут многое создавать (атмосферные загрязнения, нескальные рисунки, картины, такие как "Мона Лиза", книги, такие как эта), и для этого им не нужны компьютеры. Для автора наиболее привлекательным аспектом применения компьютера была способность решать логические задачи быстрее и точнее, чем могли бы когда-либо сделать он или любые другие люди. Автор считает, что наиболее полезным нововведением в области компьютеризированных вычислений было понятие логического цикла — оно оказалось для компьютерных наук таким же важным открытием, каким стало изобретение колеса для всего человечества. Автор неоднократно наблюдал, как осуществимые задачи, внешне кажущиеся трудными, внезапно становились легко решаемыми и благодаря способности компьютера точно повторять снова и снова поручаемое ему задание до тех пор, пока не станет истинным некоторое логическое условие или не будет достигнута та или иная заключительная точка.

Итак, первой программой, которую было приложение, которое принимало на входе целое число и вырабатывало на выходе результаты разложения этого числа на множители. Невозможно описать, какое чувство переполняло меня, когда я впервые нажал на клавишу ввода, и увидел, как на экране магическим путем появились результаты выполнения моего приложения; это чувство до сих пор живо в воображении автора. Именно в этом заключается его стандартный подход к изучению новых технологий за последние два десятилетия, и он получает огромное удовольствие, начиная с того, как созданная им конструкция начинает действовать самостоятельно после нажатия на клавишу или щелчка мышью.

В этой главе с помощью именно такого практического подхода исследуются средства ввода-вывода операционной системы Windows. Вначале в ней изложены концептуальные описания и показана общая структура системы ввода-вывода; после этого приведены важные подробности и некоторые примеры кода. После изучения этой главы читатель будет иметь цельное представление о том, как работают средства ввода-вывода Windows и как они используются в приложениях, таких как SQL Server.

Основы ввода-вывода

Начнем исследование средств ввода-вывода Windows с самых основ. Вначале в этом разделе даны основные термины и определения, касающиеся ввода-вывода, после этого приведено подробное описание того, как функционируют средства ввода-вывода Windows с точки зрения их структуры.

Основы ввода-вывода. Основные термины и определения

- **Файловая система.** Общая структура, с помощью которой обеспечивается именование, хранение и систематизация файлов в операционной системе. Файловые системы состоят из устройств хранения данных, файлов, каталогов и метаданных, необходимых для поиска и доступа к ним. Файловая система — это не только логическое представление внешней памяти компьютера, но и часть операционной системы, обеспечивающая преобразование запросов приложений по выполнению операций с файлами в вызовы низкого уровня, содержащие аппаратные адреса, которые передаются в драйверы устройств, управляющие устройствами хранения данных, в том числе дисководами.

  Формат файловой системы определяет тот способ, с помощью которого организовано хранение данных в файлах, и поэтому от этого формата непосредственно зависят характеристики файловой системы. Например, формат, в котором не допускается применение файлов с размерами больше 2 Гбайт, ограничивает размеры файлов, которые могут поддерживаться файловой системой.

  В семействе Windows NT поддерживается несколько файловых систем (например, FAT16, FAT32, NTFS и т. д.), но NTFS — собственная файловая

система этого семейства. NTFS предоставляет больше средств и обладает более высокой производительностью по сравнению с любой другой файловой системой, которая поддерживается во всех версиях Windows. В NTFS индексы кластеров являются 64-битовыми, поэтому теоретически эта система позволяет адресовать до 16 Эбайт (1 эксабайт равен 1 миллиарду гигабайтов) дискового пространства. Но поскольку в операционной системе Windows размеры томов NTFS ограничиваются для того, чтобы они оставались адресуемыми с помощью 32-битовых индексов кластера, наибольший из томов NTFS может иметь объем 128 Тбайт (при использовании кластеров с размером 64 Кбайт).

- **Объект файла.** Объект привилегированного режима, применяемый для доступа к файлам и устройствам под управлением операционной системы Windows. Как и объекты привилегированного режима других типов, объекты файлов представляют собой ресурсы системы, которые могут совместно использоваться несколькими процессами; эти объекты могут быть именованными, они поддерживают синхронизацию (т.е. на них распространяется возможность перевода в сигнальное или несигнальное состояние) и могут быть защищены с помощью средств, действующих на основе понятия объекта.
- **Средства синхронного ввода-вывода.** Средства, которые вынуждают поток перейти в состояние ожидания до тех пор, пока не завершится выполняемая операция ввода-вывода.
- **Средства асинхронного ввода-вывода.** Средства, которые позволяют потоку, инициализировавшему операцию ввода-вывода, продолжить свое выполнение, не ожидая завершения этой операции. В потоке можно проверить состояние выполнения необходимой ему операции ввода-вывода с помощью различных способов, которые будут описаны ниже.

Основы ввода-вывода. Основные функции API-интерфейсов

Основные функции API-интерфейсов, применяющиеся для ввода-вывода, приведены в табл. 5.1.

Таблица 5.1. Основные функции API-интерфейсов, применяющиеся для ввода-вывода

Функция	Описание
`CreateFile`	Создать или открыть файл; эта функция может также использоваться для создания или открытия объектов других типов
`ReadFile`	Прочитать буфер из файла в память
`WriteFile`	Записать буфер из памяти на диск
`CloseHandle`	Закрыть дескриптор, связанный с объектом привилегированного режима (например, с файлом)

Основы ввода-вывода. Основные инструментальные средства

Основные инструментальные средства текущего контроля над вводом-выводом приведены в табл. 5.2.

Таблица 5.2. Основные инструментальные средства текущего контроля над вводом-выводом

	Количество операций чтения	Количество считанных байтов	Количество операций записи	Количество записанных байтов	Процентная доля затрат времени на чтение	Процентная доля затрат времени на запись	Средняя длина очереди к диску	Количество операций чтения с диска в секунду	Количество операций записи на диск в секунду
Perfmon	+	+	+	+	+	+	+	+	+
pmon	+	+	+	+					
TaskMgr	+	+	+	+					

Основные счетчики программы Perfmon

Программа Perfmon, безусловно, занимает ведущее положение среди всех программ, позволяющих получить информацию о состоянии ввода-вывода в операционной системе Windows. В этой программе предусмотрено слишком много счетчиков, поэтому в данной главе все эти счетчики не рассматриваются. Достаточно отметить, что с помощью программы Perfmon можно получить наиболее полный объем информации, касающейся ввода-вывода. В табл. 5.3 перечислены некоторые наиболее полезные счетчики, с помощью которых можно получить данные о производительности ввода-вывода.

Таблица 5.3. Основные счетчики программы Perfmon, относящиеся к вводу-выводу

Счетчик	Описание
`Physical Disk:Disk reads/sec`	Частота выполнения операций чтения на диске
`Physical Disk:Disk writes/sec`	Частота выполнения операций записи на диске
`Physical Disk:% Disk Read Time`	Процентная доля затрат времени, связанных с обслуживанием запросов на чтение, по отношению к общему затраченному времени
`Physical Disk:% Disk Write Time`	Процентная доля затрат времени, связанных с обслуживанием запросов на запись, по отношению к общему затраченному времени
`Physical Disk:Avg. Disk Queue Length`	Среднее число запросов на чтение и на запись, поставленных в очередь к указанному диску в течение интервала измерения
`Cache:Lazy Write Flushes/sec`	Частота, с которой поток с отложенной записью сбрасывает данные на диск
`Cache:Lazy Write Pages/sec`	Частота, с которой поток с отложенной записью записывает страницы на диск

Краткий обзор

В операционной системе Windows большинство операций ввода-вывода выполняется с помощью объектов файлов привилегированного режима. Отличительной особенностью объектов файлов является то, что их нельзя рассматривать как конструкции памяти в строгом смысле этого термина. Они представляют собой находящиеся в памяти объекты, которые обеспечивают доступ к самим ресурсам хранения данных, находящимся на запоминающем устройстве. В отличие от событий, семафоров и объектов привилегированного режима других типов, объекты файлов не управляют ресурсами, находящимися только в оперативной памяти — они предоставляют возможности взаимодействия с ресурсами, находящимися главным образом за пределами оперативной памяти.

Читатели, имеющие навыки программирования в операционной системе Windows, могли заметить, что в документации комплекта SDK рекомендуется использовать при открытии файла функцию `CreateFile`, а не `OpenFile`. (Функция `OpenFile` считается устаревшей и поддерживается лишь для обеспечения обратной совместимости с 16-битовыми приложениями.) С чем это связано? Разве не кажется немного противоречащей здравому смыслу попытка создать файл, когда требуется лишь его открыть? А что если этот файл уже существует? Не приведет ли такая попытка к его перезаписи? Нет, этого не произойдет, при том условии, что функция создания файла будет вызвана правильно. Причина использования для открытия файла функции `CreateFile`, а не `OpenFile` состоят в том, что для работы с файлом применяется объект привилегированного режима, поэтому должен быть создан именно этот объект, при этом не обязательно создается файл на диске. Создается объект привилегированного режима, позволяющий читать из файла или записывать в файл с помощью других функций API-интерфейса Win32. Функция `CreateFile` создает объект файла привилегированного режима и возвращает дескриптор этого объекта, локальный по отношению к процессу. Будет ли при вызове этой функции предпринята попытка создать физический файл на диске, полностью зависит от переданных ей параметров. Эта функция позволяет открыть существующий файл или создать новый, и все это зависит от того, как она вызвана.

Поскольку объект файла фактически является не ресурсом, которым он управляет, а лишь представлением этого ресурса в оперативной памяти, сам объект файла содержит только данные, относящиеся к конкретному дескриптору объекта, а данные, предназначенные для общего доступа, содержатся в самом файле. При открытии в потоке дескриптора файла операционная система Windows создает новый объект файла, содержащий отдельный набор атрибутов, которые описывают дескриптор файла. Поэтому на один и тот же файл может ссылаться несколько объектов файлов, но каждый из них содержит данные, относящиеся к его собственному дескриптору (например, текущее смещение в файле, с которого начнется следующая операция ввода-вывода).

Каждый объект файла является уникальным для данного процесса, но процесс может создать дубликат дескриптора файла в другом процессе с помощью вызова функции `DuplicateHandle` API-интерфейса. Иными словами, объекты файлов могут иметь имена, как и любые другие объекты привилегированного режима, но два процесса, которые открывают объект файла с одним и тем же именем, получают два разных объекта. В этом состоит отличие объектов файлов от объектов

привилегированного режима, других потоков в первую очередь потому, что сам ресурс, управляемый с помощью объекта файла, находится на диске, а не в памяти.

Даже несмотря на то, что дескриптор файла локален по отношению к процессу, сам файл таковым не является. Поэтому потоки должны синхронизировать свой доступ к этому файлу по аналогии с тем, как они синхронизируют доступ к любому другому совместно используемому ресурсу. Вряд ли можно допустить, чтобы один поток, например, выполнял запись в файл, в то время как другой предпринимает попытку чтения из этого файла. Поток, в котором следует выполнить запись в файл, должен либо открыть этот файл с исключительным правом доступа для записи, либо применить функцию `LockFile` API-интерфейса Win32 для блокировки других потоков, желающих получить доступ к этому файлу, в то время как происходит запись.

Средства синхронного ввода-вывода

В большинстве приведенных операций ввода-вывода является синхронными. Это значит, что вызвавший такую операцию поток ожидает ее завершения и только после этого продолжает свое функционирование. Средства синхронного ввода-вывода Windows основаны средствами ввода-вывода, которые предусмотрены в других операционных системах и других вариантах среды программирования, поэтому большинство разработчиков не испытывают затруднений при их использовании.

Приложения выполняют операции синхронного ввода-вывода, вызывая основные функции ввода-вывода подсистемы Win32 и ожидая их завершения. Если файл не был явно открыт для асинхронного ввода-вывода, то потоки ожидают завершения операций ввода-вывода, применяемых к этому файлу, и только после этого продолжают свою работу. Рассмотрим некоторые примеры кода.

Упражнение

Ниже приведен пример приложения, которое выводит конец текстового файла на терминал. Существует несколько версий подобных утилит, известных под общим названием `tail`. В данном упражнении приведен лишь очень простой вариант такой утилиты, позволяющий продемонстрировать, как осуществляется синхронный файловый ввод-вывод с помощью функции ввода-вывода подсистемы Win32. При использовании этого приложения можно указать, какой файл должен быть выведен на внешнее устройство, а также задать необязательный параметр смещения от конца файла, с которого должен начаться вывод содержимого файла. Если это смещение не задано, то рассматриваемая утилита выводит либо последний 1 Кбайт файла, либо последнюю его четверть, в зависимости от того, какой из этих размеров меньше.

Упражнение 5.1. Простая утилита, которая демонстрирует применение синхронного ввода-вывода

1. Загрузите и откомпилируйте приложение, приведенное в листинге 5.1, исходный код которого находится в подкаталоге `Ch05\tail` компакт-диска, прилагаемого к этой книге.

Листинг 5.1. Простая утилита tail

```
// tail.cpp. Утилита, позволяющая вывести конец файла на терминал
//

#include "stdafx.h"
#include "windows.h"
#include "stdlib.h"

int main(int argc, char* argv[])
{
  if (argc<2) {
    printf("Usage is:  tail filename [number of bytes]\n");
     return 1;
  }
  HANDLE hFile=CreateFile(argv[1],
                          GENERIC_READ,
                          FILE_SHARE_READ,
                          NULL,
                          OPEN_EXISTING,
                          FILE_ATTRIBUTE_NORMAL,
                          NULL);

  if (INVALID_HANDLE_VALUE==hFile) {
    printf("Unable to open file %s.  Last error=%d\n",
           argv[1],GetLastError());
     return 1;
  }

  DWORD dwFileOfs=1024;
if (argc>=3)
    dwFileOfs=atoi(argv[2]);

  DWORD dwFileSizeHigh;
  DWORD dwFileSizeLow;
  dwFileSizeLow=GetFileSize(hFile,&dwFileSizeHigh);

  if ((-1==dwFileSizeLow) &&
      (NO_ERROR!=(dwError=GetLastError()))) {
    printf("Unable to get the size of file %s.  Last error=%d\n",
           argv[1],GetLastError());
    return 1;
  }

  DWORDLONG dwlFileSize=(dwFileSizeHigh * MAXDWORD) +
                        dwFileSizeLow;

DWORDLONG dwlOfs = dwlFileSize / 4;
  if (dwlOfs<dwFileOfs)
    dwFileOfs = dwlOfs;

  DWORD dwNewPos=SetFilePointer(hFile,dwFileOfs * -1,0,FILE_END);

  char *pszTail = (char *)HeapAlloc(GetProcessHeap(),
                                    HEAP_ZERO_MEMORY,
                                     dwFileOfs+1);
  DWORD dwBytesRead;
  ReadFile(hFile,pszTail,dwFileOfs,&dwBytesRead,NULL);
```

```
  printf("%s\n",pszTail);

  HeapFree(GetProcessHeap(),0,pszTail);

  CloseHandle(hFile);

  return 0;
}
```

2. Вызовите это приложение на выполнение из среды разработки VC++ или из приглашения к вводу команд и передайте ему имя текстового файла, из которого нужно вывести на экран последний 1 Кбайт.
3. Вполне очевидно, что это приложение принимает в качестве входных данных имя файла и выводит последние *n* байтов. Такая возможность вывода удобна при обработке больших файлов, в которых интерес представляет только конец файла, поэтому желательно избежать необходимости выводить на внешнее устройство весь файл или искать конец этого файла с помощью операции поиска.
4. Работа приложения начинается с вызова функции `CreateFile`. В этом вызове функции `CreateFile` задано условие, что файл должен существовать (`OPEN_EXISTING`); в противном случае вызов этой функции окончится неудачей. Кроме того, поскольку в вызове `CreateFile` не передается флажок `FILE_FLAG_OVERLAPPED`, в приложении фактически указано, что выполнение операции ввода-вывода применительно к этому файлу должно происходить в синхронном режиме.
5. После открытия файла вычисляется смещение, с которого можно начать вывод содержимого файла. По умолчанию это значение составляет последний 1 Кбайт файла, но объем вывода может быть также указан в командной строке вызова этой утилиты.
6. Затем происходит вызов функции `SetFilePointer` API-интерфейса Win32, которая обеспечивает корректировку текущего смещения файла. Функция `SetFilePointer` перемещает указатель файла в позицию, определяемую относительно начала, текущей позиции или конца файла. Поскольку передается параметр `FILE_END`, тем самым дается указание, что смещение необходимо определить относительно конца файла. Для того чтобы сместить указатель файла в обратном направлении, необходимо задать отрицательное смещение в файле, поэтому предварительно вычисленное смещение в файле при передаче его в функцию `SetFilePointer` умножается на `-1`. С учетом того, что файл открыт в режиме синхронного ввода-вывода, главный поток процесса блокируется на то время, пока происходит перемещение указателя файла. Выполнение функции `SetFilePointer` в принципе может окончиться неудачей, но, поскольку в данной программе смещение было вычислено таким образом, что ошибки не должны возникать, автор для простоты исключил код контроля ошибок.
7. Затем выполняется операция распределения буфера из динамической области памяти процесса для хранения той части файла, которая будет считываться и выводиться на внешнее устройство. При распределении буфера используется параметр `HEAP_ZERO_MEMORY`. С учетом того, что при чтении данных из файла должно быть перекрыто все содержимое буфера, кроме последнего символа в буфере, фактически был бы более эффективным вариант, в котором значение 0 присваивается только последнему символу в буфере, но про-

граммисты часто экономят не машинное время, а собственные усилия, поэтому и в данном случае решено дать задание обнулить весь буфер диспетчеру динамической области памяти. Следует также отметить, что размер этого буфера превышает на один символ размер той области, которая считывается в конце рассматриваемого файла. Это позволяет гарантировать, что конец буфера будет содержать символ ASCII 0, давая возможность без опасений выводить содержимое этого буфера на терминал с помощью функции `printf`.

8. После этого вызывается функция `ReadFile` API-интерфейса Win32 для чтения конца файла в память. Выполнение функции `ReadFile` также может окончиться неудачей, в результате чего она возвратит меньше байтов, чем было затребовано. Однако для простоты автор исключил из этой программы код обработки данной ошибки и принял предположение, что из функции `ReadFile` будет получен результат, позволяющий правильно заполнить буфер.

9. Поскольку файл открывается для синхронного ввода-вывода, то в функцию `ReadFile` в качестве значения параметра структуры `OVERLAPPED` передается `NULL`. А при организации асинхронного ввода-вывода необходимо было бы передать указатель на действительную структуру. Поскольку в данном приложении применяется синхронный ввод-вывод, выполнение функции `ReadFile` будет блокироваться до окончания операции чтения.

10. Работа программы завершается выводом информации, только что считанной в буфер, на терминал, после чего буфер освобождается и дескриптор файла закрывается. После завершения работы с файлом для его закрытия вызывается функция `CloseHandle` API-интерфейса Win32. Выше приведен характерный пример использования синхронного ввода-вывода. Организация этого приложения весьма напоминает механизмы ввода-вывода, применяемые в других операционных системах и вариантах среды программирования — открывается файл, осуществляется чтение или запись в него, а после этого он закрывается; в этом нет ничего особо сложного.

Основы ввода-вывода. Резюме

В операционной системе Windows предусмотрен обширный набор средств, относящихся к вводу-выводу. В данном разделе рассматривается лишь часть из них, а остальные будут описаны в других разделах данной главы.

Большинство операций ввода-вывода в ОС Windows осуществляется с помощью дескриптора файла. Для открытия существующего файла используется функция `CreateFile`, а не `OpenFile`, поскольку при этом фактически создается объект файла привилегированного режима. От параметров, переданных в функцию `CreateFile`, зависит, будет система создавать новый файл на диске или открывать существующий.

Отличительной особенностью объекта файла привилегированного режима по сравнению с другими объектами привилегированного режима является то, что он фактически не содержит управляемого им ресурса. Управляемый ресурс находится на диске; это — сам файл. Если для внесения изменений в один и тот же файл требуется использовать многочисленные потоки, то, безусловно, необходимо синхронизировать их доступ к файлу, как и к любому другому совместно используемому ресурсу.

Синхронный ввод-вывод, поддерживаемый операционной системой Windows, представляет собой тот тип ввода-вывода, который применяется в приложениях наиболее часто. Синхронная операция ввода-вывода блокирует вызывающий поток до своего завершения. Ввод-вывод такого типа широко используется в приложениях и операционных системах; он представляет собой наиболее простой тип файлового ввода-вывода, поддерживаемого операционной системой Windows.

В ОС Windows предусмотрен стандартный набор функций API-интерфейса, позволяющих в максимальной степени упростить внедрение синхронного ввода-вывода в приложениях Windows. Для синхронного ввода-вывода файл открывается, как и для любого другого типа ввода-вывода — с помощью вызова функции `CreateFile`. Для чтения из файла применяется функция `ReadFile`, а для записи в файл — функция `WriteFile`. После завершения работы дескриптор файла закрывается с помощью вызова функции `CloseHandle` API-интерфейса.

Основы ввода-вывода. Вопросы для самопроверки

1. Какой функцией API-интерфейса Win32, `OpenFile` или `CreateFile`, должна быть вызвана при открытии существующего файла?
2. Подтвердите или опровергните следующее утверждение. Независимо от того, производятся ли инициализация операции синхронного или асинхронного ввода-вывода, в функцию `ReadFile` и `WriteFile` следует всегда передавать указатель на действительную структуру `OVERLAPPED`.
3. Подтвердите или опровергните следующее утверждение. После завершения работы с файлом, открытым в программе, его следует всегда закрывать с помощью функции `CloseFile` API-интерфейса Win32.
4. Какую функцию API-интерфейса Windows можно вызвать, чтобы переместить текущий указатель файла?
5. Если операционной системе MS-DOS было дано указание открыть существующий файл, но файл фактически не существует, какое значение дескриптора будет возвращено в этом случае?
6. Подтвердите или опровергните следующее утверждение. Если при открытии файла в приложении не передается параметр `FILE_FLAG_OVERLAPPED`, то тем самым операционной системе Windows дается указание, что операции ввода-вывода, применяемые к этому файлу, должны выполняться в синхронном режиме.
7. Каков наибольший размер тома NTFS, поддерживаемый операционной системой Windows?
8. Подтвердите или опровергните следующее утверждение. В потоке может быть вызвана функция `LockFile` API-интерфейса для блокировки части файла и предотвращения для других потоков возможности обращаться к этому файлу, пока сам поток вновь не обратится к файлу.

9. Может ли вызов функции `SetFilePointer` API-интерфейса Win32 завершиться неудачей?
10. В отличие от объектов привилегированного режима других типов, фактически ресурс, управляемый объектом файла, находится в другом месте. Где именно?

Асинхронный и небуферизованный ввод-вывод

В этом разделе продолжается описание средств ввода-вывода Windows и более подробно рассматриваются асинхронный и небуферизованный ввод-вывод. Если читатель еще не ознакомился с первой частью данной главы, рекомендуем ее быстро просмотреть, прежде чем приступать к изучению настоящего раздела.

Асинхронный и небуферизованный ввод-вывод. Основные термины и определения

- **Сектор.** Адресуемый с помощью аппаратных средств блок на носителе информации, таком как жесткий диск. Размер сектора на жестких дисках, применяемых на компьютерах с процессором x86, почти всегда равен 512 байтам. Это означает, что, если в операционной системе Windows необходимо выполнить запись в байт с номером 11 или 27, эти данные будут записаны в первом секторе диска, а запись в байт 1961 должна осуществляться в четвертом секторе диска. Чтобы узнать размер сектора диска, можно воспользоваться функцией `GetDiskFreeSpace` API-интерфейса Win32.
- **Кластер.** Адресуемый блок секторов, который используется во многих файловых системах. Кластер представляет собой наименьшую единицу хранения для файла. Кластер обычно превышает по размеру сектор, и его размер всегда составляет кратное от размера сектора. Кластеры применяются в файловых системах для организации более эффективного управления дисковым пространством, чем было бы возможно при использовании отдельных секторов. Кластер охватывает несколько секторов, поэтому позволяет разделить диск на более легко управляемые части. Недостатки кластеров состоят в том, что в случае применения кластеров больших размеров может неэффективно расходоваться дисковое пространство или возникать фрагментация, поскольку размеры файлов редко бывают равны кратным значениям от размера кластера. Чтобы определить размер кластера диска, можно воспользоваться функцией `GetDiskFreeSpace` API-интерфейса Win32.
- **Асинхронный ввод-вывод.** Тип ввода-вывода, который позволяет потоку, инициализирующему операцию ввода-вывода, продолжить свое выполнение, не ожидая завершения этой операции. В потоке проверка кода выполнения вызванной в нем операции ввода-вывода может осуществляться с помощью различных методов, которые будут вскоре описаны.

- **Совмещенный ввод-вывод.** Синоним асинхронного ввода-вывода.
- **Небуферизованный ввод-вывод.** Тип ввода-вывода, который позволяет операционной системе Windows открывать файл, не предусматривая промежуточной буферизации или кэширования. При использовании в сочетании с асинхронным вводом-выводом небуферизованный ввод-вывод позволяет добиться максимальной общей производительности выполнения асинхронных операций, поскольку эти операции не замедляются под влиянием синхронных операций диспетчера памяти. Несмотря на это, выполнение некоторых операций фактически происходит медленнее, поскольку не удается загрузить данные из кэша. В программе SQL Server небуферизованный ввод-вывод используется для устранения задержки между логическими операциями записи на диск и физической записью данных на диск.
- **Асинхронный вызов процедур.** Способ применения функции обратного вызова особого типа, которая выполняется в асинхронном режиме в контексте данного конкретного потока. Каждый поток имеет свою собственную очередь APC (Asynchronous Procedure Call – асинхронный вызов процедур). Существует два типа вызовов APC – привилегированного и непривилегированного режима. При постановке в очередь потока вызова APC привилегированного режима этот вызов APC выполняется после того, как данный поток будет в очередной раз запланирован на выполнение. А при постановке в очередь потока вызова APC непривилегированного режима этот вызов APC выполняется после того, как поток в очередной раз входит в состояние, в котором он может получать предупреждающие сообщения. Поток входит в состояние, в котором он может получать предупреждающие сообщения, после того как в нем вызывается функция ожидания, которая поддерживает получение предупреждающих сообщений. К примерам функций ожидания, позволяющих получать предупреждающие сообщения, относятся `WaitForSingleObjectEx`, `WaitForMultipleObjectsEx` и `SleepEx`.

  Функции асинхронного ввода-вывода `ReadFileEx` и `WriteFileEx` осуществляют постановку в очередь вызова APC непривилегированного режима после завершения асинхронной операции. В приложении можно также поставить в очередь его собственный вызов APC с помощью функции `QueueUserAPC` подсистемы Win32.

Асинхронный и небуферизованный ввод-вывод. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к асинхронному и небуферизованному вводу-выводу, приведены в табл. 5.4.

Краткий обзор

Как было указано выше, средства асинхронного ввода-вывода Windows позволяют в приложении инициализировать операцию ввода-вывода и продолжить выполнение приложения одновременно с тем, как осуществляется эта операция.

Безусловно, что такой подход позволяет повысить производительность приложения, поскольку при его использовании в приложении одновременно происходит несколько действий. По такому же принципу, как многопоточная организация позволяет повысить общую пропускную способность приложения, асинхронный ввод-вывод может помочь выполнить в приложении больше работы и добиться повышения быстродействия, поскольку при его использовании приложение может решать другие задачи, в то время как происходит операция ввода-вывода.

Таблица 5.4. Основные функции API-интерфейсов, относящиеся к асинхронному и небуферизованному вводу-выводу

Функция	Описание
`ReadFileEx`	Прочитать буфер асинхронно из файла в память
`WriteFileEx`	Записать буфер асинхронно из памяти на диск
`GetOverlappedResult`	Получить результат асинхронной (совмещенной) операции ввода-вывода; в качестве дополнительной возможности функция обеспечивает ожидание завершения этой операции
`HasOverlappedIoCompleted`	Возвратить логическое значение, которое указывает, завершена ли асинхронная операция
`GetDiskFreeSpace`	Возвратить системную информацию о диске, включая размеры его сектора и кластера
`WaitForSingleObjectEx`	Ожидать перехода объекта в сигнальное состояние, находясь в состоянии, которое при желании может быть определено как допускающее получение сообщений

При осуществлении асинхронного ввода-вывода, главным образом, используются такие же основные функции файлового ввода-вывода API-интерфейса Win32, как и при выполнении синхронного ввода-вывода; в них просто передаются другие параметры. (Хотя функции `ReadFileEx` и `WriteFileEx` применяются исключительно для операций асинхронного ввода-вывода, функции `ReadFile` и `WriteFile` могут использоваться для операций обоих типов.) Чтобы подготовить файл для обработки с помощью асинхронного ввода-вывода, необходимо передать в функцию `CreateFile` параметр `FILE_FLAG_OVERLAPPED`. В дальнейшем при вызове функции `ReadFile`/`ReadFileEx` или `WriteFile`/`WriteFileEx` передается указатель на структуру `OVERLAPPED`, которая задаст начальную позицию для текущей операции (а также используется операционной системой Windows для управления асинхронной операцией).

Проверка хода выполнения незавершенной асинхронной операции осуществляется с помощью функций `HasOverlappedIoCompleted` и `GetOverlappedResult` API-интерфейса Win32. Если в какой-то момент необходимо подождать завершения асинхронной операции перед тем, как приступить к дальнейшей работе, для этого можно воспользоваться одним из нескольких описанных ниже подходов.

- Вызов одной из функций ожидания Win32 (например, `WaitForSingleObject`) и передача ей либо необязательного объекта события, связанного со структурой `OVERLAPPED`, либо дескриптора самого объекта файла. Если приме-

няется событие, оно должно быть событием, отключаемым вручную, а не автоматически. Операционная система Windows после завершения асинхронной операции, которая была начата с помощью функции ReadFile или WriteFile (но не ReadFileEx или WriteFileEx), переводит событие, связанное со структурой OVERLAPPED, в сигнальное состояние. Кроме того, операционная система Windows переводит в сигнальное состояние дескриптор того файла, в котором закончилась асинхронная операция, но если в этом файле одновременно выполняется несколько асинхронных операций, то одна лишь эта информация о переходе дескриптора файла в сигнальное состояние не позволяет узнать, какая именно из этих операций была завершена, поэтому, если принято решение о том, что для контроля над выполнением асинхронной операции должен применяться сам объект файла, необходимо соблюдать осторожность.

- Передача функции GetOverlappedResult информации о том, что нужно ожидать завершения операции и только после этого выполнять возврат, с помощью указания значения TRUE в качестве параметра bWait этой функции. Если для контроля над завершением асинхронной операции требуется использовать функцию GetOverlappedResult, а в указанном файле одновременно выполняется несколько асинхронных операций, то необходимо создать объект события и присвоить его дескриптор элементу hEvent структуры OVERLAPPED, передаваемой в функции ReadFile/WriteFile и GetOverlappedResult. Дело в том, что если со структурой OVERLAPPED не будет связан объект события, а в качестве параметра bWait функции GetOverlappedResult указано значение TRUE, эта функция будет ожидать перехода в сигнальное состояние указанного объекта файла. Но в соответствии с тем, что было сказано в предыдущем пункте, этот метод является ненадежным, если в файле одновременно осуществляется несколько асинхронных операций, поэтому необходимо вместо этого предусмотреть событие, которое будет связано с рассматриваемой операцией с помощью структуры OVERLAPPED. Если структура OVERLAPPED содержит ссылку на объект события, то функция GetOverlappedResult ожидает перехода в сигнальное состояние этого объекта события, а не файла. А поскольку каждая асинхронная операция должна иметь свою собственную структуру OVERLAPPED, данный способ позволяет надежно определить состояние выполнения запроса ввода-вывода. Пример применения данного метода приведен в описании образца приложения fstring ниже в данной главе.
- Обеспечение контроля выполнения с помощью какого-то другого объекта. При этом используется функция ожидания, обеспечивающая получение предупреждающих сообщений, и предусматривается такая организация работы, при которой функция APC, переданная в функцию ReadFileEx или WriteFileEx, вынуждает выполнить возврат из функции ожидания после завершения асинхронной операции. Применение данного метода описано в примере приложения unicode_convert ниже в этой главе.

- Использование порта завершения ввода-вывода для управления процессом ожидания завершения асинхронного ввода-вывода. Подробные сведения об этом методе приведены в разделе "Порты завершения ввода-вывода" ниже в данной главе.

В программе может также возникать необходимость синхронизировать доступ к файлам, открытым для выполнения асинхронных операций записи. Если одновременно происходят многочисленные асинхронные операции записи, то, безусловно, необходимо обеспечить предотвращение искажения файла из-за возможных нарушений синхронизации потоков. Чтобы обеспечить выполнение в любой момент времени только одной асинхронной операции записи, можно использовать функцию `GetOverlappedResult` и функции ожидания Win32.

Независимо от того, применяются ли в приложении средства синхронного или асинхронного выполнения операций ввода-вывода, внутри самой операционной системы Windows большинство запросов ввода-вывода выполняется с использованием асинхронного ввода-вывода. Иными словами, сразу после того как подсистема ввода-вывода Windows инициализирует на устройстве запрос ввода-вывода, обычно драйвер устройства немедленно выполняет возврат. Но будет ли затем подсистема ввода-вывода немедленно возвращать управление в приложение или нет, зависит от того, был ли создан объект файла с параметром `FILE_FLAG_OVERLAPPED`, был ли передан действительный указатель структуры `OVERLAPPED` в функцию API-интерфейса, отвечающую за выполнение рассматриваемой операции ввода-вывода, и от многих других факторов.

Автор на этом хочет завершить более подробное изложение данной темы и подчеркнуть один факт, который может оказаться не очевидным на первый взгляд. Дело в том, что даже создание объекта файла с параметром `FILE_FLAG_OVERLAPPED` и передача действительного указателя структуры `OVERLAPPED` в функцию `ReadFile` или `WriteFile` могут не привести к инициализации асинхронной операции ввода-вывода. При использовании в программе функций `ReadFile` и `WriteFile` окончательное решение о том, следует выполнять заданную в них операцию синхронно или асинхронно, принимает сама операционная система Windows. Даже если в программе эти функции вызываются в расчете на то, что предусмотренная в них операция будет выполнена асинхронно, необходимо учесть в коде возможность, что этого не произойдет. Если функция `ReadFile` выполняет операцию синхронно, она возвращает значение `TRUE`. Если же операция выполняется асинхронно, то функция `ReadFile` возвращает `FALSE`, а функция `GetLastError` возвращает `ERROR_IO_PENDING`. Ниже перечислены примеры таких ситуаций, в которых операции `ReadFile`/`WriteFile`, инициализированные как асинхронные, выполняются синхронно.

- Обрабатываемый файл сжат с помощью средств сжатия NTFS. Именно поэтому не следует применять сжатие файлов, используемых в программе SQL Server, особенно файлов данных и файлов журналов.
- Выполнение затребованной операции приводит к увеличению размера файла (в качестве примера можно назвать выполнение операции `WriteFile` в конце файла, что приводит к увеличению размера файла).

- Операция может быть завершена немедленно, поскольку она приводит к обработке кэшированных данных. Большинство драйверов ввода-вывода разработано таким образом, что если операция может быть немедленно выполнена применительно к данным в кэше (она может сводиться к чтению данных из кэша или к записи данных в кэш), эта операция осуществляется синхронно.
- Операция применяется к буферизованному (кэшированному) файлу, но диспетчер кэша и диспетчер памяти чрезмерно загружены. Такая ситуация является более вероятной, если в приложении выполняется большое количество запросов ввода-вывода к данным, не находящимся в кэше. В программе SQL Server эта проблема полностью исключена, поскольку в ней осуществляется небуферизованный ввод-вывод в применяемых в ней файлах данных и файлах журналов. Дополнительные сведения о небуферизованном вводе-выводе приведены ниже в этой главе.

Для того чтобы операция ввода-вывода осуществлялась асинхронно, следует использовать функции `ReadFileEx` и `WriteFileEx`, а не функции `ReadFile` и `WriteFile`. Функции `ReadFileEx` и `WriteFileEx` всегда выполняются асинхронно, независимо от того, какие еще действия происходят в системе. В действительности, в отличие от `ReadFile` и `WriteFile`, функции `ReadFileEx` и `WriteFileEx` даже не принимают в качестве параметра счетчик, в котором можно было бы вернуть данные о количестве обработанных байтов, поскольку сами эти функции не предназначены для файлового ввода-вывода, а для получения данных о количестве обработанных байтов применяется функция APC, вызов которой функции `ReadFileEx` и `WriteFileEx` помещают в очередь после завершения операции ввода-вывода.

Упражнение

Рассмотрим некоторый код. В данном примере приложения рассматривается утилита, которая преобразует текст из файла в кодировке UNICODE в текст, который относится к другой кодовой странице (например, ANSI). Это инструментальное средство действует аналогично команде Save As редактора Notepad, поскольку оно позволяет записать текстовый файл UNICODE в другой кодировке символов.

Упражнение 5.2. Утилита, которая преобразует текстовый файл UNICODE с использованием асинхронного ввода-вывода

1. Загрузите пример приложения, приведенный в листинге 5.2, из подкаталога `CH05\unicode_convert` компакт-диска, прилагаемого к данной книге, в среду разработки Visual Studio (MSDEV).

Листинг 5.2. Утилита преобразования файлов UNICODE

```
// unicode_convert.cpp. Утилита, которая преобразует файлы UNICODE
//                      в файлы, заданные в другой кодовой странице
//
```

```
#include "stdafx.h"
#include "windows.h"
#include "stdlib.h"

#define INPUT_BUFFER_SIZE 0x1000

// Предусмотреть возможность четырехкратного увеличения объема
// во время преобразования
#define OUTPUT_BUFFER_SIZE (INPUT_BUFFER_SIZE * 4)

DWORD dwBytesWritten=0;
DWORD dwTotalBytesWritten=0;

VOID CALLBACK WriteCompleted(
  DWORD dwErrorCode,                // Код завершения
  DWORD dwNumberOfBytesTransfered,  // Количество переданных байтов
  LPOVERLAPPED lpOverlapped         // Буфер с информацией ввода-вывода
)
{
  printf("Async operation completed.  Transferred %d bytes.
      Error code=%d.\n",dwNumberOfBytesTransfered,dwErrorCode);
  dwBytesWritten=dwNumberOfBytesTransfered;
  dwTotalBytesWritten+=dwBytesWritten;
}

int main(int argc, char* argv[])
{
  if (argc<3) {
    printf("Usage is:  unicode_convert inputfilename
        outputfilename [codepage]\n");
     return 1;
  }

  DWORD dwCodePage=CP_ACP;
  if (4==argc)
    dwCodePage=atoi(argv[3]);

  HANDLE hInputFile=CreateFile(argv[1],
                               GENERIC_READ,
                               FILE_SHARE_READ,
                                NULL,
                               OPEN_EXISTING,
                                FILE_ATTRIBUTE_NORMAL,
                               NULL);

  if (INVALID_HANDLE_VALUE==hInputFile) {
    printf("Unable to open file %s.  Last error=%d\n",argv[1],
           GetLastError());
     return 1;
  }

  HANDLE hOutputFile=CreateFile(argv[2],
                                 GENERIC_WRITE,
                                0,
                                NULL,
                                 CREATE_ALWAYS,
```

```
                                    FILE_ATTRIBUTE_NORMAL |
                                     FILE_FLAG_OVERLAPPED,
                                    NULL);

  if (INVALID_HANDLE_VALUE==hOutputFile) {
    printf("Unable to open file %s.  Last error=%d\n",argv[2],
           GetLastError());
     return 1;
  }

  wchar_t *pwszBuffer = (wchar_t *)HeapAlloc(GetProcessHeap(),

HEAP_ZERO_MEMORY,

    INPUT_BUFFER_SIZE * sizeof(wchar_t));
    char *pszBuffer = (char *)HeapAlloc(GetProcessHeap(),
    HEAP_ZERO_MEMORY,
    OUTPUT_BUFFER_SIZE);

    OVERLAPPED olIO;

    ZeroMemory(&olIO,sizeof(olIO));

    DWORD dwBytesRead;
    DWORD dwTotalBytesRead=0;

    while ((ReadFile(hInputFile,
                             pwszBuffer,
                             INPUT_BUFFER_SIZE * sizeof(wchar_t),
                             &dwBytesRead,NULL))
                              && (dwBytesRead)) {

    if (dwTotalBytesRead) {

    WaitForSingleObjectEx(GetCurrentProcess(),INFINITE,true);
      if ((MAXDWORD - olIO.Offset) < dwBytesWritten) {
        olIO.OffsetHigh++;
        olIO.Offset=(MAXDWORD - olIO.Offset);
      }
      else olIO.Offset+=dwBytesWritten;
    }

    DWORD dwBytesConverted=
      WideCharToMultiByte(dwCodePage,
                             0,
                             pwszBuffer,
                             dwBytesRead / sizeof(wchar_t),
                             pszBuffer,
                              OUTPUT_BUFFER_SIZE,
                             NULL,
                             NULL);

    if (!dwBytesConverted) {
      printf("Error converting file %s near offset %d.  Last
          error=%d\n",argv[1],dwTotalBytesRead,GetLastError());
      return 1;
```

```
    }
WriteFileEx(hOutputFile,
                    pszBuffer,
                    dwBytesConverted,
                    &olIO,&WriteCompleted);

    dwTotalBytesRead+=dwBytesRead;

  }

  WaitForSingleObjectEx(GetCurrentProcess(),INFINITE,true);

  printf("Converted %s to %s using code page %s.  Read %d bytes,
      Wrote %d bytes\n", argv[1], argv[2], argv[3],
      dwTotalBytesRead, dwTotalBytesWritten);

  HeapFree(GetProcessHeap(),0,pwszBuffer);
  HeapFree(GetProcessHeap(),0,pszBuffer);

  CloseHandle(hInputFile);
  CloseHandle(hOutputFile);

  return 0;
}
```

2. Вызовите эту утилиту на выполнение из среды MSDEV и передайте ей в качестве первого параметра имя файла UNICODE, а качестве второго параметра — намеченное имя выходного файла, в который поступят полученные результаты. (Нажмите клавиши <Alt+F7> в среде Visual Studio 6.0 и выберите вкладку Debug, чтобы задать параметры командной строки.) Если у вас нет под руками готового файла UNICODE, такой файл с именем "UNICODE.TXT" можно взять с компакт-диска.
3. В качестве третьего параметра этой утилиты можно указать необязательный номер кодовой страницы (список широко применяемых кодовых страниц и соответствующих им целочисленных значений приведен в табл. 5.5). Если третий параметр не указан, по умолчанию используется кодовая страница ANSI.

Таблица 5.5. Широко применяемые кодовые страницы и обозначающие их константы API-интерфейса Win32

Кодовая страница	Целочисленное значение параметра функции API-интерфейса Win32
ANSI	0
OEM	1
MAC	2
Кодовая страница текущего потока	3
Symbol	42
UTF-7	65000
UTF-8	65001

4. Работа приложения `unicode_convert` начинается с открытия обоих файлов. В нем входной файл открывается в синхронном режиме. Поскольку преобразуется содержимое входного файла, невозможно что-либо сделать до тех пор, пока не будет выполнен каждый запрос ввода-вывода к этому файлу. Именно поэтому нет смысла предпринимать попытки асинхронного чтения данного файла.
5. Тем не менее выходной файл открывается в асинхронном режиме. Такое решение принято на основании следующих соображений. Не ожидая окончания операции записи в выходной файл, прежде чем приступить к чтению из входного файла следующей порции данных для заполнения буфера, мы предоставляем приложению возможность читать входные и писать выходные данные одновременно. Как только будет заполнен данными из входного файла каждый новый буфер, мы записываем предыдущий буфер в выходной файл.
6. Для того чтобы в приложении Windows можно было осуществлять ввод или вывод асинхронно, не только необходимо открыть соответствующий файл с параметром `FILE_FLAG_OVERLAPPED`, но и передать указатель на структуру `OVERLAPPED` в вызов функции `ReadFile/ReadFileEx` или `WriteFile/WriteFileEx`, который служит для передачи данных между файлом на диске и памятью. А прежде чем передавать указатель на структуру `OVERLAPPED`, в приложении необходимо инициализировать эту структуру, задавая смещение в файле, с которого должно начинаться чтение или запись. Именно поэтому данная структура передается в функцию `ZeroMemory` перед входом в цикл обработки. По этой же причине значение смещения файла, хранящееся в структуре `OVERLAPPED`, наращивается после каждой операции записи. Значения элементов `Offset` и `OffsetHigh` структуры `OVERLAPPED` сообщают системе, с какого смещения в файле должна начаться асинхронная операция. Поскольку запись в файл осуществляется асинхронно, то невыполнение описанных выше действий привело бы к тому, что каждый преобразованный текстовый буфер записывался, начиная с одного и того же смещения в файле (с нулевого смещения), а это совсем не то, что нужно.
7. Сами операции по преобразованию входного текста из одной кодировки в другую выполняются с помощью функции `WideCharToMultiByte` подсистемы Win32. Входной файл обрабатывается в цикле, каждый считанный буфер преобразуется с помощью функции `WideCharToMultiByte`, затем вызывается функция `WriteFileEx` для записи каждого преобразованного буфера в выходной файл.

   В этой программе используется функция `WriteFileEx`, а не `WriteFile`, чтобы можно было гарантировать асинхронное выполнение операции записи. Как уже было указано выше, если выполнение операции `WriteFile` приводит к увеличению записываемого с ее помощью целевого файла, то эта функция выполняется синхронно. А поскольку в данном случае создается новый файл, складываются именно такие условия. Вызывая функцию `WriteFileEx`, а не `WriteFile`, мы форсируем асинхронное, а не синхронное выполнение этой операции.
8. В функцию `WriteFileEx` передается указатель на структуру `OVERLAPPED`, а также адрес функции APC. Операционная система Windows сигнализирует о завершении затребованной асинхронной операции, вызывая указанную функцию APC. В этом случае используется определенная в программе глобальная функция `WriteCompleted`.
9. Для того чтобы можно было обеспечить запись каждого преобразованного текстового буфера на диск, прежде чем произойдет изменение его содержимого

в результате еще одного вызова функции WideCharToMultiByte, используется функция WaitForSingleObjectEx для приостановки выполнения вызывающего потока до завершения начатой ранее асинхронной операции записи. При ее вызове функции WaitForSingleObjectEx передается дескриптор текущего процесса. Этот объект фактически не применяется функцией WriteFileEx и не становится сигнальным в результате асинхронной операции (в действительности он не станет сигнальным до выхода из процесса). Здесь этот объект используется в качестве своего рода фиктивного объекта, чтобы можно было с его помощью получать извещения о завершении операции записи, начатой функцией WriteFileEx. Для того чтобы можно было вызвать процедуру APC, переданную в функцию WriteFileEx, необходимо перейти в состояние, допускающее получение предупреждающих сообщений. Для этого вызывается одна из функций ожидания Win32, которая поддерживает возможность получения предупреждающих сообщений во время, установленное с их помощью, и передачи значения TRUE в качестве параметра bAlertable этой функции. Поэтому асинхронная операция записи, инициализированная с помощью вызова WriteFileEx, будет завершена и вызовет возврат из функции ожидания, поскольку указано, что эта функция допускает получение предупреждающих сообщений, даже несмотря на то, что в функции WaitForSingleObjectEx установлена продолжительность ожидания перехода объекта процесса в сигнальное состояние на неопределенно долгое время.

У читателя может возникнуть вопрос: почему нельзя просто ожидать перехода в сигнальное состояние самого объекта файла с помощью функции WaitForSingleObjectEx, вместо применения описанной выше конструкции, в которой предусмотрено ожидание перехода в сигнальное состояние того объекта (т.е. процесса), который не станет сигнальным, пока работает сам процесс? Причина состоит в том, что ожидание перехода в сигнальное состояние того объекта, применительно к которому выполняется асинхронная операция ввода-вывода, в режиме, допускающем предупреждающие сообщения, не позволяет вызвать на выполнение заданную функцию APC после завершения данной операции. Дело в том, что операционная система Windows просто переведет этот объект в сигнальное состояние и принудительно выполнит возврат из функции ожидания без выполнения указанной функции APC. Если же состояние ожидания в режиме, допускающем получение предупреждающих сообщений, будет инициализировано с использованием другого объекта, операционная система Windows прервет это ожидание с помощью предупреждающего сообщения после завершения асинхронной операции, что приведет к выполнению функции APC в контексте потока, который инициализировал эту операцию.

Еще один способ, позволяющий приостановить выполнение потока до завершения операции записи, состоит в вызове функции GetOverlappedResult API-интерфейса Win32 с передачей в качестве параметра bWait этой функции значения TRUE. Это приведет к тому, что вызывающий поток перейдет в состояние ожидания до тех пор, пока не завершится текущая операция асинхронного ввода-вывода в указанном файле (на который ссылается передаваемая в качестве параметра структура OVERLAPPED).

10. В этой программе используется счетчик dwTotalBytesRead, который не только позволяет подытожить общее количество считанных байтов, но и служит в качестве флажка, с помощью которого можно определить, когда происходит первая итерация цикла. Сразу же после первого вхождения в цикл значение

указанного счетчика равно 0, поскольку еще не достигнут оператор в конце цикла, предназначенный для увеличения значения этого счетчика. В первой итерации цикла не нужно переходить в состояние ожидания завершения операции записи по той причине, что эта операция еще не инициализирована. Если бы в первой итерации была вызвана функция `WaitForSingleObjectEx` и началось ожидание получения предупреждающего сообщения еще до того, как была инициализирована сама асинхронная операция, то фактически произошло бы зависание вызывающего потока.

Причина, по которой приостанавливается выполнение вызывающего потока до завершения инициализированной перед этим асинхронной операции, является двоякой: во-первых, необходимо предотвратить внесение изменений в буфер, записываемый с помощью функции `WriteFileEx` на диск, инициализируя еще один вызов функции `WideCharToMultiByte` до того, как закончится эта запись, и, во-вторых, необходимо исключить возможность инициализации еще одной асинхронной операции записи до того, как будет завершена текущая операция асинхронной записи. Именно это автор имел в виду, говоря о том, что если в приложении используются асинхронные операции записи, то в нем необходимо предусмотреть синхронизацию потоков. В данном случае организация приложения остается довольно несложной и просто предотвращается одновременное выполнение нескольких асинхронных операций записи в одном и том же объекте файла. В более сложных приложениях вполне может возникнуть такая ситуация, при которой будут происходить одновременно несколько асинхронных операций ввода-вывода в разных объектах файлов, поэтому, чтобы следить за завершением сразу всех этих операций, можно использовать одну из мультиобъектных функций ожидания (например, `WaitForMultipleObjectsEx`) или прибегнуть к использованию более сложного механизма ввода-вывода, такого как порт завершения ввода-вывода.

11. Итерации в цикле продолжаются до тех пор, пока не будет обработан весь входной файл. Выход из цикла происходит после того, как функция `ReadFile` возвращает `FALSE` или обнаруживается, что количество байтов, считанных из входного файла, равно 0. Заслуживает внимания последний вызов функции `WaitForSingleObjectEx`. Он позволяет полностью выполнить последний запрос на запись в файл, прежде чем будут выведены итоговые данные о работе приложения и закрыты файлы. В этом вызове также происходит блокировка до завершения асинхронной операции.
12. Более полного понимания работы этого приложения можно достичь с помощью его пошаговой реализации в отладчике Visual C++. Начните с поэтапного выполнения главного цикла обработки и функции APC. Обращайте особое внимание на счетчики `dwBytesRead` и `dwBytesWritten`; они служат наилучшими индикаторами того, насколько продвинулось выполнение операций `ReadFile` и `WriteFileEx` в любой конкретный момент времени.

Итак, в общих чертах именно в этом и состоит выполнение асинхронных операций ввода-вывода в операционной системе Windows. Объект файла создается с использованием параметра `FILE_FLAG_OVERLAPPED`, затем для инициализации асинхронной операции передается указатель на структуру `OVERLAPPED` в функцию `ReadFile`/`ReadFileEx` или `WriteFile`/`WriteFileEx`. После этого для синхронизации доступа к файлу и проверки хода выполнения асинхронной операции применяется либо функция `GetOverlappedResult`, либо одна из функций ожидания Win32.

Небуферизованный ввод-вывод

Как было указано выше, небуферизованный ввод-вывод позволяет обходить в приложении диспетчер кэша Windows и осуществлять чтение и запись в файл непосредственно, без промежуточного буфера или кэша. Такая организация работы позволяет повысить производительность, особенно при выполнении асинхронного ввода-вывода, поскольку при этом исключается возможность того, что из-за синхронной организации работы диспетчера кэша ввод-вывод станет узким местом. Но при использовании небуферизованного ввода-вывода выполнение некоторых операций может замедлиться, поскольку в них невозможно будет воспользоваться преимуществами чтения данных из кэша.

В отношении небуферизованного ввода-вывода следует также упомянуть, что он позволяет устранить описанную выше проблему, которая заключается в том, что в операционной системе Windows может быть принято решение выполнить запрос на асинхронный ввод-вывод синхронно из-за перегрузки диспетчера памяти или диспетчера кэша. А поскольку операции небуферизованного ввода-вывода обходят системный кэш, такая возможность полностью исключена.

Небуферизованный ввод-вывод широко применяется в программе SQL Server. Благодаря тому, что программа SQL Server обходит системный кэш (и обеспечивает управление своим собственным внутренним кэшем), она имеет больший контроль над тем, какие операции должны выполняться асинхронно или синхронно, и может обеспечивать лучшую целостность данных, поскольку не приходится сталкиваться с тем, что операции записи на диск, которые внешне кажутся завершившимися, фактически не выполняются на физическом носителе до конца, пока диспетчер кэша не примет решение их осуществить.

Чтобы открыть файл без буферизации, необходимо передать параметр `FILE_FLAG_NO_BUFFERING` в функцию `CreateFile`. Для открытия файла с параметром `FILE_FLAG_NO_BUFFERING` в каком-то потоке необходимо выполнить определенные описанные ниже требования.

1. Доступ к файлу должен начинаться со смещения в байтах, которое делится без остатка на размер сектора диска.
2. В операциях чтения и записи файлов должно быть указано количество байтов, которое делится без остатка на размер сектора диска. При условии, что применяемый по умолчанию размер сектора равен 512 байт, в приложении можно считывать и записывать буфера с размерами, равными 1024 и 8192 байт, но не 1025 или 10 000 байт.
3. Буфера, используемые для операций чтения и записи, должны быть выровнены по адресам памяти, которые делятся без остатка на размер сектора диска. Это означает, что `0x7FF01000` — допустимый начальный адрес буфера, а `0x7FF01001` — нет.

Удобный способ обеспечить соблюдение последнего требования — использовать функцию `VirtualAlloc` при распределении памяти, предназначенной для небуферизованного ввода-вывода. Как было указано в главе 4, функция `VirtualAlloc` распределяет память по границам, кратным размерам страниц системы. А поскольку

и размер страницы системы, и размер сектора диска выражаются как степени числа 2, распределение буфера с помощью функции `VirtualAlloc` гарантирует, что буфер будет выровнен по границам размера сектора.

Следующее упражнение является первым из целого ряда упражнений, приведенных в данной книге, в которых показан процесс создания примера приложения, позволяющего выполнить поиск в текстовом файле указанной строки. В каждом примере приложения используется другой тип файлового ввода-вывода Windows, или этот ввод-вывод осуществляется по другому принципу по сравнению с остальными приложениями. А поскольку различные типы ввода-вывода применяются для решения одной и той же задачи, читатель получает возможность сравнивать и сопоставлять типы средств ввода-вывода, предоставляемые в ОС Windows для приложений непривилегированного режима, и изучать преимущества и недостатки каждого из них по сравнению с другими. Мы начнем с простых приложений, затем будем постепенно увеличивать их сложность, переходя к следующему; это позволяет лучше понять уникальные особенности каждого нового примера приложения на основании того, что было изучено в предыдущих примерах.

Упражнение

В упражнении 5.3 рассматривается приложение, в котором используется небуферизованный ввод-вывод. В нем открывается каждый файл, соответствующий указанной маске файла, в режиме, который одновременно является и небуферизованным, и перекрывающимся (асинхронным), после чего осуществляется поиск в файле заданной строки. В данном приложении для обеспечения параллельного поиска в файле применяется несколько рабочих потоков, ведется подсчет итоговых значений количества обнаруженных совпадений, после чего выводятся все строки, содержащие подстроку, совпадающую с искомой.

Упражнение 5.3. Утилита поиска строк, в которой используется небуферизованный асинхронный ввод-вывод

1. Загрузите пример приложения `fstring` из подкаталога `CH05\fstring` компакт-диска, прилагаемого к данной книге, в среду разработки Visual C++. Откомпилируйте приложение и вызовите его на выполнение, указав в качестве параметров текстовый файл и искомую строку. Если у вас нет готового файла, в котором можно было бы выполнить поиск, воспользуйтесь файлом `INPUT.TXT` на компакт-диске. Он содержит несколько экземпляров строки "ABCDEF", которую можно использовать в качестве искомой.
2. Найдите в программе `fstring.cpp` вызов функции `CreateFile`, относящийся к входному файлу. Вы обнаружите, что в этом вызове передаются сразу два параметра, `FILE_FLAG_NO_BUFFERING` и `FILE_FLAG_OVERLAPPED`. Применение этих параметров приводит к тому, что обработка файла, если есть такая возможность, осуществляется без использования системного кэша и в асинхронном режиме.
3. Приложение `fstring` состоит из двух основных модулей исходного кода — `fstring.cpp` и `bufsrch.cpp`. Модуль `fstring.cpp` содержит функцию точки

входа программы и те части кода, в которых инициализируется поиск в каждом файле, соответствующем заданной маске файла. В модуле `bufsrch.cpp` реализован код, необходимый для поиска указанной строки в данном конкретном буфере, подсчета количества совпадений и вывода результатов поиска на терминал. С работой этого кода лучше всего ознакомиться, изучая сам этот код. Начнем с модуля `fstring.cpp` (листинг 5.3).

Листинг 5.3. Главный модуль исходного кода для утилиты `fstring` (`fstring.cpp`)

```
// fstring.cpp. Многопотоковая процедура поиска в файле, в которой
//              используется небуферизованный ввод-вывод

#include "stdafx.h"
#include "windows.h"
#include "stdlib.h"
#include "process.h"
#include "bufsrch.h"

#define IO_STREAMS_PER_PROCESSOR 6

// Процедура точки входа для рабочих потоков
unsigned __stdcall StartSearch(LPVOID lpParameter)
{

  // Привести параметр, передаваемый в функцию _beginthreadex, к типу
  // CBufSearch * и вызвать метод Search класса CBufSearch

  return ((CBufSearch*)lpParameter)->Search();

}

// Выполнить поиск в указанном файле заданной искомой подстроки
// с использованием небуферизованного, асинхронного ввода-вывода
DWORD SearchFile(DWORD dwClusterSize,
                         DWORD dwNumStreams,
                         LPCRITICAL_SECTION pcsOutput,
                         char *szPath,
                         char *szFileName,
                         char *szSearchStr)
{
  char szFullPathName[MAX_PATH+1];
  DWORD dwNumThreads;
  HANDLE hPrivHeap;
  HANDLE *hThreads;
  HANDLE *hEvents;

  strcpy(szFullPathName,szPath);
  strcat(szFullPathName,szFileName);

  // Открыть файл как для небуферизованного, так и для совмещенного
  // (асинхронного) ввода-вывода
HANDLE hFile=CreateFile(szFullPathName,
                            GENERIC_READ,FILE_SHARE_READ,
                            NULL,
                            OPEN_EXISTING,
```

```
                                FILE_ATTRIBUTE_NORMAL
                                | FILE_FLAG_OVERLAPPED
                                | FILE_FLAG_NO_BUFFERING
                                ,NULL);

    if (INVALID_HANDLE_VALUE==hFile) {
    printf("Error opening file.  Last error=%d\n",
           GetLastError());
     return 1;
    }

    DWORD dwFileSizeHigh;

    DWORD dwFileSizeLow=GetFileSize(hFile,&dwFileSizeHigh);

    DWORD dwlFileSize=(dwFileSizeHigh*MAXDWORD)+dwFileSizeLow;

    DWORD dwNumClusts=dwlFileSize / dwClusterSize;
    if (dwNumClusts<1) dwNumClusts=1;

    // Если размер файла меньше 4 Гбайт и количество затребованных
    // потоков (речь идет о потоках ввода-вывода) меньше количества
    // кластеров, установить количество потоков равным количеству
    // кластеров
    if ((dwlFileSize<0xFFFFFFFF) && (dwNumStreams>dwNumClusts))
    dwNumThreads=dwNumClusts;
    else
    dwNumThreads=dwNumStreams;

    // Создать закрытую динамическую область памяти, чтобы можно было
    // за один раз освободить память, распределенную во всех операциях
    hPrivHeap=HeapCreate(0,0,0);

    // Создать массивы потоков и синхронизационных событий
    hThreads=(HANDLE *)HeapAlloc(hPrivHeap,
                                 HEAP_ZERO_MEMORY,
                                 dwNumThreads*sizeof(HANDLE));
    if (NULL==hThreads) {
    printf("Error allocating worker thread array.  Aborting.\n");
     return 1;
    }
hEvents=(HANDLE *)HeapAlloc(hPrivHeap,
                            HEAP_ZERO_MEMORY,
                            dwNumThreads*sizeof(HANDLE));
    if (NULL==hEvents) {
    printf("Error allocating event array.  Aborting.\n");
    return 1;
    }

    // Создать рабочие потоки и для каждого потока создать
    // экземпляр CBufSearch
    CBufSearch *pbFirst=NULL;
    unsigned uThreadId;

    for (DWORD i=0; i<dwNumThreads; i++) {
```

```
hEvents[i]=CreateEvent(NULL,false,false,NULL);

pbFirst=new CBufSearch(pbFirst,
                       pcsOutput,
                       szFileName,
                       hFile,
                       dwClusterSize,
                       szSearchStr,
                       hEvents[i]);

hThreads[i]= (HANDLE)_beginthreadex(NULL,
                                    0,
                                   &StartSearch,
                                   pbFirst,
                                    0,
                                   &uThreadId);

if (!hThreads[i]) {
  printf("Error creating thread.  Aborting.\n");
  return -1;
}
}

// Ожидать поступления от всех потоков сигнала о том,
// что произошел их запуск
WaitForMultipleObjects(dwNumThreads,hEvents,true,INFINITE);

// Главный цикл - обрабатывать файл в цикле, читая из него
// фрагменты с размером dwClusterSize и запуская потоки
// в количестве dwNumThreads для параллельного поиска
// в этих фрагментах
DWORDLONG dwlFilePos=0;
do {
for (CBufSearch *pbCurrent=pbFirst;
   NULL!=pbCurrent;
  pbCurrent=pbCurrent->m_pbNext) {

    pbCurrent->m_OverlappedIO.Offset=
         (DWORD)(dwlFilePos / MAXDWORD);
    pbCurrent->m_OverlappedIO.Offset=
         (DWORD)(dwlFilePos % MAXDWORD);

   // Заполнить буфер чтения нулями, чтобы не встречались
   // совпадения с искомой подстрокой в конце частично
   // заполненного буфера (оставшиеся от ранее считанных
   // фрагментов)
    ZeroMemory(pbCurrent->m_szBuf,dwClusterSize+1);

    // Считать из файла данные в объеме полного буфера,
    // по возможности используя асинхронный ввод-вывод
    if (!ReadFile(hFile,pbCurrent->m_szBuf,
                        dwClusterSize,
                        &pbCurrent->m_dwBytesRead,
                       &pbCurrent->m_OverlappedIO)) {

      DWORD dwLastErr=GetLastError();
```

```
            if (ERROR_IO_PENDING!=dwLastErr)     {

               // Завершить выполнение главного цикла потока при
               // обнаружении любой ошибки, исключая ERROR_IO_PENDING,
               // но включая EOF
               pbCurrent->m_bTerminated=true;

               // Осуществить аварийное завершение, если ошибка не
               // относится к типу EOF
               if (ERROR_HANDLE_EOF!=dwLastErr) {
                 printf("Error reading file.  Last
                         error=%d",dwLastErr);
                 throw -1;
               }
             }
            else {
               // Имеет место асинхронная операция
               pbCurrent->m_bOverlapped=true;
             }
          }
else {
            // Функция ReadFile возвратила истинное значение; операция
            // является синхронной
            pbCurrent->m_bOverlapped=false;
          }

          // Передать сигнал рабочему потоку, чтобы он приступил к поиску
          SetEvent(pbCurrent->m_hMainEvent);

          dwlFilePos+=dwClusterSize;
      }

      // Ожидать завершения поиска в своих буферах всеми рабочими
      // потоками. Каждый рабочий поток переводит в сигнальное состояние
      // предоставленный ему объект события, когда становится готовым
      // к обработке следующего буфера
      WaitForMultipleObjects(dwNumThreads,hEvents,true,INFINITE);

   } while (dwlFilePos<dwlFileSize);

   // Получить общий итог и уничтожить объекты поиска
   DWORD dwFindCount=0;
   CBufSearch *pbNext;
   for (; NULL!=pbFirst; pbFirst=pbNext) {
     pbFirst->m_bTerminated=true;
     dwFindCount+=pbFirst->m_dwFindCount;
      pbNext=pbFirst->m_pbNext;
     delete pbFirst;
   }

   // Закрыть дескрипторы потока и события
   for (i=0; i<dwNumThreads; i++) {
      CloseHandle(hThreads[i]);
     CloseHandle(hEvents[i]);
   }
```

```
  CloseHandle(hFile);

  // Освободить память, полученную во всех предыдущих операциях
  // распределения памяти из динамической области памяти, уничтожив
  // созданную в программе закрытую динамическую область памяти
  HeapDestroy(hPrivHeap);

  // Возвратить данные о количестве найденных совпадений, относящиеся
  // к указанному файлу
  return dwFindCount;
}
// Выполнить поиск заданной подстроки в файлах, имена которых
// соответствуют указанной маске
bool SearchFiles(char *szFileMask, char *szSearchStr)
{

  char szPath[MAX_PATH+1];

  // Извлечь обозначение пути к файлу из указанной маски
  char *p=strrchr(szFileMask,'\\');
  if (p) {
    strncpy(szPath,szFileMask,(p-szFileMask)+1);
    szPath[(p-szFileMask)+1]='\0';
  }
  else
    // Если путь не указан, использовать текущий каталог
    GetCurrentDirectory(MAX_PATH,szPath);

  // В случае необходимости добавить заключительный символ обратной
  // косой черты
  if ('\\'!=szPath[strlen(szPath)-1])
    strcat(szPath,"\\");

  printf("Searching for %s in %s\n\n",szSearchStr,szFileMask);

  // Обработать в цикле все файлы, имена которых соответствуют
  // указанной маске, и выполнить поиск заданной подстроки
  WIN32_FIND_DATA fdFiles;
  HANDLE hFind=FindFirstFile(szFileMask,&fdFiles);

  if (INVALID_HANDLE_VALUE == hFind) {
    printf("No files match the specified mask\n");
    return false;
  }

  // Определить количество процессоров в текущей системе. Эта величина
  // будет использоваться для вычисления количества потоков
  // ввода-вывода, с помощью которых должен осуществляться поиск
  // в каждом файле
  SYSTEM_INFO si;
  GetSystemInfo(&si);

  // Определить размер кластера на диске. Эта величина всегда кратна
  // размеру сектора, поэтому хорошо подходит для использования
  // в небуферизованном вводе-выводе
  DWORD dwSectorsPerCluster;
```

```
  DWORD dwBytesPerSector;
  DWORD dwNumberOfFreeClusters;
  DWORD dwTotalNumberOfClusters;
  GetDiskFreeSpace(NULL,&dwSectorsPerCluster,&dwBytesPerSector,
              &dwNumberOfFreeClusters,&dwTotalNumberOfClusters);

  DWORD dwClusterSize=(dwSectorsPerCluster * dwBytesPerSector);

  CRITICAL_SECTION csOutput;
  InitializeCriticalSection(&csOutput);

  DWORD dwFindCount=0;
  do {
    dwFindCount+=SearchFile(dwClusterSize,
      si.dwNumberOfProcessors*IO_STREAMS_PER_PROCESSOR,
      &csOutput,
      szPath,fdFiles.cFileName,
      szSearchStr);
  } while (FindNextFile(hFind,&fdFiles));

  FindClose(hFind);

  DeleteCriticalSection(&csOutput);

  printf("\nTotal hits for %s in %s:\t%d\n",szSearchStr,
       szFileMask,dwFindCount);
  return true;
}

int main(int argc, char* argv[])
{
  if (argc<3) {
    printf("Usage is: fstring filemask searchstring\n");
    return 1;
  }

  try
  {
    return (!SearchFiles(argv[1], argv[2]));
  }
catch (...)
  {
    printf("Error reading file\n");
    return 1;
  }
}
```

4. Начнем с функции `main`. Она принимает переданные в нее параметры и вызывает глобальную функцию `SearchFiles`. Функция `SearchFiles` принимает в качестве параметров маску файла и искомую строку, после этого находит все файлы, соответствующие заданной маске, с помощью функций `FindFirstFile` и `FindNextFile` API-интерфейса Win32.
5. Функция `SearchFiles` создает критическую секцию, которая будет использоваться для синхронизации доступа к терминалу, после чего эта секция передается

в процедуру, отвечающую за поиск каждого файла, `SearchFile`. Ниже будет описано, с чем связана необходимость применения критической секции.

6. Кроме того, в функции `SearchFiles` определяется размер кластера для текущего диска с помощью вызова функции `GetDiskFreeSpace`. Значение размера кластера диска необходимо определить, поскольку данная программа предназначена для обработки файлов с использованием небуферизованного ввода-вывода, а операционная система Windows требует, чтобы запросы ввода-вывода к небуферизованным файлам были выровнены по границам секторов диска. Поскольку размер кластера диска всегда имеет значение, кратное размеру сектора диска, то установка размера буфера чтения, соответствующего размеру кластера, — это удобный способ выполнять требования, заложенные в ОС Windows.
7. В функции `SearchFiles` осуществляется выборка значения количества процессоров, установленных в системе, и это значение используется для указания затребованного количества потоков ввода-вывода при вызове в ней функции `SearchFile`. Количество потоков ввода-вывода определяет количество потоков процесса, применяемых для поиска файла. В программе используется значение, кратное количеству процессоров в системе, поскольку каждый поток проводит определенное время в состоянии ожидания завершения ввода-вывода, а в приложении желательно поддерживать максимально возможную загрузку процессора (процессоров).
8. В функции `SearchFiles` вызывается глобальная функция `SearchFile` для поиска в каждом файле заданной строки.
9. Функция `SearchFile` начинает свою работу с того, что открывает указанный файл с помощью функции `CreateFile` и передает в вызове этой функции параметры `FILE_FLAG_NO_BUFFERING` и `FILE_FLAG_OVERLAPPED`. После этого функция `SearchFile` определяет размер файла и проверяет, не превышает ли количество потоков ввода-вывода количества кластеров в файле, поскольку чтение входного файла осуществляется в виде фрагментов с размерами, равными размеру кластера диска.
10. Затем в функции `SearchFile` создается закрытая динамическая область памяти, которая будет использоваться для хранения двух массивов дескрипторов, необходимых для данной функции, — массива дескрипторов рабочих потоков и массива синхронизационных событий. Затем в данной функции указанная закрытая динамическая область памяти применяется для распределения этих двух массивов. Поскольку оба массива распределяются из закрытой динамической области памяти, в функции `SearchFile` можно легко освободить отведенную для них область памяти, просто уничтожив динамическую область памяти перед выходом из функции. Напомним, что такой метод рассматривался также в главе 4.
11. После этого в функции `SearchFile` начинается выполнение цикла, в котором для каждого потока ввода-вывода распределяется объект `CBufSearch` и для каждого экземпляра `CBufSearch` создается рабочий поток. Точкой входа для каждого рабочего потока является глобальная функция `StartSearch`, которая приводит пользовательский параметр, переданный в функции `_beginthreadex`, к типу указателя `CBufSearch` и использует полученный указатель для вызова метода `CBufSearch::Search`. Дополнительные сведения об объекте `CBufSearch` приведены ниже.

12. После создания объектов поиска и рабочих потоков в функции `SearchFile` происходит ожидание поступления от рабочих потоков сигналов о том, что они готовы приступить к обработке данных (передаваемых с помощью массива синхронизационных событий).
13. После того как рабочие потоки передадут сигналы о своей готовности, в функции `SearchFile` начинается выполнение цикла, в котором происходят итерации по объектам `CBufSearch` и асинхронное чтение по одному кластеру из файла для каждого из этих объектов. Как только функция `SearchFile` считает кластер для каждого объекта `CBufSearch`, она передает объекту сигнал о том, что можно приступить к обработке буфера. После постановки в очередь запроса на выполнение поиска для каждого из объектов `CBufSearch` в функции `SearchFile` происходит ожидание завершения всех этих запросов перед передачей в очередь следующих запросов. Указанные действия продолжаются до тех пор, пока не будет закончен поиск во всем файле.
14. Функция `SearchFile` проверяет значение, возвращаемое функцией `ReadFile`, чтобы можно было учесть вероятность того, что в операционной системе Windows будет принято решение выполнять операцию чтения синхронно, даже несмотря на то, что файл открыт с параметром `FILE_FLAG_OVERLAPPED`, и в функции `ReadFile` передана действительная структура `OVERLAPPED`. Как было указано выше, в некоторых ситуациях операционная система Windows обрабатывает асинхронный запрос ввода-вывода синхронно, поэтому в коде приложения необходимо учесть такую возможность. В данном случае для указания на то, была ли успешно инициализирована асинхронная операция ввода-вывода, присваивается определенное значение одной из переменных объекта `CBufSearch` (переменной `m_bOverlapped`). При использовании объекта `CBufSearch` необходимо знать, было ли чтение инициализировано с помощью асинхронной операции, чтобы можно было определить, следует ли вызывать функцию `GetOverlappedResult` для перехода к ожиданию завершения текущей операции перед попыткой выполнить поиск в буфере чтения.
15. Чтобы проверить работу этого приложения, установите точку останова на функции `SearchFile` в каждой строке, в которой присваивается значение `m_bOverlapped`, затем вызовите приложение `fstring` на выполнение под управлением отладчика Visual C++. Если вы передадите в качестве маски файла строку `INPUT.TXT`, то должны обнаружить, что чтение этого файла происходит асинхронно. После этого остановите отладчик, откройте программу Explorer, перейдите в окно со свойствами файла, относящееся к файлу `INPUT.TXT`, и отметьте его как сжатый файл, затем повторно запустите процедуру проверки. Вы должны обнаружить, что теперь чтение файла происходит в синхронном режиме. Как было указано выше, одним из надежных способов исключить в ОС Windows возможность выполнять асинхронно чтение или запись в файл состоит в том, чтобы сжать его с помощью средств сжатия файлов NTFS.
16. После того как операции чтения и поиска в файле будут полностью закончены, функция `SearchFile` закрывает входной файл, освобождает распределенные для него ресурсы и возвращает итоговые данные о количестве совпадений с искомой строкой в функцию `SearchFiles`.
17. Реальная работа по поиску в каждом файле осуществляется с помощью класса `CBufSearch`. Ниже приведено более подробное описание этого класса (листинг 5.4).

Листинг 5.4. Модуль исходного кода для класса CBufSearch (bufsrch.cpp)

```
// bufsrch.cpp. Вспомогательный класс, который используется для поиска
//              подстроки в буфере

#include "bufsrch.h"

// Конструктор
CBufSearch::CBufSearch(CBufSearch *pbNext, LPCRITICAL_SECTION
pcsOutput, char *szFileName, HANDLE hFile, DWORD dwClusterSize,
char *szSearchStr, HANDLE hSearchEvent)
{
  // Инициализировать структуру OVERLAPPED
  ZeroMemory(&m_OverlappedIO, sizeof(m_OverlappedIO));
  m_OverlappedIO.hEvent=CreateEvent(NULL,true,false,NULL);

  // Записать в кэш параметры конструктора для дальнейшего
  // использования
  m_pbNext=pbNext;
  m_szFileName=szFileName;
  m_hFile=hFile;
  m_pcsOutput=pcsOutput;
  m_szSearchStr=szSearchStr;
  m_dwClusterSize=dwClusterSize;
  m_hSearchEvent=hSearchEvent;

  // Создать объект события; главный поток будет переводить этот
  // объект в сигнальное состояние после того, как появится возможность
  // передать буфер на обработку одному из рабочих потоков
  m_hMainEvent=CreateEvent(NULL,false,false,NULL);

  // Распределить память для буфера чтения с помощью функции
  // VirtualAlloc, чтобы можно было обеспечить выравнивание буфера
  // по границе, соответствующей размеру страницы. Это позволяет также
  // обеспечить выравнивание буфера по границе, соответствующей размеру
  // сектора, поскольку оба эти размера выражаются числами, равными
  // степени числа 2. Для того чтобы можно было осуществлять
  // небуферизованный ввод-вывод, буфер чтения или записи должен быть
  // выровнен по границе, соответствующей четному кратному размера
  // сектора диска. Распределить объем памяти, превышающий на один байт
  // размер кластера (что приведет к резервированию и закреплению еще
  // одной, дополнительной страницы виртуальной памяти), чтобы можно
  // было не беспокоиться о том, что функция strstr выйдет за конец
  // данного буфера, не найдя нулевой завершающий символ
  m_szBuf=(char *)VirtualAlloc(NULL,m_dwClusterSize+1,
                               MEM_RESERVE | MEM_COMMIT,
                               PAGE_READWRITE);

  // Инициализировать все остальные переменные экземпляра
  m_bTerminated=false;
  m_bOverlapped=true;
  m_dwFindCount=0;

}

// Деструктор
CBufSearch::~CBufSearch()
```

```
{
  // Закрыть дескрипторы объектов событий, созданных в конструкторе
  CloseHandle(m_OverlappedIO.hEvent);
  CloseHandle(m_hMainEvent);

  // Отменить закрепление и освободить память, распределенную для
  // буфера чтения
  VirtualFree(m_szBuf,0,MEM_RELEASE);
}

// Найти начало строки по данным о смещении в буфере
char *CBufSearch::FindLineStart(char *szStartPos)
{
  char *szStart;
  for (szStart=szStartPos; ((szStart>m_szBuf) &&
      (cLINE_DELIM!=*(szStart-1))); szStart--);
  return szStart;
}

// Найти конец строки по данным о смещении в буфере; предполагается,
// что строка завершается нулевым символом
char *CBufSearch::FindLineEnd(char *szStartPos)
{
  return strchr(szStartPos,cLINE_DELIM);
}

// Провести поиск в буфере чтения, чтобы найти каждую строку,
// содержащую ранее заданную искомую подстроку
bool CBufSearch::Search()
{
  char *szBol;
  char *szEol;
  char *szStringPos;
  DWORD dwNumChars;
  char *szStartPos;
  bool bRes=false;
  char szFmt[32];
  DWORDLONG dwlFilePos;

  // Передать в главный поток сигнал о готовности к проведению
  // обработки
  SetEvent(m_hSearchEvent);

  // В главном потоке значение m_bTerminated устанавливается равным
  // FALSE при обнаружении признака EOF или возникновении ошибки чтения
  // файла
  while (!m_bTerminated) {

    // Ожидать поступления от главного потока сигнала о том, что можно
    // приступать к обработке буфера чтения
    WaitForSingleObject(m_hMainEvent,INFINITE);

    // Если во время пребывания потока в состоянии ожидания присвоено
    // значение TRUE переменной экземпляра, указывающей на завершение
    // работы, выйти из цикла
    if (m_bTerminated) break;
```

```
    // Поиск начинается с начала буфера чтения
    szStartPos=m_szBuf;

    // Данные о текущей позиции в файле (которые в дальнейшем
    // потребуются для обозначения того места, где была найдена искомая
    // подстрока) могут быть извлечены из структуры OVERLAPPED,
    // используемой в операции чтения
    dwlFilePos=(m_OverlappedIO.OffsetHigh*MAXDWORD)
    +m_OverlappedIO.Offset;

     // Если операция выполняется в совмещенном (асинхронном) режиме,
     // использовать для ожидания ее завершения функцию
     // GetOverlappedResult
    if (m_bOverlapped) {
if ((!GetOverlappedResult(m_hFile,&m_OverlappedIO,
      &m_dwBytesRead,true)) ||
    (!m_dwBytesRead)) {
    printf("Error getting pending IO.  Last error=
        %d\n",GetLastError());
    break;
    }
}

__try
{
    // Продолжать выполнение итераций цикла до тех пор, пока маркер
    // начала поиска не равен NULL, находится в пределах буфера чтения,
    // а функция strstr продолжает находить в буфере чтения искомую
    // подстроку
    while ((szStartPos) &&
        (szStartPos<(m_szBuf+m_dwBytesRead)-1) &&
        (NULL!=(szStringPos=
        strstr(szStartPos,m_szSearchStr)))) {

    // Если управление перешло в эту точку, то обнаружено совпадение
    // с искомой подстрокой
    m_dwFindCount++;

    // Вычислить позиции начала и конца строки, чтобы можно было
    // вывести ее на терминал
    szBol=FindLineStart(szStringPos);
    szEol=FindLineEnd(szStringPos);

    // Вычислить количество выводимых символов. Это значение
    // в дальнейшем будет использоваться для подготовки строки
    // формата printf
    if (szEol) {
      dwNumChars=szEol-szBol;
      if (szEol<(m_szBuf+m_dwBytesRead)-1)
          szStartPos=szEol+1;
      else szStartPos=NULL;
    }
    else {
      dwNumChars=MAXLINE_LEN;
      szStartPos=NULL;
    }
```

```
    EnterCriticalSection(m_pcsOutput);

#if(_DEBUG)
    printf("Thread %08d: Offset: %010I64d %s ",
    GetCurrentThreadId(),dwlFilePos+
    (szStringPos-m_szBuf),m_szFileName);
#else
         printf("Offset: %010I64d %s ",dwlFilePos+
            (szStringPos-m_szBuf),m_szFileName);
    #endif
         // Подготовить строку формата, которая ограничивает
         // вывод текущей строкой
        strcpy(szFmt,"%.");
        sprintf(szFmt+2,"%ds\n",dwNumChars);

         // Вывести текущую строку
         printf(szFmt,szBol);

        LeaveCriticalSection(m_pcsOutput);

        bRes=true;

      }
     }
    __except(EXCEPTION_EXECUTE_HANDLER)
     {
       // Уничтожить эту исключительную ситуацию; управление не должно
       // никогда передаваться в данную точку, если учесть, что в конце
       // буфера чтения гарантировано наличие нулевого символа
  #if(_DEBUG)

      // Предполагается, что возникла ошибка, связанная с нарушением
      // прав доступа, из-за выхода за конец буфера чтения; но
      // фактически может иметь место ошибка какого-то другого типа
       printf("Thread %08d reached end of buffer\n",
       GetCurrentThreadId());
  #endif
     }

    // Передать в главный поток сигнал о том, что обработка данного
    // буфера закончена
     SetEvent(m_hSearchEvent);
  }

  // Если выход из этого цикла произошел аварийно, следует обязательно
  // перевести объект события в сигнальное состояние, чтобы в главном
  // потоке не происходило бесконечное ожидание
  SetEvent(m_hSearchEvent);
  return bRes;
}
```

18. Работа конструктора `CBufSearch` начинается с обнуления структуры `OVERLAPPED` класса и с создания события, которое должно быть связано с этой структурой. Как было указано выше в данной главе, объект события требуется, если намечено для ожидания завершения асинхронной операции использовать функцию `GetOverlappedResult`, и имеет место ситуация, в ко-

торой в одном и том же файле одновременно выполняется несколько асинхронных операций. Хотя, с формальной точки зрения, создание объекта события является необязательным, функция `GetOverlappedResult` не будет надежно работать без этого объекта события, если в указанном файле одновременно выполняется несколько асинхронных операций. Характерным признаком возникновения связанной с этим проблемы является то, что функция `GetOverlappedResult` возвращает `FALSE`, а функция `GetLastError` — 0.

19. Затем в конструкторе `CBufSearch` осуществляется создание объекта события, который будет использоваться в главном потоке для передачи одному из рабочих потоков сигнала о том, что он может приступить к обработке буфера чтения. Синхронизация действий между главным потоком и рабочими потоками осуществляется с применением двух объектов события для каждого рабочего потока. Первый объект события (создаваемый конструктором `CBufSearch`) предназначен для передачи рабочему потоку сигнала о том, что можно приступать к обработке буфера чтения. Второй объект события (создаваемый главным потоком) используется для передачи главному потоку сигнала о том, что рабочий поток завершил обработку одного буфера и готов приступить к обработке другого.

20. После этого в конструкторе `CBufSearch` осуществляется распределение памяти для буфера чтения. В нем для этой цели применяется функция `VirtualAlloc`, чтобы можно было гарантировать выравнивание буфера по границе сектора в памяти; в этом состоит одно из обязательных требований, связанных с использованием средств небуферизованного ввода-вывода Windows. Как было указано выше, функция `VirtualAlloc` всегда распределяет память с учетом границ страниц, а размер страницы системы и размер сектора диска всегда выражаются в виде степеней числа 2, поэтому можно быть уверенным в том, что любой буфер, распределенный с помощью функции `VirtualAlloc`, будет выровнен должным образом для использования в операциях небуферизованного ввода-вывода.

21. Обратите внимание на тот факт, что область памяти, распределенная с помощью функции `VirtualAlloc`, превышает по объему на 1 байт размер кластера. Это сделано для того, чтобы можно было обеспечить наличие нулевого завершающего символа в конце буфера поиска. В этой программе буфер чтения всегда заполняется нулями перед очередной операцией чтения, поэтому буфер чтения всегда содержит в конце символ 0, даже после завершения операции чтения, поскольку этот буфер превышает по величине размер считываемой части файла, указанный в вызове функции `ReadFile`. Необходимо обеспечить, чтобы буфер всегда оканчивался нулевым символом, поскольку для поиска в буфере строк и символов, совпадающих с искомыми данными, используются такие функции библиотеки RTL языка C/C++, как `strstr` и `strchr`. В применяемой библиотеке RTL нет функций `memstr` или `strnstr`, поэтому для обозначения конца буфера поиска необходимо предусмотреть, чтобы он оканчивался нулевым символом. Еще одним способом работы в этой ситуации является распределение вслед за буфером чтения защищенной страницы или страницы с запретом доступа, после чего при попытке в функции `strstr` перейти в область памяти, выходящую за пределы буфера чтения, будет просто перехватываться исключительная ситуация. Пример практического применения одного из вариантов этого метода приведен в разделе "Ввод-вывод с использованием файлов, отображаемых на память" ниже в этой главе.

22. Фактические действия по осуществлению поиска в каждом буфере чтения выполняются с помощью метода `CBufSearch::Search`. Как только рабочий поток приступает к работе, он вызывает этот метод и не выходит из него до достижения конца входного файла.

23. Выполнение метода `CBufSearch::Search` начинается с передачи главному потоку сигнала о том, что он приступил к работе и готов выполнить поиск в буфере. Сразу после создания рабочих потоков главный поток передает массив объектов событий в функцию `WaitForMultipleObjects`, чтобы организовать ожидание запуска всех рабочих потоков и перехода к выполнению метода `CBufSearch::Search`, и только после этого в главном потоке начинается чтение входного файла. Как только все рабочие потоки передадут сигналы о своей готовности, в главном потоке начинается обработка файла.

24. Затем в методе `CBufSearch::Search` выполняется цикл, для управления которым применяется переменная `m_bTerminated`. В данном методе выполнение цикла продолжается до тех пор, пока не возникнет какая-либо катастрофическая ошибка, которая приведет к принудительному выходу из цикла, или в главном потоке указанной переменной будет присвоено значение `FALSE`, поскольку был достигнут конец входного файла.

25. После этого в методе `Search` происходит проверка для определения того, есть ли какая-либо незавершенная асинхронная операция ввода-вывода; для этого проверяется значение переменной `m_bOverlapped`. Если остается незавершенной одна из асинхронных операций ввода-вывода, в методе `Search` вызывается функция `GetOverlappedResult` для контроля над завершением данной операции. Это приводит к тому, что функция `GetOverlappedResult` ожидает перехода в сигнальное состояние объекта события, связанного со структурой `OVERLAPPED`. После того как происходит этот переход, продолжается выполнение программы.

26. После приобретения действительного буфера чтения метод `Search` выполняет его обработку в цикле, отыскивая каждое вхождение искомой строки. Найдя соответствие, этот метод выводит строку, в которой было обнаружено соответствие, и перемещается к следующей строке для продолжения поиска. Это означает, что в каждой строке входного файла регистрируется не больше одного совпадения, независимо от того, сколько раз искомая подстрока встречается в этой строке.

27. После того как метод `Search` обнаруживает совпадение и начинает подготовку к выводу строки файла на терминал, он входит в критическую секцию, первоначально созданную в процедуре `SearchFiles`, чтобы исключить возможность выполнения одновременной записи на терминал другими рабочими потоками. Необходимость в этом возникает в связи с тем, что для вывода на терминал применяются два вызова функции `printf` — в одном из них указаны имя файла и смещение в строке, по которому найдена подстрока, а другой служит для вывода самой строки с обнаруженным совпадением. Если бы не использовалась критическая секция, то другой поток мог бы передать выходную информацию на терминал в промежутке между этими двумя вызовами функции `printf`, в результате чего было бы трудно разобраться в том, что появляется на терминале. В данной программе два вызова функции `printf` применяются для того, чтобы можно было обойтись без копирования строки с найденным совпадением во вторичный буфер перед выводом ее на терминал. Вместо этого вычисляются начало и конец строки в самом буфере чтения,

затем используется строка формата функции printf для вывода фрагмента текста, начинающегося от начала строки с совпадением и продолжающегося на количество символов от начала и до конца данной строки. Это позволяет обойтись без предварительного копирования строки с совпадением во вторичный буфер, поскольку вывод осуществляется непосредственно из буфера чтения. Кроме того, как будет показано в других примерах приложений, приведенных ниже в данной главе, в этот код можно внести небольшое изменение, чтобы избежать необходимости выполнять два отдельных вызова функции printf и использовать критическую секцию. Но автор применил в настоящем приложении именно такой подход, чтобы показать, для чего обычно применяется критическая секция, — для предотвращения возникновения такой ситуации, в которой несколько потоков выполняют один и тот же блок кода одновременно.

28. После того как метод Search заканчивает обработку своего буфера чтения, он передает главному потоку сигнал о том, что он готов к обработке другого буфера, и ожидает получения от главного потока сигнала о готовности для обработки нового буфера чтения. Если достигнут конец входного файла, в то время как метод Search ожидает сигнала от главного потока, переменной m_bTerminated класса CBufSearch присваивается значение TRUE к тому времени, когда метод Search выходит из функции WaitForSingleObject, что приводит к немедленному выходу в данном методе из главного цикла. Это, в свою очередь, приводит к выходу из потока. Как было указано в главе 3, всегда следует стремиться к тому, чтобы поток мог завершить работу обычным образом, а не форсировать этот процесс, вызывая функцию TerminateThread или ExitThread.

Итак, выше описан пример приложения fstring от начала до конца. Вполне очевидно, что в программе можно успешно применять сочетание небуферизованного и асинхронного ввода-вывода для эффективного выполнения некоторых очень важных задач.

Читатель мог заметить, что приложение fstring не позволяет обнаруживать соответствие с искомой подстрокой тех подстрок в файле, которые пересекают границы буфера. Если предположить, что в используемой системе размер кластера равен, скажем, 4 Кбайт, то в приложении fstring не будет обнаружена искомая подстрока, которая пересекает границу 4 Кбайт. В этом состоит одно из ограничений алгоритмов поиска, предназначенных для обработки фрагментов текста в виде отдельных страниц. Это ограничение можно преодолеть с помощью средств, которые будут описаны в разделе "Файловый ввод-вывод с отображением на память" ниже в данной главе. Проект приложения fstring в том виде, в каком он описан в этом разделе, предназначен лишь для демонстрации способа использования асинхронного, небуферизованного ввода-вывода в многопотоковой программе для параллельной обработки файла; он не предназначен для применения в качестве инструментального средства текстового поиска общего назначения. Тем не менее инструментальные средства поиска, в которых поддерживается страничная организация данных, имеют очень широкую область применения. Например, программа постраничного поиска может успешно использоваться для обработки данных, если отдельные элементы данных не пересекают границы буфера (по такому принципу, например, организовано хранение данных в базах данных).

Приложение fstring имеет еще одну важную особенность, которая заслуживает внимания. Поскольку это приложение является многопотоковым, посту-

пающие от него сообщения о найденных совпадениях с искомой подстрокой могут оказаться не упорядоченными. Нет никакой гарантии, что совпадения, находящиеся ближе к началу файла, будут выведены на внешнее устройство перед теми, которые находятся позже. Для обработки входного файла (файлов) используются одновременно несколько потоков, поэтому невозможно предсказать, за какое время будет найдено данное конкретное соответствие, и информация о нем будет выведена на терминал. К тому же, последовательность вывода строк с искомыми подстроками вполне может изменяться от одного прогона приложения к другому. Данный недостаток можно устранить, записывая результаты поиска в память и сортируя их перед выводом на терминал, но для этого может потребоваться огромный объем виртуальной памяти, а сама работа приложения значительно замедлится. Еще один вариант состоит в том, чтобы изменения были внесены в алгоритм поиска. В этом варианте можно предусмотреть, чтобы не осуществлялся параллельный поиск в отдельных файлах, а вместо этого рабочие потоки распределялись по файлам, и поиск в каждом файле осуществлялся с помощью отдельного потока. Такой подход позволяет устранить указанный недостаток, но не дает возможность полностью использовать ресурсы системы (особенно на многопроцессорном компьютере), если поиск проводится только в одном файле. При осуществлении поиска только в одном файле придется ждать, пока в одном потоке этот файл не будет полностью просмотрен в синхронном режиме, даже если этот файл очень велик. Лучшее решение состоит в применении фильтра SORT. В операционной системе Windows предусмотрено несколько фильтров, которые могут использоваться для фильтрации или обработки результатов выполнения вывода терминальных приложений и команд операционной системы. Фильтром SORT можно воспользоваться для упорядочения вывода приложения `fstring` таким образом, чтобы все строки с найденными совпадениями из каждого файла выводились на терминал в порядке их смещения от начала файла. Ниже приведена команда, позволяющая решить эту задачу.

```
fstring INPUT.TXT ABCDEF | SORT
```

Автор намеренно применил такой формат выходных строк, чтобы их можно было легко переупорядочивать с помощью фильтра SORT. Именно поэтому данные о смещении дополняются нулями, а текст в каждой строке, в которой обнаружено совпадение, имеет одинаковую длину, вплоть до той точки, где начинается имя файла.

Асинхронный и небуферизованный ввод-вывод. Резюме

Асинхронный ввод-вывод позволяет продолжить выполнение потока одновременно с тем, как происходит операция ввода-вывода. Для того чтобы инициализировать асинхронную операцию, необходимо создать объект файла с соответствующими битовыми флажками и передать в функцию подсистемы Win32, используемую для чтения или записи файла, действительную структуру OVERLAPPED. Но даже если в программе будут переданы операционной системе

Windows указания инициализировать асинхронное выполнение операции ввода-вывода, в ОС в некоторых обстоятельствах может быть принято решение выполнить эту операцию синхронно.

В потоке для ожидания выполнения текущего запроса на асинхронный ввод-вывод может быть вызвана функция `GetOverlappedResult`. Эта функция позволяет также контролировать переход в сигнальное состояние события, связанного со структурой `OVERLAPPED` или с самим объектом файла. Кроме того, такие функции подсистемы Win32, как `ReadFileEx` и `WriteFileEx`, могут ставить в очередь вызов функции APC для передачи потоку извещения о том, что асинхронная операция завершена. Еще одним механизмом, который может использоваться для передачи потоку извещения о завершения асинхронной операции, является порт завершения ввода-вывода.

Средства небуферизованного ввода-вывода позволяют потоку обойти системный кэш, поскольку с их помощью операции ввода-вывода выполняются непосредственно в файле. Для того чтобы эти средства можно было использовать в некотором потоке, необходимо выполнить целый ряд требований. Для получения возможности обрабатывать файл с помощью средств небуферизованного ввода-вывода объект файла должен быть создан с параметром `FILE_FLAG_NO_BUFFERING`. Операции доступа к этому файлу должны начинаться с четных кратных значений размера сектора диска. Операции чтения и записи в файле должны выполняться применительно к такому количеству байтов, которое также является четным кратным от размера сектора диска. Наконец, буфер, используемый в любой небуферизованной операции чтения или записи, должен быть выровнен по адресу памяти, который является четным кратным размера сектора диска. Один из надежных способов обеспечить выполнение этого требования состоит в применении функции `VirtualAlloc` для распределения этого буфера. Поскольку функция `VirtualAlloc` всегда распределяет память по границам страницы системы, а размер сектора диска и размер страницы системы всегда выражаются в виде степеней числа 2, то буфер, распределенный с помощью функции `VirtualAlloc`, всегда будет выровнен по границе размера сектора.

Использование небуферизованного ввода-вывода позволяет обеспечить наибольшую производительность в сочетании с асинхронным вводом-выводом. Небуферизованный ввод-вывод представляет собой также хороший способ вынудить операционную систему Windows действительно выполнять асинхронные операции ввода-вывода асинхронно, поскольку он позволяет обойти системный кэш, который может служить потенциальной причиной синхронной обработки асинхронных операций.

Небуферизованный и асинхронный ввод-вывод широко используется в программе SQL Server. Все выполняемые в программе SQL Server операции записи в файлы данных или в файлы журналов являются небуферизированными и асинхронными.

Асинхронный и небуферизованный ввод-вывод. Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Фактически сектор на жестком диске может быть больше, чем кластер.
2. Какой размер сектора чаще всего используется на компьютерах с процессором x86?
3. Подтвердите или опровергните следующее утверждение. Передача параметров `FILE_FLAG_OVERLAPPED` и `FILE_FLAG_NO_BUFFERING` в функцию `CreateFile` гарантирует, что операционная система Windows не будет обрабатывать запросы ввода-вывода к файлу синхронно.
4. Какую функцию распределения памяти можно использовать для обеспечения того, чтобы буфер чтения или записи для операции небуферизованного ввода-вывода выравнивался по границе сектора диска?
5. Какая функция API-интерфейса Win32 возвращает данные о размере сектора и кластера диска?
6. Подтвердите или опровергните следующее утверждение. Передача параметра `FILE_FLAG_NO_BUFFERING` в функцию `CreateFile` служит для операционной системы Windows указанием, что при обработке запросов ввода-вывода к указанному файлу нужно обходить системный кэш.
7. Применение функции `ReadFileEx` или `WriteFileEx` приводит к тому, что после завершения асинхронной операции ввода-вывода происходит постановка в очередь процедуры обратного вызова. К какому типу относится эта процедура?
8. Подтвердите или опровергните следующее утверждение. Одним из обстоятельств, в которых операционная система Windows выполняет запрос на асинхронный ввод-вывод синхронно, является чтение или запись файла, находящегося на сетевом диске.
9. В каких обстоятельствах дескриптор события, который является необязательным элементом структуры `OVERLAPPED`, становится обязательным, поскольку без него функция `GetOverlappedResult` не может функционировать должным образом?
10. Если в приложении выполняется цикл, в котором происходит чтение файла с использованием функции `ReadFileEx`, то нужно ли корректировать значения элементов структуры `OVERLAPPED`, которая передается в функцию `ReadFileEx`, от одного вызова функции к другому?
11. Подтвердите или опровергните следующее утверждение. Для определения количества байтов, обработанных в асинхронной операции ввода-вывода, которая была инициализирована функцией `WriteFileEx`, можно передать в функцию `WriteFileEx` по ссылке в качестве параметра `dwBytesTransferred` двойное слово `DWORD`.

12. Какая функция API-интерфейса Win32 может использоваться для постановки в очередь вызова APC?
13. Какие действия должны быть осуществлены в потоке для того, чтобы некоторая функция APC была вызвана на выполнение?
14. Какие два типа функций APC поддерживаются в операционной системе Windows?
15. Для чего, как правило, используется критическая секция?
16. Допустим, что со структурой `OVERLAPPED`, которая используется для асинхронного ввода-вывода, связан объект события; будет ли это событие переведено в сигнальное состояние после завершения операции, инициализированной с помощью функции `ReadFileEx`?
17. Объясните назначение элементов `Offset` и `OffsetHigh` структуры `OVERLAPPED` применительно к асинхронному вводу-выводу.
18. Подтвердите или опровергните следующее утверждение. В приложении можно вызвать функцию `HasOverlappedIoCompleted` интерфейса Win32 для определения того, завершено ли выполнение текущей операции асинхронного ввода-вывода.
19. Объясните, почему не следует инициировать ожидание в режиме приема предупреждений применительно к объекту файла, в котором выполняется текущая асинхронная операция ввода-вывода, инициализированная с помощью функции `WriteFileEx`.
20. Подтвердите или опровергните следующее утверждение. В программе SQL Server практически не используется небуферизованный ввод-вывод, поскольку он реализован на основе средств отложенной записи Windows в целях достижения максимальной производительности ввода-вывода.
21. Можете ли вы так задать параметры для функции `GetOverlappedResult`, чтобы она обеспечивала ожидание завершения текущей асинхронной операции и только после этого выполняла возврат?
22. Какое значение возвращает функция `GetLastError` после того, как функция `ReadFile` успешно инициализирует асинхронную операцию ввода-вывода?
23. Подтвердите или опровергните следующее утверждение. Сжатие файла с помощью средств сжатия NTFS исключает возможность выполнить его асинхронную обработку с помощью функций `ReadFile` и `WriteFile`.
24. Термин "совмещенный ввод-вывод" является синонимом другого термина; какого именно?
25. Подтвердите или опровергните следующее утверждение. Одним из требований, предъявляемых при инициализации небуферизованной операции ввода-вывода, применяемой к какому-либо файлу, является то, что доступ к файлу должен начинаться со смещения в байтах, которое делится без остатка на размер сектора диска.

Ввод-вывод со сборкой-разборкой

В данном разделе рассматриваются средства ввода-вывода со сборкой-разборкой операционной системы Windows, и приведен пример приложения, в котором они используются. Если читатель пропустил первую часть этой главы, то рекомендуем вначале с ней ознакомиться и только после этого продолжать изучение данного раздела.

Ввод-вывод со сборкой-разборкой. Основные термины и определения

- **Сектор.** Адресуемый с помощью аппаратных средств блок на таком носителе данных, как жесткий диск. Размер сектора (на жестких дисках, применяемых на компьютерах с процессором x86) почти всегда равен 512 байтам. Это означает, что если в операционной системе Windows необходимо выполнить запись в байт 11 или 27, эти данные будут записаны в первом секторе диска, а запись в байт 1961 должна осуществляться в четвертом секторе диска. Чтобы узнать размер сектора для диска, можно воспользоваться функцией `GetDiskFreeSpace` API-интерфейса Win32.
- **Кластер.** Адресуемый блок секторов, который используется во многих файловых системах. Кластер представляет собой наименьшую единицу хранения для файла. Кластер обычно превышает сектор по размеру и его размер всегда составляет кратное от размера сектора. Кластеры применяются в файловых системах для организации более эффективного управления дисковым пространством, чем было бы возможно при использовании отдельных секторов. Кластер охватывает несколько секторов, поэтому позволяет разделить диск на более легко управляемые части. Недостатки кластеров состоят в том, что в случае применения кластеров больших размеров может непроизводительно расходоваться дисковое пространство или возникать фрагментация, поскольку размеры файлов редко бывают равны кратным значениям от размера кластера. Чтобы определить размер кластера диска, можно воспользоваться функцией `GetDiskFreeSpace` API-интерфейса Win32.
- **Асинхронный ввод-вывод.** Тип ввода-вывода, который позволяет потоку, инициализирующему операцию ввода-вывода, продолжить свое выполнение, не ожидая завершения этой операции. В потоке проверка кода выполнения вызванной в нем операции ввода-вывода может осуществляться с помощью различных методов, которые будут вскоре описаны.
- **Небуферизованный ввод-вывод.** Тип ввода-вывода, который позволяет операционной системе Windows открывать файл, не предусматривая промежуточной буферизации или кэширования. При использовании в сочетании с асинхронным вводом-выводом небуферизованный ввод-вывод позволяет добиться максимальной общей производительности выполнения асинхронных операций, поскольку эти операции не замедляются под влиянием синхронных

операций диспетчера памяти. Несмотря на это, выполнение некоторых операций фактически происходит медленнее, поскольку не удается загрузить данные из кэша. В программе SQL Server небуферизованный ввод-вывод используется для устранения задержки между логическими операциями записи на диск и физической записью данных на диск.

- **Ввод-вывод со сборкой-разборкой**. Тип ввода-вывода, позволяющий использовать в приложении единственную операцию чтения или записи для перемещения данных между несколькими буферами виртуальной памяти и непрерывной областью файла. Прежде чем в операционной системе Windows была введена поддержка ввода-вывода со сборкой-разборкой, операции чтения или записи, применяемые к файлу, всегда были связаны с непрерывным буфером в памяти. А средства ввода-вывода со сборкой-разборкой позволяют составить такой буфер памяти из нескольких меньших частей, которые не обязательно должны образовывать одну непрерывную область.

Ввод-вывод со сборкой-разборкой. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к вводу-выводу со сборкой-разборкой, приведены в табл. 5.6.

Таблица 5.6. Основные функции API-интерфейсов, относящиеся к вводу-выводу со сборкой-разборкой

Функция	Описание
`ReadFileScatter`	Выполнить асинхронное чтение буфера из файла в массив буферов памяти
`WriteFileScatter`	Выполнить асинхронную запись буфера из массива буферов памяти в файл

Краткий обзор

В современных версиях Windows поддерживается особый тип файлового ввода-вывода, известный под названием *ввода-вывода со сборкой-разборкой*, который был впервые введен в сервисном пакете для Windows NT 4.0. Ввод-вывод со сборкой-разборкой можно использовать в приложении, вызывая функции `ReadFileScatter` и `WriteFileScatter`. Этот тип ввода-вывода действует по такому принципу: пользователь задает массив буферов либо для "разборки" данных из файла в память, либо для "сборки" данных из памяти и записи их в непрерывную область файла. Буфера, в которые записываются данные во время разборки или из которых они считываются во время сборки, не обязательно должны находиться в одной непрерывной области памяти, а область в файле, в которой выполняется чтение или запись, всегда является непрерывной. Назначение ввода-вывода со сборкой-разборкой состоит в том, чтобы обеспечить возможность передачи

данных между непрерывной областью файла и несколькими буферами, которые могут быть распределены произвольным образом в памяти, без использования промежуточного непрерывного буфера.

Для того чтобы в приложении можно было применять ввод-вывод со сборкой-разборкой, оно должно соответствовать описанным ниже требованиям.

- Файл должен быть открыт с параметрами `FILE_FLAG_OVERLAPPED` и `FILE_FLAG_NO_BUFFERING`.
- Буфера для сборки-разборки должны быть выровнены по границам размера сектора.
- Количество байтов, указанное в запросах на передачу, должно равняться четным кратным значениям от размера сектора диска назначения.

Как мог заметить читатель, эти требования объединяют в себе требования, предъявляемые к приложениям, в которых применяются небуферизованный и асинхронный ввод-вывод. Это связано с тем, что для ввода-вывода со сборкой-разборкой требуется небуферизованный ввод-вывод, а сам он по умолчанию является асинхронным. Как и при обычных запросах на выполнение асинхронного ввода-вывода, в операционной системе в некоторых обстоятельствах может быть принято решение о синхронном выполнении ввода-вывода со сборкой-разборкой (например, если целевой файл был сжат с помощью средств сжатия NTFS).

Средства ввода-вывода со сборкой-разборкой были с самого начала введены в операционную систему Windows именно для использования в программе SQL Server. Эти средства широко применяются в программе SQL Server при выполнении операций чтения и записи в файлы данных и файлы журналов. Указанные средства позволяют быстро заполнять страницы буферного пула данными, считанными с диска во время чтения из файлов данных или файлов журналов базы данных, а также быстро сбрасывать многочисленные страницы из буферного пула на диск во время записи данных.

Со средствами ввода-вывода со сборкой-разборкой проще всего ознакомиться, увидев их в действии. Рассмотрим некоторый код.

Упражнение

В этом упражнении приведен пример приложения, который представляет собой разновидность примера приложения `fstring`, описанного выше. Напомним, что программа `fstring` принимает в качестве параметров маску файла и искомую подстроку, а после этого выводит список всех строк, содержащих искомую подстроку в каждом из файлов, имя которого соответствует маске. В приложении, описанном в предыдущем разделе, для обработки файла в виде фрагментов с размерами, равными кластеру, использовался обычный асинхронный ввод-вывод. А в данном разделе рассматривается приложение, в котором для считывания сразу нескольких кластеров применяется ввод-вывод со сборкой-разборкой, после чего выполняется поиск в каждом из этих кластеров с помощью отдельного потока.

Упражнение 5.4. Утилита поиска в файле, в котором используется ввод-вывод со сборкой-разборкой

1. Загрузите пример приложения `fstring_scatter` из подкаталога `CH05\fstring_scatter` компакт-диска в среду разработки Visual Studio и откомпилируйте его.
2. Исходный код приложения `fstring_scatter` распределен в основном по двум файлам — `fstring_scatter.cpp` и `bufsrch.cpp`. Читатель может заметить, что программа `bufsrch.cpp` фактически идентична той, которая использовалась в приложении `fstring`.
3. По принципу функционирования приложение `fstring_search` весьма напоминает приложение `fstring`. В нем также выполняется обработка в цикле файлов, имена которых соответствуют заданной маске, и в каждом из этих файлов осуществляется поиск указанной подстроки. При обработке каждого файла осуществляется его асинхронное чтение, и каждый заполненный буфер чтения передается в рабочий поток для выполнения поиска. Различие между этими двумя приложениями состоит в том, что в них асинхронный ввод-вывод происходит по-разному. В случае приложения `fstring` содержимое буфера, передаваемого каждому рабочему потоку, считывается отдельно, а в приложении `fstring_scatter` содержимое всех буферов считывается за один раз с помощью одного вызова функции `ReadFileScattered`. Рассмотрим исходный код (листинг 5.5).

Листинг 5.5. Исходный код утилиты `bufsrch.cpp`

```
// bufsrch.cpp. Вспомогательный класс, который используется для поиска
//              подстроки в буфере

#include "bufsrch.h"

// Конструктор
CBufSearch::CBufSearch(CBufSearch *pbNext, char *szFileName,
HANDLE hFile, DWORD dwClusterSize, int iIndex, char *szBuf,
char *szSearchStr, HANDLE hSearchEvent, OVERLAPPED *pOverlappedIO)
{
  // Записать в кэш параметры конструктора для дальнейшего
  // использования
  m_pbNext=pbNext;
  m_szFileName=szFileName;
  m_hFile=hFile;
  m_szSearchStr=szSearchStr;
  m_dwClusterSize=dwClusterSize;
  m_hSearchEvent=hSearchEvent;
  m_pOverlappedIO=pOverlappedIO;
  m_iIndex=iIndex;

  // Создать объект события; главный поток будет переводить этот объект
  // в сигнальное состояние после того, как появится возможность
  // передать буфер на обработку одному из рабочих потоков
  m_hMainEvent=CreateEvent(NULL,false,false,NULL);

  m_szBuf=szBuf;
```

```
  // Инициализировать все остальные переменные экземпляра
  m_bTerminated=false;
  m_bOverlapped=true;
  m_dwFindCount=0;

}

// Деструктор
CBufSearch::~CBufSearch()
{
  // Закрыть дескрипторы объектов событий, созданных в конструкторе
  CloseHandle(m_hMainEvent);
}

// Найти начало строки по данным о смещении в буфере
char *CBufSearch::FindLineStart(char *szStartPos)
{
  char *szStart;
  for (szStart=szStartPos;
    ((szStart>m_szBuf) && (cLINE_DELIM!=*(szStart-1)));
      szStart--);
  return szStart;
}

// Найти конец строки по данным о смещении в буфере; предполагается,
// что строка завершается нулевым символом
char *CBufSearch::FindLineEnd(char *szStartPos)
{
  return strchr(szStartPos,cLINE_DELIM);
}

// Провести поиск в буфере чтения, чтобы найти каждую строку,
// содержащую ранее заданную искомую подстроку
bool CBufSearch::Search()
{
  char *szBol;
  char *szEol;
  char *szStringPos;
  DWORD dwNumChars;
  char *szStartPos;
  bool bRes=false;
  char szFmt[32];
  char szOffsetOutput[255];

  DWORDLONG dwlFilePos;

  // Передать в главный поток сигнал о готовности к проведению
  // обработки
  SetEvent(m_hSearchEvent);

  // В главном потоке значение m_bTerminated устанавливается равным
  // FALSE при обнаружении признака EOF или возникновении ошибки
  // чтения файла
  while (!m_bTerminated) {
    // Ожидать поступления от главного потока сигнала о том, что можно
    // приступать к обработке буфера чтения
    WaitForSingleObject(m_hMainEvent,INFINITE);

    // Если во время пребывания потока в состоянии ожидания присвоено
```

```
    // значение TRUE переменной экземпляра, указывающей на завершение
    // работы, выйти из цикла
    if (m_bTerminated) break;

    // Поиск начинается с начала буфера чтения
    szStartPos=m_szBuf;

    // Данные о текущей позиции в файле (которые в дальнейшем
    // потребуются для обозначения того места, где была найдена
    // искомая подстрока) могут быть извлечены из структуры OVERLAPPED,
    // используемой в операции чтения
    dwlFilePos=
          (m_pOverlappedIO->OffsetHigh*MAXDWORD)+
           m_pOverlappedIO->Offset+(m_iIndex*m_dwClusterSize);

    // Если операция выполняется в совмещенном (асинхронном) режиме,
    // использовать для ожидания ее завершения функцию
    // GetOverlappedResult
    if (m_bOverlapped) {
      if ((!GetOverlappedResult(m_hFile,m_pOverlappedIO,
        &m_dwBytesRead, true)) ||
        (!m_dwBytesRead)) {
        printf("Error getting pending IO.  Last error=%d\n",
          GetLastError());
          break;
        }
    }

    __try
    {
      // Продолжать выполнение итераций цикла до тех пор, пока маркер
      // начала поиска не равен NULL, находится в пределах буфера
      // чтения, а функция strstr продолжает находить в буфере чтения
      // искомую подстроку
      while ((szStartPos) &&
             (szStartPos<(m_szBuf+m_dwBytesRead)-1) &&
             (NULL!=(szStringPos=
                strstr(szStartPos,m_szSearchStr)))) {
// Если управление перешло в эту точку, то обнаружено совпадение
// с искомой подстрокой
        m_dwFindCount++;

        // Вычислить позиции начала и конца строки, чтобы можно было
        // вывести ее на терминал
        szBol=FindLineStart(szStringPos);
        szEol=FindLineEnd(szStringPos);

        // Вычислить количество выводимых символов. Это значение в
        // дальнейшем будет использоваться для подготовки строки
        // формата printf
        if (szEol) {
          dwNumChars=szEol-szBol;
          if (szEol<(m_szBuf+m_dwBytesRead)-1)
              szStartPos=szEol+1;
          else szStartPos=NULL;
        }
        else {
```

```
            dwNumChars=MAXLINE_LEN;
            szStartPos=NULL;
          }

      #if(_DEBUG)
          sprintf(szOffsetOutput,
          "Thread %08d: Offset: %010I64d %s ",
          GetCurrentThreadId(),dwlFilePos+
          (szStringPos-m_szBuf),m_szFileName);
      #else
          sprintf(szOffsetOutput,
          "Offset: %010I64d %s ",dwlFilePos+
          (szStringPos-m_szBuf),m_szFileName);
      #endif
          // Подготовить строку формата, которая ограничивает вывод
          // текущей строкой
          strcpy(szFmt,"%s %.");
          sprintf(szFmt+5,"%ds\n",dwNumChars);

          // Вывести текущую строку
          printf(szFmt,szOffsetOutput,szBol);

          bRes=true;

        }
      }
__except(EXCEPTION_EXECUTE_HANDLER)
      {
       // Уничтожить эту исключительную ситуацию; управление не должно
       // никогда передаваться в данную точку, если учесть, что в конце
       // буфера чтения гарантировано наличие нулевого символа
  #if(_DEBUG)

       // Предполагается, что возникла ошибка, связанная с нарушением
       // прав доступа, из-за выхода за конец буфера чтения; но
       // фактически может иметь место ошибка какого-то другого типа
        printf("Thread %08d reached end of buffer\n",
           GetCurrentThreadId());
  #endif
      }

    // Передать в главный поток сигнал о том, что обработка данного
    // буфера закончена
    SetEvent(m_hSearchEvent);
  }

  // Если выход из этого цикла произошел аварийно, следует обязательно
  // перевести объект события в сигнальное состояние, чтобы в главном
  // потоке не происходило бесконечное ожидание
  SetEvent(m_hSearchEvent);

  return bRes;
}
```

4. Сравнение версии программы `bufsrch.cpp`, представленное здесь, с той версией, которая применялась в упражнении 5.3, показывает, что в обеих про-

граммах класс CBufSearch в основном остается одинаковым. Основное различие между этими двумя вариантами заключается в том, что в версии, применяемой в настоящем упражнении, индекс буфера, передаваемый в класс CBufSearch для вычисления смещения в файле, соответствующего искомой подстроке, используется по-другому.

5. Как уже было сказано выше, приложение fstring_scatter во многих отношениях весьма напоминает приложение fstring. Но, как и можно было ожидать, одно из важных различий между ними заключается в том, что в приложении fstring_scatter вызывается функция ReadFileScatter для загрузки непрерывной области из каждого входного файла в набор буферов памяти, не находящихся в одной непрерывной области памяти. Изучите в листинге 5.6 код функции SearchFile, в которой вызывается функция ReadFileScatter. Вы обнаружите, что вызов этой функции находится за пределами цикла, в котором итерации проходят по объектам CBufSearch. Это связано с тем, что, в отличие от приложения fstring, в приложении fstring_scatter операцию ввода-вывода, предназначенную для передачи данных всем рабочим потокам, можно выполнить с помощью одного асинхронного вызова.

Листинг 5.6. Исходный код утилиты fstring_scatter

```
// fstring_scatter.cpp. Многопотоковая процедура поиска в файле,
//                      в которой используется ввод-вывод на основе
//                      сборки-разборки
//

#include "stdafx.h"
#include "windows.h"
#include "stdlib.h"
#include "process.h"
#include "bufsrch.h"

#define IO_STREAMS_PER_PROCESSOR 6

// Процедура точки входа для рабочих потоков
unsigned __stdcall StartSearch(LPVOID lpParameter)
{

  // Привести параметр, передаваемый в функцию _beginthreadex, к типу
  // CBufSearch * и вызвать метод Search класса CBufSearch

  return ((CBufSearch*)lpParameter)->Search();

}

// Выполнить поиск в указанном файле заданной искомой подстроки с
// использованием ввода-вывода на основе сборки-разборки
DWORD SearchFile(DWORD dwClusterSize,
                 DWORD dwNumStreams,
                 char *szPath,
                 char *szFileName,
                 char *szSearchStr)
{
  char szFullPathName[MAX_PATH+1];
  DWORD dwNumThreads;
```

```
  HANDLE hPrivHeap;
  HANDLE *hThreads;
  HANDLE *hEvents;
  FILE_SEGMENT_ELEMENT *pSegments;

  strcpy(szFullPathName,szPath);
  strcat(szFullPathName,szFileName);

  // Открыть файл как для небуферизованного, так и для совмещенного
  // (асинхронного) ввода-вывода
HANDLE hFile=CreateFile(szFullPathName,
                        GENERIC_READ,FILE_SHARE_READ,
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL
                        | FILE_FLAG_OVERLAPPED
                        | FILE_FLAG_NO_BUFFERING
                        ,NULL);

   if (INVALID_HANDLE_VALUE==hFile) {
   printf("Error opening file.  Last error=%d\n",
         GetLastError());
   return 1;
   }

   DWORD dwFileSizeHigh;

   DWORD dwFileSizeLow=GetFileSize(hFile,&dwFileSizeHigh);

   DWORD dwlFileSize=
      (dwFileSizeHigh*MAXDWORD)+dwFileSizeLow;

   DWORD dwNumClusts=dwlFileSize / dwClusterSize;
   if (dwNumClusts<1) dwNumClusts=1;

   // Если размер файла меньше 4 Гбайт и количество затребованных
   // потоков (речь идет о потоках ввода-вывода) меньше количества
   // кластеров, установить количество потоков равным количеству
   // кластеров
   if ((dwlFileSize<0xFFFFFFFF) &&
   (dwNumStreams>dwNumClusts))
   dwNumThreads=dwNumClusts;
   else
   dwNumThreads=dwNumStreams;

   // Создать закрытую динамическую область памяти, чтобы можно было
   // за один раз освободить память, распределенную во всех операциях
   hPrivHeap=HeapCreate(0,0,0);

   // Создать массивы потоков и синхронизационных событий
   hThreads=(HANDLE *)HeapAlloc(hPrivHeap,
                                HEAP_ZERO_MEMORY,
                                dwNumThreads*sizeof(HANDLE));
if (NULL==hThreads) {
    printf("Error allocating worker thread array.  Aborting.\n");
    return -1;
```

```
}

hEvents=(HANDLE *)HeapAlloc(hPrivHeap,
                            HEAP_ZERO_MEMORY,
                            dwNumThreads*sizeof(HANDLE));
if (NULL==hEvents) {
printf("Error allocating event array.  Aborting.\n");
 return -1;
}

// Создать массив указателей на элементы файловых сегментов,
// который будет использоваться в функции ReadFileScatter. Размер
// этого массива на единицу больше количества потоков, поскольку
// его последним элементом должен быть NULL
pSegments=(FILE_SEGMENT_ELEMENT *)HeapAlloc(hPrivHeap,
                                            HEAP_ZERO_MEMORY,
                                            (dwNumThreads+1)*
                                  sizeof(FILE_SEGMENT_ELEMENT));
if (NULL==pSegments) {
printf("Error allocating segment array.  Aborting.\n");
 return -1;
}

// Подготовить структуру OVERLAPPED, которая требуется для функции
// ReadFileScatter и которая будет использоваться всеми рабочими
// потоками
OVERLAPPED OverlappedIO;
ZeroMemory(&OverlappedIO,sizeof(OverlappedIO));
OverlappedIO.hEvent=CreateEvent(NULL,true,false,NULL);

// Создать рабочие потоки и для каждого потока создать
// экземпляр CBufSearch
CBufSearch *pbFirst=NULL;
unsigned uThreadId;

// Выполнять отсчет значений переменной цикла в обратном
// направлении, чтобы элементы связного списка экземпляров
// CBufSearch имели правильные порядковые значения индекса;
// эти порядковые значения используются каждым экземпляром
// CBufSearch для вычисления смещения переданного ему
// фрагмента в файле
for (int i=dwNumThreads-1; i>=0; i--) {

hEvents[i]=CreateEvent(NULL,false,false,NULL);
// Распределить память для буфера чтения с помощью функции
// VirtualAlloc, чтобы можно было обеспечить выравнивание буфера
// по границе, соответствующей размеру страницы. Это позволяет
// также обеспечить выравнивание буфера по границе, соответствующей
// размеру сектора, поскольку оба эти размера выражаются числами,
// равными степени числа 2. Для того чтобы можно было осуществлять
// ввод-вывод на основе сборки-разборки, буфер чтения или записи
// должен быть выровнен по границе, соответствующей четному
// кратному размера сектора диска. Распределить объем памяти,
// превышающий на один байт размер кластера (что приведет к
// резервированию и закреплению еще одной, дополнительной,
// страницы виртуальной памяти), чтобы можно было не беспокоиться
```

```
// о том, что функция strstr выйдет за конец данного буфера, не
// найдя нулевой завершающий символ
pSegments[i].Buffer=
  (PVOID64)VirtualAlloc(NULL,dwClusterSize+1,
                        MEM_RESERVE | MEM_COMMIT,
                        PAGE_READWRITE);

pbFirst=new CBufSearch(pbFirst,
                        szFileName,
                       hFile,
                       dwClusterSize,
                        i,
                       (char *)pSegments[i].Buffer,
                        szSearchStr,
                        hEvents[i],
                       &OverlappedIO);

hThreads[i]=
(HANDLE)_beginthreadex(NULL,
                        0,
                       &StartSearch,
                       pbFirst,
                        0,
                        &uThreadId);

if (!hThreads[i]) {
  printf("Error creating thread.  Aborting.\n");
  return -1;
}
}
// Ожидать поступления от всех потоков сигнала о том, что
// произошел их запуск
WaitForMultipleObjects(dwNumThreads,hEvents,true,INFINITE);

bool bTerminated=false;
bool bOverlapped;

// Главный цикл - обрабатывать файл в цикле, читая из него
// фрагменты с размером dwClusterSize * dwNumThreads. После
// заполнения каждого набора буферов разборки передавать
// рабочим потокам сигнал о том, что они могут выполнить
// в них поиск
DWORDLONG dwlFilePos=0;
do {

bOverlapped=true;

OverlappedIO.Offset=
     (DWORD)(dwlFilePos / MAXDWORD);
OverlappedIO.Offset=
     (DWORD)(dwlFilePos % MAXDWORD);

// Заполнить буфер чтения нулями, чтобы не встречались совпадения
// с искомой подстрокой в конце частично заполненного буфера
// (оставшиеся от ранее считанных фрагментов)
for (DWORD j=0; j<dwNumThreads; j++)
```

```
    ZeroMemory(pSegments[j].Buffer,dwClusterSize+1);

  // Заполнить буфера разборки, по возможности используя асинхронный
  // ввод-вывод
  if (!ReadFileScatter(hFile,pSegments,
                       dwClusterSize*dwNumThreads,
                       NULL,
                       &OverlappedIO)) {

    DWORD dwLastErr=GetLastError();
    if (ERROR_IO_PENDING!=dwLastErr)     {

       // Завершить выполнение главного цикла потока при обнаружении
       // любой ошибки, исключая ERROR_IO_PENDING, но включая EOF
      bTerminated=true;

      // Осуществить аварийное завершение, если ошибка не относится
      // к типу EOF
       if (ERROR_HANDLE_EOF!=dwLastErr) {
              printf("Error reading file.  Last error=%d",
              dwLastErr);
        return -1;
      }
     }
    else {
       // Имеет место асинхронная операция
      Overlapped=true;
     }
   }
  else {
    // Функция ReadFile возвратила истинное значение; операция
    // является синхронной
     bOverlapped=false;
   }

   for (CBufSearch *pbCurrent=pbFirst;
    NULL!=pbCurrent;
    pbCurrent=pbCurrent->m_pbNext) {

     pbCurrent->m_bTerminated=bTerminated;
     pbCurrent->m_bOverlapped=bOverlapped;

    // Передать сигнал рабочему потоку, чтобы он приступил к поиску
     SetEvent(pbCurrent->m_hMainEvent);
   }

  // Ожидать завершения поиска в своих буферах всеми рабочими
  // потоками. Каждый из рабочих потоков переводит в сигнальное
  // состояние предоставленный ему объект события, когда становится
  // готовым к обработке следующего буфера
  WaitForMultipleObjects(dwNumThreads,hEvents,true,INFINITE);

   dwlFilePos+=dwClusterSize*dwNumThreads;

  } while (dwlFilePos<dwlFileSize);

  // Получить общий итог и уничтожить объекты поиска
```

```
    DWORD dwFindCount=0;
    CBufSearch *pbNext;
    for (; NULL!=pbFirst; pbFirst=pbNext) {
    pbFirst->m_bTerminated=true;
     dwFindCount+=pbFirst->m_dwFindCount;
    pbNext=pbFirst->m_pbNext;
    delete pbFirst;
    }
  // Закрыть дескрипторы файла, потока и события
  for (i=0; i<dwNumThreads; i++) {
    CloseHandle(hThreads[i]);
    CloseHandle(hEvents[i]);
  }

  CloseHandle(hFile);
  CloseHandle(OverlappedIO.hEvent);

  // Освободить буфера разборки
  for (DWORD j=0; j<dwNumThreads; j++)
    VirtualFree(pSegments[j].Buffer,0,MEM_RELEASE);

  // Освободить память, полученную во всех предыдущих операциях
  // распределения памяти из динамической области памяти, уничтожив
  // созданную в программе динамическую область памяти
  HeapDestroy(hPrivHeap);

  // Возвратить данные о количестве найденных совпадений, относящиеся
  // к указанному файлу
  return dwFindCount;
}

// Выполнить поиск заданной подстроки в файлах, имена которых
// соответствуют указанной маске
bool SearchFiles(char *szFileMask, char *szSearchStr)
{

  char szPath[MAX_PATH+1];

  // Извлечь обозначение пути к файлу из указанной маски
  char *p=strrchr(szFileMask,'\\');
  if (p) {
    strncpy(szPath,szFileMask,(p-szFileMask)+1);
    szPath[(p-szFileMask)+1]='\0';
  }
  else
    // Если путь не указан, использовать текущий каталог
    GetCurrentDirectory(MAX_PATH,szPath);

  // В случае необходимости добавить заключительный символ обратной
  // косой черты
  if ('\\'!=szPath[strlen(szPath)-1])
    strcat(szPath,"\\");

  printf("Searching for %s in %s\n\n",szSearchStr,szFileMask);

    // Обработать в цикле все файлы, имена которых соответствуют
```

```
    // указанной маске, и выполнить поиск заданной подстроки
    WIN32_FIND_DATA fdFiles;
    HANDLE hFind=FindFirstFile(szFileMask,&fdFiles);

    if (INVALID_HANDLE_VALUE == hFind) {
    printf("No files match the specified mask\n");
    return false;
    }

    // Определить количество процессоров в текущей системе. Эта
    // величина будет использоваться для вычисления количества потоков
    // ввода-вывода, с помощью которых должен осуществляться поиск
    // в каждом файле
    SYSTEM_INFO si;
    GetSystemInfo(&si);

    // Определить размер кластера на диске. Эта величина всегда
    // является кратной размеру сектора, поэтому хорошо подходит
    // для использования во вводе-выводе на основе сборки-разборки
    DWORD dwSectorsPerCluster;
    DWORD dwBytesPerSector;
    DWORD dwNumberOfFreeClusters;
    DWORD dwTotalNumberOfClusters;
    GetDiskFreeSpace(NULL,&dwSectorsPerCluster,
                     &dwBytesPerSector,
                     &dwNumberOfFreeClusters,
                     &dwTotalNumberOfClusters);

    DWORD dwClusterSize=(dwSectorsPerCluster * dwBytesPerSector);

    DWORD dwFindCount=0;
    do {
    dwFindCount+=SearchFile(dwClusterSize,
                            si.dwNumberOfProcessors*
                            IO_STREAMS_PER_PROCESSOR,
                            szPath,
                            fdFiles.cFileName,
                            szSearchStr);
    } while (FindNextFile(hFind,&fdFiles));

    FindClose(hFind);
    printf("\nTotal hits for %s in %s:\t%d\n",
         szSearchStr,szFileMask,
         dwFindCount);
  return true;
}

int main(int argc, char* argv[])
{
  if (argc<3) {
    printf("Usage is: fstring_scatter filemask searchstring\n");
    return 1;
  }

  try
  {
```

```
    return (!SearchFiles(argv[1], argv[2]));
  }
  catch (...)
  {
    printf("Error reading file\n");
    return 1;
  }
}
```

6. Обратите внимание на то, что, как и в других примерах использования операций асинхронного ввода-вывода, приведенных в данной книге, необходимо учитывать в коде возможность того, что в операционной системе Windows будет принято решение о синхронном выполнении затребованной операции ввода-вывода. Если это произойдет, то каждой переменной `m_bOverlapped` каждого экземпляра класса `CBufSearch` будет присвоено значение `FALSE` для того, чтобы в нем не предпринимались попытки перейти в состояние ожидания завершения этой операции ввода-вывода с помощью функции `GetOverlappedResult`.

7. Каждому экземпляру класса `CBufSearch` присваивается порядковый индексный номер для того, чтобы можно было использовать этот индекс для вычисления смещения в файле, с которого он начинает обработку переданного ему буфера. Это смещение требуется экземпляру класса для того, чтобы он мог точно указывать местонахождение во входном файле каждого обнаруженного им совпадения. В примере приложения `fstring` для выборки начального смещения применялась структура `OVERLAPPED`, поскольку с каждым объектом `CBufSearch` была связана отдельная такая структура. А в данном приложении для всех объектов `CBufSearch` используется одна структура `OVERLAPPED`, поскольку в данный конкретный момент времени выполняется лишь одна асинхронная операция в силу того, что функция `ReadFileScatter` способна заполнять несколько буферов чтения в одном вызове. А поскольку структура `OVERLAPPED` теперь отражает начальную позицию для всей операции ввода-вывода со сборкой-разборкой, а не для отдельной асинхронной операции, требуется другой метод вычисления точного смещения в файле для каждого найденного совпадения. Именно для этой цели предназначена переменная `m_iIndex` класса `CBufSearch`. Значение этой переменной присваивается конструктором класса с использованием значения индекса, переданного во время создания объекта. Значение переменной представляет собой порядковый номер объекта `CBufSearch` в связанном списке этих объектов. В том цикле, в котором происходит создание связанного списка объектов `CBufSearch`, итерации осуществляются в обратном направлении, поскольку добавление новых объектов всегда происходит в голове списка. Это приводит к тому, что последний добавленный объект становится головой списка, поэтому при выполнении итераций в цикле в прямом направлении значения индексов изменялись бы от больших к меньшим. Но важно правильно сопровождать значения порядковых индексов и не нарушать порядок объектов в связанном списке, поскольку функция `ReadFileScatter` помещает данные в буфера чтения в последовательном порядке. Это означает, что в первый буфер поступает первый фрагмент, считанный из файла, во второй буфер — второй фрагмент и так далее, пока не будут заполнены все буфера.

8. Обратите внимание на то, что количество элементов в массиве `pSegments` превышает на единицу количество рабочих потоков. В этом состоит одно из требований по использованию функций ввода-вывода со сборкой-разборкой — последним элементом в массиве буферов должен быть NULL-указатель. Учитывая то, что в этом приложении предусмотрено заполнение нулями массива при его распределении, после чего с последним элементом массива больше не выполняются какие-либо действия, указанные выше условия полностью соблюдаются.
9. Указатель на буфер чтения каждого объекта `CBufSearch` передается при создании объекта. Это тот же буфер, который непосредственно заполняется данными с помощью функции `ReadFileScatter`; поскольку функция `ReadFileScatter` способна выполнять разборку данных, считанных ею из файла, с последовательной записью в несколько буферов, то нет необходимости заполнять эти буфера один за другим или использовать промежуточный непрерывный буфер.
10. После возврата из функции `ReadFileScatter` функция `SearchFile` передает каждому из рабочих потоков сигнал о том, что они могут приступить к обработке их выходных буферов. Если операция была инициализирована асинхронно, то в каждом объекте `CBufSearch` вызывается функция `GetOverlappedResult` для перехода в состояние ожидания завершения этой операции. А поскольку во всех этих объектах совместно используется одна и та же структура `OVERLAPPED`, которая была инициализирована и передана из функции `SearchFile`, то во всех этих объектах, по сути, происходит ожидание перехода в сигнальное состояние одного и того же объекта события, который был первоначально связан со структурой `OVERLAPPED`. В этом состоит одна из причин того, почему так важно, чтобы в качестве этого события применялось событие с отключением вручную. Если бы оно было создано как событие с автоматическим отключением, то после перехода события в сигнальное состояние активизировался бы только один ожидающий поток, поскольку побочным результатом успешного завершения состояния ожидания этого потока стало бы немедленное отключение сигнального состояния события, т.е. переход в несигнальное состояние.

Это приложение может служить характерным примером использования ввода-вывода со сборкой-разборкой. Внимательно изучите данное приложение, применив пошаговое выполнение его кода с помощью отладчика Visual C++. Обратите особое внимание на то, выполняется ли операция ввода-вывода синхронно или асинхронно, и каким образом в приложении учитывается каждая из этих ситуаций.

Ввод-вывод со сборкой-разборкой. Резюме

Ввод-вывод со сборкой-разборкой позволяет заполнять в потоке несколько буферов данными из непрерывной области файла, а также записывать содержимое нескольких буферов памяти, не относящихся к одной непрерывной области, в непрерывную область файла. До появления средств ввода-вывода со сборкой-разборкой в приложении в случае необходимости записать несколько не расположенных непрерывно буферов на диск приходилось либо записывать их отдельно, либо копировать перед записью в промежуточный непрерывный буфер. Ни тот, ни другой вариант не является достаточно эффективным, поэтому на уровне операционной системы была введена поддержка ввода-вывода со сборкой-

разборкой для обеспечения более эффективного выполнения операций ввода-вывода такого типа в программах, аналогичных SQL Server.

Требования к выполнению ввода-вывода со сборкой-разборкой представляют собой сочетание требований, которые распространяются на асинхронный и небуферизованный ввод-вывод. Это связано с тем, что ввод-вывод со сборкой-разборкой является небуферизированным, кроме того, по умолчанию выполняется асинхронно.

Ввод-вывод со сборкой-разборкой широко используется в программе SQL Server при выполнении таких операций, как чтение и запись файлов базы данных и файлов журналов, связанных с базами данных. Поскольку буфера, в которые должны быть считаны данные или из которых они должны быть записаны, могут находиться в буферном пуле в областях, не являющихся непрерывными, ввод-вывод со сборкой-разборкой позволяет в программе SQL Server обеспечить чтение и запись данных буферного пула с высокой производительностью.

Ввод-вывод со сборкой-разборкой. Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Буфера, распределенные для использования при вводе-выводе со сборкой-разборкой, должны занимать непрерывную область памяти.
2. Можно ли вызвать в потоке операцию `GetOverlappedResult`, чтобы обеспечить ожидание завершения операции ввода-вывода со сборкой-разборкой?
3. Подтвердите или опровергните следующее утверждение. При заполнении с помощью операции разборки множества буферов памяти данными из файла такое заполнение происходит в обратном порядке (в последний буфер массива поступает первый фрагмент дискового файла, в предпоследний — второй фрагмент и т.д.).
4. Какую функцию API-интерфейса Win32 можно вызвать в потоке для определения размера сектора диска?
5. Подтвердите или опровергните следующее утверждение. Объект файла, соответствующий файлу, который используется в операциях ввода-вывода со сборкой-разборкой, должен быть создан с установленным параметром `FILE_FLAG_SCATTER_GATHER`.
6. Какая функция API-интерфейса Win32 используется для сборки содержимого буферов из памяти и записи его в непрерывную область файла?
7. Подтвердите или опровергните следующее утверждение. В отличие от операций асинхронного ввода-вывода других типов операция ввода-вывода со сборкой-разборкой никогда не выполняется операционной системой Windows в синхронном режиме.
8. Допустим, что в приложении осуществляется чтение файла в цикле с помощью функции `ReadFileScatter`. Нужно ли корректировать в этом прило-

жении значения элементов структуры OVERLAPPED, передаваемой в функцию ReadFileScatter при каждой операции чтения? Почему да или почему нет?

9. Можно ли использовать в потоке ввод-вывод со сборкой-разборкой для выполнения операций записи в области, состоящие из нескольких несмежных участков файла, если файл был открыт для произвольного доступа?
10. Подтвердите или опровергните следующее утверждение. По умолчанию операционная система Windows обходит системный кэш при выполнении операций ввода-вывода со сборкой-разборкой.

Порты завершения ввода-вывода

В этом разделе рассматриваются средства порта завершения ввода-вывода операционной системы Windows и приведен пример приложения, в котором эти средства используются для управления параллельной работой и обеспечения синхронизации потоков. Если читатель еще не ознакомился с предыдущей частью данной главы, рекомендуем сделать это, прежде чем приступать к изучению темы этого раздела. Многие рассматриваемые здесь понятия основаны на тех определениях, которые были приведены выше в настоящей главе.

Порты завершения ввода-вывода. Основные термины и определения

- **Порт завершения ввода-вывода.** Механизм, позволяющий в потоках обеспечивать ожидание завершения операций асинхронного ввода-вывода и/или координировать доступ к ресурсам с помощью высокоэффективных средств. Порт завершения ввода-вывода ассоциируется с файлом, в результате чего после завершения операции асинхронного ввода-вывода, которая была инициализирована в этом файле, в очередь данного порта передается пакет завершения ввода-вывода. В потоках можно организовать ожидание поступления этих пакетов и выполнение соответствующих действий вместо ожидания завершения непосредственно самих асинхронных операций. Более того, порт завершения ввода-вывода позволяет управлять степенью распараллеливания (количеством потоков, которым разрешено одновременно обрабатывать пакеты завершения) и может активизировать ожидающие потоки, если какой-либо из активных потоков становится заблокированным. Порт завершения ввода-вывода может быть также создан, не будучи ассоциированным с файлом. При такой организации работы в приложении можно вызывать функцию PostQueuedCompletionStatus API-интерфейса для передачи в очередь порта пакетов завершения ввода-вывода специального назначения, после чего эти пакеты могут извлекать из очереди другие потоки и отвечать на них.

Порты завершения ввода-вывода. Основные функции API-интерфейсов

Основные функции API-интерфейсов, относящиеся к портам завершения ввода-вывода, приведены в табл. 5.7.

Таблица 5.7. Основные функции API-интерфейсов, относящиеся к портам завершения ввода-вывода

Функция	Описание
`CreateIoCompletionPort`	Создать порт завершения ввода-вывода и в качестве необязательного действия связать его с файлом
`GetQueuedCompletionStatus`	Проверить поставленный в очередь пакет завершения ввода-вывода и удалить его из очереди, если он имеется; кроме того, связать вызывающий поток с портом
`PostQueuedCompletionStatus`	Отправить определяемый приложением пакет завершения специального назначения в порт завершения ввода-вывода

Краткий обзор

Одна из трудностей, с которыми сталкиваются проектировщики серверных приложений, состоит в принятии обоснованного решения о том, насколько параллельным должно быть сделано это приложение – сколько потоков должно быть создано для обработки клиентских запросов. Серверное приложение, в котором слишком мало потоков, скорее всего, вынудит клиентов сталкиваться с исчерпанием ресурсов; клиентам придется проводить лишнее время в ожидании того момента, когда освободится один из ограниченного количества рабочих потоков сервера, чтобы предоставить им доступ к серверу. (Хотя работа серверного приложения может быть организована с помощью одного потока таким образом, чтобы этот поток мог отвечать на запросы сразу многих клиентов, данный подход характеризуется значительной сложностью и не позволяет воспользоваться преимуществами наличия на компьютере нескольких процессоров.)

После перехода в другую крайность может оказаться, что в серверном приложении имеется слишком много потоков, которые, скорее всего, будут затрачивать значительное количество времени на переключение контекста. Применение слишком большого количества потоков обычно приводит к возникновению ситуации пробуксовки потоков, в которой многочисленные потоки активизируются, выполняют за отведенный им интервал процессорного времени определенную работу, переходят в заблокированное состояние, ожидая завершения операции ввода-вывода или перехода в сигнальное состояние какого-то синхронизационного объекта, а затем снова блокируются на время ожидания нового запроса. При этом непрерывно происходит так, что потоки планируются для выполнения на процессоре (процессорах) системы и снова отключаются от процессора, хотя в конечном итоге объем выполненной ими полезной работы оказывается весьма небольшим.

В эффективном серверном приложении поддерживается равновесие между стремлениями ограничить переключение контекста и обеспечить высокую степень распараллеливания. Планировщик высокопроизводительного программного обеспечения поддерживает поток в рабочем состоянии лишь до тех пор, пока остаются невыполненными рабочие запросы, а не передает процессор в распоряжение другого потока, который затем выполняет ту же самую работу, пока данный поток простаивает. Для разработки проекта такого типа необходимо применять разные критерии при оценке количества рабочих потоков и запросов на выполнение работы. В идеальном случае должно быть достаточно иметь только один активный поток в расчете на каждый процессор в любой конкретный момент времени и стараться избегать необходимости блокировать поток, пока еще имеется работа, которая должна быть выполнена с его помощью. Как будет описано в главе 10, именно таким образом организовано функционирование планировщика непривилегированного режима (User Mode Scheduler) программы SQL Server.

Для того чтобы можно было исключить необходимость переключения контекста в серверной программе и вместе с тем обеспечить максимальное распараллеливание, нужно иметь в своем распоряжении способ активизации другого потока на то время, пока один из потоков, обрабатывающих клиентский запрос, становится заблокированным (например, на то время, когда поток ожидает завершения операции ввода-вывода или освобождения другого ресурса). Именно для этой цели могут применяться порты завершения ввода-вывода. Механизм завершения ввода-вывода операционной системы Windows позволяет активно управлять степенью распараллеливания работы в приложении, поэтому порт завершения ввода-вывода можно использовать для максимального уменьшения количества переключений контекста и вместе с тем поддержания максимально возможной нагрузки процессора (процессоров) компьютера на то время, пока заблокирован один из работающих потоков.

Для создания порта завершения ввода-вывода необходимо вызвать функцию `CreateIoCompletionPort` API-интерфейса Win32. Как правило, порт завершения ввода-вывода при его создании ассоциируется с некоторым файлом. После завершения одной из операций асинхронного ввода-вывода в этом файле операционная система Windows направляет в порт завершения ввода-вывода пакет завершения. Затем приложение может извлечь данный пакет из очереди и выполнить соответствующие действия.

Следует отметить, что порт завершения ввода-вывода может быть также создан таким образом, чтобы он не был ассоциирован с каким-либо конкретным файлом. При такой организации работы в приложении обычно вызывается функция `PostQueuedCompletionStatus` API-интерфейса Win32 для постановки в очередь порта пакетов завершения специального назначения. Как было указано в главе 3, в приложении этот механизм может использоваться для синхронизации доступа многочисленных потоков к ресурсу.

Функция `CreateIoCompletionPort` API-интерфейса Win32 позволяет задавать значение степени распараллеливания при создании порта в той программе, в которой вызывается эта функция. От значения степени распараллеливания зависит количество потоков, которые могут параллельно обрабатывать пакеты завершения ввода-вывода, полученные из порта. Если в качестве этого параметра

указано значение 0, операционная система Windows по умолчанию разрешает распараллеливать выполнение по такому количеству потоков, которое совпадает с количеством процессоров в системе. Как правило, компания Microsoft рекомендует задавать значение степени распараллеливания, приблизительно равное количеству процессоров в системе, и разрабатывать код приложения таким образом, чтобы можно было предотвратить блокировку любого рабочего потока в тех ситуациях, когда еще остаются необработанные запросы на выполнение работы. При такой организации функционирования приложения можно легко проводить эксперименты с различными значениями степени распараллеливания, чтобы добиться максимальной отдачи от эксплуатируемой системы.

Поток изымает пакет завершения ввода-вывода из очереди, вызывая функцию `GetQueuedCompletionStatus` API-интерфейса Win32. Побочным следствием этой операции является также то, что данный поток ассоциируется с портом завершения. В любой момент времени поток может быть ассоциирован только с одним портом завершения ввода-вывода. Независимо от того, чему равно количество рабочих потоков, которые ассоциируют себя с некоторым портом завершения, система предпринимает попытки поддерживать количество рабочих потоков, равное или меньшее по сравнению со значением степени распараллеливания, переданным в функцию `CreateIoCompletionPort`.

Функция `GetQueuedCompletionStatus` возвращает целый ряд выходных параметров, которые могут использоваться в вызывающем потоке. Наиболее важным из этих параметров является указатель на структуру `OVERLAPPED`, которая служила для инициализации текущего запроса асинхронного ввода-вывода. Этот указатель можно применять для проверки того, какая именно операция ввода-вывода была только что завершена, если в приложении одновременно выполняются несколько перекрывающихся операций ввода-вывода. Порядок, в котором осуществляется выборка пакетов завершения ввода-вывода из очереди порта, не говорит ни о чем, поскольку операции асинхронного ввода-вывода, по завершении которых эти пакеты были поставлены в очередь, могли потребовать для своего выполнения разное количество времени. А в связи с тем, что для каждой операции асинхронного ввода-вывода должна быть предусмотрена отдельная структура `OVERLAPPED`, то всегда существует возможность спроектировать программное обеспечение таким образом, чтобы можно было использовать указатель `OVERLAPPED`, возвращаемый функцией `GetQueuedCompletionStatus`, для поиска буфера, который был только что заполнен считанными данными или записан на диск с помощью операции асинхронного ввода-вывода.

Вторым важным выходным параметром, возвращаемым функцией `GetQueuedCompletionStatus`, является количество переданных байтов. Это значение можно использовать для определения того, сколько байтов было считано или записано с помощью операции асинхронного ввода-вывода, ассоциированной с портом.

Операционная система Windows следит за тем, какие потоки ассоциированы с тем или иным портом завершения, и находятся ли эти потоки в рабочем состоянии. Если определенный поток, который активно обрабатывал пакеты завершения, становится заблокированным, то операционная система Windows предпринима-

ет попытки активизировать другой поток, который ожидает освобождения данного порта. Безусловно, это означает, что количество рабочих потоков, активно обрабатывающих пакеты завершения, может на время превысить заданное значение степени распараллеливания. Рассмотрим, что происходит, если активный поток становится заблокированным, активизируется другой поток, а первый поток выходит из заблокированного состояния, в то время как второй поток все еще работает. При таком развитии событий количество активных рабочих потоков на время превышает значение степени распараллеливания, которое задано для этого порта. Ни один из прочих потоков, ожидающих освобождения порта, не получит разрешения на выполнение до тех пор, пока количество активных потоков не станет ниже значения степени распараллеливания, которое было первоначально задано для этого порта.

Операционная система Windows выводит из заблокированного состояния потоки, которые заблокированы в ходе ожидания освобождения порта завершения, в порядке очереди со стековой организацией (первым активизируется поток, который перешел в состояние ожидания самым последним). После того как поток выходит из заблокированного состояния для того, чтобы изъять из очереди порта один из пакетов завершения ввода-вывода, он получает самый старый пакет из этой очереди.

Организация работы программы с помощью порта завершения ввода-вывода позволяет получить значительные преимущества с точки зрения масштабирования, которые лучше всего продемонстрировать с помощью примера. Предположим, что рассматривается серверное приложение, в котором предусмотрено двенадцать рабочих потоков, и допустим, что в это приложение поступили двенадцать клиентских запросов. Если не используется порт завершения ввода-вывода или аналогичный механизм создается разработчиком самостоятельно, то может быть предусмотрено одновременное выполнение всех двенадцати потоков. Такой подход предоставляет максимальное распараллеливание работы по выполнению клиентских запросов в данном приложении, но если количество процессоров в системе меньше двенадцати, то это сопряжено с относительно высокой интенсивностью переключения контекста. Иными словами, процессор (процессоры) компьютера не будет расходовать все свое время на выполнение рабочих запросов, поскольку определенная часть процессорного времени (возможно, даже имеющая относительно высокое процентное соотношение с другими частями) будет затрачиваться на переключение между потоками; это связано с тем, что количество процессоров является недостаточным для того, чтобы действительно обеспечить с их помощью параллельную работу. Операционная система Windows вынуждена имитировать такое кажущееся распараллеливание, непрерывно меняя местами потоки, подключаемые к процессору (процессорам) системы, и отключая их от процессора.

Но предположим, что в приложении для управления организацией распараллеливания используется порт завершения ввода-вывода, а также, что в системе имеются только два процессора. Если порт завершения ввода-вывода создан с предусмотренным в нем по умолчанию значением степени распараллеливания, то он позволит обеспечить параллельное выполнение только двух рабочих потоков. Но вместо непрерывного отключения или подключения к процессорам эти два рабочих потока будут продолжать активно использовать отведенные им процессоры

(при условии, что данные потоки не станут заблокированными) до тех пор, пока не будут обработаны все двенадцать запросов на выполнение работы. При условии, что все запросы на выполнение работы имеют одинаковую продолжительность выполнения, каждый поток, по всей вероятности, обработает примерно по шесть из этих запросов. Таким образом, подобная организация функционирования позволяет добиться чистого выигрыша в производительности, поскольку процессоры затрачивают почти все свое время на обработку клиентских запросов, и не происходит потерь процессорного времени на переключение между рабочими потоками.

У читателя могут возникнуть вопросы: "Почему бы просто не ограничить общее количество рабочих потоков, чтобы оно соответствовало количеству процессоров на компьютере? Позволит ли это решить проблему с переключением контекста?". Недостатком данного подхода является то, что потоки могут оказаться заблокированными в разное время в ходе их выполнения. А если на эксплуатируемом компьютере имеются только два процессора и поэтому создаются всего лишь два рабочих потока, то что произойдет, если оба эти потока станут заблокированными на время ожидания завершения ввода-вывода или перехода в сигнальное состояние какого-то синхронизационного объекта? Может оказаться, что процессоры простаивают, в то время как потоки находятся в состоянии ожидания. Если же, с другой стороны, создается больше рабочих потоков, чем процессоров, но для управления степенью распараллеливания используется порт завершения ввода-вывода, то после перехода в состояние ожидания одного из активных потоков операционная система Windows разрешает приступить к выполнению работы другому потоку, тем самым более полно используя процессорное время системы.

Упражнения

Как и применительно ко всем другим типам ввода-вывода Windows, которые рассматривались в этой книге, изучение возможностей портов завершения ввода-вывода также будет проведено на основе примера. Пример приложения `fstring_io_comp`, представленный в следующем упражнении, является вариантом примера приложения `fstring`, который рассматривался в разных формах во всей данной главе. Он аналогичен всем другим указанным примерам приложений ввода-вывода в том, что в нем осуществляется поиск указанной подстроки в файлах с именами, соответствующими заданной маске. А отличие данного приложения от других заключается в том, что в нем для управления степенью распараллеливания, а также для упрощения синхронизации потоков используется порт завершения ввода-вывода.

Упражнение 5.5. Многопотоковая утилита поиска в нескольких файлах, в которой используется порт завершения ввода-вывода

1. Загрузите пример приложения `fstring_io_comp` из подкаталога `CH05\fstring_io_comp` компакт-диска в среду разработки Visual Studio.
2. В приложении `fstring_io_comp` используются два основных класса — `CBufSearch` и `CIoBuf`. Описание класса `CBufSearch` приведено в других

примерах приложений, в которых применяется асинхронный ввод-вывод, показанных выше в данной главе. В данном приложении этот класс в основном остался неизменным. Но в рассматриваемом приложении применяется новый класс — `CIoBuf`. Дополнительные сведения о нем приведены ниже.

3. Общие принципы работы приложения `fstring_io_comp` состоят в следующем.
   а. В главном потоке открывается файл, предназначенный для выполнения в нем поиска, который ассоциируется с портом завершения ввода-вывода.
   б. В главном потоке начинается выполнение цикла, в котором буфер чтения изымается из очереди, представляющей собой связный список объектов `CIoBuf` (путем изменения состояния буфера с `BUF_STATE_INACTIVE` на `BUF_STATE_READING`), и этот буфер передается в функцию `ReadFile` для использования в операции асинхронного чтения файла.
   в. После завершения каждой асинхронной операции в очередь порта завершения ставится пакет завершения ввода-вывода.
   г. Метод `CIoBuf::CheckForIoPacketAndSetState` (вызываемый из рабочего потока) изымает из очереди этот пакет завершения, вызывая функцию `GetQueuedCompletionStatus` API-интерфейса Win32, и использует предусмотренный в нем указатель на структуру `OVERLAPPED` для поиска соответствующего объекта `CIoBuf`, который был с самого начала ассоциирован с этой асинхронной операцией. После того как указанная функция находит соответствующий буфер, она переводит буфер в состояние `BUF_STATE_READY`.
   д. Метод `CBufSearch::Search` непрерывно опрашивает очередь запросов на выполнение работы для определения наличия в ней буфера, состояние которого было установлено в значение `BUF_STATE_READY`. Найдя такой буфер, этот метод изменяет состояние буфера на `BUF_STATE_SEARCHING`, осуществляет просмотр буфера для обнаружения искомой подстроки и выводит строки с обнаруженными совпадениями на терминал. Закончив работу с буфером, данный метод выводит его из очереди, снова задавая значение его состояния, равное `BUF_STATE_INACTIVE`.
   е. Эти действия продолжаются до тех пор, пока не будут полностью считаны все входные файлы и в них не будет завершен поиск.
4. Приложение `fstring_io_comp` состоит из трех основных модулей исходного кода — `fstring_io_comp.cpp`, `bufsrch.cpp` и `iobuf.cpp`. Первый из них, `fstring_io_comp.cpp` (листинг 5.7), аналогичен главным модулям исходного кода в остальных примерах приложений `fstring`, приведенных в этой главе. Начнем с этого модуля.

Листинг 5.7. Главный модуль исходного кода для `fstring_io_comp` (`fstring_io_comp.cpp`)

```
// fstring_io_comp.cpp. Многопотоковая процедура поиска в файле,
//                      в которой используется порт завершения
//                      ввода-вывода
//

#define _WIN32_WINNT 0x500

#include "stdafx.h"
```

```
#include "windows.h"
#include "stdlib.h"
#include "process.h"
#include "bufsrch.h"
#include "iobuf.h"

#define IO_STREAMS_PER_PROCESSOR 2

// Процедура точки входа для рабочих потоков
unsigned __stdcall StartSearch(LPVOID lpParameter)
{

  // Привести параметр, передаваемый в функцию _beginthreadex, к типу
  // CBufSearch * и вызвать метод Search класса CBufSearch
return ((CBufSearch*)lpParameter)->Search();

}

void __stdcall DisplayOutput(DWORD dwParam)
{
  char *pszMsg=(char *)dwParam;
  printf(pszMsg);
}

// Выполнить поиск в указанном файле заданной искомой подстроки с
// использованием небуферизованного, асинхронного ввода-вывода
DWORD SearchFile(DWORD dwClusterSize,
                         DWORD dwNumStreams,
                         char *szPath,
                          char *szFileName,
                          char *szSearchStr,
                         HANDLE hMainThread)
{
  char szFullPathName[MAX_PATH+1];
  DWORD dwNumThreads;
  HANDLE hPrivHeap;
  HANDLE *hThreads;

  strcpy(szFullPathName,szPath);
  strcat(szFullPathName,szFileName);

  // Открыть файл как для небуферизованного, так и для совмещенного
  // (асинхронного) ввода-вывода
  HANDLE hFile=CreateFile(szFullPathName,
                            GENERIC_READ,FILE_SHARE_READ,
                            NULL,
                            OPEN_EXISTING,
                            FILE_ATTRIBUTE_NORMAL
                            | FILE_FLAG_OVERLAPPED
                            | FILE_FLAG_NO_BUFFERING
                            ,NULL);

  if (INVALID_HANDLE_VALUE==hFile) {
    printf("Error opening file.  Last error=%d\n",
      GetLastError());
    return -1;
```

```
  }
DWORD dwFileSizeHigh;

  DWORD dwFileSizeLow=GetFileSize(hFile,&dwFileSizeHigh);

  DWORD dwlFileSize=(dwFileSizeHigh*MAXDWORD)+
         dwFileSizeLow;

  DWORD dwNumClusts=dwlFileSize / dwClusterSize;
  if (dwNumClusts<1) dwNumClusts=1;

  // Если размер файла меньше 4 Гбайт и количество затребованных
  // потоков (речь идет о потоках ввода-вывода) меньше количества
  // кластеров, установить количество потоков равным количеству
  // кластеров
  if ((dwlFileSize<0xFFFFFFFF) && (dwNumStreams>dwNumClusts))
    dwNumThreads=dwNumClusts;
  else
    dwNumThreads=dwNumStreams;

#if(_DEBUG)
  printf("Using %d threads\n\n",dwNumThreads);
#endif

  // Создать закрытую динамическую область памяти, чтобы можно было за
  // один раз освободить память, распределенную во всех операциях
  hPrivHeap=HeapCreate(0,0,0);

  // Создать массив потоков
  hThreads=(HANDLE *)HeapAlloc(hPrivHeap,
                               HEAP_ZERO_MEMORY,
                               dwNumThreads*sizeof(HANDLE));
  if (NULL==hThreads) {
    printf("Error allocating worker thread array.  Aborting.\n");
     return -1;
  }

  // Создать порт завершения ввода-вывода
  HANDLE hPort=CreateIoCompletionPort(hFile,NULL,0,0);
  if (INVALID_HANDLE_VALUE==hPort) {
    printf(
      "Error creating IO completion port.  Last error=%d\n",
       GetLastError());
    return -1;
  }
    // Создать рабочие потоки, а также объекты CBufSearch и CIoBuf
    CBufSearch *pbFirst=NULL;
    CIoBuf *pIoFirst=NULL;
    unsigned uThreadId;

    for (DWORD i=0; i<dwNumThreads; i++) {

    pIoFirst=new CIoBuf(pIoFirst,hPort,dwClusterSize+1);

    pbFirst=new CBufSearch(pbFirst,
                           szFileName,
```

```
                                szSearchStr,
                                &DisplayOutput,
                                 hMainThread);

    hThreads[i]=
    (HANDLE)_beginthreadex(NULL,
                                0,
                                &StartSearch,
                                pbFirst,
                                CREATE_SUSPENDED,
                                 &uThreadId);

    if (!hThreads[i]) {
      printf("Error creating thread.  Aborting.\n");
      return -1;
    }
    }

    // Установить указатель объектов CBufSearch на начало списка CIoBuf
    pbFirst->s_pIoFirst=pIoFirst;

    // Установить указатель объектов CIoBuf на начало списка CIoBuf
    pIoFirst->s_pIoFirst=pIoFirst;

    // Задать значения статических переменных для того, чтобы
    // можно было осуществлять сразу несколько операций
    // поиска в файле
pIoFirst->s_bTerminated=false;
    pIoFirst->s_bOverlapped=true;

    // После определения значений статических переменных
    // запустить рабочие потоки
    for (i=0; i<dwNumThreads; i++)
    ResumeThread(hThreads[i]);

    // Главный цикл - обрабатывать файл в цикле, читая из него
    // фрагменты с размером dwClusterSize
    DWORDLONG dwlFilePos=0;
    do {
    for (CBufSearch *pbCurrent=pbFirst;
      NULL!=pbCurrent;
      pbCurrent=pbCurrent->m_pbNext) {

      CIoBuf *pIoBuf=
            pIoFirst->SpinToFindBuf(BUF_STATE_INACTIVE,
                                    BUF_STATE_READING);

       // Задать начальное смещение для следующей операции чтения
      pIoBuf->m_OverlappedIO.OffsetHigh=
         (DWORD)(dwlFilePos / MAXDWORD);
      pIoBuf->m_OverlappedIO.Offset=
         (DWORD)(dwlFilePos % MAXDWORD);

      // Заполнить буфер чтения нулями, чтобы не встречались совпадения
      / с искомой подстрокой в конце частично заполненного буфера
      / (оставшиеся от ранее считанных фрагментов)
```

```
    ZeroMemory(pIoBuf->m_szBuf,dwClusterSize+1);

    // Считать из файла данные в объеме полного буфера, по
    // возможности используя асинхронный ввод-вывод
    if (!ReadFile(hFile,pIoBuf->m_szBuf,
                        dwClusterSize,
                         &pIoBuf->m_dwBytesRead,
                        &pIoBuf->m_OverlappedIO)) {

      DWORD dwLastErr=GetLastError();
     if (ERROR_IO_PENDING!=dwLastErr)     {

        // Завершить выполнение главного цикла потока при
        // обнаружении любой ошибки, исключая ERROR_IO_PENDING,
        // но включая EOF
        InterlockedExchange(
              (LPLONG)&pIoBuf->s_bTerminated,
              (long)true);

       // Осуществить аварийное завершение, если ошибка не
       // относится к типу EOF
       if (ERROR_HANDLE_EOF!=dwLastErr) {
         printf(
         "Error reading file.  Last error=%d",
         dwLastErr);
            return -1;
       }
       break;
      }
     else {
        // Имеет место асинхронная операция
        InterlockedExchange(
              (LPLONG)&pIoBuf->s_bOverlapped,
              (long)true);
      }
   }
   else {
      // Функция ReadFile возвратила истинное значение; операция
      // является синхронной
      InterlockedExchange(
            (LPLONG)&pIoBuf->s_bOverlapped,
           (long)false);
     pIoBuf->SetState(BUF_STATE_READY);

   }

   dwlFilePos+=dwClusterSize;
}

} while ((dwlFilePos<dwlFileSize) &&
   (!pIoFirst->s_bTerminated));

// Передать сигнал о том, что чтение файла закончено
InterlockedExchange(
     (LPLONG)&pIoFirst->s_bTerminated,
     (long)true);
```

```
  // Ожидать завершения поиска в своих буферах всеми рабочими потоками
  WaitForMultipleObjects(dwNumThreads,hThreads,
                         true,
                         INFINITE);

  // Удалить из очереди сообщения обо всех операциях вывода, которые
  // были поставлены в очередь с помощью вызовов
  while (WAIT_IO_COMPLETION==SleepEx(0,true));

  // Получить общий итог и уничтожить объекты поиска
  DWORD dwFindCount=0;
  CBufSearch *pbNext;
  for (; NULL!=pbFirst; pbFirst=pbNext) {
    dwFindCount+=pbFirst->m_dwFindCount;
     pbNext=pbFirst->m_pbNext;
    delete pbFirst;
  }

  // Удалить объекты буферов
  CIoBuf *pIoNext;
  for (; NULL!=pIoFirst; pIoFirst=pIoNext) {
    pIoNext=pIoFirst->m_pIoBufNext;
    delete pIoFirst;
  }

  // Закрыть порт завершения ввода-вывода
  CloseHandle(hPort);

  // Закрыть дескрипторы потоков
  for (i=0; i<dwNumThreads; i++) {
     CloseHandle(hThreads[i]);
  }

  CloseHandle(hFile);

  // Освободить память, полученную во всех предыдущих операциях
  // распределения памяти из динамической области памяти,
  // уничтожив созданную в программе динамическую область памяти
  HeapDestroy(hPrivHeap);

  // Возвратить данные о количестве найденных совпадений, относящиеся
  // к указанному файлу
  return dwFindCount;
}
// Выполнить поиск заданной подстроки в файлах, имена которых
// соответствуют указанной маске
bool SearchFiles(char *szFileMask, char *szSearchStr)
{

  char szPath[MAX_PATH+1];

  // Извлечь обозначение пути к файлу из указанной маски
  char *p=strrchr(szFileMask,'\\');
  if (p) {
    strncpy(szPath,szFileMask,(p-szFileMask)+1);
    szPath[(p-szFileMask)+1]='\0';
```

```
}
else
  // Если путь не указан, использовать текущий каталог
  GetCurrentDirectory(MAX_PATH,szPath);

// В случае необходимости добавить заключительный символ обратной
// косой черты
if ('\\'!=szPath[strlen(szPath)-1])
  strcat(szPath,"\\");

printf("Searching for %s in %s\n\n",szSearchStr,
  szFileMask);

HANDLE hMainThread=
      OpenThread(THREAD_ALL_ACCESS,
      0,
      GetCurrentThreadId());

// Обработать в цикле все файлы, имена которых соответствуют
// указанной маске, и выполнить поиск заданной подстроки
WIN32_FIND_DATA fdFiles;
HANDLE hFind=FindFirstFile(szFileMask,&fdFiles);

if (INVALID_HANDLE_VALUE == hFind) {
  printf("No files match the specified mask\n");
  return false;
}

// Определить количество процессоров в текущей системе. Эта величина
// будет использоваться для вычисления количества потоков
// ввода-вывода, с помощью которых должен осуществляться поиск
// в каждом файле
SYSTEM_INFO si;
GetSystemInfo(&si);

// Определить размер кластера на диске. Эта величина всегда является
// кратной размеру сектора, поэтому хорошо подходит для использования
// в небуферизованном вводе-выводе
DWORD dwSectorsPerCluster;
DWORD dwBytesPerSector;
DWORD dwNumberOfFreeClusters;
DWORD dwTotalNumberOfClusters;
GetDiskFreeSpace(NULL,&dwSectorsPerCluster,
                          &dwBytesPerSector,
                          &dwNumberOfFreeClusters,
                          &dwTotalNumberOfClusters);

DWORD dwClusterSize=(dwSectorsPerCluster * dwBytesPerSector);

DWORD dwFindCount=0;
do {
  dwFindCount+=
        SearchFile(  dwClusterSize,
                     si.dwNumberOfProcessors*
                     IO_STREAMS_PER_PROCESSOR,
                     szPath,
```

```
                        fdFiles.cFileName,
                        szSearchStr,
                        hMainThread);

  } while (FindNextFile(hFind,&fdFiles));

  FindClose(hFind);

  printf("\nTotal hits for %s in %s:\t%d\n",
    szSearchStr,szFileMask,dwFindCount);

  CloseHandle(hMainThread);

  return true;
}

int main(int argc, char* argv[])
{
  if (argc<3) {
    printf(
          "Usage is: fstring_io_comp filemask searchstring\n");
    return 1;
  }

  try
  {
    return (!SearchFiles(argv[1], argv[2]));
  }
  catch (...)
  {
    printf("Error reading file.  Last error=%d\n",
      GetLastError());
    return 1;
  }
}
```

5. Организация работы программы, необходимая для обработки в цикле файлов, имена которых соответствуют заданной маске, и вызова процедуры `SearchFile`, аналогична той, которая применяется в других примерах `fstring`, поэтому дополнительные сведения об этом здесь не приведены. Начнем изучение программы с описания вызова функции `CreateFile` в функции `SearchFile`.
6. Обратите внимание на то, что рассматриваемый файл открывается как для асинхронного (перекрывающегося), так и для небуферизованного ввода-вывода. Поскольку нам необходимо ассоциировать с этим файлом порт завершения ввода-вывода, то следует обеспечить в максимально возможной степени создание предпосылок для успешной инициализации операций асинхронного ввода-вывода. Порт завершения ввода-вывода, который ассоциирован с файлом, практически не применим, если ввод-вывод осуществляется в нем синхронно. А поскольку при открытии файла заданы оба из указанных необходимых параметров, то шансы на то, что будет обеспечено асинхронное выполнение ввода-вывода, становятся максимальными.

7. Затем необходимо рассмотреть, как осуществляется вызов функции `CreateIoCompletionPort`. Ей передается дескриптор рассматриваемого файла для того, чтобы она ассоциировала его с портом. В вызове этой функции в качестве значения степени распараллеливания указан 0, чтобы операционная система установила значение степени распараллеливания для данного порта, соответствующее количеству процессоров в системе.
8. После этого создаются рабочие потоки и объекты `CIoBuf` и `CBufSearch`. Еще до вызова функции `SearchFile` в функции `SearchFiles` осуществляется выборка значения количества процессоров в системе с помощью функции `GetSystemInfo` API-интерфейса Win32, затем это значение умножается на константу `IO_STREAMS_PER_PROCESSOR`, которая в настоящее время установлена равной 2. Затем это значение передается в функцию `SearchFile` с помощью параметра `dwNumStreams`. Данный параметр задает количество рабочих потоков, создаваемых в функции `SearchFile` для поиска в каждом файле. Это означает, что при количестве процессоров в системе, равном четырем, функция `SearchFile` создаст восемь рабочих потоков для поиска в файлах. Такая организация приложения принята для того, чтобы можно было обеспечить создание максимально возможной нагрузки на процессоры, поскольку предполагается, что каждый рабочий поток будет проводить определенное количество времени в состоянии ожидания завершения ввода-вывода. Но независимо от того, какое количество рабочих потоков будет создано в приложении. используемый порт завершения ввода-вывода все еще определяет количество активных потоков.
9. Класс `CBufSearch` имеет такое же назначение, как и во всех других примерах приложений, описанных в данной главе, в которых встречался этот класс. Класс `CBufSearch` выполняет поиск в текстовом буфере указанной подстроки и передает строки с обнаруженными совпадениями на терминал. А класс `CIoBuf` впервые введен в данном упражнении. При эксплуатации портов завершения ввода-вывода необходимо отдельно рассматривать запросы на выполнение работы и сами рабочие потоки (как уже было указано выше), поэтому относящиеся к вводу-выводу элементы, которые в других примерах приложения `fstring` были инкапсулированы в классе `CBufSearch`, теперь перенесены в их собственный класс `CIoBuf`. Класс `CIoBuf` представляет один из элементов в очереди запросов на выполнение работы; каждый экземпляр `CIoBuf` представляет собой буфер, в котором должен быть проведен поиск. Дополнительная информация о классах `CBufSearch` и `CIoBuf`, используемых в данном приложении, приведена ниже.
10. Каждый из потоков создается в приостановленном состоянии, чтобы можно было присвоить значение некоторым статическим переменным до того, как эти рабочие потоки приступят к выполнению. Успешное функционирование рабочих потоков зависит от того, правильно ли будут присвоены значения этим переменным, поэтому запуск потоков откладывается до того момента, как будет закончена их настройка.
11. Затем приложение входит в главный цикл обработки, в котором считывается входной файл, а запросы на выполнение операций асинхронного ввода-вывода и запросы на выполнение работы помещаются в очередь. Для передачи в очередь запросов на выполнение операций асинхронного ввода-вывода используются вызовы функции `ReadFile`. С другой стороны, запросы на

выполнение работы помещаются в очередь после того, как рабочий поток обнаруживает, что операционная система Windows отправила пакет завершения в очередь порта завершения ввода-вывода после окончания очередной операции асинхронного ввода-вывода.

12. Все структуры, необходимые для выполнения операции ввода-вывода, инкапсулированы в объекте `CIoBuf`. Членами класса `CIoBuf` являются буфер, в который поступают данные, считанные из файла, а также поле состояния и структура `OVERLAPPED`, необходимая для операций асинхронного ввода-вывода.

13. Поле состояния в экземпляре класса `CIoBuf` может принимать одно из четырех значений. В табл. 5.8 перечислены эти значения и указано, для чего они предназначены.

Таблица 5.8. Состояния `CIoBuf` и их значения

Состояние	Описание
`BUF_STATE_INACTIVE`	Буфер не занят и может использоваться в асинхронной операции чтения
`BUF_STATE_READING`	Буфер используется в операции чтения
`BUF_STATE_READY`	Буфер был заполнен в результате выполнения операции чтения и готов для проведения в нем поиска
`BUF_STATE_SEARCHING`	В буфере осуществляется поиск

14. Выполнение цикла чтения начинается с поиска буфера, состояние которого обозначено константой `BUF_STATE_INACTIVE`. Таковым является неактивный буфер, т.е. буфер, не используемый для какой-либо цели. Для поиска буфера применяется вызов функции `SpinToFindBuf`, поэтому поиск и приобретение данного буфера осуществляются за один шаг путем изменения значения его состояния на `BUF_STATE_READING`. Состояние `BUF_STATE_READING` указывает, что происходит чтение данных в буфер.

15. Как и в других примерах приложений с асинхронным вводом-выводом, учитывается возможность того, что в операционной системе Windows будет принято решение выполнять какой-то из запросов ввода-вывода синхронно. Если вызов функции `ReadFile` возвращает значение `TRUE`, то становится известно, что вызов выполняется в синхронном режиме, и затребованная в нем операция чтения уже закончена, поэтому состояние буфера немедленно изменяется на `BUF_STATE_READY`. Это означает, что буфер готов для проведения в нем поиска.

16. Кроме того, в случае синхронного чтения присваивается значение соответствующей статической переменной `CIoBuf`, чтобы в дальнейшем в программе не предпринимались попытки перейти в состояние ожидания появления пакета в очереди порта завершения ввода-вывода. Решение о применении в программе статической переменной основано на том предположении, что если одна из операций чтения оказалась синхронной, то с наибольшей вероятностью все остальные операции чтения в данном конкретном файле также будут выполнены синхронно.

17. Если же вызов функции `ReadFile` привел к инициализации операции асинхронного чтения, то необходимо подождать завершения этой операции и только по-

сле этого появится возможность изменить состояние буфера. Способ, применяемый для осуществления этих действий в классе CIoBuf, будет описан ниже.

18. Если читатель уже ознакомился с другими примерами приложений, в которых используется асинхронный ввод-вывод, приведенными в данной главе, то он, вероятно, заметил, что настоящее приложение характеризуется отсутствием объектов событий, которые обычно использовались для синхронизации рабочих потоков. Например, вместо ожидания перехода в сигнальное состояние одного из объектов в массиве объектов событий после завершения чтения из файла происходит ожидание перехода в сигнальное состояние (завершения самих потоков). А в данном приложении объекты событий не требуются, поскольку синхронизация потоков осуществляется с использованием порта завершения ввода-вывода и ряда функций комплексной блокировки. (Напомним, что семейство функций комплексной блокировки рассматривалось в главе 3.)
19. После завершения работы потоков происходит освобождение ресурсов и возврат данных о количестве совпадений в функцию SearchFiles.
20. Теперь рассмотрим класс CBufSearch этого приложения (листинг 5.8). В нем нет существенных различий по сравнению с классами CBufSearch, применяемыми в других примерах этой главы.

Листинг 5.8. Модуль исходного кода для класса CBufSearch (bufsrch.cpp)

```
// bufsrch.cpp. Вспомогательный класс, который используется для поиска
//              подстроки в буфере

#include "bufsrch.h"

CIoBuf *CBufSearch::s_pIoFirst=NULL;

// Конструктор
CBufSearch::CBufSearch(CBufSearch *pbNext,
                       char *szFileName,
                       char *szSearchStr,
                       PAPCFUNC pOutputCallback,
                       HANDLE hMainThread)
{

  // Записать в кэш параметры конструктора для дальнейшего
  // использования
  m_pbNext=pbNext;
  m_szFileName=szFileName;
  m_szSearchStr=szSearchStr;
  m_pOutputCallback=pOutputCallback;
  m_hMainThread=hMainThread;

  // Инициализировать все остальные переменные экземпляра
  m_dwFindCount=0;

  // Создать закрытую динамическую область памяти, которая будет
  // использоваться для вывода
  m_hOutputHeap=HeapCreate(HEAP_NO_SERIALIZE,0x1000,0);

}
```

```
CBufSearch::~CBufSearch()
{
  // Уничтожить закрытую динамическую область памяти, предназначенную
  // для вывода
  HeapDestroy(m_hOutputHeap);
}

// Найти начало строки по данным о смещении в буфере
char *CBufSearch::FindLineStart(char *szStartPos)
{
  char *szStart;
  for (szStart=szStartPos;
    ((szStart>m_pIoCurrent->m_szBuf) &&
     (cLINE_DELIM!=*(szStart-1))); szStart--);
  return szStart;
}

// Найти конец строки по данным о смещении в буфере; предполагается,
// что строка завершается нулевым символом
char *CBufSearch::FindLineEnd(char *szStartPos)
{
return strchr(szStartPos,cLINE_DELIM);
}

// Провести поиск в буфере чтения, чтобы найти каждую строку,
// содержащую ранее заданную искомую подстроку
bool CBufSearch::Search()
{
  char *szBol;
  char *szEol;
  char *szStringPos;
  DWORD dwNumChars;
  char *szStartPos;
  bool bRes=false;
  char szFmt[32];
  char szOffsetOutput[255];
  DWORDLONG dwlFilePos;
  char szMsg[1024];

  while (1) {

    __try
    {

      // Выполнять итерации цикла до тех пор, пока не будет найден
      // буфер, предназначенный для проведения поиска
      if ((NULL==(m_pIoCurrent=
        s_pIoFirst->SpinToFindBuf(BUF_STATE_READY,
                                  BUF_STATE_SEARCHING))) &&
                                 (s_pIoFirst->s_bTerminated))
        break;

      // Поиск начинается с начала буфера чтения
      szStartPos=m_pIoCurrent->m_szBuf;

       // Получить данные о смещении в файле из объекта буфера для
       // последующего вывода данных на терминал
```

```
    dwlFilePos=m_pIoCurrent->FilePos();

    // Продолжать выполнение итераций цикла до тех пор, пока маркер
    // начала поиска не равен NULL, находится в пределах буфера
    // чтения, а функция strstr продолжает находить в буфере чтения
    // искомую подстроку
    while ((szStartPos) &&
           (szStartPos<(m_pIoCurrent->m_szBuf+
                     m_pIoCurrent->m_dwBytesRead)-1) &&
            (NULL!=(szStringPos=
                strstr(szStartPos,m_szSearchStr)))) {

     // Если управление перешло в эту точку, то обнаружено
     // совпадение с искомой подстрокой
     m_dwFindCount++;

     // Вычислить позиции начала и конца строки, чтобы можно было
     // вывести ее на терминал
     szBol=FindLineStart(szStringPos);
     szEol=FindLineEnd(szStringPos);

     // Вычислить количество выводимых символов. Это значение в
     // дальнейшем будет использоваться для подготовки строки
     // формата printf
     if (szEol) {
        dwNumChars=szEol-szBol;
       if (szEol<(m_pIoCurrent->m_szBuf+
         m_pIoCurrent->m_dwBytesRead)-1)
         szStartPos=szEol+1;
       else szStartPos=NULL;
     }
     else {
        dwNumChars=MAXLINE_LEN;
       szStartPos=NULL;
     }

#if(_DEBUG)
     sprintf(szOffsetOutput,
             "Thread %08d: Offset: %010I64d %s ",
             GetCurrentThreadId(),
              dwlFilePos+
                (szStringPos-m_pIoCurrent->m_szBuf),
              m_szFileName);
#else
     sprintf(szOffsetOutput,
             "Offset: %010I64d %s ",
              dwlFilePos+
                (szStringPos-m_pIoCurrent->m_szBuf),
              m_szFileName);
#endif
      // Подготовить строку формата, которая ограничивает вывод
      // текущей строкой
      strcpy(szFmt,"%s %.");
      sprintf(szFmt+5,"%ds\n",dwNumChars);

      // Вывести текущую строку
     sprintf(szMsg,szFmt,
```

```
                szOffsetOutput,
                szBol);

        char *pszMsg=(char *)HeapAlloc(m_hOutputHeap,
                                       0,
                                       strlen(szMsg)+1);

         strcpy(pszMsg,szMsg);
         if (!QueueUserAPC(m_pOutputCallback,
                           m_hMainThread,
                            (DWORD)pszMsg))
          printf("Error queuing output\n");

        bRes=true;

      }
     }
    __except(EXCEPTION_EXECUTE_HANDLER)
     {
       // Уничтожить эту исключительную ситуацию; управление не должно
       // никогда передаваться в данную точку, если учесть, что в конце
       // буфера чтения гарантировано наличие нулевого символа
  #if(_DEBUG)

      // Предполагается, что возникла ошибка, связанная с нарушением
      // прав доступа, из-за выхода за конец буфера чтения; но
      // фактически может иметь место ошибка какого-то другого типа
       printf("Thread %08d reached end of buffer\n",
         GetCurrentThreadId());
  #endif
     }

    m_pIoCurrent->SetState(BUF_STATE_INACTIVE);

  }

  return bRes;
}
```

21. Наиболее важной отличительной особенностью этого кода является вызов функции `SpinToFindBuf` в начале кода метода `Search`. Именно таким образом в методе `Search` происходит получение буфера чтения, в котором затем осуществляется поиск подстрок, совпадающих с искомой.
22. Еще одной особенностью класса `CBufSearch`, заслуживающей внимания, является использование вызовов APC для передачи выходных данных на терминал. Предпринимая попытки предотвратить блокировку применяемых рабочих потоков на то время, пока они ожидают завершения ввода-вывода на терминал, мы передаем каждую строку с найденным совпадением в очередь главного потока с помощью вызова APC. Каждая выходная строка с найденным совпадением размещается в закрытой динамической области памяти (каждый экземпляр `CBufSearch` имеет свою собственную динамическую область памяти), затем указатель на эту строку передается в вызове функции `QueueUserAPC`. После того как в главном потоке закончится чтение файла и ожидание завершения работы рабочих потоков, в нем повторно вызывается функция `SleepEx` до тех пор,

пока не будут выполнены все вызовы APC, поставленные в очередь. В каждом стоящем в очереди вызове APC просто осуществляется вывод на терминал выходной строки, для которой перед этим было выделено место в динамической области памяти. Такая организация работы программы позволяет направить весь вывод на экран через главный поток и обеспечить обработку в рабочих потоках текущих запросов без перехода в состояние ожидания до тех пор, пока появится возможность вывода на терминал. Безусловно, что при этом требуется, чтобы все строки с найденными совпадениями на время записывались в память, поэтому данный метод будет далек от идеала, если количество строк с обнаруженными совпадениями чрезвычайно велико. Но в подобной ситуации можно создать отдельный поток, предназначенный для вывода на терминал, и сформировать цикл обработки сообщений таким образом, чтобы рабочие потоки могли передавать сообщения в этот выходной поток, когда в них возникнет необходимость отправить текст на терминал. Еще один вариант состоит в использовании порта завершения ввода-вывода для передачи в очередь выходных данных каждого потока по такому же принципу, который применялся для управления входной очередью. Пример использования такого метод показан в следующем упражнении.

После удаления из очереди всех поставленных в очередь вызовов APC главный поток уничтожает находящийся в нем список объектов `CBufSearch`, что влечет за собой также удаление закрытых динамических областей памяти данных объектов (содержащих выходные строки). Это позволяет гарантировать, что при переходе программы от одного файла к другому не остается непроизводительно расходуемая память.

23. После полного просмотра буфера метод `Search` передает этот буфер в очередь, снова обозначая его состояние как `BUF_STATE_INACTIVE`. Это позволяет главному потоку воспользоваться данным буфером, когда в очередной раз потребуется получить буфер для применения с функцией `ReadFile`.
24. После того как метод `Search` обнаруживает, что список буферов пуст, а из приложения поступает сигнал, что считан весь входной файл (для этого переменной экземпляра `CIoBuf::s_bTerminated` присваивается значение `TRUE`), этот метод выходит из своего цикла, что влечет за собой нормальный выход из потока.
25. Теперь рассмотрим класс `CIoBuf` (листинг 5.9). Как было описано выше, каждый объект `CIoBuf` инкапсулирует метод, предназначенный для выполнения операции ввода-вывода. Связанный список объектов `CIoBuf` играет роль своего рода очереди запросов на выполнение работы. До тех пор пока в очереди находятся буфера, для которых требуется проведение обработки, метод `Search` класса `CBufSearch` продолжат их поиск.

Листинг 5.9. Модуль исходного кода для класса `CIoBuf` (`iobuf.cpp`)

```
// iobuf.cpp. Класс, который реализует простой диспетчер буферов,
//            предназначенных для асинхронных операций чтения из файла

#include "iobuf.h"

bool CIoBuf::s_bOverlapped=true;
bool CIoBuf::s_bTerminated=false;
CIoBuf *CIoBuf::s_pIoFirst=NULL;
```

```
// Конструктор
CIoBuf::CIoBuf(CIoBuf * pIoBufNext,
                       HANDLE hPort,
                        DWORD dwBufSize)
{
  m_dwState=BUF_STATE_INACTIVE;

  m_dwBufSize=dwBufSize;
  m_szBuf=(char *)VirtualAlloc(NULL,
                                  m_dwBufSize,
                                  MEM_RESERVE
                                | MEM_COMMIT,
                                PAGE_READWRITE);
  m_pIoBufNext=pIoBufNext;
  m_hPort=hPort;

  ZeroMemory(&m_OverlappedIO,
                 sizeof(m_OverlappedIO));
  m_OverlappedIO.hEvent=
     CreateEvent(NULL,true,false,NULL);
}

// Деструктор
CIoBuf::~CIoBuf()
{
  VirtualFree(m_szBuf,0,MEM_RELEASE);
  CloseHandle(m_OverlappedIO.hEvent);
}

// Выполнять итерации в цикле до тех пор, пока не будет найден буфер,
// находящийся в состоянии dwOldState, перевести его с помощью
// неразрывной операции в состояние dwNewState и возвратить в
// вызывающую функцию
CIoBuf *CIoBuf::SpinToFindBuf(DWORD dwOldState,
                                  DWORD dwNewState)
{
  bool bWasTerminated;
  do {
    // Следить за появлением новых пакетов завершения ввода-вывода
    // и обрабатывать эти пакеты
     CheckForIoPacketAndSetState(BUF_STATE_READY);

    // Сохранить значение статуса завершения, прежде чем войти
    // в цикл поиска
    bWasTerminated=s_bTerminated;

    // Проходить в цикле по списку буферов. Если будет найден буфер,
    // находящийся в требуемом состоянии, перевести его в новое
    // состояние и возвратить в вызывающую функцию
    for (CIoBuf *pIoCurrent=this;
         NULL!=pIoCurrent;
        pIoCurrent=pIoCurrent->m_pIoBufNext) {
      if (dwOldState==
        (DWORD)InterlockedCompareExchange(
               (volatile long *)&pIoCurrent->m_dwState,
                dwNewState,
```

```
                dwOldState))
            return pIoCurrent;
        }

        // Если был получен сигнал о завершении до того, как начался цикл,
        // и не найдено ни одного буфера с подстрокой, совпадающей с
        // искомой, то выйти из функции
    } while (!bWasTerminated);
    return NULL;
}

// Задать состояние буфера с помощью неразрывной операции
void CIoBuf::SetState(DWORD dwNewState)
{
    InterlockedExchange((long *)&m_dwState,dwNewState);
}

// Возвратить данные о состоянии буфера
DWORD CIoBuf::GetState()
{
    return m_dwState;
}

// Вычислить текущую позицию в файле с помощью соответствующего
// элемента структуры OVERLAPPED
DWORDLONG CIoBuf::FilePos()
{
    return (m_OverlappedIO.OffsetHigh*MAXDWORD)+
        m_OverlappedIO.Offset;
}

// Проверить наличие необработанных пакетов завершения ввода-вывода.
// Обнаружив такой пакет, найти в списке буферов буфер с
// соответствующим указателем на структуру OVERLAPPED и задать для
// этого буфера значения считанных байтов и состояния
void CIoBuf::CheckForIoPacketAndSetState(DWORD dwNewState)
{
    if (!s_pIoFirst->s_bOverlapped) return;
    DWORD dwKey;
    DWORD dwBytesRead;
    LPOVERLAPPED pOverlappedIO;
    if (GetQueuedCompletionStatus(m_hPort,
                                  &dwBytesRead,
                                  &dwKey,
                                  &pOverlappedIO,
                                  1))
    {
        for (CIoBuf *pIoCurr=s_pIoFirst;
            NULL!=pIoCurr;
            pIoCurr=pIoCurr->m_pIoBufNext) {
            if (&pIoCurr->m_OverlappedIO==pOverlappedIO) {
                InterlockedExchange(
                    (long *)&pIoCurr->m_dwBytesRead,
                    dwBytesRead);
                pIoCurr->SetState(dwNewState);
                return;
```

```
        }
      }

      // Управление не должно никогда передаваться в данную точку
      assert(false);
  }

}
```

26. Подробное описание класса `CIoBuf` следует начать с объяснения того, для чего он вообще предусмотрен. В данном приложении для управления распараллеливанием работы, а также для обеспечения синхронизации рабочих потоков используется порт завершения ввода-вывода. Поэтому приложение необходимо спроектировать таким образом, чтобы каждый конкретный рабочий поток мог продолжать действовать и обрабатывать максимальное количество неудовлетворенных запросов на выполнение работы перед тем, как он перейдет в состояние ожидания или произойдет переключение на другой поток. В конечном итоге, именно в этом состоит весь смысл использования порта завершения ввода-вывода — он должен свести количество переключений контекста к минимуму и вместе с тем предоставить приемлемый уровень распараллеливания. Это, безусловно, означает, что в данном приложении не может применяться такая же модель, как в других примерах приложения `fstring`, в которых запросы на выполнение работы (в виде необработанных буферов чтения), по сути, были вложены в класс `CBufSearch`, который инкапсулировал также и сами рабочие потоки. Применявшийся до сих пор проект не позволял предоставлять всем рабочим потокам доступ ко всем запросам на выполнение работы, поскольку существовало неявное взаимно-однозначное соответствие между рабочими потоками и запросами на выполнение работы. Для обработки нового запроса на выполнение работы требовалось запланировать для выполнения содержащий этот запрос рабочий поток и предоставить ему возможность провести поиск в его собственном буфере. С другой стороны, для полной реализации возможностей порта завершения ввода-вывода необходимо иметь средства, позволяющие поместить каждый буфер чтения и сопровождающую его информацию в глобальную очередь, к которой могли бы иметь доступ все рабочие потоки, и изымать из нее буфера по мере необходимости. В предыдущих примерах приложения `fstring` класс `CBufSearch` содержит буфер чтения и структуру `OVERLAPPED`, используемую в функции `ReadFile` API-интерфейса Windows. А в данном приложении эти члены класса `CBufSearch` перенесены в класс `CIoBuf`, кроме того, к классу `CIoBuf` добавлен новый член, который показывает состояние данного конкретного буфера. Связанный список объектов `CIoBuf` служит в качестве очереди запросов на выполнение работы, из которой каждый рабочий поток выбирает буфера, считанные из файла и подготовленные для проведения в них поиска. Поскольку информация буфера больше не хранится вместе с тем классом, в котором функционирует сам рабочий поток, все рабочие потоки могут получать доступ ко всем буферам чтения и извлекать их из очереди один за другим для осуществления в них поиска.

    Еще одним подходом к созданию проекта приложения, в котором структура `OVERLAPPED`, необходимая для асинхронного ввода-вывода, перенесена в другой объект, могло бы быть создание класса `CIoBuf` на основе данной

структуры. Это позволило бы передавать указатель на экземпляр CIoBuf в те функции, в которых требуется указатель на структуру OVERLAPPED (например, ReadFile), и исключить необходимость в использовании кода поиска в методе CheckForIoPacketAndSetState (описанном ниже). Пример применения такого подхода описан в следующем упражнении.

27. Для того чтобы поставить буфер в очередь или изъять его из очереди, в потоке просто изменяют значение переменной, в которой указано его состояние. Присваивание переменной состояния буфера значения BUF_STATE_INACTIVE или BUF_STATE_READY равносильно постановке его в очередь, а присваивание этой переменной значения BUF_STATE_READING или BUF_STATE_SEARCHING равносильно изъятию его из очереди. В этом и состоит назначение вызова функции SpinToFindBuf — эта функция отыскивает в списке буферов такой буфер, состояние которого соответствует затребованному состоянию (dwOldState) и изменяет его на указанное состояние (dwNewState). Соответствующее изменение состояния сводится либо к постановке буфера в очередь, либо к изъятию его из очереди, в зависимости от нового состояния и потенциального пользователя этого буфера. Функция SpinToFindBuf изменяет состояние буфера с помощью неразрывной операции в целях предотвращения коллизий между потоком, в котором она применяется, и другими потоками. В рабочих потоках функция SpinToFindBuf используется для получения буферов, в которых должен быть выполнен поиск, а в главном потоке эта функция применяется для получения буферов, которые требуются для функции ReadFile.

28. Метод SpinToFindBuf выполняет итерации (проходит по циклу), пока не обнаруживает буфер, находящийся в требуемом состоянии. В случае метода CBufSearch::Search требуемым состоянием всегда служит BUF_STATE_READY. После обнаружения соответствующего буфера в функции SpinToFindBuf используется функция InterlockedCompareExchange для изменения состояния буфера на dwNewState, что позволяет исключить возможность получения того же буфера другими потоками. После этого функция SpinToFindBuf возвращает адрес соответствующего объекта CIoBuf вызывающей функции.

29. Следует отметить, что на входе в функцию SpinToFindBuf вызывается функция CheckForIoPacketAndSetState. Она представляет собой метод, в котором фактически вызывается функция GetQueuedCompletionStatus. В этом методе происходит проверка наличия необработанного пакета завершения ввода-вывода, и после обнаружения такого пакета используется указатель OVERLAPPED, возвращенный функцией GetQueuedCompletionStatus для поиска объекта CIoBuf, который соответствует первоначальной операции асинхронного ввода-вывода. После этого в данном методе переменной состояния заданного объекта присваивается значение BUF_STATE_READY, обозначающее, что операция асинхронного чтения, в которой используется этот буфер, теперь завершена. Кроме того, данный метод присваивает значение переменной экземпляра, содержащей данные о количестве считанных байтов, для того, чтобы метод CBufSearch::Search имел информацию о том, в каком объеме буфера должен быть выполнен поиск (это предусмотрено на тот случай, если во время заполнения буфера будет обнаружен конец файла и буфер окажется неполным).

30. До начала выполнения цикла просмотра очереди функция SpinToFindBuf проверяет, был ли передан главным потоком сигнал о том, что в нем завершено чтение входного файла. После этого в функции SpinToFindBuf осуществ-

ляется поиск буфера, находящегося в требуемом состоянии, в списке буферов. Если эта функция не обнаруживает такой буфер, а перед началом выполнения цикла поиска получен сигнал о завершении чтения входного файла, осуществляется выход из данной функции. Такая организация работы позволяет предотвратить ситуацию, в которой завершение работы функции могло бы происходить в ходе выполнения цикла, до того как будут найдены все требуемые буфера, и вместе с тем позволяет определить с помощью данного объекта, было ли выполнено чтение всего файла, после которого в списке буферов должны находиться все необработанные запросы на выполнение работы. Нельзя отрицать, что какая-то операция асинхронного ввода-вывода может потребовать для своего завершения так много времени, что очередь буферов станет пустой после присваивания переменной `s_bTerminated` значения `TRUE`, а это приведет к преждевременному прекращению опроса буферов для передачи их рабочим потокам; тем не менее, применяемые в данном приложении объемы операций ввода-вывода настолько малы, что такая ситуация весьма маловероятна, поэтому автор решил не учитывать ее в данном примере приложения.

В следующем упражнении рассматривается такой вариант приложения `fstring_io_comp`, в котором порт завершения ввода-вывода используется для создания общей очереди выходных данных для всех рабочих потоков. Этот способ организации программы, как и предыдущий, позволяет предотвратить возможность того, что поток `CBufSearch`, начавший активно обрабатывать пакеты завершения ввода-вывода, перейдет в заблокированное состояние, ожидая завершения ввода-вывода, или перестанет получать доступ к ресурсам процессора по каким-то другим причинам, если в этом не будет абсолютной необходимости.

Упражнение 5.6. Утилита поиска в файлах, в которой используются порты завершения ввода-вывода и для ввода, и для вывода

1. Загрузите пример приложения `fstring_io_comp_out` из подкаталога `CH05\fstring_io_comp_out` компакт-диска, прилагаемого к этой книге, в среду разработки Visual Studio.
2. Основная часть кода приложения `fstring_io_comp_out` идентична той, которая была описана в предыдущем упражнении; единственным модулем, который подвергся существенным изменениям, является `bufsrch.cpp`. Рассмотрим этот модуль (листинг 5.10).

Листинг 5.10. Версия класса `CBufSearch`, в которой для передачи в очередь выходных данных используется порт завершения ввода-вывода

```
// bufsrch.cpp. Вспомогательный класс, который используется для поиска
//              подстроки в буфере

#include "bufsrch.h"

CIoBuf *CBufSearch::s_pIoFirst=NULL;

// Конструктор
CBufSearch::CBufSearch(CBufSearch *pbNext,
```

```
                                        char *szFileName,
                                        char *szSearchStr)
{

  // Записать в кэш параметры конструктора для дальнейшего
  // использования
  m_pbNext=pbNext;
  m_szFileName=szFileName;
  m_szSearchStr=szSearchStr;

  // Инициализировать все остальные переменные экземпляра
  m_dwFindCount=0;

// Создать закрытую динамическую область памяти, которая будет
// использоваться для вывода
  m_hOutputHeap=HeapCreate(HEAP_NO_SERIALIZE,0x1000,0);

  // Создать порт завершения ввода-вывода, который будет использоваться
  // для постановки в очередь запросов на выполнение операций вывода
  m_hOutputIoCompletionPort=
    CreateIoCompletionPort(INVALID_HANDLE_VALUE,NULL,0,0);

}

CBufSearch::~CBufSearch()
{

  // Удалить пакеты завершения ввода-вывода из выходной очереди и
  // вывести выходные данные
  DWORD dwLineCount=0;
  DWORD dwBytesWritten;
  DWORD dwKey;
  OUTPUT_OVERLAPPED *pOutputOverlapped;
  while (dwLineCount<m_dwFindCount)  {
    GetQueuedCompletionStatus(m_hOutputIoCompletionPort,
                              &dwBytesWritten,
                              &dwKey,
                              (OVERLAPPED **)
                                &pOutputOverlapped,INFINITE);
    printf(pOutputOverlapped->pszMsg);
    dwLineCount++;
  }

  // Уничтожить закрытую динамическую область памяти, предназначенную
  // для вывода
  HeapDestroy(m_hOutputHeap);

  // Закрыть выходной порт завершения ввода-вывода
  CloseHandle(m_hOutputIoCompletionPort);
}

// Найти начало строки по данным о смещении в буфере
char *CBufSearch::FindLineStart(char *szStartPos)
{
  char *szStart;
  for (szStart=szStartPos;
```

```
((szStart>m_pIoCurrent->m_szBuf) &&
      (cLINE_DELIM!=*(szStart-1))); szStart--);
  return szStart;
}

// Найти конец строки по данным о смещении в буфере; предполагается,
// что строка завершается нулевым символом
char *CBufSearch::FindLineEnd(char *szStartPos)
{
  return strchr(szStartPos,cLINE_DELIM);
}

// Провести поиск в буфере чтения, чтобы найти каждую строку,
// содержащую ранее заданную искомую подстроку
bool CBufSearch::Search()
{
  char *szBol;
  char *szEol;
  char *szStringPos;
  DWORD dwNumChars;
  char *szStartPos;
  bool bRes=false;
  char szFmt[32];
  char szOffsetOutput[255];
  DWORDLONG dwlFilePos;
  char szMsg[1024];

  while (1) {

    __try
    {

      // Выполнять итерации цикла до тех пор, пока не будет найден
      // буфер, предназначенный для проведения поиска
      if ((NULL==(m_pIoCurrent=
        s_pIoFirst->SpinToFindBuf(BUF_STATE_READY,
              BUF_STATE_SEARCHING))) &&
              (s_pIoFirst->s_bTerminated))
        break;

      // Поиск начинается с начала буфера чтения
      szStartPos=m_pIoCurrent->m_szBuf;

      // Получить данные о смещении в файле из объекта буфера для
      // последующего вывода данных на терминал
      dwlFilePos=m_pIoCurrent->FilePos();

      // Продолжать выполнение итераций цикла до тех пор, пока маркер
      // начала поиска не равен NULL, находится в пределах буфера
      // чтения, а функция strstr продолжает находить в буфере чтения
      // искомую подстроку
      while ((szStartPos) &&
              (szStartPos<(m_pIoCurrent->m_szBuf+
                        m_pIoCurrent->m_dwBytesRead)-1) &&
               (NULL!=(szStringPos=
                  strstr(szStartPos,m_szSearchStr)))) {
```

```
        // Если управление перешло в эту точку, то обнаружено
        // совпадение с искомой подстрокой
        m_dwFindCount++;

        // Вычислить позиции начала и конца строки, чтобы можно было
        // вывести ее на терминал
        szBol=FindLineStart(szStringPos);
        szEol=FindLineEnd(szStringPos);

        // Вычислить количество выводимых символов. Это значение в
        // дальнейшем будет использоваться для подготовки строки
        // формата printf
        if (szEol) {
          dwNumChars=szEol-szBol;
          if (szEol<(m_pIoCurrent->m_szBuf+
                m_pIoCurrent->m_dwBytesRead)-1)
                 szStartPos=szEol+1;
          else szStartPos=NULL;
        }
        else {
          dwNumChars=MAXLINE_LEN;
          szStartPos=NULL;
        }

    #if(_DEBUG)
    sprintf(szOffsetOutput,"Thread %08d: Offset: %010I64d %s ",
                GetCurrentThreadId(),
                dwlFilePos+(szStringPos-m_pIoCurrent->m_szBuf),
                m_szFileName);
    #else
sprintf(szOffsetOutput,"Offset: %010I64d %s ",
                dwlFilePos+(szStringPos-m_pIoCurrent->m_szBuf),
                m_szFileName);
    #endif
      // Подготовить строку формата, которая ограничивает вывод
      // текущей строкой
        strcpy(szFmt,"%s %.");
        sprintf(szFmt+5,"%ds\n",dwNumChars);

        // Сформировать выходную строку
        sprintf(szMsg,szFmt,
                    szOffsetOutput,
                    szBol);

      // Сформировать структуру, которая должна служить в качестве
      // выходного пакета
        OUTPUT_OVERLAPPED *pOutputOverlapped=
        (OUTPUT_OVERLAPPED *)
              HeapAlloc(m_hOutputHeap,
              0,
              sizeof(OUTPUT_OVERLAPPED));

        // Распределить память для строки сообщения в структуре
        // выходного пакета
        pOutputOverlapped->pszMsg=
          (char *)HeapAlloc(m_hOutputHeap,
```

```
                                0,
                                strlen(szMsg)+1);

        // Скопировать выходное сообщение в выходной пакет
        strcpy(pOutputOverlapped->pszMsg,szMsg);

        // Поставить выходной пакет в очередь

        PostQueuedCompletionStatus(
                    m_hOutputIoCompletionPort,
                    0,
                    0,(OVERLAPPED *)pOutputOverlapped);

        bRes=true;

      }
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
{
      // Уничтожить эту исключительную ситуацию; управление не должно
      // никогда передаваться в данную точку, если учесть, что в конце
      // буфера чтения гарантировано наличие нулевого символа
  #if(_DEBUG)

      // Предполагается, что возникла ошибка, связанная с нарушением
      // прав доступа, из-за выхода за конец буфера чтения; но
      // фактически может иметь место ошибка какого-то другого типа
       printf("Thread %08d reached end of buffer\n",
        GetCurrentThreadId());
  #endif
     }

    m_pIoCurrent-> SetState(BUF_STATE_INACTIVE);

  }

  return bRes;
}
```

3. Начнем с изучения конструктора `CBufSearch`. Заслуживает внимания то, что он теперь создает порт завершения ввода-вывода, предназначенный для исключительного использования в объекте `CBufSearch`, и что этот порт завершения не связан с файлом. Следует также отметить, что больше не предусмотрено получение дескриптора главного потока или адреса обратного вызова. Этого не требуется, поскольку выходные данные больше не передаются в очередь главного потока с использованием вызова APC, как было предусмотрено в приложении `fstring_io_comp`.

4. Рассмотрим часть метода `CBufSearch::Search`, в котором формируется выходная строка после обнаружения совпадения. В этом коде метода распределяется память для структуры `OUTPUT_OVERLAPPED` и в нее копируется выходная строка. Структура `OUTPUT_OVERLAPPED` является производной от структуры `OVERLAPPED` и определена в файле заголовка `bufsrch.h` следующим образом:

```
struct OUTPUT_OVERLAPPED: OVERLAPPED
{
  char *pszMsg;
};
```

Это означает, что данная структура включает все элементы структуры OVERLAPPED и, кроме того, один дополнительный указатель на строку. А поскольку данная структура является производной от структуры OVERLAPPED, то ее можно передавать в такие функции, которым требуется указатель на структуру OVERLAPPED, как PostQueuedCompletionStatus и GetQueuedCompletionStatus. После того как в коде этих процедур тип полученного в них указателя приводится к типу указателя на структуру OVERLAPPED и осуществляется его разадресация, заданные процедуры получают возможность обращаться к нужным им элементам, которые находятся в тех местах, где они и должны быть. В данном примере указанный способ переопределения структуры используется для того, чтобы можно было передать наряду со структурой OVERLAPPED некоторые дополнительные данные, а в дальнейшем осуществить выборку этих данных. Структура OUTPUT_OVERLAPPED включает указатель на выходной пакет. Она позволяет поставить выходные данные метода CBufSearch в очередь в то время, как в рабочем потоке происходит поиск, и отложить выполнение ввода-вывода на терминал на более позднее время.

5. После распределения структуры OUTPUT_OVERLAPPED и копирования в нее указателя на выходную строку эта структура передается в выходной порт завершения ввода-вывода с помощью функции PostQueuedCompletionStatus. Напомним, что функция PostQueuedCompletionStatus может использоваться в приложении для отправки в очередь его собственных пакетов завершения ввода-вывода специального назначения. Именно для этой цели указанная функция применяется и в данном приложении. В порт завершения ввода-вывода передается выходная информация, выборка которой в дальнейшем будет осуществляться с помощью функции GetQueuedCompletionStatus.

6. Прежде чем завершить описание данного примера, рассмотрим применяемый в нем деструктор, ~CBufSearch. Он обеспечивает запись всех выходных данных из очереди вывода на терминал и в конечном итоге освобождает все ресурсы, распределенные данным объектом, включая все закрытые динамические области памяти, содержащие эти выходные данные. Работа деструктора начинается с выполнения цикла, в котором неоднократно вызывается функция GetQueuedCompletionStatus для изъятия из очереди пакетов завершения ввода-вывода. После вывода из очереди каждого пакета в этом деструкторе берется указатель на структуру OVERLAPPED, полученный в составе параметров вызова, тип этой структуры приводится к типу структуры OUTPUT_OVERLAPPED, после чего выходная строка, на которую ссылается этот указатель, выводится на терминал. Поскольку с каждым результатом обнаружения совпадения должна быть связана одна строка вывода, такие действия продолжаются до тех пор, пока количество выбранных из очереди пакетов завершения ввода-вывода не станет равным количеству обнаруженных совпадений с искомой подстрокой.

У читателя может возникнуть вопрос, почему в этом приложении нельзя было просто открыть дескриптор терминала с использованием параметра FILE_FLAG_OVERLAPPED и выполнять операции асинхронной записи на терминал из метода CBufSearch::Search. Причина этого состоит в том, что весь вывод на терминал осуществляется синхронно. В приложении нельзя выводить данные на терминал асинхронно. Безусловно,

в приложении можно открыть дескриптор нового файла для вывода на терминал (используя специальную строку CONOUT$ в качестве имени файла), но все операции записи на терминал будут выполняться в синхронном режиме. Любая попытка записи на терминал с помощью операции WriteFileEx будет оканчиваться неудачей, а функция GetLastError будет сообщать, что дескриптор файла является недействительным. (Фактически в данном случае дескриптор файла является действительным, но становится недействительным при его использовании для асинхронного ввода-вывода.) Операционная система Windows синхронизирует все операции ввода-вывода на терминал, что позволяет упростить разработку многопотоковых приложений, поэтому и в настоящем приложении нельзя предусмотреть применение асинхронного вывода полученных результатов (непосредственно) на терминал. Именно поэтому в данном приложении предпринята попытка использовать вызов функции APC и порт завершения ввода-вывода, чтобы исключить необходимость перехода рабочих потоков в состояние ожидания завершения ввода-вывода в ходе осуществления поиска с помощью этих потоков.

Выше были описаны два практически применимых приложения, в которых порты завершения ввода-вывода используются для управления распараллеливанием работы, обеспечивают синхронизацию потоков и служат в качестве механизмов ведения очередей. Количество типов задач, для которых порты завершения ввода-вывода могут предоставить готовое решение, весьма велико. Порты завершения ввода-вывода применяются также в программе SQL Server, поэтому важно понять, как они работают и как могут использоваться в приложениях.

Порты завершения ввода-вывода. Резюме

Порт завершения ввода-вывода предоставляет эффективный механизм, с помощью которого можно обеспечить ожидание завершения операции асинхронного ввода-вывода для многочисленных потоков. Этот механизм может также использоваться в качестве механизма передачи сигналов общего назначения, который не связан с файлами и файловым вводом-выводом. Реальные перспективы применения портов завершения ввода-вывода состоят в том, что они позволяют управлять распараллеливанием работы в приложении, активно помогая приложению поддерживать нагрузку процессоров на максимально возможном высоком уровне за счет выполнения кода приложения, а не переключения контекста.

Порты завершения ввода-вывода. Вопросы для самопроверки

1. На основании чего по умолчанию происходит принятие решения о том, скольким потокам будет разрешено активно обрабатывать пакеты завершения ввода-вывода в порту завершения ввода-вывода?
2. Подтвердите или опровергните следующее утверждение. Характерной особенностью высокоэффективного серверного приложения является то, что в нем предпринимаются попытки уменьшить количество переключений контекстов среди рабочих потоков в максимально возможной степени и вместе с тем поддерживать достаточную степень распараллеливания.

3. В каком состоянии должен находиться поток, чтобы можно было выполнить вызов APC непривилегированного режима?
4. Что произойдет, если при вызове функции `GetQueuedCompletionStatus` будет указано значение тайм-аута 0, а пакет завершения ввода-вывода, ждущий обработки, будет отсутствовать?
5. Каково максимальное количество портов завершения, с которыми поток может быть ассоциирован одновременно?
6. Подтвердите или опровергните следующее утверждение. Пакеты удаляются из очереди порта завершения ввода-вывода в порядке последовательной очереди (первым удаляется пакет, поступивший первым).
7. Какая функция API-интерфейса используется для создания порта завершения ввода-вывода?
8. Возможно ли создать порт завершения ввода-вывода, который не связан с файлом?
9. Подтвердите или опровергните следующее утверждение. Чтобы обеспечить ожидание, сигналом об окончании которого служит поступление пакета завершения в очередь порта завершения ввода-вывода, в потоке вызывается одна из функций ожидания Win32, и ей передается дескриптор порта завершения ввода-вывода.
10. На что ссылается указатель на структуру `OVERLAPPED`, возвращаемый функцией `GetQueuedCompletionStatus`?
11. Как можно определить в приложении количество байтов, переданных с помощью асинхронной операции ввода-вывода, которая была инициализирована в результате применения вызова функции `ReadFile` к файлу, ассоциированному с портом завершения ввода-вывода?
12. Подтвердите или опровергните следующее утверждение. В планировщике непривилегированного режима (User Mode Scheduler) программы SQL Server предпринимаются попытки максимизировать загрузку процессора путем предотвращения переключений контекста в наибольшей возможной степени.
13. Опишите назначение функции `InterlockedExchange` API-интерфейса Win32.
14. Подтвердите или опровергните следующее утверждение. При работе с портами завершения ввода-вывода обычно предпочтительно не привязывать запросы на выполнение работы к конкретным рабочим потокам, чтобы любой запрос на выполнение работы мог быть обработан с помощью любого потока.
15. Какая распространенная ситуация часто возникает в тех приложениях, где допускается одновременное функционирование слишком большого количества рабочих потоков?
16. Какая функция API-интерфейса Win32 может применяться для отправки пакета завершения ввода-вывода специального назначения в порт завершения ввода-вывода?

17. Подтвердите или опровергните следующее утверждение. По возможности в потоках, в которых активно обрабатываются пакеты завершения ввода-вывода, следует избегать применения операций, которые вынуждают эти потоки переходить в заблокированное состояние.
18. Можно ли использовать функцию `SleepEx` для удаления вызова APC из очереди по аналогии с тем, как для этой цели используется функция `WaitForSingleObjectEx`?
19. Подтвердите или опровергните следующее утверждение. После того как для порта завершения ввода-вывода устанавливается значение степени распараллеливания, система начинает следить за соблюдением требования о том, чтобы количество потоков, активно обрабатывающих пакеты завершения ввода-вывода, никогда не превышало указанное значение.
20. Какую функцию API-интерфейса Win32 можно вызвать в потоке, чтобы явно поставить некоторый вызов APC в очередь потока?
21. Подтвердите или опровергните следующее утверждение. Результатом вызова функции `GetQueuedCompletionStatus` становится то, что поток ассоциируется с определенным портом завершения ввода-вывода.
22. В чем состоит назначение функции `InterlockedCompareExchange` API-интерфейса Win32?
23. Может ли запрос на выполнение операции асинхронного ввода-вывода в файле, ассоциированном с портом завершения ввода-вывода, обрабатываться операционной системой Windows синхронно?
24. Объясните, почему подход к проектированию программного обеспечения, в котором предусматривается установка жесткого предела количества рабочих потоков в приложении, равного количеству процессоров в системе, является потенциальной причиной низкой эффективности созданного приложения.
25. Вызов функции `ReadFileEx` влечет за собой постановку в очередь объекта определенного типа после завершения асинхронной операции, инициализированной в результате этого вызова. К какому типу относится этот объект?

Ввод-вывод с помощью отображаемых на память файлов

В данном разделе завершается обсуждение средств ввода-вывода Windows. Последняя тема посвящена описанию того, как осуществляется обработка файлов на основе средств ввода-вывода с помощью отображаемых на память файлов. В этом разделе приведен ряд примеров приложений, в основе которых лежат сведения, приведенные выше в данной главе, но для поиска подстроки в текстовом файле используется ввод-вывод с помощью отображаемых на память файлов. Кроме того, в настоящем разделе рассматриваются не только темы, описанные выше в этой главе, но и темы, которые необходимы для изучения остальной части данной книги. Если

читатель еще не ознакомился с предыдущими разделами этой главы, рекомендуем их прочитать, прежде чем приступать к дальнейшему изучению данного материала. К тому же, читателю потребуется прочитать главу 4, если он этого еще не сделал.

Ввод-вывод с помощью отображаемых на память файлов. Основные термины и определения

- **Совместно используемая память.** Память, доступ к которой предоставляется многочисленным процессам, или память, распределенная в пространстве виртуальных адресов многочисленных процессов.
- **Отображаемый на память файл.** Файл на диске, который отображен на виртуальную память так, что он служит в качестве реальной памяти для данной виртуальной памяти.
- **Объект секции памяти.** Объект привилегированного режима, предназначенный для реализации совместно используемой памяти и отображаемых на память файлов.

Ввод-вывод с помощью отображаемых на память файлов. Основные функции API-интерфейсов

Основные функции API-интерфейсов для работы с отображаемыми на память файлами приведены в табл. 5.9.

Таблица 5.9. Основные функции API-интерфейсов для работы с отображаемыми на память файлами

Функция	Описание
`CreateFileMapping`	Создать объект отображения файла (объект секции памяти) для использования с общей памятью или с отображаемым на память файлом
`MapViewOfFile`	Отобразить представление файла на память так, чтобы этот файл служил в качестве реальной памяти для виртуальной. Файл может представлять собой файл на диске или системный файл подкачки
`FlushViewOfFile`	Записать модифицированные страницы из представления отображаемого файла на диск

Краткий обзор

Как было указано в главе 4, средства ввода-вывода с помощью отображаемых на память файлов операционной системы Windows позволяют выполнять ввод-вывод в файл с помощью операций чтения и записи в память. При этом вместо отображения диапазона адресов виртуальной памяти на реальную память, находящуюся в системном файле подкачки, применяется такой метод доступа, что сам файл становится реальной памятью, на которую отображается виртуальная память, используемая в отображаемом на память файле.

Потоки, получающие доступ к файлу, просто получают доступ к памяти, как если бы эта память представляла собой большой, непрерывный массив. В ходе осуществления доступа к этой памяти диспетчер памяти Windows осуществляет незаметно для пользователя страничный обмен, передавая страницы файла в физическую память и из физической памяти. После внесения каким-то потоком изменений в содержимое этой памяти диспетчер памяти записывает эти изменения в файл в результате обычного процесса страничного обмена.

Для поддержки средств подсистемы ввода-вывода с помощью отображаемых на память файлов операционной системы Windows совместно используются система ввода-вывода и диспетчер памяти. Поддержка этих средств является важной частью подсистемы ввода-вывода и широко применяется в самой операционной системе. Например, ввод-вывод с помощью отображаемых на память файлов используется диспетчером системного кэша для отображения файлов на виртуальную память и обеспечения лучших показателей времени отклика для приложений, производительность которых зависит от ввода-вывода. В большинстве систем кэширования для записи файлов в кэш распределяется ограниченный объем памяти, тогда как в операционной системе Windows для этой цели используется ввод-вывод с помощью отображаемых на память файлов, а это означает, что объем физической памяти, отведенной для кэширования файлов, может изменяться в зависимости от того, какие еще операции выполняются в системе. В случае увеличения потребления физической памяти буфер кэша уменьшается с учетом этого увеличенного потребления. Если же физическая память используется в относительно меньшей степени, то кэш может становиться весьма значительным и обеспечивать превосходную производительность даже при обработке очень больших файлов.

Еще один способ применения собственных средств ввода-вывода с помощью отображаемых на память файлов в операционной системе Windows касается активизации файлов образов. В то время как исполняемый файл или файл библиотеки DLL переносится в пространство адресов процесса, он загружается как отображаемый на память файл. А после того как операционной системе Windows потребуется доступ к конкретной странице кода или данных из этого исполняемого двоичного файла, происходит автоматическая загрузка этой страницы в физическую память с применением обычного процесса страничного обмена. В данном случае диапазон адресов виртуальной памяти, занятый исполняемым двоичным файлом, отображается на сам исполняемый файл или файл библиотеки DLL, а не на системный файл подкачки.

Для отображения файла на виртуальную память в приложении необходимо выполнить описанные ниже шаги.

1. Открыть файл с помощью вызова функции `CreateFile`.
2. Создать объект отображения файла на память (ему соответствует такой объект привилегированного режима, как объект секции памяти) с помощью вызова функции `CreateFileMapping`.
3. Передать дескриптор объекта отображения файла на память в функцию `MapViewOfFile`. Функция `MapViewOfFile` фактически осуществляет отображение файла на виртуальную память и возвращает указатель на начальный виртуальный адрес, с которого начинается отображение.

После отображения файла на виртуальную память доступ к нему может осуществляться так, как если бы он действительно был скопирован с диска в память. Преимущество отображения файла на память по сравнению с фактическим копированием его с диска состоит в том, что файл в действительности не копируется из его первоначального местонахождения в системный файл подкачки, поэтому его "загрузка" происходит чрезвычайно быстро и не расходуется физическая память, которая могла бы потребоваться для других целей.

Заслуживает внимания также то, что отображаемый на память файл занимает часть пространства адресов виртуальной памяти, поэтому на него распространяются такие же ограничения, как и на другие объекты, для которых распределяется виртуальная память. Если в пространстве памяти процесса отсутствует достаточный объем непрерывного пространства адресов для создания требуемого отображения, то попытка отображения файла на память оканчивается неудачей, так же как могут оканчиваться неудачей любые другие попытки резервирования виртуальной памяти, для которых требуется слишком много памяти. Кроме того, все пространство адресов непривилегированного режима не может иметь объем больше 3 Гбайт, поэтому невозможно полностью отобразить на память файл с размером больше 3 Гбайт. В этом случае можно предусмотреть отображение на память меньшего сегмента данного файла, но это не позволит получить доступ ко всему файлу как к одному последовательному буферу в памяти.

Упражнения

Рассмотрим более подробно, как осуществляется ввод-вывод с помощью отображаемого на память файла, на примере приложения, в котором он используется. В следующем упражнении представлено приложение, в котором ввод-вывод с помощью отображаемого на память файла применяется для поиска в текстовом файле.

Упражнение 5.7. Использование ввода-вывода с помощью отображаемого на память файла для выполнения поиска в файле

В данном упражнении рассматривается пример приложения, в котором ввод-вывод с помощью отображаемого на память файла используется для поиска указанной подстроки в файлах, имена которых соответствуют заданной маске. Это приложение представляет собой один из вариантов приведенного выше в данной главе примера приложения, в котором для выполнения аналогичного поиска применяется небуферизованный, асинхронный ввод-вывод.

1. Загрузите пример приложения `fndstr` из подкаталога `CH05\fndstr` компакт-диска, прилагаемого к данной книге, в среду разработки Visual Studio и откомпилируйте его.
2. Вызовите приложение на выполнение под управлением отладчика VC++, указав текстовый файл, в котором должен быть выполнен поиск, и искомую подстроку. Если в вашем распоряжении нет готового текстового файла, то вы можете найти на компакт-диске файл с именем `INPUT.TXT`, который может применяться для тестирования. Этот файл содержит несколько экземпляров строки "ABCDEF".

3. В ходе пошагового выполнения данного кода вы обнаружите, что процесс поиска в указанном файле фактически осуществляется с помощью одного вызова функции `strstr` библиотеки RTL языка C/C++. Поскольку создается впечатление, что весь файл загружен в непрерывный буфер памяти, в приложении можно легко осуществить поиск в этом файле с помощью функции `strstr`. Отпадает необходимость выполнять обработку файла в виде фрагментов, размеры которых определяются размером буфера, а также учитывать вероятность того, что подстроки в файле, совпадающие с искомой подстрокой, могут оказаться на границе буфера. В отличие от некоторых других примеров приложений, демонстрирующих использование средств ввода-вывода, приложение `fndstr` позволяет найти все вхождения искомой подстроки в файле, независимо от того, где они расположены физически.
4. Но следует отметить, что рассматриваемый файл отображается на виртуальную память, поэтому приходится учитывать ограничения, которые обусловлены использованием пространства адресов виртуальной памяти. Прежде всего, если система не способна найти непрерывную область адресов виртуальной памяти, которая достаточно велика для отображения в нее всего файла, то работа приведенного ниже кода оканчивается неудачей. Такая ситуация может возникнуть, если пространство адресов непривилегированного режима фрагментировано в результате выполнения других операций распределения или применения других отображений файлов на память. Более того, если файл превышает по размеру объем пространства непривилегированного режима (2 или 3 Гбайт в 32-битовых версиях Windows), попытка выполнить отображение также оканчивается неудачей. Таким образом, данный способ не может применяться для обработки чрезвычайно больших файлов или для обработки даже файлов умеренных размеров в тех ситуациях, когда может иметь место интенсивная фрагментация виртуальной памяти.
5. Как и в большинстве других примеров приложений, приведенных в данной книге, наилучший способ понять, как они работают, состоит в изучении самого кода. В листинге 5.11 показан основной модуль исходного кода для примера приложения `fndstr` (`fndstr.cpp`).

Листинг 5.11. Основной модуль исходного кода для утилиты `fndstr` (`fndstr.cpp`)

```
// fndstr.cpp. Утилита поиска в файле, в которой для чтения каждого
//             файла используется ввод-вывод на основе отображаемого
//             на память файла
//

#include "stdafx.h"
#include "windows.h"
#include "stdlib.h"

#define MAXLINE_LEN 0x1000
const char cLINE_DELIM='\n';

// Найти начало строки по данным о смещении в буфере
char *FindLineStart(char *szStartPos, char *szFileStart)
{
  char *szStart;
  for (szStart=szStartPos;
```

```
    ((szStart>szFileStart) && (cLINE_DELIM!=*(szStart-1)));
        szStart--)
;
  return szStart;
}

// Найти конец строки по данным о смещении в буфере; предполагается,
// что строка завершается нулевым символом
char *FindLineEnd(char *szStartPos)
{
return strchr(szStartPos,cLINE_DELIM);
}

// Выполнить поиск заданной подстроки в буфере
DWORD Search(char *szStart, char *szEnd, char *szSearchStr,
    char *szFileName)
{
  DWORD dwFindCount=0;

  char *szBol;
  char *szEol;
  char *szStringPos;
  DWORD dwNumChars;
  char *szStartPos=szStart;
  char szFmt[32];
  __try
  {
    while ((szStartPos) &&
          (szStartPos<szEnd) &&
         (NULL!=(szStringPos=strstr(szStartPos,szSearchStr)))) {

      dwFindCount++;

      szBol=FindLineStart(szStringPos, szStart);
      szEol=FindLineEnd(szStringPos);

      if (szEol) {
         dwNumChars=szEol-szBol;
        if (szEol<szEnd) szStartPos=szEol+1;
        else szStartPos=NULL;
       }
      else {
         dwNumChars=MAXLINE_LEN;
        szStartPos=NULL;
       }

      printf("%s Offset: %010d ",szFileName,
              szStringPos-szStart);

      // Подготовить строку формата, которая ограничивает вывод
      // текущей строкой

      strcpy(szFmt,"%.");
      sprintf(szFmt+2,"%ds\n",dwNumChars);
      // Вывести текущую строку
      printf(szFmt,szBol);
```

```
    }
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    // Уничтожить эту исключительную ситуацию
#if(_DEBUG)
    printf("Scanned past end of buffer\n");
#endif
  }

  if (!dwFindCount)
    printf("Not found\n");
  return dwFindCount;
}

// Выполнить поиск искомой подстроки во всем файле
DWORD SearchFile(char *szPath, char *szFileName,
   char *szSearchStr)
{
  char *szFileData;
  char szFullPathName[MAX_PATH+1];
  DWORD dwFindCount;

  strcpy(szFullPathName,szPath);
  strcat(szFullPathName,szFileName);

  // Открыть файл
  HANDLE hFile=CreateFile(szFullPathName,
                          GENERIC_READ,
                           FILE_SHARE_READ,
                          NULL,
                          OPEN_EXISTING,
                          FILE_ATTRIBUTE_NORMAL,
                          NULL);

  // Создать объект отображения файла на память для данного файла
  HANDLE hMappingObject=
    CreateFileMapping(hFile,
                      NULL,
                     PAGE_READONLY,
                      0,
                      0,
                     NULL);

  // Получить указатель на данные файла, выполнив отображение файла
  // на виртуальную память
  szFileData=
    (char *)MapViewOfFile(hMappingObject,
                          FILE_MAP_READ,
                          0,
                          0,
                         0);

  // Получить данные о размере отображенной области с помощью функции
  // VirtualQueryEx, чтобы были известны границы области поиска
```

```
  MEMORY_BASIC_INFORMATION mbi;
  VirtualQueryEx(GetCurrentProcess(),
                  szFileData,
                 &mbi,
                 sizeof(mbi));

  // Выполнить поиск в файле
  dwFindCount=Search(szFileData,
                     szFileData+mbi.RegionSize,
                    szSearchStr,
                     szFileName);

  // Отменить отображение файла
  UnmapViewOfFile(szFileData);

  // Закрыть дескрипторы объекта отображения и файла
  CloseHandle(hMappingObject);
  CloseHandle(hFile);

  return dwFindCount;
}

// Выполнить поиск заданной подстроки в файлах, имена которых
// соответствуют указанной маске
bool SearchFiles(char *szFileMask, char *szSearchStr)
{

  char szPath[MAX_PATH+1];
  char *p=strrchr(szFileMask,'\\');
  if (p) {
     strncpy(szPath,szFileMask,(p-szFileMask)+1);
     szPath[(p-szFileMask)+1]='\0';
}
  else
    GetCurrentDirectory(MAX_PATH,szPath);

  if ('\\'!=szPath[strlen(szPath)-1])
    strcat(szPath,"\\");

  printf("Searching for %s in %s\n\n",szSearchStr,szFileMask);

  WIN32_FIND_DATA fdFiles;
  HANDLE hFind=FindFirstFile(szFileMask,&fdFiles);

  if (INVALID_HANDLE_VALUE == hFind) {
    printf("No files match the specified mask\n");
     return false;
  }

  DWORD dwFindCount=0;
  do {
     dwFindCount+=
          SearchFile(szPath,fdFiles.cFileName,szSearchStr);
  } while (FindNextFile(hFind,&fdFiles));

  FindClose(hFind);
```

```
  printf("\nTotal hits for %s in %s:\t%d\n",szSearchStr,
         szFileMask,dwFindCount);
  return true;
}

int main(int argc, char* argv[])
{
  if (argc<3) {
    printf("Usage is: fndstr filemask searchstring\n");
    return 1;
  }

  return (!SearchFiles(argv[1], argv[2]));
}
```

6. Основные программные конструкции, обеспечивающие перебор в цикле файлов, имена которых соответствуют заданной маске, и вызов функции поиска, являются одинаковыми и в этом, и в других примерах приложений, предназначенных для поиска в файлах, которые описаны в данной книге. Поэтому автор не повторяет здесь утомительные подробности, связанные с их описанием. Чтобы узнать подробности о том, как работает процедура `SearchFiles`, обратитесь к разделу "Асинхронный и небуферизованный ввод-вывод" этой главы, где была первоначально представлена функция `SearchFiles` и подробно описана вся эта процедура. А для читателей с новаторским мышлением будет несложно взять алгоритмы поиска, описанные в примерах ввода-вывода данной главы (и реализованные с помощью функции `SearchFile` в каждом примере приложения), и оформить их таким образом, чтобы они реализовывали принцип проектирования Strategy (описанный в книге[1] Эриха Гаммы и его соавторов, посвященной описанию шаблонов проектирования) и были взаимозаменяемы. Сам автор не может привести описание этого шаблона проектирования в данной главе из-за недостатка времени и места, но его изучение было бы интересным упражнением для любознательных.
7. Начнем с рассмотрения глобальной функции `Search`, которая имеет довольно простую структуру. Эта функция получает начальный указатель и конечный указатель, после чего находит все экземпляры искомой подстроки, находящиеся между ними. Фактические действия по обнаружению каждого экземпляра подстроки осуществляются с помощью функции `strstr` библиотеки RTL языка C/C++. После обнаружения соответствия функция `Search` выводит строку, в которой найдено соответствие, переустанавливает позицию начала поиска в точку, находящуюся сразу после конца предыдущей строки, и продолжает поиск. После завершения поиска во всем буфере функция `Search` возвращает в вызывающую функцию данные о количестве найденных совпадений.
8. Здесь заслуживает внимания код обработки исключительных ситуаций, который используется для перехвата исключительных ситуаций, возникающих при выходе поиска с помощью функции `strstr` за пределы конца буфера в ходе поиска нулевого символа завершения. Поскольку невозможно дополнить файл

[1] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley, 1995.

нулевым символом в том виде, в каком файл представлен в виртуальной памяти, не внося в него физические изменения, то появляется возможность, что чтение будет происходить после конца буфера, если по стечению обстоятельств содержимое файла оканчивается точно на границе страницы системы. Поэтому, если предположить, что размер страницы системы равен 4 Кбайт, длина файла точно равна 4 Кбайт, а в конце файла не записан нулевой символ, то функция `strstr`, отыскивая конец строки, выйдет за пределы буфера. Если учесть то, что дополнительная страница не может быть закреплена непосредственно за концом отображаемой области, такое развитие ситуации, к сожалению, является неизбежным при использовании метода отображения файла на память. Если же конец файла не попадает на границу страницы, то можно по крайней мере рассчитывать на то, что оставшаяся часть его последней страницы будет заполнена нулями при первом доступе к этому файлу. Поэтому функция `strstr` обнаружит необходимый ей нулевой символ завершения независимо от содержимого самого файла. Однако, если дело обстоит иначе, и функция `strstr` предпримет попытку обратиться к незакрепленному пространству адресов, активизируется ошибка нарушения доступа. Если это произойдет, то применяемый в программе код структурированной обработки исключительных ситуаций отбросит возникшую исключительную ситуацию и позволит программе продолжить поиск других файлов. А если эта программа откомпилирована как отладочная версия, то приложение `fndstr` позволит узнать, что в ходе поиска, по-видимому, пройден конец буфера, поскольку будет выведено сообщение на терминал. В приложении `fndstr` такая исключительная ситуация игнорируется на основании предположения о том, что нарушение доступа из-за выхода функции `strstr` за пределы памяти, на которую отображен файл, является единственным типом исключительной ситуации, которая должна быть обнаружена в главном цикле поиска. Это достаточно безопасное предположение, даже несмотря на то, что могут также возникать непонятные исключительные ситуации.

9. Теперь рассмотрим саму процедуру `SearchFile`. Работа этой процедуры начинается с открытия файла и создания объекта отображения файла на память; это обязательные операции для ввода-вывода с помощью отображаемого на память файла. Затем в этой процедуре вызывается функция `MapViewOfFile` подсистемы Win32 для отображения файла на непрерывный диапазон адресов виртуальной памяти; указанная функция возвращает указатель на начало этого диапазона. Указатель, возвращенный функцией `MapViewOfFile`, используется как точка доступа к файлу. Этот указатель будет служить в качестве начального адреса для процедур поиска данного приложения.

10. После этого в процедуре `SearchFile` вызывается функция `VirtualQueryEx` для определения размера области, зарезервированной для отображения файла на память. Полученное при этом значение должно быть равно размеру файла, округленному в большую сторону до границы следующей страницы. Это значение используется для определения конца буфера поиска. Поскольку система заполняет нулями закрепленную страницу виртуальной памяти при первом доступе к ней, то можно быть уверенным в том, что процедуры поиска, основанные на использовании функции `strstr`, не обнаружат ложных совпадений вслед за концом файла из-за наличия остатков данных, которые могли быть не стерты в памяти после обработки предыдущего файла.

Таким образом, выше приведено исчерпывающее описание приложения `fndstr`. Благодаря применению ввода-вывода с помощью отображаемого на память файла само это приложение остается довольно простым, и его разработка не требует особых усилий, связанных с поиском в многочисленных буферах, а также позволяет исключить сложности при поиске подстроки, которая может оказаться на границе буфера.

У читателя может возникнуть вопрос, почему в этом приложении поиск в отображаемом на память файле не осуществляется параллельно, при том, что приведенное выше в данной главе приложение `fstring`, в котором использовался небуферизованный, асинхронный ввод-вывод, было многопотоковым. В конечном итоге, в настоящем приложении поиск в файле сводится просто к просмотру содержимого памяти, поэтому данную операцию можно выполнить, не заботясь о синхронизации многочисленных операций одновременного доступа со стороны нескольких потоков, поскольку в памяти осуществляется чтение, а не запись.

В следующем примере приложения рассматривается именно такая возможность. Поскольку в нем файл будет подразделяться на несколько частей для того, чтобы в этом файле можно было выполнить просмотр с помощью нескольких потоков, при разработке этого приложения мы снова столкнемся с такой вероятностью, что искомая подстрока пересечет границу между буферами. Тем не менее, поскольку файл целиком отображается в память и выглядит как один непрерывный буфер пространства адресов, эту проблему можно решить новаторским способом, не отказываясь от возможности просмотра файла с помощью нескольких потоков.

Упражнение 5.8. Программа многопотокового просмотра файла, в которой используется ввод-вывод с помощью отображаемого на память файла

1. Загрузите пример приложения `findstring` из подкаталога `CH05\findstring` компакт-диска в среду Visual Studio и откомпилируйте его.
2. Приложение `findstring` состоит из двух основных модулей исходного кода — `findstring.cpp` и `rngsrch.cpp`. В модуле `findstring.cpp` реализованы программные конструкции, необходимые для итерации по файлам, имена которых соответствуют заданной маске, и для вызова процедуры поиска, которая отыскивает в каждом файле указанную подстроку. Главная функция этого модуля, `SearchFiles`, напоминает соответствующие функции в других примерах данной главы, поэтому автор не будет снова повторять здесь ее описание. Для ознакомления с подробными сведениями о функции `SearchFiles` обратитесь к описанию приложения `fstring`, приведенному выше в этой главе, где она рассматривается более подробно.
3. Поиск в каждом файле происходит с помощью функции `SearchFile`. В этой функции каждый файл открывается с помощью функции `CreateFile`, для него создается объект отображения файла на память, затем этот объект отображается на память с помощью функции `MapViewOfFile`. Как и в предыдущем упражнении, указатель, возвращенный функцией `MapViewOfFile`, используется в качестве начального адреса для операции поиска.
4. Одним из параметров, передаваемых в функцию `SearchFile`, является количество процессоров в системе. Процедура `SearchFiles` определяет это количество при запуске программы и передает его в функцию `SearchFile`. После

этого функция SearchFile создает по одному рабочему потоку для каждого процессора в системе. Например, если используется двухпроцессорная система, то создаются два рабочих потока, а в четырехпроцессорной системе создаются четыре рабочих потока. Поскольку поиск осуществляется исключительно в виртуальной памяти, то не будет никакого выигрыша от создания большего количества потоков по сравнению с количеством процессоров.

5. Обратите внимание на то, какой способ применяется в функции SearchFile для проверки количества страниц в файле и уменьшения количества используемых потоков, если в файле меньше страниц, чем процессоров в системе. Выполнение такой корректировки позволяет гарантировать, что каждый поток получит по меньшей мере одну страницу памяти для осуществления в ней поиска.
6. Затем в функции SearchFile создается связный список объектов CRangeSearch и приостановленный рабочий поток, который соответствует каждому экземпляру. После создания очередного рабочего потока в функции SearchFile передается указатель на объект CRangeSearch в качестве определяемого пользователем параметра пустого указателя для функции _beginthreadex. После этого в функции точки входа потока, StartSearch, этот параметр снова приводится к типу указателя CRangeSearch, и вызывается его метод Search. После вызова метода Search выход из него не происходит до тех пор, пока поток не переходит в состояние готовности к завершению работы. Подробное описание класса CRangeSearch (реализованного в модуле rngsrch.cpp) приведено ниже.
7. Каждому объекту CRangeSearch передается начальное и конечное смещения, которые определяют область поиска. Итак, в силу самого того факта, что файл отображен на ряд непрерывных адресов виртуальной памяти, достаточно лишь определить пары смещений, чтобы организовать в нем поиск с помощью нескольких потоков. При наличии двух рабочих потоков каждый поток просматривает примерно половину файлов.
8. После создания всех рабочих потоков в функции SearchFile начинается их запуск с помощью вызова функции ResumeThread. Рабочие потоки первоначально создаются в приостановленном состоянии для того, чтобы в каждом объекте CRangeSearch можно было откорректировать путем повторных вычислений значение конца диапазона поиска еще до начала фактической операции поиска. Как было указано выше, область виртуальной памяти, на которую отображается файл, подразделяется на несколько логических частей, чтобы можно было выполнять поиск в них параллельно. Поэтому приходится снова сталкиваться с такой ситуацией, в которой искомая подстрока может пересечь границу буфера. С учетом этого при создании каждого объекта CRangeSearch значение передаваемого ему конечного смещения области поиска корректируется таким образом, чтобы оно совпадало с концом последней полной текстовой строки в этой области. Иными словами, если последним символом в буфере не является символ с обозначением конца строки, то конец буфера смещается в обратном направлении до тех пор, пока не будет найден последний символ обозначения конца строки в данном буфере. Это позволяет предотвратить ситуацию, в которой искомая подстрока будет пересекать границу буфера. Такая организация работы также требует, чтобы следующий объект CRangeSearch начинал свой поиск непосредственно вслед за этим последним символом обозначения кон-

ца строки в буфере, поэтому метод RecalcEnd класса CRangeSearch возвращает новое начальное смещение, которое передается в конструктор следующего объекта CRangeSearch, чтобы этот конструктор мог правильно установить начальную позицию поиска для указанного объекта.

9. После запуска всех потоков вызывается функция WaitForMultipleObjects для перехода в состояние ожидания завершения работы всех этих потоков. Как только все потоки завершают свою работу, подытоживаются полученные результаты, освобождаются распределенные ресурсы, а результаты поиска возвращаются в процедуру SearchFiles.

10. Работу описанной процедуры проще всего понять, рассматривая сам ее код. Код модуля findstring.cpp приведен в листинге 5.12.

Листинг 5.12. Главный модуль исходного кода для утилиты findstring (findstring.cpp)

```
// findstring.cpp. Многопотоковая процедура поиска в файле, в которой
//                 используется ввод-вывод на основе файла с
//                 отображением на память
//

#include "stdafx.h"
#include "windows.h"
#include "stdlib.h"
#include "process.h"
#include "rngsrch.h"

// Функция точки входа потока
unsigned __stdcall StartSearch(LPVOID lpParameter)
{
  return ((CRangeSearch*)lpParameter)->Search();
}

// Выполнить поиск в файле заданной подстроки
DWORD SearchFile(DWORD dwPageSize,
                 DWORD dwNumProcessors,
                 char *szPath,
                  char *szFileName,
                  char *szSearchStr)
{
  char *szFileData;
  char szFullPathName[MAX_PATH+1];
strcpy(szFullPathName,szPath);
   strcat(szFullPathName,szFileName);

   // Открыть файл
   HANDLE hFile=
    CreateFile(szFullPathName,
               GENERIC_READ,
              FILE_SHARE_READ,
              NULL,
               OPEN_EXISTING,
              FILE_ATTRIBUTE_NORMAL,
              NULL);
```

```
    // Создать объект отображения файла на память
    HANDLE hMappingObject=
     CreateFileMapping(hFile,
                       NULL,
                      PAGE_READONLY,
                       0,
                       0,
                       NULL);

    // Отобразить файл на память и возвратить указатель
    // на начало памяти
    szFileData=
     (char *)MapViewOfFile(hMappingObject,
                           FILE_MAP_READ,
                           0,
                           0,
                           0);

    // Определить размер отображенной области
    MEMORY_BASIC_INFORMATION mbi;
    VirtualQueryEx(GetCurrentProcess(),
                       szFileData,
                        &mbi,
                       sizeof(mbi));

    // Убедиться в том, что количество потоков не превышает
    // количества страниц
    DWORD dwNumThreads;
    DWORD dwNumPages=(mbi.RegionSize / dwPageSize);
if (dwNumProcessors>dwNumPages)
    dwNumThreads=dwNumPages;
    else
    dwNumThreads=dwNumProcessors;

    // Вычислить количество страниц, которые будут просматриваться
    // каждым потоком
    DWORD dwPagesPerThread=dwNumPages / dwNumThreads;

    // Распределить массив дескрипторов потоков
    HANDLE *hThreads=
              (HANDLE *)HeapAlloc(GetProcessHeap(),
                                  0,
                                  dwNumThreads*sizeof(HANDLE));
    if (NULL==hThreads) {
    printf("Error allocating worker thread array.  Aborting.\n");
    return 1;
    }

    CRangeSearch *prsFirst=NULL;
    char *szNextStartOfs=szFileData;
    char *szEndOfs=szFileData;
    unsigned uThreadId;

    // Распределить память для объектов CRangeSearch и создать
    // рабочие потоки
    for (DWORD i=0; i<dwNumThreads; i++) {
```

```
   if (i<dwNumThreads-1) {
      szEndOfs+=(dwPagesPerThread*dwPageSize)-1;
   }
    else szEndOfs=szFileData+mbi.RegionSize-1;

   prsFirst = new CRangeSearch(prsFirst,
                               szFileName,
                               szFileData,
                               szNextStartOfs,
                              szEndOfs,
                               szSearchStr);

   if (i<dwNumThreads-1)
     szNextStartOfs=prsFirst->RecalcEnd()+1;

   hThreads[i]=
(HANDLE)_beginthreadex(NULL,
                           0,
                           &StartSearch,
                           prsFirst,
                           CREATE_SUSPENDED,
                            &uThreadId);

 }

 // После создания всех объектов CRangeSearch выполнить запуск потоков
 for (i=0; i<dwNumThreads; i++)
   ResumeThread(hThreads[i]);

 // Ожидать завершения поиска в файле всеми рабочими потоками
 WaitForMultipleObjects(dwNumThreads,hThreads,true,INFINITE);

 // Получить общий итог и уничтожить объекты поиска
 DWORD dwFindCount=0;
 CRangeSearch *prsNext;
 for (; NULL!=prsFirst; prsFirst=prsNext) {
    dwFindCount+=prsFirst->m_dwFindCount;
   prsNext=prsFirst->m_prsNext;
   delete prsFirst;
 }

 // Освободить дескрипторы потоков
 for (i=0; i<dwNumThreads; i++)
   CloseHandle(hThreads[i]);

 // Освободить массив дескрипторов потоков
 HeapFree(GetProcessHeap(),0,hThreads);

 // Отменить отображение файла, а также закрыть дескрипторы объекта
 // отображения и файла
 UnmapViewOfFile(szFileData);
 CloseHandle(hMappingObject);
 CloseHandle(hFile);

 return dwFindCount;
```

```
}

bool SearchFiles(char *szFileMask, char *szSearchStr)
{
char szPath[MAX_PATH+1];
  char *p=strrchr(szFileMask,'\\');
  if (p) {
    strncpy(szPath,szFileMask,(p-szFileMask)+1);
     szPath[(p-szFileMask)+1]='\0';
  }
  else
    GetCurrentDirectory(MAX_PATH,szPath);

  if ('\\'!=szPath[strlen(szPath)-1])
    strcat(szPath,"\\");

  printf("Searching for %s in %s\n\n",szSearchStr,szFileMask);

  WIN32_FIND_DATA fdFiles;
  HANDLE hFind=FindFirstFile(szFileMask,&fdFiles);

  if (INVALID_HANDLE_VALUE == hFind) {
    printf("No files match the specified mask\n");
    return false;
  }

  SYSTEM_INFO si;
  GetSystemInfo(&si);

  DWORD dwFindCount=0;
  do {
    dwFindCount+=
       SearchFile(si.dwPageSize,
                  si.dwNumberOfProcessors,
                  szPath,
                  fdFiles.cFileName,
                   szSearchStr);

  } while (FindNextFile(hFind,&fdFiles));

  FindClose(hFind);

  printf("\nTotal hits for %s in %s:\t%d\n",szSearchStr,
           szFileMask,dwFindCount);
  return true;
}

int main(int argc, char* argv[])
{
  if (argc<3) {
    printf("Usage is: findstring filemask searchstring\n");
    return 1;
  }

  return (!SearchFiles(argv[1], argv[2]));
}
```

11. В действительности все действия по поиску в каждом логическом фрагменте файла осуществляются с помощью класса `CRangeSearch`. Он реализован в модуле `rngsrch.cpp`. Код указанного модуля приведен в листинге 5.13.

Листинг 5.13. Модуль исходного кода для класса `CRangeSearch` (`rngsrch.cpp`)

```
// rngsrch.cpp. Вспомогательный класс, который используется для поиска
//              заданной подстроки в области виртуальной памяти

#include "rngsrch.h"

// Конструктор
CRangeSearch::CRangeSearch(CRangeSearch *prsNext,
char *szFileName, char *szFileData, char *szStart, char *szEnd,
char* szSearchStr)
{
  // Записать в кэш параметры конструктора для дальнейшего
  // использования
  m_prsNext=prsNext;
  m_szFileName=szFileName;
  m_szFileData=szFileData;
  m_szStart=szStart;
  m_szEnd=szEnd;
  m_szSearchStr=szSearchStr;

  m_dwFindCount=0;
}

// Уточнить с помощью вычислений положение конца буфера поиска
// для того, чтобы ни одна строка не пересекала границу между
// двумя буферами
char *CRangeSearch::RecalcEnd()
{
m_szEnd=FindLineStart(m_szEnd);
  if (m_szEnd) m_szEnd--;
  return m_szEnd;
}

// Найти начало строки по данным о смещении в буфере
char *CRangeSearch::FindLineStart(char *szStartPos)
{
  char *szStart;
  for (szStart=szStartPos;
    ((szStart>m_szStart) && (cLINE_DELIM!=*(szStart-1)));
      szStart--);
  return szStart;
}
// Найти конец строки по данным о смещении в буфере; предполагается,
// что строка завершается нулевым символом
char *CRangeSearch::FindLineEnd(char *szStartPos)
{
  return strchr(szStartPos,cLINE_DELIM);
}

// Непрерывно выполнять поиск заданной подстроки в указанном буфере
```

```
bool CRangeSearch::Search()
{
  char *szBol;
  char *szEol;
  char *szStringPos;
  DWORD dwNumChars;
  char *szStartPos=m_szStart;
  bool bRes=false;
  char szFmt[32];
  char szOffsetMsg[255];

  __try
  {
    while ((szStartPos) &&
          (szStartPos<m_szEnd) &&
          (NULL!=(szStringPos=strstr(szStartPos,m_szSearchStr)))) {

      m_dwFindCount++;

      szBol=FindLineStart(szStringPos);
      szEol=FindLineEnd(szStringPos);
if (szEol) {
         dwNumChars=szEol-szBol;
        if (szEol<m_szEnd) szStartPos=szEol+1;
        else szStartPos=NULL;
      }
      else {
         dwNumChars=MAXLINE_LEN;
        szStartPos=NULL;
      }

  #if(_DEBUG)
  sprintf(szOffsetMsg,"Thread %08d: Offset: %010d %s ",
           GetCurrentThreadId(),
            szStringPos-m_szFileData,m_szFileName);
  #else
  sprintf(szOffsetMsg,"Offset: %010d %s ",
            szStringPos-m_szFileData,m_szFileName);
  #endif

    // Подготовить строку формата, которая ограничивает вывод текущей
    // строкой
      strcpy(szFmt,"%s %.");
      sprintf(szFmt+5,"%ds\n",dwNumChars);

      // Вывести текущую строку
      printf(szFmt,szOffsetMsg,szBol);

      bRes=true;

    }
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    // Уничтожить эту исключительную ситуацию
#if(_DEBUG)
    printf("Thread %08d reached end of buffer\n",
```

```
            GetCurrentThreadId());
#endif
  }

  if (!bRes)
    printf("Not found\n");
  return bRes;
}
```

12. Основным методом в классе `CRangeSearch` является его метод `Search`. Как было указано выше, после того как в рабочем потоке начинается выполнение кода данного метода, выход из этого кода не происходит до тех пор, пока поток не закончит просмотр области памяти, которая ему отведена для поиска.
13. В методе `Search` для поиска совпадений с искомой подстрокой в отведенном для него буфере используется простой цикл, основанный на вызовах функции `strstr` библиотеки RTL языка C/C++. После обнаружения каждого совпадения этот метод выводит строку, содержащую искомое соответствие, затем продолжает просмотр, начиная с конца текущей строки. После того как все совпадения будут найдены, поток оканчивает свою работу и осуществляет выход обычным образом.
14. Как и в предыдущем примере приложения, остается возможность того, что функция `strstr` в ходе поиска нулевого символа завершения выйдет за пределы конца диапазона виртуальных адресов, на который отображен файл. Вероятность возникновения такой ситуации значительно повышается, если по стечению обстоятельств конец файла точно попадает на границу страницы. Если функция `strstr` выйдет за пределы области, на которую отображен файл, то может быть активизирована ошибка нарушения доступа, поэтому в приложении перехватываются и уничтожаются любые исключительные ситуации, активизированные в цикле поиска. Еще раз отметим, что такая организация приложения принята на основе предположения, что единственной причиной исключительной ситуации, возникающей в данной процедуре, является выход функции `strstr` за пределы области, на которую отображен файл, даже несмотря на то, что не исключена вероятность возникновения некоторых других неопределенных условий, которые могут привести к активизации исключительных ситуаций в этом цикле. Дело в том, что приведенный здесь код не предназначен для демонстрации исчерпывающей или даже достаточно надежной обработки исключительных ситуаций; идея его состоит в том, чтобы показать рассматриваемые средства ввода-вывода на наиболее простом и вместе с тем работоспособном примере и дать читателю возможность получить представление о некоторых практических областях применения ввода-вывода с помощью отображаемого на память файла.

Читатель уже должен иметь определенное представление о том, что можно сделать в программе с помощью ввода-вывода на основе отображаемого на память файла и многопотоковой организации работы. Отображаемые на память файлы используются во многих компонентах самой программы SQL Server, поэтому знакомство с основами того, как ввод-вывод с помощью отображаемого на память файла может быть осуществлен в приложении, позволит читателю лучше разобраться в том, как эти средства применяются в программе SQL Server.

Ввод-вывод с помощью отображаемых на память файлов. Резюме

Средства совместного использования памяти операционной системы Windows могут применяться не только для упрощения обмена данными между процессами, но и для отображения файлов на виртуальную память в целях упрощения доступа к ним. Отображаемый на память файл выполняет роль реальной памяти для диапазона виртуальных адресов, на который он отображен, поэтому этот файл не требуется копировать с диска в системный файл подкачки. Операция чтения файла сводится просто к операции чтения памяти, а операция записи в файл – к операции изменения содержимого памяти. Поскольку создается впечатление, что файл загружен в единый, непрерывный буфер, появляется возможность выполнять такие операции обработки этого файла, которые в других обстоятельствах были бы неосуществимыми или гораздо более сложными. Пример применения функциональных возможностей указанного типа продемонстрирован в методе `RecalcEnd` из приложения `findstring`.

Ввод-вывод с помощью отображаемых на память файлов. Вопросы для самопроверки

1. Какая функция API-интерфейса Win32 осуществляет отображение файла на память и возвращает указатель на начальный адрес памяти?
2. Подтвердите или опровергните следующее утверждение. Операционная система Windows перемещает библиотеки DLL и другие двоичные образы, которые были отображены в пространство адресов процесса, чтобы освободить место для файла при осуществлении в программе попытки отобразить файл на память.
3. Каким самым простым способом можно модифицировать в приложении содержимое файла, отображаемого на память?
4. Подтвердите или опровергните следующее утверждение. Для того чтобы файл мог быть отображен на память, его объект файла должен быть создан с параметром `FILE_FLAG_MAPPED`.
5. Какая функция API-интерфейса Win32 используется для создания объекта отображения файла?
6. Подтвердите или опровергните следующее утверждение. На файл, отображаемый на память, распространяются такие же ограничения, как и на саму виртуальную память, поэтому в 32-битовой версии Windows на виртуальную память нельзя полностью отобразить файл, размер которого превышает 3 Гбайт.
7. Подтвердите или опровергните следующее утверждение. Средства ввода-вывода на основе отображаемых на память файлов операционной системы Windows реализованы благодаря совместному использованию системы ввода-вывода и диспетчера памяти.

8. Если на виртуальную память отображен некоторый файл, то в какой момент операционная система Windows копирует этот файл в системный файл подкачки?
9. Какая функция API-интерфейса Win32 может быть вызвана в потоке для немедленного сброса на диск модифицированных страниц отображаемого на память файла?
10. Какой объект привилегированного режима обеспечивает реализацию совместно используемой памяти?
11. Какая функция API-интерфейса Win32 использовалась в настоящей главе для определения точного размера области памяти, на которую отображен файл?
12. Подтвердите или опровергните следующее утверждение. Файл, который был отображен на виртуальную память, служит в качестве реальной памяти для виртуальной. Но для того, чтобы отобразить файл на память, требуется больше времени, чем для того, чтобы распределить буфер в памяти и скопировать файл с диска, поскольку диспетчер памяти Windows почти всегда осуществляет ввод-вывод в синхронном режиме.
13. Какая функция API-интерфейса Win32 используется для отмены отображения файла?
14. Подтвердите или опровергните следующее утверждение. Диапазон адресов, зарезервированный для отображаемого на память файла, берется из применяемой по умолчанию системной динамической области памяти.
15. Подтвердите или опровергните следующее утверждение. Одним из способов использования в операционной системе Windows ее собственного средства ввода-вывода на основе отображаемых на память файлов является активизация загрузочных файлов.

ГЛАВА 6

Основы организации сетей

Назначение сетевого программного обеспечения состоит в том, что оно принимает запрос клиента на получение доступа к ресурсу, выполняет этот запрос на удаленном компьютере, где находится требуемый ресурс, и возвращает результаты клиенту. До того как сетевые функциональные средства были встроены в операционные системы, такая задача была весьма сложной. Для предоставления простейших средств связи между компьютерами и обеспечения их максимально возможной согласованности использовались всевозможные малоприемлемые и промежуточные решения (например, резидентные программы; утилиты, зависящие от топологии; и т.д.). А с появлением сетевой поддержки, встроенной в операционную систему, средства обеспечения связи между компьютерами стали не только обычными и широко распространенными, но и просто необходимыми. В настоящее время даже происходит переход от основных средств сетевой связи к таким новейшим технологиям, как WiFi и Gigabit Ethernet, а сама межмашинная связь по сети стала такой привычной, как водопровод и электричество.

Краткий обзор

Организация сетей. Основные термины и определения

- **Именованный канал.** Протокол организации взаимодействия по сети с установлением логического соединения, основанный на протоколах SMB (Server Message Block – блок серверных сообщений) и NetBIOS.
- **Сокет.** Конечная точка межпроцессной связи через сетевую транспортную систему. В операционной системе Windows для установления соединений через сокеты и обмена данными используется API-интерфейс Winsock.
- **Дистанционный вызов процедур.** Дистанционный вызов процедур (Remote Procedure Call – RPC) – это API-интерфейс организации сетей, который позволяет воспользоваться интерфейсом уровня вызовов (Call-Level Interface – CLI) вместо традиционной модели сетевого программирования, основанной на выполнении операций ввода-вывода.
- **Приложение Winsock с установлением логического соединения.** Приложение, в котором используется потоковое, или надежное, соединение для обмена данными с использованием API-интерфейса Winsock.

- **Приложение Winsock без установления логического соединения.** Приложение, в котором используется дейтаграммное, или ненадежное, соединение для обмена данными с использованием API-интерфейса Winsock.
- **Преобразование имен.** Операция преобразования имени компьютера в сетевой адрес.
- **Стек сетевых протоколов.** Совокупность взаимосвязанного программного обеспечения, позволяющего приложениям взаимодействовать по сети.

Организация сетей. Основные функции API-интерфейсов

Основные функции API-интерфейса Win32, относящиеся к организации сетей, перечислены в табл. 6.1.

Таблица 6.1. Основные функции API-интерфейса Win32, относящиеся к организации сетей

Функция	Описание
`CreateNamedPipe`	Создать именованный канал
`ConnectNamedPipe`	Ожидать подключения клиента к именованному каналу
`WSAStartup`	Инициализировать библиотеку API-интерфейса Winsock
`WSASocket`	Создать новый сокет (эта функция применяется исключительно в продуктах Microsoft)
`socket`	Создать новый сокет (эта функция совместима с интерфейсом BSD Sockets)
`listen`	Перевести сокет в режим приема клиентских запросов на установление соединения
`accept`	Ожидать подключения клиента Winsock
`connect`	Подключиться к серверу Winsock
`send`	Передать данные в сокет Winsock
`recv`	Получить данные из сокета Winsock
`ReadFile`	Прочитать данные из ресурса, представленного дескриптором файла, включая сокеты и именованные каналы
`WriteFile`	Записать данные в ресурс, представленный дескриптором файла, включая сокеты и именованные каналы

По традиции программное обеспечение для организации взаимодействия по сети всегда формировалось по принципу ввода-вывода; это означает, что клиент обычно получает доступ к сетевым ресурсам с использованием вызовов функций стандартного API-интерфейса операционной системы. В операционной системе Windows любая операция доступа по сети инициализируется автоматически после того, как приложение запрашивает доступ к удаленному ресурсу. Система обнаруживает, что данный запрос относится к удаленному ресурсу и перенаправляет его в специальную программу — *редиректор*. Редиректор действует как своего

рода удаленная файловая система. Он передает запрос на выполнение операции в операционную систему удаленного компьютера, на котором находится этот ресурс, после чего данный запрос направляется на выполнение в соответствующий компонент операционной системы. После выполнения запроса удаленная система возвращает результаты по сети в редиректор, который затем передает их клиенту. В ходе этого в клиентской программе даже не требуется учитывать, что для выполнения данной операции потребовался доступ по сети, поскольку запрос клиента был перехвачен, перенаправлен и выполнен незаметно для самого клиента.

Для успешной организации сетевого взаимодействия необходимо, чтобы операционная система на одном компьютере обладала способностью определить, как обратиться к операционной системе на удаленном компьютере, содержащем требуемый ресурс, указав сетевой адрес данного компьютера, и какие протоколы связи можно использовать для взаимодействия с этим компьютером. Операция преобразования имени компьютера в сетевой адрес называется *преобразованием имен*, а операция определения совместимого набора протоколов связи называется *согласованием протоколов*.

После успешного выполнения операций преобразования имен и согласования протоколов сетевой запрос необходимо подготовить для передачи по сети, разделив его на пакеты, подходящие для передачи по физической сетевой среде. После того как запрос поступает на компьютер получателя, он должен быть снова собран из пакетов, проверен на полноту, декодирован и передан соответствующему компоненту операционной системы на выполнение. Вслед за тем как этот компонент операционной системы выполняет данный запрос, описанная последовательность операций должна быть выполнена снова, но со сменой "действующих лиц", чтобы полученные результаты были отправлены клиенту.

Эталонная модель OSI

Сетевое программное обеспечение можно подразделить на четыре основные категории: службы, API-интерфейсы, протоколы и драйверы сетевых адаптеров. Кроме того, сетевое программное обеспечение подразделяется на уровни, которые расположены друг над другом, образуя структуру, которую принято называть *стеком сетевых протоколов*. Компоненты операционной системы Windows, реализующие стек сетевых протоколов, примерно соответствуют эталонной модели взаимодействия открытых систем (Open Systems Interconnection — OSI), которая была впервые введена в 1974 году Международной организацией по стандартизации (International Organization for Standardization — ISO). Описание эталонной модели OSI можно получить с Web-узла ISO по адресу `http://www.iso.org`. Семь уровней этой модели показаны в табл. 6.2.

Следует учитывать, что это, скорее, концептуальная модель, чем конструкция, точно реализуемая поставщиками программного обеспечения. Она часто используется для абстрактного описания сетевого взаимодействия, а задача определения деталей реализации возлагается на поставщиков. Поэтому важно усвоить лишь основные особенности этой модели и понять, как она реализована в операционной системе Windows.

Таблица 6.2. Эталонная модель OSI

Номер уровня	Название	Описание
7	Прикладной уровень	Передача информации между двумя компьютерами в сетевом сеансе связи. На этом уровне осуществляются такие операции, как идентификация компьютера, проверка соблюдения требований защиты и инициализация обмена данными
6	Представительный уровень	Форматирование данных, включая определение типа используемых символов обозначения конца строки, применение сжатия данных, кодирования данных и т.д.
5	Сеансовый уровень	Непосредственное управление соединением, включая координацию действий, и определение того, кто должен передавать и принимать данные в каждый конкретный момент времени
4	Транспортный уровень	Разделение сообщений на пакеты и присваивание им порядковых номеров на стороне отправителя, а также последующая сборка пакетов, полученных от отправителя, и восстановление исходного сообщения на стороне получателя. Кроме того, на этом уровне создается абстрактное представление аппаратного уровня, чтобы можно было защитить сеансовый уровень от воздействия изменений на аппаратном уровне
3	Сетевой уровень	Формирование заголовков пакетов, а также маршрутизация пакетов, обеспечение межсетевого обмена и устранение заторов. Это – самый верхний уровень, на котором учитывается топология сети, т.е. физическая конфигурация сети (наличие других компьютеров в сети, ограничения по пропускной способности и т.д.)
2	Канальный уровень	Передача фреймов данных низкого уровня, проверка их успешного приема и повторная передача фреймов, потерянных или искаженных из-за недостаточной ненадежности линий связи
1	Физический уровень	Передача и прием данных через физическую сетевую передающую среду (сетевой кабель, беспроводные устройства и т.д.)

Назначение каждого уровня эталонной модели состоит в том, что он предоставляет услуги более высоким уровням и создает абстрактный интерфейс к услугам, предоставляемым более низкими уровнями. Если считать, что на физическом уровне происходит фактический процесс передачи битов по сетевой кабельной системе, то каждый следующий уровень, опирающийся на физический уровень, удобно рассматривать как постепенное повышение уровня абстракции. По мере перехода вверх по стеку протоколов мы все больше и больше уходим от проблем физической передачи данных, пока не достигаем прикладного уровня, на котором абсолютно не учитывается сам способ физической передачи данных между компьютерами.

Рассмотрим, как уровни стека на одном компьютере взаимодействуют с соответствующими уровнями стека на другом компьютере при обмене данными по сети. Можно считать, что каждый уровень на первом компьютере взаимодействует с тем же уровнем на другом компьютере, и на обоих уровнях используется один и тот же протокол. Например, приложение на одном компьютере, в котором

используется интерфейс Windows Sockets, действует так, как будто оно обменивается данными непосредственно с приложением Windows Sockets на другом компьютере. Однако в действительности данные должны спускаться по стеку сетевых протоколов каждого компьютера до уровня физической среды, затем передаваться по этой среде на другой компьютер и снова подниматься по стеку сетевых протоколов другого компьютера. Поэтому, хотя и создается впечатление, что происходит взаимодействие лишь между транспортными уровнями каждого компьютера, участвующего в обмене данными по сети, фактически такое взаимодействие происходит также между сетевыми, канальными и физическими уровнями, а непосредственная передача данных по сетевой кабельной системе осуществляется между физическими уровнями двух компьютеров.

В соответствии с общепринятым соглашением семь уровней эталонной модели дополнительно подразделяются на два более широких *яруса*. Нижние четыре уровня обычно рассматриваются как средства сетевого транспорта, а верхние три — как клиенты, или пользователи, средств сетевого транспорта. Изучение практических особенностей передачи данных с одного компьютера на другой начинается с транспортного уровня OSI, поэтому дополнительное разделение семи уровней по признаку принадлежности к средствам транспорта полезно и с концептуальной точки зрения.

Компоненты организации сетевого взаимодействия Windows

В операционной системе Windows, как и в других операционных системах, эталонная модель OSI не была реализована абсолютно точно. В этой операционной системе предусмотрено несколько уровней, которые отсутствуют в модели OSI, а некоторые ее уровни охватывают несколько уровней OSI. Соответствие между уровнями сетевых компонентов Windows и уровнями OSI показано в табл. 6.3.

Таблица 6.3. Сетевые компоненты Windows и их соответствие эталонной модели OSI

Номер уровня	Сетевой компонент Windows	Название уровня OSI	Примечание
7	Сетевое приложение	Прикладной	
6	Библиотека DLL сетевого API-интерфейса	Представительный	Позволяет приложениям взаимодействовать по сети в форме, независимой от транспортного протокола
5		Сеансовый	
5	Драйвер сетевого API-интерфейса	Сеансовый	Драйверы привилегированного режима, обеспечивающие реализацию той части сетевого API-интерфейса, которая относится к привилегированному режиму
	Интерфейс транспортного драйвера		Задает общий API-интерфейс для драйверов устройств привилегированного режима

Окончание табл. 6.3

Номер уровня	Сетевой компонент Windows	Название уровня OSI	Примечание
4	Драйверы протоколов (TCP/IP, IPX и т.д.)	Транспортный	
3		Сетевой	
	Драйверы протоколов NDIS		Драйверы привилегированного режима, которые обрабатывают запросы ввода-вывода от клиентов TDI (Transport Driver Interface – интерфейс транспортного драйвера)
2	Библиотека и минипорт NDIS	Канальный	Библиотека NDIS формирует среду привилегированного режима для драйверов адаптера. Драйверы минипорта NDIS – это драйверы привилегированного режима, которые взаимодействуют со средствами транспорта TDI с помощью определенного сетевого адаптера
	Уровень абстракции аппаратных средств		
1	Ethernet, IrDa и т.д.	Физический	

В операционной системе Windows поддерживается несколько сетевых API-интерфейсов для обеспечения совместимости с промышленными стандартами и предоставления поддержки для приложений, унаследованных от других операционных систем. Например, реализация API-интерфейса Sockets операционной системы Windows полностью аналогична API-интерфейсу Sockets, относящемуся к дистрибутиву программного обеспечения Беркли (Berkeley Software Distribution – BSD), который является стандартом для связи по Internet в операционной системе UNIX с 1980-х годов. Это позволяет упростить перенос приложений UNIX в ОС Windows.

Принятие решения об использовании определенного сетевого API-интерфейса в конкретном приложении сводится к определению того, насколько полно этот API-интерфейс соответствует требованиям приложения. Возможности, предоставляемые приложению сетевым API-интерфейсом, могут в значительной степени зависеть от того, какие сетевые протоколы позволяет использовать этот API-интерфейс, какие типы связи он поддерживает (надежную или ненадежную, двухстороннюю или одностороннюю связь и т.д.), а также от того, насколько легко можно будет переносить разрабатываемое приложение в другие версии Windows или в другие операционные системы, если есть необходимость либо эксплуатировать, либо легко переносить это приложение в среду другой операционной системы. Поэтому не существует такого единственного сетевого API-интерфейса, который превосходил бы все прочие сетевые API-интерфейсы во всех ситуациях.

Основные сетевые API-интерфейсы Windows перечислены ниже.

- Протокол CIFS (Common Internet File System – общий протокол доступа к файлам Internet).

- Именованные каналы (Named Pipes).
- Сокеты Windows (Windows Sockets).
- Дистанционный вызов процедур (Remote Procedure Call – RPC).
- Протокол NetBIOS.

В данной главе из числа этих API-интерфейсов будут рассматриваться только RPC, Named Pipes и Windows Sockets, поскольку в программе SQL Server предусмотрены сетевые библиотеки для каждого из них (к тому же, средства RPC операционной системы Windows используются в мультипротокольной библиотеке Net-Library). Общий протокол доступа к файлам Internet (Common Internet File System) – это механизм, с помощью которого предоставляется совместный доступ к файлам сети Windows.

С другой стороны, NetBIOS – это в основном устаревший API-интерфейс, который предшествовал появлению набора протоколов TCP/IP и API-интерфейса Sockets, ставших доминирующими технологиями организации сетевого взаимодействия для компьютеров.

API-интерфейс Named Pipes

Технология именованных каналов Windows, лежащая в основе API-интерфейса Named Pipes, была первоначально разработана для системы LAN Manager операционной системы OS/2 и перенесена в первый выпуск Windows NT. Некоторые следы применения технологии LAN Manager можно все еще обнаружить и в современных сетевых средствах Windows (например, в этой операционной системе по-прежнему используется файл `LMHOSTS`, где "LM" сокращенно обозначает LAN Manager), а именованные каналы – это важная технология, которая выдержала проверку временем.

Именованные каналы позволяют устанавливать в приложениях надежную, двухстороннюю связь по сети. В этой технологии поддерживаются средства защиты Windows, которые позволяют серверу управлять тем, какие клиенты могут получить доступ к каналу и какие операции они могут выполнять с помощью этого канала.

Средства именованных каналов тесно интегрированы со средствами Windows. Основная часть функций API-интерфейса Named Pipes реализована в библиотеке `Kernel32.dll` – в клиентской библиотеке DLL подсистемы Win32. Функция `ImpersonateNamedPipeClient` фактически реализована в библиотеке `AdvApi32.dll`, но другие функции Named Pipes реализованы в библиотеке `Kernel32.dll`. Применение именованных каналов для доступа к сетевым ресурсам осуществляется столь же просто, как и применение средств Windows других типов, таких как файлы и синхронизационные объекты.

Имена именованных каналов соответствуют стандарту универсального соглашения об именовании (Universal Naming Convention – UNC) операционной системы Windows (независимому от протоколов методу обозначения сетевых ресурсов) и имеют форму `\\`*`Server`*`\Pipe\`*`PipeName`*. Часть *`Server`* этого имени может представлять собой имя DNS, IP-адрес или имя NetBIOS. При создании канала данная

часть имени должна указывать на текущий компьютер с учетом того, что каналы не могут создаваться на других компьютерах. Операционная система Windows предоставляет возможность использовать точку (.) в качестве сокращенного обозначения текущего компьютера, поэтому, указывая на локальный канал, можно применять обозначение `\\.\` вместо фактического именования компьютера. Часть *`Pipe`* должна быть фактически представлена словом "`Pipe`". Часть *`PipeName`* может указывать на любой выбранный вами объект и включать обозначения подкаталогов.

Чтобы можно было приступить к обмену данными с использованием именованных каналов, на одном из компьютеров создается канал с помощью вызова функции `CreateNamedPipe` API-интерфейса Win32. Компьютер, на котором выполняется эта операция, называется *сервером именованного канала*. После создания канала сервер может приступить к приему клиентских запросов на установление соединения, вызвав функцию `ConnectNamedPipe`. Функция `ConnectNamedPipe` может выполняться в синхронном или асинхронном режиме.

Клиент подключается к серверу именованного канала с помощью вызова функции `CreateFile`, в котором в качестве имени файла указано имя канала. Клиент передает и принимает данные через канал, вызывая функции `WriteFile` и `ReadFile` API-интерфейса Win32, которые были описаны выше в данной книге. Клиент не обнаруживает различия между операциями, выполняемыми во время работы с файлом или с каналом. В действительности в программе можно обеспечить возможность работать с именованным каналом (т.е. стать клиентом именованного канала) без внесения изменений в само приложение, как будет вскоре показано, когда мы перейдем к упражнениям.

Функция `CreateNamedPipe` позволяет указать в приложении режим, в котором должен быть открыт канал (входящий, исходящий или дуплексный), режим работы канала (байтовый или строковый), задать применяемые по умолчанию размеры буферов ввода-вывода, атрибуты защиты, применяемые к каналу, а также многие другие параметры. Заслуживает особого внимания поддержка в именованных каналах строкового режима. Большинство сетевых API-интерфейсов поддерживает исключительно байтовый режим. Это означает, что для приема одного сообщения может потребоваться сделать множество вызовов функции приема для составления полного сообщения. Но для предотвращения указанной необходимости может использоваться строковый режим именованных каналов; в таком случае для выполнения каждой операции передачи требуется одна и только одна операция приема.

Еще одной важной характерной особенностью именованных каналов является то, что сервер может обеспечить анонимность мандата защиты клиента. Для этого вызывается функция `ImpersonateNamedPipeClient`. Данное средство используется в программе SQL Server при создании цепочки серверных соединений и доступе к ресурсам других типов, когда для аутентификации доступа необходимо применить мандат клиента именованного канала.

В именованных каналах для связи по сети используется драйвер редиректора файловой системы Windows. Это означает, что в технологии именованных каналов косвенно применяется общий протокол доступа к файлам Internet (Common Internet File System – CIFS), а также означает, что технология именованных каналов может функционировать на основе любых протоколов, поддерживаемых

протоколом CIFS, включая IPX, TCP/IP и NetBEUI. Приложение с именованными каналами может поддерживаться в любой системе, где установлены хотя бы некоторые из этих протоколов.

Упражнение

В следующем упражнении мы воспользуемся примером приложения `fstring`, первоначально представленным в главе 5, и приспособим это приложение в целях предоставления доступа по именованному каналу для ввода, вывода или выполнения обеих этих операций. Напомним, что приложение `fstring` принимало в качестве входных данных маску файла и записывало свои выходные данные на терминал. В данном упражнении мы приспособим пример `fstring_io_comp_out` из главы 5 для работы с именованными каналами. В этом новом приложении, как и прежде, все еще будут использоваться маски файла и вывод на терминал, но будет также предусмотрена возможность получать входные или выходные данные из именованного канала.

Упражнение 6.1. Утилита поиска подстроки, в которой используются именованные каналы

1. Загрузите пример проекта `fstring_pipe` из подкаталога `CH06\fstring_pipe` компакт-диска, прилагаемого к данной книге, в среду разработки Visual Studio.
2. Значительные отличия от примера `fstring_io_comp_out` обнаруживаются лишь в главном модуле исходного кода, `fstring_pipe.cpp`, поэтому в данном упражнении рассматривается только этот модуль, код которого показан в листинге 6.1.

Листинг 6.1. Главный модуль исходного кода для утилиты `fstring_pipe` (`fstring_pipe.cpp`)

```
// fstring_pipe.cpp : Многопотоковая процедура поиска в файле, которая
//                    позволяет открывать каналы для ввода и вывода
//

#include "stdafx.h"
#include "windows.h"
#include "stdlib.h"
#include "process.h"
#include "bufsrch.h"
#include "iobuf.h"

#define IO_STREAMS_PER_PROCESSOR 2

// Процедура точки входа для рабочих потоков
unsigned __stdcall StartSearch(LPVOID lpParameter)
{

  // Привести параметр, передаваемый в функцию _beginthreadex, к типу
  // CBufSearch * и вызвать метод Search класса CBufSearch

  return ((CBufSearch*)lpParameter)->Search();
```

```
}

// Выполнить поиск в указанном файле заданной искомой подстроки с
// использованием небуферизованного, асинхронного ввода-вывода
DWORD SearchFile(DWORD dwClusterSize,
                         DWORD dwNumStreams,
                         char *szPath,
                         char *szFileName,
                         HANDLE hOutputFile,
                         char *szSearchStr,
                         HANDLE hInputFile=INVALID_HANDLE_VALUE
                         )
{
 char szFullPathName[MAX_PATH+1];
 DWORD dwNumThreads;
 HANDLE hPrivHeap;
 HANDLE *hThreads;
 bool bPipe;

 char szMsg[1024];
 DWORD dwOutput;

 strcpy(szFullPathName,szPath);
 strcat(szFullPathName,szFileName);

 DWORD dwFileSizeHigh;
 DWORD dwFileSizeLow;
 DWORD dwlFileSize;

 bPipe=(INVALID_HANDLE_VALUE!=hInputFile);

 if (!bPipe) {

    // Открыть файл как для небуферизованного, так и для совмещенного
    // (асинхронного) ввода-вывода
  hInputFile=CreateFile(szFullPathName,
                              GENERIC_READ,FILE_SHARE_READ,
                               NULL,
                              OPEN_EXISTING,
                               FILE_ATTRIBUTE_NORMAL
                               | FILE_FLAG_OVERLAPPED
                              | FILE_FLAG_NO_BUFFERING
                               ,NULL);

  if (INVALID_HANDLE_VALUE==hInputFile) {
     printf("Error opening file.  Last error=%d\n",
       GetLastError());
    return -1;
  }

}

sprintf(szMsg,"Searching for %s in %s\n\n",szSearchStr,
       szFileName);

WriteFile(hOutputFile,szMsg,strlen(szMsg),&dwOutput,NULL);
```

```
   DWORD dwRetries=0;

   do {

      dwFileSizeLow=GetFileSize(hInputFile,&dwFileSizeHigh);

      dwlFileSize=(dwFileSizeHigh*MAXDWORD)+
             dwFileSizeLow;

   } while ((bPipe) &&
                 (0==dwlFileSize) &&
                (++dwRetries<12) &&
                (printf("Waiting on data from pipe client\n")) &&
                 (!SleepEx(5000,false))
                );

   if (0==dwlFileSize) return -1;

   DWORD dwNumClusts=dwlFileSize / dwClusterSize;

if (dwNumClusts<1) dwNumClusts=1;

   // Если размер файла меньше 4 Гбайт и количество затребованных
   // потоков (речь идет о потоках ввода-вывода) меньше количества
   // кластеров, установить количество потоков равным количеству
   // кластеров
   if ((dwlFileSize<0xFFFFFFFF) && (dwNumStreams>dwNumClusts))
     dwNumThreads=dwNumClusts;
   else
     dwNumThreads=dwNumStreams;

#if(_DEBUG)
   sprintf(szMsg,"Using %d threads\n\n",dwNumThreads);
   WriteFile(hOutputFile,szMsg,strlen(szMsg),&dwOutput,NULL);
#endif

   // Создать закрытую динамическую область памяти, чтобы можно было за
   // один раз освободить память, распределенную во всех операциях
   hPrivHeap=HeapCreate(0,0,0);

   // Создать массив потоков
   hThreads=(HANDLE *)HeapAlloc(hPrivHeap,
                                HEAP_ZERO_MEMORY,
                                dwNumThreads*sizeof(HANDLE));
   if (NULL==hThreads) {
      printf("Error allocating worker thread array.  Aborting.\n");
      return -1;
   }

   // Создать порт завершения ввода-вывода
   HANDLE hPort=CreateIoCompletionPort(hInputFile,NULL,0,0);
   if (INVALID_HANDLE_VALUE==hPort) {
      printf("Error creating IO completion port.  Last error=%d\n",
       GetLastError());
      return -1;
```

```
  }

  // Создать рабочие потоки, а также объекты CBufSearch и CIoBuf
  CBufSearch *pbFirst=NULL;
  CIoBuf *pIoFirst=NULL;
  unsigned uThreadId;
for (DWORD i=0; i<dwNumThreads; i++) {

    pIoFirst=new CIoBuf(pIoFirst,hPort,dwClusterSize+1);

    pbFirst=new CBufSearch(pbFirst,
                           szFileName,
                           szSearchStr,
                            hOutputFile);

    hThreads[i]=
    (HANDLE)_beginthreadex(NULL,
                            0,
                            &StartSearch,
                            pbFirst,
                            CREATE_SUSPENDED,
                             &uThreadId);

    if (!hThreads[i]) {
      printf("Error creating thread.  Aborting.\n");
      return -1;
    }
    }

    // Установить указатель объектов CBufSearch на начало списка
    pbFirst->s_pIoFirst=pIoFirst;

    // Установить указатель объектов CIoBuf на начало списка CIoBuf
    pIoFirst->s_pIoFirst=pIoFirst;

    // Задать значения статических переменных для того, чтобы
    // можно было осуществлять сразу несколько операций поиска
    // в файле
    pIoFirst->s_bTerminated=false;
    pIoFirst->s_bOverlapped=true;

    // После задания значений статических переменных
    // CBufSearch запустить рабочие потоки
    for (i=0; i<dwNumThreads; i++)
    ResumeThread(hThreads[i]);
    // Главный цикл - обрабатывать файл в цикле, читая из него
    // фрагменты с размером dwClusterSize
    DWORDLONG dwlFilePos=0;
    bool bEof=false;
    do {
     for (CBufSearch *pbCurrent=pbFirst;
      NULL!=pbCurrent;
      pbCurrent=pbCurrent->m_pbNext) {

      CIoBuf *pIoBuf=
         pIoFirst->SpinToFindBuf(BUF_STATE_INACTIVE,
```

```
                                    BUF_STATE_READING);

      // Задать начальное смещение для следующей операции чтения
      pIoBuf->m_OverlappedIO.OffsetHigh=
        (DWORD)(dwlFilePos / MAXDWORD);
     pIoBuf->m_OverlappedIO.Offset=
        (DWORD)(dwlFilePos % MAXDWORD);

      // Заполнить буфер чтения нулями, чтобы не встречались
      // совпадения с искомой подстрокой в конце частично
      // заполненного буфера (оставшиеся от ранее считанных
      // фрагментов)
      ZeroMemory(pIoBuf->m_szBuf,dwClusterSize+1);

      // Считать из файла данные в объеме полного буфера, по
      // возможности используя асинхронный ввод-вывод
      if (!ReadFile(hInputFile,pIoBuf->m_szBuf,
                          dwClusterSize,
                           &pIoBuf->m_dwBytesRead,
                          &pIoBuf->m_OverlappedIO)) {

        DWORD dwLastErr=GetLastError();
       if (ERROR_IO_PENDING!=dwLastErr)     {

          // Завершить выполнение главного цикла потока при
          // обнаружении любой ошибки, исключая ERROR_IO_PENDING,
          // но включая EOF
         InterlockedExchange(
                (LPLONG)&pIoBuf->s_bTerminated,
               (long)true);

         // Осуществить аварийное завершение, если ошибка не
         // относится к типу EOF
          if ((ERROR_HANDLE_EOF!=dwLastErr) &&
              (ERROR_BROKEN_PIPE!=dwLastErr)) {
printf(
           "Error reading file.  Last error=%d\n",
           dwLastErr);
              return -1;
         }
          else
              bEof=true;
         break;
        }
        else {
         // Имеет место асинхронная операция
        InterlockedExchange(
            (LPLONG)&pIoBuf->s_bOverlapped,
            (long)true);
        }
      }
     else {
       // Функция ReadFile возвратила истинное значение; операция
       // является синхронной
        InterlockedExchange(
            (LPLONG)&pIoBuf->s_bOverlapped,
```

```
            (long)false);
        pIoBuf->SetState(BUF_STATE_READY);

      }

     dwlFilePos+=dwClusterSize;
   }

   } while ((((!pIoFirst->s_bOverlapped) && (!bEof)) ||
   ((pIoFirst->s_bOverlapped) &&
   (dwlFilePos<dwlFileSize) &&
   (!pIoFirst->s_bTerminated)));

   // Передать сигнал о том, что чтение файла закончено
   InterlockedExchange((LPLONG)&pIoFirst->s_bTerminated,
      (long)true);

   // Ожидать завершения поиска всеми рабочими потоками
   WaitForMultipleObjects(dwNumThreads,hThreads,
                          true,
                          INFINITE);
// Получить общий итог и уничтожить объекты поиска
DWORD dwFindCount=0;
CBufSearch *pbNext;
for (; NULL!=pbFirst; pbFirst=pbNext) {
  dwFindCount+=pbFirst->m_dwFindCount;
   pbNext=pbFirst->m_pbNext;
  delete pbFirst;
}

// Удалить объекты буферов
CIoBuf *pIoNext;
for (; NULL!=pIoFirst; pIoFirst=pIoNext) {
  pIoNext=pIoFirst->m_pIoBufNext;
  delete pIoFirst;
}

// Закрыть порт завершения ввода-вывода
CloseHandle(hPort);

// Закрыть дескрипторы потоков
for (i=0; i<dwNumThreads; i++) {
   CloseHandle(hThreads[i]);
}

if (!bPipe)
  CloseHandle(hInputFile);

// Освободить память, полученную во всех предыдущих операциях
// распределения памяти из динамической области памяти, уничтожив
// созданную в программе динамическую область памяти
HeapDestroy(hPrivHeap);

sprintf(szMsg,"\nTotal hits for %s in %s:\t%d\n",
       szSearchStr,szFileName,dwFindCount);
```

```
  WriteFile(hOutputFile,szMsg,strlen(szMsg),&dwOutput,NULL);

  // Возвратить данные о количестве найденных совпадений, относящиеся к
  // указанному файлу
  return dwFindCount;
}

HANDLE OpenOutputFile(char *szOutput)
{
  HANDLE hOutputFile;
if (stremp(szOutput,"CONOUT$")) {
    do {
      hOutputFile=
        CreateFile(szOutput,
                    GENERIC_WRITE,
                   FILE_SHARE_READ,
                   NULL,
                    CREATE_ALWAYS,
                   FILE_ATTRIBUTE_NORMAL,
                   NULL);

      if (INVALID_HANDLE_VALUE==hOutputFile) {
        printf(
          "Waiting on output file/pipe.  Last error=%d\n",
          GetLastError());
      }
    } while ((INVALID_HANDLE_VALUE==hOutputFile) &&
                  (!SleepEx(5000,false)));

  }
  else
    hOutputFile=
      GetStdHandle(STD_OUTPUT_HANDLE);

  return hOutputFile;
}

// Выполнить поиск заданной подстроки в файлах, имена которых
// соответствуют указанной маске
bool SearchFiles(char *szFileMask, char *szSearchStr,
    char *szOutput, DWORD dwPeriod=0)
{

  char szPath[MAX_PATH+1];
  char szMsg[1024];
  DWORD dwOutput;
  HANDLE hOutputFile;

  // Извлечь обозначение пути к файлу из указанной маски
  char *p=strrchr(szFileMask,'\\');
  if (p) {
    strncpy(szPath,szFileMask,(p-szFileMask)+1);
    szPath[(p-szFileMask)+1]='\0';
  }
else
     // Если путь не указан, использовать текущий каталог
```

```
    GetCurrentDirectory(MAX_PATH,szPath);

    // В случае необходимости добавить заключительный символ обратной
    // косой черты
    if ('\\'!=szPath[strlen(szPath)-1])
    strcat(szPath,"\\");

    printf("Searching for %s in %s\n\n",szSearchStr,
    szFileMask);

    // Определить количество процессоров в текущей системе. Эта
    // величина будет использоваться для вычисления количества
    // потоков ввода-вывода, с помощью которых должен осуществляться
    // поиск в каждом файле
    SYSTEM_INFO si;
    GetSystemInfo(&si);

    // Определить размер кластера на диске. Эта величина всегда
    // является кратной размеру сектора, поэтому хорошо подходит
    // для использования в небуферизованном вводе-выводе
    DWORD dwSectorsPerCluster;
    DWORD dwBytesPerSector;
    DWORD dwNumberOfFreeClusters;
    DWORD dwTotalNumberOfClusters;
    GetDiskFreeSpace(NULL,&dwSectorsPerCluster,
                             &dwBytesPerSector,
                            &dwNumberOfFreeClusters,
                            &dwTotalNumberOfClusters);

    DWORD dwClusterSize=(dwSectorsPerCluster * dwBytesPerSector);

    DWORD dwFindCount=0;

    HANDLE hInputPipe=INVALID_HANDLE_VALUE;

    strupr(szFileMask);

    char *pipestr=strstr(szFileMask,"\\PIPE\\");
    if (pipestr) {
while (1) {

      hOutputFile=OpenOutputFile(szOutput);

      printf("Opening pipe %s\n",szFileMask);

      hInputPipe=CreateNamedPipe(szFileMask,
                                 PIPE_ACCESS_INBOUND
                                  ,PIPE_TYPE_BYTE,
                                 PIPE_UNLIMITED_INSTANCES,
                                  si.dwPageSize,
                                  si.dwPageSize,
                                 INFINITE,
                                 NULL);

      if (INVALID_HANDLE_VALUE==hInputPipe) {
      sprintf(szMsg,
```

```
        "Error creating named pipe.  Last error=%d\n",
        GetLastError());

        WriteFile(hOutputFile,
              szMsg,
             strlen(szMsg),
             &dwOutput,
              NULL);
             return false;
      }

      printf(
      "Waiting on client to connect to pipe %s\n",
          szFileMask);
      ConnectNamedPipe(hInputPipe
           ,NULL
          );

      dwFindCount+=SearchFile(dwClusterSize,
                              si.dwNumberOfProcessors*
                             IO_STREAMS_PER_PROCESSOR,
                             szFileMask,
                             "",
                              hOutputFile,
                              szSearchStr,
                             hInputPipe
                             );
DisconnectNamedPipe(hInputPipe);

       CloseHandle(hInputPipe);

      if (GetStdHandle(STD_OUTPUT_HANDLE)!=hOutputFile)
         CloseHandle(hOutputFile);

    };
    }
    else {

    do {

       DWORD dwpFindCount=0;  // Учесть наличие точки

       // Обработать в цикле все файлы, имена которых соответствуют
       // указанной маске, и выполнить поиск заданной подстроки
       WIN32_FIND_DATA fdFiles;
        HANDLE hFind=FindFirstFile(szFileMask,&fdFiles);
       if (INVALID_HANDLE_VALUE == hFind) {
         printf("No files match the specified mask\n");
        return false;
      }

      do {
         hOutputFile=OpenOutputFile(szOutput);
         dwpFindCount+=SearchFile(dwClusterSize,
                              si.dwNumberOfProcessors*
                               IO_STREAMS_PER_PROCESSOR,
```

```
                                szPath,
                                 fdFiles.cFileName,
                                 hOutputFile,
                                 szSearchStr,
                                hInputPipe
                                );

        dwFindCount+=dwpFindCount;

         if (GetStdHandle(STD_OUTPUT_HANDLE)!=
                        hOutputFile)
          CloseHandle(hOutputFile);

       } while (FindNextFile(hFind,&fdFiles));
FindClose(hFind);

      if (dwPeriod)
        printf(
           "\nTotal hits for %s in %s:\t%d for this
               polling period\n",
            szSearchStr,
           szFileMask,
           dwpFindCount);

      printf("\nTotal hits for %s in %s:\t%d\n",
        szSearchStr,szFileMask,dwFindCount);

    } while ((dwPeriod) && (!SleepEx(dwPeriod,false)));
  }

  return true;
}

int main(int argc, char* argv[])
{
  if (argc<3) {
    printf("Usage is: fstring_pipe filemask|pipe searchstring
         outputfilename|pipe polling_interval_secs \n");
    return 1;
  }

  try
  {

    // Определить необязательное значение имени выходного файла; по
    // умолчанию в качестве выходного файла применяется терминал
    char szOutpath[MAX_PATH+1];
    if (argc>=4)
       strncpy(szOutpath,argv[3],MAX_PATH);
    else
       strcpy(szOutpath,"CONOUT$");

    // Определить необязательное значение интервала опроса
    DWORD dwPeriod=0;
    if (argc>=5)
       dwPeriod=atol(argv[4])*1000;
```

```
      return (!SearchFiles(argv[1], argv[2],
                  szOutpath,dwPeriod));
   }
catch (...)
   {
     printf("Error reading file.  Last error=%d\n",
       GetLastError());
     return 1;
   }
}
```

3. Начнем с описания функции создания именованного канала `CreateNamedPipe` из состава процедуры `SearchFiles`. Код этой функции выглядит следующим образом:

```
hInputPipe=CreateNamedPipe(szFileMask,
                           PIPE_ACCESS_INBOUND
                           ,PIPE_TYPE_BYTE,
                           PIPE_UNLIMITED_INSTANCES,
                           si.dwPageSize,
                           si.dwPageSize,
                           INFINITE,
                           NULL);
```

   Канал создается как входящий (т.е. позволяет принимать данные от клиентов, но не передавать данные). Канал открывается для работы в байтовом (а не в строковом) режиме, и дается указание, что может быть создано неограниченное количество экземпляров этого канала. По умолчанию размеры буферов передачи и приема устанавливаются в соответствии с размером страницы системы, а заданное по умолчанию время ожидания канала устанавливается равным `INFINITE`, чтобы любой клиент, в котором вызывается функция `WaitNamedPipe`, мог ожидать в течение неопределенно долгого времени, пока канал не станет доступным.

4. В коде этого приложения поиск пути к ресурсу, с помощью которого формируется вызов функции `CreateNamedPipe`, осуществляется с помощью поиска подстроки "`\pipe\`" в маске файла, заданной в командной строке. Если обнаруживается, что в качестве входного параметра задано имя канала, в приложении пропускается участок кода, в котором предпринимается попытка перебора в цикле имен файлов, соответствующих маске, и происходит переход непосредственно к созданию канала и ожиданию подключения к нему какого-то клиента.

5. Следующим участком кода, заслуживающим внимания, является вызов функции `ConnectNamedPipe`. Он расположен непосредственно после вызова функции `CreateNamedPipe` и должен выглядеть следующим образом:

```
ConnectNamedPipe(hInputPipe
     ,NULL
     );
```

   Дескриптор `hInputPipe` получен после вызова функции `CreateNamedPipe`. В качестве указателя на структуру `OVERLAPPED` передается `NULL`, поскольку данный параметр становится необязательным, если при создании канала не задан параметр `FILE_FLAG_OVERLAPPED`. Если же канал создается с использованием

параметра FILE_FLAG_OVERLAPPED, то параметр с указателем на структуру OVERLAPPED в вызове функции ConnectNamedPipe не может быть равен NULL.

6. В функции ConnectNamedPipe не выполняется возврат до тех пор, пока не будет получен запрос на установление соединения от клиента. А после поступления такого запроса вызывается процедура SearchFile, и осуществляется поиск по такому же принципу, как и при чтении файла.

7. Обратите внимание на то, что в функции SearchFile появился новый параметр — hInputFile. Это позволяет нам открыть файл, который будет просматривать функция SearchFile, и передать дескриптор файла в эту процедуру, а не предоставлять возможность функции SearchFile самой открывать данный файл. После того как функции SearchFile передается действительный дескриптор файла, в ней не предпринимается попытка открывать или закрывать файл; она просто проводит поиск и выполняет возврат. Во всех иных отношениях функция SearchFile действует так же, как и во всех других примерах fstring — обрабатывает данные, полученные через канал, как если бы чтение выполнялось из файла. В этой функции предпринимается попытка асинхронного чтения из канала, но если это не удается, то происходит возврат к синхронной обработке.

8. Как уже было отмечено выше, функция fstring_pipe поддерживает также возможность записи формируемого ею вывода непосредственно в канал. Вообще говоря, можно было бы просто использовать средства перенаправления интерпретатора команд в командной строке для вывода на терминал результатов работы приложения fstring_pipe или применить любую другую утилиту командной строки для вывода в указанный канал. Но автор хотел показать, как сделать это с помощью вызовов API-интерфейса Win32, поэтому разработал код приложения fstring_pipe, чтобы эта программа принимала новый необязательный параметр командной строки, который указывает назначение вывода. Если этот параметр исключен (или указано имя выходного файла терминала CONOUT$), то выходные данные будут выведены на терминал, как и в других примерах приложений fstring. А если задано имя файла, то этот файл будет перезаписан с заменой содержимого выходными данными приложения fstring_pipe. Если же указан канал, то приложение fstring_pipe запишет свои выходные данные в канал, а не на терминал или в файл. Это означает, что один экземпляр приложения fstring_pipe может служить в качестве клиента именованного канала для другого экземпляра. Для этого необходимо запустить два экземпляра программы с помощью примерно таких командных строк:

```
REM Это - вызов сервера
start fstring \\.\pipe\fstring ABCDEF

REM Это - вызов клиента
fstring INPUT*.TXT ABCDEF \\.\pipe\fstring
```

Это также означает, что можно последовательно соединять в цепочки несколько экземпляров приложения fstring_pipe для прогона текста через ряд фильтров перед выводом этого текста на терминал. Если вы захотите устроить для своего сетевого администратора забавный сюрприз, то запустите от 20 до 30 таких экземпляров приложений, работающих одновременно, чтобы провести через них текстовый файл длиной в несколько гигабайтов и проверить в нем совпадения с заданными подстроками.

Итак, в этом состоят основные особенности приложения `fstring_pipe`. В нем открывается новый канал с помощью функции `CreateNamedPipe`, после чего осуществляются операции записи и чтения через именованные каналы с использованием основных функций файлового ввода-вывода Win32 – `ReadFile` и `WriteFile`.

API-интерфейс Windows Sockets

API-интерфейс Windows Sockets, широко известный под названием Winsock, представляет собой реализацию API-интерфейса BSD Sockets в операционной системе Windows. Как было указано выше, BSD Sockets – это API-интерфейс, который стал стандартом сетевого взаимодействия по Internet в системах UNIX в 1980-х годах. В операционной системе Windows предусмотрена совместимость со стандартом Sockets на уровне API-интерфейса, поэтому задача переноса в Windows сетевых приложений UNIX становится относительно несложной. Но кроме поддержки функциональных средств, уже присутствующих в API-интерфейсе BSD Sockets, API-интерфейс Winsock включает также некоторые усовершенствования, характерные для продуктов компании Microsoft. Описание некоторых из этих усовершенствований приведено ниже.

Как и API-интерфейс BSD Sockets, API-интерфейс Winsock поддерживает и надежную (с установлением логических соединений), и ненадежную связь (без установления логических соединений). Такие соединения известны под названием соответственно *потоковых* и *дейтаграммных* соединений. Кроме того, API-интерфейс Winsock поддерживает ввод-вывод на основе сборки-разборки и асинхронный ввод-вывод, возможность расширения перечня протоколов (что позволяет использовать этот интерфейс с протоколами, отличными от протоколов, которые рассматриваются в продуктах компании Microsoft как обязательные), включает встроенную поддержку пространств имен (что позволяет серверу публиковать свое имя в службе каталогов, а клиенту находить его), позволяет устанавливать соглашения о качестве и классе предоставляемых услуг передачи данных (Quality of Service – QoS) (если сеть, в которой применяется API-интерфейс Winsock, поддерживает соглашения QoS, то приложения могут регламентировать требования к пропускной способности и к задержке с помощью соглашений QoS и получать наиболее благоприятные условия функционирования в сети), а также передавать многоадресатные сообщения (что позволяет передавать одно сообщение сразу нескольким получателям).

API-интерфейс Winsock реализован в библиотеке `Ws2_32.dll`. Проще всего можно определить, используется ли в приложении API-интерфейс Winsock, проверив, отображается ли библиотека `Ws2_32.dll` на его пространство адресов процесса. (Можно также проверить таблицу импорта исполняемого файла этого приложения, но данная таблица не всегда включает ссылку на библиотеку `Ws2_32.dll`, поскольку для загрузки этой библиотеки может применяться API-интерфейс LoadLibrary или другая библиотека DLL.) Например, можно проверить, используется ли библиотека Winsock в инсталляции программы SQL Server, которая принимает запросы на установление соединений с помощью набора протоколов TCP/IP, подключившись с помощью утилиты Tlist к процессу SQL Server. Если сервер

принимает запросы на установление соединений через сокет TCP/IP, то в списке загруженных модулей должна присутствовать библиотека `Ws2_32.dll`.

После того как будет вызвана относящаяся к операционной системе Windows функция инициализации, `WSAStartup`, в любом приложении Winsock можно вызывать функции API-интерфейса, совместимые с BSD Sockets, для создания сокетов и обмена данными с их помощью. Кроме того, часто встречаются относящиеся к операционной системе Windows функции API-интерфейса, которые выполняют те же самые основные действия, что и их аналоги, относящиеся к BSD, но поддерживают дополнительные опции или предоставляют расширенные функциональные возможности (эти функции обычно содержат в своем имени префикс `WSA`). Например, первое действие, которое выполняется в приложении сервера сокетов, обычно состоит в создании сокета и переходе к приему через него запросов на установление соединения, чтобы к серверу могли подключаться клиенты. В этом случае сокет сервера представляет собой одну из оконечных точек связи в режиме обмена данными через сокет. В приложении сокет может быть создан путем вызова обычной функции `socket` или функции `WSASocket`. Но, независимо от способа создания сокета, после этого приложение переходит к приему клиентских запросов на установление соединения, поступающих через сокет, путем вызова функции `listen`. Обе эти функции, `socket` и `listen`, представляют собой варианты функций BSD Sockets, перенесенные в операционную систему Windows. А функция `WSASocket` представляет собой расширение, характерное для продуктов Microsoft.

Прежде чем появится возможность использовать сокет для приема клиентских запросов на установление соединения, поступающих по сети, сокет необходимо привязать к одному из адресов на хост-компьютере. Необходимо помнить, что Winsock — это API-интерфейс, а не сетевой протокол. В функциях этого API-интерфейса необходимо использовать какой-то сетевой протокол для обеспечения взаимодействия с другими компьютерами по сети. А при вызове функции данного API-интерфейса может быть выполнена привязка к любым протоколам, поддерживаемым в операционной системе Windows в качестве основополагающих транспортных средств, включая TCP/IP, NetBEUI и IPX.

Привязка — это способ, обеспечивающий взаимодействие между уровнями стека сетевых протоколов. В типичном стеке сетевых протоколов предусмотрено несколько разных типов привязок. В частности, обычно предусмотрены определенные способы привязки API-интерфейсов к протоколам, а для самих протоколов применяется привязка к сетевым интерфейсным платам. Как правило, если речь идет о сетевой привязке, то подразумевается последний вариант, но в данной книге будут также рассматриваться вопросы привязки API-интерфейсов, в частности, вопросы привязки API-интерфейса Winsock к конкретным сетевым протоколам.

Приложения Winsock с установлением логических соединений

После привязки к определенному адресу приложение Winsock, действующее с установлением логических соединений (т.е. потоковое), приступает к приему клиентских запросов на установление соединения путем вызова функции `listen`.

В вызове функции listen приложение указывает, какое количество одновременно действующих соединений разрешается установить в этом приложении. Затем приложение вызывает функцию accept для перехода в заблокированное состояние до момента поступления от клиента запроса на установление соединения. После поступления такого запроса функция accept возвращает в вызывающее приложение новый объект сокета. Этот сокет представляет собой относящийся к серверу конец соединения, предназначенного для обмена данными. Передавая и принимая данные через сокет (например, с использованием таких функций, как send и recv), сервер может отправлять данные клиенту и получать их от клиента.

Обратите внимание на то, что в сервере может быть также вызвана относящаяся к продуктам Microsoft функция WSAAccept, которая обеспечивает прием клиентских запросов на установление соединения с учетом некоторого условия, а также функция AcceptEx, которая предоставляет некоторые дополнительные возможности повышения производительности по сравнению с функцией accept, описанные ниже. Но независимо от используемого метода приема запросов на установление соединения, после установления самого соединения сервер может приступить к взаимодействию с клиентом, используя сокет, возвращенный функцией типа accept, в вызовах таких функций, как send и recv.

Клиент, действующий с установлением логических соединений, устанавливает соединение с сервером, вызывая функцию connect API-интерфейса Sockets и задавая сетевой адрес. После подключения к серверу клиент может приступить к передаче и приему данных, вызывая функции send и recv.

Упражнение

Рассмотрим простое приложение для работы с сокетами, чтобы ознакомиться с тем, как оно работает. В следующем упражнении представлено простое приложение сервера сокетов и простое клиентское приложение, которые можно изучить, чтобы лучше понять, как в приложениях используется интерфейс Windows Sockets для обмена данными по сети.

Упражнение 6.2. Простое серверное и клиентское приложение для работы с сокетами

1. Загрузите рабочее пространство socket_server языка Visual C++ из подкаталога CH06\socket_server компакт-диска в среду разработки Visual Studio и вызовите его на выполнение.
2. Как показывают данные, выводимые на экран, работа этого сервера начинается с привязки сокета и перехода к приему клиентских запросов на установление соединения. Ознакомимся с кодом этого приложения подробнее (листинг 6.2).

Листинг 6.2. Простой сервер сокетов

```
// socket_serv.cpp : Простое серверное приложение Winsock
//

#include "windows.h"
```

```
#include "stdafx.h"
#include "winsock2.h"

#define BUFF_SIZE 0x1000

int main(int argc, char* argv[])
{

  DWORD dwError;
  WORD wVersionRequested;
  WSADATA wsaData;

  char szBuf[BUFF_SIZE+1];

  // Инициализировать средства WSA (средства сервера Winsock) и
  // убедиться в том, что применяется правильная версия
  wVersionRequested=MAKEWORD(2,0);

  dwError=WSAStartup(wVersionRequested,&wsaData);
  if (dwError!= 0 ) {
printf("Error starting Winsock: %d\n",dwError);
    return 1;
  }

  if (LOBYTE(wsaData.wVersion) != 2 ||
    HIBYTE(wsaData.wVersion) != 0 ) {
     WSACleanup( );
     printf("Cannot locate Winsock 2.0 or later\n");
    return 1;
  }

  // Получить сокет для сервера
  SOCKET hServerSocket=
    WSASocket(AF_INET,
              SOCK_STREAM,
               0,
              NULL,
              0,
               0);
  SOCKET hClientSocket;

  sockaddr_in  soServerAddress;
  ZeroMemory(&soServerAddress,sizeof(soServerAddress));

  soServerAddress.sin_family=AF_INET;
  soServerAddress.sin_addr.s_addr=htonl(INADDR_ANY);
  soServerAddress.sin_port=htons(1234);

  printf("Binding socket.\n");

  // Выполнить привязку к указанному адресу/порту
  bind(hServerSocket,
     (sockaddr *)&soServerAddress,
    sizeof(soServerAddress));

  printf("Listening...\n");
```

```
// Разрешить одновременное использование только одного соединения
listen(hServerSocket, 1);

sockaddr_in  soClientAddress;
int iAddrSize=sizeof(soClientAddress);
ZeroMemory(&soClientAddress, sizeof(soClientAddress));

// Ожидать поступления клиентского запроса на установление соединения
hClientSocket = accept(hServerSocket,
                       (sockaddr*)(&soClientAddress),
                       &iAddrSize);

printf("Client connected\n");

HANDLE hStdOut=GetStdHandle(STD_OUTPUT_HANDLE);

// Опрашивать буфера для проверки наличия строк и выводить
// полученные строки на терминал
dwError=0;
do {
  int iBytesRead=recv(hClientSocket,szBuf,BUFF_SIZE,0);
  if (SOCKET_ERROR!=iBytesRead) {
    if (iBytesRead) {

      // Обеспечить наличие нулевого завершающего символа
      // в конце буфера
      szBuf[iBytesRead]='\0';

      // Выделить на дисплее количество считанных байтов
      // красным цветом
      CONSOLE_SCREEN_BUFFER_INFO cbi;
      GetConsoleScreenBufferInfo(hStdOut,&cbi);
       WORD wAttribs=cbi.wAttributes;
      SetConsoleTextAttribute(hStdOut,
          FOREGROUND_RED|
          FOREGROUND_INTENSITY);
      printf(
         "Received %d bytes.  Contents=\n",iBytesRead);

      // Восстановить обычные атрибуты фактически
      // отображаемого текста
      SetConsoleTextAttribute(hStdOut,wAttribs);
      printf("%s\n",szBuf);
    }
   }
  else {
    dwError=WSAGetLastError();
    printf(
      "Error receiving data from socket.  Last error=%d\n",
      dwError);
   }
} while (!dwError);

// Закрыть сокеты
closesocket(hClientSocket);
```

```
  closesocket(hServerSocket);

  // Отменить инициализацию средств WSA
  WSACleanup();
  return 0;
}
```

3. Работа приложения начинается с вызова необходимой функции инициализации `WSAStartup` и проверки для определения того, установлена ли новейшая версия интерфейса Winsock. В этом коде нет ничего, что зависело бы от наличия на компьютере установленной версии Winsock 2.0, но все равно рекомендуется проводить эту проверку, чтобы быть уверенным в том, что не приходится работать с устаревшей версией библиотеки Winsock. Операционная система Windows 2000 поставляется с версией Winsock 2.2, а наиболее современные версии библиотеки Winsock для всех поддерживаемых платформ Windows доступны для загрузки с Web-узла Microsoft.
4. После проверки версии Winsock создается сокет. В данном случае вызывается функция `WSASocket` и ей передается параметр с обозначением семейства протоколов `AF_INET`, который указывает на то, что в дальнейшем должна быть выполнена привязка к набору протоколов TCP/IP.
5. Теперь необходимо рассмотреть, какие операции присваивания выполняются применительно к элементам структуры `soServerAddress`. Именно в этой структуре задается адрес, к которому будет осуществляться привязка. Прежде всего задается принадлежность семейства адресов к `AF_INET`, а это означает, что присваиваемый адрес является адресом TCP/IP. После этого в качестве действительного адреса задается `INADDR_ANY`. Это означает, что привязка должна осуществляться к любому доступному IP-адресу на компьютере. Если компьютер является многоадресным (это означает, что привязка протоколов TCP/IP выполнена к нескольким сетевым платам на компьютере), то возможным присваиваемым адресом является один из нескольких адресов. В файле заголовка `Winsock2.h` константа `INADDR_ANY` определена как целочисленное значение, равное `0`, но это условие может когда-либо измениться, поэтому в программах следует всегда использовать константу `INADDR_ANY`, а не `0`. Обратите внимание на то, что в данном коде используется функция `htonl`. Функция `htonl` преобразует длинное целое число без знака (в данном случае содержащее IP-адрес) из порядка байтов, принятого на хост-компьютере, в сетевой порядок байтов TCP/IP, в котором числа оканчиваются старшим байтом.
6. Присваивание значений элементам структуры `soServerAddress` оканчивается заданием используемого номера порта, который в данном случае равен `1234`. При этом снова необходимо изменить порядок байтов, поэтому вызывается функция `htons`, которая преобразует короткие целые числа без знака в числа, представленные в сетевом порядке байтов TCP/IP.
7. После определения адреса, к которому должна быть выполнена привязка, вызывается функция `bind` для фактического осуществления этой привязки. После завершения работы функции `bind` вызывается функция `listen` для перехода к приему клиентских запросов на установление соединения. Функция `listen` не блокируется, а подготавливает сокет для приема клиентских запросов на установление соединения. Обратите внимание на то, что в качестве

второго параметра функции `listen` передается 1. Это означает, что к сокету разрешено подключаться только одному клиенту одновременно.

8. После того как был начат прием клиентских запросов на установление соединения, вызывается функция `accept` для перехода в состояние ожидания поступления запросов на установление соединения. Функция `accept` имеет два выходных параметра, в которые помещается информация об адресе клиента, подключающегося к серверу. Кроме того, эта функция возвращает новый сокет, который в дальнейшем используется для передачи данных к клиенту и от клиента.
9. Работа приложения завершается переходом в цикл, в котором вызывается функция `recv` для чтения данных, полученных от клиента. Функция `recv` возвращает данные о количестве байтов, полученных от клиента. До тех пор пока это количество остается ненулевым, продолжается чтение строк, поступающих от клиента, и запись их на терминал.

Итак, на этом завершается описание серверного приложения. Теперь рассмотрим клиентское приложение.

1. Пока серверное приложение все еще работает, запустите еще один экземпляр среды разработки Visual C++ и загрузите в эту среду проект `socket_client` из подкаталога `CH06\socket_client` компакт-диска.
2. Рассмотрим код клиентского приложения (листинг 6.3), прежде чем вызвать его на выполнение. Заслуживает внимания то, что он во многом напоминает код приложения `socket_server`.

Листинг 6.3. Простой клиент сокетов

```
// socket_client.cpp : Простое клиентское приложение Winsock
//

#include "windows.h"
#include "stdafx.h"
#include "winsock2.h"
#include "stdlib.h"

#define BUFF_SIZE 0x1000

// Клиентское приложение, работа которого заключается лишь в том,
// чтобы подключиться к серверу и отправить сообщение
int main(int argc, char* argv[]) {

  if (argc<2) {
    printf(
      "Usage: socket_client hostname:port [L] (for looping) \n");
      return -1;
  }

  WORD wVersionRequested;
  WSADATA wsaData;
  int iErr;

wVersionRequested = MAKEWORD(2,0);

    iErr = WSAStartup(wVersionRequested,&wsaData);
```

```
    if (iErr != 0) {
     return -1;
    }

    if (LOBYTE(wsaData.wVersion) != 2 ||
    HIBYTE(wsaData.wVersion) != 0 ) {
      WSACleanup( );
      return -1;
    }

    char szHostName[MAX_PATH+1];
    char *p=strchr(argv[1],':');
    if (!p) {
    printf("Invalid or missing port specification\n");
     return -1;
    }
    strncpy(szHostName,argv[1],p-argv[1]);
    szHostName[p-argv[1]]='\0';

    p++;
    int iPort=atoi(p);

    HOSTENT *heServer=gethostbyname(szHostName);

    if (!heServer) {
    printf("Unknown host %s\n",szHostName);
     return -1;
    }

    bool bLoop=((argc>2) && (!stricmp(argv[2],"L")));

    SOCKET soServer=WSASocket(AF_INET,
                              SOCK_STREAM,
                               0,
                              NULL,
                              0,
                              0
                               ) ;

    char szBuffer[BUFF_SIZE];
    sockaddr_in saServerAddress;
ZeroMemory(&saServerAddress, sizeof(saServerAddress));

  saServerAddress.sin_family = AF_INET;

  memcpy(&(saServerAddress.sin_addr),
    heServer->h_addr_list[0],
    heServer->h_length);

  saServerAddress.sin_port = htons(iPort);

  printf("Connecting to the server ...\n");
  if (connect(soServer,
                (sockaddr *)&saServerAddress,
                sizeof(saServerAddress))) {
    printf("Connection failed %d\n",WSAGetLastError());
  }
```

```
  DWORD dwLastErr=0;

  do {
    sprintf(szBuffer,
      "Greetings from process %d\n",GetCurrentProcessId());
    printf ("Sending message ...\n");
      if (SOCKET_ERROR==
        send(soServer, (char*)(&szBuffer), BUFF_SIZE, 0))
      {
        dwLastErr=WSAGetLastError();
        printf("Send error. %d\n", dwLastErr);
      }
      else Sleep(5000);
  } while ((bLoop) && (!dwLastErr));
  closesocket(soServer);
  WSACleanup();
  return 0;
}
```

3. Выполнение кода снова начинается с проверки версии Winsock и создания сокета. Затем происходит разбивка параметра `hostname:port`, переданного в это приложение, на составные части, поскольку имя хоста и порт назначения — это отдельные элементы в структуре `sockaddr_in`, передаваемой в функцию `connect`. После выполнения указанной операции берется имя хоста и определяется соответствующий ему IP-адрес с помощью вызова функции `gethostbyname`. Функция `gethostbyname` возвращает сетевой адрес указанного хоста. (Следует отметить, что в функцию `gethostbyname` нельзя передавать IP-адрес; должно быть передано действительное сетевое имя.) Адрес, возвращаемый функцией `gethostbyname`, уже имеет правильный порядок байтов, поэтому просто вызывается функция `memcpy` для пересылки адреса в структуру `sockaddr_in`, которая используется для подключения к серверу.
4. После определения порта назначения вызывается функция `connect` для подключения к серверу. Если вызов функции `connect` завершается успешно, эта функция возвращает `0`. Кроме того, указанная функция возвращает два выходных параметра, которые содержат информацию об адресе сервера, к которому только что было выполнено подключение.
5. После подключения к серверу клиентское приложение входит в цикл, в котором осуществляется передача на сервер строки через каждые 5 секунд до тех пор, пока не возникнет ошибка. В случае возникновения ошибки (например, из-за того, что сервер закроет свой сокет или не сможет ответить), выполняется завершающая часть цикла, и происходит выход из программы.
6. Теперь попытаемся вызвать приложение на выполнение. Проверьте параметры командной строки в интегрированной среде разработки Visual Studio и обязательно задайте следующие значения этих параметров:

   `localhost:1234 L`

   Имя хоста `localhost` должно быть преобразовано в `127.0.0.1`, в адрес петли обратной связи. Любой адрес TCP/IP в формате `127.n.n.n` представляет собой петлевой интерфейс, который указывает на текущий компьютер. Число справа от двоеточия указывает порт получателя. В приведенных выше параметрах командной строки задано значение `1234`, поскольку именно это значение

было указано в серверном приложении. Параметр L сообщает клиентскому приложению, что цикл должен осуществляться непрерывно, а одно и то же сообщение — передаваться снова и снова. После проверки правильности заданных параметров вызовите клиентское приложение на выполнение.

7. Теперь переключитесь на работающее серверное приложение. Вы должны обнаружить, что в окне терминала серверного приложения появляется примерно такой вывод:

```
Received 4096 bytes. Contents=
Greetings from process 3512
```

8. Эти сообщения будут продолжать появляться в окне терминала серверного приложения до тех пор, пока не будет остановлено клиентское приложение. Можно остановить и клиент, и сервер, нажав клавиши <Ctrl+C> после перехода в соответствующее окно терминала. Теперь выполните это указание и остановите работы обоих приложений. Итак, простые клиентское и серверное приложения Winsock выглядят аналогично описанным выше. В коде Net-Library программы SQL Server вызываются некоторые из указанных выше функций API-интерфейса для передачи данных между клиентом и сервером после подключения сервера с помощью библиотеки Super Socket Net-Library.

Приложения Winsock без установления логических соединений

После привязки к сетевому адресу сервер без установления логических соединений и клиент без установления логических соединений действуют одинаково — каждый из них передает и принимает сообщения через подходящий для этого сокет, просто задавая удаленный адрес в каждом сообщении. При этом вызов функции accept или connect не требуется, поскольку каждый участник обмена данными просто включает адрес получателя в каждую передаваемую дейтаграмму с использованием функций, подобных sendto и recvfrom.

Сокеты и API-интерфейс Win32

Поскольку сокет в действительности представляет собой просто дескриптор файла, хотя и не кажется таковым на первый взгляд, в приложениях Windows можно использовать для передачи и приема данных через сокеты такие простые функции ввода-вывода, как WriteFile и ReadFile. По сути, при этом в требуемую функцию ввода-вывода Win32 достаточно лишь передать сокет после приведения его типа к типу дескриптора.

Упражнение

Лучше всего можно понять, каким образом дескрипторы сокетов применяются вместо дескрипторов файлов, изучив некоторый код, в котором показано, как выполняется эта операция. В следующем упражнении представлено приложение сервера сокетов, в котором для чтения данных, полученных от клиента сокетов, используется функция ReadFile вместо recv.

Упражнение 6.3. Серверное приложение Winsock, в котором для взаимодействия с клиентом используются функции ввода-вывода Win32

1. Закройте рабочее пространство `socket_server`, если оно еще загружено, и загрузите проект `socket_server_rf` из подкаталога `CH06\socket_server_rf` компакт-диска.
2. Единственной частью этого кода, которая значительно отличается от приложения `socket_server`, является цикл, в котором происходит получение данных от клиента. Рассмотрим эту часть (листинг 6.4).

Листинг 6.4. Вариант приложения `socket_server`, в котором используются функции ввода-вывода Win32

```
dwError=0;
    do {
      DWORD dwBytesRead;
      if (!ReadFile((HANDLE)hClientSocket,
                szBuf,
                BUFF_SIZE,
                &dwBytesRead,
                NULL)) {
        dwError=GetLastError();
if (ERROR_HANDLE_EOF==dwError)
          break;
        printf("Error reading socket %d\n",dwError);
      }
      else {
        if (dwBytesRead) {

          // Обеспечить наличие нулевого завершающего символа
          // в конце буфера
          szBuf[dwBytesRead]='\0';

          // Выделить на дисплее количество считанных байтов
          // красным цветом
          CONSOLE_SCREEN_BUFFER_INFO cbi;
          GetConsoleScreenBufferInfo(hStdOut,&cbi);
          WORD wAttribs=cbi.wAttributes;
          SetConsoleTextAttribute(hStdOut,
               FOREGROUND_RED|
               FOREGROUND_INTENSITY);
          printf(
               "Received %d bytes.  Contents=\n",
               dwBytesRead);

          // Восстановить обычные атрибуты фактически
          // отображаемого текста
          SetConsoleTextAttribute(hStdOut,wAttribs);
          printf("%s\n",szBuf);
        }
        else break;
      }
    } while (!dwError);
```

3. В этом коде заслуживает внимания вызов функции `ReadFile` интерфейса Win32. Эта функция вызывается синхронно (об этом свидетельствует значение `NULL` указателя на структуру `OVERLAPPED`), поэтому данная функция действует в такой форме, которая очень напоминает действия функции `recv` в приложении `socket_server`, описанном выше. Но вместо использования возвращаемого значения функции для определения полученного количества байтов, в функцию `ReadFile` передается параметр в формате `DWORD`, как и при осуществлении операций синхронного чтения применительно к файлу или к именованному каналу.
4. Чтобы можно было передать сокет в функцию `ReadFile`, используется приведение типов. Даже несмотря на то, что и сокет, и дескриптор файла фактически являются указателями, все равно нельзя успешно передать сокет в функцию ввода-вывода Win32, такую как `ReadFile`, если сокеты еще не встроены в систему ввода-вывода Windows. Но благодаря интеграции средств ввода-вывода и сокетов можно рассматривать сокеты с более общей точки зрения и использовать их главным образом в таких же операциях, в которых используются файлы. В данном упражнении показан лишь самый простой пример применения таких функциональных возможностей; более сложный пример будет приведен немного позже.
5. Теперь продолжите работу и вызовите приложение на выполнение. Пока оно еще работает, перезапустите приложение `socket_client`. Вы обнаружите, что клиентское приложение все еще способно передавать данные через сокет в новое приложение, как и в случае первоначального приложения `socket_server`. Ознакомившись с тем, как работают оба приложения в течение нескольких секунд, продолжите выполнение данного упражнения. Для этого остановите оба приложения и закройте экземпляры среды Visual Studio, в которые они загружены. Итак, на этом завершается описание приложения `socket_server_rf`. В нем используется функция ввода-вывода `ReadFile` интерфейса Win32 для чтения из сокета TCP/IP по такому же принципу, как если бы этот сокет был файлом.

Расширения Winsock

Следует отметить, что интерфейс Winsock поддерживает не только асинхронные, но и синхронные операции. Интерфейс Winsock поддерживает операции подключения, чтения, записи и другие операции, в которых сокет используется в асинхронном режиме. По аналогии с другими асинхронными операциями ввода-вывода, после завершения асинхронной операции с сокетом в приложении может быть получено извещение с помощью сообщения Windows, функции обратного вызова или пакета завершения ввода-вывода.

Но кроме функций, соответствующих функциям API-интерфейса BSD Sockets, API-интерфейс Winsock поддерживает две функции, которые полностью отсутствуют в API-интерфейсе BSD Sockets. В частности, первая функция, `TransmitFile`, используется для отправки клиенту полного файла. API-интерфейс Winsock интегрирован с системным файловым кэшем, поэтому файл может быть передан непосредственно из кэша без предварительного копирования его во вторичный буфер (эта операция называется *передачей файла из нулевой копии*, поскольку файл не приходится копировать для передачи в другой буфер). Кроме того, функция `TransmitFile` дает возможность добавлять в вызывающей функции

данные до или после файла в потоке передачи (например, снабжать файл заголовком), а это также позволяет избежать необходимости предварительно копировать файл во вторичный буфер перед передачей его клиенту.

Вторая функция, `AcceptEx`, не только выполняет действия стандартной функции `accept`, но и возвращает адрес клиента и его первое сообщение, что позволяет избежать необходимости выполнять в вызывающем приложении несколько вызовов для получения идентичной информации. Такой способ организации работы позволяет добиться существенного повышения производительности, особенно если клиент подключается, передает одно сообщение и отключается. В подобной ситуации применение вместо `accept` функции `AcceptEx` или `WSAAccept` позволяет вдвое уменьшить количество вызовов функций API-интерфейса, которые должны быть сделаны сервером.

Сравнение сокетов и каналов

В этом последнем разделе, посвященном интерфейсу Winsock, автор представит пример простого приложения, который позволяет сравнить преимущества и недостатки работы с именованными каналами, с одной стороны, и сокетами, с другой стороны. Между этими подходами не существует значительных различий, но каждый API-интерфейс имеет свои сильные и слабые стороны.

Упражнение

В следующем упражнении берется знакомый пример приложения `fstring`, применяемый во всей данной книге, и дорабатывается в целях его использования для ввода или вывода через сокет, именованный канал или файл. При выводе в качестве файла можно использовать терминал системы (который имеет специальное системное имя `CONOUT$`). Выполняя это упражнение и сравнивая код, применяемый при создании канала, с кодом, необходимым при создании сокета, читатель сможет получить представление о том, чем отличаются эти два подхода и в каких ситуациях можно предпочесть один из них другому.

Упражнение 6.4. Вариант приложения `fstring`, в котором применяются и сокеты, и каналы

1. Загрузите рабочее пространство `fstring_pipe_socket` из подкаталога `CH06\fstring_pipe_socket` компакт-диска в среду разработки Visual Studio.
2. Настоящее приложение представляет собой вариант примера `fstring_pipe`, представленного выше в данной главе, в котором, кроме поддержки ввода-вывода по именованным каналам, поддерживается также использование интерфейса Windows Sockets. Существенные изменения по сравнению с предыдущим вариантом произошли только в двух процедурах, `SearchFiles` и `OpenOutputFile`, поэтому в данном упражнении в основном сосредоточимся на них. Начнем с процедуры `SearchFiles` (листинг 6.5).

Листинг 6.5. Процедура `SearchFiles` приложения `fstring_pipe_socket`

```
// Выполнить поиск заданной подстроки в файлах, имена которых
// соответствуют указанной маске
bool SearchFiles(char *szFileMask,
                 char *szSearchStr,
                 char *szOutput,
                 DWORD dwPeriod=0)
{

  char szPath[MAX_PATH+1];
  char szMsg[1024];
  DWORD dwOutput;
  HANDLE hOutputFile;

// Извлечь обозначение пути к файлу из указанной маски
   char *p=strrchr(szFileMask,'\\');
   if (p) {
   strncpy(szPath,szFileMask,(p-szFileMask)+1);
   szPath[(p-szFileMask)+1]='\0';
   }
   else
    // Если путь не указан, использовать текущий каталог
   GetCurrentDirectory(MAX_PATH,szPath);

   // В случае необходимости добавить заключительный символ обратной
   // косой черты
   if ('\\'!=szPath[strlen(szPath)-1])
   strcat(szPath,"\\");

   printf("Searching for %s in %s\n\n",szSearchStr,
   szFileMask);

   // Определить количество процессоров в текущей системе. Эта
   // величина будет использоваться для вычисления количества потоков
   // ввода-вывода, с помощью которых должен осуществляться поиск
   // в каждом файле
   SYSTEM_INFO si;
   GetSystemInfo(&si);

   // Определить размер кластера на диске. Эта величина всегда
   // является кратной размеру сектора, поэтому хорошо подходит
   // для использования в небуферизованном вводе-выводе
   DWORD dwSectorsPerCluster;
   DWORD dwBytesPerSector;
   DWORD dwNumberOfFreeClusters;
   DWORD dwTotalNumberOfClusters;
   GetDiskFreeSpace(NULL,&dwSectorsPerCluster,
                            &dwBytesPerSector,
                           &dwNumberOfFreeClusters,
                           &dwTotalNumberOfClusters);

   DWORD dwClusterSize=(dwSectorsPerCluster * dwBytesPerSector);

   DWORD dwFindCount=0;

   HANDLE hInputPipe=INVALID_HANDLE_VALUE;
```

```
DWORD dwInputType=INPUT_TYPE_FILE;

    strupr(szFileMask);

    char *pszPipe=strstr(szFileMask,"\\PIPE\\");
    char *pszPort=PortString(szFileMask);
    if (pszPipe) dwInputType=INPUT_TYPE_PIPE;
    else {
     if (pszPort)
      dwInputType=INPUT_TYPE_SOCKET;
     }
    switch (dwInputType) {
    case INPUT_TYPE_PIPE :
     {
       while (1) {

        hOutputFile=OpenOutputFile(szOutput);

         printf("Opening pipe %s\n",szFileMask);

        hInputPipe=CreateNamedPipe(szFileMask,
                            PIPE_ACCESS_INBOUND
                           ,PIPE_TYPE_BYTE,
                           PIPE_UNLIMITED_INSTANCES,
                            si.dwPageSize,
                            si.dwPageSize,
                           INFINITE,
                           NULL);

         if (INVALID_HANDLE_VALUE==hInputPipe) {
        sprintf(szMsg,
            "Error creating named pipe.  Last error=%d\n",
           GetLastError());
          WriteFile(hOutputFile,
                      szMsg,
                     strlen(szMsg),
                     &dwOutput,
                      NULL);
         return false;
        }

        printf(
            "Waiting on client to connect to pipe %s\n",
                 szFileMask);
ConnectNamedPipe(hInputPipe
                ,NULL
                 );

         dwFindCount+=SearchFile(dwClusterSize,
                      si.dwNumberOfProcessors*
                      IO_STREAMS_PER_PROCESSOR,
                      szFileMask,
                      "",
                      hOutputFile,
                      szSearchStr,
                      hInputPipe);
```

```
        DisconnectNamedPipe(hInputPipe);
         CloseHandle(hInputPipe);
         CloseOutputFile(szOutput,hOutputFile);
       };
       break;
      }
    case INPUT_TYPE_SOCKET :
     {
       while (1) {
        if (!InitializeWSA()) {
           return false;
        }

         hOutputFile=OpenOutputFile(szOutput);

        printf("Opening socket for %s\n",szFileMask);
         // Получить сокет для сервера
         SOCKET hServerSocket=
           WSASocket(AF_INET,
                     SOCK_STREAM,
                      0,
                     NULL,
                      0,
                      0);
         SOCKET hClientSocket;

         sockaddr_in  soServerAddress;
        ZeroMemory(&soServerAddress,
           sizeof(soServerAddress));
u_short usPort=atoi(pszPort);
        soServerAddress.sin_family=AF_INET;

        soServerAddress.sin_addr.s_addr=
              htonl(INADDR_ANY);

        soServerAddress.sin_port=htons(usPort);

        // Выполнить привязку к указанному адресу/порту
        bind(hServerSocket,
            (sockaddr *)&soServerAddress,
            sizeof(soServerAddress));

        // Разрешить одновременное использование только
        // одного соединения
        listen(hServerSocket, 1);

        sockaddr_in  soClientAddress;
        int iAddrSize=sizeof(soClientAddress);
        ZeroMemory(&soClientAddress,
              sizeof(soClientAddress));

        // Ожидать поступления клиентского запроса
        // на установление соединения
        printf(
            "Waiting on client to connect to socket on
            %s\n",
            szFileMask);
```

```
        hClientSocket = accept(hServerSocket,
            (sockaddr*)(&soClientAddress),
            &iAddrSize);

        dwFindCount+=SearchFile(dwClusterSize,
                            si.dwNumberOfProcessors*
                               IO_STREAMS_PER_PROCESSOR,
                            szFileMask,
                            "",
                            hOutputFile,
                            szSearchStr,
                            (HANDLE)hClientSocket);

        // Закрыть сокеты
        closesocket(hClientSocket);
        closesocket(hServerSocket);

        CloseOutputFile(szOutput, hOutputFile);
    // Отменить инициализацию средств WSA
        WSACleanup();
     }
      break;

    }
case INPUT_TYPE_FILE:
    {
     do {

        DWORD dwpFindCount=0;  // Учесть наличие точки

       // Обработать в цикле все файлы, имена которых соответствуют
       // указанной маске, и выполнить поиск заданной подстроки
       WIN32_FIND_DATA fdFiles;
        HANDLE hFind=
              FindFirstFile(szFileMask,&fdFiles);

        if (INVALID_HANDLE_VALUE == hFind) {
         printf("No files match the specified mask\n");
         return false;
       }

        do {
         hOutputFile=OpenOutputFile(szOutput);
          dwpFindCount+=SearchFile(dwClusterSize,
                 si.dwNumberOfProcessors*
                 IO_STREAMS_PER_PROCESSOR,
                  szPath,
                  fdFiles.cFileName,
                  hOutputFile,
                  szSearchStr,
                  hInputPipe);

          dwFindCount+=dwpFindCount;

         CloseOutputFile(szOutput,hOutputFile);

        } while ((FindNextFile(hFind,&fdFiles)));

        FindClose(hFind);
```

```
if (dwPeriod)
          printf(
            "\nTotal hits for %s in %s:\t%d for this
                polling period\n",
            szSearchStr,
            szFileMask,
            dwpFindCount);

        printf("\nTotal hits for %s in %s:\t%d\n",
          szSearchStr,szFileMask,dwFindCount);

      } while ((dwPeriod) &&
           (!SleepEx(dwPeriod,false)));
      break;
    }
  }

  return true;
}
```

3. Наиболее заметное изменение, которое заслуживает описания в этом упражнении, можно обнаружить при изучении оператора `switch (dwInputType)`. После определения того, к какому типу относится применяемое средство ввода (канал, сокет или файл, имя которого указано с помощью маски), управление передается в этот оператор `switch`, поэтому для каждого типа средства ввода выполняется разный код. При использовании файла, имя которого указано с помощью маски, происходит обработка в цикле файлов, имена которых соответствуют маске, и применительно к каждому из них вызывается функция `SearchFile`. При использовании в качестве средства ввода именованного канала создается канал, происходит ожидание поступления запроса на подключение от клиента, после чего дескриптор канала передается в функцию `SearchFile`, в которой осуществляется чтение через канал по такому же принципу, как если бы это был входной файл. При использовании сокета проверяется версия библиотеки Winsock, открывается сокет, вызывается функция `listen` для перехода к приему запросов на установление соединения, а затем вызывается функция `accept` для перехода в заблокированное состояние до тех пор, пока не будет получен такой запрос. После того как будет создано соединение, полученный в результате дескриптор сокета передается в функцию `SearchFile`, в которой снова происходит чтение через сокет по такому же принципу, как если бы это был входной файл. Основная концепция, которая становится очевидной при изучении данного кода, состоит в том, что мы имеем возможность рассматривать каналы и сокеты практически на одинаковых основаниях по двум причинам: во-первых, в них применяются аналогичные функции API-интерфейса, и, во-вторых, те и другие объединены с моделью ввода-вывода Windows и поэтому могут использоваться как взаимозаменяемые объекты в вызовах функций ввода-вывода Win32.
4. Теперь рассмотрим функцию `OpenOutputFile`. В этой функции учтена возможность применения трех различных типов средств вывода — канала, сокета или файла (в последнем случае в качестве файла может служить терминал). Код указанной функции показан в листинге 6.6.

Листинг 6.6. Процедура `OpenOutputFile` приложения `fstring_pipe_socket`

```
HANDLE OpenOutputFile(char *szOutput)
{
  HANDLE hOutputFile;
  if (stricmp(szOutput,"CONOUT$")) {

     // Сокет
    char *pszPort=PortString(szOutput);
    if (pszPort) {
        if (!InitializeWSA()) {
        return INVALID_HANDLE_VALUE;
         }

        char szHostName[MAX_PATH+1];
         strncpy(szHostName,szOutput,pszPort-szOutput-1);
         szHostName[pszPort-szOutput-1]='\0';

      HOSTENT *heServer=gethostbyname(szHostName);

        if (!heServer) {
          printf("Unknown host %s\n",szHostName);
           return INVALID_HANDLE_VALUE;
         }
        hOutputFile=(HANDLE)WSASocket(AF_INET,
                                      SOCK_STREAM,
                                      0,
                                      NULL,
                                       0,
                                       0) ;

        sockaddr_in saOutput;
         ZeroMemory((char *)&saOutput, sizeof(saOutput));

        u_short usPort=atoi(pszPort);

         saOutput.sin_family = AF_INET;
        memcpy(&(saOutput.sin_addr),
          heServer->h_addr_list[0],
          heServer->h_length);
         saOutput.sin_port = htons(usPort);

         do {
} while ((connect((SOCKET)hOutputFile,
                              (sockaddr *) &saOutput,
                              sizeof(saOutput))) &&
           (printf(
             "Waiting on output socket.  Last error=%d\n",
              WSAGetLastError())) &&
              (!SleepEx(5000,false)));

    }
    else {
      // Файл или канал
      do {
        hOutputFile=
         CreateFile(szOutput,
                    GENERIC_WRITE,
```

```
                        FILE_SHARE_READ,
                       NULL,
                        CREATE_ALWAYS,
                       FILE_ATTRIBUTE_NORMAL,
                       NULL);

     } while ((INVALID_HANDLE_VALUE==hOutputFile) &&
        (printf(
         "Waiting on output file/pipe.  Last error=%d\n",
         GetLastError())) &&
          (!SleepEx(5000,false)));
    }

  }
  else
     hOutputFile=
       GetStdHandle(STD_OUTPUT_HANDLE);

  return hOutputFile;
}
```

5. Поскольку в данном случае происходит настройка на формирование выходного, а не входного файла, функция `OpenOutputFile` преобразуется в клиентское приложение, если задан либо канал, либо сокет. Если же задан файл, эта функция просто открывает файл или возвращает указатель на терминал, если в качестве имени файла указано `CONOUT$`. Для сокета или канала в функции `OpenOutputFile` применяются необходимые вызовы для подключения к серверу и возврата дескриптора сокета или канала, который должен использоваться в качестве средства вывода для остальной части приложения. В функции `SearchFile` этот дескриптор применяется для записи выходных данных этой функции. Если функция `OpenOutputFile` подключается к именованному каналу или серверу сокетов, то функция `SearchFile` действует как обычный клиент этого сервера, а для передачи данных используются простые вызовы функции `WriteFile`.
6. Задайте в качестве параметров этого приложения строку `localhost:1234 ABCDEF` и вызовите его на выполнение. В результате приложение откроет сокет в порту `1234` текущего компьютера и будет ждать клиентских запросов на установление соединения.
7. Затем откройте еще одно окно с приглашением к вводу команд и запустите второй экземпляр приложения `fstring_pipe_socket` с такими параметрами: `INPUT*.TXT ABCDEF localhost:1234`. Этот экземпляр будет перенаправлять свой вывод в первый экземпляр. Вы должны обнаружить, что в серверном приложении создается отдельное соединение в расчете на каждый файл, имя которого соответствует маске `INPUT*.TXT`, обнаруженный клиентским приложением. Обратите внимание на то, что количество совпадений с искомой подстрокой больше обычного, поскольку названия заголовка и концевика приложения `fstring` также включают искомую подстроку. Эти данные перенаправляются в качестве входных данных в первый экземпляр приложения `fstring_pipe_socket`, что приводит к обнаружению совпадений с искомой строкой по аналогии с тем, как обнаруживаются совпадения с подстроками, содержащимися в файлах `INPUT*.TXT`.

8. Проведите эксперименты, используя различные сочетания средств ввода и вывода на основе канала и сокета. Кроме того, попробуйте применить перенаправление для ввода данных в экземпляр `fstring_pipe_socket`, который вызван на выполнение в качестве сервера каналов, примерно таким образом:

   ```
   TYPE INPUT3.TXT >\\.\pipe\fstring
   ```

   Этот пример показывает, что фактически для передачи входных данных на сервер каналов не требуется клиентское приложение, поскольку с этой задачей вполне позволяет справиться простое перенаправление.

9. Кроме того, попытайтесь запустить приложение `socket_client` и передать из него текст в приложение `fstring_pipe_socket`. Но если вы не откорректируйте приложение `socket_client`, то вам потребуется изменить искомую подстроку приложения `fstring_pipe_socket`, чтобы она совпадала с какой-то частью строки "`greetings`", передаваемой приложением `socket_client`.

На этом завершается обсуждение интерфейса Winsock. Теперь перейдем к описанию средств RPC операционной системы Windows.

Дистанционный вызов процедур

В начале 1980-х годов организацией Open Software Foundation (которая теперь называется The Open Group) был разработан стандарт сетевого программирования RPC в составе стандарта распределенных вычислений DCE (Distributed Computing Environment – среда распределенных вычислений). Реализация RPC, применяемая в продуктах Microsoft, совместима с этой первоначальной спецификацией.

В спецификации RPC для предоставления модели программирования, которая скрывает от разработчика основную часть подробностей организации связей по сети, используются другие сетевые API-интерфейсы (например, Named Pipes, Message Queuing или Winsock). Поскольку средства RPC фактически опираются в своей работе на другие API-интерфейсы, в них могут использоваться любые средства сетевого транспорта, поддерживаемые этими API-интерфейсами, поэтому средства RPC совместимы с любыми средствами сетевого транспорта, применяемыми в системе.

Приложение RPC состоит из ряда процедур; некоторые из них являются локальными, а другие находятся на удаленных компьютерах. Но с точки зрения приложения все эти процедуры рассматриваются как локальные. Процедурам, фактически находящимся на удаленных компьютерах, соответствуют процедуры-заглушки, находящиеся на локальном компьютере и полностью согласованные с ними по принципам вызова и спискам параметров. Разработчик приложения просто вызывает эти процедуры-заглушки, не всегда даже зная о том, где физически расположены удаленные процедуры, соответствующие заглушкам. В простых приложениях обычно применяется статическое связывание процедур-заглушек с приложениями. А в более сложных приложениях процедуры-заглушки часто находятся в отдельных библиотеках DLL (например, в отдельных библиотеках DLL в технологии DCOM (Distributed COM), в которой технология RPC используется в качестве средства выполнения кода на удаленных компьютерах).

Маршалинг

После вызова процедуры-заглушки необходимо подготовить переданные ей параметры для передачи их по сети в фактически выполняемую процедуру. Такая операция подготовки называется *маршалингом*. Маршалинг можно сравнить с действиями, выполняемыми уличными регулировщиками или сопровождающими важную особу лицами; средства маршалинга выполняют всю работу, необходимую для перенаправления данных из одного контекста выполнения в другой. В том случае, если маршалинг осуществляется для передачи данных с одного компьютера на другой, очевидно, что данные не только переходят с одного компьютера на другой, но и из одного процесса в другой, поэтому при выполнении операции маршалинга необходимо обеспечить разадресацию любых указателей, переданных в процедуру-заглушку, включить в состав передаваемых данных те данные, на которые ссылаются эти указатели, а затем передать их на удаленный компьютер. Такую работу приходится выполнять, поскольку указатель, применяемый в одном процессе, вряд ли действительно будет на что-то указывать после его передачи в другой процесс, особенно если вместе с ним не передаются данные, на которые он указывает. Автору нравится аналогия, в которой маршалинг рассматривается как операция, в которой берутся данные, представленные параметрами, переданные в процедуры в непосредственном виде, а не по ссылке, после чего обеспечивается передача этих данных получателю в удобной для него форме.

После вызова процедуры-заглушки в ней вызываются процедуры этапа прогона RPC, которые осуществляют действия по установлению связи с компьютером получателя, согласованию совместимого набора протоколов и передаче запроса на удаленный компьютер. После получения запроса компьютером получателя выполняется операция *демаршалинга* параметров (в частности, инкапсулированные в запрос данные снова распределяются в памяти по таким адресам, на которые затем можно ссылаться с помощью первоначальных указателей, а сами указатели корректируются таким образом, чтобы они ссылались на правильные адреса), после чего вызывается требуемая процедура с правильными значениями параметров. Между тем все эти операции остаются прозрачными для клиентского кода приложения — в нем просто вызывается процедура. После выполнения в удаленной системе такого вызова все указанные операции осуществляются в обратном порядке в целях возврата результатов в вызывающую функцию.

Асинхронные средства RPC

Операционная система Windows поддерживает не только синхронные, но и асинхронные средства RPC. После выполнения асинхронного вызова RPC вызывающее приложение продолжает выполняться. Как только завершается выполнение вызова, средства RPC операционной системы Windows обозначают как сигнальное некоторое событие, первоначально связанное с этим вызовом. В вызывающей функции можно обнаружить такое сигнальное состояние с помощью традиционных средств проверки сигнального состояния объекта привилегированного режима, таких как вызов функции `WaitForSingleObject`.

Библиотека этапа прогона RPC

Библиотека этапа прогона RPC находится в файле `Rpcrt4.dll`. Для проверки того, загружена ли эта библиотека в пространство процесса SQL Server, можно использовать утилиту TList. Но отметим, что эта библиотека DLL всегда загружена, независимо от того, была ли выполнена настройка конфигурации программы SQL Server для приема запросов на установление соединения с помощью мультипротокольной библиотеки Net-Library, в которой используется API-интерфейс RPC. Это связано с тем, что в исполняемом файле SQL Server предусмотрено непосредственное импортирование данной библиотеки DLL путем указания ее имени. (Читатель может убедиться в этом самостоятельно, используя инструментальные средства Depends или DumpBin, которые входят в поставку программы Visual Studio.)

Резюме

В программе SQL Server поддерживается целый ряд сетевых транспортных средств и протоколов. Важно иметь основное представление о том, как функционируют эти технологии и как они обычно используются в приложениях. Понимание того, как можно применить указанные технологии в своих собственных приложениях, позволяет лучше понять, как они используются в программе SQL Server.

С точки зрения организации ее работы программа SQL Server ничем не отличается от других приложений в сети. Взаимодействуя по сети с клиентом, эта программа осуществляет все свои операции с использованием сетевых API-интерфейсов, как и любое другое приложение. Подключаясь к программе SQL Server по сети, клиент выполняет операцию установления соединения с помощью стека сетевых протоколов, по такому же принципу, как и при вступлении во взаимодействие с сервером любого другого типа.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Функция `listen` API-интерфейса Winsock вынуждает вызывающую программу перейти в заблокированное состояние, которое продолжается до тех пор, пока от клиента не поступит запрос на установление соединения.
2. Какую функцию API-интерфейса Win32 вызывает сервер именованного канала, чтобы перейти в состояние ожидания поступления клиентского запроса на установление соединения?
3. Может ли дескриптор сокета использоваться с основными функциями ввода-вывода Win32, такими как `ReadFile`?
4. Подтвердите или опровергните следующее утверждение. Для работы с API-интерфейсом Winsock может применяться лишь протокол TCP/IP.
5. Опишите, как происходит маршалинг параметров RPC, и укажите, для чего он требуется.

6. Как называется верхний уровень эталонной модели OSI?
7. Подтвердите или опровергните следующее утверждение. Каждый уровень в модели OSI предназначен для предоставления доступа к службам данного уровня для более высоких уровней и создания абстрактного интерфейса к службам, предоставляемым более низкими уровнями.
8. Подтвердите или опровергните следующее утверждение. Модель организации сетей операционной системы Windows полностью соответствует модели OSI.
9. В чем состоит различие между байтовым режимом и строковым режимом в именованном канале?
10. Из чего должна состоять вторая часть имени именованного канала?
11. Какое сокращенное обозначение применяется для ссылки на текущий компьютер в имени именованного канала?
12. Подтвердите или опровергните следующее утверждение. Средства именованных каналов операционной системы Windows позволяют использовать другие средства операционной системы, такие как средства сетевого транспорта, но не позволяют использовать средства защиты Windows, поскольку основаны на коде, перенесенном из программы LAN Manager операционной системы OS/2.
13. Подтвердите или опровергните следующее утверждение. Назначение сетевого программного обеспечения состоит в том, чтобы получить клиентский запрос на предоставление ресурса, выполнить этот запрос на удаленном компьютере, содержащем требуемый ресурс, и возвратить результаты клиенту.
14. Подтвердите или опровергните следующее утверждение. Сетевой редиректор действует как своего рода распределенная файловая система.
15. Что означает термин *преобразование имен*?
16. Подтвердите или опровергните следующее утверждение. Средство именованных каналов операционной системы Windows функционирует исключительно на основе протокола NetBEUI.
17. Какое специальное имя файла присвоено устройству вывода на терминал в операционной системе Windows?
18. Назовите две функции, которые были упомянуты в этой главе как предназначенные для использования с клиентами и серверами Winsock без установления логических соединений.
19. Какой порядок байтов должен использоваться для передачи IP-адресов и номеров портов в функции API-интерфейса Winsock?
20. Какое общее название применяется для обозначения четырех нижних уровней эталонной модели OSI?
21. Какой основной механизм используется в технологии DCOM для управления выполнением кода по сети?
22. В какой системной библиотеке DLL реализована основная часть API-интерфейса Named Pipes?

23. На каком API-интерфейсе, унаследованном от операционной системы UNIX, основан API-интерфейс Winsock операционной системы Windows?
24. Сколько уровней предусмотрено в эталонной модели OSI?
25. Подтвердите или опровергните следующее утверждение. Одно из преимуществ API-интерфейса Named Pipes перед API-интерфейсом Winsock состоит в том, что Named Pipes может использоваться с асинхронным вводом-выводом, тогда как Winsock поддерживает только синхронный ввод-вывод.
26. Назовите три сетевых протокола, на которых может быть основано функционирование API-интерфейса Winsock в операционной системе Windows.
27. Каков самый низкий уровень эталонной модели OSI?
28. Какая библиотека DLL содержит реализацию API-интерфейса Winsock?
29. Какой флажок должен быть задан в приложении при создании именованного канала, чтобы этот канал мог участвовать в асинхронных операциях?
30. Какая библиотека DLL представляет собой библиотеку этапа прогона для средства RPC операционной системы Windows?

Технология COM

Технология COM находит исключительно широкое применение как в программе SQL Server, так и в других программах, эксплуатируемых в операционной системе Windows, поэтому без ее обсуждения в определенном объеме не будет полной ни одна книга, в которой рассматриваются основные технологии прикладного программирования. Автор не имеет ни времени, ни места для того, чтобы описать технологию COM в этой книге с такой степенью детализации, с какой он хотел бы, поэтому предлагает ознакомиться с такими книгами, как *Inside COM* Дэйла Роджерсона (Redmond, WA: Microsoft Press, 1997) и *Essential COM* Дона Бокса (Reading, MA: Addison-Wesley, 1998), чтобы узнать все подробности о том, как функционирует технология COM и как она может использоваться в приложениях. В настоящей главе автор приводит обновленное описание COM, взятое из его ранее изданных книг, и рассматривает технологию COM на более высоком уровне. Кроме того, в этой главе описано, каким образом программа SQL Server предоставляет доступ к некоторым из ее функциональных средств с помощью технологии COM и как в ней применяются внешние компоненты COM. Дополнительная информация о том, как получить доступ к COM-объектам из сценария на языке Transact-SQL, приведена в главе 15, посвященной технологии ODSOLE.

Краткий обзор

Разработчики, которым доводилось создавать приложения Windows, по-видимому, уже имеют представление о технологиях COM, OLE и ActiveX. Технология OLE первоначально рассматривалась как технология связывания и внедрения объектов (Object Linking and Embedding) и представляла собой первое поколение средств доступа и манипулирования объектами в операционной системе Windows, распространяющихся на многочисленные приложения. Идея этого подхода состояла в реализации технологии, в которой в центре внимания находится не приложение, а универсальный информационный объект – документ, который может включать объекты из одного приложения, а применяться в другом. В версии OLE 1.0 для обеспечения взаимодействия между объектами использовалась технология DDE (Dynamic Data Exchange – динамический обмен данными). DDE – это механизм межпроцессной связи с помощью сообщений, который основан на архитектуре передачи сообщений операционной системы Windows. Технология DDE характеризуется целым рядом ограничений (она отличается низким быстродействием, недостаточной гибкостью, сложностью для программирования и т.д.), поэтому разработчики второй версии OLE отказались от этой технологии.

Вторая версия OLE была полностью переработана для того, чтобы она зависела исключительно от технологии COM. Но даже несмотря на то, что технология

COM является более эффективной и быстродействующей по сравнению с DDE, при использовании технологии OLE все еще возникают значительные сложности. С чем это связано? Причина состоит в том, что OLE — это самая первая реализация технологии COM. Со времени ее создания разработчики приобрели весьма значительный опыт. Но несмотря на сказанное выше и независимо от ее реализации, технология OLE предоставляет функциональные возможности, которые являются очень мощными и разнообразными. Технология OLE остается громоздкой, недостаточно быстродействующей и сложной для программирования, но это не зависит от самой технологии COM; причиной является то, каким образом была создана сама технология OLE.

Технология ActiveX также основана на COM. ActiveX с самого начала была сосредоточена на создании и развертывании компонентов, которые могут применяться в Internet, и все еще остается предназначенной в основном для этой цели. По сути, ActiveX — это даже не одна технология, а множество технологий, которые в основном предназначены для поддержки интерактивного информационного наполнения (отсюда и обозначение "Active") на Web-страницах. Элементы управления ActiveX, которые прежде именовались элементами управления OLE или элементами управления OCX, представляют собой компоненты, которые можно вставлять в Web-страницы или в приложения Windows для обеспечения использования функциональных средств, оформленных в виде отдельного пакета и предоставленных сторонним разработчиком.

Технология COM — это основа, на которой формируются элементы управления OLE и ActiveX. С помощью технологии COM любой объект может предоставить свои функциональные возможности другим компонентам и приложениям. Технология COM не только регламентирует жизненный цикл объекта и способ предоставления его функциональных средств для внешнего мира, но и определяет, как эти методы предоставления доступа к функциональным средствам обеспечивают взаимодействие процессов в локальной и сетевой среде.

Одна из важнейших проблем современного программирования состоит в обеспечении доступа к классам, которые определены в коде отдельного программного компонента, для других приложений в форме, не зависящей от конкретного языка. В продуктах Microsoft для решения этой проблемы применяется технология COM. Эта технология предоставляет пользователям возможность применять объектно-ориентированные способы доступа к специализированным библиотекам DLL и пользоваться плодами труда других разработчиков, не нуждаясь в исходном коде или файлах заголовков.

Эпоха в программировании, предшествующая появлению технологии COM

Не так давно в области разработки программного обеспечения была вполне обычной такая практика, когда вместе с библиотеками сторонних разработчиков распространялся полный исходный код и/или файлы заголовков. Для подключе-

ния этих библиотек пользователи просто компилировали их (или включали в файлы заголовков библиотек) вместе с приложениями. Конечным результатом становился единственный исполняемый файл, который мог содержать код, полученный от многих разных поставщиков. А многие разработчики обычно использовали в своих продуктах одни и те же библиотеки сторонних разработчиков, поэтому в исполняемых файлах, включенных в состав многочисленных продуктов, могла присутствовать какая-то конкретная версия определенной библиотеки. В связи с этим исполняемые файлы, как правило, имели довольно значительные размеры, а совместное использование в них одного и того же кода либо полностью отсутствовало, либо было минимальным. В случае обновления одной из таких библиотек сторонних разработчиков требовалась повторная компиляция и/или повторное связывание, поскольку библиотеки встраивались непосредственно в исполняемый файл на этапе компиляции.

Вся эта ситуация изменилась со времени появления подхода, основанного на применении библиотек DLL. С тех пор прошло совсем немного времени, но все привыкли к тому, что сторонние разработчики программных продуктов поставляют только файлы заголовков и двоичные образы библиотек. Но при этом разработчики утратили возможность предлагать для развертывания свои продукты в виде единственного исполняемого файла, и в конечном итоге им приходилось предоставлять вместе со своим приложением целую коллекцию библиотек DLL, которая иногда достигала значительных размеров. А на этапе прогона приходилось предусматривать в приложении операций загрузки (либо неявной, либо явной) библиотек DLL, предоставленных сторонним поставщиком. По мере дальнейшего усложнения приложений нередко приходилось видеть такие исполняемые файлы, для которых требовались десятки библиотек DLL со сложными взаимозависимостями между ними.

ПРИМЕЧАНИЕ. В действительности именно на этом принципе основана работа самой операционной системы Windows, поскольку она представляет собой исполняемый файл, сопровождаемый большой коллекцией динамически загружаемых библиотек. А в приложениях Windows предусмотрены вызовы функций, доступ к которым предоставляется этими библиотеками DLL.

Такой подход оказался вполне приемлемым, но имел несколько недостатков. Одним из основных его недостатков было то, что интерфейсы к подобным библиотекам DLL не были объектно-ориентированными, и поэтому их было сложно расширять, а сами они были восприимчивыми к отказам, вызванным даже самыми незначительными изменениями в функциях, к которым предоставлялся доступ через эти интерфейсы. Если сторонний разработчик вводил в поставляемую им библиотеку всего лишь один новый параметр, такое изменение вполне могло нарушить работу кода во всех приложениях, в которых в настоящее время применялась эта библиотека, после установки ее модифицированной версии. В связи с этим большинство сторонних разработчиков для решения указанной проблемы применяли такой подход — просто создавали новую версию расширенной функции, включающую новый параметр (присваивая ей другое имя, часто с суффиксом "Ex" — сокращенное обозначение расширенной функции — или какой-то по-

добный суффикс), оставляя при этом старую версию функции неизменной. Результатом становилось появление интерфейсов уровня вызовов, которые очень быстро оказывались буквально неуправляемыми. Нередко приходилось сталкиваться с тем, что библиотеки сторонних разработчиков (и даже библиотеки самой операционной системы Windows) включали многочисленные версии вызова одной и той же функции; тем самым разработчики предпринимали попытки обеспечить совместимость новой версии с любой другой версией той же библиотеки, которая когда-либо существовала. Ситуация стала быстро выходить из-под контроля, усугубляясь в связи с тем фактом, что пользователи этих библиотек не имели простого, непосредственного метода определения того, какую из многих версий данной конкретной функции им следует использовать. Разработка кода с применением подобных интерфейсов превратилась в бесконечный эксперимент, в ходе которого приходилось перелистывать огромные тома руководств по API-интерфейсам и часто теряться в догадках – какая же версия правильная?

Еще одной важной проблемой, связанной с использованием этого подхода, было распространение многочисленных копий одной и той же библиотеки DLL на компьютере пользователя. В то время пространство на жестком диске было гораздо более дорогостоящим, чем в наши дни, поэтому поставщики программного обеспечения стремились избежать необходимости хранить многочисленные копии одной и той же библиотеки в разных местах системы конечного пользователя. К сожалению, решение этой проблемы, на котором в свое время остановились, не было в действительности достаточно хорошо продуманным. Дело в том, что было решено помещать библиотеки DLL, необходимые для разных приложений, в системный каталог Windows. Это позволяло решить проблему, связанную с наличием многочисленных копий одной и той же библиотеки DLL, но привело к появлению весьма значительного числа других проблем.

Главными среди них были неразрешимые, по сути, проблемы, связанные с наличием конфликтующих версий одной и той же библиотеки DLL. Если продукты поставщика А и поставщика Б зависели от разных версий библиотеки DLL, производимой поставщиком В, то возникала весьма существенная вероятность того, что функционирование продуктов одного из этих поставщиков будет нарушено после установки той версии библиотеки DLL, которая применяется другим поставщиком. Если интерфейс к библиотеке DLL изменялся от одной версии к другой даже совсем незначительно, существовала весьма высокая вероятность того, что по крайней мере одно из приложений будет функционировать неправильно (или даже вообще не станет функционировать), столкнувшись с той версией DLL, на работу с которой это приложение не рассчитано.

Еще одна проблема, связанная с сосредоточением библиотек DLL, была обусловлена теми сложностями, которые возникли в связи с использованием централизованной, но все равно оставшейся неуправляемой информации о конфигурации. Во времена, предшествующие появлению реестра Windows, нередко приходилось сталкиваться с такой ситуацией, что для каждого приложения применялся отдельный файл конфигурации (обычно с расширением `.INI`), а в некоторых приложениях было даже несколько файлов конфигурации. Эти файлы конфигурации могли включать обозначения путей к библиотекам DLL, которые

использовались в приложении, что приводило к дополнительному усложнению задачи устранения проблем поддержки версий DLL. Дело в том, что эти файлы конфигурации не контролировались самой операционной системой Windows. Поэтому невозможно было предотвратить такие ситуации, при которых приложение полностью стирало какой-либо необходимый файл конфигурации, помещало в него записи, которые могли нарушить работу других приложений, или полностью игнорировало файл конфигурации. Эти файлы .INI представляли собой просто текстовые файлы, которые могли либо использоваться, либо не использоваться в приложении, в зависимости от решения, принятого его разработчиком.

Новые средства, внедренные в операционную систему Windows и позволяющие упростить поиск библиотек DLL, были логичными и хорошо документированными. Но в приложении оставалась возможность использовать функцию `LoadLibrary` операционной системы Windows и загружать библиотеку DLL из любого каталога на жестком диске компьютера пользователя, выбранного разработчиком приложения. Поэтому, несмотря на возможность централизованного размещения библиотек DLL, нельзя было определить, от какого кода фактически зависит приложение. В приложении путь загрузки библиотеки мог быть выбран из файла конфигурации, о существовании которого больше никому не известно, кроме самого разработчика приложения. Либо даже в этом приложении мог быть просто предусмотрен поиск на жестком диске и загрузка той версии библиотеки, которая рассматривалась как наиболее подходящая. Между приложениями часто возникали тонкие взаимозависимости, в результате чего сами приложения становились весьма восприимчивыми к любым изменениям. Итак, в процессе поиска оптимального способа организации программ разработчики перебрали весь спектр подходов, начиная с тех дней, когда создавались раздутые исполняемые файлы, а совместное использование кода сводилось к минимуму или вообще отсутствовало, и заканчивая современной ситуацией, в которой любое приложение зависит от всех других, а инсталляция нового приложения часто приводит к нарушению работы существующего.

Появление технологии COM

В продуктах Microsoft эти проблемы были решены с помощью технологии COM. В первом приближении достаточно отметить, что технология COM — это способ предоставления интерфейса к библиотекам кода сторонних разработчиков, который обладает перечисленными ниже характеристиками.

- Объектно-ориентированный.
- Централизованный.
- Позволяющий учитывать версии.
- Независимый от языка.

Поскольку в технологии COM используется системный реестр, с ее появлением ушла в прошлое та эпоха, когда встречалась неуправляемая или неправильно

применяемая информация о конфигурации. При создании в приложении экземпляра COM-объекта (обычно с помощью вызова функции `CreateObject`) операционная система Windows обращается к системному реестру, чтобы определить местонахождение объекта на диске и загрузить этот объект. При этом не приходится задумываться над тем, где расположен этот объект, а наличие многочисленных копий одно и того же объекта не допускается — каждый COM-объект должен находиться в одном и только одном месте в системе.

ПРИМЕЧАНИЕ. В продуктах Microsoft недавно были введены концепции перенаправления и параллельного развертывания COM. Эти концепции позволяют успешно использовать в одной и той же системе несколько версий одного и того же COM-объекта. Такие функциональные средства обладают всеми признаками разработок, созданных под влиянием устаревших подходов, и могут применяться лишь в ограниченных условиях. (Например, перенаправление COM нельзя использовать для загрузки различных копий одного и того же объекта в разные Web-приложения одной и той же реализации сервера IIS. Хотя у пользователей может сложиться впечатление, что созданные по такому принципу Web-страницы представляют собой разные приложения, фактически эти страницы представляют собой результат функционирования только одного приложения — сервера IIS, а технология COM все еще налагает свои ограничения, согласно которым каждое конкретное приложение должно иметь только одну копию конкретной версии объекта.) В подавляющем большинстве приложений COM все еще соблюдаются стандартные ограничения, касающиеся применения различных версий COM.

Из этого не следует, что в системе нельзя иметь многочисленные версии одного и того же объекта. В технологии COM такая возможность предоставляется благодаря использованию нескольких интерфейсов — каждая новая версия объекта имеет свой собственный интерфейс, а также может вполне рассматриваться как полностью отдельный объект, по крайней мере с точки зрения тех, кто его применяет. При этом может предусматриваться или не предусматриваться совместное использование кода в разных версиях объекта. Разработчику приложения не приходится об этом задумываться, поскольку он просто разрабатывает код с учетом предоставляемого ему интерфейса.

Вкратце можно отметить, что интерфейс напоминает класс без тела или реализации. Он представляет собой программную конструкцию, которая определяет соглашение о предоставлении функциональных возможностей между тем, кто эти возможности предоставляет, и теми, кто ими пользуется. Реализуя интерфейс, автор объекта гарантирует, что клиенты, пользующиеся этим объектом, могут рассчитывать на то, что в объекте воплощен заданный набор функций. Пользователь объекта может разрабатывать код с учетом интерфейса, не задумываясь над тем, что фактически представляет собой сам объект и в чем состоят подробности его реализации. Если же автору объекта когда-либо потребуется усовершенствовать свой код таким образом, что может нарушиться работа зависящих от него пользовательских приложений, автор может просто определить новый интерфейс и оставить старый интерфейс неизменным.

Технология COM имеет свои ограничения (большинство из них устранено в инфраструктуре .NET), но она получила всеобщее распространение и была

в значительной степени стандартизирована. Сообщество разработчиков приняло технологию COM на вооружение, поэтому программа SQL Server включает механизм для работы с COM-объектами из сценариев на языке Transact-SQL.

Основная архитектура

Ниже перечислены основные элементы технологии COM.

- Интерфейсы (особенно `IUnknown`).
- Подсчет ссылок.
- Метод `QueryInterface`.
- Маршалинг (прямое и обратное преобразование из внутреннего представления во внешнее).
- Агрегирование.

Ниже каждый из этих элементов рассматривается отдельно.

Интерфейсы

Как было указано выше, с точки зрения объектно-ориентированного программирования, а также исходя из подхода, принятого в технологии COM, интерфейс — это механизм предоставления доступа к функциональным возможностям. Как правило, в объекте интерфейс используется для того, чтобы возможности этого объекта стали доступными для внешнего мира. Если в объекте применяется некоторый интерфейс, то принято говорить, что данный объект реализует этот интерфейс. Пользователи объекта могут взаимодействовать с интерфейсом, не зная о том, что фактически представляет собой объект, а один объект может реализовывать многочисленные интерфейсы.

Вообще говоря, для реализации интерфейса методы, доступ к которым предоставляется с помощью интерфейса, связываются с методами объекта. Для самого интерфейса память не требуется, и в действительности он просто определяет функциональные возможности, которые должен иметь реализующий его объект.

Каждый COM-интерфейс основан на `IUnknown` — фундаментальном COM-интерфейсе. Интерфейс `IUnknown` обеспечивает переход к другим интерфейсам, доступ к которым предоставляется объектом.

Каждый интерфейс имеет идентификатор интерфейса (Interface ID — IID), называемый также GUID (Globally Unique IDentifier — глобально уникальный идентификатор), который однозначно обозначает этот интерфейс. Благодаря использованию такого подхода упрощается поддержка многочисленных версий одного и того же интерфейса. Новая версия любого COM-интерфейса фактически становится отдельным интерфейсом со своим собственным идентификатором IID. Идентификаторы IID в стандартных интерфейсах ActiveX, OLE и COM определены заранее и остаются неизменными.

Подсчет ссылок

В отличие от инфраструктуры .NET и среды этапа прогона Java Runtime, технология COM не предусматривает автоматического сбора мусора. Задача уничтожения объектов, которые больше не требуются, возложена на разработчика. Для определения того, можно ли уничтожить объект, используются результаты подсчета количества ссылок на объект.

Подсчетом ссылок в интерфейсах COM-объекта управляют методы `AddRef` и `Release` интерфейса `IUnknown`. После получения клиентом указателя на COM-интерфейс (производный от интерфейса `IUnknown`), должен быть вызван метод `AddRef` применительно к этому интерфейсу. А после завершения клиентом использования интерфейса он должен вызвать метод `Release`.

В своей простейшей форме каждый вызов метода `AddRef` увеличивает значение переменной со счетчиком внутри объекта, обладающего этим методом, а каждый вызов метода `Release` приводит к уменьшению значения данной переменной. После того как значение счетчика достигает нуля, интерфейс становится несвязанным с какими-либо клиентами и может быть уничтожен.

Подсчет ссылок может быть также реализован так, чтобы подсчитывалась каждая ссылка на объект (а не на интерфейс, реализованный этим объектом). В подобном случае выполнение вызовов методов `AddRef` и `Release` делегируется (поручается) централизованной реализации средств подсчета ссылок. А после того как количество ссылок на объект достигает нуля, метод `Release` освобождает весь объект.

Метод `QueryInterface`

Фундаментальным механизмом COM, используемым для доступа к функциональным средствам объекта, является метод `QueryInterface` интерфейса `IUnknown`. Поскольку любой COM-интерфейс происходит от `IUnknown`, каждый COM-интерфейс имеет свою реализацию метода `QueryInterface`.

Метод `QueryInterface` запрашивает объект с использованием того идентификатора IID интерфейса, на который хочет получить указатель вызывающая функция. Если объект реализует требуемый интерфейс, то метод `QueryInterface` получает указатель на него, а также вызывает метод `AddRef`. Если же объект не реализует требуемый интерфейс, метод `QueryInterface` возвращает код ошибки `E_NOINTERFACE`.

Маршалинг

Автор любит сравнивать средства маршалинга с уличными регулировщиками или с лицами, сопровождающими важную персону, поскольку эти средства обеспечивают доставку данных из одного процесса в другой (маршалинг иногда применяется также для доставки данных из одного потока в другой в пределах одного и того же процесса). *Маршалинг* обеспечивает доступ других процессов к COM-интерфейсам, предоставляемым для доступа одним из объектов определенного процесса. Дело в том, что если какая-то структура данных содержит указатель на

фрагмент данных в пространстве адресов некоторого процесса, то этот указатель не может использоваться в других процессах, поскольку в них он уже ни на что не указывает. Маршалинг сводится к тому, что ссылка на данные (указатель) и сами данные копируются и представляются в таком формате, который обеспечивает их передачу в другой процесс (т.е. преобразуются из внутреннего представления во внешнее). А после получения в другом процессе данных, к которым была применена операция маршалинга, эта операция осуществляется в обратном порядке (иными словами, выполняется *демаршалинг*). Данные копируются в какие-то другие места в пространстве адресов нового процесса, а ссылки на эти данные устанавливаются с учетом нового расположения данных.

В технологии COM средства маршалинга используются для предоставления кода или применения кода, предоставленного средством реализации интерфейса, в целях преобразования параметров метода в формат, с помощью которого эти параметры могут передаваться из одного процесса в другой на одном и том же компьютере или проходить по сети на другие компьютеры, а также обратного преобразования этих параметров во время вызова метода. А после того как вызванный на выполнение метод возвратит результаты, данный процесс осуществляется в обратном направлении.

Маршалинг обычно не требуется, если интерфейс используется в том же процессе, в каком функционирует тот объект, который его предоставляет. Но для передачи параметров вызова метода и возврата полученных результатов из одного потока в другой все еще может потребоваться маршалинг.

Агрегирование

В некоторых ситуациях средству реализации объекта может потребоваться воспользоваться услугами, предоставляемыми еще одним объектом (например, созданным сторонним разработчиком), но при этом часто должно соблюдаться условие, чтобы этот второй объект функционировал как неотъемлемая часть первого. Для решения подобных задач в технологии COM предусмотрены средства агрегирования и включения.

Автор определяет принцип агрегирования как создание во включающем объекте экземпляра включаемого объекта в ходе создания самого включающего объекта, после чего включающий объект предоставляет доступ к интерфейсам включаемого объекта через свой собственный интерфейс. Одни объекты могут участвовать в агрегировании, а другие – нет. Для того чтобы участвовать в агрегировании, объект должен соответствовать определенному набору правил.

Практическое применение технологии COM

С точки зрения практики существуют два основных способа использования COM-объектов – раннее связывание и позднее связывание. Если в приложении применяются ссылки на объекты, которые могут быть разрешены на этапе компиляции, то, по сути, происходит раннее связывание объекта. Чтобы применить

раннее связывание объекта в приложении на языке Visual Basic, необходимо ввести ссылку на библиотеку, содержащую объект, в разрабатываемый проект, после чего создать конкретные экземпляры этого объекта с помощью оператора `Dim`. А чтобы выполнить раннее связывание объекта в таких инструментальных средствах, как Visual C++ и Delphi, необходимо импортировать библиотеку типов объекта и работать с предоставляемыми этим объектом интерфейсами. Но в том или ином случае в коде приложения непосредственно используются интерфейсы, предоставляемые объектом, как если бы это были интерфейсы, созданные самим разработчиком. Сам же объект может находиться совсем на другом компьютере, так что доступ к нему предоставляется с помощью технологии DCOM (Distributed COM) или с помощью операций маршалинга, выполняемых таким диспетчером транзакций, как Transaction Server или Component Services компании Microsoft. Вообще говоря, разработчику не приходится об этом задумываться; он просто разрабатывает код с учетом существующего интерфейса.

Если же ссылки на объект остаются неизвестными до этапа прогона, то к объекту применяется позднее связывание. В приложении на языке Visual Basic экземпляр объекта для позднего связывания создается с помощью вызова функции `CreateObject` и сохранения экземпляра объекта в переменной типа `Variant`. В приложении на языке Visual C++ необходимо получить указатель на интерфейс `IDispatch` объекта (все объекты автоматизации реализуют интерфейс `IDispatch`), после чего применять методы `GetIDsOfNames` и `Invoke` для вызова методов объекта, а также определения и задания его свойств.

Но поскольку при позднем связывании компилятор не имеет на этапе компиляции информации о том, на какие объекты будет сделана ссылка в приложении, то на этапе прогона могут возникать ошибки, связанные с неправильными вызовами методов или с обращениями к несуществующим свойствам. В этом состоит оборотная сторона позднего связывания — оно является более гибким, поскольку позволяет принимать на этапе прогона решение о том, какие экземпляры объектов должны быть созданы, и даже создавать экземпляры объектов, которые не существуют в системе разработчика, но позднее связывание более восприимчиво к нарушениям в работе. При его использовании можно легко допустить ошибку, выполняя позднее связывание объектов, поскольку среда разработки не может предоставить помощь на том же уровне, который возможен, когда в ней известны объекты, с которыми оперирует разработчик.

Кроме того, доступ к COM-объектам с помощью позднего связывания происходит медленнее по сравнению с ранним связыванием, причем иногда различие в быстродействии становится весьма существенным. Дело в том, что при позднем связывании программные идентификаторы ProgID необходимо преобразовать на этапе прогона в идентификаторы IID, а также выполнить поиск идентификаторов доступа для методов и свойств, предоставляемых интерфейсом, для того, чтобы они могли применяться в вызовах. Для этого требуется время, поэтому различия в скорости выполнения часто становятся весьма значительными.

После создания экземпляра объекта в приложении можно вызывать его методы и обращаться к его свойствам, как и при использовании любого другого объекта. Технология COM поддерживает понятие событий (хотя при этом приходится сталки-

ваться со значительно большими сложностями, чем следовало бы), поэтому в приложении можно также регистрировать заявки на получение сообщений о возникающих событиях и отвечать на события, касающиеся применяемых COM-объектов.

Модели многопотоковой поддержки

Объекты COM поддерживают две основные модели потоковой организации работы – однопотоковую апартаментную модель (Single-Threaded Apartment – STA), называемую также "апартаментно-потоковой", и многопотоковую апартаментную модель (MultiThreaded Apartment – MTA), называемую также "свободно-потоковой". Термин "апартаментная модель" не говорит о чем-либо слишком сложном, а просто помогает определить систему понятий, которая описывает взаимосвязи между потоками, объектами и процессами. Термин "апартаментный" основан на аналогии, согласно которой процесс рассматривается как жилой дом, а логический контейнер, в котором в рамках процесса существуют потоки и COM-объекты, рассматривается как апартаменты (или квартира), т.е. часть комнат, относящихся к жилому зданию. Таким образом, апартаментный контейнер – это просто один из логических контейнеров в рамках процесса. Как подсказывает эта аналогия, апартаментный контейнер может содержать множество потоков и/или объектов, в зависимости от его типа, а один процесс может иметь много отдельных апартаментных контейнеров.

Каждый объект и каждый поток может принадлежать только к одному апартаментному контейнеру. Кроме того, только потоки, относящиеся к некоторому апартаментному контейнеру, могут получать непосредственный доступ к объектам в этом контейнере; остальные потоки должны обращаться к COM-посредникам того или иного типа. Поток может определить свое пребывание в апартаментном контейнере (и в случае необходимости указать модель потоковой поддержки, которую желательно использовать) с помощью вызова функции инициализации COM, такой как `CoInitialize`, `CoInitializeEx` или `OleInitialize`.

В модели STA каждый апартаментный контейнер включает единственный поток и может содержать несколько объектов. В модели MTA апартаментный контейнер может включать несколько потоков и несколько объектов. С другой стороны, процесс может иметь несколько контейнеров STA, но только один контейнер MTA. Однако этот единственный контейнер MTA может сосуществовать с многочисленными контейнерами STA в одном и том же процессе.

Как уже было сказано выше, любой внепроцессный COM-сервер (т.е. исполняемый файл) определяет применяемую в нем модель многопотоковой поддержки с помощью вызова функции инициализации COM. При этом функция `CoInitializeEx` является одной из трех основных функций инициализации COM, которая позволяет задавать модель многопотоковой поддержки; вызов любой из функций `CoInitialize` и `OleInitialize` вынуждает систему применить модель STA. Вызов либо `CoInitialize`, либо `OleInitialize` в конечном итоге приводит к вызову функции `CoInitializeEx` с заданным в коде вызова указанием об использовании STA в качестве модели многопотоковой поддержки.

Модель многопотоковой поддержки для внутрипроцессного COM-сервера (т.е. библиотеки DLL) не задается с помощью вызова функции CoInitializeEx. Вместо этого данная модель задается с помощью ключа реестра, как показано ниже.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\InprocServer32\
➥ThreadingModel
```

Этот ключ может иметь значение Apartment, Free или Both. Если ключ не задан, то предполагается использование модели STA.

Как уже было сказано выше, только потоки, находящиеся в апартаментном контейнере, в котором был создан COM-объект, могут иметь непосредственный доступ к этому объекту. Другие потоки получают доступ к объекту через объекты-посредники. Напомним, что в каждом конкретном апартаментном контейнере STA может находиться только один поток, поэтому за синхронизацию доступа к объекту со стороны других потоков отвечают средства COM. Такая синхронизация обеспечивается с помощью средств обмена сообщениями Windows. Эта операционная система создает скрытое окно для каждого апартаментного контейнера и определяет посредников, которые должны передавать сообщения в апартаментный контейнер, к которому относится объект, для обращения к этому объекту. Сериализация доступа к объекту осуществляется с помощью обычных средств ведения очередей сообщений операционной системы Windows. В ответ на сообщения, направленные в скрытое окно апартаментного контейнера, вызываются методы объекта. По мере того как сообщения извлекаются из очереди сообщений (с помощью функций PeekMessage и GetMessage) и доставляются к получателю (с помощью функции DispatchMessage), оконная процедура для данного потока, которая реализована с помощью средств COM, вызывает соответствующие методы объекта. После выполнения вызова метода и появления необходимости возвратить результаты вызывающему потоку эти действия осуществляются в обратном порядке. В скрытое окно, относящееся к апартаментному контейнеру вызывающего потока, передаются сообщения для предоставления этому потоку результата (результатов) и указания на завершение выполнения затребованной функции. Прием этих сообщений осуществляется вызывающим потоком, который, в свою очередь, выполняет возврат с помощью вызова метода объекта-посредника; на этом вызов метода объекта, находящегося в другом апартаментном контейнере, заканчивается.

Если конфигурация некоторого компонента определена для работы с моделью STA, то объекты, к которым предоставляет доступ этот компонент, создаются в потоке Win32, в котором создан данный компонент. Другие потоки не могут получить доступ к экземплярам этих объектов.

Если же конфигурация некоторого компонента определена для работы с моделью MTA, то средства COM автоматически запускают хост-контейнер MTA и создают экземпляры объектов в этом контейнере. С другой стороны, если конфигурация модели многопотоковой поддержки определена как совместимая и с моделью STA, и с моделью MTA, объект создается в вызывающем контейнере STA.

Главный контейнер STA

Первый поток, в котором инициализируются средства COM с использованием модели многопотоковой поддержки STA, становится главным контейнером STA. Этот контейнер STA должен оставаться работоспособным до тех пор, пока в процессе не будет завершена вся работа COM, поскольку некоторые внутрипроцессные серверы всегда создаются в контексте главного контейнера STA.

Для технологии OLE требуется, чтобы был определен один поток, который должен отвечать на сообщения, относящиеся к контейнеру STA. Таковым становится первый поток, в котором вызвана функция `OleInitialize`, поэтому первый поток в процессе, в котором вызвана функция `OleInitialize`, становится главным контейнером STA для данного процесса.

Технология COM и программа SQL Server

Технология COM играет важную роль на многих этапах функционирования программы SQL Server. Прежде всего, если доступ к программе SQL Server осуществляется с помощью средств ADO или OLE DB, то для этого используется технология COM, поскольку средства ADO и OLE DB состоят из коллекций COM-объектов и COM-интерфейсов. Если для подключения к серверу применяется программа Enterprise Manager, то для этого также служит технология COM. Как будет показано в главе 15, работа программы Enterprise Manager основана на использовании коллекции COM-объектов, называемой SQL-DMO. А во время выполнения связанного запроса к серверу для этого используются средства доступа COM – OLE DB, которые, как уже было сказано, представляют собой компоненты COM. Работа с технологией COM осуществляется и при вызове некоторых команд Transact-SQL. Например, выполнение команды `BULK INSERT` основано на использовании COM-объекта, который загружается и предоставляется для доступа внутри сервера.

Работа со средствами DTS или объектами репликации ActiveX также осуществляется с помощью технологии COM. В сценариях ActiveX в программе SQL Server Agent, а также в аналогичных сценариях в пакетах DTS для определения языков сценариев и выполнения пользовательского кода используются COM-интерфейсы для сценариев ActiveX.

Кроме того, взаимодействие со средствами COM осуществляется и при обращении к пакету SQLXML с использованием средства доступа SQLXMLOLEDB или процедуры `sp_xml_preparedocument`. Как указывает само название процедуры SQLXMLOLEDB, она представляет собой средство доступа OLE DB. А в процедуре `sp_xml_preparedocument` (и в ее аналоге `sp_xml_removedocument`) используется MSXML – интерпретатор XML компании Microsoft, доступ к которому предоставляется таким приложениям, как SQL Server, с помощью COM-интерфейсов.

Кроме того, безусловно, с технологией COM приходится сталкиваться при создании COM-объектов из сценария на языке T-SQL с использованием хранимых процедур `sp_OA`. Как будет подробно описано в главе 15, функциональные средства `sp_OA` основаны на интерфейсе `IDispatch` технологии COM и взаимодей-

ствуют с ним таким же образом, как и простые инструментальные средства автоматизации, подобные VBScript. К тому же, во многих других модулях программы SQL Server либо используются средства COM, либо предоставляется доступ к функциональным возможностям этих модулей с помощью COM. Средства COM пронизывают все аспекты функционирования программы SQL Server, как и многих других сложных приложений Windows.

Резюме

Технология COM предоставляет независимый от языка механизм обеспечения для внешнего мира доступа к функциональным возможностям DLL или исполняемого файла. Эта технология основана на применении интерфейсов и концепции связывания – связывание приложения с COM-интерфейсами, которые в нем используются, может осуществляться по принципу либо раннего, либо позднего связывания.

С одной стороны, программа SQL Server предоставляет доступ к своим собственным функциональным средствам с помощью COM, с другой стороны, в этой программе применяются функциональные возможности, предоставляемые внешними компонентами COM. К некоторым примерам функциональных возможностей SQL Server, предоставляемых с помощью технологии COM, относятся применяемые в программе SQL Server средства OLE DB и собственное средство доступа OLE DB этой программы (SQLOLEDB); COM-объекты SQL-DMO, на которых основано функционирование программы Enterprise Manager; средство доступа OLE DB, которое предоставляет доступ к клиентским функциональным возможностям SQLXML (SQLXMLOLEDB). Примеры внешних COM-интерфейсов, используемых в программе SQL Server, включают MSXML; интерфейсы поддержки сценариев ActiveX; средства OLE DB, доступ к которым предоставляется с помощью связанных серверных запросов.

Вопросы для самопроверки

1. С каким интерфейсом взаимодействует приложение при обеспечении доступа к COM-объекту с помощью позднего связывания?
2. Какие три метода должны реализовывать все COM-интерфейсы?
3. От какого интерфейса в конечном итоге происходят все COM-интерфейсы?
4. Опишите, что происходит при передаче данных из одного процесса в другой с помощью средств маршалинга.
5. Какая библиотека COM используется в хранимой процедуре `sp_xml_preparedocument` программы SQL Server?
6. Подтвердите или опровергните следующее утверждение. Доступ к средствам COM ограничивается такими языковыми инструментальными средствами программирования Microsoft, как Visual Basic и Visual C++. Поскольку

COM – собственная технология компании Microsoft, невозможно создавать или получать доступ к COM-объектам из языков сторонних разработчиков.

7. Какой механизм используется в сочетании COM-объектами для слежения за тем, сколько ссылок в настоящее время указывают на данный конкретный объект?
8. Опишите назначение интерфейса `QueryInterface`.
9. Назовите технологию, которая позволяет дистанционно создавать экземпляры COM-объектов на другом компьютере.
10. Подтвердите или опровергните следующее утверждение. При доступе к объекту по принципу позднего связывания с использованием идентификатора объекта ProgID необходимо вначале преобразовать ProgID в идентификатор интерфейса, поскольку лишь после этого появляется возможность обратиться к интерфейсам, которые реализованы в объекте.

ГЛАВА 8

Язык XML

Автор включил в настоящую книгу главу со вступительными сведениями о языке XML по четырем причинам. Во-первых, он хотел обновить описание XML, приведенное в его последней книге, *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML* (Boston, MA: Addison-Wesley, 2002), и посчитал, что для этого настал удобный момент. Язык XML с того времени, как автор написал указанную книгу, получил дальнейшее развитие, поэтому автор хотел уточнить то, что было сказано об этом языке в свое время.

Во-вторых, по мнению автора, XML настолько глубоко проник во все сферы программирования и стал таким вездесущим, что его вполне можно назвать одной из основных технологий программирования. Он широко поддерживается в программе SQL Server, и на него опираются применяемые в этой программе технологии SQLXML, наряду с тем, как в других частях этого программного продукта применяются и поддерживаются такие технологии, как COM, совместно используемая память и сокеты Windows. Согласно прогнозам, в следующем выпуске SQL Server будет предоставляться еще большая поддержка XML и еще больше средств, основанных на этом языке. Язык XML за последние несколько лет достиг своего полного развития, и автор считает, что настало время признать этот факт и ввести язык XML в список основополагающих технологий, о которых должен знать каждый разработчик, желающий освоить такие современные, сложные приложения, как SQL Server. Приближается тот день, когда ни один специалист в области программирования не сможет быть успешно работающим практиком, не имея хотя бы элементарных знаний о языке XML.

В-третьих, в настоящее время средства поддержки XML стали одним из ключевых компонентов программы SQL Server. Поэтому, по мнению автора, этот язык вполне заслуживает того, чтобы рассматриваться в книгах вместе с программой SQL Server. Автор включил в настоящую книгу описание таких тем, как управление виртуальной памятью и синхронизация потоков, поскольку считает их существенно важными с точки зрения архитектуры и проекта программы SQL Server, а также потому, что, по его мнению, это позволяет лучше понять работу программы SQL Server. Кроме того, автор считает себя обязанным рассмотреть и средства языка XML. Дело в том, что средства этого языка также стали одним из ключевых составляющих элементов в архитектуре SQL Server. Изучение средств XML помогает лучше понять, как они используются в программе SQL Server, и дает ключ к постижению того, как спроектирован этот программный продукт. На применении средств XML основаны важнейшие части программы SQL Server, и это утверждение становится со временем все более очевидным. Как в книге, посвященной проектированию архитектуры программы SQL Server, заслуживают

описания такие средства операционной системы Windows, как сокеты и совместно используемая память, подобного описания заслуживают и средства XML.

В-четвертых, многие пользователи программы SQL Server плохо знакомы со средствами XML. Автор считает, что изучение средств XML, описанных в книге, которая в основном посвящена программе SQL Server, дает возможность тем специалистам, которым приходится интенсивно использовать указанную программу, приобрести ценные знания и навыки. Зная язык XML, вы сможете стать более успешным пользователем программы SQL Server, а понимая основные особенности архитектуры SQL Server, более подробно изучите не только сам этот программный продукт, но и применяемые в нем технологии. Изучение принципов работы средств XML является основой для понимания не только проекта и реализации поддержки средств XML в программе SQL Server, но и текущего состояния, а также будущих направлений развития этого программного продукта.

К сожалению, автор не имеет ни времени, ни места, чтобы обсудить средства XML настолько полно, насколько ему хотелось бы. Темам, относящимся к языку XML и семейству технологий, основанных на языке XML, посвящены целые книги. Автор попытается в этой главе рассмотреть наиболее важные вопросы, касающиеся XML, о которых необходимо знать, чтобы иметь возможность использовать средства программы SQL Server, связанные с языком XML. Из этого следует, что читателю придется дополнить изложенный здесь материал с помощью изучения специальной литературы (некоторые рекомендации по выбору этой литературы приведены в разделе "Дополнительные источники информации" ниже в данной главе).

Краткий обзор

Безусловно, благодаря World Wide Web язык HTML завоевал весь мир. Но несмотря на его популярность, язык HTML всегда имел и имеет целый ряд серьезных ограничений. Любой программист, перед которым возникает задача создания приложений для Web, начинает сталкиваться с этими ограничениями на самых первых этапах своей работы. Язык HTML достаточно хорошо подходит для решения задачи форматирования документов, к которым не предъявляются слишком строгие требования, но гораздо менее успешно позволяет решать более сложные задачи. Он никогда не был предназначен для описания структуры данных, но потребности деловых предприятий вынуждали использовать его именно для этой цели. А тот факт, что язык HTML приходилось применять для выполнения работы, для которой он не был предназначен, лишь сильнее подчеркивал многие его недостатки. В результате возникла необходимость в создании более мощного языка разметки, предназначенного для представления данных, а не вывода этих данных на экран; языка, в котором ничего не сказано о том, как форматировать данные, но предусмотрена возможность придавать этим данным контекстное значение.

Язык XML не только позволил устранить многие недостатки HTML, но и в целом стал основой для создания расширяемых приложений. Язык XML несложно осваивать тем, кто знаком с языком HTML, но является гораздо более мощным по сравнению с HTML. Язык XML далеко выходит за рамки простого языка раз-

метки; он представляет собой метаязык — язык, который может использоваться для определения новых языков. С помощью XML можно создать язык, специально приспособленный к нуждам данного конкретного приложения или производственной проблемной области, и использовать его для обмена данными с поставщиками, торговыми партнерами, заказчиками и любыми другими пользователями, применяющими язык XML.

Язык XML предназначен не для замены языка HTML, а для его дополнения. Язык XML не просто предоставляет возможность форматировать данные, но и позволяет определять их контекст. Дело в том, что если для данных определено контекстное значение, их отображение становится гораздо проще. Но отображение данных — лишь одна из многих операций, которые могут быть проделаны с данными, после того как они приобрели контекст. Эффективно отделив способы представления данных от способов их хранения и управления, мы открываем почти бесконечное количество возможностей для использования данных и обмена ими с другими заинтересованными лицами.

В этой главе рассматривается история языков разметки и показано, как впервые появился язык XML. Кроме того, в ней показано, как данные могут быть представлены на языке HTML, и приведено сравнение этого способа представления со способом представления на языке XML, позволяющее судить о том, насколько более совершенным является последний способ. Далее в настоящей главе приведены основные сведения о системе обозначений XML и описаны способы отображения данных XML путем их преобразования в формат HTML с помощью таблиц стилей XML. В ней также описаны способы проверки допустимости документов с использованием определений типов документов (Document Type Definition — DTD) и схем XML, затем обсуждаются некоторые нюансы каждого из этих способов. В конце главы кратко рассматривается объектная модель документа (Document Object Model — DOM) и показано, как она используется для манипулирования документами XML, представленными в виде объектов.

Недостатки языка HTML

Язык HTML предназначен, в частности, для форматирования документов. Он позволяет определить элементы, применяемые при выводе на экран, — названия, заголовки, шрифты, надписи и т.д. Но в целом этот язык предназначен для определения форматов представления. С его помощью очень удобно определять компоновку данных. А для описания этих данных или предоставления к ним общего доступа язык HTML подходит не так хорошо.

Разработчики Web-узлов сумели преодолеть многие ограничения языка HTML, применяя для этого некоторые исключительно новаторские способы. Но язык HTML все равно обладает серьезными недостатками и поэтому плохо подходит для создания сложных, открытых информационных систем. Некоторые из этих недостатков перечислены ниже.

- Язык HTML не является расширяемым. Каждый браузер поддерживает постоянный набор дескрипторов, и пользователю не разрешается вводить свои собственные дескрипторы.
- Язык HTML в основном предназначен для форматирования. Несмотря на то, что язык HTML позволяет достаточно успешно отображать данные, в этом языке данные применяются без какого-либо контекста. В случае изменения формата данных, представленных на языке HTML, к которым обращается некоторая программа, работа этой программы чаще всего нарушается.
- Документ HTML после его создания остается статическим и плохо поддается обновлению. Для преодоления этого недостатка используются DHTML (Dynamic HTML) и другие технологии, но язык HTML по самой своей сути никогда не был предназначен для оформления в нем непрерывно изменяющихся данных.
- Язык HTML предоставляет возможность создать только единственный вариант представления данных. Дело в том, что этот язык предназначен для вывода данных на экран, поэтому задача изменения способа представления данных, содержащихся в документе на этом языке, становится более сложной, чем могла бы быть. И этот недостаток позволяют до определенной степени устранить такие технологии, как DHTML, но в конечном итоге опыт работы с языком HTML показывает, что нужен другой язык разметки, на котором можно представить не только сами данные, но и сведения о них.
- Язык HTML плохо поддерживает семантическую структуру данных или вообще ее не поддерживает. В нем отсутствуют средства представления данных с помощью указания их смысла, а не их компоновки. Как уже было сказано выше, сильной стороной языка HTML является его способность представлять данные, однако и в этом отношении он иногда становится не совсем применимым.

Многие программисты с большим стажем работы знакомы с языком SGML (Standard Generalized Markup Language – стандартный обобщенный язык разметки) и считают, что он способен преодолеть многие недостатки языка HTML. Безусловно, язык SGML не имеет тех слабостей, которые присущи языку HTML, но характеризуется чрезвычайной гибкостью и поэтому является исключительно сложным. Для форматирования документов SGML применяется язык DSSSL (Document Style Semantics and Specification Language – язык определения семантики и спецификации стилей документа), который является очень мощным и гибким, но за использование его возможностей приходится платить – он чрезвычайно сложен в применении. Поэтому требуется язык, который подобен языку HTML по простоте, но обладает гибкостью языка SGML.

Краткая история языка XML

После того как началось стремительное развертывание Web, и в связи с этим объемы разработок, основанных на применении языка HTML, невероятно увеличились, очень быстро обнаружились многочисленные недостатки языка HTML.

Дело в том, что язык HTML — это приложение языка SGML, а поскольку возникла необходимость найти для Web более совершенный язык, то специалисты по языку SGML, чья работа была почти незаметна для широких кругов пользователей в течение многих лет, начали искать способ использования в Web языка SGML вместо HTML. К тому времени стало очевидно, что сам язык SGML слишком сложен для подобной задачи (и большинство разработчиков либо не было способно, либо не желало его использовать), поэтому нужно было найти какую-то альтернативу. Итак, следующая попытка создания языка Web осуществлялась в направлении поиска такого языка, который объединял в себе наилучшие характеристики HTML и SGML.

В середине 1996 года Джон Босак (Jon Bosak) из компании Sun Microsystems обратился в консорциум World Wide Web (W3C) с предложением сформировать комитет по использованию языка SGML в Web. Это предложение было поддержано представителем консорциума W3C Дэном Коннолли (Dan Connolly). Работу нового комитета возглавила, руководила и финансировала компания Sun, но фактически разработки проводились с участием не только представителя этой компании, Босака, но и специалистов других компаний, включая Тима Брея (Tim Bray), К. М. Сперберг-Маккуина (C. M. Sperberg-McQueen) и Джина Паули (Jean Paoli) из компании Microsoft. К ноябрю 1996 года комитет заложил основы упрощенной формы языка SGML, которая была не более сложной в изучении и использовании, чем HTML, но сохранила многие из самых лучших характеристик языка SGML. Это было зарождение языка XML в том виде, в каком он нам теперь известен.

Сравнение возможностей языков XML и HTML на примере

Язык XML позволяет разработчику создавать собственные дескрипторы. Эта возможность языка XML является настолько мощной и важной, что ей обязательно нужно посвятить несколько строк. Для тех, кто привык работать на языке HTML, такое нововведение весьма непривычно, поскольку язык HTML не позволяет определять собственные дескрипторы. Безусловно, различные поставщики браузеров расширяли язык HTML, вводя собственные специализированные дескрипторы, но такая возможность не предоставлялась рядовому разработчику. Он был вынужден использовать дескрипторы, поддерживаемые тем браузером, для которого создается документ HTML.

Так как же объявить новый дескриптор в документе XML? Ответ на этот вопрос весьма прост — этого в действительности не приходится делать. Достаточно лишь ввести новый дескриптор в документ. Для контроля над тем, какие дескрипторы являются допустимыми в документе XML, можно использовать документы DTD и схемы XML (дополнительная информация о каждом из этих средств приведена ниже), но общий вывод состоит в следующем — чтобы определить новый дескриптор в документе XML, достаточно просто ввести его в данный документ. Для этого не нужно применять операторы определения типа наподобие `typedef` или аналогичные конструкции.

Для того чтобы сравнить и сопоставить способы представления данных на языках HTML и XML, рассмотрим, как могут быть представлены одни и те же данные с использованием каждого из этих языков. В листинге 8.1 приведен пример документа HTML, в котором содержится кулинарный рецепт.

Листинг 8.1. Простой документ HTML

```
<!-- Первоначальный рецепт на языке HTML -->
<HTML>
<HEAD>
<TITLE>Henderson's Hotter-than-Hell Habanero Sauce</TITLE>
</HEAD>
<BODY>
<H3>Henderson's Hotter-than-Hell Habanero Sauce</H3>
Homegrown from stuff in my garden
  (you don't want to know exactly what).
<H4>Ingredients</H4>
<TABLE BORDER="1">
<TR BGCOLOR="#308030"><TH>Qty</TH><TH>Units</TH><TH>Item</TH></TR>
<TR><TD>6</TD><TD>each</TD><TD>Habanero peppers</TD></TR>
<TR><TD>12</TD><TD>each</TD><TD>Cowhorn peppers</TD></TR>
<TR><TD>12</TD><TD>each</TD><TD>Jalapeno peppers</TD></TR>
<TR><TD></TD><TD>dash</TD><TD>Tequila (optional)</TD></TR>
</TABLE>
<P>
<H4>Instructions</H4>
<OL>
<LI>Chop up peppers, removing their stems,
  then grind to a liquid.</LI>
<!-- Остальная часть страницы -->
</BODY>
</HTML>
```

Прочитав код HTML, приведенный в листинге 8.1, можно легко обнаружить, что ингредиенты этого блюда представлены в виде таблицы HTML. На рис. 8.1 показано, как данный документ выглядит в браузере.

Ниже перечислены некоторые положительные характеристики того способа представления, который используется в коде HTML для вывода на экран указанных данных.

- Код HTML является удобным для чтения — внимательный анализ этого кода позволяет понять, какие данные содержатся в соответствующем документе HTML.
- Документ HTML может быть отображен в любом браузере, даже в неграфическом.
- Для дополнительного управления форматированием может использоваться каскадная таблица стилей.

Но этот код содержит один действительно важный недостаток, который перевешивает все достоинства, если речь идет о разметке данных, — в нем нет ничего, что обозначало бы смысл какого-либо из его элементов. Данные, содержащиеся в документе, не имеют контекста. Несмотря на то что этот документ можно обра-

ботать в программе и извлечь элементы, содержащиеся в таблице, в подобной программе нельзя определить, каково назначение этих данных. Безусловно, в программу можно ввести описание искомых данных (отметив, что столбец 1 содержит данные о количестве — `Qty`, в столбце 2 указаны единицы измерения — `Units` и т.д.), но если формат страницы изменится, то работа приложения будет нарушена.

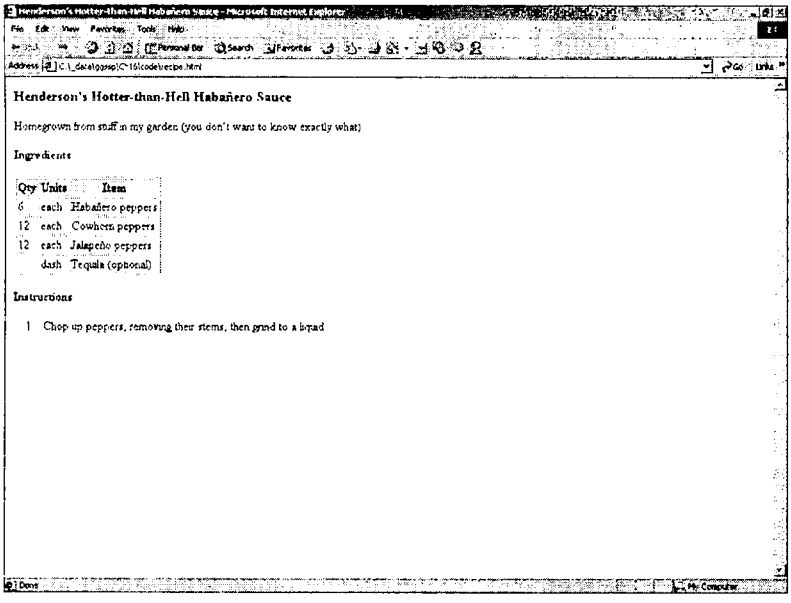


Рис. 8.1. Простая страница HTML, содержащая некоторые данные

Указанная проблема становится еще сложнее, если предпринимается попытка извлечь из документа данные и сохранить их в базе данных. Поскольку семантическая информация, относящаяся к данным, при преобразовании данных в код HTML отсекается, то приходится снова задавать эту информацию для того, чтобы сохранить данные в базе данных с учетом их смысла. Иными словами, приходится снова переводить данные с языка HTML на какой-то промежуточный язык, поскольку сам язык HTML не подходит для представления семантической информации.

Теперь рассмотрим те же данные, представленные в виде XML. Вполне очевидно, что применяемая разметка ничего не говорит о том, как должны быть отображены эти данные; вся разметка полностью посвящена описанию информационного наполнения. Код на языке XML приведен в листинге 8.2.

Листинг 8.2. Данные кулинарного рецепта, хранящиеся в виде кода XML

```
<?xml version="1.0" ?>
<Recipe>
  <Name>Henderson&apos;s Hotter-than-Hell Habanero Sauce</Name>
  <Description> Homegrown from stuff in my garden
    (you don&apos;t want to know exactly what).</Description>
  <Ingredients>
     <Ingredient>
```

```
      <Qty unit="each">6</Qty>
      <Item>Habanero peppers</Item>
     </Ingredient>
     <Ingredient>
      <Qty unit="each">12</Qty>
      <Item>Cowhorn peppers</Item>
     </Ingredient>
    <Ingredient>
      <Qty unit="each">12</Qty>
      <Item>Jalapeno peppers</Item>
    </Ingredient>
    <Ingredient>
      <Qty unit="dash" />
      <Item optional="1">Tequila</Item>
    </Ingredient>
  </Ingredients>
  <Instructions>
    <Step> Chop up peppers, removing their stems, then grind to a
liquid.</Step>
    <!-- Остальная часть страницы... -->
  </Instructions>
</Recipe>
```

Заметили различие? Дескрипторы в листинге 8.2 описывают сам рецепт, а не способ форматирования данных рецепта. Текст документа остается удобным для чтения, поэтому сохраняет всю простоту формата HTML, но данные теперь имеют контекст. Программа, которая интерпретирует подобный документ, сумеет точно определить, что представляет собой `Jalapeno`: это — элемент `Item` в составе одного из ингредиентов `Ingredient` в рецепте `Recipe`.

К тому же, несмотря на все удобство использования этого документа XML, по мнению автора, он фактически является также более удобным для чтения, чем документ HTML. Разработчикам языка XML удалось достичь своей цели и создать язык по меньшей мере столь же простой в использовании, как язык HTML, и вместе с тем более мощный на несколько порядков величины. Он описывает информацию в рецепте в тех терминах, которые подходят именно для описания рецептов, а не в тех терминах, которые определяют способ отображения рецептов на экране. Описание способов форматирования документов XML и инструментальных средств, наиболее подходящих для этой цели, приведено ниже.

Нюансы обозначений

Прежде чем продолжить обсуждение особенностей языка XML, необходимо на данном этапе определить некоторые понятия. Еще раз рассмотрим часть приведенного выше документа XML.

```
<Item optional="1">Tequila</Item>
```

В этом коде заслуживают внимания перечисленные ниже особенности.

1. `Item` — это имя дескриптора. Как и в языке HTML, в языке XML дескрипторы отмечают начало элемента. Элементы — это основные фрагменты головоломки XML. Документы XML состоят главным образом из элементов и атрибутов.

2. Кроме того, `optional` — это имя атрибута. Атрибут — это поле, которое содержит дополнительное описание элемента. Атрибут `optional` можно было бы назвать как-то иначе; выбор имени остается полностью прерогативой разработчика. Обратите внимание на то, что остальные элементы в листинге 8.2 не имеют этого атрибута.
3. Строка `"1"` — это значение данного необязательного атрибута, а вся часть текста документа, начиная от слова `optional` и заканчивая строкой `"1"`, представляет собой сам атрибут.
4. Дескриптор `</Item>` — это конечный дескриптор элемента `Item`.
5. Часть текста документа от дескриптора `<Item>` до дескриптора `</Item>` представляет собой элемент `Item`. Дескрипторы XML не всегда содержат текст. Они могут быть пустыми или содержать только атрибуты. Например, рассмотрим следующий фрагмент:

```
<Qty unit="dash" />
```

Здесь `Qty` — это имя элемента, а `unit` — его единственный атрибут. Косая черта в конце текста указывает на то, что сам элемент пуст и поэтому не требует закрывающего дескриптора. В целом приведенный выше дескриптор является сокращением от двух следующих дескрипторов:

```
<Qty unit="dash"></Qty>
```

Пустые дескрипторы могут иметь или не иметь атрибуты.

В документах XML не только соблюдаются эти простые правила определения структуры, но и учитывается необходимость более строгого форматирования, чем в документах HTML. Для того чтобы синтаксический анализатор XML имел возможность обрабатывать документы XML, эти документы должны быть формально правильными. В математике уравнения должны иметь некоторые определенные формы для того, чтобы их можно было считать правильными. Если же правила оформления уравнений нарушены, такие уравнения вряд ли можно применять в каких-либо полезных расчетах. К документам XML предъявляются аналогичные требования. Синтаксический анализатор будет иметь возможность обрабатывать документ XML лишь при том условии, что при создании этого документа соблюдаются определенные правила. Наиболее важные из этих правил перечислены ниже.

- Каждый документ должен иметь корневой элемент, который охватывает всю оставшуюся часть документа. Корневой элемент не обязан иметь имя. В приведенном выше примере корневым элементом является `Recipe`.
- Все открывающие дескрипторы должны иметь закрывающие дескрипторы либо в форме конечного дескриптора, либо в виде символа пустого дескриптора, описанного выше. В языке HTML это правило часто не соблюдается, и в таком случае попытка определить, где должен находиться закрывающий дескриптор, если он пропущен, обычно предпринимается браузером.
- Все дескрипторы должны быть правильно вложенными. Если дескриптор `Qty` содержится внутри дескриптора `Ingredient`, то необходимо закрыть дескриптор `Qty` и только после этого закрыть дескриптор `Ingredient`.

И это правило не является жестким требованием в документе HTML, но если правило вложенности нарушено в документе XML, то синтаксический анализатор XML отказывается обрабатывать такие дескрипторы.

- В отличие от текста элементов, значения атрибутов должны быть всегда заключены в одинарные или двойные кавычки.
- Символы <, > и " не могут быть представлены буквально; вместо них в документе следует использовать символьные сущности. Символьная сущность — это строка, которая начинается с символа амперсанда (&), заканчивается точкой с занятой (;) и ставится на место специального символа, что позволяет предотвратить нарушение работы синтаксического анализатора. Поскольку все указанные символы, <, > и ", имеют особый смысл в языке XML, их необходимо представлять с применением специальных символьных сущностей соответственно `<`, `>` и `&quot;`. В языке XML имеются еще две заранее определенные специальные символьные сущности, которые могут использоваться в тексте в случае необходимости — `&` и `&apos;`. Сущность `&` занимает место амперсанда. Поскольку символы амперсанда обычно обозначают в документе XML символьные сущности, то непосредственное использование их в данных может нарушить работу синтаксического анализатора. Аналогично сущность `&apos;` представляет одинарную кавычку, или апостроф. Поскольку значения атрибутов можно задавать в виде строки, заключенной в одинарные кавычки, появление в тексте документа непреобразованного в символьную сущность апострофа может нарушить работу синтаксического анализатора.
- Если в документе XML потребуется использовать символьные сущности, отличные от пяти определенных заранее, которые были описаны в предыдущем абзаце, необходимо их вначале объявить в документе DTD. Документы DTD будут описаны ниже.
- Имена элементов и атрибутов не должны начинаться с букв "`XML`" в любом регистре. Этот префикс зарезервирован в языке XML для собственного использования.
- В языке XML учитывается регистр. Это означает, что элемент с именем `Customer` рассматривается как элемент, отличный от элемента с именем `customer`.

Сравнение формально правильных и допустимых документов

Между формально правильными и допустимыми документами XML существует важное различие. Допустимый документ XML представляет собой формально правильный документ, к которому применяются дополнительные критерии проверки допустимости. Обеспечение формальной правильности документа — это только первый шаг в подготовке этого документа. Дело в том, что документ XML не только должен обеспечивать его успешный синтаксический анализ, но и представлять

определенные связи между данными и соответствовать требованиям, лишь при соблюдении которых он приобретает смысл. Документ, в котором нарушаются заданные правила, нельзя рассматривать как допустимый, даже если он является формально правильным. Например, рассмотрим фрагмент кода XML, показанный ниже.

```
<Car Name="Mustang" Make="Ford" Model="1966" LicensePlate="OU812">
  <Engine Type="Cleveland">341</Engine>
  <Engine Type="Winchester">302</Engine>
</Car>
```

Является ли он формально правильным? Разумеется, да. Но можно его считать допустимым? Скорее всего, нет. В нем описан автомобиль с двумя двигателями, но вряд ли можно найти такие легковые автомобили, в которых установлено два двигателя. Рассмотрим также следующий модифицированный фрагмент из приведенного выше примера документа.

```
<Ingredient>
  <Qty unit="each">12</Qty>
  <Qty unit="each">10</Qty>
  <Item>Jalapeno peppers</Item>
</Ingredient>
```

Имеет ли смысл включать две спецификации количества `Qty` для одного ингредиента? Скорее всего, нет. Хотя этот документ является формально правильным, его нельзя считать допустимым.

Правила проверки допустимости для документа определяются с помощью документов DTD и схем XML. Конструкции обоих типов рассматриваются в двух следующих разделах.

Определения типа документа

Синтаксические анализаторы XML подразделяются на два типа — с проверкой допустимости и без проверки допустимости. Синтаксический анализатор без проверки допустимости проверяет документ XML для определения того, является ли он формально правильным, после чего возвращает документ пользователю в виде дерева объектов. С другой стороны, синтаксический анализатор с проверкой допустимости проверяет, является ли документ формально правильным, а затем сопоставляет его с определением DTD или схемой для выяснения того, является ли этот документ допустимым. В настоящем разделе рассматривается первый из этих двух методов проверки, основанный на определении DTD.

Способ проверки допустимости документов с помощью определения DTD является немного устаревшим, но все еще достаточно широко используется. Определения DTD имеют сложный и довольно ограниченный синтаксис, но их еще можно встретить во многих реализациях XML. Со временем преимущественно применяемым инструментальным средством обеспечения проверки допустимости данных, по-видимому, станут схемы XML. Несмотря на это, до сих пор продолжает использоваться большой объем кода DTD (к тому же, с помощью определений DTD можно выполнить несколько таких операций, которые не поддерживаются схемами XML), поэтому все еще имеет смысл ознакомиться с определениями DTD.

Определения DTD позволяют формализовать и регламентировать дескрипторы, используемые в документе конкретного типа. Поскольку сам язык XML позволяет использовать в документе практически любые дескрипторы, которые только потребуются, при условии, что сам документ остается формально правильным, необходимо иметь в своем распоряжении какое-то средство, позволяющее регламентировать структуру документа, с помощью которого можно убедиться в том, имеет ли смысл данный документ. Первой попыткой достичь этой цели стали определения DTD. А поскольку определения DTD позволяют указать, какие дескрипторы могут и не могут применяться в документе, а также задать определенные характеристики этих дескрипторов, то определения DTD, кроме всего прочего, используются для описания новых диалектов XML, формализованных подмножеств дескрипторов XML и правил проверки. Определения DTD явились самым первым средством, которое оправдало наличие буквы X в аббревиатуре XML (а эта буква характеризует язык XML как расширяемый), поэтому определения DTD стали тем средством, с помощью которого разрабатывались новые приложения XML.

Рассмотрим определение DTD для приведенного выше примера рецепта. Пример подобного определения приведен в листинге 8.3.

Листинг 8.3. Определение DTD для данных рецепта

```
<!-- Recipe.DTD, пример определения DTD для файла recipe.xml -->
<!ELEMENT Recipe (Name, Description?, Ingredients?,
  Instructions?, Step?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Ingredients (Ingredient)*>
<!ELEMENT Ingredient (Qty, Item)>
<!ELEMENT Qty (#PCDATA)>
<!ATTLIST Qty unit CDATA #REQUIRED>
<!ELEMENT Item (#PCDATA)>
<!ATTLIST Item optional CDATA "0">
<!ELEMENT Instructions (Step)+>
<!ELEMENT Step (#PCDATA)>
```

Это определение DTD регламентирует некоторые характеристики документа, заслуживающие упоминания. Во-первых, следует отметить самую верхнюю строку этого файла, отличную от комментария (которая обозначена полужирным шрифтом). Она обозначает элементы, которые могут быть представлены с помощью документа, сопровождаемого этим определением DTD. Знак вопроса после элемента указывает на то, что данный элемент является необязательным.

Во-вторых, интерес представляют обозначения #PCDATA. Они указывают на то, что элемент или атрибут может содержать символьные данные и ничто иное.

В-третьих, следует отметить, что в данном тексте применяется обозначение #REQUIRED. Оно указывает на то, что атрибут `unit` элемента `Qty` является обязательным. В документах, для которых применяется это определение DTD, нельзя опускать указанный атрибут.

В-четвертых, обратите внимание на применяемое по умолчанию значение, заданное для атрибута `optional` элемента `Item`. Как показывает само имя этого

атрибута (optional — необязательный), его можно не задавать в документе и не показывать. Более того, если в каких-то элементах этот атрибут не задан, то по умолчанию применяется его значение, равное "0".

Как показывает листинг 8.3, синтаксис определения DTD — это не диалект XML, а самостоятельный язык, не очень удобный для восприятия. Именно поэтому вместо определений DTD все чаще используются схемы XML. Сами схемы XML рассматриваются ниже.

Чтобы связать определение DTD с документом, можно воспользоваться элементом объявления типа документа, находящимся в верхней части документа (непосредственно вслед за строкой <?xml...>). Объявление типа документа может содержать либо встроенную копию DTD, либо ссылку на имя файла, в котором содержится это определение, заданную с помощью универсального идентификатора ресурса (Uniform Resource Identifier — URI). Соответствующая ссылка в документе recipe.xml может выглядеть примерно следующим образом:

```
<!DOCTYPE Recipe SYSTEM "recipe.dtd">
```

Ниже еще раз приведен рассматриваемый документ с включенной в него строкой со ссылкой на DTD (листинг 8.4).

Листинг 8.4. Документ recipe.xml со включенной в него строкой со ссылкой на определение DTD

```
<?xml version="1.0" ?>
<!DOCTYPE Recipe SYSTEM "recipe.dtd">
<Recipe>
  <Name>Henderson&apos;s Hotter-than-Hell Habanero Sauce</Name>
  <Description> Homegrown from stuff in my garden
    (you don&apos;t want to know exactly what).</Description>
  <Ingredients>
    <Ingredient>
      <Qty unit="each">6</Qty>
       <Item>Habanero peppers</Item>
    </Ingredient>
    <Ingredient>
       <Qty unit="each">12</Qty>
      <Item>Cowhorn peppers</Item>
    </Ingredient>
    <Ingredient>
       <Qty unit="each">12</Qty>
       <Item>Jalapeno peppers</Item>
    </Ingredient>
    <Ingredient>
      <Qty unit="dash" />
      <Item optional="1">Tequila</Item>
    </Ingredient>
  </Ingredients>
  <Instructions>
    <Step> Chop up peppers, removing their stems, then grind to a
      liquid.</Step>
    <!-- Остальная часть страницы... -->
  </Instructions>
</Recipe>
```

Для проверки допустимости данных в соответствии с определением DTD может применяться целый ряд средств. В частности, при использовании браузера Internet Explorer 5.0 или более поздней версии можно применить встроенную программу проверки допустимости документа по определению DTD компании Microsoft, просто загрузив документ XML в браузер, щелкнув на нем правой кнопкой и выбрав из всплывающего меню команду Validate. Кроме того, разработан целый ряд инструментальных средств с графическим интерфейсом и интерфейсом командной строки, которые могут использоваться для той же цели. Несколько из них указаны в списке, который можно найти на узле W3C по адресу `http://www.w3c.org`.

Схемы XML

Как было указано выше, в настоящее время определения DTD считаются несколько устаревшими. Причина этого состоит в том, что теперь появилась более новая и совершенная технология проверки допустимости документов XML. Она называется XML Schema. В отличие от определений DTD, для формирования документов XML Schema используется язык XML. Схемы XML состоят из элементов и атрибутов, как и документы XML, для проверки допустимости которых они применяются. Схемы XML имеют также целый ряд других преимуществ перед определениями DTD, включая перечисленные ниже.

- Определения DTD не позволяют управлять тем, какого рода информация может содержаться в данном конкретном элементе или атрибуте. Для удовлетворения информационных потребностей большинства предприятий недостаточно просто иметь возможность указать, что в некотором элементе хранится текст. Дело в том, что часто требуется определить, какой формат должен иметь этот текст, а также, содержит ли данный текст, допустим, дату или число. Спецификация XML Schema предоставляет широкую поддержку для средств управления областью определения данных.
- Определения DTD поддерживают только 10 стандартных типов данных. А спецификация XML Schema поддерживает свыше 44 базовых типов данных и предоставляет возможность создавать собственные типы данных.
- Все объявления в любом определении DTD являются глобальными. Это означает, что в определении нельзя объявить несколько элементов с одним и тем же именем, даже если они применяются в совершенно разных контекстах.
- Синтаксис DTD отличен от синтаксиса XML, поэтому определения DTD требуют специальной обработки. Эти определения нельзя обрабатывать с помощью синтаксического анализатора XML. В результате документы, к которым прилагаются определения DTD, становятся более сложными, что может привести к замедлению их обработки.

Полное описание спецификации XML Schema выходит за рамки настоящей книги, но мы все равно должны коснуться некоторых ее основных особенностей. В листинге 8.5 приведена схема проверки допустимости для документа `recipe.xml`, который был сформирован выше.

Листинг 8.5. Схема XML для документа `Recipe`

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
    elementFormDefault="qualified">
  <xsd:element name="Recipe">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Name" type="xsd:string"/>
        <xsd:element name="Description" type="xsd:string"/>
        <xsd:element name="Ingredients">
          <xsd:complexType>
          <xsd:sequence>
              <xsd:element name="Ingredient"
                 maxOccurs="unbounded">
               <xsd:complexType>
                 <xsd:sequence>
                   <xsd:element name="Qty">
                     <xsd:complexType>
                       <xsd:simpleContent>
                         <xsd:restriction base="xsd:byte">
                             <xsd:attribute name="unit"
                                 use="required">
                              <xsd:simpleType>
                                <xsd:restriction
                                      base="xsd:NMTOKEN">
                                   <xsd:enumeration value="dash"/>
                                   <xsd:enumeration value="each"/>
                                   <xsd:enumeration value="dozen"/>
                                   <xsd:enumeration value="cups"/>
                                   <xsd:enumeration value="teasp"/>
                                   <xsd:enumeration value="tbls"/>
                                </xsd:restriction>
                              </xsd:simpleType>
                            </xsd:attribute>
                         </xsd:restriction>
                       </xsd:simpleContent>
                     </xsd:complexType>
                   </xsd:element>
                   <xsd:element name="Item">
                     <xsd:complexType>
                       <xsd:simpleContent>
                         <xsd:restriction base="xsd:string">
                            <xsd:attribute name="optional"
                                type="xsd:boolean"/>
                         </xsd:restriction>
                       </xsd:simpleContent>
                     </xsd:complexType>
                   </xsd:element>
                 </xsd:sequence>
               </xsd:complexType>
             </xsd:element>
           </xsd:sequence>
         </xsd:complexType>
        </xsd:element>
        <xsd:element name="Instructions">
```

```
            <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Step" type="xsd:string"/>
             </xsd:sequence>
           </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

На первый взгляд эта схема кажется довольно громоздкой. В частности, она намного длиннее по сравнению с определением DTD, которое рассматривалось выше. Но это не такой уж большой недостаток. Основная часть этого документа схемы состоит из открывающих и закрывающих дескрипторов, а сама схема не является слишком сложной.

Прежде всего следует отметить, что каждому из элементов и атрибутов в документе XML поставлен в соответствие тип данных. При проверке допустимости документа по этой схеме каждый фрагмент данных в документе проверяется для определения того, является ли он допустимым применительно к назначенному ему типу данных. Если это требование не соблюдается, то документ не проходит проверку допустимости.

Кроме того, рассмотрим атрибут `maxOccurs`. С помощью схемы можно определить количество вспомогательных свойств элемента, включая то, каково максимальное (или минимальное) количество вхождений некоторого элемента в документ. По умолчанию значения `minOccurs` и `maxOccurs` равны `1`. Чтобы обозначить некоторый элемент как необязательный, достаточно присвоить его атрибуту `minOccurs` значение `0`.

Далее рассмотрим элементы `xsd:enumeration`, которые относятся к атрибуту `unit`. В схеме XML можно задать список допустимых значений для элемента или атрибута. Если будет предпринята попытка предъявить на проверку документ, в котором элемент или атрибут содержит значение, не заданное в этом списке, то проверка допустимости документа оканчивается неудачей.

Наконец, отметим, что в атрибуте `optional` элемента `Item` применяется новый тип данных. Автор заменил целочисленное значение этого атрибута логическим значением, которое представляет собой один из стандартных типов данных, поддерживаемых спецификацией XML Schema. Автор хотел этим показать, какой чрезвычайно широкий набор типов данных предлагает спецификация XML Schema. Важно также понять, что существует возможность создавать новые типы, расширяя существующие. Кроме того, можно создавать сложные типы, т.е. типы элементов, содержащих другие элементы и атрибуты. В приведенной выше схеме примером сложного типа является тип данных `Qty`, а также типы `Ingredients` и `Instructions`. По определению сложный тип данных имеет любой элемент схемы, который содержит другие элементы или атрибуты.

У читателя может возникнуть вопрос, как связать схему с документом XML. Для этого достаточно ввести ряд атрибутов в корневой элемент документа. На-

пример, после связывания со схемой корневой элемент документа `recipe.xml` должен выглядеть примерно таким образом:

```
Recipe xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\Ch08\code\recipe.xsd">
```

Первый атрибут позволяет предоставить доступ в документе к элементам пространства имен `xsi` (сокращение от XML Schema Instance — экземпляр схемы XML). Пространство имен — это коллекция имен, которые идентифицируются с помощью ссылки на некоторый идентификатор URI. Разработчик может определить собственное пространство имен или поступить так, как было сделано в предыдущем фрагменте кода, и сослаться на пространство имен, определение которого находится на Web-узле W3C. Пространство имен XML предоставляет возможность задавать в приложении область определения имен для того, чтобы имена, объявленные в разных частях приложения, не конфликтовали друг с другом. Аналогичный подход применяется также во многих других технологиях программирования. Но в отличие от традиционных пространств имен, имена в пространстве имен XML не обязаны быть уникальными. Автор не объясняет причины того, с чем это связано, а лишь подчеркивает, что пространство имен задает область определения имен, используемых в документе XML. В данном конкретном случае пространство имен предоставляет доступ в пространстве имен `xsi`, в котором находятся элементы экземпляра схемы XML. Поскольку ссылка на пространство имен осуществляется таким способом, в документе можно использовать элементы экземпляра схемы XML, обозначая их префиксом `xsi:`.

Второй атрибут описывает местонахождение документа XML Schema. Он представляет собой документ, указанный выше, и содержит информацию схемы для рассматриваемого документа.

После определения всех этих атрибутов можно воспользоваться инструментальными средствами, поддерживающими спецификацию XML Schema, для проверки допустимости документа с помощью схемы, указанной одним из атрибутов.

Преобразование документа XML в документ HTML с использованием таблицы стилей

Для преобразования документов HTML широко используются каскадные таблицы стилей. По такому же принципу для преобразования документов XML применяется язык XSLT (Extensible Stylesheet Language Transformations — преобразования расширяемого языка таблиц стилей). Этот язык позволяет преобразовывать документы XML из одного формата документа в другой, преобразовывать документы на других диалектах XML или преобразовывать документы, имеющие полностью отличные от XML форматы, такие как PostScript, RTF и TeX.

Наиболее удобной особенностью языка XSLT является то, что он сам определен на языке XML. Любой документ XSLT представляет собой обычный документ XML.

Читатель может спросить: как это может быть; разве при этом не возникают проблемы, связанные с наличием циклических ссылок? Нет, такие ссылки не возникают. Дело в том, что язык XSLT — это просто еще один диалект языка XML. Современные синтаксические анализаторы XML являются достаточно интеллектуальными для того, чтобы определить способ использования инструкций, закодированных в документе XSLT (в виде обычных дескрипторов XML, атрибутов и тому подобного), для преобразования документа XML или описания структуры другого документа.

Таблица стилей XSLT — это документ XML, состоящий из ряда закодированных правил, называемых *шаблонами*, которые применяются к другому документу XML для выработки третьего документа. Эти шаблоны оформляются на языке XML с использованием специальных дескрипторов, имеющих определенный смысл. Каждый раз, когда какой-то шаблон совпадает с некоторым фрагментом исходного документа XML, формируется новая структура как результат применения шаблона. Такая структура часто представляет собой код HTML, как в приведенном ниже примере, но язык XSLT может также применяться для формирования результатов с другой структурой.

В листинге 8.6 приведена таблица стилей XSLT, которая преобразует рассматриваемый документ `recipe.xml` в код HTML, который весьма напоминает код HTML, сформированный вручную и приведенный выше в данной главе (листинг 8.1).

Листинг 8.6. Таблица стилей XSLT, предназначенная для преобразования рассматриваемого документа XML в документ HTML

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <HEAD>
  <TITLE>Henderson&apos;s Hotter-than-Hell Habanero Sauce</TITLE>
  </HEAD>
  <body>
    <H3>Henderson&apos;s Hotter-than-Hell Habanero Sauce</H3>
      Homegrown from stuff in my garden
      (you don&apos;t want to know exactly what).
      <H4>Ingredients</H4>
      <table border="2">
       <tr BGCOLOR="#00FF00">
        <TH>Qty</TH>
        <TH>Units</TH>
        <TH>Item</TH>
       </tr>
      <xsl:for-each select="Recipe/Ingredients/Ingredient">
      <tr>
        <td><xsl:value-of select="Qty"/></td>
        <td><xsl:value-of select="Qty/@unit"/></td>
        <td><xsl:value-of select="Item"/></td>
       </tr>
      </xsl:for-each>
    </table>
<P/>
```

```
      <H4>Instructions</H4>
      <OL>
     <xsl:for-each select="Recipe/Instructions">
       <LI><xsl:value-of select="Step"/></LI>
     </xsl:for-each>
     </OL>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

В этой таблице стилей выполняется несколько интересных действий. Прежде всего заслуживает внимания элемент `xsl:template match="/"`. Как было сказано выше, преобразования XSLT осуществляются путем применения шаблонов к конкретным частям документа XML. Атрибут `match` указанного элемента определяет, к какой части документа должен быть применен этот шаблон, с помощью синтаксиса языка, известного под названием XML Path (XPath). В данном случае рассматриваемым элементом является корневой элемент. Поэтому в данной таблице стилей сказано: "Найти корневой элемент документа, а после того, как он будет найден, вставить следующий текст во выходной документ". После обработки указанного шаблона формируется несколько строк стандартного кода HTML, которые соответствуют заголовку Web-страницы.

Заслуживает также внимания префикс `xsl:` в элементе шаблона. Он ссылается на пространство имен `xsl`. Именно в пространстве имен `xsl` определен элемент шаблона и другие имена с префиксом `xsl:`. Благодаря введению ссылки на пространство имен префикс `xsl:` становится доступным в документе, что позволяет ссылаться в документе на указанные имена. Ссылка на идентификатор URI находится в верхней части таблицы стилей. Она имеет следующую форму:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Затем следует отметить, какая информация заголовка таблицы HTML формируется с помощью данной таблицы стилей. Она содержит три набора дескрипторов <TH> языка HTML, которые образуют заголовки столбцов для данной таблицы. Полученная таким образом часть кода совпадает с аналогичной частью кода в оригинальном документе HTML, созданном ранее.

Наиболее интересной частью этого документа является осуществляемая в нем циклическая обработка. Именно в этом проявляется истинная мощь языка XSLT. Обратите внимание на первый цикл `xsl:for-each` (выделенный полужирным шрифтом). Цикл `for-each` языка XSLT выполняет именно то, о чем говорит его название (*для каждого*) — он осуществляет итерации в коллекции узлов, находящихся на одном и том же уровне в документе. Базовый узел, с которого начинается выполнение цикла, обозначается с помощью атрибута `select`. В данном случае ведется обработка узла `Recipe/Ingredients/Ingredient`. Как и для описанного выше атрибута `match`, в данном случае задан путь к узлу XPath, к которому требуется получить доступ. Это означает, что выполняется цикл по компонентам рецепта. Для каждого найденного компонента формируется новая строка в таблице.

Обратите внимание на то, какой способ применяется для оформления ссылок на узлы в каждом элементе Ingredient. Для вставки значения каждого поля в каждый ингредиент, который встречается в цикле, используется элемент xsl:value-of. С другой стороны, для доступа к атрибуту unit элемента Qty применяется синтаксис атрибута XPath, /@name, где name — атрибут, к которому должен быть получен доступ.

Заслуживает также внимания дескриптор абзаца <P/>, который следует за кодом цикла. В традиционном синтаксисе языка HTML разрешено задавать открывающий дескриптор абзаца без соответствующего ему закрывающего дескриптора, а в синтаксисе языка XML это не разрешено. Из этого следует важный вывод — код HTML, предоставляемый как образец для формирования таблицы стилей, должен быть формально правильным. Это означает, что он должен соответствовать правилам, регламентирующим структуру формально правильного документа XML. Напомним, что таблица стилей является документом XML в полном смысле этого понятия. Таблица стилей должна быть формально правильной, так как в противном случае нельзя будет осуществить ее синтаксический анализ.

Рассматриваемый код завершается еще одним циклом for-each. В этом цикле создается список элементов Step в каждом элементе Instructions. Обратите внимание на то, как используются дескрипторы <OL> (Ordered List — упорядоченный список) и <LI> (List Item — элемент списка) языка HTML. Эти дескрипторы действуют так же, как и в стандартном языке HTML, — формируют нумерованный список.

Рассматриваемая таблица стилей может применяться для преобразования документа recipe.xml с помощью нескольких способов. В частности, для этого может использоваться автономная программа преобразования XSLT компании Microsoft, программа преобразования XSLT стороннего разработчика или та программа преобразования, которая встроена в используемый браузер, если этот браузер поддерживает прямые преобразования XSLT. Дополнительная информация на эту тему приведена ниже, в подразделе "Инструментальные средства", но сам автор использует встроенную программу преобразования XSLT браузера Internet Explorer. Для этого необходимо добавить элемент <?xml-stylesheet> непосредственно в сам документ XML сразу после дескриптора <?xml version>. Ниже показан такой элемент в окончательном виде.

```
<?xml-stylesheet type="text/xsl" href="recipe3.xsl"?>
```

Вполне очевидно, что данный элемент содержит атрибут href, который ссылается на таблицу стилей с использованием формата URI. После этого при каждом просмотре данного документа XML в браузере Internet Explorer к этому документу автоматически применяется таблица стилей, и происходит преобразование документа. Код HTML, сформированный с использованием таблицы стилей, приведен в листинге 8.7.

Листинг 8.7. Код HTML, сформированный в результате преобразования

```
<html>
<HEAD>
<TITLE>Henderson's Hotter-than-Hell Habanero Sauce</TITLE>
</HEAD>
```

```
<body>
<H3>Henderson's Hotter-than-Hell Habanero Sauce</H3>
Homegrown from stuff in my garden
(you don't want to know exactly what).
<H4>Ingredients</H4>
<table border="2">
<tr BGCOLOR="#00FF00">
<TH>Qty</TH>
<TH>Units</TH>
<TH>Item</TH>
</tr>
<tr>
<td>6</td>
<td>each</td>
<td>Habanero peppers</td>
</tr>
<tr>
<td>12</td>
<td>each</td>
<td>Cowhorn peppers</td>
</tr>
<tr>
<td>12</td>
<td>each</td>
<td>Jalapeno peppers</td>
</tr>
<tr>
<td></td>
<td>dash</td>
<td>Tequila</td>
</tr>
</table>
<P />
<H4>Instructions</H4>
<OL>
<LI>Chop up peppers, removing their stems, then grind to a
liquid.</LI>
</OL>
</body>
</html>
```

На рис. 8.2 показано, какое изображение формируется в окне браузера при обработке листинга 8.7.

Безусловно, приятно иметь возможность преобразовать документ XML в формально правильный документ HTML, который совпадает с первоначальным примером, но фактически интерес представляет не это. Ведь если бы задача состояла лишь в создании документа HTML, то было бы проще сразу разработать его вручную.

Мало того, было бы, наверное, гораздо проще создать этот документ в коде HTML без использования языка XML и таблицы стилей. Но разделение формата хранения данных и формата их представления позволяет коренным образом изменить способ форматирования данных, не изменяя способ их представления. При использовании языка HTML в непосредственном виде такая возможность отсутствует. Чтобы понять, с чем это связано, рассмотрим таблицу стилей, показанную в листинге 8.8.

Эта таблица стилей может использоваться для преобразования документа XML совсем в иную компоновку HTML по сравнению с первой (новую таблицу стилей для документа можно указать, откорректировав элемент документа <?xml-stylesheet> или перекрыв это значение в применяемом инструментальном средстве преобразования XSLT). На рис. 8.3 показано, как выглядит в браузере новая Web-страница.

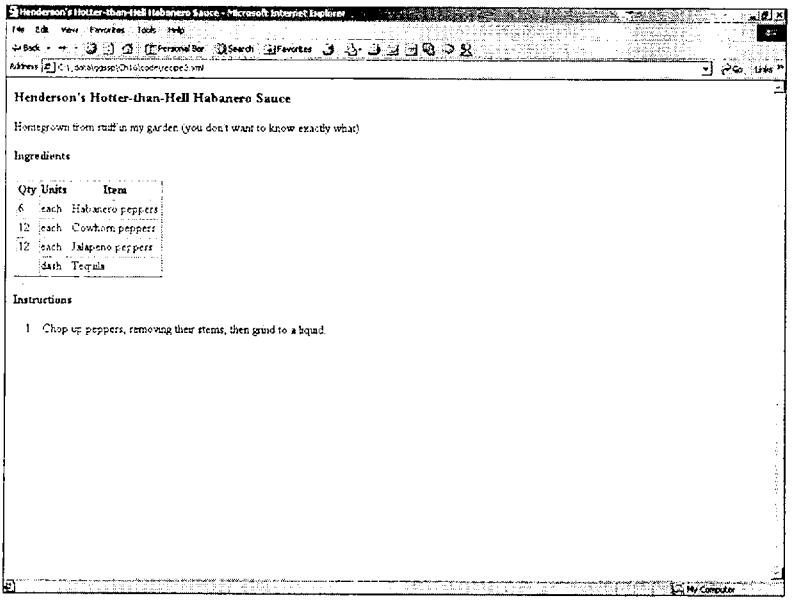


Рис. 8.2. Изображение текста рецепта в окне браузера

Листинг 8.8. Полностью иной способ преобразования того же документа XML

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <HEAD>
  <TITLE>Henderson&apos;s Hotter-than-Hell Habanero Sauce</TITLE>
  </HEAD>
  <body>
    <H3>Henderson&apos;s Hotter-than-Hell Habanero Sauce</H3>
      Homegrown from stuff in my garden
      (you don&apos;t want to know exactly what).
      <H4>Ingredients</H4>
      <UL>
        <xsl:for-each select="Recipe/Ingredients/Ingredient">
        <LI>
        <xsl:value-of select="Qty"/>&#9;<xsl:value-of
            select="Qty/@unit"/> of <xsl:value-of select="Item"/>
        </LI>
        </xsl:for-each>
       </UL>
```

```
      <P/>
      <H4>Instructions</H4>
     <table border="2">
      <tr BGCOLOR="#00FF00">
       <TH>#</TH>
        <TH>Step</TH>
      </tr>
     <xsl:for-each select="Recipe/Instructions">
     <tr>
       <td><xsl:value-of select="position()"/></td>
       <td><xsl:value-of select="Step"/></td>
      </tr>
      </xsl:for-each>
     </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

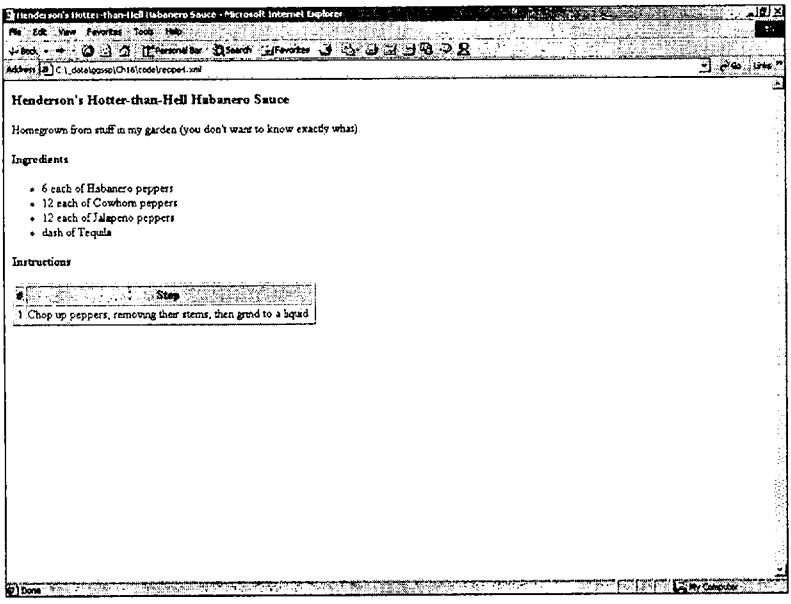


Рис. 8.3. Новая версия страницы с описанием рецепта в окне браузера

Вполне очевидно, что эта страница имеет полностью иное форматирование. Таблица с ингредиентами исчезла, а ее место занял маркированный список. И наоборот, шаги инструкции `Instructions` были перенесены из упорядоченного списка в таблицу. Форматирование полностью изменилось, а данные остались прежними. Сам документ XML вообще не изменился.

Поскольку теперь данные имеют контекст, к ним можно обращаться непосредственно. Поэтому нет необходимости задавать в коде ссылки на столбцы или строки таблицы, представленной на языке HTML, после чего преобразовывать

данные из формата HTML в более удобный формат данных, поскольку данные уже находятся в таком формате. Мало того, это утверждение остается справедливым независимо от того, какой способ будет решено применить для преобразования или форматирования данных. Поскольку данные хранятся в коде XML, манипуляции с ними могут осуществляться с помощью практически любого способа, который будет рассматриваться как приемлемый. Одним из наглядных свидетельств некоторых возможностей языка XSLT является элемент `xsl:for-each` рассматриваемых таблиц стилей. Как и в большинстве языков, основная часть полезных свойств языка XSLT обусловлена возможностью неоднократно выполнять с его помощью одну и ту же задачу. В языке XSLT определен также целый ряд столь же мощных конструкций, в том числе перечисленных ниже.

- Элемент `xsl:if`.
- Элемент `xsl:choose`.
- Элемент `xsl:sort`.
- Элемент `xsl:attribute`.
- Встроенные средства поддержки сценариев. Пакет LotusXSL компании IBM предоставляет доступ к основной части функциональных средств XSLT, включая возможность вызывать из шаблонов XSLT встроенные сценарии на языке ECMAScript (это версия языка JavaScript, утвержденная европейским стандартом).

С полным списком элементов языка XSLT можно ознакомиться в спецификации XSLT, но достаточно сказать, что именно язык XSLT позволяет выявить истинную мощь и расширяемость языка XML. Язык XSLT представляет собой пример того, что автор называет аспектом языка XML, связанным с использованием "программируемых данных". Благодаря языку XSLT мы получаем возможность не только указывать способ форматирования данных, но и применять к данным программируемые модификации самих данных. Безусловно, такие средства обработки данных являются весьма мощными.

Поскольку в настоящее время с помощью языков XSLT и XML в основном осуществляются задачи форматирования, то может показаться, что XML — это просто технология управления информационным наполнением. Но такой вывод не верен; язык XML открывает гораздо более широкие возможности. В действительности, с точки зрения Web-мастера, семейство технологий XML представляет собой значительный шаг вперед по сравнению с HTML. Но для того чтобы воспользоваться возможностями XML, необходимо понять, что этот язык не должен служить лишь для форматирования данных или управления информационным наполнением. Он прежде всего должен рассматриваться как средство отображения данных и предоставления этим данным достаточно широкого контекста для того, чтобы они могли применяться в самых разнообразных ситуациях. В связи с этим появился широкий спектр приложений, которые вышли за пределы проблемной области браузеров и Web-страниц. Для того чтобы вся мощь языка XML могла использоваться в приложениях такого типа, предназначена модель DOM.

Объектная модель документа

До сих пор язык XML рассматривался с точки зрения обобщения форматов документов. Но истинная мощь языка XML обнаруживается при его использовании для структуризации информации.

Все документы XML состоят из вложенных множеств элементов. Каждый документ заключен в корневой элемент, который, в свою очередь, служит для размещения в нем других элементов. Вполне естественно, что структура такого документа представляет собой дерево (имеется в виду дерево элементов) объектов, которые представляют все информационное наполнение документа. Технология DOM (Document Object Model – объектная модель документа) выходит за рамки простого подхода, основанного на обработке текста в виде линейного потока, и предоставляет независимые от языка средства работы с документом XML как с деревом объектов.

Реализуемый при этом объектно-ориентированный доступ к документам XML создает предпосылки для формирования целого ряда других областей применения языка XML. В частности, становится тривиальной задача включения средств языка XML в любое приложение в качестве механизма обмена данными между процессами или приложениями, поскольку любые объекты, с которыми при этом приходится иметь дело, становятся объектами применяемого в приложении языка программирования. При этом не имеет значения, является ли этим языком Visual Basic, Java или C#. Документы XML можно читать, выполнять с ними любые манипуляции и вообще любые операции обработки, вызывая методы объектов и обращаясь к свойствам этих объектов.

Трудно даже представить себе, какие возможности открываются при использовании такого подхода. Например, допустим, что создана система баз данных, в которой вся информация в базе данных представлена в виде документа XML. Если нужно будет получить схему этой базы данных, то для этого достаточно извлечь данные схемы XML из модели DOM, пропустить эту схему через средства преобразования XSLT и получить отображаемую в браузере схему базы данных, которая всегда остается актуальной. Если же нужно будет создать унифицированное инструментальное средство, позволяющее администратору базы данных управлять объектами в СУБД SQL Server, Oracle, DB2 и в СУБД других широко применяемых типов, не создавая для каждой из них отдельный API-интерфейс администрирования, то достаточно сформировать схемы баз данных в этих СУБД в виде деревьев DOM и получить возможность создать единственное инструментальное средство для работы со всеми объектами этих СУБД.

Поставщики СУБД уже рассматривают вопрос об использовании модели DOM и языка XML в рамках подходов, подобных описанному в предыдущем абзаце. В частности, как будет описано ниже в данной главе, значительная часть подобных средств реализована в программе SQL Server.

Обработка документов XML с помощью средств MSXML

Средства MSXML поддерживают два основных API-интерфейса для обработки документов XML — DOM и SAX (сокращение от Simple API for XML — простой API-интерфейс для XML). Начнем с описания модели DOM.

Средства MSXML и модель DOM

Как было указано выше, метод обработки документов XML на основе модели DOM предусматривает синтаксический анализ документа XML и загрузку этого документа в память в виде древовидной структуры. Результаты синтаксического анализа документа XML с помощью метода DOM принято называть документом DOM (или просто DOM). В листинге 8.9 приведено простое приложение на языке VB, в котором показано, как выполнить синтаксический анализ документа XML с помощью метода DOM и определить наличие в нем конкретного множества узлов с помощью запроса. (Исходный код этого приложения можно найти в подкаталоге `CH08\domltest` на компакт-диске, прилагаемом к данной книге.)

Листинг 8.9. Приложение на языке VB, в котором осуществляется обработка документа XML с помощью метода DOM

```
Private Sub Command1_Click()

  Dim bstrDoc As String

  bstrDoc = "<Songs> " & _
  "<Song title='One More Day' artist='Diamond Rio' />" & _
  "<Song title='Hard Habit to Break' artist='Chicago' />" & _
  "<Song title='Forever' artist='Kenny Loggins' />" & _
  "<Song title='Boys of Summer' artist='Don Henley' />" & _
  "<Song title='Cherish' artist='Kool and the Gang' />" & _
  "<Song title='Dance' artist='Lee Ann Womack' />" & _
  "<Song title='I Will Always Love You' artist= _
      'Whitney Houston' />" & _
"</Songs>"

  Dim xmlDoc As New DOMDocument30

  If Len(Text1.Text) = 0 Then
     Text1.Text = bstrDoc
  End If

  If Not xmlDoc.loadXML(Text1.Text) Then
    MsgBox "Error loading document"
  Else
    Dim oNodes As IXMLDOMNodeList
    Dim oNode As IXMLDOMNode
```

```
    If Len(Text2.Text) = 0 Then
     Text2.Text = "//Song/@title"
    End If
    Set oNodes = xmlDoc.selectNodes(Text2.Text)

    For Each oNode In oNodes
      If Not (oNode Is Nothing) Then
        sName = oNode.nodeName
        sData = oNode.xml
        MsgBox "Node <" + sName + ">:" _
          + vbNewLine + vbTab + sData + vbNewLine
      End If
    Next

     Set xmlDoc = Nothing
  End If
End Sub
```

Работа приложения начинается с создания экземпляра объекта DOMDocument. Объект DOMDocument является основой для осуществления всех прочих операций, выполняемых с помощью метода DOM, с использованием средств MSXML. На следующем этапе вызывается метод DOMDocument.loadXML для синтаксического анализа документа XML и загрузки его в дерево DOM. После загрузки документа в память этот документ можно запрашивать с помощью запросов XPath или выполнять с ним дальнейшие манипуляции, применяя вызовы методов объекта DOMDocument. В данном примере вызывается метод selectNodes для выполнения запроса к документу с помощью языка XPath. Метод selectNodes объекта DOMDocument возвращает объект списка узлов, который затем может быть обработан в цикле с помощью оператора For Each. Для каждого узла из этого множества отображается имя узла, за которым следует его содержимое. Итак, в результате синтаксического анализа документа XML с помощью методов модели DOM документ преобразуется в объект, находящийся в памяти, к которому могут в дальнейшем применяться различные операции, как и к любому другому объекту. Благодаря этому появляется возможность обращаться к документу и манипулировать им так, как если бы это был обычный объект, поскольку он и становится именно таковым.

Средства MSXML и интерфейс SAX

В отличие от модели DOM, интерфейс SAX — это стандарт, утвержденный консорциумом W3C. Интерфейс SAX не предоставляет приложению доступ к данным XML, создавая полную копию документа в памяти, а используется в качестве API-интерфейса, управляемого событиями. Приложение обрабатывает документ XML с помощью интерфейса SAX, отвечая на события SAX. По мере того как процессор SAX читает документ, активизируется событие после обнаружения в процессоре SAX каждого нового узла или раздела документа. Затем процессор SAX активизирует в приложении соответствующий код обработчика событий и передает в это приложение все необходимые данные о собы-

тии. В дальнейшем в приложении может быть принято решение о том, какие должны быть выполнены ответные действия. В частности, в приложении можно сохранить данные о событии в древовидной структуре того или иного типа, как и в случае применения обработки по методу DOM; это событие может быть проигнорировано; в данных о событии может быть проведен поиск конкретного узла или значения; могут быть также предприняты некоторые другие действия. После обработки в приложении активизированного события процессор SAX продолжает обработку документа. Процессор SAX ни при каких обстоятельствах не сохраняет весь документ в памяти, как это предусмотрено в методе DOM. На самом деле процессор SAX действует как механизм синтаксического анализа, к которому приложение может присоединять свои собственные функциональные средства. В действительности обработка с помощью процессора SAX осуществляется и в загрузчике DOM из состава средств MSXML, поскольку в загрузчике DOM используется базовый механизм синтаксического анализа процессора SAX. В загрузчике DOM из состава средств MSXML выполняется настройка обработчиков событий SAX, которые сохраняют данные, переданные им через интерфейс SAX, в дереве DOM.

Процессор SAX по умолчанию не сохраняет данные документа в памяти, поэтому в силу самого этого свойства требует меньшего объема памяти, чем при использовании метода DOM. Тем не менее применение интерфейса SAX связано с более значительными сложностями. В методе DOM документы XML сохраняются в памяти, поэтому операции с этими документами выполняются столь же просто, как и с объектами любых других типов.

В листинге 8.10 приведен некоторый код на языке VB, который демонстрирует способ применения процессора SAX. Этот код состоит из трех основных модулей: главная форма, класс обработчика информационного наполнения и класс обработчика ошибок. (Полный исходный код этого приложения приведен в подкаталоге SAX каталога CH08 компакт-диска, прилагаемого к данной книге.)

Листинг 8.10. Приложение на языке VB, в котором осуществляется обработка документа XML с помощью интерфейса SAX

```
' Главная форма
Option Explicit

Private Sub Command1_Click()

    ' Создать объект чтения для процессора SAX
    Dim reader As New SAXXMLReader

    ' Выполнить настройку конфигурации обработчиков ошибок
     Dim CHandler As New ContentHandler
    Set reader.ContentHandler = CHandler

    Dim EHandler As New ErrorHandler
     Set reader.ErrorHandler = EHandler

    Text1.text = ""
    On Error GoTo ErrorTrap
```

```
    reader.parseURL (App.Path & "\" & Text2.text)
    Exit Sub

ErrorTrap:
    Text1.text = Text1.text & "Error: " & Err.Number & " : "
        & Err.Description

End Sub

' Обработчик информационного наполнения
Option Explicit

Implements IVBSAXContentHandler

Private Sub IVBSAXContentHandler_startElement(strNamespaceURI
    As String, strLocalName As String, strQName As String, ByVal
    attributes As MSXML2.IVBSAXAttributes)

    Form1.Text1.text = Form1.Text1.text & "__ELEMENT START__" &
        vbCrLf & "<" & strLocalName
Dim i As Integer
    For i = 0 To (attributes.length - 1)
        Form1.Text1.text = Form1.Text1.text & " " &
            attributes.getLocalName(i) & "=""" &
            attributes.getValue(i) & """"
    Next

    Form1.Text1.text = Form1.Text1.text & ">" & vbCrLf

End Sub

Private Sub IVBSAXContentHandler_endElement(strNamespaceURI
    As String, strLocalName As String, strQName As String)

    Form1.Text1.text = Form1.Text1.text & "__ELEMENT END__" &
        vbCrLf & "</" & strLocalName & ">" & vbCrLf

End Sub

Private Sub IVBSAXContentHandler_characters(text As String)
    text = Replace(text, vbLf, vbCrLf)
    Form1.Text1.text = Form1.Text1.text & "__CHARACTERS__" &
        vbCrLf & text & vbCrLf
End Sub

Private Property Set IVBSAXContentHandler_documentLocator
    (ByVal RHS As MSXML2.IVBSAXLocator)
    Form1.Text1.text = Form1.Text1.text & "__DOCUMENT_LOCATOR__" &
        vbCrLf
End Property

Private Sub IVBSAXContentHandler_endDocument()
    Form1.Text1.text = Form1.Text1.text & "__DOCUMENT END__" &
        vbCrLf
```

```
End Sub

Private Sub IVBSAXContentHandler_endPrefixMapping(strPrefix
    As String)
   Form1.Text1.text = Form1.Text1.text & "__PREFIX MAPPING__" &
       vbCrLf & strPrefix & vbCrLf
End Sub

Private Sub IVBSAXContentHandler_ignorableWhitespace(strChars
    As String)
Form1.Text1.text = Form1.Text1.text & "__IGNORABLE
       WHITESPACE__" & vbCrLf & strChars & vbCrLf
End Sub

Private Sub IVBSAXContentHandler_processingInstruction(target
   As String, data As String)
    Form1.Text1.text = Form1.Text1.text & "__PROCESSING
        INSTRUCTION__" & vbCrLf & "<?" & target & " " &
        data & ">" & vbCrLf
End Sub

Private Sub IVBSAXContentHandler_skippedEntity(strName As String)
   Form1.Text1.text = Form1.Text1.text & "__SKIPPED ENTITY__" &
       vbCrLf & strName & vbCrLf
End Sub

Private Sub IVBSAXContentHandler_startDocument()
   Form1.Text1.text = Form1.Text1.text & "__DOCUMENT START__" &
       vbCrLf
End Sub

Private Sub IVBSAXContentHandler_startPrefixMapping(strPrefix
   As String, strURI As String)
   Form1.Text1.text = Form1.Text1.text & "__START PREFIX
    MAPPING__" & strPrefix & " " & strURI & " " & vbCrLf
End Sub

' Обработчик ошибок
Option Explicit

Implements IVBSAXErrorHandler

Private Sub IVBSAXErrorHandler_fatalError
    (ByVal lctr As IVBSAXLocator, msg As String, ByVal
   errCode As Long)
    Form1.Text1.text = Form1.Text1.text & "Fatal error: " &
   msg & " Code: " & errCode
End Sub

Private Sub IVBSAXErrorHandler_error(ByVal lctr As IVBSAXLocator,
   msg As String, ByVal errCode As Long)
    Form1.Text1.text = Form1.Text1.text & "Error: " & msg &
        " Code: " & errCode
End Sub
```

```
Private Sub IVBSAXErrorHandler_ignorableWarning
    (ByVal oLocator As MSXML2.IVBSAXLocator,
    strErrorMessage As String, ByVal nErrorCode As Long)

End Sub
```

Как уже было сказано выше, в приложении машина SAX используется путем вызова синтаксического анализатора SAX и формирования ответов на активизируемые им события. Для того чтобы можно было воспользоваться в приложении на языке VB машиной SAX из состава средств MSXML, необходимо реализовать такие интерфейсы SAX, как `IVBSAXContentHandler`, `IVBSAXErrorHandler`, `IVBSAXDeclHandler`, `IVBSAXDTDHandler` и `IVBSAXLexicalHandler`. Реализация этих интерфейсов сводится к тому, что настраиваются обработчики событий для формирования ответов на определяемые ими события. В данном примере кода реализованы обработчики событий `IVBSAXContentHandler` и `IVBSAXErrorHandler` с помощью классов `ErrorHandler` и `ContentHandler`.

Работа приложения начинается с создания объекта `SAXXMLReader`. Этот объект обрабатывает передаваемый ему документ XML и активизирует соответствующее событие по мере чтения документа. Код, содержащийся в классах `ContentHandler` и `ErrorHandler`, отвечает на эти события и записывает содержательный текст в главную форму.

Дополнительные источники информации

Дополнительные материалы для чтения

- По мнению автора, книга Лиз Кастро *XML for the World Wide Web: Visual QuickStart Guide* (Berkeley, CA: Peachpit Press, 2000) — образец краткого и вместе с тем достаточно полного изложения данной темы. Лиз всегда пишет хорошие книги, а эта, наверное, — лучшая из выпущенных ею книг.
- В книге У. Скотта Минса и Эллиотта Расти Гарольда *XML in a Nutshell, 2nd Edition* (Sebastopol, CA: O'Reilly, 2001) также можно найти краткое изложение интересующего нас материала и даже описание незаметных на первый взгляд нюансов этого языка.
- Кроме того, хорошим источником информации является книга Стива Холзнера *Inside XML* (Indianapolis, IN: New Riders, 2000). Эта книга является всеобъемлющей, и в ней очень подробно раскрываются многие важные темы, касающиеся XML.
- Еще одним хорошим учебником начального уровня является книга Эрика Т. Рэя *Learning XML* (Sebastopol, CA: O'Reilly, 2001). В ней содержится превосходное вводное описание многих существующих синтаксических анализаторов XML, а некоторые темы, не раскрытые в других книгах (например, схема XML), рассматриваются более глубоко.

- Книга Майкла Кэя *XSLT Programmer's Reference* (Indianapolis, IN: Wrox, 2001) позволяет читателю узнать все необходимое о языке XSLT. К тому же, сам Майкл является автором программы SAXON – одного из лучших современных процессоров XSLT.
- Непревзойденным источником информации о языках XML и HTML, а также обо всем, что касается Web, безусловно, может служить Web-узел консорциума W3C (`http://www.w3c.org`). Некоторые представленные на этом узле документы спецификаций являются немного сложными для восприятия, но усилия, затраченные на их изучение, вполне оправдываются. На узле W3C можно также найти множество ссылок на учебные руководства, относящиеся к XML, бесплатные инструментальные средства и другие ресурсы.
- На узлах большинства крупных поставщиков программных продуктов можно найти ссылку на портал того или иного типа, посвященный языку XML. Автор пришел к выводу, что наиболее информативными являются порталы компаний Microsoft и Sun.

Инструментальные средства

- Рекомендуем читателю начать с приобретения хорошего редактора XML/XSLT/XSD. Самому автору нравится программа XML Spy (`http://www.xmlspy.com`), но имеется также целый ряд других современных редакторов. Не позволяйте сбить себя с толку разговорами о том, что графические пользовательские интерфейсы рассчитаны только на начинающих. Использование обычного текстового редактора наподобие Notepad и расходование долгих часов на выполнение той работы, которую с помощью инструментального средства с графическим пользовательским интерфейсом можно сделать за секунды, просто не имеет смысла. Вы в конечном итоге тратите впустую много времени, преодолевая все связанные с разработкой сложности и не получая помощи от неуклюжего инструментального средства, тогда как эти усилия могли бы пойти на освоение самой технологии.
- Кроме того, для разработки может потребоваться программа проверки допустимости документа с помощью схемы XML Schema или определения DTD. Сам автор использует программу, которую можно бесплатно загрузить с Web-узла Microsoft, но есть также много других современных программ.
- В зависимости от наличия у разработчика других инструментальных средств, возможно, ему нужно будет приобрести отдельное инструментальное средство преобразования XSLT. Автор пользуется инструментальными средствами XSLT компании Microsoft, а также инструментальным средством XT, которое разработано Джеймсом Кларком (James Clark). К тому же, есть бесплатные версии и таких программ.

- Если разработка проводится на компьютере с операционной системой Windows, то следует получить новейшую версию синтаксического анализатора MSXML. Эта программа на данный момент – лучшая на рынке.
- Разработчику стоит также приобрести инструментальное средство SAXON Майкла Кэя, даже если в его распоряжении имеются другие процессоры XSLT. Это прекрасная программа, которая создана человеком, полностью овладевшим данной технологией.
- Тем программистам, которые проводят свои разработки в операционной системе Windows, желательно также получить комплект MSXML SDK. В состав этого комплекта входят хорошие примеры кода и документация, которая может пригодиться при создании приложений на основе API-интерфейсов MSXML.

Резюме

XML – это язык представления программируемых, иерархических данных. Разработчики данного языка поставили перед собой задачу, чтобы он был сравнимым с языком HTML с точки зрения простоты использования и сопоставимым с языком SGML по своей мощи, расширяемости и выразительности.

Документы XML можно преобразовывать из одних форматов в другие с помощью языка XSLT. Чаще всего целевым форматом является формат HTML, но такое требование не является обязательным. Нередко применяется также преобразование в форматы документов на языках, отличных от языков разметки.

Определения DTD и схемы XML позволяют обеспечить, чтобы документы XML были не только формально правильными, но и допустимыми. Дело в том, что документ может быть формально правильным и, тем не менее, содержать недопустимые данные. В задачу синтаксического анализатора XML с проверкой допустимости входит контроль документа по определению DTD или по схеме XML для выяснения того, содержит ли он допустимые данные.

Одним из широко применяемых API-интерфейсов для обработки документов XML в виде объектов является DOM. Вкратце можно отметить, что метод DOM позволяет загрузить документ XML в древовидный объект, с которым затем можно манипулировать, определяя и задавая свойства объекта и вызывая его методы.

Кроме того, для обработки документов XML все чаще применяется API-интерфейс SAX. В отличие от DOM, API-интерфейс SAX не предусматривает загрузки всего документа XML в память. Вместо этого с его помощью осуществляется чтение документа и активизация в ходе этого соответствующих событий. Задача формирования ответов на эти события возлагается на приложение.

С языком XML связана не только одна технология, а целое семейство технологий. Эти технологии продолжают развиваться и с каждым днем находят все более широкое применение во всем мире. Для специалиста крайне важно узнать уже сейчас о языке XML все, что возможно, поскольку лишь при этом условии можно

наилучшим образом использовать средства программы SQL Server, связанные с языком XML, причем не только те из них, которые уже существуют, но и те, которые появятся в будущем. Автор высказывает недвусмысленный прогноз — уже не за горами то время, когда язык XML приобретет не меньшую важность в области разработки приложений SQL Server, чем язык Transact-SQL. Безусловно, необходимо заняться изучением языка XML уже сейчас.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Язык HTML допускает введение собственных дескрипторов, но введение определяемых пользователем атрибутов запрещено.
2. Назовите имя объекта чтения MSXML SAX.
3. Поддерживает ли язык XML понятие пустых элементов?
4. Каково максимальное количество корневых узлов, которые может иметь документ XML?
5. Подтвердите или опровергните следующее утверждение. Схема XML используется для преобразования документа XML из одного формата в другой.
6. К какому типу узла документа относится `"bar"` во фрагменте кода `<foo "bar"/>`?
7. Подтвердите или опровергните следующее утверждение. Возможна такая ситуация, при которой документ является допустимым, но не формально правильным.
8. Какую классическую структуру данных наиболее близко напоминает документ XML, загруженный в память с помощью интерфейса DOM?
9. Подтвердите или опровергните следующее утверждение. Язык XML, как и язык HTML, не чувствителен к регистру.
10. Объясните назначение конструкции `xsl:for-each`.
11. Подтвердите или опровергните следующее утверждение. Для обработки документа с помощью интерфейса DOM требуется больше памяти, чем для обработки документа с помощью интерфейса SAX.
12. Считается ли пустым элемент XML, который содержит атрибуты, но не содержит какие-либо другие элементы или данные?
13. Является ли фрагмент кода `<Customer lastname=Brown/>` допустимым элементом XML?
14. Подтвердите или опровергните следующее утверждение. В синтаксических анализаторах XML многие нарушения правил вложения дескрипторов не рассматриваются как ошибка, тогда как синтаксические анализаторы HTML предъявляют более строгие требования к формату документов.

15. Опишите функциональное различие между следующими фрагментами кода XML (если оно есть):

```
<foo></foo>
```

и

```
<foo/>
```

ЧАСТЬ II

Подсистемы, компоненты и технологии

ГЛАВА 8

Программа SQL Server как серверное приложение

В данной главе программа SQL Server рассматривается с той точки зрения, что она представляет собой такое же серверное приложение Windows, как и любое другое. Выше в этой книге рассматривались сетевые средства Win32 и функции API-интерфейсов ввода-вывода, которые используются в серверах Windows. Речь шла об API-интерфейсах, предназначенных для поддержки процессов и потоков, планирования и синхронизации потоков, управления памятью, а также о средствах COM. В настоящей главе показано, как некоторые из этих средств используются в самой программе SQL Server и какое место они занимают в общей классификации серверных приложений Windows.

ПРИМЕЧАНИЕ. В данной главе предполагается, что читатель уже ознакомился с главами 5, "Основные принципы ввода-вывода", и 6, "Основы организации сетей". Если вы еще не прочитали эти главы, то вам следует сделать это, прежде чем приступать к дальнейшей работе над этой главой.

Программа SQL Server и организация сетей

Как было сказано в главе 6, "Основы организации сетей", в программе SQL Server для приема и обработки запросов на установление соединений используются стандартные вызовы функций сетевых API-интерфейсов. Читатель, скорее всего, уже знает о том, что в программе SQL Server вызывается код ее сетевых библиотек (который находится в отдельных библиотеках DLL) для приема и обработки пользовательских запросов на установление соединения. Но далеко не все знают о том, что в функциях этих библиотек DLL для выполнения требуемой работы, в свою очередь, вызываются стандартные функции сетевого API-интерфейса Windows.

При приеме запросов на установление соединений, передаваемых по протоколам TCP/IP, широко используются API-интерфейсы сокетов Windows, которые были описаны в главе 6. Для приема новых запросов на установление соединения в коде библиотеки Net-Library программы SQL Server вызываются функции

accept и WSAAccept, а для обработки клиентских запросов и возврата полученных данных вызываются функции API-интерфейса ввода-вывода Win32. Как было описано в главе 6, дескрипторы сокетов, возвращаемые той реализацией API-интерфейса сокетов, которая применяется в операционной системе Windows, могут использоваться с такими стандартными функциями файлового ввода-вывода Win32, как ReadFile и WriteFile.

При приеме запросов на установление соединения, поступающих через именованные каналы, в коде библиотеки Net-Library используются стандартные функции API-интерфейсов файлового ввода-вывода Win32 для обработки новых запросов на установление соединения, а также обработки самих клиентских запросов и возврата результатов. Как было описано в главе 6, приложения Windows работают с именованными (и анонимными) каналами с помощью тех же функций ввода-вывода Win32, которые используются при работе с дисковыми файлами.

В программе SQL Server средства предоставления доступа клиентам к базе данных поддерживаются с использованием мультипротокольной библиотеки Net-Library на основе функций API-интерфейса RPC подсистемы Win32. Приложения Win32 взаимодействуют друг с другом с помощью API-интерфейса RPC на основе функций-заглушек (или функций-посредников), вызываемых в клиентской программе. В этих функциях-заглушках, в свою очередь, выполняются вызовы функций API-интерфейса RPC для маршалинга (преобразования из внутреннего представления во внешнее) каждого вызова функции и данных ее параметров перед отправкой на сервер-получатель. На сервере-получателе осуществляется демаршалинг (для преобразования из внешнего представления во внутреннее) и вызываются действительные серверные функции, которым соответствуют функции-посредники, вызванные пользователем. Таким образом, API-интерфейс RPC предоставляет пользователю для связи по сети интерфейс уровня вызовов вместо тех интерфейсов, предназначенных для передачи пакетов, байтов или строк, которые обычно представляют сетевые библиотеки API-интерфейсов Windows. При подключении клиента к программе SQL Server с помощью мультипротокольной библиотеки Net-Library клиент, по сути, выполняет вызов процедуры из серверной версии библиотеки, после чего этот вызов преобразуется во внешнюю форму и передается по сети в тот экземпляр программы SQL Server, для которого он предназначен. Затем функции серверной библиотеки Net-Library обрабатывают полученные запросы и передают клиентские запросы обычным программным средствам выполнения сетевого ввода-вывода, которые используются в других библиотеках Net-Library.

В программе SQL Server для обеспечения асинхронной обработки запросов на осуществление ввода-вывода, связанных с поддержкой соединения, используется порт завершения ввода-вывода. Как было указано в главе 5, порт завершения ввода-вывода, который связан с объектом файла (или сокета), получает новый пакет завершения после выполнения каждой асинхронной операции ввода-вывода, инициализированной тем объектом, который связан с этим портом. Такой проект позволяет поддерживать в программе SQL Server большое количество одновременно действующих клиентских соединений с помощью минимального количества рабочих потоков, относящихся к сети.

Каждая библиотека Net-Library получает отдельный рабочий поток, который используется для приема запросов на установление соединения и обработки

запросов на выполнение операций ввода-вывода, относящихся к сети. Если сервер осуществляет прием запросов на установление соединения через сокеты TCP/IP и именованные каналы, то в каждой соответствующей библиотеке Net-Library применяется свой собственный рабочий поток.

Для ознакомления с тем, каким образом в коде библиотеки Net-Library программы SQL Server используются функции API-интерфейсов, описанные в главах 5 и 6, можно применить отладчик WinDbg. Выполнение упражнения 9.1 позволит читателю самому узнать о том, какого типа вызовы сетевых API-интерфейсов выполняются в программе SQL Server и как часто это происходит.

Упражнение

При выполнении этого упражнения вы должны быть единственным пользователем сервера.

Упражнение 9.1. Ознакомление с тем, как в программе SQL Server используются сетевые функции API-интерфейсов Windows

1. Остановите применяемый вами экземпляр программы SQL Server, который служит для разработки или экспериментирования.
2. Запустите отладчик WinDbg и убедитесь в том, что в этом отладчике пути к файлам с отладочной информацией установлены правильно, как описано в главе 2.
3. В меню File отладчика WinDbg выберите команду Open Executable и найдите исполняемый файл применяемой программы SQL Server (`sqlservr.exe`). Задайте следующие параметры командной строки:

   ```
   -c -sYourInstanceName
   ```

   и укажите здесь вместо параметра `YourInstanceName` имя применяемого вами экземпляра программы SQL Server. Если используется экземпляр, предусмотренный по умолчанию, то полностью исключите параметр `-s`.
4. Щелкните на кнопке OK, чтобы запустить программу SQL Server под управлением отладчика WinDbg. Как только на экране появится приглашение к вводу команд WinDbg, установите следующие точки останова:

   ```
   bp WS2_32!WSAAccept
   bp WS2_32!accept
   bp WS2_32!listen
   ```
5. Теперь введите `g` в командном окне WinDbg и нажмите <Enter>, чтобы разрешить запуск программы SQL Server.
6. Запустите программу Query Analyzer и сделайте попытку подключиться к используемому экземпляру сервера. Вы должны обнаружить, как сразу же активизируются некоторые из установленных вами точек останова. Введите следующие дополнительные точки останова:

   ```
   bp kernel32!GetQueuedCompletionStatus
   bp kernel32!ReadFile
   ```

7. Введите `g` и снова нажмите клавишу <Enter>, чтобы разрешить продолжить выполнение программы SQL Server. Введенный вами новый запрос на установление соединения в программе Query Analyzer должен быть выполнен успешно. Теперь откройте новое соединение. Вы должны опять обнаружить, как активизируются некоторые из установленных вами точек останова, включая те новые точки останова, которые были установлены только что. Введите `g` и нажмите клавишу <Enter>, чтобы пройти эти точки останова. Вы должны обнаружить, как некоторые из них активизируются снова и снова, по мере выполнения операций, связанных с поддержкой нового соединения и обработки сервером предусмотренных по умолчанию опций соединения ODBC.
8. На данный момент стоящая перед вами задача выполнена. Нажмите клавиши <Shift+F5>, чтобы прекратить отладку, затем закройте отладчик WinDbg. Вам потребуется перезапустить сервер, чтобы продолжить работать с ним, поскольку после завершения сеанса отладки сервер должен быть остановлен.

В ходе выполнения предыдущего упражнения читатель должен был обнаружить, что сетевые функции API-интерфейсов, которые рассматривались в главе 6, а также многие функции ввода-вывода, описанные в главе 5, очень интенсивно используются в коде библиотеки Net-Library программы SQL Server. Понимание того, как работают эти функции и для чего обычно используются, позволит читателю получить полное представление о работе самой программы SQL Server.

Исполняемый файл программы SQL Server

Исполняемый файл программы SQL Server носит имя `sqlservr.exe` и находится в подкаталоге `binn` главного инсталляционного каталога SQL Server. Указанный подкаталог называется `binn`, поскольку в поставку предыдущих выпусков программы SQL Server входили 16-битовые клиентские исполняемые файлы и библиотеки, которые хранились в подкаталоге `bin`, а подкаталог `binn` был зарезервирован для 32-битовых исполняемых файлов и библиотек (дополнительная буква "n" сокращенно обозначала версию "NT" операционной системы Windows). В поставку программы SQL Server больше не входят какие-либо 16-битовые исполняемые двоичные файлы, но имя подкаталога для 32-битовых исполняемых двоичных файлов осталось неизменным.

Исполняемый файл `sqlservr.exe` представляет собой многопотоковое приложение Win32 для работы в терминальном режиме. Эта программа может работать как терминальное приложение или как служба и записывать свои выходные данные на терминал, в журнал регистрации ошибок и в журнал регистрации событий Windows.

При связывании исполняемого файла программы `sqlservr.exe` применяется параметр редактора связей `LARGEADDRESSAWARE`. Это означает, что данная программа способна воспользоваться пространством адресов непривилегированного режима, выходящим за пределы 2 Гбайт. Как было указано в главе 4, при загрузке любой версии операционной системы из семейства Windows NT Server (например, Windows 2000 Server, Windows Server 2003 и т.д.) с параметром `/3GB` (или с параметром `/USERVA` в версии Windows Server 2003) операционная система Windows увеличивает часть пространства виртуальных адресов процесса,

относящуюся к непривилегированному режиму, за счет той части, которая относится к привилегированному режиму. Код программы SQL Server разработан таким образом, чтобы можно было воспользоваться этой возможностью и захватить объем пространства адресов виртуальной памяти для непривилегированного режима, превышающий 2 Гбайт, если это позволяет операционная система; кроме того, с учетом такой возможности осуществляется связывание программы SQL Server.

Библиотеки DLL программы SQL Server

В главном исполняемом файле программы SQL Server, `sqlservr.exe`, применяется статический импорт девяти различных библиотек DLL. Эти библиотеки перечислены в табл. 9.1 и указано основное назначение каждой из них.

Для того чтобы можно было успешно осуществить запуск программы SQL Server, каждая из этих библиотек должна присутствовать в системе хост-компьютера. Кроме того, успешный запуск сервера происходит лишь при том условии, что в системе присутствуют все библиотеки DLL, которые требуются для работы указанных в табл. 9.1 библиотек (`Gdi32.dll`, `Ntdll.dll` и т.д.). К тому же, в программе SQL Server при запуске динамически загружаются многочисленные библиотеки DLL, состав которых зависит от выбранных опций (например, от того, какая библиотека Net-Library указана в конфигурации сервера как предназначенная для использования при приеме клиентских запросов на установление соединения). Кроме того, состав загружаемых библиотек DLL определяется с учетом того, какие из них потребуются для обработки клиентских запросов и выполнения других действий на сервере (например, запросов OPENXML, связанных серверных запросов, операций BULK INSERT и т.д.).

Три библиотеки DLL, обозначенные в табл. 9.1 полужирным шрифтом, входят в поставку программы SQL Server и, вообще говоря, представляют собой согласованный набор библиотек. Как правило (но не всегда), при выпуске нового служебного пакета или текущего исправления для программного продукта SQL Server эти библиотеки обновляются одновременно. Каждая из таких библиотек включает ресурс с обозначением версии Windows, поэтому читатель может легко ознакомиться со строками, содержащими обозначение каждой из этих библиотек, щелкнув на имени библиотеки правой кнопкой мыши в программе Windows Explorer, выбрав команду Properties, а затем — вкладку Version в диалоговом окне Properties.

Средства ввода-вывода программы SQL Server

При эксплуатации программы SQL Server в одной из версий семейства Windows NT максимально возможный объем файлового ввода-вывода в этой программе осуществляется асинхронно. Как было указано в главе 5, стандартные функции API-интерфейса Win32 могут выполняться асинхронно, если объект файла,

с которым они работают, был открыт с параметром FILE_FLAG_OVERLAPPED, а в функции, для которых требуется действительная структура OVERLAPPED, передается указатель на эту структуру. Например, если в программе вызвана функция CreateFile, и ей передан параметр FILE_FLAG_OVERLAPPED, то при последующем вызове функции ReadFileEx с указанием возвращенного дескриптора файла и указателя на действительную структуру OVERLAPPED операционная система Windows предпринимает попытку обработать сформированный таким образом запрос асинхронно. В программе SQL Server указанное средство Windows используется при любой возможности в целях предотвращения блокировок, связанных с выполнением операций ввода-вывода.

Таблица 9.1. Статически связанные библиотеки DLL программы SQL Server

Имя библиотеки DLL	Назначение
Kernel32.dll	Библиотека функций привилегированного режима подсистемы Win32
AdvApi32.dll	Библиотека функций защиты подсистемы Win32
User32.dll	Библиотека для работы с окнами и приложениями подсистемы Win32
Rpcrt4.dll	Библиотека этапа прогона для поддержки средств RPC подсистемы Win32
Opends60.dll	Библиотека ODS (Open Data Services – открытые службы данных) программы SQL Server
Ums.dll	Библиотека UMS (User Mode Scheduler – планировщик непривилегированного режима) программы SQL Server
Msvcrt.dll	Многопотоковая библиотека этапа прогона для Visual C++
Sqlsort.dll	Библиотека сортировки и сравнения строк программы SQL Server
Msvcirt.dll	Старая библиотека Iostream для Visual C++

Тем не менее, безусловно, возникают такие ситуации, которые исключают возможность асинхронного выполнения операций ввода-вывода в программе SQL Server. Одной из очевидных причин является эксплуатация этой программы в одной из версий семейства Win9x. Версии операционной системы Windows из семейства Win9x не поддерживают асинхронный файловый ввод-вывод, поэтому все операции файлового ввода-вывода программы SQL Server в версиях Win9x выполняются синхронно. За осуществление операций планирования выполнения запросов ввода-вывода отвечает компонент UMS программы SQL Server, который с помощью специального кода обнаруживает, что применяется версия Win9x, и выполняет операции ввода-вывода синхронно. Дополнительная информация о компоненте UMS, а также о том, как с помощью этого компонента обрабатываются все операции асинхронного ввода-вывода, приведена в главе 10.

Еще одним примером ситуации, в которой программа SQL Server не может использовать асинхронный ввод-вывод, является сжатие файлов MDF и LDF, из которых состоит база данных, с помощью средств сжатия файлов NTFS. Как было указано в главе 5, в операционной системе Windows запрещается применение асинхронных операций файлового ввода-вывода к сжатым файлам, поэтому

любые запросы на использование асинхронных операций к этим файлам преобразуются в запросы на выполнение синхронных операций при применении тех функций API-интерфейсов, которые поддерживают синхронный ввод-вывод (например, функции `ReadFile`), а если используются такие функции, которые не поддерживают синхронный ввод-вывод (например, функция `ReadFileEx`), то система возвращает ошибку. В этом состоит одна из причин, по которым в продуктах Microsoft не поддерживается сжатие файлов базы данных.

В программе SQL Server для быстрой загрузки непрерывной области дискового файла в целый ряд буферов, которые могут занимать или не занимать непрерывную область памяти, применяется ввод-вывод со сборкой-разборкой. Как было описано в главе 5, функции `ReadFileScatter` и `WriteFileGather` API-интерфейса Win32 принимают в качестве параметра указатель на массив буферов ввода-вывода, а затем либо загружают данные с диска в эти буфера, либо записывают данные из этих буферов на диск. Таким образом данные функции API-интерфейса обеспечивают возможность использовать для ввода и вывода буфера в памяти, которые не занимают непрерывную область, поэтому позволяют избежать необходимости применять в программе SQL Server непрерывный промежуточный буфер, который соответствует по размеру области дискового файла, предназначенной для чтения или записи, и последующего копирования из этого промежуточного буфера в массив буферов в памяти, не занимающих непрерывную область. Ввод-вывод со сборкой-разборкой был впервые введен в служебном пакете для Windows NT 4.0 и позволил добиться в программе SQL Server лучших показателей масштабируемости и производительности, чем было бы возможно с использованием других средств. Функции ввода-вывода со сборкой-разборкой используются в диспетчерах памяти BPool и MemToLeave и обеспечивают наилучшие возможности этих программ (способность управлять большими количествами буферов, не занимающих непрерывные области памяти, без учета необходимости определять местонахождение каких-либо из этих буферов на определенном расстоянии друг от друга исходя из потребностей ввода-вывода), вместе с тем позволяя этим машинам управления памятью обрабатывать запросы на выполнение операций ввода-вывода с максимально возможной скоростью. Дополнительная информация о вводе-выводе со сборкой-разборкой приведена в главе 5.

Компоненты программы SQL Server

В завершение этой главы будут кратко описаны основные компоненты программы SQL Server, участвующие в обработке типичного клиентского запроса. Многие из этих компонентов рассматриваются более подробно в других главах, а в данном разделе описаны только основные особенности этих компонентов. Типичным клиентским запросом является запрос к серверу на получение данных, находящихся в базе данных. Понимание того, как взаимодействуют компоненты программы SQL Server и как между ними распределяется работа, позволяет лучше разобраться во внутреннем функционировании этого сервера.

Как было указано выше, клиентские запросы передаются на сервер с помощью функций библиотек Net-Library программы SQL Server. Полученные запросы планируются для обработки с помощью компонента UMS. Затем каждый запрос получает компонент сервера, ответственный за языковую обработку и выполнение (Language Processing and Execution — LPE), и передает этот запрос на оптимизацию процессору запросов (Query Processor — QP). Компоненты LPE и QP являются модулями реляционной машины. После оптимизации клиентского запроса и подготовки для него плана выполнения компонент LPE выполняет план, направляя из реляционной машины вызовы в машину хранения (Storage Engine — SE). Машина хранения осуществляет операции физического ввода-вывода, просмотра таблиц и индексов, выборки данных и все другие операции, необходимые для выполнения запроса, полученного от реляционной машины. Взаимодействие между компонентами LPE и SE осуществляется на основе технологии COM с использованием вызовов функций интерфейса OLE DB. Дополнительная информация об интерфейсах и, в частности, о технологии COM приведена в главе 7.

Таким образом, к числу основных компонентов и технологий, участвующих в обработке типичного клиентского запроса, относятся библиотеки Net-Library, компоненты UMS, LPE, QP, SE и интерфейс OLE DB. Программа SQL Server обращается к этим компонентам и средствам каждый раз, когда клиент передает на обработку запрос к базе данных SQL Server. А в ходе выполнения всех описанных выше операций при возникновении в любом из компонентов сервера потребности в распределении памяти происходит обращение к различным диспетчерам памяти программы SQL Server. Подробная информация о том, какая последовательность событий происходит во время обработки запроса, приведена в главе 12, а дополнительные сведения об управлении памятью в программе SQL Server можно найти в главе 11.

Резюме

SQL Server — это сложное приложение Windows. В работе этой программы нет ничего загадочного, что отличало бы ее от других приложений Windows; в ней просто весьма интенсивно используются функции API-интерфейса Win32, функции библиотек этапа прогона на языке C/C++, интерфейсы COM и другие средства, которые могут применяться также в любом другом приложении Windows. SQL Server представляет собой сложное, многопотоковое приложение Windows, к которому могут подключаться клиенты и передавать запросы на получение данных из баз данных, запросы, связанные с сохранением данных и другие команды администрирования в форме сценариев на языке Transact-SQL. По мере обработки запросов полученные результаты (если они имеются) передаются клиенту, отправившему запрос, с помощью тех же механизмов, которые применялись для доставки самого запроса.

Вопросы для самопроверки

1. Какой компонент программы SQL Server отвечает за планирование операций ввода-вывода?
2. Какой основополагающий API-интерфейс Windows используется при подключении клиента к программе SQL Server с помощью мультипротокольной библиотеки Net-Library?
3. В очередь какого объекта привилегированного режима Windows передается пакет завершения после завершения асинхронной операции ввода-вывода в файле или сокете, с которым связан этот объект?
4. В состав какого более крупного компонента сервера входят компоненты LPE и QP программы SQL Server?
5. Подтвердите или опровергните следующее утверждение. Средства ввода-вывода со сборкой-разборкой поддерживают операцию записи данных с помощью единственного вызова функции API-интерфейса из ряда смежных буферов памяти в последовательность областей дискового файла, состоящую из нескольких несмежных участков.
6. Для чего предназначен файл `Sqlsort.dll`, который поставляется вместе с программой SQL Server?
7. Подтвердите или опровергните следующее утверждение. Редактирование связей программы SQL Server осуществляется с параметром редактора связей `LARGEADDRESSAWARE`, поэтому в данной программе может использоваться пространство адресов виртуальной памяти непривилегированного режима с объемом 3 Гбайт, если его предоставляет операционная система.
8. Какая технология используется для обмена сообщениями между такими компонентами сервера, как реляционная машина и машина хранения?
9. Подтвердите или опровергните следующее утверждение. В исполняемый файл программы SQL Server включено по методу статического связывания несколько библиотек DLL, но, несмотря на это, некоторые из этих библиотек загружаются также явно либо при запуске программы, либо на этапе ее прогона, в зависимости от способа настройки конфигурации программы и от команд, применяемых пользователями.
10. Какой еще параметр, кроме `/3GB`, может использоваться при эксплуатации версии Windows Server 2003 для увеличения размера пространства виртуальных адресов непривилегированного режима, доступного для таких приложений, которые способны работать в большом пространстве адресов, наподобие SQL Server?

ГЛАВА 6

Планировщик непривилегированного режима

Вплоть до версии 6.5 в программе SQL Server для планирования рабочих потоков, переключения между потоками и в целом для осуществления многозадачной организации работы использовались средства планирования Windows. Такой подход вполне себя оправдывал и позволял применять в программе SQL Server те достижения, касающиеся масштабируемости и эффективного использования процессора, которые были накоплены в операционной системе Windows в результате долгой и упорной работы.

Но на этапе перехода от версии 6.5 к версии 7.0 стало очевидно, что программа SQL Server начинает достигать "потолка масштабируемости". Ее возможности поддерживать тысячи одновременно работающих пользователей и эффективно масштабироваться в системах с количеством процессоров больше четырех ограничивались в связи с тем фактом, что планировщик Windows обслуживал программу SQL Server на таких же основаниях, как и любое другое приложение. Вопреки тому мнению, которое было широко распространено в то время, в версии SQL Server 6.5 не использовались какие-либо скрытые API-интерфейсы для достижения тех уровней масштабируемости, которые поддерживались в этой версии. В версии 6.5 применялись основные примитивы организации работы потоков и синхронизации потоков, которые были описаны выше в этой книге, а операционная система Windows планировала рабочие потоки SQL Server для выполнения на процессоре (процессорах) и депланировала их так же, как и рабочие потоки любого другого процесса. Очевидно, что такой подход с подгонкой всех приложений под одну мерку не был наиболее оптимальным решением применительно к такому высокопроизводительному приложению, как SQL Server, поэтому группа разработчиков SQL Server приступила к поиску способов оптимизации процесса планирования.

Цели проекта UMS

С самого начала проведения этих исследований было намечено несколько целей. Разрабатываемое средство планирования должно было обладать перечисленными ниже особенностями.

- Поддержка микропотоков (нового средства в версии Windows NT 4.0) и унификация средств, применяемых для работы с потоками и микропотоками, для того, чтобы в основной машине планировщика не приходилось применять отдельные строки кода для режима потоков и режима микропотоков.
- Предотвращение необходимости переключения потоков в привилегированный режим при любой возможности.
- Предотвращение необходимости переключения контекста при любой возможности.
- Поддержка асинхронного ввода-вывода и унификация средств, применяемых для работы с синхронными и асинхронными операциями ввода-вывода, для того, чтобы в основной машине планировщика не приходилось применять отдельные строки кода для версий Windows, в которых не поддерживается асинхронный файловый ввод-вывод (например, в версиях Windows 9x и Windows ME).

В конечном итоге было решено, что в версии SQL Server 7.0 должны применяться собственные средства планирования. На основании этого решения был создан компонент, известный как планировщик непривилегированного режима (User Mode Scheduler — UMS).

Компонент UMS действует как незаметный посредник между сервером и операционной системой. Код этого компонента находится в файле `UMS.DLL`, а сам этот компонент предназначен для предоставления модели программирования, которая весьма напоминает модель планирования потоков и асинхронного ввода-вывода подсистемы Win32. Программисты, знакомые с одной из этих моделей (UMS или Win32), не будут испытывать никаких затруднений при работе с другой моделью.

Основное назначение компонента UMS состоит в том, чтобы максимально возможная часть операций планирования в программе SQL Server выполнялась в непривилегированном режиме. Это означает, что в компоненте UMS обязательно должны предприниматься попытки избежать переключения контекста, поскольку при переключении контекста происходит переход в привилегированный режим. Как было указано в главе 3, операции переключения контекста могут оказаться дорогостоящими и ограничить масштабируемость. В крайних случаях может обнаружиться такая ситуация, что в процессе затрачивается больше времени на переключение контекста потоков, чем на реальную работу.

Сравнение средств планирования в непривилегированном и привилегированном режимах

У читателя может возникнуть вопрос — в чем заключается преимущество переноса средств управления планированием в сам процесс SQL Server. Разве при этом программа SQL Server не будет просто дублировать функциональные возможности, уже предоставленные операционной системой Windows? Ведь в компании Microsoft

над созданием операционной системы Windows и ее планировщика трудилось столько талантливых людей! Разве возможно такое, что группа разработчиков SQL Server предложит нечто более масштабируемое?

Автор ответит на эти вопросы более подробно ниже в этой главе, но вкратце можно отметить, что собственные потребности в планировании можно определить в программе SQL Server гораздо точнее, чем в операционной системе Windows или в любом другом программном компоненте, не входящем в состав программного продукта SQL Server. К тому же, компонент UMS не дублирует полные функциональные возможности планировщика Windows, а лишь реализует основные средства, относящиеся к планированию заданий, таймеров и асинхронных операций ввода-вывода. Кроме того, в действительности этот компонент опирается на собственные примитивы планирования и синхронизации потоков операционной системы Windows. Некоторые концепции планирования Windows (например, приоритет потоков) не имеют прямых аналогов в UMS.

Сравнение вытесняющего и кооперативного управления задачами

Одно из важных различий (а в действительности, возможно, наиболее важное различие) между планировщиком Windows и компонентом UMS программы SQL Server состоит в том, что планировщик Windows является вытесняющим планировщиком, а в компоненте UMS реализована кооперативная модель. Это означает, что операционная система Windows исключает возможность монопольного использования процессора одним потоком. Как было описано в главе 3, каждый поток получает для выполнения определенный промежуток времени, после чего операционная система Windows автоматически отменяет его планирование для выполнения на данном процессоре и разрешает приступить к работе другому потоку, если он уже готов это сделать. В отличие от этого, работа компонента UMS основана на том, что потоки добровольно отказываются от использования процессора и возвращают управление. Если же рабочий поток SQL Server не возвращает управление добровольно, такая ситуация, по всей вероятности, становится препятствием для выполнения работы другими потоками.

У читателя может возникнуть вопрос – почему в компоненте UMS принят такой подход. Те специалисты, которые, подобно автору, уже давно работают в этой области, наверно помнят, что операционная система Windows 3.x действовала именно таким образом – в ней использовался кооперативный планировщик и поэтому для неправильно работающего приложения не составляло никакого труда монопольно захватить всю систему. Именно в этом фактически состояла та причина, по которой операционная система Windows NT с самого начала проектировалась с расчетом на применение вытесняющего планировщика. До тех пор пока сохраняется вероятность нарушения работы всей системы под воздействием единственного приложения, не остается ни малейшей возможности даже приблизиться к цели создания надежной операционной системы.

Но указанный подход принят в компоненте UMS для того, чтобы избежать необходимости использовать привилегированный режим операционной системы Windows, кроме тех ситуаций, когда в этом есть абсолютная необходимость. В той системе, где можно надеяться, что рабочие потоки сами будут возвращать управление, когда это потребуется, кооперативный планировщик может стать фактически более эффективным, чем вытесняющий, поскольку процесс планирования может быть нацелен на удовлетворение конкретных потребностей приложения. Как было сказано выше, компоненту UMS потребности в планировании программы SQL Server известны гораздо лучше, чем самой операционной системе.

Выполнение задач планирования компонентом UMS

Поскольку необходимо, чтобы потребности в планировании программы SQL Server обеспечивал компонент UMS, не позволяя этого делать операционной системе Windows, то компонент UMS должен каким-то образом помешать операционной системе делать то, что она делает со всеми прочими процессами — планировать и депланировать потоки для выполнения на процессоре (процессорах) системы, руководствуясь определенными алгоритмами. Как добиться вывода потоков из-под непосредственного управления вытесняющей операционной системы? В компоненте UMS эта задача решается с помощью некоторых остроумных манипуляций с объектами событий Windows. С каждым потоком, действующим под управлением компонента UMS, связан объект события. Решая задачи планирования, операционная система Windows игнорирует потоки, которые не рассматриваются как применимые для планирования; таковыми считаются потоки, которые нельзя вызвать на выполнение, поскольку они переведены в состояние ожидания на неопределенно долгое время. Такая особенность учитывается в компоненте UMS. Те потоки, которые не должны быть запланированы, компонент UMS переводит в состояние ожидания, вынуждая их вызывать функцию `WaitForSingleObject` для ожидания перехода в сигнальное состояние соответствующего им объекта события и передавать `INFINITE` в качестве значения тайм-аута. Как было указано в главе 3, если поток вызывает функцию `WaitForSingleObject` для ожидания перехода в сигнальное состояние некоторого объекта и передает `INFINITE` в качестве значения тайм-аута, то единственным способом активизации потока является перевод ожидаемого им объекта в сигнальное состояние. Если в компоненте UMS возникает необходимость вызвать на выполнение определенный поток, то данный компонент переводит в сигнальное состояние объект события соответствующего потока. Это позволяет потоку выйти из состояния ожидания, в результате чего операционная система Windows получает возможность запланировать заданный поток для выполнения на одном из процессоров.

Для того чтобы исключить возможность планирования операционной системой Windows сразу нескольких потоков для выполнения на одном и том же процессоре, что приводит к дополнительным издержкам и расходам, связанным с переключением

контекста, в компоненте UMS предпринимается попытка сохранять такое положение, в котором применимыми для планирования (т.е. не находящимися в состоянии ожидания в течение неопределенно долгого времени) остаются только по одному потоку в расчете на один процессор. Из этого правила есть исключения (например, запросы поиска по полному тексту, операции проверки соблюдения требований защиты, вызовы расширенных процедур, связанные серверные запросы и т.д.), но вся система спроектирована для обеспечения возможности одновременного выполнения только по одному потоку в расчете на каждый процессор.

Планировщик UMS

Механизм UMS, предназначенный для управления процессом планирования и обеспечения того, чтобы в каждый конкретный момент времени оставался активным только один поток в расчете на каждый процессор, называется планировщиком. После запуска программы SQL Server создается по одному экземпляру планировщика UMS для каждого процессора на компьютере. По умолчанию не определяется родственность этих планировщиков конкретным процессорам, но алгоритмы планирования Windows действуют таким образом, что со временем каждый поток приобретает свой собственный процессор, поскольку каждый планировщик UMS разрешает выполняться только одному потоку одновременно.

Пул рабочих (микро)потоков для сервера (независимо от того, состоит ли он из потоков или микропотоков) распределяется равномерно по экземплярам планировщиков UMS. Это означает, что при наличии заданного по умолчанию максимального количества рабочих потоков, равного 255, и четырехпроцессорного компьютера в программе SQL Server создаются четыре экземпляра планировщиков UMS, а каждый из них управляет максимальным количеством рабочих (микро)потоков, приблизительно равным 64.

Рабочие (микро)потоки распределяются на сервере по экземплярам планировщиков UMS равномерно, поэтому с увеличением количества экземпляров планировщиков UMS уменьшается вероятность того, что неправильно действующие соединения, поддерживаемые некоторыми из этих планировщиков, станут причиной нарушения организации одновременной работы и возникновения других проблем на сервере.

Например, на восьмипроцессорном компьютере каждый экземпляр планировщика UMS может поддерживать приблизительно 32 рабочих (микро)потока. Если серверный процесс с определенным идентификатором, связанный с конкретным планировщиком, владеет блокировками на ресурсах так, что в связи с этим возникает цепочка блокировок глубиной в 32 процесса (безусловно, такая ситуация иногда встречается на практике), причем оказывается, что процессы из заблокированной цепочки также связаны с тем же планировщиком, то планировщик может стать неспособным реагировать на дальнейшие запросы и потеряет способность обрабатывать новые запросы на выполнение работы. Обработка новых запросов на выполнение работы этим планировщиком фактически приостановится до тех пор, пока не будет устранена проблема блокировки.

Списки планировщика UMS

В каждом экземпляре планировщика UMS сопровождаются пять списков, которые обеспечивают работу по планированию потоков – список рабочих (микро)потоков, список работоспособных (микро)потоков, список ожидающих (микро)потоков, список запросов на ввод-вывод и контролируемый по таймеру список. Каждый из этих списков выполняет отдельную роль, а элементы этих списков часто перемещаются из одного списка в другой.

Список рабочих (микро)потоков

Список рабочих (микро)потоков – это список доступных рабочих (микро)потоков UMS. Рабочий (микро)поток UMS – это унификация представления потока (микропотока), которая позволяет использовать в программе либо поток, либо микропоток, при этом в остальной части кода не требуется учитывать, какой из объектов фактически используется. Как уже было сказано, одной из целей проекта компонента UMS было предоставление поддержки для микропотоков в такой форме, которая не требовала бы учитывать в коде основной машины планировщика, используются ли в системе микропотоки или потоки. Дело в том, что рабочий (микро)поток UMS инкапсулирует в себе поток или микропоток, который выполняет на сервере определенные задачи, и унифицирует эти объекты таким образом, что на сервере (по большей части) не требуется учитывать, происходит ли его функционирование в режиме потоков или в режиме микропотоков. Во всей оставшейся части данной главы будет использоваться термин *рабочий (микро)поток UMS*, а не рабочий поток или рабочий микропоток.

Если программа SQL Server работает в режиме потоков (в режиме, предусмотренном по умолчанию), то рабочий (микро)поток UMS инкапсулирует объект потока Windows. Если же сервер работает в режиме микропотоков, рабочий (микро)поток UMS инкапсулирует микропоток Windows, поддержка которого, как было указано в главе 3, фактически осуществляется за пределами ядра Windows.

Процесс поддержки соединения

После подключения к программе SQL Server нового клиента этому клиенту назначается конкретный экземпляр планировщика UMS. При выборе экземпляра планировщика UMS применяется очень простая эвристическая функция – для нового соединения назначается тот планировщик, с которым связано наименьшее количество соединений. После назначения для соединения некоторого экземпляра планировщика связь между соединением и экземпляром планировщика сохраняется постоянно.

Компонент UMS не передает серверный процесс с определенным идентификатором от одного экземпляра планировщика к другому, независимо от того, насколько сильно загружен экземпляр планировщика, с которым связан этот процесс, и есть ли в системе другие незагруженные экземпляры планировщиков. Это означает, что вполне возможно представить себе такой ход развития ситуации, при котором

поддержка симметричной многопроцессорной обработки в программе SQL Server фактически нарушается, поскольку приложение открывает большое количество постоянных соединений, в которых не выполняется примерно одинаковый объем работы, и в связи с этим нагрузка не распределяется по процессорам равномерно.

Допустим, например, что эксплуатируется двухпроцессорный компьютер, а клиентское приложение SQL Server открывает четыре постоянных соединения с сервером, причем в двух из этих соединений выполняется 90% работы приложения. Если происходит так, что эти два соединения связываются с одним и тем же экземпляром планировщика, то можно обнаружить такую ситуацию, при которой один из процессоров постоянно загружен, в то время как другой остается относительно слабо загруженным. В такой ситуации наиболее приемлемое решение состоит в том, что нагрузка должна распределяться по соединениям равномерно, а если рабочая нагрузка является несбалансированной, то не должны поддерживаться постоянные соединения. Разъединение и повторное соединение — это единственный способ переместить серверный процесс с определенным идентификатором от одного экземпляра планировщика к другому. (Будет ли это перемещение успешным, вовсе не гарантируется — разъединение и повторное соединение некоторого серверного процесса может в конечном итоге привести к связыванию с тем же экземпляром планировщика, в зависимости от того, чему равно количество пользователей, поддерживаемых другими планировщиками.)

После назначения экземпляра планировщика серверному процессу с определенным идентификатором дальнейшие события зависят от состояния списка рабочих (микро)потоков и от того, достигнуто ли значение максимального количества рабочих потоков, заданное в конфигурации программы SQL Server. Если в списке рабочих (микро)потоков имеется доступный рабочий (микро)поток, последний принимает запрос на установление соединения и обрабатывает его. Если же в списке отсутствует доступный рабочий (микро)поток, а пороговое значение максимального количества рабочих потоков не достигнуто, то создается новый рабочий (микро)поток, который обрабатывает поступивший запрос. А если отсутствуют доступные рабочие (микро)потоки и достигнуто значение максимального количества рабочих потоков, то запросы на установление соединения помещаются в список ожидающих (микро)потоков и обрабатываются в порядке очереди со последовательной организацией по мере того, как становятся доступными рабочие (микро)потоки.

Клиентские запросы на установление соединения рассматриваются в компоненте UMS как запросы логических (а не физических) пользователей. Ситуация, в которой количество логических пользователей намного превышает количество рабочих (микро)потоков UMS, является и нормальной, и желательной. Как было указано выше, именно это позволяет в программе SQL Server с установленным значением максимального количества рабочих потоков, равным 255, обслуживать сотни или даже тысячи пользователей.

Запросы на выполнение работы

В компоненте UMS запросы на выполнение работы осуществляются по принципу неразрывной операции. Это означает, что рабочий (микро)поток обрабатывает весь запрос на выполнение работы (например, пакетное задание на языке T-SQL)

и рассматривается как простаивающий только после выполнения указанной работы. Это также означает, что в компоненте UMS не может происходить переключение контекста во время выполнения любого рабочего запроса. Например, в ходе выполнения какого-то конкретного пакетного задания на языке T-SQL рабочий (микро)поток не переключается с него на обработку другого задания. Единственный вариант, в котором рабочий (микро)поток может приступить к обработке другого запроса на выполнение работы, состоит в том, что он вначале полностью завершает обработку текущего запроса на выполнение работы. Он может завершить текущее задание, после чего выполнить, например, процедуру завершения ввода-вывода, которая была первоначально поставлена в очередь другим рабочим (микро)потоком, но не будет рассматриваться как простаивающий до тех пор, пока полностью не обработает свой запрос на выполнение работы, и не будет приступать к обработке следующего запроса на выполнение работы, пока не завершит обработку текущего запроса на выполнение работы. А после того, как это произойдет, рабочий (микро)поток либо активизирует другой рабочий (микро)поток, а сам возвращается в список рабочих (микро)потоков, либо переходит к выполнению строки кода с циклом простоя, если нет других работоспособных рабочих (микро)потоков и не осталось запросов на выполнение работы, как будет вскоре описано более подробно.

Такая неразрывность выполнения запросов является причиной того, что количество рабочих (микро)потоков в программе SQL Server можно увеличить, просто выполнив ряд одновременных запросов с оператором `WAITFOR`, как было сделано с использованием инструментального средства `STRESS.CMD` в главе 3. Пока выполняется каждый запрос `WAITFOR`, обслуживающий его рабочий (микро)поток рассматривается в программе SQL Server как занятый, поэтому для новых запросов, поступающих в сервер, требуется другой рабочий (микро)поток. Если количество запросов указанных типов достаточно велико, то может быть быстро достигнуто значение максимального количества рабочих потоков, а после того как это произойдет, прекратится прием новых запросов на установление соединения до тех пор, пока не освободится хотя бы один из рабочих (микро)потоков.

Если сервер работает в режиме потоков и обнаруживает, что какой-то рабочий (микро)поток простаивал в течение 15 минут, то программа SQL Server уничтожает его при условии, что в результате этого количество рабочих (микро)потоков не станет меньше заранее заданного порогового значения. Это позволяет освободить виртуальную память, требуемую для размещения стека потока простаивающего рабочего (микро)потока (0,5 Мбайт), и применить это пространство виртуальной памяти для каких-то других целей.

Список работоспособных (микро)потоков

Список работоспособных (микро)потоков представляет собой список рабочих (микро)потоков UMS, готовых обработать существующий запрос на выполнение работы. Каждый рабочий (микро)поток в этом списке остается в состоянии ожидания на неопределенно долгое время до тех пор, пока связанный с ним объект события не становится сигнальным. Из того, что некоторый рабочий (микро)поток находится в списке работоспособных (микро)потоков, не следует,

что он может быть сразу же запланирован на выполнение операционной системой Windows. Он будет запланирован операционной системой Windows на выполнение, как только станет сигнальным связанный с ним объект события, в соответствии с алгоритмами, применяемыми в компоненте UMS.

Поскольку известно, что в компоненте UMS реализован принцип работы кооперативного планировщика, возникает вопрос о том, какой объект фактически отвечает за передачу одному из рабочих (микро)потоков, находящихся в списке работоспособных (микро)потоков, сигнала о наступлении ожидаемого события с тем, чтобы он мог приступить к работе. Ответ состоит в том, что такое действие может выполнить любой рабочий (микро)поток UMS. В программном обеспечении SQL Server предусмотрено немало функций, позволяющих возвращать управление компоненту UMS, для того, чтобы любая конкретная операция не монополизировала главный планировщик этого компонента. В компоненте UMS предусмотрен целый ряд типов функций возврата выполнения, которые могут быть вызваны рабочими (микро)потоками. Как уже было сказано выше, в среде кооперативного управления задачами потоки должны добровольно возвращать управление друг другу, чтобы система могла работать бесперебойно. Программа SQL Server спроектирована таким образом, чтобы в ней возврат управления происходил настолько часто, насколько это необходимо, и в нужных местах, чтобы работа системы продолжалась бесперебойно.

После того как в одном из рабочих (микро)потоков UMS возникает необходимость возвратить управление (либо потому, что он завершил выполнение стоящей перед ним задачи, например, обработал пакетное задание T-SQL или выполнил вызов RPC, либо потому, что в этом рабочем (микро)потоке был выполнен код с явным вызовом одной из функций возврата управления UMS), он обязан проверить в списке работоспособных (микро)потоков планировщика наличие готовых рабочих (микро)потоков и отметить событие, связанное с этим рабочим (микро)потоком как сигнальное, чтобы последний мог приступить к работе. Такая проверка осуществляется в самой процедуре возврата управления. Таким образом, в результате вызова одной из функций возврата управления компонента UMS рабочий (микро)поток фактически выполняет работу компонента UMS; это означает, что в планировщике нет ни одного потока, который отвечал бы за управление его работой. Дело в том, что если существовал бы такой поток, то операционная система Windows должна была бы планировать его для выполнения на процессоре каждый раз, когда в планировщике должно было бы произойти какое-то событие. При такой организации работы не было бы достигнуто никаких преимуществ по сравнению с тем, как всю работу по планированию осуществляла бы сама операционная система Windows. А в действительности дела обстояли бы еще хуже из-за конкуренции за такой гипотетический поток-планировщик, а также в связи с тем, что сам код UMS вносил бы дополнительные издержки. Но так как фактически любому рабочему (микро)потоку разрешено управлять работой планировщика, каждый поток, работающий на процессоре, уже имеет возможность продолжать свою работу до тех пор, пока он может это делать. В этом состоит фундаментальное требование к проекту механизма планирования, который предназначен для минимизации переключений контекста. Как уже было отмечено в главе 5 при обсуждении

рассматриваемого в ней планировщика, основанного на использовании порта завершения ввода-вывода, в планировщике, предназначенном для минимизации переключений контекста, должна быть предусмотрена развязка между очередью запросов на выполнение работы и рабочими (микро)потоками, которые выполняют работу, обусловленную в запросах. В идеальной ситуации выполнение работы, предусмотренной любым запросом, может взять на себя любой поток. Благодаря этому поток, который уже запланирован на выполнение операционной системой, может оставаться запланированным и продолжать работать до тех пор, пока для него еще остается невыполненная работа. Таким образом, устраняются непроизводительные издержки, связанные с планированием любого потока для выполнения той работы, которую может сделать уже работающий поток.

Список ожидающих (микро)потоков

В списке ожидающих (микро)потоков поддерживается очередь рабочих (микро)потоков, ожидающих освобождения того или иного ресурса. После того как рабочий (микро)поток UMS обнаруживает, что его запрос относится к ресурсу, принадлежащему другому рабочему (микро)потоку, он помещает сам себя в список ожидающих (микро)потоков, чтобы дождаться освобождения ресурса и своего планировщика, для чего переходит на неопределенное время в состояние ожидания перехода в сигнальное состояние объекта события, связанного с этим ресурсом. Рабочий (микро)поток, которому принадлежит требуемый ресурс, после того, как будет готов освободить этот ресурс, обязан просмотреть список рабочих (микро)потоков, ожидающих освобождения ресурса, и переместить эти (микро)потоки в список работоспособных (микро)потоков должным образом. А после того как владелец ресурса достигнет точки возврата управления, он обязан перевести в сигнальное состояние то событие, которого ожидает первый рабочий (микро)поток в списке работоспособных (микро)потоков, для того, чтобы этот рабочий (микро)поток мог приступить к работе. Это означает, что при освобождении ресурса каждый рабочий (микро)поток должен также полностью взять на себя выполнение задачи по перемещению рабочих (микро)потоков, ожидающих освобождения ресурса, из списка ожидающих (микро)потоков в список работоспособных (микро)потоков и передачу одному из них сигнала о том, что он может приступить к выполнению.

Список запросов ввода-вывода

В списке запросов ввода-вывода сопровождается очередь невыполненных запросов на осуществление операций асинхронного ввода-вывода. Эти запросы оформляются в виде объектов запросов ввода-вывода UMS. После инициализации в программе SQL Server запроса ввода-вывода UMS компонент UMS проходит по одному из двух путей в кода в зависимости от того, в какой версии Windows он эксплуатируется. Если компонент UMS эксплуатируется в операционной системе Windows 9x или Windows ME, то в нем инициализируется синхронная операция ввода-вывода (в версиях Windows 9x и ME асинхронный файловый ввод-вывод не поддерживается). Если же этот компонент UMS эксплуатируется в семействе Windows NT, то он инициализирует операцию асинхронного ввода-вывода.

Как было описано выше в данной книге при обсуждении средств ввода-вывода Windows, если в потоке возникает необходимость выполнить одну из операций ввода-вывода асинхронно, то в вызовах функций ReadFile/ReadFileEx или WriteFile/WriteFileEx должна быть передана структура OVERLAPPED. Первоначально в операционной системе Windows значение элемента Internal этой структуры устанавливается равным STATUS_PENDING для указания на то, что рассматриваемая операция ввода-вывода находится на этапе выполнения. До тех пор пока данная операция продолжается, вызов функции HasOverlappedIoCompleted API-интерфейс Win32 возвращает ложное значение. (Фактически HasOverlappedIoCompleted – не функция, а макрокоманда, которая просто проверяет значение элемента OVERLAPPED.Internal для определения того, имеет ли он все еще значение STATUS_PENDING.)

Для того чтобы инициализировать запрос на осуществление операции асинхронного ввода-вывода с помощью компонента UMS, программа SQL Server создает экземпляр объекта запроса ввода-вывода UMS и передает этот экземпляр в метод, который семантически аналогичен функции ReadFile/ReadFileScatter или WriteFile/WriteFileGather, в зависимости от того, должна ли быть осуществлена операция чтения или записи, а также от того, должна ли эта операция ввода-вывода быть выполнена по принципу сборки-разборки. Запрос ввода-вывода UMS представляет собой структуру, которая инкапсулирует запрос на осуществление операции асинхронного ввода-вывода и содержит в качестве одного из своих элементов структуру OVERLAPPED. Метод асинхронного ввода-вывода компонента UMS, вызванный сервером, передает эту структуру OVERLAPPED в соответствующую функцию асинхронного ввода-вывода Win32 (например, ReadFile) для использования в требуемой операции асинхронного ввода-вывода. После этого структура, содержащая запрос ввода-вывода UMS, помещается в список запросов ввода-вывода, с которым работает хост-планировщик.

После добавления одного из запросов ввода-вывода к списку запросов ввода-вывода любой рабочий (микро)поток, который должен вернуть управление, обязан проверить этот список для определения того, была ли завершена какая-либо операция асинхронного ввода-вывода. Для этого рабочий (микро)поток просто проходит по списку запросов ввода-вывода и вызывает для каждого из этих запросов макрокоманду HasOverlappedIoCompleted, передавая в нее элемент OVERLAPPED рассматриваемого запроса ввода-вывода. Найдя запрос, который был завершен к этому времени, рабочий (микро)поток удаляет его из списка запросов ввода-вывода, затем вызывает процедуру завершения ввода-вывода, связанную с запросом. Процедура завершения ввода-вывода определяется при первоначальном создании запроса ввода-вывода UMS.

Как было указано выше при обсуждении асинхронного ввода-вывода, после завершения одной из операций асинхронного ввода-вывода операционная система Windows может дополнительно поставить в очередь вызывающего потока, который был инициатором запроса, вызов APC завершения ввода-вывода. Кроме того, как описано выше, одной из целей проектирования UMS было предоставление в основном таких же функциональных средств планирования и организации асинхронного ввода-вывода, которые предусмотрены в ядре операционной системы,

но без необходимости переключения в привилегированный режим. Предусмотренная в компоненте UMS поддержка процедур завершения ввода-вывода представляет собой еще один пример применения указанного принципа проектирования. Основное различие между тем, как процедуры завершения ввода-вывода выполняются в операционной системе Windows и в компоненте UMS, состоит в том, что в компоненте UMS процедура завершения ввода-вывода выполняется в контексте любого рабочего (микро)потока, который предпринимает попытку возвратить выполнение (и поэтому проверяет наличие в списке запросов ввода-вывода информации о завершенных операциях ввода-вывода), тогда как в операционной системе Windows такая процедура всегда выполняется в контексте того потока, который первоначально инициализировал рассматриваемую операцию асинхронного ввода-вывода. Преимущество подхода, принятого при проектировании компонента UMS, состоит в том, что для выполнения процедуры завершения ввода-вывода не требуется переключение контекста. Обязанности по вызову этой процедуры до того, как перейти в состояние ожидания, берет на себя рабочий (микро)поток, который уже выполняется, но готовится к возврату управления. В связи с этим не требуется какое-либо взаимодействие с ядром Windows.

При эксплуатации планировщика UMS в версии Windows 9x или ME процедура завершения ввода-вывода вызывается непосредственно после вызова функции ввода-вывода API-интерфейса Win32. А поскольку операция ввода-вывода осуществляется операционной системой синхронно, то нет необходимости обращаться к списку запросов ввода-вывода и привлекать, возможно, другой рабочий (микро)поток к фактическому выполнению процедуры завершения ввода-вывода. Поскольку известно, что после возврата из вызова функции API-интерфейса Win32 операция ввода-вывода уже завершена, то можно просто продолжить работу и вызвать процедуру завершения ввода-вывода перед возвратом из вызова метода ввода-вывода UMS. Это означает, что в версии Windows 9x/ME процедура завершения ввода-вывода всегда вызывается в контексте рабочего (микро)потока, который сам инициализировал соответствующую операцию асинхронного ввода-вывода.

Контролируемый по таймеру список

Контролируемый по таймеру список применяется для сопровождения очереди контролируемых по таймеру запросов UMS. Контролируемый по таймеру запрос инкапсулирует регламентированный по времени запрос на выполнение работы. Например, если в рабочем (микро)потоке возникает необходимость перейти в состояние ожидания освобождения ресурса на определенное количество времени и после этого завершить свою работу по тайм-ауту, то такой рабочий (микро)поток добавляется к контролируемому по таймеру списку. После того как рабочий (микро)поток становится готовым возвратить управление, в нем происходит проверка истекших установок таймера в контролируемом по таймеру списке вслед за проверкой наличия завершенных запросов ввода-вывода. Если этот поток находит запрос с истекшей установкой таймера, то удаляет его из контролируемого по таймеру списка и перемещает связанный с данным запросом рабочий (микро)поток в список работоспособных (микро)потоков. Если при осуществлении

этого действия список работоспособных (микро)потоков пуст (а это означает, что больше ни один из рабочих (микро)потоков не готов приступить к работе), то рассматриваемый рабочий (микро)поток переводит также в сигнальное состояние событие, связанное с обнаруженным им рабочим (микро)потоком с истекшей установкой таймера, чтобы последний мог быть запланирован операционной системой Windows на выполнение.

Цикл простоя

Если после проверки наличия завершенных запросов ввода-вывода и истекших установок таймера рабочий (микро)поток обнаруживает, что список работоспособных (микро)потоков пуст, то входит в своего рода цикл простоя. Он просматривает контролируемый по таймеру список для того, чтобы найти очередное истекшее значение таймера, а затем переходит с помощью вызова функции `WaitForSingleObject` в состояние ожидания перехода в сигнальное состояние объекта события, который связан с самим планировщиком; для этого используется значение тайм-аута, равное времени истечения очередной установки таймера. Как было указано при обсуждении асинхронного ввода-вывода, структура `OVERLAPPED` подсистемы Win32 содержит элемент события, который может хранить ссылку на объект события Windows. При создании одного из планировщиков UMS создается объект события и связывается с самим созданным планировщиком. После того как планировщик инициализирует запрос на выполнение операции асинхронного ввода-вывода, это событие сохраняется в элементе `hEvent` структуры `OVERLAPPED`, которая относится к объекту запроса ввода-вывода. В результате этого завершение операции асинхронного ввода-вывода вызывает переход в сигнальное состояние данного объекта события, относящегося к планировщику. В ходе того как рабочий (микро)поток ожидает перехода в сигнальное состояние этого объекта события с помощью тайм-аута, установленного равного времени истечения следующей установки таймера, данный рабочий (микро)поток фактически ожидает либо выполнения запроса ввода-вывода, либо истечения установки таймера, в зависимости от того, какое из этих событий произойдет раньше. А поскольку такое ожидание происходит в результате вызова функции `WaitForSingleObject`, то не происходит какой-либо опрос и не используются ресурсы процессора до тех пор, пока не произойдет одно из указанных двух событий.

Переход в режим вытеснения

Некоторые операции, выполняемые в программе SQL Server, могут потребовать, чтобы один из рабочих (микро)потоков "перешел в режим вытеснения" (как принято называть это действие), т.е. был выведен из-под контроля планировщика. Примером такой ситуации является вызов одной из расширенных процедур. Как уже было сказано, UMS – это среда кооперативной многозадачной организации работы, поэтому ее функционирование основано на том, что рабочие (микро)потоки возвращают управление в соответствии с правилами, заложенными в коде компонента UMS,

для того, чтобы работа сервера продолжалась бесперебойно. Очевидно, что нет ни малейшей гарантии того, что расширенная процедура будет возвращать управление согласно каким-либо установленным правилам. К тому же, в действительности не существует каких-либо документированных функций API-интерфейса ODS, которые могли быть вызваны в расширенной процедуре для осуществления операции возврата управления планировщику. Поэтому планировщик действует на основании предположения, что для расширенной процедуры требуется собственный поток, в котором она могла бы выполняться. В связи с этим, прежде чем в каком-либо рабочем (микро)потоке вызывается расширенная процедура, этот рабочий (микро)поток удаляет очередной работоспособный (микро)поток из списка работоспособных (микро)потоков и задает свое событие так, чтобы планировщик мог продолжить свою работу по передаче одному из рабочих (микро)потоков для обработки очередных запросов на выполнение работы. Между тем первоначальный рабочий (микро)поток выполняет расширенную процедуру и, по существу, игнорируется планировщиком до тех пор, пока в нем не будет осуществлен возврат. После возврата из расширенной процедуры рабочий (микро)поток продолжает обработку своего запроса на выполнение работы (например, оставшейся части пакетного задания T-SQL, в котором была вызвана расширенная процедура), затем возвращает сам себя в список рабочих (микро)потоков, как только станет простаивающим (как было указано выше).

При этом наиболее важная особенность выполняемых действий состоит в следующем: для некоторых операций, осуществляемых в сервере, требуются собственные рабочие (микро)потоки, поэтому возникает вероятность того, что на какое-то время в расчете на один планировщик могут стать активными несколько потоков (а следовательно, несколько потоков станут активными в расчете на один процессор компьютера, поскольку логические планировщики часто закрепляются за отдельными процессорами). Это означает, что операционная система Windows, как обычно, должна планировать эти потоки для выполнения на процессоре по принципу вытеснения, поэтому вполне вероятно, что между такими потоками будет происходить переключение контекста. Это также означает, что к выполнению расширенной процедуры фактически привлекается отдельный рабочий (микро)поток UMS, поэтому вызов на выполнение большого количества расширенных процедур может оказать весьма отрицательное влияние на такие показатели, как масштабируемость и степень распараллеливания. Каждая выполняемая расширенная процедура снижает способность компонента UMS обслуживать большое количество логических пользователей с помощью относительно малого количества рабочих (микро)потоков.

Переход рабочего (микро)потока в режим вытеснения может быть вызван не только выполнением расширенных процедур, но и некоторыми другими действиями. В качестве примеров таких действий можно указать вызовы `sp_OA`, связанные серверные запросы, распределенные запросы, междусерверные вызовы RPC, отладку T-SQL и многие другие действия. Безусловно, следует избегать выполнения этих действий при любой возможности, если вы стремитесь в основном добиться повышения масштабируемости и обеспечить эффективное использование ресурсов.

Режим микропотоков

Если сервер работает в режиме микропотоков, то дела обстоят немного иначе. Как было указано в главе 3, микропоток — это понятие, относящееся к непривилегированному режиму; ядро не получает никакой информации о функционировании микропотоков. Поскольку фактически единственным механизмом выполнения кода в операционной системе Windows является поток, то код, выполняемый с помощью микропотока, все равно в какой-то момент должен быть выполнен с помощью потока. Такая организация работы основана на том, что с помощью функций API-интерфейсов управления микропотоками операционной системы Windows группа микропотоков связывается с одним объектом потока. При выполнении фрагмента кода одним из микропотоков этот код фактически выполняется с помощью относящегося к нему базового потока. Это означает, что за переключение на другой микропоток для того, чтобы он мог выполнить свою работу, отвечает код непривилегированного режима; этот принцип весьма напоминает принцип кооперативного управления задачами, который реализован в компоненте UMS.

С учетом того, что при эксплуатации программы SQL Server в режиме микропотоков единственный поток Windows может использоваться несколькими рабочими (микро)потоками, процедура, применяемая при переводе рабочего микропотока в режим вытеснения, не будет работать, когда потребуется переключиться в режим вытеснения при использовании некоторого рабочего микропотока. Дело в том, что механизмом выполнения в операционной системе Windows все еще является поток, поэтому тот поток, который является базовым для такого микропотока, должен быть выведен из-под управления планировщика, а это, в свою очередь, повлечет за собой вывод из-под управления планировщика всех других рабочих микропотоков, для которых базовым является тот же поток, а эта ситуация весьма нежелательна.

Вместо этого происходит следующее: создается скрытый планировщик, рассчитанный на отдельный поток, который предназначен для обслуживания расширенных процедур и других внешних вызовов, которые вынуждают рабочий (микро)поток переключиться в режим вытеснения. (Этот планировщик является скрытым в том смысле, что он не обнаруживается в выводе команды DBCC SQLPERF(umsstats).) Если в одном из рабочих микропотоков возникает необходимость переключиться в режим вытеснения, чтобы выполнить одно из указанных действий (наподобие вызова расширенной процедуры), то запрос на выполнение работы передается такому скрытому планировщику и обрабатывается. После завершения подобного действия микропоток снова возвращается к первоначальному планировщику, и обработка запросов продолжается как обычно.

Недостатком такой организации работы является то, что выполнение действий, подобных вызовам расширенных процедур и связанным серверным запросам, в режиме микропотоков может стать крайне неэффективным. Фактически в сервере имеется целый ряд компонентов, которые даже не поддерживаются в режиме микропотоков (например, sp_xml_preparedocument и ODSOLE). Если в ходе эксплуатации программы SQL Server требуется выполнять большое количество вызовов расширенных процедур, связанных серверных запросов, распределенных транзакций и подобных действий, то режим микропотоков может оказаться не совсем приемлемым.

Скрытые планировщики

В сервере скрытые планировщики создаются также для других целей. Такие же защелки, средства управления ресурсами и службы планирования, которые применяются в компоненте UMS для планирования запросов на выполнение работы, требуются и в других операциях, осуществляемых на сервере, поэтому сервер создает скрытые планировщики, позволяющие использовать указанные функциональные средства в подобных операциях без необходимости реализовывать их самостоятельно. Примером подобного средства является средство резервного копирования и восстановления программы SQL Server. Учитывая то, что многие устройства резервного копирования не поддерживают асинхронный ввод-вывод, а также тот факт, что выполнение большого объема синхронного ввода-вывода с помощью обычного планировщика UMS отрицательно сказывается на показателе степени распараллеливания всего планировщика, поскольку может привести к тому, что единственный блокирующий синхронный вызов ввода-вывода монополизирует весь рабочий (микро)поток (во многом аналогично тому, что происходит при вызове внешнего кода), в программе SQL Server операции резервного копирования и восстановления передаются в их собственный планировщик. Это позволяет добиться того, чтобы эти операции конкурировали за процессорное время друг с другом, а операционная система Windows планировала их работу в режиме вытеснения наряду с операциями, выполняемыми с помощью других планировщиков.

Команда DBCC SQLPERF(umsstats)

Команда `DBCC SQLPERF(umsstats)` уже упоминалась выше, к тому же, читатель уже мог знать о ее существовании, поскольку о ней многое сказано в общедоступной базе знаний Microsoft (Microsoft Knowledge Base), хотя сама эта команда не документирована. Команда `DBCC SQLPERF(umsstats)` позволяет получить результирующий набор, представляющий собой листинг статистических данных, которые относятся к планировщикам UMS, видимым в системе. Эта команда позволяет определить общее количество пользователей и рабочих (микро)потоков, относящихся к данному планировщику, количество рабочих (микро)потоков в списке работоспособных (микро)потоков, количество простаивающих рабочих (микро)потоков, количество подлежащих обработке запросов на выполнение работы и т.д. Это очень удобно, когда возникает подозрение, что в планировщике обнаружились какие-то проблемы, и требуется узнать, что фактически происходит. Например, вывод этой команды позволяет быстро определить, не достигнуто ли в планировщике максимально допустимое количество рабочих (микро)потоков, а также заняты ли они в настоящее время. В табл. 10.1 приведены подробные сведения о результирующем наборе, возвращаемом командой `DBCC SQLPERF(umsstats)`.

Вывод команды `DBCC SQLPERF(umsstats)` проще всего описать на примере. Предположим, что речь идет о программе SQL Server, которая эксплуатируется в однопроцессорной системе. В этой программе должен работать только один

видимый планировщик UMS, поэтому максимальное количество рабочих (микро)потоков этого планировщика будет равно значению параметра максимального количества рабочих (микро)потоков процедуры sp_configure. Если вывод команды DBCC SQLPERF(umsstats) показывает, что количество рабочих (микро)потоков, связанных с планировщиком, уже достигло значения максимального количества рабочих (микро)потоков и остается неизменным, а также обнаружится, что в столбце помещенных в очередь запросов на выполнение работы постоянно присутствует ненулевое значение, то на полном основании может быть сделан вывод, что планировщик очень сильно загружен, и в нем могут возникать сложности со своевременным выполнением работы.

Таблица 10.1. Поля в выводе команды DBCC SQLPERF(umsstats)

Статистический показатель	Описание
Scheduler ID	Идентификационный номер планировщика с отсчетом от нуля
num users	Количество пользовательских соединений, связанных с планировщиком
num runnable	Количество рабочих (микро)потоков в списке работоспособных (микро)потоков
num workers	Общее количество рабочих (микро)потоков, связанных с планировщиком
idle workers	Количество простаивающих рабочих (микро)потоков
work queued	Количество элементов, ожидающих обработки в очереди запросов на выполнение работы
cntxt switches	Количество переключений между рабочими (микро)потоками, относящимися к рассматриваемому планировщику
cntxt switches(idle)	Количество случаев перехода в цикл простоя
Scheduler Switches	Количество переключений между планировщиками (не используется)
Total Work	Общее количество единиц работы, выполненных всеми планировщиками

Резюме

Начиная с версии 7.0, в программе SQL Server в целях повышения масштабируемости и поддержки микропотоков Windows применяются собственные средства планирования, основанные на использовании компонента UMS. Компонент UMS выполняет функции тонкой прослойки между сервером и операционной системой, которая предоставляет в основном такие же функциональные возможности, которые поддерживаются с помощью примитивов многопотоковой обработки и планирования подсистемы Win32, но осуществляет это, не требуя такого же количества переходов в привилегированный режим или такого же количества переключений контекста.

Основное различие между планировщиком UMS и планировщиком Windows обусловлено тем, что планировщик UMS является кооперативным. Его функционирование основано на том, что рабочие (микро)потоки добровольно возвращают управление достаточно часто для того, чтобы поддерживалась бесперебойная работа системы. А поскольку контроль над тем, когда должно осуществляться планирование потока под управлением сервера, теперь возложен на один из компонентов программы SQL Server, разработчики этой программы несут гораздо большую ответственность за то, чтобы созданный ими код работал эффективно и возвращал управление достаточно часто и в подходящих местах. Но такая организация работы обеспечивает также гораздо более тонкую детализацию управления и позволяет масштабировать операции сервера гораздо лучше по сравнению с тем, на что можно было бы надеяться при использовании предусмотренного в операционной системе Windows подхода к планированию (с единственной меркой, применяемой ко всем потокам), поскольку программа SQL Server способна лучше всего учитывать собственные потребности в планировании.

Вопросы для самопроверки

1. Какой механизм, относящийся к компоненту UMS, воплощает в себе средство планирования, предназначенное для одного логического процессора?
2. Какой список, поддерживаемый в компоненте UMS, позволяет следить за рабочими (микро)потоками, которые готовы к выполнению?
3. Какой объект привилегированного режима используется в компоненте UMS для перевода рабочих (микро)потоков в состояние ожидания, если в какой-то момент не нужно, чтобы они находились в состоянии выполнения?
4. Подтвердите или опровергните следующее утверждение. В компоненте UMS для каждого процессора предусмотрен специальный поток, который обеспечивает планирование рабочих (микро)потоков для выполнения, обрабатывает рабочие (микро)потоки с истекшей установкой таймера и т.д.
5. Подтвердите или опровергните следующее утверждение. По умолчанию при эксплуатации программы SQL Server на многопроцессорном компьютере определяется родственность каждого экземпляра UMS по отношению к какому-то конкретному процессору.
6. Какой список, поддерживаемый в компоненте UMS, обеспечивает отслеживание подлежащих выполнению запросов ввода-вывода?
7. Какому компоненту UMS в конечном итоге принадлежит объект события, который становится сигнальным после завершения выполнения запроса асинхронного ввода-вывода?
8. Опишите, что происходит в компоненте UMS, если программа SQL Server находится в режиме микропотоков и на выполнение вызывается расширенная процедура.

9. Допустим, что сервер находится в режиме потоков. Назовите средство, относящееся к серверу, в котором используется скрытый экземпляр средства планирования UMS.
10. Подтвердите или опровергните следующее утверждение. В версии Windows 9x все операции ввода-вывода UMS выполняются программой SQL Server в синхронном режиме.
11. После того как один из рабочих (микро)потоков возвращает управление, какой компонент берет на себя ответственность за проверку контролируемого по таймеру списка для определения того, являются ли истекшими какие-либо установки таймеров?
12. Опишите, в чем суть ситуации, при которой один из рабочих (микро)потоков UMS "переходит в режим вытеснения".
13. Подтвердите или опровергните следующее утверждение. Одной из первоначальных целей проектирования UMS было обеспечение возможности использовать микропотоки Windows в программе SQL Server.
14. Подтвердите или опровергните следующее утверждение. Наиболее важное различие между планированием UMS и планированием Windows состоит в том, что операционная система Windows предоставляет кооперативное средство планирования, а компонент UMS — вытесняющее.
15. Какой список используется в компоненте UMS для отслеживания простаивающих рабочих (микро)потоков?
16. Объясните, что подразумевается под утверждением, что поток, переведенный в состояние ожидания на неопределенно долгое время, не рассматривается планировщиком Windows как "применимый для обработки".
17. В какой версии Windows впервые появилась поддержка микропотоков?
18. Если какой-то рабочий (микро)поток находится в списке, который используется в компоненте UMS для отслеживания рабочих (микро)потоков, готовых для выполнения, то что должно произойти для того, чтобы этот рабочий (микро)поток действительно стал применимым для планирования на выполнение в операционной системе Windows?
19. Подтвердите или опровергните следующее утверждение. Вызов на выполнение связанного серверного запроса приводит к тому, что возникает необходимость переключить один из рабочих (микро)потоков UMS в вытесняющий режим.
20. Подтвердите или опровергните следующее утверждение. Параметр с определением максимального количества рабочих потоков программы SQL Server определяет максимальное количество рабочих (микро)потоков в расчете на каждый экземпляр средства планирования UMS, поэтому на компьютере с двумя процессорами суммарное значение максимального количества рабочих потоков может быть по умолчанию равно 510.

ГЛАВА 6

Управление памятью программы SQL Server

В настоящей главе рассматривается архитектура управления памятью программы SQL Server. Применяемый в приложении способ управления такими важными ресурсами, как память, может многое сказать о том, как спроектировано данное приложение. В частности, изучая эти свойства приложения, можно определить, какой приоритет проектировщики приложения придают эффективному использованию ресурсов и максимальному повышению производительности приложения. Как будет показано в приведенном ниже описании, разработчики программы SQL Server уделяют максимум внимания эффективному управлению памятью и повышению производительности системы. Значительная часть сложного кода этого программного продукта посвящена обеспечению экономичного и эффективного управления памятью. Решение задач управления памятью всегда связано с применением компромиссов. Если в приложении используется слишком мало памяти, то приложение экономит ресурсы памяти, но работает медленнее. А приложение, в котором применяется слишком большой объем памяти, работает быстро, но может помешать функционированию других приложений и в целом стать единственным потребителем общих ресурсов. Как будет показано в этой главе, разработчики программы SQL Server предприняли попытку найти равновесие между стремлением больше взять из ресурсов, доступных этой программе, и, с другой стороны, обеспечить ее успешное функционирование наряду с другими приложениями в системе.

Области памяти

В программе SQL Server распределяемая память состоит из двух отдельных областей — BPool (сокращение от buffer pool — пул буферов) и MemToLeave (сокращение от memory to leave — зарезервированная память). А если в операционной системе используется память AWE, то фактически существует и третья область — физическая память, выходящая за пределы 3 Гбайт, доступ к которой предоставляется средствами поддержки AWE операционной системы Windows (подробные сведения о поддержке AWE приведены в главе 4).

Область BPool является самой важной из трех указанных областей. В этой области находится основной пул распределения памяти программы SQL Server. Область MemToLeave представляет собой пространство виртуальной памяти

в пространстве адресов непривилегированного режима, которое не используется областью BPool. Память AWE, выходящая за пределы 3 Гбайт, функционирует как расширение BPool и предоставляет дополнительное пространство для кэширования страниц данных и страниц индексов.

Определение размеров областей памяти

После запуска сервера прежде всего происходит вычисление верхнего предела области BPool. Этот верхний предел представляет собой максимальный размер, до которого сервер допускает расширение области BPool. В системе без поддержки AWE этот размер устанавливается равным объему физической памяти на компьютере или размеру пространства адресов непривилегированного режима за вычетом размера области MemToLeave, в зависимости от того, какое из этих значений размера меньше. (Если значение параметра максимального объема памяти сервера `max server memory` процедуры `sp_configure` изменено по сравнению с применяемым по умолчанию и установлено меньшим или равным объему физической памяти на компьютере, то вместо значения, полученного путем вычислений, применяется указанное значение.) Поэтому, если в системе установлен 1 Гбайт физической памяти, то размер BPool будет принят равным 1 Гбайт при условии, что значение параметра `max server memory` не откорректировано.

В системе с поддержкой AWE верхний предел области BPool устанавливается равным либо суммарному объему физической памяти на компьютере, либо значению параметра `max server memory`, в зависимости от того, какое из этих значений меньше. Если используются средства AWE, то размер области BPool не ограничивается в зависимости от размера пространства адресов непривилегированного режима или размера области MemToLeave.

По умолчанию размер области MemToLeave устанавливается равным 384 Мбайт. Из этого объема 128 Мбайт резервируются для стеков рабочих потоков (поскольку для каждого стека потока требуется 0,5 Мбайт, а максимальное количество рабочих потоков равно 255); остальные 256 Мбайт резервируются для операций распределения, выходящих за пределы области BPool. К примерам операций распределения памяти, в которых используется область MemToLeave, относятся операции распределения средства доступа OLE DB, операции распределения памяти для внутрипроцессных COM-объектов, а также любые операции распределения памяти, используемые в коде самого сервера, для которых требуется объем больше 8 Кбайт. Последний тип распределения памяти важен, поскольку означает, что память для больших процедур или планов выполнения может быть распределена из области MemToLeave.

Верно, что для всех операций распределения непрерывных блоков памяти с объемом больше 8 Кбайт память берется из области MemToLeave, но обратное утверждение не всегда верно. Поскольку сервер предпринимает попытки использовать незакрепленную часть любого зарезервированного блока для удовлетворения других запросов на распределение памяти, возможно, что операции распределения объема памяти, меньше 8 Кбайт, в конечном итоге будут выполнены за

счет области MemToLeave, в зависимости от используемой версии программы SQL Server и установленного в ней служебного пакета. Для того чтобы выдать запрос на распределение памяти, любой потребитель памяти в сервере вначале создает объект распределения памяти, который затем используется для выдачи запроса на распределение (как будет описано ниже в данной главе, в объекте распределения реализуется интерфейс `IMalloc` технологии COM). Если через какой-то конкретный объект распределения выдается несколько запросов, то существует вероятность того, что некоторые из них будут выполнены с использованием памяти MemToLeave, даже если в них запрашивается меньше 8 Кбайт. Например, если один из потребителей в сервере запрашивает 10 Кбайт непрерывного объема памяти, то объект распределения распределяет этот объем из области MemToLeave. В случае необходимости объект распределения вначале резервирует область, размер которой устанавливается в соответствии со степенью детализации распределения в системе (которая составляет 64 Кбайт в 32-битовых версиях Windows), затем закрепляет две страницы по 8 Кбайт для удовлетворения этого запроса. Если же потребитель памяти затем использует тот же объект распределения для выдачи запроса, скажем, на 4 Кбайт дополнительного пространства, то система обнаруживает нераспределенный объем пространства 6 Кбайт в конце только что распределенной двухстраничной области и удовлетворяет новый запрос за счет этого пространства. Поэтому существует вероятность того, что запросы на распределение объема памяти меньше 8 Кбайт будут удовлетворены за счет области, отличной от BPool.

Средства доступа OLE DB, COM-объекты и другие внешние потребители памяти, которые могут действовать в рамках процесса SQL Server, не имеют никакой информации об области Bpool или других средствах управления памятью программы SQL Server, поэтому очень важно, чтобы в сервере был зарезервирован некоторый объем свободной виртуальной памяти в пространстве непривилегированного режима. С этим связано само существование области MemToLeave – она, по существу, представляет собой неиспользуемую память в пространстве процесса SQL Server. Если какой-то внутрипроцессный COM-объект или другой внешний потребитель памяти самостоятельно вызывает функцию `VirtualAlloc` или `HeapAlloc` для распределения памяти, то удовлетворение соответствующего запроса осуществляется за счет пространства адресов виртуальной памяти. Дело в том, что если бы все это пространство адресов непривилегированного режима сводилось только к области BPool, то запросы на распределение памяти указанного типа всегда оканчивались бы неудачей.

Размер области MemToLeave можно откорректировать с помощью параметра командной строки `-g`. Общий объем памяти, требуемый для стеков рабочих потоков, невозможно изменить, не корректируя значение параметра `max worker threads` (максимальное количество рабочих потоков) процедуры `sp_configure`, а также не модифицируя установленный по умолчанию размер стека потока с помощью внесения изменений в исполняемый файл программы SQL Server, что в любом случае, разумеется, не следует делать. Но существует возможность увеличить или уменьшить объем 256 Мбайт, зарезервированный для операций распределения, выходящих за пределы области BPool, передавая другое значение

параметра -g. Этот параметр может оказаться удобным в тех ситуациях, когда применяется большое количество связанных серверных запросов, внутрипроцессных COM-объектов или других потребителей памяти, конкурирующих за пространство в области MemToLeave. Увеличив размеры этой области, можно предоставить этим потребителям памяти больший объем памяти для работы. И наоборот, уменьшая размер области MemToLeave, можно предоставить больший объем пространства виртуальной памяти для области Bpool, что способствует повышению производительности в некоторых ситуациях.

Даже несмотря на то, что размер области BPool можно устанавливать с учетом объема физической памяти на компьютере, по умолчанию эта область все еще базируется полностью в виртуальной памяти. Исключением из этого правила является такая ситуация, когда для этой области используется память AWE. В таком случае одна часть области BPool располагается в виртуальной памяти (но отображается на страницы, зафиксированные в физической памяти), а другая часть находится в физической памяти, выходящей за пределы 3 Гбайт. Средства AWE предоставляют для процесса непривилегированного режима единственный способ получить доступ к памяти с объемом больше 3 Гбайт. Поскольку максимальный объем пространства виртуальных адресов непривилегированного режима равен 3 Гбайт (даже если разрешена опция /3GB или /USERVA), то использование средств AWE – это единственный способ предоставить процессу доступ к памяти, выходящей за пределы 3 Гбайт. Но отличительная особенность памяти AWE заключается в том, что она – физическая, а не виртуальная и для получения доступа к этой памяти ее необходимо отобразить на пространство адресов непривилегированного режима.

Область BPool

Подавляющее большинство операций распределения памяти в программе SQL Server удовлетворяется за счет области BPool. Область BPool состоит из отдельных областей памяти, организованных в виде страниц с объемом 8 Кбайт, количество которых может достигать 32. В главе 4 была описана функция `VirtualAlloc` и показан способ, который может в ней использоваться для резервирования непрерывных блоков памяти. После того как сервер вычисляет максимальный размер области BPool, он резервирует область MemToLeave, пытаясь при этом обеспечить, чтобы она занимала непрерывный диапазон адресов. По возможности резервирование данной области осуществляется с помощью одного вызова функции `VirtualAlloc`. Если такая попытка оказывается неосуществимой (что весьма маловероятно), сервер выполняет несколько вызовов функции `VirtualAlloc` для резервирования области MemToLeave. Затем сервер предпринимает попытку зарезервировать память для области BPool из пространства адресов непривилегированного режима. (Пока не будем рассматривать, как учитывается при этом наличие памяти AWE, а вернемся к этому вопросу чуть позже.) Учитывая то, что в пространстве непривилегированного режима уже должны быть выполнены операции распределения памяти для библиотек DLL, отображаемых на память файлов и других объектов, весьма маловероятно, что

удастся зарезервировать весь объем памяти для области BPool с помощью одного вызова функции `VirtualAlloc`. Вместо этого с наибольшей вероятностью область BPool будет состоять из нескольких фрагментов, разбросанных в пространстве непривилегированного режима. В таком случае сервер снова и снова вызывает функцию `VirtualAlloc`, задавая каждый раз все меньший объем в запросе на резервирование до тех пор, пока вызов этой функции не завершится успешно. Такая операция повторяется вплоть до 32 раз для того, чтобы можно было резервировать для области BPool максимально возможный объем памяти. После завершения этой процедуры сервер освобождает область MemToLeave, чтобы она стала доступной внешним потребителям. Поскольку область MemToLeave резервируется до выполнения операций резервирования области Bpool, а после резервирования последней освобождается, то область MemToLeave с самого начала обычно представляет собой единственный, непрерывный блок памяти из свободного пространства виртуальных адресов.

Если же в операционной системе предусмотрены средства AWE, то применяется немного иной алгоритм. Повторные вызовы функции `VirtualAlloc` для распределения части пространства непривилегированного режима в целях создания области BPool все еще осуществляются. Но операционная система Windows не поддерживает резервирование и закрепление памяти AWE с использованием отдельных операций. Как было описано в главе 4, при применении средств AWE память либо распределяется, либо не распределяется. Функция `AllocateUserPhysicalPages` не поддерживает операции, в которых осуществлялось бы резервирование физической памяти, а распределение этой памяти не происходило, — память одновременно и резервируется, и закрепляется. Сразу после выполнения операции распределения память должна быть отображена на окно в пространстве непривилегированного режима, чтобы к ней можно было получить доступ с помощью 32-битового указателя.

Поэтому в указанной ситуации вся память, выделенная для области BPool, блокируется в физической памяти. Эта операция блокировки распространяется не только на ту часть области BPool, которая находится в памяти AWE, но и на ту часть, которая находится в пространстве адресов непривилегированного режима. Поскольку при этом блокируется физическая память, это может привести к серьезным проблемам производительности, касающимся других приложений, которые работают на том же компьютере, включая другие экземпляры программы SQL Server. Поэтому обычно при использовании средств AWE настройка конфигурации программы SQL Server осуществляется таким образом, чтобы эта программа была единственным важным приложением, работающим на компьютере. Если в программе SQL Server разрешена поддержка AWE, то учетная запись пользователя, от имени которой эксплуатируется сервер, должна иметь право резервировать страницы в памяти. Программа начальной установки SQL Server автоматически предоставляет это право пусковой учетной записи, выбранной для сервера. Если же запуск сервера осуществляется из командной строки или заменяется пусковая учетная запись, то необходимо предусмотреть предоставление этого права самостоятельно.

Хэширование

Для того чтобы поиск конкретных страниц осуществлялся быстрее, программа SQL Server хэширует страницы в области BPool. Для этого в основном выполняются такие действия: на основе данных о страницах области BPool создается такая хэш-таблица, с помощью которой сервер может быстро определить по данным об идентификаторе базы данных, номере файла и номере страницы данных, содержится ли в области BPool искомая страница данных, и где она находится (если эта страница действительно присутствует в области).

При возникновении в сервере необходимости получить доступ к конкретной странице данных, сервер хэширует идентификатор базы данных, номер файла и номер страницы таким образом, чтобы эта страница отобразилась на конкретный сегмент хэш-таблицы. Сегмент представляет собой связный список указателей на страницы BPool. Этот сегмент проверяется для определения того, имеется ли искомая страница в списке. Если страница в сегменте имеется, то к ней можно быстро получить доступ в памяти. Если же страница в сегменте отсутствует, то ее необходимо вначале загрузить с диска.

Примитивные операции распределения

Прежде чем появится возможность распределять какие-либо страницы из области BPool, в программе SQL Server необходимо распределить вспомогательные структуры, которые требуются для управления этой областью. Первая из структур, которая будет рассматриваться в данном разделе, представляет собой глобальную переменную, предназначенную для хранения ссылки на экземпляр класса, определяющего область BPool. Поскольку эта переменная имеет глобальную область определения, то к ней можно обратиться самому с помощью отладчика WinDbg и общедоступной отладочной информации, которая входит в поставку программы SQL Server. В упражнении 11.1 показано, во-первых, как найти эту глобальную переменную, и, во-вторых, как узнать, к какому типу данных она относится.

Упражнение

Упражнение 11.1. Использование отладчика WinDbg для поиска пула буферов

1. Подключитесь к экземпляру программы SQL Server, отличному от применяемого на производстве, с помощью отладчика WinDbg.
2. Убедитесь в том, что путь к файлам с отладочной информацией установлен правильно, как описано в главе 2.
3. Следующий шаг будет основан на двух указанных ниже предположениях.

3.1. Ссылка на область BPool имеет такой характер и применяется в сервере так широко, что должна, по-видимому, храниться в глобальной переменной или в аналогичной конструкции.

3.2. По всей вероятности, этой переменной присвоено имя `BPool`, `BufferPool` или какая-то другая разновидность подобного имени.

4. В приглашении к вводу команд отладчика WinDbg введите следующие команды:

```
.reload -f sqlservr.exe
x sqlservr!*
```

Эти команды позволяют получить список всех общедоступных символических имен, которые включены в относящийся к программе SQL Server файл базы данных программы (файл с отладочной информацией) `sqlservr.pdb`.

5. Выполните прокрутку к верхней части командного окна и щелкните курсором над началом вывода команды `x`. Нажмите клавиши <Ctrl+F>, введите `BPool` и нажмите <Enter>.

6. Вы должны найти первый экземпляр ссылки на класс `BPool`. Фактически этот экземпляр представляет собой ссылку на один из методов указанного класса. На основании этого можно сделать вывод, что в программе SQL Server имеется класс с именем `BPool`. Поэтому предположение о том, что таковым является тип данных объекта, в котором хранится пул буферов программы SQL Server, вполне оправдано, но это предположение будет подтверждено с достаточно высокой степенью определенности чуть позже.

7. Теперь отыщем глобальную переменную, в которой согласно предположению должна храниться ссылка на пул буферов. Выполните прокрутку к верхней части вывода и повторите поиск, на этот раз задав в качестве строки поиска "`bPool`" (без кавычек). В диалоговом окне обязательно укажите, что поиск должен осуществляться с учетом регистра. Поскольку в языке C++ учитывается регистр, разработчики часто называют экземпляр переменной по имени ее типа, но с использованием другого регистра. Начнем с проверки указанного предположения.

8. Если вы проводили поиск с учетом регистра, то не должны были обнаружить какие-либо отладочные символы с именем "`bPool`", поэтому теперь необходимо продолжить поиск. Разработчики часто используют также другой подход — применяют для обозначения переменной имя типа, но сокращают в них имена экземпляров, или наоборот. Поскольку уже известно, что область BPool имеет имя типа `BPool`, попытаемся найти переменную, обозначенную именем `BufferPool`. Повторите поиск, указав на этот раз в качестве критерия поиска `BufferPool`.

9. Вы должны найти символическое имя `sqlserver!BufferPool`. Обратите внимание на то, что перед этим именем не стоит префикс в виде имени класса и пары двоеточий (`::`), как в ссылке на класс `BPool`, обнаруженной ранее. Это означает, что найденное глобальное имя относится к какому-то другому типу. Исходя из самого этого имени, можно сделать вывод, что оно не относится к глобальной функции. Таким образом, это имя, вероятно, обозначает глобальную переменную, причем, скорее всего, ту, в которой хранится ссылка на пул буферов программы SQL Server.

10. В этот момент можно вывести содержимое переменной `BufferPool` на терминал с использованием команды `dd` или какой-то аналогичной команды. В настоящий момент само значение этой переменной не представляет для нас

особой ценности; автор просто хотел, чтобы читатель убедился, что это действительно глобальная переменная. Все функциональные средства области BPool программы SQL Server заключены в классе `BPool` и в единственном глобальном экземпляре этого класса — `BufferPool`.

11. Выполните прокрутку к верхней части командного окна и повторите поиск, на этот раз используя в качестве строки поиска "`DropCleanBuffers`". Вы должны найти в списке символических имен запись, которая относится к имени `BPool::DropCleanBuffers`.
12. Читатели предыдущих книг автора могут помнить приведенное в них обсуждение команды `DBCC DROPCLEANBUFFERS`. Эта команда была когда-то недокументированной, но она является удобным средством удаления чистых буферов из области BPool для того, чтобы можно было проверить какой-то запрос на холодном (пустом) кэше без перезагрузки программы SQL Server. Зная о том, что в классе `BPool` имеется метод с именем `DropCleanBuffers`, можно ли предположить, что именно он соответствует методу, который вызывается с помощью указанной команды `DBCC`? Установим на этой команде точку останова и выполним проверку.
13. В приглашении к вводу команд отладчика WinDbg введите следующие команды:
```
bp sqlserver!BPool::DropCleanBuffers
g
```
14. Теперь перейдите в программу Query Analyzer, подключитесь к серверу и выполните в окне редактора команду `DBCC DROPCLEANBUFFERS`. Снова перейдите в отладчик WinDbg. Вы должны обнаружить, что выполнение остановилось на указанной точке останова. Это позволяет со всей определенностью установить, что команда `DBCC DROPCLEANBUFFERS` реализована с помощью одного из методов класса `BPool`, который имеет имя `DropCleanBuffers`. Тем самым подтверждается также предположение о том, что класс `BPool`, обнаруженный в отладчике, действительно представляет собой тип данных глобального экземпляра пула буферов программы SQL Server.
15. Введите `q` в командном окне отладчика WinDbg и нажмите <Enter>, чтобы остановить отладку. Вам потребуется перезапустить программу SQL Server.

Массивы страниц

Как было указано выше, в программе SQL Server выполняется вплоть до 32 отдельных операций резервирования памяти для резервирования области BPool. В области BPool эти операции распределения отслеживаются в двух параллельных массивах — в одном массиве хранится список указателей на начало каждой области, а в другом — данные о количестве страниц с объемом 8 Кбайт, зарезервированных в этой области. Оба массива определены как закрытые переменные экземпляра BPool.

Массив BUF

В программе SQL Server для управления каждой страницей в области BPool используется специальная структура BUF. Прежде чем приступить к резервированию области BPool, программа SQL Server вызывает функцию `VirtualAlloc` для

распределения массива структур BUF из области MemToLeave, устанавливая размер этого массива с учетом количества страниц, которые должны быть зарезервированы для области BPool (включая физические страницы AWE). Каждая страница в области BPool имеет соответствующую ей структуру BUF, как и каждая страница памяти AWE, распределенная сервером, независимо от того, отображается ли эта страница на виртуальную память. Каждая структура BUF имеет размер 64 байта, поэтому указанный массив обычно не очень велик, если только в сервере не используется значительный объем памяти AWE.

Структура BUF каждой страницы действует как своего рода заголовок для этой страницы. В ней хранится такая информация, как указатель на фактическую страницу в области BPool, количество ссылок на эту страницу, защелка страницы и биты статуса, которые указывают, является ли эта страница незафиксированной, относятся ли к ней невыполненные операции ввода-вывода, закреплена ли она в памяти и т.д.

При просмотре средствами отложенной записи пула буферов для поиска страниц, подлежащих освобождению, фактически просматривается не пул, а этот массив структур BUF. Информация об упомянутой процедуре отложенной записи будет приведена ниже.

Структура отображения с информацией о закреплении

После запуска программы SQL Server из применяемой по умолчанию динамической области памяти процесса распределяется структура отображения, которая используется для отслеживания закрепленных страниц в области BPool. Сами операции резервирования, отслеживаемые с помощью массива страниц, представляют собой именно операции резервирования, а не закрепления. Как было описано в главе 4, существует возможность резервировать пространство адресов виртуальной памяти, не закрепляя за этим пространством какую-либо реальную память. Поэтому по мере того как происходит закрепление страниц в области BPool, устанавливаются соответствующие биты в структуре отображения закрепления.

Средства AWE

В области BPool предусмотрено отслеживание того, какие страницы памяти AWE используются в этой области, но сама область не может обеспечить непосредственный доступ к указанным страницам, поскольку такова организация средств AWE операционной системы Windows. Как было указано в главе 4, доступ к памяти AWE осуществляется по принципу отображения физических страниц на пространство адресов непривилегированного режима и последующей отмены этого отображения.

Обратите внимание на то, что, если на серверном компьютере имеется объем физической оперативной памяти меньше 3 Гбайт, притом что разрешено использование средств AWE с помощью опции awe enabled процедуры sp_configure, то программа SQL Server игнорирует предлагаемые ей средства AWE. Как было

описано выше в данной книге, для применения средств AWE операционной системы Windows в программе SQL Server необходимо иметь объем физической памяти 3 Гбайт или больше.

Средства отложенной записи

Средства отложенной записи имеют два назначения: во-первых, эти средства поддерживают в свободном виде заданное количество буферов BPool, чтобы их можно было распределить для использования сервером, и, во-вторых, средства отложенной записи осуществляют текущий контроль и корректируют степень использования закрепленной памяти областью BPool, для того чтобы в системе оставался свободным достаточно большой объем физической памяти, и операционной системе Windows не приходилось применять страничный обмен (при условии, что разрешено динамическое управление памятью, чтобы средства отложенной записи могли корректировать размер области BPool по мере необходимости). Программа SQL Server прогнозирует количество буферов BPool, которые должны поддерживаться в свободном виде, с учетом нагрузки системы и количества возникающих остановов в работе (количества тех случаев, когда потребителям памяти в сервере приходилось ждать появления свободной страницы в буфере).

Объем физической памяти, который процедура отложенной записи пытается поддерживать в свободном виде, обычно изменяется в пределах от 4 до 10 Мбайт. Отчасти это значение определяется с учетом расчетной продолжительности ожидаемого срока существования страницы для пула (количества секунд, в течение которых страница будет находиться в пуле буферов без обращения к ней). Для определения того, каким является это значения для конкретного экземпляра программы SQL Server, можно отслеживать значение счетчика `Buffer Manager:Page life expectancy` программы Perfmon. По мере увеличения ожидаемого срока существования страницы (т.е. по мере снижения нагрузки на память данного экземпляра программы SQL Server и поэтому увеличения продолжительности пребывания страниц в кэше даже при том, что к этим страницам никто не обращается), объем физической памяти, зарезервированной для операционной системы, приближается к 10 Мбайт. А по мере того как ожидаемый срок существования страницы уменьшается, объем физической памяти, зарезервированной для операционной системы, продолжает уменьшаться, пока не достигает приблизительно 4 Мбайт.

Поддержка минимального объема физической памяти, доступного для использования операционной системой, позволяет добиться того, чтобы в операционной системе не применялся ненужный страничный обмен, а сама ОС и другие процессы на компьютере с программой SQL Server функционировали бесперебойно. Если операционная система начинает испытывать недостаток физической памяти (даже если нехватка памяти не вызвана действиями программы SQL Server), область BPool корректирует закрепленный за ней объем памяти BPool в сторону уменьшения для того, чтобы увеличился объем доступной физической памяти.

Вычисление объема физической памяти

Способ, применяемый в области BPool для определения объема доступной физической памяти, зависит от операционной системы. В версии Windows 2000 вызывается функция `GlobalMemoryStatusEx` API-интерфейса Win32. Если программа SQL Server эксплуатируется в версии Windows 2000, то в этом можно убедиться самому, подключившись к программе SQL Server с помощью отладчика WinDbg и установив точку останова на вызове `kernel32!GlobalMemoryStatusEx`. Если сразу после активизации точки останова вы получите распечатку содержимого стека вызовов, то обнаружите, что функция `GlobalMemoryStatusEx` вызывается либо методом `BPool::AvailablePagingFile`, либо методом `BPool::AvailablePhysicalMemory`. Последний метод представляет собой вызов, используемый в области BPool для контроля над тем, соответствует объем физической памяти заданному пороговому значению или превышает это значение.

В версиях Windows NT 4.0 и Windows 9x/ME вызывается функция `GlobalMemoryStatus` API-интерфейса Win32. Эта функция используется вместо `GlobalMemoryStatusEx`, поскольку в версии Windows NT 4.0 или Windows 9x/ME функция `GlobalMemoryStatusEx` не поддерживается.

В версиях Windows XP и Windows Server 2003 в области BPool используются функции `CreateMemoryResourceNotification` и `QueryMemoryResourceNotification` API-интерфейса для передачи операционной системе Windows указания, чтобы ОС передавала в область BPool извещение в том случае, если объем физической памяти становится слишком малым. Указанные функции API-интерфейса не доступны в ранних версиях операционной системы Windows, поэтому они используются исключительно в версиях Windows XP и Windows Server 2003.

Сброс на диск и освобождение страниц

Как было указано выше, в массиве `BUF` содержится по одному элементу для каждой страницы в области BPool. Каждая структура `BUF` функционирует как своего рода заголовок страницы BPool и содержит данные о количестве ссылок на соответствующую этой структуре страницу. После формирования каждой новой ссылки на страницу указанное количество ссылок увеличивается. Использование некоторых более ценных страниц (например, таких, которые содержат план выполнения) начинается с установки более высокого значения количества ссылок по сравнению со страницами других типов. Это позволяет дольше держать указанные страницы в памяти и обеспечивает исключение излишних затрат ресурсов сервера на их воссоздание или повторную загрузку с диска без особой необходимости.

Указанный массив `BUF` периодически просматривается. Количество ссылок, записанное в каждой структуре `BUF`, делится на четыре, а остаток отбрасывается. После того как количество ссылок на страницу достигает нуля, страница проверяется для определения того, является ли она незафиксированной; в случае положительного ответа планируется операция записи с помощью компонента UMS для сброса незафиксированной страницы на диск. (Если количество ссылок на страницу достигает нуля, а сама страница не является незафиксированной, она

просто освобождается, т.е. перемещается в список свободных страниц, без записи каких-либо данных на диск.) Поскольку в программе SQL Server используется журнал опережающей записи, то сброс на диск незафиксированной страницы блокируется до тех пор, пока содержимое этой страницы не будет записано в журнал транзакций. После того как незафиксированная страница успешно записывается на диск, для нее отменяется хэширование (она удаляется из хэш-таблицы), и эта страница добавляется к списку свободных страниц. При таком развитии событий обычно не происходит отмена закрепления незафиксированной страницы; страница просто перемещается в список свободных страниц, для того чтобы ее можно было сразу же использовать повторно. Исключение ненужных операций отмены закрепления и повторного закрепления позволяет значительно повысить производительность операций с памятью программы SQL Server; указанный принцип проектирования является ключом к обеспечению эффективного сопровождения кэша памяти.

Размер списка свободных страниц вычисляется внутри программы SQL Server с учетом размера области BPool. Тот факт, что в программе SQL Server используются отдельные физические структуры для страниц и их заголовков, позволяет сбрасывать страницы на диск и якобы "перемещать" их из списка в список, не внося фактически каких-либо изменений в саму страницу.

Например, в то время как незафиксированная страница сбрасывается на диск и освобождается, изменяется содержимое заголовка страницы (структуры BUF этой страницы), но в саму страницу не требуется вносить изменения. После сброса страницы на диск ее содержимое считается утратившим важность и может быть перезаписано при повторном использовании страницы. Если же страницу необходимо переместить из одного списка в другой (например, при ее освобождении), то из списка в список перемещается 64-байтовая структура BUF, а не сама страница с размером 8 Кбайт. Причина, по которой в данном случае такую важную роль играет размер указанной структуры, связана с использованием локального кэша процессора. Чем меньше структура, тем больше ее экземпляров можно сохранить во встроенном кэше микросхемы процессора, и тем более эффективным становится процесс перемещения узлов из одного списка в другой. Эффективное использование локального кэша процессора становится еще одной важной составляющей повышения продуктивности кэша в памяти.

В семействе Windows NT все эти операции контроля над устареванием страниц, сброса страниц на диск и перемещения в список свободных страниц обычно выполняются отдельными рабочими (микро)потоками UMS, а выделенный поток отложенной записи, как правило, несет очень низкую нагрузку. С другой стороны, в версиях Windows 9x и ME поток отложенной записи играет гораздо более важную роль. Дело в том, что в версиях Windows 9x и ME не поддерживается асинхронный файловый ввод-вывод, поэтому компонент UMS вынужден выполнять все операции файлового ввода-вывода в синхронном режиме (см. главу 10). В результате способность какого-либо рабочего (микро)потока UMS брать на себя функции потока отложенной записи ограничивается; в связи с этим выделенный поток отложенной записи в версиях Windows 9x/ME является гораздо более активным, чем в версиях семейства Windows NT.

Поток отложенной записи проверяет количество свободных буферов и объем доступной физической памяти один раз в секунду или при получении сигнала. В этом можно убедиться самому, подключившись к программе SQL Server с помощью отладчика WinDbg и установив точку останова на вызове метода `BPool::AvailablePhysicalMemory`, который уже рассматривался в этой главе, как показано ниже. (Следует отметить, что из-за ограничений по ширине страницы книги этот код напечатан на двух строках, но его необходимо полностью ввести в виде одной строки.)

```
bp sqlservr!BPool::AvailablePhysicalMemory
".echo checking availphys; g"
```

После перезапуска отладчика WinDbg вы обнаружите, что сообщение, которое следует за командой `.echo`, будет отображаться один раз в секунду до тех пор, пока не произойдет останов отладчика.

Теперь необходимо подчеркнуть, что поток отложенной записи не освобождает страницы, сбрасывая их на диск, и не перемещает их в список свободных страниц, до тех пор, пока не достигнут верхний предел размера области BPool. Вместо этого поток отложенной записи просто закрепляет дополнительное количество зарезервированных страниц в области BPool каждый раз, когда количество страниц в списке свободных страниц становится ниже встроенного порогового значения, и в связи с этим изменяет значения соответствующих битов в структуре отображения закрепления.

Поток отложенной записи каждый раз проверяет по 16 структур `BUF`. Этот поток следит за тем, в каком месте была прекращена проверка в каждой итерации, и на следующем этапе своей работы (как уже было сказано, через секунду или после поступления сигнала) начинает с того места, где перед этим остановился. После того как поток отложенной записи достигает конца массива `BUF`, он возвращается в начало, по принципу работы с циклическим буфером, поэтому просмотр массива `BUF` продолжается непрерывно и бесконечно, по аналогии с тем, как стрелки часов обходят циферблат.

Контрольная точка

Проверка области BPool и сброс незафиксированных страниц на диск осуществляется также в процессе контрольной точки. Но в этом процессе не предусмотрено перемещение страниц в список свободных страниц. Задача по освобождению страниц возложена на рабочие (микро)потоки UMS и на выделенный поток отложенной записи. А в задачу процесса контрольной точки входит сокращение продолжительности времени, требуемой для восстановления системы за счет поддержки минимально возможного количества незафиксированных страниц. Но обычно процесс контрольной точки не обнаруживает слишком большое количество страниц, подлежащих сбросу на диск, поскольку незафиксированные страницы в основном уже бывают сброшены потоком отложенной записи или отдельными рабочими (микро)потоками UMS.

Секции

Для создания предпосылок улучшения масштабируемости в программе SQL Server применяется секционирование списка свободных страниц BPool с разбивкой по процессорам. Как было указано в главе 10, каждому отдельному пользователю UMS сразу после его подключения назначается конкретный планировщик, и пользователь остается связанным с этим планировщиком до своего отключения. С другой стороны, каждый планировщик UMS обычно связан с определенным процессором. Если у пользователя компонента UMS возникает необходимость получить свободную страницу, то вначале проверяется секция, относящаяся к тому процессору, с которым работает данный пользователь, а затем при необходимости – другие секции. Секционирование списка свободных страниц области BPool с разбивкой по процессорам позволяет лучше использовать локальный кэш каждого процессора и способствует повышению масштабируемости сервера.

Диспетчеры памяти

Все сложные средства управления памятью области BPool программы SQL Server не реализованы в самой области BPool, а распределены по пяти основным классам диспетчеров памяти. Это позволяет различным потребителям памяти распределять память и управлять ею независимо друг от друга, используя общий, управляемый пул памяти.

Но следует учитывать, что целый ряд операций распределения памяти выполняется без какого-либо участия этих диспетчеров. Например, средство ведения журнала транзакций реализует собственный механизм кэширования так же, как и средство резервного копирования (восстановления). В системе имеются также другие аналогичные примеры. Выполняемые при этом операции восстановления происходят без участия какого-либо диспетчера памяти, а осуществляются непосредственно с помощью операционной системы на основе вызовов функции `VirtualAlloc`.

К числу указанных пяти диспетчеров относятся диспетчеры памяти Connection, Query Plan, Optimizer, Utility и General. Каждый из этих диспетчеров отвечает за выполнение в сервере отдельного класса операций управления памятью. Ниже каждый диспетчер рассматривается отдельно.

Диспетчер памяти Connection

Этот диспетчер памяти отвечает за выполнение операций распределения памяти, относящихся к соединениям. Для каждого соединения распределяется структура PSS (process status structure – структура статуса процесса) и структура `SRV_PROC`, а также один буфер сетевой передачи и два буфера сетевого приема. Диспетчер памяти Connection управляет операциями распределения указанного типа.

Диспетчер памяти Query Plan

В программе SQL Server диспетчер памяти Query Plan используется в основном для выполнения операций распределения, связанных с откомпилированными

планами и планами выполнения, которые вырабатываются оптимизатором запроса (эти планы можно найти в системной таблице syscacheobjects). Кроме того, указанный диспетчер памяти осуществляет операции распределения, относящиеся к курсорам, параметрам RPC, создаваемым индексам и к некоторым командам DBCC, которые применяются с индексами на вычисленных столбцах.

Следует отметить, что Query Plan является единственным диспетчером памяти, освобождающим память по "указанию" потока отложенной записи. Страница, в заголовке BUF которой имеется бит статуса, указывающий, что эта страница относится к плану запроса (query plan), может быть удалена потоком отложенной записи из кэша в связи с устареванием, после того, как количество ссылок на эту страницу достигнет нуля.

Диспетчер памяти Optimizer

В программе SQL Server диспетчер памяти Optimizer используется для распределения и управления метаданными и древовидными структурами, относящимися к оптимизации запросов. Сервер устанавливает для этого диспетчера лимит, равный 80% общего объема памяти сервера.

Диспетчер памяти Utility

В программе SQL Server имеется специальный диспетчер памяти, предназначенный для использования вспомогательными функциями. Как указывает само имя диспетчера памяти Utility, этот диспетчер применяется для выполнения различных операций распределения, относящихся к утилитам, в том числе для осуществления операций распределения буферов, которые используются серверным средством трассировки, а также функциями, относящимися к инициализации диспетчера журнала, пересылке кластера и журнала, сравнению структур отображения, поиску в хэш-таблицах и т.д.

Диспетчер памяти General

Этот диспетчер памяти предназначен для выполнения операций распределения, которые не попадают ни в одну из категорий, упомянутых выше. С помощью диспетчера памяти General выполняются операции распределения для нескольких различных типов структур данных внутренней машины хранения, включая блокировки.

Операции распределения памяти с помощью диспетчера памяти OS

Как правило, пять указанных выше основных диспетчеров памяти предпринимают попытки распределять память в основном из области BPool, при условии, что объем памяти, указанный в запросе на распределение, не превышает 8 Кбайт.

С другой стороны, запросы на выполнение операций распределения, превышающих 8 Кбайт непрерывной памяти, обычно передаются диспетчеру памяти, который отвечает за распределение памяти из области MemToLeave. Этот диспетчер памяти обычно упоминается под именем диспетчера памяти OS или Reserved. Он не реализован в виде отдельного класса диспетчера (т.е. не реализован по такому же принципу, как пять основных описанных выше диспетчеров памяти); вместо этого рассматриваемый диспетчер предоставляет функциональные возможности, которые могут использоваться всеми пятью другими диспетчерами для выполнения операций распределения больших объемов памяти.

При обработке запроса на распределение памяти диспетчер памяти OS вызывает функцию `VirtualAlloc` для резервирования достаточно большой области из пула MemToLeave. Этот запрос на резервирование округляется в большую сторону до величины степени детализации распределения системы (64 Кбайт в 32-битовых версиях Windows), как было описано в главе 4. В ходе выполнения соответствующей операции резервирования указанный диспетчер памяти закрепляет такое количество страниц, которое является достаточным для удовлетворения запроса. Если до освобождения указанной области будут получены дополнительные запросы на распределение памяти, то они также могут быть выполнены путем закрепления страниц из этой области. А после отмены закрепления всех страниц рассматриваемый диспетчер памяти вызывает функцию `VirtualFree` для освобождения данной области.

Диспетчер памяти, применяемый в условиях нехватки памяти

В программе SQL Server предусмотрен специальный диспетчер памяти для работы в аварийных условиях, который используется в тех редких обстоятельствах, когда работа обычных диспетчеров памяти оканчивается неудачей, а сервер должен получить еще немного памяти, чтобы продолжить свое функционирование без серьезных последствий (например, без искажения данных) во время выполнения таких действий, как ведение журнала или восстановление. Обычные диспетчеры не прибегают автоматически к использованию этого диспетчера памяти после того, как оканчиваются неудачей предпринятые ими попытки выполнения операций распределения; указанный диспетчер памяти применяется только в некоторых весьма критических путях выполнения кода в сервере.

После запуска программы SQL Server диспетчер памяти, предназначенный для работы в условиях нехватки памяти, закрепляет область виртуальной памяти с объемом 64 Кбайт. Эту память может одновременно применять только один потребитель. Каждый новый потребитель должен ожидать, пока другие потребители не освободят указанную память, и только после этого приступать к ее использованию.

При обращении к указанному диспетчеру памяти происходит запись следующего предупреждающего сообщения в журнал ошибок программы SQL Server:

```
Warning: Due to low virtual memory, special reserved memory used %d
% times since startup. Increase virtual memory on server.
```

В связи с самим характером ситуации, в которой происходит появление этих сообщений, обычно обнаруживаются другие ошибки или предупреждения, сопровождающие такие сообщения.

Интерфейс `IMalloc`

В программе SQL Server для выполнения отдельных операций распределения памяти используется собственная реализация интерфейса `IMalloc` технологии COM. Как было указано выше в данной главе, если какому-то потребителю памяти требуется выдать запрос к памяти, то данный потребитель вначале создает объект распределения памяти, а затем использует этот объект для выдачи запроса. С помощью одного объекта распределения можно выдать целый ряд запросов. Указанный объект распределения реализует стандартный интерфейс `IMalloc` технологии COM.

Интерфейс `IMalloc` включает все средства, которые можно надеяться обнаружить в стандартной программе распределения памяти: функции распределения, отмены распределения, изменения размеров, определения величины блока распределения и т.д. В технологии COM предусмотрена стандартная реализация указанного интерфейса, в которой используются средства динамической области памяти Win32 для выполнения запросов на распределение памяти. Но в программе SQL Server предусмотрены собственные функции управления памятью, поэтому в этой программе используется не стандартная реализация COM, а внутренняя реализация, позволяющая выполнять запросы на распределение с применением памяти из области BPool или области MemToLeave, в зависимости от характера запроса.

Описание общей картины

До сих пор достаточно подробно рассматривались отдельные компоненты архитектуры управления памятью программы SQL Server. А в этом разделе все указанные компоненты описаны вместе, чтобы можно было лучше понять, как в целом происходит управление памятью в программе SQL Server и почему эта программа в определенных обстоятельствах действует так, а не иначе.

После запуска программы SQL Server вычисляется верхний предел объема области BPool с учетом объема физической памяти, установленной на компьютере, значения параметра `max server memory` (максимальный объем памяти сервера) процедуры `sp_configure` и размера области MemToLeave. После вычисления указанного верхнего предела распределяется (резервируется) область MemToLeave для того, чтобы эта область не была фрагментирована в результате выполняемых в дальнейшем операций резервирования области BPool. Затем распределяется область BPool с использованием не больше чем 32 отдельных операций резервирования, которые могут потребоваться с учетом того, что ко времени резервирования области BPool процессом SQL Server уже может быть распределена память для библиотек DLL, а также выполнены другие операции распределения, в результате которых пространство виртуальных адресов процесса становится фрагментированным.

После резервирования области BPool область MemToLeave освобождается. Программа SQL Server не оставляет за собой область MemToLeave, поскольку эта область предназначена для "внешнего" потребления (т.е. для использования в компонентах, не относящихся к основному коду программы SQL Server). Область MemToLeave используется для внутренних операций распределения SQL Server, которые превышают по объему 8 Кбайт непрерывного пространства, а также для операций распределения, выполняемых такими внешними потребителями, как средства доступа OLE DB, внутрипроцессные COM-объекты и т.п. Как уже было сказано, программа SQL Server резервирует всю область MemToLeave при запуске, а затем освобождает эту область после резервирования области BPool, чтобы предотвратить фрагментацию области MemToLeave в результате выполнения операций резервирования области BPool.

Итак, после запуска сервера область BPool становится зарезервированной, но не закрепленной, а область MemToLeave, по существу, представляет собой свободное пространство в пространстве адресов виртуальной памяти процесса сервера. При просмотре содержимого счетчика `Virtual Bytes` программы Perfmon, относящегося к процессу SQL Server, непосредственно после запуска программы SQL Server можно обнаружить, что в этом счетчике отражены результаты резервирования области BPool. Автор сталкивался с тем, как пользователи выражали свою озабоченность, поскольку значение этого счетчика часто бывает весьма велико, но в конечном итоге в этом значении обычно отражается либо общий объем физической памяти на компьютере, либо максимальный объем пространства адресов непривилегированного режима за вычетом объема области MemToLeave. Тем не менее не следует об этом беспокоиться, поскольку область BPool представляет собой лишь зарезервированное, а не закрепленное пространство. Как было указано в главе 4, зарезервированное пространство — это просто пространство адресов; с указанным пространством не связана какая-либо реальная память, до тех пор, пока не произойдет его закрепление.

Со временем объем памяти, закрепленный за областью BPool, возрастает, пока не достигает верхнего предела, вычисленного сразу после запуска сервера. За изменением указанного объема можно следить с помощью счетчика `SQL Server:Buffer Manager\Target Pages` программы Perfmon. По мере того как в различных компонентах сервера возникает потребность в памяти, в области BPool происходит закрепление страниц с объемом 8 Кбайт, которые были первоначально зарезервированы, пока указанный закрепленный объем не достигает вычисленного предельного значения. При этом расходуется виртуальное пространство, а не физическое, и основной объем виртуальной памяти отображается на системный файл подкачки, а не на физическую оперативную память, поэтому такое увеличение потребления памяти не обязательно приводит к увеличению используемого объема физической памяти. Для контроля над тем, как используется закрепленная виртуальная память в области BPool, можно использовать счетчик `SQL Server:Buffer Manager\Total Pages` программы Perfmon. А за изменением общего объема применяемой закрепленной виртуальной памяти в сервере можно следить с помощью счетчика `Private Bytes`, относящегося к процессу SQL Server.

Основное потребление виртуальной памяти в программе SQL Server происходит за счет области BPool, поэтому значения двух указанных выше счетчиков, вообще говоря, увеличиваются или уменьшаются одновременно. Если же значение счетчика `Total Pages` уменьшается, а значение счетчика `Private Bytes` продолжает увеличиваться, это обычно указывает на продолжающееся выполнение операций распределения за счет области MemToLeave. Такие операции распределения могут полностью укладываться в рамки нормального функционирования (например, таковыми могут быть операции распределения, относящиеся к выделению памяти для стеков потоков в связи с созданием в сервере дополнительных рабочих потоков), но эти операции распределения могут также указывать на то, что внешние потребители, такие как внутрипроцессные COM-объекты или расширенные процедуры, допускают утечку памяти. Если в процессе сервера исчерпывается пространство адресов виртуальной памяти в связи с полным расходованием объема памяти области MemToLeave из-за утечек или чрезмерного потребления памяти (или если максимальный объем свободных страниц в области MemToLeave падает ниже заданного по умолчанию размера стека потока, равного 0,5 Мбайт), сервер утрачивает возможность создавать новые рабочие потоки, даже если еще не достигнуто значение параметра `max worker threads` (максимальное количество рабочих потоков) процедуры `sp_configure`. Если в этой ситуации в сервере возникнет необходимость создать новый рабочий поток для обработки запроса на выполнение работы (например, для обработки нового запроса на установление соединения), обработка этого запроса на выполнение работы будет отложена до тех пор, пока сервер не сможет создать поток, или не станет доступным другой рабочий (микро)поток, который можно было бы применить для указанной цели. В таких условиях может оказаться невозможным подключение пользователя к серверу, поскольку попытки подключения будут завершаться неудачей из-за истечения тайм-аута еще до того, как освободится достаточный объем пространства MemToLeave или станет доступным другой рабочий (микро)поток, способный обработать запрос на установление соединения.

Любой потребитель памяти в сервере инициализирует операцию распределения памяти путем предварительного создания объекта распределения памяти, предназначенного для управления запросом на распределение памяти. Такой объект распределения памяти представляет собой реализацию стандартного интерфейса `IMalloc` технологии COM. После того как указанный объект распределяет память для запроса, он вызывает соответствующий диспетчер памяти, относящийся к серверу, для того, чтобы этот запрос был выполнен либо за счет области BPool, либо за счет области MemToLeave. Если объем памяти, указанный в запросе, составляет 8 Кбайт или меньше, то запрос обычно выполняется с использованием памяти из области BPool, а запросы на распределение памяти с объемом, превышающим 8 Кбайт непрерывного пространства, как правило, выполняются с применением памяти из области MemToLeave. Для осуществления нескольких операций распределения может использоваться единственный объект распределения памяти, поэтому в действительности возможно, что какая-то операция распределения объема меньше 8 Кбайт будет выполнена за счет области MemToLeave, как было указано выше.

Потребителями памяти, относящейся к пространству процесса SQL Server, обычно являются внутренние потребители, т.е. потребители или объекты из самого кода программы SQL Server, которым требуется память для выполнения какой-либо задачи, но это условие не является обязательным. Как уже было сказано, таковыми могут быть также внешние потребители. К внешним потребителям относятся средства доступа OLE DB, расширенные процедуры, внутрипроцессные COM-объекты и т.д. Как правило, в этих внешних потребителях для распределения и управления памятью используются обычные функции работы с памятью API-интерфейса Win32, и поэтому память для внешних потребителей распределяется из пространства MemToLeave, поскольку для указанных потребителей область MemToLeave представляется как единственная доступная область в пространстве памяти процесса SQL Server. Но расширенные процедуры представляют собой исключение из этого правила. Если какая-либо расширенная процедура вызывает функцию `srv_alloc` API-интерфейса ODS, то подобная процедура рассматривается наряду с любыми другими потребителями в сервере. Вообще говоря, запросы `srv_alloc` на распределение памяти с объемом 8 Кбайт или меньше выполняются за счет области BPool, а операции распределения большего объема выполняются за счет пространства MemToLeave.

В ходе функционирования сервера поток отложенной записи периодически проводит проверку для обеспечения того, чтобы на серверном компьютере оставался доступным обусловленный объем физической памяти, поскольку лишь при таком условии приложения Windows и другие приложения на сервере смогут продолжать функционировать бесперебойно. Регламентированный объем доступной физической памяти может находиться в пределах от 4 до 10 Мбайт (в версии Windows Server 2003 он обычно приближается к 10 Мбайт), причем зависит от загрузки системы и от ожидаемой продолжительности существования страницы в области BPool. Если объем доступной физической памяти на серверном компьютере начинает опускаться ниже указанного порогового значения, то сервер отменяет закрепление страниц области BPool для того, чтобы уменьшить занимаемый этой областью объем реальной памяти (при условии, что разрешено использование средств динамического определения конфигурации памяти).

Применение средств отложенной записи позволяет также гарантировать, что в каждый конкретный момент времени будет оставаться свободным обусловленное количество страниц для того, чтобы при поступлении новых запросов на распределение памяти не приходилось ожидать освобождения памяти перед выполнением затребованных операций распределения. Под термином "свободная страница" подразумевается, что страница закреплена, но не используется. Неиспользуемые закрепленные страницы BPool отслеживаются с помощью списка свободных страниц. По мере того как становятся используемыми все новые и новые страницы из списка свободных страниц, поток отложенной записи закрепляет дополнительные страницы из памяти, зарезервированной в области BPool, до тех пор, пока вся зарезервированная память не становится закрепленной. В связи с выполнением указанных действий можно наблюдать, как постепенно (и обычно линейно) происходит увеличение значения счетчика `Process:Private Bytes` программы Perfmon.

С каждым процессором в системе связан отдельный список свободных страниц. Если возникает необходимость в получении свободной страницы для удовлетворения какого-либо запроса на распределение памяти, то вначале проверяется список свободных страниц, который относится к тому рабочему (микро)потоку UMS, который затребовал это распределение, и лишь после этого проверяются списки, относящиеся к другим процессорам в системе. Такой подход применяется в целях повышения масштабируемости за счет лучшего использования локального кэша каждого процессора в многопроцессорной системе. За конкретной секцией BPool можно следить с помощью объекта `SQL Server:Buffer Partition` программы Perfmon, а для контроля над списком свободных страниц, относящимся ко всем секциям, может применяться счетчик `SQL Server:Buffer Manager\Free Pages` программы Perfmon.

Итак, на протяжении всего времени функционирования программы SQL Server средства отложенной записи (выполняющие свои функции либо с помощью потока отложенной записи, либо с помощью одного из рабочих (микро)потоков UMS) контролируют состояние памяти системы для обеспечения того, чтобы для остальной части системы поддерживался достаточно большой объем физической памяти, а для использования в новых запросах распределения памяти всегда оставались доступными свободные страницы в нужном количестве.

В случае необходимости в указанные подходы вносятся некоторые изменения, если на сервере используется память AWE. Средства AWE операционной системы Windows не поддерживают принцип резервирования памяти без ее закрепления (т.е. не поддерживают операции динамического закрепления и отмены закрепления памяти), поэтому при использовании памяти AWE программа SQL Server не поддерживает средства динамического управления памятью. Создание области BPool начинается с получения и блокировки физической памяти на компьютере. Объем памяти, заблокированной для этой области, изменяется в зависимости от того, было ли задано значение параметра `max server memory` (максимальный объем памяти сервера). Если значение параметра `max server memory` задано, то в области BPool предпринимается попытка заблокировать объем, определяемый этим параметром. Если же значение данного параметра не задано, то для области BPool блокируется вся физическая память на компьютере, за исключением объема, приблизительно равного 128 Мбайт, который остается доступным для других процессов. Затем в области BPool физическая память, выходящая за пределы 3 Гбайт (память AWE), используется как своего рода файл подкачки для страниц данных и страниц индексов. В этой области физические страницы, относящиеся к указанному участку памяти, отображаются на пространство адресов виртуальной памяти по мере необходимости для того, чтобы к ним можно было обращаться с помощью 32-битовых указателей.

Резюме

В настоящей главе были приведены основные сведения об управлении памятью в программе SQL Server. Понимание того, как в любом приложении происходит распределение и управление памятью, существенно важно для понимания работы

самого приложения. Память — это крайне важный ресурс, а эффективное использование памяти представляет собой неотъемлемую составляющую качественного проекта приложения. Это означает, что знание особенностей управления памятью в приложении позволяет глубоко изучить общий проект этого приложения.

Вопросы для самопроверки

1. Область BPool — это пул, который используется в большинстве операций распределения памяти, выполняемых в программе SQL Server. Какая область предназначена для внешних потребителей наподобие средства доступа OLE DB, которые функционируют в сервере в качестве внутрипроцессного объекта?
2. Какой из пяти диспетчеров памяти SQL Server управляет памятью, относящейся к соединениям?
3. Насколько часто средство отложенной записи вызывается на выполнение?
4. Какой счетчик программы Perfmon показывает суммарный объем виртуальной памяти, закрепленной в некотором процессе?
5. Опишите в общих терминах две основные задачи, выполняемые с помощью средства отложенной записи.
6. Подтвердите или опровергните следующее утверждение. В версии SQL Server 2000 и во всех последующих версиях операции распределения памяти с объемом 8 Кбайт или меньше всегда выполняются за счет области BPool.
7. Подтвердите или опровергните следующее утверждение. Операции распределения, выполняемые расширенной процедурой с помощью функции `srv_alloc` API-интерфейса ODS, осуществляются за счет области MemToLeave, независимо от объема памяти, которая должна быть распределена.
8. Какой стандартный COM-интерфейс используется в программе SQL Server для управления отдельными операциями распределения памяти?
9. Средство отложенной записи способно вынудить освободить распределенные страницы один-единственный диспетчер из пяти. Назовите этот диспетчер памяти.
10. Каков максимальный размер, до которого может вырасти область BPool в системе с 512 Мбайт физической памяти и с параметром запуска SQL Server `-g768`?
11. Какая функция API-интерфейса Win32 используется средством отложенной записи в версии Windows 2000 для проверки объема доступной физической памяти на компьютере?
12. Подтвердите или опровергните следующее утверждение. Рабочим (микро)потокам UMS во многих случаях разрешено выполнять работу вместо средств отложенной записи, поэтому выделенное средство отложенной

записи при эксплуатации в экземпляре SQL Server, установленном в семействе Windows NT операционной системы Windows, часто обнаруживает, что для него осталось очень мало работы.

13. Почему пространство адресов виртуальной памяти области MemToLeave резервируется при запуске системы?
14. Сколько секций списка свободных страниц области BPool создается в программе SQL Server на четырехпроцессорном компьютере?
15. Какой параметр командной строки можно передать в программу SQL Server, чтобы откорректировать размер области MemToLeave?
16. Какое право доступа операционной системы должна иметь пусковая учетная запись программы SQL Server для того, чтобы в этой программе можно было использовать память AWE?
17. Подтвердите или опровергните следующее утверждение. Если настройка конфигурации программы SQL Server выполнена так, чтобы в ней использовалась память AWE, а объем физической памяти в системе составляет меньше 3 Гбайт, сервер отказывается запускаться и регистрирует ошибку в системном журнале регистрации ошибок.
18. Какой механизм используется в программе SQL Server для отслеживания количества закрепленных страниц в области BPool?
19. Какой механизм используется в программе SQL Server поиска страницы данных в области BPool с использованием таких данных о странице, как идентификатор базы данных, номер файла и номер страницы?
20. Какое общее количество операций резервирования может быть выполнено при запуске системы для резервирования области BPool?
21. До какого максимального размера может вырасти область BPool, если программа SQL Server эксплуатируется с параметром запуска `-g768` и значением параметра `max server memory`, равным 384, на компьютере с объемом физической памяти 2 Гбайт?
22. Какая часть пула MemToLeave отведена по умолчанию для стеков потоков?
23. Подтвердите или опровергните следующее утверждение. Операции распределения памяти с объемом 8 Кбайт или меньше для внутрипроцессных COM-объектов выполняются за счет области BPool, поэтому после уничтожения объекта распределенные страницы автоматически освобождаются.
24. Подтвердите или опровергните следующее утверждение. Если в программе SQL Server используется память AWE, то физическая память, на которую отображается область BPool, блокируется и становится недоступной для других процессов.
25. Подтвердите или опровергните следующее утверждение. По умолчанию средства отложенной записи предпринимают попытки сохранить в области памяти процесса SQL Server не меньше 384 Мбайт памяти доступными для области MemToLeave.

ГЛАВА 12

Процессор запросов

Задачи обработки запросов и повышения производительности являются настолько важными для оптимальной эксплуатации программы SQL Server, что автор включает главу, посвященную каждой из этих тем, во все свои книги, посвященные программе SQL Server. В данной книге приведено обновленное изложение указанного материала из последней книги автора, *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML*, и продолжено описание основных принципов оптимизации запросов, начатое в указанной книге. Безусловно, предоставление читателю целого ряда отдельных рекомендаций по повышению производительности позволило бы получить определенный кратковременный выигрыш, но, по мнению автора, понимание принципов, на которых основаны подобные рекомендации, намного важнее и в долговременном плане позволит читателю добиться гораздо более существенных успехов. Как было указано во "Введении", понимание проекта, лежащего в основе определенной технологии, гораздо важнее, чем простое изучение способов применения этой технологии. Способность успешно настраивать запросы на языке Transact-SQL является производной от знания и понимания самой программы SQL Server – как работает эта программа, какие действия выполняет при обработке запроса, какие ресурсы требует для формулировки эффективного плана и т.д. Поэтому в настоящей главе изложение практических сведений, необходимых для пользователя, сопровождается обсуждением некоторых малозаметных особенностей обработки запросов в программе SQL Server. В этой главе описаны различные этапы обработки запросов и указано, что происходит на каждом из них. Получив более полное представление о том, как действуют средства обработки запросов программы SQL Server, читатель сможет разрабатывать собственные приемы ускоренного выполнения кода T-SQL и применять при этом обоснованные и безопасные методы благодаря лучшему пониманию того, на каком проекте основано функционирование программы SQL Server.

Основные термины и определения

- **Предикат.** Выражение, которое принимает значение `true` (истина), `false` (ложь) или `unknown` (неизвестное значение).
- **Кардинальность.** Строго говоря, кардинальностью называется количество уникальных значений в таблице. Программа SQL Server допускает наличие в таблицах дублирующихся строк, а в индексах – дублирующихся значений ключа, поэтому при обсуждении программного продукта SQL Server кардинальность часто определяют в более общих терминах – как количество строк в таблице или количество строк, возвращенных оператором плана

запроса. Понятие кардинальности можно также определить как количество уникальных значений в индексе.

- **Плотность.** Плотность определяют как уникальность значений в наборе данных. Плотность индекса вычисляется путем деления количества строк, которые соответствуют данному конкретному значению ключа, на количество строк в таблице.
- **Избирательность.** Мера оценки количества строк, которые должны быть возвращены с помощью данного конкретного предиката. Этот показатель выражается в виде процентного соотношения, определяемого количеством строк в таблице.

Синтаксический анализ

Для того чтобы запрос на языке Transact-SQL можно было оптимизировать с помощью оптимизатора запросов SQL Server, запрос необходимо прежде всего представить в виде дерева реляционных операторов с помощью синтаксического анализа. Дерево реляционных операторов состоит из логических операторов, необходимых для выполнения работы, затребованной в запросе. Например, простой запрос, в котором осуществляется соединение таблиц `Customers` и `Orders` базы данных `Northwind`, может быть преобразован с помощью синтаксического анализа в дерево реляционных операторов, которое включает логический оператор `Inner Join` (внутреннее соединение), находящийся между двумя входными таблицами.

В программе SQL Server предпринимаются попытки избежать излишних действий по синтаксическому анализу и оптимизации запросов путем хэширования текста каждого вновь поступившего запроса и кэширования его в памяти, наряду с первоначальным текстом запроса и ссылкой на контекст выполнения и план выполнения, которые были получены после обработки первоначального запроса. Сервер сверяет хэшированный текст запроса каждого вновь поступившего запроса с тем текстом, который уже находится в памяти. Поскольку существует вероятность коллизий в хэше, сервер, обнаружив совпадение, сравнивает первоначальный текст нового запроса с первоначальным текстом запроса, находящегося в памяти, для определения того, имеется ли между ними точное совпадение. Если эти тексты действительно совпадают, то новый запрос не требует повторного синтаксического анализа и повторной оптимизации; для него можно использовать ранее созданный контекст выполнения и/или план выполнения.

Средства программы SQL Server, которые обеспечивают синтаксический анализ поступивших запросов с помощью языковых событий и вызовов RPC с последующим преобразованием в деревья реляционных операторов, называются средствами языковой обработки и выполнения (Language Processing and Execution – LPE). Средства LPE обеспечивают передачу в оптимизатор дерева реляционных операторов, сформированного в результате синтаксического анализа, а затем получение оптимизированного плана выполнения, выработанного оптимизатором на этапе преобразования, и выполнение запроса. Средства LPE не представлены в сервере в виде одного компонента или фрагмента кода, а фактически состоят из многочисленных модулей и привлекают к своей работе целый ряд других средств сервера.

Этапы оптимизации

Процесс оптимизации запроса представлен в общих чертах на рис. 12.1. После получения дерева реляционных операторов, в которое был преобразован первоначальный запрос в результате синтаксического анализа, оптимизатор приступает к его оптимизации — поиску способов выполнения требуемой работы с помощью меньшего объема издержек и модификации запроса в случае необходимости для достижения этой цели. Вообще говоря, в каждой последующей фазе или на каждом этапе осуществления процедуры оптимизации предусмотрена возможность рассматривать все более сложные (и часто более дорогостоящие) варианты преобразования исходного дерева реляционных операторов, сформированного синтаксическим анализатором, в оптимизированный план выполнения. Если в ходе выполнения этой процедуры удается найти подходящий эффективный план, этот план может быть возвращен без необходимости дальнейшей оптимизации или оценки потенциальных преобразований. Процедура оптимизации состоит из четырех отдельных этапов: тривиальная оптимизация плана, упрощение, полная оптимизация и преобразование. Ниже каждый из этих этапов рассматривается отдельно.

Тривиальная оптимизация плана

Применительно к некоторым запросам наиболее эффективный план выполнения можно выявить на основе самого запроса, поэтому какая-либо оценка издержек или проведение сравнений между планами не требуется. Например, если в запросе используется предикат равенства для выборки данных с помощью запроса из столбца с уникальным индексом, то нет необходимости оценивать стоимости различных планов и проводить сравнение этих стоимостей; очевидно, что наилучший вариант состоит в использовании поиска по уникальному индексу. Кроме того, в оптимизаторе предусмотрены тривиальные варианты оптимизации для некоторых типов запросов, отвечающих определенным критериям; соединений таблиц с условиями ограничения; операторов DML и некоторых других операций. (Но следует отметить, что тривиальный план не может применяться для обработки запроса, содержащего подзапрос.) Если оптимизатор обнаруживает, что наименее дорогостоящим планом, который можно надеяться получить, является тривиальный план, то оптимизатор передает этот план на этап преобразования для подготовки к выполнению. Для контроля над тем, был ли выбран тривиальный план, можно разрешить флажок трассировки 8759. Если этот флажок разрешен, то оптимизатор записывает первую часть запроса, для которого создан тривиальный план, в журнал регистрации ошибок.

Ниже показан пример запроса, который приводит к созданию тривиального плана.

```
DECLARE @ordno int
SET @ordno=10248
SELECT * FROM Orders WHERE OrderId=@ordno
```

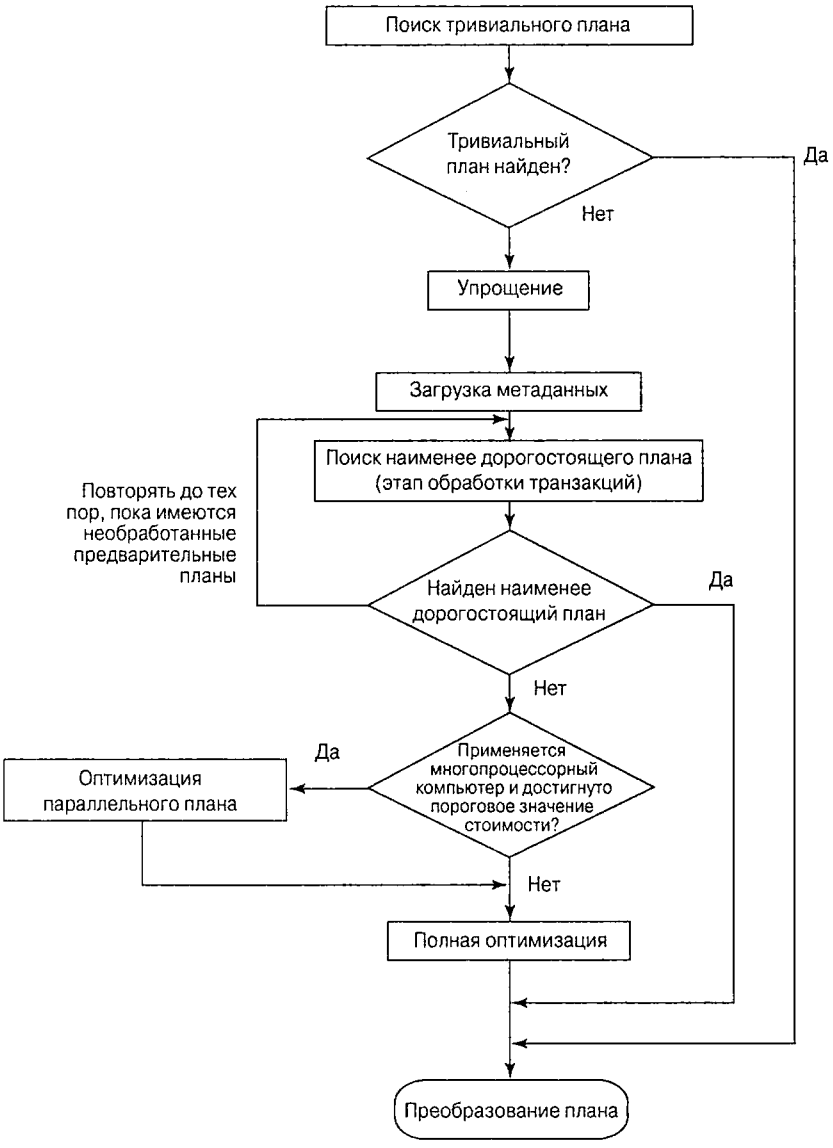


Рис. 12.1. Процедура оптимизации запроса

В этом запросе единственным используемым предикатом является предикат сравнения на равенство, а на столбце `Orders.OrderNo` имеется уникальный индекс, поэтому оптимизатор выбирает тривиальный план, в котором для обслуживания этого запроса предусмотрен поиск по индексу. Заслуживает внимания тот факт, что использование в данном запросе переменной в качестве условия запроса не исключает возможности для оптимизатора выбрать тривиальный план. Независимо от того, какое значение принимает эта переменная на этапе прогона, для сравнения

значения переменной с ключом столбца, на котором определен уникальный индекс, применяется простой оператор сравнения на равенство, поэтому оптимизатор может сразу же определить, что наилучшим вариантом является тривиальный план.

Этапы тривиальной оптимизации и полной оптимизации представляют собой единственные два этапа оптимизации, на которых может быть сформирован план, после чего процедура оптимизации может завершиться. Если какой-то запрос не удается оптимизировать на этапе тривиальной оптимизации, то этот запрос передается на этап упрощения, а затем на этап полной оптимизации.

Упрощение

На этапе упрощения применяются некоторые эвристические алгоритмы, которые позволяют использовать в дальнейшем более простые и эффективные методы оптимизации. Как указывает само название этого этапа, он предназначен для упрощения выражения запроса и обеспечения лучшего применения индекса и статистических данных во время полной оптимизации. При этом операторы могут перемещаться в дереве, заменяться другими операторами и в целом упрощаться, чтобы осуществление полной оптимизации проходило более просто, быстро и эффективно. Например, подзапросы, используемые в предикате, могут быть заменены полусоединениями и внутренними соединениями; критерии выборки, которые применяются только к одной таблице в многотабличном запросе, могут быть передвинуты в дереве таким образом, чтобы они применялись перед выполнением операций соединения; могут быть распознаны соотношения, которые имеют вид звездообразной схемы; некоторые операторы могут быть заменены более эффективными эквивалентами и т.д. Ниже приведен пример запроса, содержащего подзапрос, который преобразуется оптимизатором в полусоединение на этапе упрощения.

```
SELECT a.au_id
FROM authors a
WHERE au_id IN (SELECT au_id FROM titleauthor ta WHERE
   ta.au_id=a.au_id)
```

При просмотре графического изображения плана этого запроса в программе Query Analyzer можно обнаружить, что оптимизатор вырабатывает план, в котором используется оператор `Nested Loops/Left Semi Join` (соединение с помощью вложенных циклов и/или левое полусоединение). Этот запрос представлен в коде T-SQL с помощью подзапроса, но в нем фактически выражено требование на получение строк из таблицы `authors`, для которых имеются соответствия в таблице `titleauthor`. Таким образом, вместо выполнения подзапроса применительно к каждой строке в таблице `authors`, оптимизатор находит достаточно эффективное решение, преобразуя этот запрос в простое полусоединение двух таблиц.

Полная оптимизация

Если оптимизатор не способен найти приемлемый тривиальный план, то существующий план упрощается, а затем передается на этап полной оптимизации. На этапе полной оптимизации предпринимаются попытки оценить стоимость

различных альтернативных способов получения данных, затребованных в запросе, и определить тот способ, который является наименее дорогостоящим. Эти действия не продолжаются до бесконечности, поскольку время, в течение которого оптимизатор перебирает различные способы обработки запроса, прежде чем выбрать наиболее подходящий, строго регламентировано с помощью тайм-аута. Если оптимизатор не находит наиболее эффективный план выполнения ко времени истечения установки тайм-аута, то выбирает наименее дорогостоящий план из тех, которые были найдены к данному моменту, и передает этот план на этап преобразования для последующей передачи на выполнение.

Этап полной оптимизации состоит из четырех меньших этапов (которые можно даже назвать шагами); оптимизатор проходит эти шаги на этапе полной оптимизации запроса. Таковыми являются шаги обработки транзакции, быстрого составления плана, параллельной оптимизации и полной оптимизации. Необходимо учитывать, что оптимизатор может фактически не проходить все эти шаги. Если оптимизатор находит приемлемый план, характеризующийся низкими издержками (план, который является менее дорогостоящим по сравнению с заранее заданным пороговым значением издержек), то передает этот план на этап преобразования для его дальнейшего выполнения, не проходя все оставшиеся шаги. План выполнения может быть выработан и передан на этап преобразования в любой момент, кроме шага параллельной оптимизации. Кроме того, если компьютер, на котором работает программа SQL Server, не является многопроцессорным, то нет необходимости выполнять шаг параллельной оптимизации. Каждый из указанных шагов подробно описан ниже.

Обработка транзакции

В шаге обработки транзакции оптимизатор применяет подмножество потенциальных преобразований, имеющихся в его распоряжении во время полной оптимизации. Назначение этого шага состоит в поиске с максимально возможным быстродействием большинства планов запросов, ориентированных на обработку транзакций, которые не были найдены на этапе создания тривиального плана. По большей части в этом шаге разрешены только преобразования, в которых предусматривается использование соединения с поиском по индексу. Если же рассматриваемый оператор не поддерживает поиск по индексу, то могут быть разрешены преобразования, в которых применяется хэширование.

В шаге обработки транзакции не разрешается применять более сложные преобразования, поскольку такие преобразования откладываются до полной оптимизации. При обработке транзакции не разрешены и не оцениваются по стоимости преобразования, связанные с переупорядочением соединений, переформатированием (динамическим созданием индексов) и т.д. Как уже было сказано, назначение этого шага состоит в том, чтобы быстро провести проверку более широкого набора планов по сравнению с тем набором планов, который рассматривался на этапе получения тривиального плана. Если обнаруживается план, оцениваемая стоимость которого (в единицах измерения времени) меньше чем примерно две десятых секунды, то выбирается этот план и дальнейшая оптимизация не происходит.

Быстрое составление плана

В шаге быстрого составления плана разрешается использовать все правила преобразования, поддерживаемые оптимизатором. При этом допускается выполнение преобразований, предусматривающих переупорядочение соединений (в которых участвуют нижние четыре таблицы дерева реляционных операторов), и выбирается первый план с такой оценкой стоимости, которая не превышает примерно одной секунды.

Параллельная оптимизация

В шаге параллельной оптимизации оптимизатор оценивает операторы и преобразования, позволяющие использовать возможности многопроцессорной обработки применяемого компьютера, если наименее дорогостоящий план, найденный к этому моменту, превышает значение параметра `cost threshold for parallelism` (пороговое значение стоимости при параллельной обработке) процедуры `sp_configure`, а программа SQL Server эксплуатируется на многопроцессорном компьютере). Рассматриваемые на этом этапе операторы будут исследованы более подробно во время полной оптимизации.

Полная оптимизация

Если все другие попытки найти приемлемый план выполнения с низкой стоимостью оканчиваются неудачей, то оптимизатор переходит к выполнению полной оптимизации. Во время выполнения этого шага оптимизатор рекурсивно проходит по дереву реляционных операторов, имеющему такой вид, который это дерево приобрело после прохождения через все другие этапы и шаги оптимизации. Дерево реляционных операторов состоит из реляционных узлов `AND/OR` и реляционных операторов. Узлы `OR` представляют собой множество взаимоисключающих операторов (например, операторов обработки с помощью вложенных циклов, либо операторов хэшированного соединения, либо операторов соединения слиянием), относящихся к конкретному шагу плана. Оптимизатор сравнивает взаимоисключающие операторы друг с другом и выбирает наименее дорогостоящий. Узлы `AND` представляют операторы, в которых оптимизации требует каждый дочерний узел. Оптимизатор проходит по дочерним узлам и оценивает эти узлы отдельно. Стоимость узла `OR` равна стоимости наименее дорогостоящего дочернего узла, а стоимость узла `AND` равна сумме стоимостей дочерних узлов плюс стоимость некоторого оператора.

Рекурсивный алгоритм, выполняемый на этом этапе, по сути, остается аналогичным тому алгоритму, который выполнялся на предыдущих этапах; его отличительная особенность состоит лишь в том, что рассматривается более широкий набор потенциальных преобразований. Например, если предположить, что шаг полной оптимизации выполняется после шага параллельной оптимизации, то в текущий момент может осуществляться оценка стоимости операторов параллельной обработки. Может проводиться сравнение друг с другом различных типов соединений, а также осуществляться оценка стоимости и сопоставление стратегий переформатирования (динамического создания индексов) и других усовершенствованных стратегий оптимизации.

Преобразование

Независимо от того, на каком этапе был фактически выработан план выполнения, этот план должен вначале пройти преобразование для последующей передачи средствам LPE на выполнение. Такое преобразование осуществляется на этапе преобразования процедуры оптимизации. Каждый оптимизированный план запроса перед передачей на выполнение должен быть обработан на данном этапе.

Пределы оптимизации

В оптимизаторе используется несколько методов обеспечения того, чтобы процедура сравнения стоимостей операторов и планов не продолжалась бесконечно или даже невыносимо долго. В какой-то точке процедуры оптимизации оптимизатор может начать приближаться к такому состоянию, что достигнутое им последовательное уменьшение стоимости будет все меньше и меньше. В конечном итоге, если время, затрачиваемое в процессе оценки, намного превышает время выполнения даже самых дорогостоящих планов, то процедура оптимизации вообще не позволит сэкономить такое уж значительное время обработки. В крайнем случае оказывается, что оптимизация и последующее выполнение становится более дорогостоящим, чем выполнение без какой-либо оптимизации. Оптимизатор спроектирован так, чтобы подобные ситуации предотвращались в максимально возможной степени, и обеспечивалась гарантия того, что затраты на оптимизацию оправдывают достигнутое повышение производительности.

В настоящем разделе рассматривается несколько методов ограничения продолжительности работы оптимизатора. Первый из этих методов основан на принципе целевой стоимости. На каждом последующем этапе процедуры оптимизации определяется целевая стоимость для рассматриваемых планов. Планы, стоимость которых превышает заданное целевое значение, больше не рассматриваются, что позволяет оптимизатору быстро сузить выполняемый им поиск и остановиться на наилучших потенциальных планах. На этапе формирования тривиального плана целевая стоимость является бесконечной, а это означает, что не отвергается ни один существующий план. А для последующих этапов оптимизатор уменьшает это значение до 90% стоимости наименее дорогостоящего плана, который был сформирован до этого времени.

Кроме того, в оптимизаторе реализован принцип использования значения тайм-аута. Значение тайм-аута определяется с учетом заданного количества преобразований, которое, согласно прогнозу оптимизатора, может быть выполнено до того, как нужно будет завершить процедуру оптимизации по тайм-ауту. На протяжении выполнения всей процедуры оптимизации оптимизатор проверяет, не было ли превышено заданное значение количества преобразований, и в случае положительного ответа прекращает работу по тайм-ауту. Поскольку количество преобразований, используемое для вычисления тайм-аута, определяется на основе предположения о том, что оптимизатор может выполнить определенное количество преобразований в секунду, само значение тайм-аута лишь приблизительно коррелирует с фактически

израсходованным временем. Связь между значением тайм-аута и фактически израсходованным временем определяется в виде полулинейной функции, которая начинается примерно с 10%-й оценки начальной стоимости и возрастает более или менее линейно, пока наконец не выравнивается после достижения значения израсходованного времени, равного приблизительно 60 секундам. Следует отметить, что точность определения значения тайм-аута зависит от точности оценки значения времени преобразования, поэтому не применяется строго регламентированное понятие значения тайм-аута, определяемого с учетом времени преобразования. Дело в том, что количество преобразований, которые могут быть выполнены оптимизатором в секунду, может изменяться в широких пределах, и в связи с этим точное количество времени, израсходованного до того, как произойдет истечение тайм-аута, выделенного на обработку запроса, может зависеть от самого запроса.

Чтобы определить, когда произошло прекращение работы оптимизатора из-за истечения тайм-аута во время формирования плана запроса, можно использовать флажок трассировки 8675. Кроме того, этот флажок позволяет узнать, в какой момент оптимизатор достиг предела объема памяти, применяемого в программе SQL Server для ограничения объема памяти, используемого оптимизатором (примерно 80% от объема области BPool). К тому же, часто можно установить, когда был достигнут тайм-аут оптимизации, по значению продолжительности компиляции, возвращаемому командой `SET STATISTICS TIME ON`. Если это значение составляет приблизительно 60 секунд, то действительно может обнаруживаться тайм-аут оптимизации, особенно в тех ситуациях, когда безусловно ясно, что оптимизатор потерпел неудачу, пытаясь найти наилучший план для данного конкретного запроса.

Перехват параметров

Прежде чем откомпилировать план выполнения для хранимой процедуры, в программе SQL Server предпринимается попытка "перехватить" (т.е. распознать) значения параметров, передаваемых в эту программу, и использовать такие значения при компиляции плана. Если значения параметров применяются для сужения объема выборки в запросе (т.е. входят в состав предиката `WHERE` или `HAVING`), то перехват параметров позволяет оптимизатору вырабатывать более точный план выполнения, в котором учитываются значения, передаваемые в хранимую процедуру (с использованием статистических гистограмм для столбцов, указанных в критериях выборки), а не исходить исключительно из показателя средней плотности значений в соответствующих столбцах таблицы. Вообще говоря, перехват параметров — очень удобное средство, применение которого приводит к повышению производительности последних версий программы SQL Server по сравнению с более старыми версиями.

Тем не менее, если при первой компиляции плана для некоторой процедуры происходит передача нетипичного значения параметра, то может возникнуть нарушение в работе. При этом в памяти будет кэширован план, который окажется неоптимальным применительно к большинству значений, передаваемых в качестве значений этого параметра. После передачи более типичных значений для них

будет использоваться план, выработанный для нетипичного значения, который потребует более продолжительного времени для выполнения по сравнению с планом, подготовленным для тех значений, которые чаще встречаются на практике.

В такой ситуации можно применить один из нескольких вариантов. Прежде всего рассматриваемая процедура может быть отмечена как предназначенная для автоматической перекомпиляции с помощью опции `WITH RECOMPILE`. Применение этого варианта влечет за собой то, что план процедуры будет перестраиваться при каждом выполнении данной процедуры. Если время перекомпиляции пренебрежимо мало, то указанное решение позволит легко найти выход из той ситуации, в которой значения параметра изменяются в широких пределах, соответствующих распределению этих значений в столбце таблицы.

Кроме того, можно просто выполнить процедуру с использованием опции `WITH RECOMPILE`; такой способ вызова процедуры также приводит к перестройке плана. Аналогично можно воспользоваться процедурой `sp_recompile`, чтобы вынудить программу SQL Server перестроить план процедуры при следующем вызове процедуры на выполнение.

Еще один вариант состоит в том, чтобы "отменить" перехват параметров, применив в качестве критериев выборки в запросе локальные переменные, в которые скопированы значения параметров. Использование в качестве критериев выборки в запросе локальных переменных вместо параметров процедуры, вообще говоря, не рекомендуется, поскольку такой подход исключает возможность применить при вычислении избирательности статистические гистограммы, относящиеся к индексу, но из этого правила есть исключения. Если оптимизатор не может использовать статистические гистограммы для вычисления количества строк, которые могут быть возвращены запросом при применении какого-то конкретного критерия выборки, то оптимизатор использует "магические" числа — заранее заданные оценки процентного соотношения строк, которые будут возвращены в зависимости от применяемого оператора сравнения. В некоторых редких случаях указанные оценки могут оказаться более точными по сравнению с полученными с помощью самой гистограммы. Одним из таких случаев является применение нетипичного значения для просмотра гистограммы, что приводит к получению искаженной оценки количества строк, которые обычно соответствуют переданному параметру.

Иногда можно также реорганизовать запрос, выполнение которого нарушается из-за неправильного перехвата параметров, таким образом, чтобы этот запрос осуществлялся в другом контексте выполнения, связанном с собственным планом выполнения. Этот подход основан на том, что при динамическом выполнении в процедуре блока T-SQL или при вызове другой процедуры, каждый из подобных вызываемых модулей получает свой собственный план выполнения. В методах, основанных на использовании этого подхода для устранения недостатков перехвата параметров, предусматривается применение процедуры `sp_executesql` или функции `EXEC()`, а также разбиение одной процедуры на несколько процедур. Подход, основанный на использовании процедуры `sp_executesql` и разбиении процедуры на несколько процедур, все еще допускает повторное применение плана. А в подходе, основанном на использовании функции `EXEC()`, такая возможность, скорее всего, будет отсутствовать, поскольку рассматриваемый

план, вероятно, придется компилировать повторно при каждом выполнении процедуры. Такой подход может оказаться приемлемым или неприемлемым в зависимости от того, какие действия выполняются рассматриваемой процедурой и сколько времени затрачивается на ее компиляцию.

К тому же, можно явно очистить кэш процедуры с помощью команды DBCC FREEPROCCACHE. Выполнение этой команды вызывает удаление из памяти всех откомпилированных планов, а в связи с этим возникает необходимость осуществить повторную компиляцию планов при очередном вызове на выполнение каждой процедуры. Это — довольно радикальная мера, но может применяться в некоторых обстоятельствах. Например, очистку кэша можно производить до и после вызова на выполнение в ночную смену некоторого задания, связанного с применением большого количества процедур с нетипичными значениями параметров, если в дневную смену те же процедуры обычно вызываются с более типичными параметрами. Это позволяет добиться того, чтобы в ночную смену не использовались планы, выработанные в дневную смену, а планы, созданные для заданий, выполняемых в ночную смену, не применялись повторно на следующий день.

Еще одна ситуация, в которой при использовании перехвата параметров могут возникнуть нарушения в работе, состоит в том, что план выполнения, хранящийся в кэше, отражает переданные в процедуру типичные значения параметров, но возникает необходимость передать в процедуру нетипичное значение одного из параметров и выполнить эту процедуру как можно быстрее. В таком случае проблема состоит не в том, что в кэше находится неоптимальный план, поскольку для большинства запросов план действительно оптимален. Причиной проблемы является то, что для указанных двух запросов не следует совместно использовать один и тот же план выполнения. Например, допустим, что эксплуатируется хранимая процедура, в которую передается код страны для получения данных о сбыте в этой стране. Предположим также, что чаще всего передается код страны "US", поскольку рассматриваемое предприятие находится в Соединенных Штатах и проводит основную часть торговых операций именно в этой стране. В связи с тем фактом, что записи с данными о сбыте в США составляют основную часть записей в таблице с данными о сбыте, оптимизатор вырабатывает план выполнения, в котором используется полный просмотр этой таблицы. В указанной ситуации полный просмотр таблицы эффективнее, чем поиск по индексу, поскольку в любой операции выборки происходит обработка основной части строк. Но иногда в процедуру передается код другой страны, допустим, такой страны, в которой происходит лишь небольшое количество сделок по сбыту. Пользователь рассчитывает на то, что соответствующий запрос вернет результаты довольно быстро, поскольку в этом запросе в конечном итоге требуется обработать лишь несколько строк, но ускорения работы не происходит. И в данном запросе осуществляется полный просмотр таблицы, поскольку повторно используется план, первоначально откомпилированный при передаче значения "US". Одним из возможных решений является реорганизация указанной процедуры и создание на ее основе нескольких процедур, допустим, одной процедуры для получения данных о сбыте в США и еще одной — для получения данных о сбыте в других странах. При условии, что количество данных о сбыте в странах, отличных от Соединенных Штатов, относительно

невелико, а сами эти данные распределены достаточно равномерно, выполнение запросов для получения данных о сбыте в этих странах, скорее всего, приведет к использованию поиска по индексу (если существует соответствующий индекс). И наоборот, в запросах на получение данных о сбыте в Соединенных Штатах будет по-прежнему применяться полный просмотр таблицы, поскольку именно в этом состоит наиболее эффективный способ обслуживания таких запросов. Указанный подход, основанный на использовании нескольких процедур в указанной форме, может оказаться наиболее приемлемым решением, особенно если для компиляции плана требуется много времени (и поэтому вариант с автоматической перекомпиляцией при каждом выполнении подходит не очень хорошо).

Автоматическая параметризация

В целях повышения вероятности повторного использования плана программа SQL Server предпринимает попытки автоматической параметризации произвольных запросов. Под "автоматической параметризацией" подразумевается выполняемая сервером замена значения константы в произвольном запросе маркером параметра для того, чтобы откомпилированный для такого запроса план можно было повторно использовать с другими значениями константы. Например, рассмотрим следующий запрос:

```
SELECT * FROM Orders WHERE OrderId=10248
```

В этом запросе значение `10248` является константой. Если бы сервер не был способен автоматически параметризировать этот запрос, то при следующем выполнении запроса со значением, отличным от `10248`, пришлось бы формировать отдельный план выполнения. Вместо этого в сервере применяются достаточно интеллектуальные алгоритмы, позволяющие заменить значение `10248` маркером параметра и передать `10248` в качестве значения параметра для данного конкретного контекста выполнения. В других случаях вызова того же запроса с иными значениями параметра каждый раз будет создаваться отдельный контекст выполнения, но использоваться один и тот же план выполнения.

Пользователь может не полагаться на способность сервера правильно подбирать параметры для конкретного запроса, а задать эти параметры непосредственно с помощью процедуры `sp_executesql` или с использованием маркеров параметров в тексте запроса. Если выбран такой подход, неоднозначность исключается, и пользователь получает больший контроль над тем, какие типы данных применяются для каждого параметра. Программа SQL Server определяет логическим путем тип данных каждого автоматически вводимого параметра с учетом переданного значения константы, но иногда выдвинутое программой предположение является ошибочным, что приводит к несовпадению типов данных между параметром и столбцом, с которым сравнивается этот параметр. Такая ситуация может препятствовать применению индекса, поэтому следует учитывать вероятность подобного развития событий. Например, в приведенном выше запросе программа SQL Server принимает предположение, что параметр с обозначением идентификатора заказа должен иметь тип данных `smallint`, поскольку

константа имеет значение 10248, для которого хорошо подходит двухбайтовый тип данных smallint программы SQL Server. Но если в действительности столбец OrderId таблицы Orders представлен в виде четырехбайтовых целочисленных значений, то в план выполнения придется включить оператор CONVERT, чтобы компенсировать различие между двумя указанными типами данных.

Узнать о том, был ли параметризован автоматически параметризуемый запрос, можно с помощью средств текстового или графического отображения плана выполнения. Если какой-либо произвольный запрос был автоматически параметризован, то в тексте запроса можно обнаружить вместо по меньшей мере некоторых значений констант такие метки-заполнители для параметров, как @1; кроме того, обнаруживается, что такие метки-заполнители используются в запросе в составе критериев выборки.

Для проверки результатов применения автоматической параметризации удобно также использовать таблицу syscacheobjects. Если в программе SQL Server был автоматически параметризован какой-либо произвольный запрос, то автоматически сформированные метки-заполнители для параметров и типы их данных можно найти в начале столбца sql в строках данного запроса таблицы syscacheobjects. Такие метки-заполнители используются также в тексте запроса, приведенном в столбце sql, в качестве критерия выборки данных, возвращаемых запросом, как показано ниже.

```
(@1 smallint)SELECT * FROM [Orders] WHERE [OrderId]=@1
```

Кроме того, для контроля над тем, происходит ли автоматическая параметризация того или иного запроса, нужно также разрешить использование флажка трассировки 8759. Если флажок трассировки 8759 разрешен, то первая часть автоматически параметризованного запроса записывается в журнал регистрации ошибок программы SQL Server следующим образом:

```
SAFE auto-paramd query: (@1 smallint) SELECT * FROM [Orders] WHERE
 [OrderId]=@1
```

Применение индексов

Трудно назвать подход, в большей степени способствующий повышению производительности запросов, чем создание удобных и эффективных индексов. Основные сложности, связанные с эксплуатацией крупных банков данных, обусловлены выполнением операции ввода-вывода, поэтому необходимо стремиться уменьшить объем ввода-вывода и затраты времени на ввод-вывод в максимально возможной степени. Для этого можно применять кэширование, наращивать обрабатывающие мощности и устанавливать быстродействующие жесткие диски. Но ничто не отражается на производительности запросов так существенно, как индексация. Если отсутствует применимый индекс, то для программы SQL Server нет иного способа, чем полностью просматривать одну или несколько таблиц в поиске нужных данных. Если же приходится применять соединение двух или большего количества таблиц, то программе SQL Server может потребоваться просматривать их несколько раз, чтобы найти все данные, необходимые для удовлетворения запроса. Индексы позволяют в значительной степени ускорить процесс поиска данных, а также процесс соединения таблиц друг с другом.

Местонахождение данных об индексах

Информация системного уровня об индексах, применяемых в программе SQL Server, хранится в системной таблице sysindexes. В таблице sysindexes имеется строка с данными о каждом индексе, а для идентификации самого индекса служит столбец indid этой таблицы. В столбце indid находятся целочисленные значения с отсчетом от 1, которые показывают, в каком порядке создавались индексы; кластеризованный индекс всегда имеет значение indid, равное 1. Если в таблице нет кластеризованного индекса, то в таблице sysindexes имеется строка со значением для самой таблицы, в которой указано значение indid, равное 0.

Схема распределения индекса

В программе SQL Server экстенты, принадлежащие таблице или индексу, отслеживаются с использованием страниц со схемами распределения индексов (Index Allocation Map – IAM). В таком объекте, как динамическая область памяти или индекс, имеется по меньшей мере по одной странице IAM для каждого файла, в котором для этого объекта распределены экстенты. Схема IAM представляет собой битовую структуру отображения, которая отображает экстенты на объекты; каждый бит этой структуры указывает, принадлежит ли соответствующий экстент тому объекту, в котором находится данная схема IAM. Каждая структура отображения IAM охватывает диапазон из 512 тысяч страниц. Первая страница IAM, относящаяся к индексу, хранится в столбце First-IAM таблицы sysindexes. Страницы IAM распределены в файле базы данных случайным образом и связаны друг с другом в виде одной цепочки. Но даже несмотря на то, что схемы IAM позволяют программе SQL Server, по существу, заранее выполнять выборку экстентов таблицы, в дальнейшем все равно приходится просматривать отдельные строки экстента, поскольку схема IAM служит просто в качестве метода доступа к самим страницам.

Типы индексов

В программе SQL Server поддерживаются индексы двух типов – кластеризованные и некластеризованные. Оба эти типа имеют много общих особенностей. В частности, оба эти типа состоят из страниц, хранящихся в B-деревьях (B – сокращение от balanced, сбалансированный). На уровнях узлов индексов каждого типа расположены указатели на страницы следующего уровня, а на листовом уровне находятся значения ключей.

B-деревья

Как уже было сказано, физическое хранение индексов в программе SQL Server осуществляется в виде B-деревьев. В частности, B-деревья обеспечивают поиск в данных с использованием определенной разновидности алгоритма двоичного поиска. Кроме того, в индексах на основе B-деревьев ключи хранятся так, что ненамного отличающиеся значения находятся близко друг к другу, а само дерево постоянно подвергается повторному уравновешиванию для обеспечения того, чтобы каждое конкретное значение могло быть достигнуто с помощью минимального количества переходов по страницам индекса. Поскольку B-деревья являются

уравновешенными, стоимость поиска любой строки остается относительно постоянной, независимо от того, где находится эта строка.

Первым узлом в индексе на основе B-дерева является корневой узел. Указатель на корневой узел каждого индекса хранится в столбце `root` таблицы `sysindexes`. Проводя поиск данных с помощью индекса, программа SQL Server начинает с корневого узла, затем проходит через все промежуточные уровни, которые могут находиться в индексе, и в конечном итоге либо находит, либо не находит требуемые данные в листовых узлах индекса, находящихся на самом нижнем уровне. Количество промежуточных уровней зависит от размера таблицы, размера ключа индекса и количества столбцов в ключе. Очевидно следующее — чем больше данных находится в таблице и чем больше места занимает каждый ключ, тем больше страниц должно находиться в индексе.

Страницы индекса, относящиеся к уровню выше листового, называются *страницами узлов*. Каждая строка на странице узлов содержит один или несколько ключей, а также указатель на страницу следующего уровня, которой соответствует строка первого ключа. На этом основана общая структура B-дерева. Программа SQL Server проходит по указанным связям до тех пор, пока не обнаруживает искомые данные или не достигает конца цепочки связей в узле листового уровня. Листовой уровень B-дерева содержит значения ключей, а при использовании некластеризованных индексов — закладки с информацией о соответствующем кластеризованном индексе или динамической области памяти. Значения ключей хранятся последовательно, причем в версии SQL Server 2000 и последующих версиях могут быть отсортированы в возрастающем или убывающем порядке.

В отличие от некластеризованных индексов, листовой узел кластеризованного индекса фактически содержит сами данные. Закладки в индексе отсутствуют, поскольку нет необходимости в их использовании. Если в базе данных имеется кластеризованный индекс, то на уровне листовых узлов этого индекса находятся сами данные.

Страницы данных в таблице хранятся в виде цепочки страниц, т.е. двусвязного списка страниц. Если в базе данных применяется кластеризованный индекс, то порядок строк на каждой странице и порядок страниц в цепочке определяются ключом индекса. Применение ключа кластеризованного индекса приводит к сортировке данных, поэтому важно выбрать этот ключ правильно. При выборе ключа необходимо учитывать некоторые важные условия, в том числе перечисленные ниже.

- Ключ должен иметь минимально возможный размер, поскольку он будет служить в качестве закладки для каждого некластеризованного индекса.
- Ключ необходимо выбирать так, чтобы он хорошо подходил для обычно применяемых запросов `ORDER BY` и `GROUP BY`.
- Ключ должен достаточно хорошо подходить для обычных запросов поиска в диапазоне (запросов, в которых происходит обращение к диапазону строк с учетом значений в столбце или столбцах).
- Ключ должен охватывать множество таких столбцов, данные в которых не обновляются слишком часто, поскольку обновление ключа кластеризованного индекса таблицы может потребовать перемещения некоторых строк и обновления закладок в каждом некластеризованном индексе таблицы.

Начиная с версии SQL Server 7.0, все кластеризованные индексы имеют уникальные ключи. Если создаваемый кластеризованный индекс не является уникальным (например, если индекс создается с помощью команды CREATE INDEX без ключевого слова UNIQUE), то программа SQL Server вынуждает этот индекс стать уникальным, добавляя по мере необходимости к значениям ключа четырехбайтовое значение, называемое *унификатором*, чтобы одинаковые значения ключа стали разными.

Страницы листового уровня в некластеризованном индексе содержат ключи индекса и закладки, указывающие на основополагающий объект — кластеризованный индекс или динамическую область памяти. Закладка может иметь одну из двух форм. Если в таблице существует кластеризованный индекс, то закладка представляет собой ключ кластеризованного индекса. Если в кластеризованном и некластеризованном индексах совместно используется общий столбец ключа, то в памяти хранится только единственный экземпляр каждой закладки. Если же кластеризованный индекс отсутствует, то закладка состоит из идентификатора строки (Row IDentifier — RID), в который входят номер файла, номер страницы и номер слота строки, на которую ссылается значение ключа некластеризованного индекса.

Даже тот факт, что применение динамической области памяти (так называется таблица без кластеризованного индекса) приводит к тому, что в некластеризованных индексах ссылка на такую таблицу содержит информацию о физическом местонахождении данных, служит достаточно весомым обоснованием рекомендации, согласно которой на каждой формируемой таблице необходимо создавать кластеризованный индекс. При отсутствии такого индекса внесение изменений в таблицу, которое вызывает разбиение страниц, может повлечь за собой цепную реакцию внесения изменений во все некластеризованные индексы таблицы, поскольку физическое местонахождение строк, на которые ссылаются эти индексы, будет изменяться довольно часто. В этом фактически состоял один из основных недостатков средств индексации SQL Server, которые применялись до версии 7.0, поскольку в некластеризованных индексах всегда хранилась информация о физическом местонахождении строк, а не значения ключей кластеризованного индекса. В результате этого средства индексации испытывали отрицательно влияние, связанное с частым изменением физического местонахождения строк в базовой таблице.

Некластеризованные индексы в наибольшей степени подходят для выполнения одноэлементных выборок — запросов, которые приводят к возврату единственной строки. Дело в том, что после перехода по узлам некластеризованного B-дерева для доступа к фактическим данным достаточно выполнить всего лишь одну операцию ввода-вывода страниц, т.е. прочитать страницу из основополагающей таблицы.

Покрывающие индексы

Некластеризованный индекс называется "покрывающим" некоторый запрос, если в нем содержатся все столбцы, требуемые в запросе. Применение такого индекса позволяет миновать шаг поиска закладки и просто возвратить данные, требуемые в запросе, из собственного B-дерева индекса. Если в таблице имеется кластеризованный индекс, то запрос может быть покрыт с использованием комбинации ключевых столбцов некластеризованного и кластеризованного индексов, поскольку

ключ кластеризованного индекса представляет собой закладку некластеризованного индекса. Иными словами, если некластеризованный индекс сформирован на столбцах LastName и FirstName, а ключом кластеризованного индекса является столбец CustomerID, то запрос, в котором необходимо получить данные из столбцов CustomerID и LastName, может быть покрыт с помощью некластеризованного индекса. Подход, в котором используется покрывающий некластеризованный индекс, не намного хуже по своим характеристикам того подхода, в котором используется несколько кластеризованных индексов, созданных на одной и той же таблице.

Проблемы производительности

Вообще говоря, рекомендуется использовать ключи индексов, имеющие как можно меньший формат представления. Применение ключей, имеющих большие размеры, приводит к увеличению объема ввода-вывода и к уменьшению количества строк с ключевыми значениями, которые могут поместиться на каждой странице B-дерева. В результате для индекса применяется большее количество страниц, чем было бы в ином случае, а индекс занимает больший объем дискового пространства. На практике принятая стратегия индексации в основном должна соответствовать конкретным деловым потребностям. Например, если применяется такой запрос, для выполнения которого требуется чрезвычайно продолжительное время, поскольку для него необходим индекс с такими ключевыми столбцами, которые отсутствуют в каком-либо из текущих индексов, то действительно может потребоваться расширить существующий индекс или создать новый.

Безусловно, при рассмотрении вопроса о введении дополнительных индексов или столбцов индексов необходимо учитывать определенные компромиссы, например, оценивать, как модификация индексов отразится на производительности операций DML. Операции сопровождения и обновления индексов таблицы выполняются в связи с любой операцией добавления или изменения данных, поэтому введение каждого нового индекса приводит к определенному повышению издержек, связанных с сопровождением индексов. Чем больше количество введенных индексов, тем медленнее выполняются операции обновления, вставки и удаления, применяемые к основополагающим таблицам, поэтому важно, чтобы индексы оставались настолько компактными и охватывающими небольшое количество столбцов, насколько это возможно, и вместе с тем соответствовали деловым потребностям, для удовлетворения которых предназначена разрабатываемая система.

Пересечение индексов

До появления версии 7.0 оптимизатор запросов программы SQL Server должен был использовать только один индекс в расчете на каждую таблицу при поиске способа выполнения запроса. Версия SQL Server 7.0 и последующие версии позволяют применять несколько индексов в расчете на каждую таблицу и находить пересечение множеств закладок этих индексов еще до того, как начнется расходование реальных ресурсов на выборку данных из основополагающей таблицы. Реализация такого подхода привела к тому, что изменилось устройство индексов и стали применяться другие критерии выбора ключей, как вскоре будет описано.

Фрагментация индекса

Величиной фрагментации индекса можно управлять, задавая для индекса значение коэффициента заполнения и регулярно выполняя операции дефрагментации. Коэффициент заполнения индекса во многом влияет на производительность. С одной стороны, если при создании индекса был задан относительно низкий коэффициент заполнения, то количество случаев разделения страниц во время выполнения операций вставки уменьшается. Очевидно, что если страницы индекса заполнены лишь частично, то вероятность возникновения необходимости разделить одну из таких страниц для того, чтобы вставить новые строки, меньше по сравнению с вероятностью разделения полностью заполненных страниц. С другой стороны, высокое значение коэффициента заполнения позволяет формировать плотно заполненные страницы, что способствует уменьшению количества операций ввода-вывода, необходимых для обслуживания запроса. Последний метод часто применяется в хранилищах данных. Выборка страниц, которые заполнены лишь частично, связана с непроизводительным расходованием пропускной способности средств ввода-вывода.

Значение коэффициента заполнения индекса влияет только на страницы индекса листового уровня. Программа SQL Server обычно резервирует достаточный объем свободного пространства в страницах индекса промежуточных уровней для хранения по меньшей мере одной строки индекса максимального размера. Если же требуется, чтобы спецификация коэффициента заполнения распространялась не только на страницы листового уровня, но и на страницы промежуточных уровней, то необходимо ввести опцию `PAD_INDEX` оператора `CREATE INDEX`. Опция `PAD_INDEX` служит для программы SQL Server указанием на то, что заданное значение коэффициента заполнения должно распространяться на страницы индекса промежуточных уровней. Если заданное значение коэффициента заполнения настолько велико, что на страницах промежуточных уровней не хватает места даже для одной строки (например, коэффициент заполнения равен 100%), то программа SQL Server корректирует заданное в процентах значение для того, чтобы можно было поместить на страницу хотя бы одну строку. Если же заданное значение коэффициента заполнения является таким низким, что на страницах промежуточных уровней нельзя сохранить по меньшей мере две строки, то программа SQL Server корректирует значение коэффициента заполнения, выраженное в процентах, которое относится к страницам промежуточных уровней, для того, чтобы на каждой странице могли поместиться по меньшей мере две строки.

Следует учитывать, что значение коэффициента заполнения, заданное для индекса, в дальнейшем не поддерживается. Это значение применяется при первоначальном создании индекса, но после этого больше не учитывается. В качестве инструментального средства, позволяющего определить, насколько заполнены страницы в таблице и/или индексе, фактически служит команда `DBCC SHOWCONTIG`. Наиболее важными показателями, которые при этом следует анализировать, являются показатели `Logical Scan Fragmentation` (Фрагментация логического просмотра) и `Avg. Page Density` (Средняя плотность страницы). Команда `DBCC SHOWCONTIG` показывает три типа фрагментации — фрагментация просмотра экстента, фрагментация логического просмотра и плотность просмотра. Для устра-

нения фрагментации логического просмотра используется команда DBCC INDEXDEFRAG, а для полного устранения фрагментации таблицы и/или индекса применяется перестройка индексов.

В листинге 12.1 приведен пример вывода команды DBCC SHOWCONTIG, применяемой к таблице Customers базы данных Northwind.

Листинг 12.1. Пример вывода команды DBCC SHOWCONTIG

```
DBCC SHOWCONTIG (Customers)

(Результаты)

DBCC SHOWCONTIG scanning 'Customers' table...
Table: 'Customers' (2073058421); index ID: 1, database ID: 6
TABLE level scan performed.
- Pages Scanned.....................................: 5
- Extents Scanned...................................: 3
- Extent Switches...................................: 4
- Avg. Pages per Extent.............................: 1.7
- Scan Density [Best Count:Actual Count].....: 20.00% [1:5]
- Logical Scan Fragmentation ..............:  40.00%
- Extent Scan Fragmentation ...............: 66.67%
- Avg. Bytes Free per Page.................: 3095.2
- Avg. Page Density (full).................: 61.76%
```

Вполне очевидно, что таблица Customers немного фрагментирована. Показатель Logical Scan Fragmentation составляет 40%, а показатель Avg. Page Density – 61,76% (строки с этими показателями выделены полужирным шрифтом). Иными словами, страницы этой таблицы остаются незаполненными в среднем приблизительно на 40%. Выполним дефрагментацию кластеризованного индекса этой таблицы и определим, достигнуто ли улучшение. Данные, полученные в результате выполнения указанных действий, приведены в листинге 12.2.

Листинг 12.2. Результаты проверки таблицы Customers после дефрагментации

```
DBCC INDEXDEFRAG(Northwind,Customers,1)

(Результаты)

Pages Scanned   Pages Moved   Pages Removed
-------------   -----------   -------------
1               0             1

   DBCC SHOWCONTIG (Customers)

(Результаты)

DBCC SHOWCONTIG scanning 'Customers' table...
Table: 'Customers' (2073058421); index ID: 1, database ID: 6
TABLE level scan performed.
- Pages Scanned.....................................: 4
- Extents Scanned...................................: 3
- Extent Switches...................................: 2
- Avg. Pages per Extent.............................: 1.3
```

```
- Scan Density [Best Count:Actual Count].......: 33.33% [1:3]
- Logical Scan Fragmentation ..............:  25.00%
- Extent Scan Fragmentation ...................: 66.67%
- Avg. Bytes Free per Page.....................: 1845.0
- Avg. Page Density (full).................: 77.21%
```

Вполне очевидно, что результаты выполнения команды DBCC INDEXDEFRAG оказались весьма благоприятными. Показатель Logical Scan Fragmentation снизился до 25%, а показатель Avg. Page Density теперь лишь ненамного превышает 77%; это означает, что достигнуто улучшение примерно на 15%.

По умолчанию команда DBCC SHOWCONTIG позволяет получить информацию только об узлах листового уровня. Для просмотра других уровней таблицы/индекса необходимо задать опцию ALL_LEVELS (листинг 12.3).

Листинг 12.3. Использование команды DBCC SHOWCONTIG для просмотра данных о фрагментации на всех уровнях

```
DBCC SHOWCONTIG (Customers) WITH TABLERESULTS, ALL_LEVELS

(Результаты приведены в сокращенном виде)

ObjectName  IndexName     AveragePageDensity   ScanDensity     LogicalFragmenta
----------  ------------  -------------------  --------------  ----------------
Customers   PK_Customers  77.205337524414063   33.3333333329   25.0
Customers   PK_Customers  0.95132195949554443  0.0             0.0
```

В табл. 12.1 перечислены основные элементы данных, приведенные в отчете команды DBCC SHOWCONTIG, и показано, что они означают.

Таблица 12.1. Основные поля в выводе команды DBCC SHOWCONTIG

DBCC SHOWCONTIG	Описание
Avg. Bytes Free per Page	Среднее количество свободных байтов на каждой странице
Pages Scanned	Количество страниц, к которым был осуществлен доступ
Extents Scanned	Количество экстентов, к которым был осуществлен доступ
Out of order pages (этот показатель не отображается, но используется для вычисления показателя Logical Scan Fragmentation)	Количество таких случаев, когда в ходе просмотра текущая страница имела меньший номер, чем предыдущая
Extent Switches	Количество таких случаев, когда в ходе просмотра текущая страница относилась к другому экстенту, чем предыдущая

Для определения общей фрагментации таблицы и/или индекса автор использует поля Logical Scan Fragmentation и Avg. Page Density. На практике обычно обнаруживается, что значения этих полей по мере увеличения или уменьшения фрагментации во времени изменяются в одну и ту же сторону.

Дефрагментация

Как было показано выше, команда DBCC INDEXDEFRAG представляет собой удобный способ дефрагментации индекса. Операция, заданная этой командой, выполняется в оперативном режиме, поэтому в ходе выполнения команды индекс все еще остается применимым. Тем не менее реорганизация индекса с помощью данной команды осуществляется только на уровне листовых узлов; при этом происходит своего рода пузырьковая сортировка страниц листового уровня. Для полной дефрагментации индекса его необходимо перестроить. Для этой цели существует несколько способов. Прежде всего, можно просто удалить и воссоздать индекс с помощью операторов DROP INDEX /CREATE INDEX. Но такой способ имеет недостаток, связанный с тем, что во время повторного создания индекс находится в автономном режиме, кроме того, не разрешено уничтожать индексы, которые поддерживают ограничения. Еще один способ состоит в том, что можно использовать команду DBCC DBREINDEX или конструкцию DROP_EXISTING оператора CREATE INDEX, но и в этом случае индекс до окончания его перестройки остается недоступным. Единственным преимуществом данного способа является то, что при эксплуатации производственной версии Enterprise Edition программы SQL Server индекс может создаваться параллельно. Весь объем работы, выполняемый в операции параллельного создания индекса в программе SQL Server, распределяется по процессорам в многопроцессорной системе почти равномерно, поэтому в многопроцессорной системе продолжительность времени пребывания индекса в автономном режиме в ходе его воссоздания может оказаться намного меньше, чем в такой однопроцессорной системе, которая ничем не отличается от многопроцессорной, если не считать того, что в этих системах имеется разное количество процессоров.

Вообще говоря, команда DBCC INDEXDEFRAG является наилучшим инструментальным средством решения задачи дефрагментации индекса, за исключением тех ситуаций, когда обнаруживается обширная фрагментация на уровнях индекса, отличных от листового, и создается впечатление, что такая фрагментация приводит к недопустимо резкому снижению производительности выполнения запроса. Как уже было сказано выше, для проверки фрагментации на других уровнях индекса можно передать опцию ALL_LEVELS в команду DBCC SHOWCONTIG.

Команда DBCC INDEXDEFRAG не только обеспечивает дефрагментацию индекса на листовом уровне за счет сортировки страниц листового уровня, но и осуществляет этап уплотнения, на котором страницы индекса уплотняются с использованием первоначального коэффициента заполнения страниц в качестве целевого значения. В ходе выполнения этапа уплотнения предпринимается попытка оставить ко времени завершения этапа на каждой странице достаточно места по меньшей мере для одной строки. Если указанная команда не может получить блокировку на какую-то конкретную страницу во время уплотнения, то пропускает эту страницу. Кроме того, команда удаляет все страницы, которые в результате уплотнения становятся полностью пустыми.

Индексы на представлениях и вычисленных столбцах

Создание индекса на представлении или вычисленном столбце обеспечивает преобразование в постоянно хранимые таких данных, которые в противном случае существовали бы только как виртуальные. Обычно данные, возвращаемые из представления, существуют только в таблицах, к которым обращается это представление. При формирования запроса к представлению этот запрос объединяется с запросом, лежащим в основе представления, после чего осуществляется выборка данных из основополагающих объектов (таблиц). Такое же утверждение является справедливым по отношению к вычисленным столбцам. В обычных обстоятельствах данные, возвращаемые вычисленным столбцом, фактически не существуют, независимо от того, какие столбцы или выражения используются для получения этих данных. Каждый раз, когда вычисленные данные запрашиваются из таблицы, в которой находятся соответствующие этим данным исходные данные, повторно вычисляется выражение, предназначенное для получения расчетных данных, поэтому расчетные данные вырабатываются динамически.

Приступая к формированию индексов на представлении, следует начинать с создания уникального кластеризованного индекса. Именно на этом этапе происходит создание реальных постоянно хранимых данных. Как и в таблицах, кластеризованный индекс, созданный на представлении, фактически хранит в своих узлах листового уровня сами данные. А после формирования кластеризованного индекса можно также приступить к созданию на представлении некластеризованных индексов.

В этом состоит отличие представлений от вычисленных столбцов в таблицах. При использовании вычисленных столбцов не требуется вначале создавать кластеризованный индекс, чтобы можно было сформировать некластеризованные индексы. Дело в том, что вычисленные столбцы служат просто в качестве значений ключа индекса, поэтому для вычисленных столбцов вполне подходят некластеризованные индексы.

Условия создания индексов

Для того чтобы можно было создать индекс на представлении или на вычисленном столбце в программе SQL Server, требуется, чтобы были заданы определенные значения семи опций SET. Опции SET и требуемые для них значения перечислены в табл. 12.2. Как показывает эта таблица, все опции SET, кроме NUMERIC_ROUNDABORT, должны иметь значение ON.

Таблица 12.2. Значения опций SET, которые должны быть заданы перед созданием индекса на представлении или на вычисленном столбце

Опция SET	Требуемое значение
ARITHABORT	ON
CONCAT_NULL_YIELDS_NULL	ON
QUOTED_IDENTIFIER	ON
ANSI_NULLS	ON

Окончание табл. 12.2

Опция SET	Требуемое значение
ANSI_PADDING	ON
ANSI_WARNINGS	ON
NUMERIC_ROUNDABORT	OFF

Для работы с индексированными представлениями и с индексами на вычисленных столбцах могут использоваться только детерминированные выражения. Детерминированным выражением является такое выражение, которое всегда возвращает одни и те же выходные данные при передаче в него одних и тех же входных данных. Например, выражение SUBSTRING('He who loves money more than truth will end up poor',23,7) является детерминированным, а выражение GETDATE — нет.

Для проверки того, является ли представление или столбец индексируемым, можно воспользоваться функциями OBJECTPROPERTY и COLUMNPROPERTY языка Transact-SQL (листинг 12.4).

Листинг 12.4. Проверка того, является ли представление или столбец индексируемым

```
USE Northwind
SELECT OBJECTPROPERTY (OBJECT_ID('Invoices'), 'IsIndexable')
SELECT COLUMNPROPERTY (OBJECT_ID('syscomments'), 'text' ,
                       'IsIndexable')
SELECT COLUMNPROPERTY (OBJECT_ID('syscomments'), 'text' ,
                       'IsDeterministic')

(Результаты)

-----------
0

-----------
0

-----------
0
```

Еще одним, последним, условием применительно к представлениям является то, что представление индексируемо, только если оно было создано с помощью опции SCHEMABINDING. Если представление создано с опцией SCHEMABINDING, это вынуждает программу SQL Server исключать возможность уничтожения объектов, на которые ссылается это представление. Соответствующие объекты могут быть уничтожены только после того, как само представление будет уничтожено или модифицировано таким образом, что в нем будет удалена опция SCHEMABINDING. Кроме того, завершаются неудачей операторы ALTER TABLE, применяемые к таблицам, на которые ссылается представление, если эти операторы не соответствуют определению представления. В приведенном выше примере представление Invoices не является индексируемым, поскольку оно не было создано с опцией

SCHEMABINDING. А ниже приведена версия Invoices2 представления Invoices, которая является индексируемой, как показывают приведенные в том же листинге результаты проверки свойства IsIndexable (листинг 12.5).

Листинг 12.5. Индексируемое представление Invoices2

```
CREATE VIEW Invoices2
WITH SCHEMABINDING
AS
SELECT Orders.ShipName, Orders.ShipAddress, Orders.ShipCity,
  Orders.ShipRegion, Orders.ShipPostalCode, Orders.ShipCountry,
  Orders.CustomerID, Customers.CompanyName AS CustomerName,
  Customers.Address, Customers.City, Customers.Region,
  Customers.PostalCode, Customers.Country, Orders.OrderID,
  Orders.OrderDate, Orders.RequiredDate, Orders.ShippedDate,
  Shippers.CompanyName As ShipperName, "Order Details".ProductID,
  Products.ProductName, "Order Details".UnitPrice,
  "Order Details".Quantity, "Order Details".Discount,
  Orders.Freight
  FROM dbo.Shippers INNER JOIN
    (dbo.Products INNER JOIN
      (
        (dbo.Employees INNER JOIN
          (dbo.Customers INNER JOIN dbo.Orders
            ON Customers.CustomerID = Orders.CustomerID)
          ON Employees.EmployeeID = Orders.EmployeeID)
        INNER JOIN dbo.[Order Details]
          ON Orders.OrderID = "Order Details".OrderID)
    ON Products.ProductID = "Order Details".ProductID)
  ON Shippers.ShipperID = Orders.ShipVia
GO
SELECT OBJECTPROPERTY (OBJECT_ID('Invoices2'), 'IsIndexable')

(Результаты)

-----------
1
```

Обратите внимание на то, что во всех ссылках на объекты в этой версии используются имена, состоящие из двух частей (отметим, что имена не были таковыми в первоначальном представлении Invoices). Для создания представления с опцией SCHEMABINDING требуется, чтобы во всех ссылках на объекты применялись имена, состоящие из двух частей.

После создания индекса на представлении оптимизатор может использовать такой индекс при выполнении запросов к этому представлению. В действительности в версии Enterprise Edition программы SQL Server оптимизатор может также использовать индекс представления для обслуживания запроса к основополагающим таблицам представления, если будет обнаружено, что применение такого индекса приводит к снижению стоимости выполнения запроса, измеряемой продолжительностью времени выполнения. При обычных обстоятельствах индексированные представления вообще не используются оптимизатором, если только представление не применяется в версии Enterprise Edition. Например, рассмотрим следующий индекс и запрос (листинг 12.6).

Листинг 12.6. Запрос к индексированному представлению

```
CREATE UNIQUE CLUSTERED INDEX inv ON invoices2 (orderid, productid)
GO
SELECT * FROM invoices2 WHERE orderid=10844 AND productid=22

(Результаты приведены в сокращенном виде)

ShipName           ShipAddress  ShipCity ShipRegion ShipPostalCode
------------------ ------------ -------- ---------- --------------
Piccolo und mehr   Geislweg 14  Salzburg NULL       5020
```

Выдержка из плана выполнения для этого запроса приведена в листинге 12.7.

Листинг 12.7. План выполнения запроса, показанного в листинге 12.6

```
StmtText
-----------------------------------------------------------------------------
SELECT * FROM [invoices2] WHERE [orderid]=@1 AND [productid]=@2
  |--Nested Loops(Inner Join)
       |--Nested Loops(Inner Join)
       |    |--Nested Loops(Inner Join, OUTER REFERENCES:([Orders].[ShipVia]))
       |    |    |--Nested Loops(Inner Join, OUTER REFERENCES:([Orders].[Employ
       |    |    |    |--Nested Loops(Inner Join, OUTER REFERENCES:([Orders].[C
       |    |    |    |    |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[
       |    |    |    |    |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[
       |    |    |    |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Employ
       |    |    |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Shippers].[
       |    |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Order Details].[
       |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Products].[PK_Product
```

В этом листинге правая часть текста плана обрезана, но рассматриваемый текст все равно позволяет определить, что индекс представления не используется, даже несмотря на то, что в этом индексе содержатся оба столбца, которые применяются в критериях выборки данного запроса. Такой результат является вполне предсказуемым в версиях программы SQL Server, отличных от версии Enterprise Edition. По умолчанию при формировании плана выполнения индексы представлений рассматриваются только в версии Enterprise Edition программы SQL Server. Тем не менее из этой ситуации есть выход. В версиях программы SQL Server, отличных от Enterprise Edition, можно использовать в запросе подсказку NOEXPAND, чтобы вынудить оптимизатор учитывать наличие индекса представления. Ниже еще раз приведен рассматриваемый запрос, но на этот раз с ключевым словом NOEXPAND, а также показан результирующий план выполнения запроса (листинг 12.8).

Листинг 12.8. Использование ключевого слова NOEXPAND, позволяющего вынудить оптимизатор применить индекс представления

```
SELECT * FROM invoices2 WITH (NOEXPAND) WHERE orderid=10844 AND productid=22

(Результаты)

StmtText
-----------------------------------------------------------------------------
SELECT * FROM invoices2 (NOEXPAND) WHERE orderid=10844 AND productid=22
  |--Clustered Index Seek(OBJECT:([Northwind].[dbo].[Invoices2].[inv]), SEEK:([
```

Обратите внимание на то, что теперь индекс используется. Введение ключевого слова NOEXPAND вынуждает оптимизатор использовать индекс представления, даже если такое решение приводит к созданию неоптимального плана. Поэтому опцию NOEXPAND следует рассматривать под таким же критическим углом, как и все другие подсказки в запросах. Лучше всего позволить оптимизатору выполнять свою работу и вмешиваться самому лишь тогда, когда нет другого выхода.

Блокировка и индексы

Одним из характерных признаков того, что на таблице должен быть создан кластеризованный индекс, является возникновение таких ситуаций, при которых на таблице устанавливаются блокировки RID. В программе SQL Server никогда не устанавливаются блокировки RID на таблице с кластеризованным индексом; вместо этого всегда устанавливаются блокировки ключей.

Вообще говоря, следует предоставить возможность управлять блокировками всех типов, включая блокировки, связанные с индексами, самой программе SQL Server. В обычных условиях эта серверная программа сама принимает качественные решения и наилучшим образом управляет своими собственными ресурсами.

Для управления вручную тем, какие типы блокировок могут быть разрешены на индексированной таблице, применяется системная процедура sp_indexoption. С помощью этой процедуры можно запретить использование блокировок строк и/или страниц, но автор не рекомендует выполнять такое действие в обычных обстоятельствах. Как и для подсказок запросов, обычно лучше разрешить серверу самому решать, какого типа блокировка должна быть установлена на том или ином ресурсе.

Следует отметить, что процедура sp_indexoption применяется только к индексам, поэтому с ее помощью нельзя управлять блокировкой страниц в динамической области памяти. Тем не менее, если таблица имеет кластеризованный индекс, то значения, заданные с помощью процедуры sp_indexoption, влияют и на такую таблицу.

Статистические данные

Читатель, по-видимому, уже знает о том, что производительность выполнения запросов в программе SQL Server связана с таким понятием, как "статистические данные". Статистические данные представляют собой метаданные, относящиеся к ключам индексов, и в качестве дополнительной опции — к значениям неиндексированных столбцов. Эти данные сопровождаются программой SQL Server. В программе SQL Server статистические данные используются для определения того, приведет ли применение индекса к ускоренному выполнению запроса. В сочетании с индексами статистические данные представляют собой один из наиболее важных источников информации, позволяющих оптимизатору разрабатывать оптимальные планы выполнения. Если статистические данные отсутствуют или

являются устаревшими, то способность оптимизатора формировать наилучшие планы выполнения запроса значительно уменьшается.

Прежде чем приступить к более подробному описанию статистических данных, рассмотрим несколько основных терминов, касающихся статистических данных.

Кардинальность

Кардинальностью данных называется количество уникальных значений в данных. Строго говоря, теория реляционных баз данных не допускает наличия дубликатов строк (кортежей) в отношении (таблице), поэтому теоретически кардинальность обозначает общее количество кортежей. Тем не менее в программе SQL Server допускается наличие дубликатов строк в таблице, поэтому будем считать, что термином "кардинальность" обозначается количество уникальных значений в наборе данных.

Плотность

Плотностью характеризует уникальность значений в наборе данных. Плотность индекса вычисляется путем деления количества строк, которые соответствуют данному конкретному значению ключа, на количество строк в таблице. Если индекс является уникальным, задача определения значения этого показателя сводится к выполнению операции деления числа 1 на общее количество строк в таблице. Значения плотности изменяются в пределах от 0 до 1; чем ниже показатель плотности, тем лучше.

Избирательность

Избирательность — это мера оценки количества строк, которые должны быть возвращены при использовании какого-то конкретного критерия запроса. Этот показатель выражает связь между критериями запроса и значениями ключей в индексе. Значение данного показателя вычисляется путем деления количества требуемых значений ключей на количество строк, к которым можно получить доступ с помощью этих ключей. Наиболее полезными для оптимизатора являются критерии запроса (обычно представленные в конструкции `WHERE`), которые характеризуются высокой избирательностью, поскольку такие критерии позволяют надежно предсказывать, какое количество операций ввода-вывода потребуется для выполнения запроса.

Проблемы производительности

Оптимизатор чаще всего игнорирует индексы, характеризующиеся высокими значениями плотности. Наиболее полезными для оптимизатора являются индексы, имеющие значения плотности от 0,10 и ниже. Рассмотрим в качестве примера таблицу `VoterRegistration` (список избирателей), которая включает 10 тыс. строк,

не имеет кластеризованного индекса, но имеет некластеризованный индекс на столбце PartyAffiliation (партия, сторонником которой является данный избиратель). Если в рассматриваемом избирательном округе зарегистрированы три политические партии и каждая из них имеет примерно одинаковое количество сторонников среди избирателей, то столбец PartyAffiliation будет, скорее всего, содержать только три уникальных значения. Таким образом, каждое конкретное значение ключа в индексе позволяет осуществить выборку из таблицы примерно 3333 строк, (возможно, и больше). Это означает, что индекс на столбце PartyAffiliation имеет плотность 0,33 (3333 ÷ 10 000), то есть фактически гарантирует, что оптимизатор не будет использовать индекс при формировании плана выполнения запросов, в которых требуются столбцы, не покрытые этим индексом.

Чтобы лучше разобраться в данной ситуации, сравним стоимости выполнения простого запроса с использованием и без использования такого индекса. Если бы нужно было получить список всех избирателей рассматриваемого округа, которые являются сторонниками демократической партии, то пришлось бы рассматривать ситуацию, в которой требуется осуществить выборку с помощью ключа примерно 3333 строк, т.е. третьей части таблицы. Если для доступа к этим строкам используется индекс PartyAffiliation, то выполняется примерно 3333 отдельных логических операций чтения страниц из основополагающей таблицы. Иными словами, после обнаружения каждого значения ключа в индексе необходимо найти закладку, которая указывает на основополагающую таблицу, чтобы получить значения столбцов, не содержащиеся в индексе, а при каждом выполнении такой операции приходится нести издержки, связанные с вводом-выводом логической (возможно, и физической) страницы. Это означает, что объем издержек при выполнении операций ввода-вывода страниц, связанных с поиском значений закладок, может достигать 26 Мбайт (3333 ключей * 8 Кбайт/страница). Теперь рассмотрим стоимость простого последовательного просмотра этой таблицы. Если на каждой странице данных помещается в среднем 50 строк, и необходимо прочитать всю таблицу, чтобы найти все строки, относящиеся к избирателям, поддерживающим демократическую партию, то придется в ходе выполнения ввода-вывода просмотреть всего лишь около 200 логических страниц (10 000 строк * 50 строк/страница = 200 страниц). Эта разница очень велика и является главной причиной того, что некластеризованные индексы с высокой плотностью игнорируются в пользу полного просмотра таблиц или просмотра с помощью кластеризованного индекса.

Теперь рассмотрим, при каких обстоятельствах некластеризованный индекс становится достаточно избирательным для того, чтобы оказаться полезным для оптимизатора. В данном примере критерием оценки применимости некластеризованного индекса является число 200. В частности, оптимизатор должен руководствоваться следующим прогнозом: с помощью индекса должна осуществляться такая выборка данных, при которой потребуется количество операций ввода-вывода страниц меньше 200. Только при этом условии применяемый индекс может рассматриваться как более эффективный путь доступа, чем простой просмотр всей таблицы. Первоначальную оценку в 3333 строк можно уменьшить, добавляя в индекс (а также в запрос) столбцы, позволяющие добиться большей избирательности. Однако попытка уменьшить таким образом количество выполняемых операций

имеет свою оборотную сторону. Введение дополнительных столбцов в индекс для того, чтобы этот индекс стал более избирательным, приводит к увеличению объема издержек, связанных с перемещением по B-дереву индекса. Если индекс занимает большой объем памяти, то велики и затраты, связанные с переходом по этому индексу. В какой-то момент обнаруживается, что менее дорогостоящим становится план, предусматривающий просто просмотр самих данных и не требующий издержек, обусловленных необходимостью продвижения в B-дереве.

Организация хранения статистических данных

В программе SQL Server статистические данные, относящиеся к ключу индекса или столбцу, хранятся в столбце `statblob` таблицы `sysindexes`. Столбец `statblob` имеет тип данных `image` и хранит гистограмму, содержащую результаты выборки значений в ключе индекса или столбце. В случае составных индексов выборка осуществляется только в первом столбце индекса, а для других столбцов сопровождаются значения плотности.

На этапе выбора индексов процедуры оптимизации запроса оптимизатор определяет, соответствует ли индекс столбцам, заданным в критериях выборки; устанавливает избирательность индекса применительно к указанным критериям и оценивает стоимость доступа к данным, требуемым в запросе.

Если индекс состоит только из одного столбца, то относящиеся к нему статистические данные включают одну гистограмму и одно значение плотности. Если же индекс состоит из нескольких столбцов, то для него сопровождается единственная гистограмма наряду со значениями плотности для каждой префиксной комбинации столбцов ключа (определяемой в порядке расположения столбцов слева направо). Оптимизатор использует указанное сочетание гистограммы индекса и его плотностей (т.е. статистические данные, относящиеся к индексу) при определении того, насколько полезным является индекс для выполнения данного конкретного запроса.

Тот факт, что предусмотрено хранение гистограммы только для первого столбца составного индекса, является одной из причин, по которым следует ставить на первое место в многостолбцовом индексе наиболее избирательный столбец; в таком случае гистограмма становится для оптимизатора более полезной. Кроме того, в этом также состоит одна из причин, по которой иногда рекомендуется разбивать составные индексы на несколько одностолбцовых индексов. Сервер может применять операции пересечения и соединения к нескольким индексам одной и той же таблицы, поэтому использование нескольких индексов позволяет сохранить преимущество, обусловленное наличием индексов на столбцах, и вместе с тем получить дополнительное преимущество, связанное с наличием гистограммы для каждого столбца (в этом может также помочь применение статистических данных столбцов). Это — не абсолютная рекомендация (поэтому не следует, прочитав данные строки, сразу же приступать к уничтожению всех существующих составных индексов). Достаточно просто учитывать, что иногда одной из применимых опций настройки производительности является разбиение составных индексов на отдельные индексы.

Статистические данные о столбцах

Программа SQL Server способна не только накапливать статистические данные об индексах, но и формировать статистические данные о неиндексированных столбцах. (Такая операция осуществляется автоматически при выполнении запроса к неиндексированному столбцу, если для базы данных разрешена опция AUTO_CREATE_STATISTICS.) Имея возможность определить вероятность того, что в столбце может встретиться какое-то конкретное значение, оптимизатор получает ценную информацию от том, как лучше всего выполнить тот или иной запрос. Наличие статистических данных позволяет оптимизатору оценивать количество строк, которые будут получены из данной конкретной таблицы, участвующей в соединении, что дает возможность более точно выбирать порядок соединения. Кроме того, оптимизатор может использовать статистические данные о столбцах для предоставления информации в форме гистограммы для других столбцов в многостолбцовом индексе. По сути, действует такой принцип – чем больше объем информации об обрабатываемых данных, тем лучше.

Структура статистических данных

В программе SQL Server статистические данные используются для контроля над распределением значений ключей в таблице. Гистограмма, хранящаяся в составе статистических данных об индексе, содержит в первом столбце ключа индекса выборку значений, состоящую не больше чем из 200 значений. Кроме гистограммы, столбец `statblob` содержит также перечисленные ниже данные.

- Количество строк, которые использовались для вычисления гистограммы и плотностей.
- Средняя длина ключа индекса.
- Дата и время выполнения последней операции формирования статистических данных.
- Значения плотности для других префиксных комбинаций столбцов ключа.

Диапазон значений ключа между каждым из 200 значений выборки гистограммы называется шагом. Каждое значение выборки обозначает конец шага, а в каждом шаге хранятся три описанных ниже значения.

1. EQ_ROWS. Количество строк со значением ключа, соответствующим значению выборки.
2. RANGE_ROWS. Количество других значений в этом диапазоне.
3. RANGE_DENSITY. Результаты вычисления плотности в самом этом диапазоне.

Команда DBCC SHOW_STATISTICS выводит в непосредственном виде данные EQ_ROWS и RANGE_ROWS, а также использует значение RANGE_DENSITY для вычисления показателей DISTINCT_RANGE_ROWS и AVG_RANGE_ROWS, относящихся к данному шагу. Это команда вычисляет значение DISTINCT_RANGE_ROWS (общее количество различных строк в диапазоне, относящемся к данному шагу) путем де-

ления 1 на RANGE_DENSITY, а также вычисляет значение AVG_RANGE_ROWS (среднее количество строк в расчете на каждое различимое значение ключа) путем умножения RANGE_ROWS на RANGE_DENSITY.

Обновление статистических данных

Для обновления статистических данных может применяться целый ряд способов. Первый и наиболее очевидный из этих способов состоит в использовании опции базы данных AUTO_UPDATE_STATISTICS. (Чтобы разрешить применение этой опции, можно воспользоваться оператором ALTER DATABASE, процедурой sp_dboption или программой Enterprise Manager.) При автоматическом формировании статистических данных для таблицы любого размера в программе SQL Server для ускорения работы используется формирование выборок (а не просмотр всей таблицы). Такой подход себя оправдывает в большинстве случаев, но иногда ведет к формированию статистических данных, которые являются менее полезными, чем могли бы быть.

С операцией автоматического обновления статистических данных тесно связана операция автоматического создания статистических данных. Эта операция выполняется, если разрешена опция базы данных AUTO_CREATE_STATISTICS и на обработку поступил запрос, предусматривающий выборку по определенному критерию значений в неиндексированном столбце (или запрос к столбцам, не относящимся к префиксу, выровненному слева направо по отношению к критериям выборки). В таком случае программа SQL Server автоматически создает набор статистических данных о столбцах.

Еще один метод обновления статистических данных состоит в использовании команды UPDATE STATISTICS. В версиях, предшествующих SQL Server 7.0, команда UPDATE STATISTICS представляла собой единственный способ обновления статистических данных. При выполнении команды UPDATE STATISTICS может либо использоваться формирование выборок, как при автоматическом обновлении, либо осуществляться полный просмотр таблицы, что приводит к получению более качественных статистических данных, но может потребовать больше времени.

Команда CREATE STATISTICS выполняет функции, аналогичные команде UPDATE STATISTICS. Команда CREATE STATISTICS может использоваться для создания статистических данных о столбцах вручную. Эти статистические данные после их создания могут обновляться с помощью средств автоматического обновления или с помощью команды UPDATE STATISTICS (по такому же принципу, как обновляются обычные статистические данные об индексах).

В программе SQL Server предусмотрено применение нескольких хранимых процедур, позволяющих упростить создание и обновление статистических данных. Процедура sp_updatestats обеспечивает выполнение команды UPDATE STATISTICS применительно ко всем определяемым пользователем таблицам в текущей базе данных. Но в отличие от самой команды UPDATE STATISTICS, процедура sp_updatestats не позволяет использовать полный просмотр таблицы для формирования статистических данных, поскольку в ней всегда осуществляется выборка

данных. Если есть необходимость сформировать статистические данные с помощью полного просмотра, следует воспользоваться командой UPDATE STATISTICS.

Такой же удобной может оказаться процедура sp_createstats. Эта процедура позволяет автоматизировать создание статистических данных о столбцах для всех применимых столбцов во всех приемлемых таблицах базы данных. К применимым столбцам относятся невычисленные столбцы с данными типов, которые отличаются от типов text, ntext или image и еще не имеют статистических данных о столбцах или статистических данных о первых столбцах индексов. К приемлемым таблицам относятся все пользовательские таблицы (таблицы, отличные от системных). Автор не рекомендует бездумно выполнять процедуру sp_createstats во всех эксплуатируемых базах данных, поскольку маловероятно, что в процессе работы потребуются статистические данные, относящиеся к каждому столбцу в таблице. Но в том случае, если такая необходимость действительно существует, процедура sp_createstats реально экономит время.

Процедура sp_autostats позволяет управлять автоматическим обновлением статистических данных на уровнях таблицы и индекса. Вместо простого ввода в действие опции базы данных AUTO_UPDATE_STATISTICS с помощью указанной процедуры можно разрешать или запрещать автоматическое формирование статистических данных на более тонко детализированном уровне. Например, если предусмотрено обновление в ночную смену статистических данных большой таблицы с использованием просмотра, то желательно было бы отменить автоматическое обновление статистических данных для такой таблицы. С помощью процедуры sp_autostats можно запретить автоматические обновления статистических данных для одной подобной таблицы и вместе с тем разрешить выполнение указанной операции для остальной части базы данных (можно также использовать команду UPDATE STATISTICS ... WITH NORECOMPUTE). Обновление статистических данных для больших таблиц, даже выполняемое с помощью выборки, может потребовать много времени, а также значительных ресурсов процессора и ресурсов ввода-вывода.

Следует учитывать, что отсутствие статистических данных или применение устаревших статистических данных оказывает отрицательное влияние на производительность, которое почти всегда перевешивает экономию ресурсов, достигнутую в связи с отказом от автоматического создания (обновления) статистических данных. Отменять автоматическое создание (обновление) статистических данных следует лишь в том случае, если всесторонняя проверка показала, что нет иного способа достичь требуемых показателей производительности или масштабируемости.

В листинге 12.9 приведена хранимая процедура, которая может использоваться для контроля над операциями обновления статистических данных. Эта процедура показывает тип статистических данных, время последнего обновления данных, а также много другой информации, которая может оказаться полезной при управлении статистическими данными об индексах и столбцах.

Листинг 12.9. Процедура sp_showstatdate, применяемая для контроля над операциями обновления статистических данных

```
CREATE PROC sp_showstatdate @tabmask sysname='%',
   @indmask sysname='%'
AS
SELECT
  LEFT(CAST(USER_NAME(uid)+'.'+o.name AS sysname),30) AS TableName,
  LEFT(i.name,30) AS IndexName,
  CASE WHEN INDEXPROPERTY(o.id,i.name,'IsAutoStatistics')=1
        THEN 'AutoStatistics'
      WHEN INDEXPROPERTY(o.id,i.name,'IsStatistics')=1
        THEN 'Statistics'
  ELSE 'Index'
  END AS Type,
  STATS_DATE(o.id, i.indid) AS StatsUpdated,
  rowcnt,
  rowmodctr,
  ISNULL(CAST(rowmodctr/CAST(NULLIF(rowcnt,0) AS decimal(20,2))*100
     AS int),0) AS PercentModifiedRows,
  CASE i.status & 0x1000000
  WHEN 0 THEN 'No'
  ELSE 'Yes'
  END AS [NoRecompute?],
  i.status
FROM dbo.sysobjects o JOIN dbo.sysindexes i ON (o.id = i.id)
WHERE o.name LIKE @tabmask
  AND i.name LIKE @indmask
  AND OBJECTPROPERTY(o.id,'IsUserTable')=1
  AND i.indid BETWEEN 1 AND 254
ORDER BY TableName, IndexName
GO
USE pubs
GO
EXEC sp_showstatdate

(Результаты приведены в сокращенном виде)

TableName          IndexName        Type        StatsUpdated
------------------ ---------------- ----------- -----------------------
dbo.authors        au_fname          Statistics 2000-07-02 19:42:04.487
dbo.authors        aunmind          Index       2000-06-30 20:54:56.737
dbo.authors        UPKCL_auidind    Index       2000-06-30 20:54:56.737
dbo.dtproperties   pk_dtproperties  Index       NULL
dbo.employee       employee_ind     Index       2000-06-30 20:54:45.280
dbo.employee       PK_emp_id        Index       2000-06-30 20:54:45.297
```

Оценка избирательности

Чтобы правильно определить относительную стоимость плана запроса, оптимизатор должен обладать способностью точно оценить количество строк, возвращаемых запросом. Как было указано выше, соответствующее значение называется избирательностью, причем крайне важно, чтобы оптимизатор был способен правильно определить это значение.

Для оценки избирательности производится сравнение критериев запроса со статистическими данными о ключе индекса или о столбце, на которые ссылаются эти критерии. Показатель избирательности позволяет определить, следует ли рассчитывать на получение одной строки или 10 тыс. строк после ввода данного конкретного значения ключа. Этот показатель позволяет получить представление о том, сколько строк соответствует искомому значению ключа или столбца. А информация о количестве строк, в свою очередь, позволяет установить, какой именно подход окажется наиболее эффективным для доступа к требуемым строкам. Очевидно, что для выборки одной строки следует применять иной метод, чем для выборки 10 тыс. строк.

Если оптимизатор обнаруживает, что в базе данных отсутствуют статистические данные об индексе или о столбце, относящиеся к одному из столбцов в критерии выборки запроса, то оптимизатор может автоматически создать статистические данные о столбце на уровне столбца, если разрешена опция базы данных AUTO_CREATE_STATISTICS. В первый момент в связи с этим придется понести затраты, связанные с формированием статистических данных, но благодаря наличию вновь полученных статистических данных последующие запросы должны выполняться быстрее.

Индексируемые выражения

В других книгах и материалах, посвященных программе SQL Server, часто можно встретить обсуждение темы об использовании индексов в планах запросов, в котором применяется термин SARG (Search ARGument in a query — параметр поиска в запросе). Кроме того, во многих информационных источниках говорится о том, что оптимизатор может преобразовать параметры SARG в операторы сравнения ключа индекса и некоторого значения. Тем не менее упоминание об указанном понятии в самом коде оптимизатора отсутствует. Термин SARG — это анахронизм, который остался от устаревшей кодовой базы Sybase и от выпущенных ранее учебных материалов Sybase. Оптимизатор во всех новейших версиях программы SQL Server является гораздо более развитым с точки зрения его способности использовать индексы для ускорения выполнения запросов, чем оптимизатор в старых версиях указанного программного продукта. Способность оптимизатора использовать индексы больше не ограничивается простыми выражениями, а само принимаемое оптимизатором решение о том, использовать или не использовать индекс, уже не обязательно зависит от того, какую форму может иметь выражение. Иными словами, может иметь или не иметь значение то, какое выражение фактически используется в предикате запроса, в зависимости от имеющихся индексов, способности оптимизатора выявить ссылки на столбцы в рассматриваемом выражении, а также от оценок стоимости, вырабатываемых оптимизатором. Термин SARG становится все менее и менее применимым. По мнению автора, его уже можно заменить более точным термином, который в большей степени соответствует определению выражений предиката запроса, позволяющих успешно использовать индексы. В качестве такого более правиль-

ного и точного термина (который, к тому же, используется в самом коде оптимизатора) можно назвать термин "индексируемое выражение" (так называется выражение, которое можно раскрыть с применением значений индекса для обеспечения более быстрого доступа к данным, возвращаемым запросом). Поэтому в настоящей книге используется термин "индексируемое выражение".

Определим индексируемое выражение как конструкцию в запросе, которая в принципе может использоваться оптимизатором в сочетании с индексом для регламентации объема результатов, возвращаемых запросом. Оптимизатор предпринимает попытки выявить индексируемые выражения в критериях запроса, чтобы иметь возможность определить наилучшие индексы, которые можно было бы применить для обслуживания запроса.

Вообще говоря, индексируемые выражения имеют следующую форму (крайние члены этого выражения могут быть указаны в обратном порядке):

```
Column op Constant/Variable
```

Здесь `Column` — столбец таблицы; `op` — один из следующих операторов: =, >=, <=, >, <, <>, !=, !>, !<, BETWEEN и LIKE (одни конструкции LIKE могут быть преобразованы в индексируемые выражения, а другие — нет); `Constant/Variable` — это значение константы, ссылка на переменную или одна из многочисленных функций.

Преобразование операторов

Некоторые из операторов, упомянутых выше, не могут непосредственно использоваться в поиске по индексу, но оптимизатор может преобразовать их в выражения, применимые для указанной цели. Например, рассмотрим следующий запрос:

```
SELECT * FROM authors
WHERE au_lname != 'Greene'
```

Является ли конструкция WHERE в этом запросе индексируемой? Ответ на этот вопрос положителен. Рассмотрим, какое при этом осуществляется преобразование, в следующей выдержке из плана запроса:

```
SEEK:([authors].[au_lname] < 'Greene' OR
[authors].[au_lname] > 'Greene')
```

Оптимизатор обладает достаточными возможностями, чтобы определить, что выражение `x != @parm` представляет собой то же, что и выражение `x < @parm OR x > @parm`, и преобразовывает первое выражение соответствующим образом. Два компонента конструкции OR можно выполнить параллельно и осуществить слияние полученных данных, поэтому применение указанного преобразования обеспечивает использование индекса для обслуживания приведенного выше критерия WHERE.

Теперь рассмотрим аналогичный запрос, в котором применяется выражение LIKE:

```
SELECT * FROM authors
WHERE au_lname LIKE 'Gr%'
```

Является ли это выражение индексируемым? И на этот вопрос можно ответить утвердительно. Оптимизатор и в данном случае преобразует критерии конструкции WHERE в такой план запроса, который считается более приемлемым, как показывает приведенная ниже выдержка из плана запроса.

```
SEEK:([authors].[au_lname] >= 'GQ' AND
[authors].[au_lname] < 'GS')
```

Рассмотрим еще один запрос:

```
SELECT * FROM authors
WHERE au_lname !> 'Greene'
```

Может ли оптимизатор использовать индекс для выполнения данного запроса? Да, такая возможность существует. Ниже приведена выдержка из плана запроса.

```
SEEK:([authors].[au_lname] <= 'Greene')
```

Рассмотрим еще один запрос:

```
SELECT * FROM authors
WHERE au_lname !< 'Greene'
```

А ниже приведен соответствующий фрагмент плана запроса.

```
SEEK:([authors].[au_lname] >= 'Greene')
```

Внимательный читатель должен заметить то, что связывает между собой эти примеры. Оптимизатор в каждом случае предпринимает попытки преобразовать выражения, которые внешне кажутся неиндексируемыми, в такие выражения, которые можно более легко обслуживать с помощью индексов.

Индексируемые выражения можно соединять друг с другом с помощью оператора AND для формирования составных конструкций. Эмпирическое правило, позволяющее определить, является ли выражение индексируемым, состоит в следующем: некоторая конструкция может обслуживаться с помощью индекса, если оптимизатор способен определить, что эта конструкция представляет собой сравнение значения ключа индекса с константой или переменной. Распространенная ошибка, которую допускают неопытные разработчики, состоит в том, что в выражение, применяемое для сравнения с константой или переменной, включают столбец. Чаще всего такой подход исключает возможность обслуживать полученную конструкцию с помощью индекса, поскольку оптимизатор не может определить, какое значение фактически приобретет данное выражение в результате вычисления (ведь это невозможно определить до этапа прогона). (Из этого правила есть исключение; ниже приведено описание подхода, позволяющего предоставить оптимизатору возможность использовать результаты свертывания выражений.) Поэтому разработчик должен представлять столбцы в выражениях подобных типов отдельно, чтобы эти столбцы не были никоим образом модифицированы или инкапсулированы. Разработчику достаточно применить простейшие алгебраические преобразования, чтобы переместить модификаторы в подготавливаемой конструкции в ту часть выражения, которая соответствует значению/константе, и оставить сам столбец немодифицированным.

Если оптимизатор не имеет возможности идентифицировать выражение как индексируемое, то не может также использовать статистические данные для оценки количества строк, возвращаемых соответствующим оператором. В таком случае оптимизатор должен руководствоваться определенными предположениями. При этом в качестве оценок применяются жестко заданные, так называемые "магические" значения, которые определены заранее для различных типов операторов сравнения. Общие сведения о "магических" числах, используемых оптимизатором для оценки некоторых операторов, приведены в табл. 12.3.

Таблица 12.3. Оценки количества строк для неиндексируемых выражений

Оператор сравнения	Оцениваемое процентное значение количества строк
=	10
>	30
<	30
BETWEEN	10

Свертывание

В некоторых редких обстоятельствах оптимизатор обладает способностью должным образом выявить индексируемое выражение, даже если столбец в этом выражении заключен в функцию или каким-то иным образом вошел в состав подвыражения. При этом применяется подход, известный под названием *свертывания* выражений, представляющий собой одно из усовершенствований по сравнению с более старыми версиями программы SQL Server, в которых оптимизатор не обладал способностью использовать индекс для обслуживания конструкции запроса, если столбец таблицы участвовал в выражении или скрывался в какой-то функции. Свертывание выражений позволяет оптимизатору выявлять некоторые типы индексируемых выражений, даже если столбец в таком выражении участвует в подвыражении или скрывается в функции. К числу свертываемых выражений относятся некоторые выражения, созданные с использованием функций DATEADD, ISNULL и ROUND, некоторые формы предиката LIKE, а также несколько других выражений. Например, рассмотрим следующий запрос:

```
SELECT *
FROM Orders
WHERE ISNULL(OrderDate,GETDATE())>'2003-04-06 19:55:00.000'
```

Оптимизатор, не способный свертывать выражение, в котором участвует столбец OrderDate и функция ISNULL, не имел бы возможности использовать индекс OrderDate таблицы Orders (ключом которой является столбец OrderDate) для обслуживания этого запроса. Тем не менее, ознакомившись с графическим изображением плана для данного запроса в программе Query Analyzer, можно убедиться в том, что индекс фактически используется.

В подобных случаях следует руководствоваться таким эмпирическим правилом: стараться при любой возможности избегать включения столбцов в подвыражения применяемых выражений. Тем не менее следует знать, что в некоторых ситуациях оптимизатор все равно обладает способностью выявлять индексируемые выражения.

Порядок соединений и выбор типа

Оптимизатор не только выбирает индексы и определяет индексируемые выражения, но и устанавливает порядок соединений, а также находит стратегию соединений для операторов, в которых требуются соединения. Задачи выбора

индексов и поиска стратегии соединения тесно связаны друг с другом — от выбора индексов зависит определение приемлемых типов стратегий соединения, а от выбранной стратегии соединения зависят типы индексов, которые потребуются оптимизатору для выработки эффективного плана.

В программе SQL Server поддерживаются три описанных ниже типа соединений.

1. Соединение с помощью вложенных циклов действует успешно, если внешняя таблица меньше внутренней, а для внутренней таблицы задан индекс.
2. Соединение слиянием действует успешно, если обе входные таблицы отсортированы по соединяющему столбцу (оптимизатор может в случае необходимости предусмотреть сортировку одной из входных таблиц).
3. Хэшированные соединения выполняются успешно в таких ситуациях, при которых отсутствуют применимые индексы. Но обычно достижение более высокой производительности обеспечивается с помощью создания индекса (что позволяет использовать другую стратегию соединения).

Оптимизатор определяет стратегию соединения, которая должна применяться для обслуживания запроса. При этом оптимизатор оценивает стоимость каждой стратегии и выбирает ту из них, которая согласно прогнозам должна быть наименее дорогостоящей. Оптимизатор оставляет за собой право переупорядочивать таблицы в конструкции FROM, если это позволяет повысить производительность запроса. Пользователь всегда может определить, применяется ли переупорядочение, ознакомившись с планом выполнения. Порядок таблиц в плане выполнения — это именно тот порядок, который согласно прогнозу оптимизатора должен обеспечить максимальную производительность.

Оптимизатор можно вынудить отказаться от использования его способности определять порядок соединения, применяя в запросе конструкцию OPTION (FORCE ORDER), опцию сеанса SET FORCEPLAN ON и подсказки, касающиеся порядка соединения (например, INNER LOOP JOIN). В случае использования каждого из этих вариантов оптимизатор вынужден соединять таблицы в том порядке, который задан в конструкции FROM.

Следует отметить, что подход, в котором предусмотрено принудительное задание порядка соединения, может повлечь за собой побочный эффект, связанный с тем, что будет также принудительно задана какая-то конкретная стратегия соединения. Например, рассмотрим запрос и соответствующий ему план выполнения, которые показаны в листинге 12.10.

Листинг 12.10. Запрос, в котором применяется правое внешнее соединение, и план выполнения этого запроса

```
(Запрос)

SELECT o.OrderId, p.ProductId
FROM [Order Details] o RIGHT JOIN Products p
ON (o.ProductId=p.ProductId)

(План выполнения)

StmtText
```

```
------------------------------------------------------------------------------
SELECT o.OrderId, p.ProductId
FROM [Order Details] o RIGHT JOIN Products p
ON (o.ProductId=p.ProductId)
  |--Nested Loops(Left Outer Join, OUTER REFERENCES:(p.ProductID))
       |--Index Scan(OBJECT:(Northwind.dbo.Products.SuppliersProducts AS p))
       |--Index Seek(OBJECT:(Northwind.dbo.[Order Details].ProductID AS o),
          SEEK:(o.ProductID=p.ProductID) ORDERED FORWARD)
```

Здесь заслуживает внимания то, что оптимизатор использует соединение с помощью вложенных циклов и переупорядочивает таблицы (на первом месте в плане указана таблица `Products`, даже несмотря на то, что в конструкции FROM первой указана таблица `Order Details`). Теперь зададим принудительно порядок соединения с применением подсказки запроса FORCE ORDER и посмотрим, как изменится план запроса (листинг 12.11).

Листинг 12.11. Принудительное задание порядка соединения с помощью подсказки FORCE ORDER

```
(Запрос)

SELECT o.OrderId, p.ProductId
FROM [Order Details] o RIGHT JOIN Products p ON
   (o.ProductId=p.ProductId)
OPTION(FORCE ORDER)

(План запроса)
StmtText
------------------------------------------------------------------------------
SELECT o.OrderId, p.ProductId
FROM [Order Details] o RIGHT JOIN Products p ON (o.ProductId=p.ProductId)
OPTION(FORCE ORDER)
  |--Merge Join(Right Outer Join, MANY-TO-MANY MERGE:(o.ProductID)=
    (p.ProductID), RESIDUAL:(o.ProductID=p.ProductID))
       |--Index Scan(OBJECT:(Northwind.dbo.[Order Details].
          ProductsOrder_Details AS o), ORDERED FORWARD)
       |--Clustered Index Scan(OBJECT:(Northwind.dbo.Products.
          PK_Products AS p), ORDERED FORWARD)
```

Оптимизатор в данном случае не имеет возможности переупорядочить таблицы, поэтому переключается на стратегию соединения слиянием. Такой подход является менее эффективным по сравнению с тем подходом, в котором оптимизатору разрешено упорядочивать таблицы так, как он считает нужным, и соединять таблицы с помощью вложенных циклов.

Соединения с помощью вложенных циклов

Соединения с помощью вложенных циклов состоят из цикла, вложенного в другой цикл. В соединении с помощью вложенных циклов одна таблица в соединении обозначается как таблица внешнего цикла (или внешняя таблица), а другая — как таблица внутреннего цикла (или внутренняя таблица). В каждой итерации внешнего цикла просматривается вся таблица внутреннего цикла. Такой способ соединения вполне подходит для таблиц с размерами от небольших до

средних, но по мере увеличения таблиц, обрабатываемых в циклах, данная стратегия становится все менее эффективной. Общая процедура выполнения соединения описана ниже.

1. Найти строку в первой таблице.
2. Использовать значение из этой строки для поиска строки во второй таблице.
3. Повторять указанные действия до тех пор, пока в первой таблице не останется больше строк, которые соответствовали бы критериям поиска.

Оптимизатор оценивает по меньшей мере четыре комбинации соединений, даже если эти комбинации не заданы в предикате соединения. При этом оптимизатор учитывает, что стоимость оценки дополнительных комбинаций не должна превосходить общую стоимость выработки плана запроса.

Соединения с помощью вложенных циклов обладают гораздо более высокой производительностью по сравнению с соединениями слиянием и хэшированными соединениями, при условии, что объемы данных, применяемых в соединении, находятся в пределах от небольших до средних. Оптимизатор запросов использует соединения с помощью вложенных циклов, если внешняя входная таблица весьма мала, а внутренняя входная таблица индексирована и весьма велика. Оптимизатор переупорядочивает таблицы так, чтобы меньшая входная таблица стала внешней таблицей, а также требует введения применимого индекса на внутренней таблице. Кроме того, оптимизатор всегда использует стратегию вложенных циклов при выполнении тета-соединений (соединений по критерию, отличному от критерия равенства).

Соединения слиянием

Применительно к большим наборам данных соединения слиянием являются гораздо более эффективными по сравнению с соединениями с помощью вложенных циклов. Для успешного использования такого соединения обе таблицы должны быть отсортированы по столбцу, применяемому для слияния. Оптимизатор обычно выбирает соединение слиянием при работе с большими наборами данных, которые уже отсортированы по столбцам соединения. Оптимизатор может использовать деревья индексов для подготовки отсортированных входных таблиц, а также может предусмотреть применение операций сортировки с помощью конструкций `GROUP BY`, `CUBE` и `ORDER BY`; при этом достаточно выполнить сортировку только один раз. Если одна из входных таблиц еще не отсортирована, то оптимизатор может принять решение о ее предварительной сортировке, чтобы иметь возможность выполнить соединение слиянием (если по прогнозу оптимизатора соединение слиянием будет более эффективным, чем соединение с помощью вложенных циклов). Такая ситуация возникает очень редко и характеризуется наличием оператора `Sort` в плане запроса.

Соединение слиянием сводится к выполнению пяти описанных ниже этапов.

1. Получить первые входные значения из каждой таблицы.
2. Сравнить их.

3. Если эти значения равны, вернуть соответствующие им строки.
4. Если эти значения не равны, отбросить меньшее значение и использовать для очередного сравнения следующее входное значение из этой таблицы.
5. Повторять указанные действия до тех пор, пока не будут обработаны все строки одной из таблиц.

Оптимизатор выполняет в каждой таблице только по одному проходу. Операция соединения прекращается после обработки всех входных значений одной из таблиц. Все значения в другой таблице, оставшиеся необработанными, не обрабатываются.

Оптимизатор может выполнить операцию соединения слиянием для реляционного оператора соединения любого типа, за исключением `CROSS JOIN` и `FULL JOIN`. Для объединения таблиц друг с другом с помощью оператора `UNION` могут также использоваться операции слияния (поскольку таблицы должны быть отсортированы для устранения дублирующихся строк).

Хэшированные соединения

Хэшированные соединения также являются более эффективными по сравнению с соединениями с помощью вложенных циклов, если обрабатываемые наборы данных велики. Кроме того, хэшированные соединения успешно применяются для обработки таблиц, которые не отсортированы по столбцу (столбцам) соединения. Оптимизатор обычно выбирает хэшированное соединение, если сталкивается с большими входными таблицами и отсутствует индекс, с помощью которого можно было бы выполнить соединение этих таблиц, или индекс существует, но является неприменимым.

В программе SQL Server хэшированные соединения осуществляются путем хэширования строк меньшей из двух таблиц (*формирующей*) и вставки полученных хэшированных значений в хэш-таблицу с последующей обработкой более крупной таблицы (*проверочной*). Обработка осуществляется последовательно, по одной строке, при этом выполняется поиск совпадений в хэш-таблице. Источником значений в хэш-таблице является меньшая из двух таблиц, поэтому хэш-таблица имеет минимально возможные размеры. Кроме того, используются хэшированные, а не реальные значения, поэтому операции сравнения значений двух таблиц выполняются очень быстро.

Хэшированные соединения основаны на использовании понятия хэшированных индексов, которые широко применяются во многих развитых продуктах СУБД в течение длительного времени. В хэшированных индексах предусмотрено постоянное хранение хэш-таблицы, поскольку индексом является именно эта таблица. Хэшированные данные распределяются по слотам, которые имеют одинаковое хэшированное значение. Если индекс имеет уникальный непрерывный ключ, то существует функция хэширования, известная под названием *минимальной идеальной* хэш-функции; при использовании такой функции каждое хэшированное значение попадает в свой собственный слот, а пустые промежутки между слотами в индексе отсутствуют. Если индекс является уникальным, но не непрерывным,

может существовать функция хэширования, находящаяся на втором месте после наилучшей функции хэширования — так называемая идеальная хэш-функция; при использовании этой функции каждое хэшированное значение попадает в свой собственный слот, но между слотами могут возникать пустые промежутки.

Если входные таблицы выбраны в качестве формирующей и проверяемой таблиц неправильно (например, из-за того, что оценки плотности оказались неточными), оптимизатор меняет эти таблицы местами динамически с использованием процедуры, называемой *обменом ролями*.

Операции хэшированного соединения позволяют обслуживать реляционные операции соединения любого типа (включая операции UNION и DIFFERENCE), за исключением операций CROSS JOIN. Хэширование может также использоваться для группирования данных и удаления дублирующихся строк (например, в векторных операциях агрегирования, таких как SUM(Quantity) GROUP BY ProductId). Если в оптимизаторе хэширование осуществляется в такой форме, то в качестве формирующей и проверяемой таблиц используется одна и та же таблица.

Если входные таблицы в операции соединения велики и имеют примерно одинаковые размеры, то производительность хэшированного соединения сравнима с производительностью соединения слиянием. Если же входные таблицы в операции соединения велики, но значительно отличаются по размерам, то хэшированные соединения обычно заметно превосходят по своей производительности соединения слиянием.

Подзапросы и альтернативные варианты соединений

Подзапросом называется запрос, вложенный в более крупный запрос. Обычно подзапрос служит для получения нескольких значений в операторах предикатов (таких как IN, ANY и EXISTS) или одного значения, которое используется для присваивания столбцу или переменной. Подзапросы могут применяться во многих местах, включая конструкции WHERE и HAVING запроса.

Следует учитывать, что операции соединения не являются во всех обстоятельствах более удобными, чем подзапросы. Оптимизатор часто выбирает соединение для выполнения подзапроса, но это не означает, что применение подзапроса в коде — это неэффективное решение.

Пытаясь усовершенствовать запрос, чтобы исключить издержки выполнения соединений, влияющие на производительность, следует помнить, что всегда можно воспользоваться переменными таблицы и временными таблицами для сохранения рабочих данных, подлежащих дальнейшей обработке. При создании чрезвычайно сложных запросов такой вариант может оказаться наилучшим, поскольку он позволяет пользователю получить больший контроль над кодом оптимизации. Запрос можно разбить на несколько шагов и взять на себя контроль над тем, что выполняется на каждом шаге.

Для запросов, принадлежащих к категории от простых до умеренно сложных, производные таблицы позволяют получить аналогичные преимущества, поскольк-

ку с их помощью можно в определенной степени улучшить порядок обработки запросов. Производные таблицы могут служить в ходе вычисления выражения в качестве круглых скобок (ведь выражения с определениями производных таблиц действительно обозначаются с помощью круглых скобок). Дело в том, что производные таблицы позволяют установить определенный порядок вычислений. Если часть сложной конструкции SELECT передается в производную таблицу, к которой затем применяется оставшаяся часть конструкции SELECT, это равносильно такому указанию, передаваемому оптимизатору: "Вначале необходимо выполнить эти действия, затем передать полученные результаты во внешнюю конструкцию SELECT". При этом оптимизатор не обязан соблюдать указанный порядок действий; он может переупорядочивать таблицы в выражениях производной таблицы и даже отказываться от вложенных конструкций в пользу более эффективного плана, но по крайней мере подзапросы предоставляют синтаксический механизм, позволяющий разработчику указывать предпочитаемый им порядок таблиц. Автор рекомендует опробовать этот метод в характерных ситуациях для определения того, позволяет ли он добиться требуемой производительности.

Логические и физические операторы

Логические операторы описывают реляционные операции, используемые для обработки запроса. В целях последующего выполнения логические операторы должны быть преобразованы в физические. Физические операторы описывают действия, которые должна осуществить программа SQL Server, чтобы выполнить работу, затребованную в запросе (например, связанную с выборкой данных). Один логический оператор часто преобразуется в несколько физических операторов. Планы выполнения состоят из физических операторов, поскольку компонент LPE сервера задает именно эти операторы, когда возникает необходимость выполнить работу, обусловленную в запросе. Компонент LPE преобразует такие физические операторы в вызовы OLE DB, передаваемые из реляционной машины в машину хранения для выполнения работы, затребованной в запросе.

Каждый шаг в плане выполнения соответствует одному физическому оператору. Как уже было сказано выше, планы выполнения состоят из последовательностей физических операторов. В графическом изображении плана выполнения программы Query Analyzer эти операторы отображаются в области заголовка желтых всплывающих окон с подсказками. Если на каком-то этапе, кроме физического, показан логический оператор, последний также отображается в области заголовка окна справа от физического оператора и отделяется от него косой чертой. В текстовом отображении плана выполнения физический оператор находится в столбце `PhysicalOp`, а логический оператор, соответствующий данному шагу, хранится в столбце `LogicalOp`.

Описанное выше проще всего продемонстрировать на примере. Рассмотрим следующий запрос, для которого требуется реляционное внутреннее соединение:

```
SELECT *
FROM Orders o JOIN [Order Details] d ON (o.OrderID = d.OrderID)
```

Оптимизатор выбирает для выполнения этого запроса соединение слиянием, несмотря на то, что в самом запросе, безусловно, не указано, что должна быть выполнена такая операция соединения. Но с точки зрения реляционной теории данный запрос должен быть выполнен с помощью внутреннего соединения двух таблиц. Ниже приведена выдержка из текстового представления плана выполнения этого запроса.

```
PhysicalOp            LogicalOp             Argument
--------------------  --------------------  ------------------------------

Merge Join            Inner Join            MERGE:([o].[OrderID])=([d].[Or
Clustered Index Scan  Clustered Index Scan  OBJECT:([Northwind].[dbo].[Ord
Clustered Index Scan  Clustered Index Scan  OBJECT:([Northwind].[dbo].[Ord
```

Обратите внимание на то, что в столбце PhysicalOp в качестве оператора указан оператор соединения слиянием Merge Join. Именно эта операция применяется незаметно для пользователя при обслуживании операции, показанной в столбце LogicalOp, т.е. операции внутреннего соединения Inner Join. Дело в том, что в запросе содержалось требование выполнить внутреннее соединение, а сервер выбрал в качестве физического оператора для выполнения этого требования оператор Merge Join.

Оптимизатор не только определяет, какой индекс должен использоваться и какая стратегия соединения должны быть принята, но и вырабатывает дополнительные решения, касающиеся операций других типов. Некоторые из таких операции описаны ниже.

Конструкция DISTINCT

Обнаружив в запросе ключевое слово DISTINCT или UNION, оптимизатор должен удалить дублирующиеся строки, полученные из входных таблиц, прежде чем возвратить результирующий набор. Для этого может применяться целый ряд вариантов: оптимизатор может отсортировать данные для удаления дублирующихся строк или осуществить их хэширование. Кроме того, оптимизатор может обеспечить обслуживание логической операции DISTINCT или UNION с использованием физического оператора хэширования или сортировки. (В данном случае также часто применяется оператор Stream Aggregate — аналогичный физический оператор, широко используемый при обработке запросов GROUP BY.)

Конструкция GROUP BY

Оптимизатор может обслуживать запросы, включающие конструкцию GROUP BY, с помощью простых операций сортировки или хэширования. И в этом случае в качестве физического оператора может применяться Hash или Sort, а в качестве логического оператора по-прежнему используется Aggregate или какой-то аналогичный оператор. Кроме того, как уже было сказано, для выполнения операций GROUP BY часто применяется оператор Stream Aggregate.

Оптимизатор может выбрать для группирования данных вариант, в котором осуществляется операция хэширования, поэтому полученный результирующий

набор может не находиться в отсортированном порядке. Это означает, что при использовании конструкции GROUP BY не следует рассчитывать на получение автоматически отсортированных данных. Если требуется, чтобы результирующий набор был отсортирован, необходимо включить в запрос конструкцию ORDER BY.

Конструкция ORDER BY

Оптимизатору приходится вырабатывать определенные решения даже применительно к конструкции ORDER BY. Допустим, существует кластеризованный индекс, в котором уже представлены отсортированные данные; в таком случае оптимизатор должен определить приемлемый способ представления результирующего набора в требуемом порядке. При этом оптимизатор может применить сортировку данных, на что вполне может рассчитывать пользователь, или же предусмотреть просмотр листового уровня некластеризованного индекса, созданного с помощью соответствующих ключей. Решение по выбору того или иного варианта, принятое оптимизатором, зависит от множества факторов. Наиболее важным из них является избирательность — количество строк, возвращаемых запросом. Кроме того, важен показатель покрытия индекса, при оценке которого оптимизатор определяет, может ли некластеризованный индекс покрыть данный запрос. Если количество строк относительно невелико, то вариант, предусматривающий использование некластеризованного индекса, может оказаться менее дорогостоящим по сравнению с сортировкой всей таблицы. Аналогичным образом, если индекс может покрыть данный запрос, то соответствующий вариант находится на втором месте по своему качеству, в сравнении с вариантом, предусматривающим применение еще одного кластеризованного индекса, поэтому оптимизатор, по всей вероятности, будет использовать покрывающий индекс.

Накопление промежуточных результатов

Наличие оператора Spooling в плане запроса свидетельствует о том, что оптимизатор предусмотрел сохранение результатов промежуточного запроса во вспомогательной таблице для дальнейшей обработки. При использовании метода отложенного накопления промежуточных результатов рабочая таблица заполняется по мере необходимости, а при использовании метода опережающего накопления промежуточных результатов таблица заполняется в одном шаге. Оптимизатор предпочитает метод отложенного, а не опережающего накопления промежуточных результатов, поскольку существует вероятность того, что удастся избежать необходимости полного заполнения рабочей таблицы благодаря тому, что логические операторы, глубже заложенные в плане запроса, позволят завершить работу с использованием не полностью реализованных планов. Тем не менее в некоторых случаях необходимо применять метод опережающего накопления промежуточных результатов (например, для защиты от проблемы порочного круга, см. главу 14), но обычно оптимизатор предпочитает метод отложенного накопления промежуточных результатов, поскольку последний характеризуется меньшими издержками.

Операции накопления промежуточных результатов могут применяться не только к таблицам, но и к индексам. Если в оптимизаторе требуется определить, существует ли какая-то конкретная строка, для этой цели используются промежуточные результаты выполнения команды `rowcount`.

Резюме

Процессор запросов программы SQL Server использует индексы, статистические данные и поступившие запросы в качестве входных данных и вырабатывает оптимизированные планы выполнения, которые после этого осуществляются сервером для доступа и возврата затребованных данных. Сервер может применять кластеризованные и некластеризованные индексы, а также статистические данные, формируемые автоматически и вручную. Выполняя каждый конкретный план запроса, сервер может использовать несколько индексов в расчете на каждую таблицу и применять к индексам операции пересечения и соединения.

Процессор запросов выбирает план с наименьшей стоимостью. Обычно в качестве критерия определения стоимости применяется оценка объема ввода-вывода, относящаяся к рассматриваемому плану, но учитываются также другие факторы, определяющие стоимость. Оценка объема ввода-вывода в расчете на каждый шаг плана обычно вычисляется на основе учета прогнозируемого количества строк, возвращаемых на данном этапе в каждом цикле выполнения, и исходит из прогнозируемого количества итераций цикла выполнения. Расхождения между прогнозируемым и фактическим количеством строк и количеством итераций выполнения могут указывать на неточности в прогнозах, которые обычно обусловлены применением устаревших статистических данных или статистических данных, основанных на выборках с интервалом выборки, слишком низким для того, чтобы можно было точно представить его базовые данные.

Ключом к эффективной обработке запросов является предоставление оптимизатору достаточного объема информации для принятия обоснованных решений. Важнейшими факторами достижения высокой производительности запросов являются индексы, статистические данные, индексируемые выражения, ограничения и повторно применяемые планы запросов.

В программе SQL Server предусмотрено несколько инструментальных средств, позволяющих упростить настройку запросов. Великолепным инструментальным средством, позволяющим определить, являются ли оптимальными применяемые индексы и статистические данные, является программа Index Tuning Wizard. Для обработки в эту программу можно направить поток результатов трассировки или отдельный запрос, после чего формируются рекомендации по внесению изменений в организацию индексов и статистических данных, которые позволили бы добиться повышения быстродействия выполняемого кода. Инструментальное средство Profiler программы SQL Server (как и программа Perfmon) позволяет ознакомиться с информацией о количестве повторных компиляций процедур и об использовании кэша. Графическое отображение плана выполнения в программе Query Analyzer дает возможность ознакомиться с конкретной информацией об оценках, принятых в плане, и о входных таблицах, применяемых на каждом этапе.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Если на таблице создается кластеризованный индекс с групповым ключом, то программа SQL Server автоматически "делает его уникальным".
2. Если на столбце `col` существует индекс, может ли его использовать оптимизатор для обслуживания предиката конструкции `WHERE col <> 1`?
3. Объясните, в чем состоит различие между избирательностью и кардинальностью.
4. Для чего предназначен оператор поиска закладки?
5. Подтвердите или опровергните следующее утверждение. При создании индекса определяемое для него значение коэффициента заполнения влияет на количество строк, хранящихся на каждой странице в B-дереве индекса, кроме корневой страницы.
6. Чему равно максимальное количество шагов в гистограмме статистических данных индекса?
7. В каком столбце таблицы `sysindexes` хранятся статистические данные индекса?
8. Какая команда `DBCC` может использоваться для вывода на внешнее устройство данных о фрагментации, относящихся к некоторому индексу?
9. Какая хранимая процедура может служить для указания типов блокировок, которые должны использоваться программой SQL Server для конкретного индекса?
10. Какая команда `DBCC` может использоваться для вывода на внешнее устройство данных гистограммы статистических данных, относящейся к некоторому индексу?
11. Что означает термин "покрывающий индекс"?
12. Какой параметр процедуры `sp_configure` определяет, будет ли оптимизатор запроса предпринимать попытки создать параллельный план выполнения на серверном многопроцессорном компьютере?
13. Подтвердите или опровергните следующее утверждение. Вычисленный столбец, который является недетерминированным, не может быть индексирован.
14. Объясните, что означает термин "плотность", применяемый в контексте оптимизации запросов.
15. Какое заключение, касающееся индексов рассматриваемой таблицы, можно сразу же сделать, обнаружив упоминание блокировки RID в выводе команды `sp_lock`?
16. Если на столбце `col` существует индекс, может ли этот индекс использоваться оптимизатором для обслуживания предиката `ISNULL(col,'')='foo'` конструкции `WHERE`?

ГЛАВА 7

Транзакции

В этой главе автор приводит обновленное описание транзакций SQL Server, которое впервые было приведено в его книге *The Guru's Guide to Transact-SQL*.

Средства управления транзакциями программы SQL Server позволяют гарантировать целостность и восстановимость данных, хранящихся в базе данных. *Транзакция* – это набор из одной или нескольких операций базы данных, которые рассматриваются как единый блок (осуществляются либо все эти операции, либо ни одна из них). Транзакция является основным критерием функциональной пригодности базы данных и фундаментальной единицей организации ее работы.

Транзакции SQL Server гарантируют восстановимость и непротиворечивость данных независимо от любых сбоев аппаратных средств и операционной системы, а также любых ошибок в приложении или в программе SQL Server, которые только могут произойти. Кроме того, транзакции гарантируют, что многочисленные команды, входящие в состав транзакций, будут либо выполнены полностью, либо совсем не выполнены, а отдельная команда, которая должна внести изменения в несколько строк, модифицирует либо все эти строки, либо ни одну из них.

Проверка соблюдения свойств ACID

Транзакции SQL Server часто характеризуют как "имеющие свойства ACID" или "успешно проходящие проверку на наличие свойств ACID", где ACID сокращенно обозначает свойства неразрывной (atomic), непротиворечивой (consistent), изолированной (isolated) и устойчивой (durable) транзакции. В современных СУБД обычной характеристикой является соблюдение в транзакциях принципов поддержки свойств ACID, которые служат предпосылкой обеспечения безопасности и надежности данных.

Неразрывность

Транзакция является неразрывной, если выполняется по принципу "все или ничего". В случае успешного выполнения транзакции все внесенные изменения становятся постоянно хранимыми, а в случае неудачного ее завершения все внесенные изменения полностью отменяются. Поэтому, например, если транзакция включает десять команд DELETE и последняя команда оканчивается неудачей, то при откате транзакции отменяются девять ранее выполненных команд. Аналогично, если в одной команде предпринимается попытка удалить десять строк, но удаление одной строки завершается неудачей, то неудачной считается вся операция.

Непротиворечивость

Транзакция является непротиворечивой, если гарантирует, что обрабатываемые данные никогда не окажутся в переходном или несогласованном состоянии, иными словами, если обрабатываемые данные никогда не перейдут в противоречивое состояние. Таким образом, данные, обрабатываемые в команде UPDATE, которая модифицирует десять строк, не должны становиться доступными для внешнего мира в каком-либо промежуточном состоянии — доступ ко всем строкам должен предоставляться либо тогда, когда эти строки находятся в начальном состоянии, либо после их перехода в конечное состояние. Это свойство исключает такую ситуацию, при которой один пользователь непреднамеренно вмешивается в незаконченную работу другого, которая еще не представлена в окончательном виде. Обеспечение непротиворечивости обычно обусловлено соблюдением других свойств ACID.

Изолированность

Транзакция является изолированной, если не испытывает влияние и сама не влияет на транзакции, одновременно применяемые к одним и тем же данным. Степень, в которой транзакция является изолированной от других транзакций, зависит от характерного для этой транзакции уровня изоляции (Transaction Isolation Level — TIL), который определяется с помощью команды SET TRANSACTION ISOLATION LEVEL. Уровни изоляции транзакции изменяются в широких пределах, начиная от полного отсутствия изоляции (когда транзакции могут читать незафиксированные данные и не могут устанавливать исключительные блокировки на ресурсах) и заканчивая изоляцией на уровне сериализуемых транзакций (в таких транзакциях блокируется весь набор данных, и другим пользователям запрещается читать и модифицировать эти данные каким-либо образом до завершения выполнения транзакции). (Дополнительная информация об этом приведена ниже в разделе "Уровни изоляции транзакций".) На каждом уровне изоляции достигается определенный компромисс между степенью распараллеливания (той степенью, в которой допускается одновременный доступ и модификация одного и того же набора данных многочисленными пользователями) и степенью непротиворечивости. Чем более непроницаемым является уровень изоляции, тем выше степень непротиворечивости данных. Но чем выше степень непротиворечивости, тем ниже степень распараллеливания. Последняя закономерность обусловлена тем, что для обеспечения непротиворечивости данных программа SQL Server блокирует ресурсы. Чем больше блокировок, тем меньше количество осуществляемых одновременно модификаций данных и тем ниже общая доступность данных.

Изолированность исключает возможность выборки в транзакции противоречивых или неполных фрагментов, подвергаемых в настоящее время модификациям в других транзакциях. Например, если в какой-либо транзакции происходит вставка ряда строк в таблицу, то изолированность исключает возможность обнаружения этих строк в других транзакциях до тех пор, пока не зафиксирована те-

кущая транзакция. Уровни изоляции транзакций программы SQL Server позволяют совместить потребности в доступе к данным с требованиями обеспечения целостности данных.

Устойчивость

Транзакция считается устойчивой, если она может быть завершена, несмотря на сбой системы, либо (что касается незафиксированной транзакции) может быть полностью отменена вслед за сбоем системы. Средства опережающей записи в журнал и средства восстановления базы данных программы SQL Server гарантируют, что результаты зафиксированных транзакций, еще не записанные в базу данных, сохраняются в базе в ходе восстановления после сбоя системы (выполняется накат), а транзакции, не завершившиеся ко времени сбоя системы, отменяются (выполняется откат).

Принципы выполнения транзакций программы SQL Server

Транзакции SQL Server напоминают пакеты команд, поскольку обычно состоят из нескольких операторов Transact-SQL, которые выполняются в виде одной группы. Отличие транзакций от пакетов команд состоит в том, что последние относятся к понятиям клиентского уровня (пакеты команд представляют собой механизм передачи совокупностей команд на сервер), а транзакции относятся к понятиям серверного уровня (транзакции управляют тем, как в программе SQL Server различаются единицы выполненной и выполняемой работы).

Между пакетами команд и транзакциями существует связь "многие ко многим". Пакеты команд могут содержать несколько транзакций, а одна транзакция может охватывать несколько пакетов команд. Как правило, следует избегать транзакций, которые предусматривают выполнение продолжительных пакетов команд, поскольку такие транзакции могут приводить к проблемам распараллеливания и производительности.

При внесении каждого изменения в данные программа SQL Server вносит запись об изменении в журнал транзакций еще до осуществления самого изменения. Именно поэтому программу SQL Server характеризуют как имеющую журнал опережающей записи — записи вносятся в журнал еще до того, как соответствующие изменения данных записываются в базу данных. Если бы не применялся такой способ внесения изменений в базу данных, то могли бы возникать ситуации, в которых не происходит откат изменений данных из-за того, что в программе сервера возник сбой еще до внесения записи в журнал.

Модификации никогда не записываются непосредственно на диск. Вместо этого программа SQL Server считывает страницы данных в область буферов по мере того, как возникает необходимость в получении соответствующих данных, после чего изменения в данных вносятся в память. Прежде чем изменить страницу,

находящуюся в памяти, сервер обеспечивает регистрацию этого изменения в журнале транзакций. Для журнала транзакций также предусмотрено кэширование, поэтому и указанные изменения первоначально выполняются в памяти. Опережающее ведение журнала гарантирует, что средства отложенной записи не запишут модифицированные страницы данных (называемые незафиксированными страницами) на диск, прежде чем будет выполнен вывод соответствующих этим страницам записей журнала.

В базу данных не вносятся какие-либо постоянные изменения до тех пор, пока не произойдет фиксация транзакции. Точное значение времени фиксации зависит от типа транзакции. После фиксации транзакции изменения, внесенные в этой транзакции, записываются в базу данных, и к этим изменениям больше не может применяться откат.

Независимо от того, должна ли быть внесена в журнал информация о выполнении какой-либо операции, или для операции не предусмотрена запись в журнал, завершение этой операции до фиксации ее результатов приводит к полному откату изменений, внесенных до ее фиксации. Возможность отката существует и в отношении операций, не записываемых в журнал, поскольку в журнал транзакций вносится информация о распределении страниц.

По отношению к транзакциям триггеры действуют так, как если бы они были вложены в транзакцию на один уровень глубже. Если происходит откат транзакции, содержащей триггер, то откату подвергаются и изменения, внесенные триггером. Если же происходит откат изменений, внесенных триггером, то откат применяется также ко всем транзакциям, включающим этот триггер.

Типы транзакций

В программе SQL Server поддерживаются четыре основных типа транзакций — автоматические, неявные, определяемые пользователем и распределенные. Каждый из этих типов характеризуется своими нюансами, поэтому ниже каждый тип рассматривается отдельно.

Автоматические транзакции

По умолчанию каждая команда Transact-SQL выполняется как отдельная транзакция. Применяемые при этом транзакции называются *автоматическими* (или транзакциями с автоматической фиксацией). Эти транзакции начинаются и фиксируются сервером автоматически. Примером автоматической транзакции может служить любая команда DML, которая выполняется вне пределов любой транзакции (а также в таком режиме, когда неявные транзакции запрещены). Автоматическую транзакцию можно рассматривать как оператор Transact-SQL, заключенный между командами `BEGIN TRAN` и `COMMIT TRAN`. Если выполнение такого оператора завершается успешно, происходит фиксация; в противном случае происходит откат.

Неявные транзакции

Неявные транзакции представляют собой автоматические транзакции, совместимые с версией языка SQL-92, которая утверждена стандартом ANSI. Неявные транзакции инициализируются автоматически при выполнении любой из нескольких команд DDL или DML и продолжаются до явной фиксации пользователем. Для ввода в действие поддержки неявных транзакций применяется команда `SET IMPLICIT_TRANSACTIONS`. По умолчанию в соединениях OLE DB и ODBC разрешен параметр `ANSI_DEFAULTS`, который, в свою очередь, разрешает неявные транзакции. Но обычно пользователи, которые устанавливают связь с сервером с помощью таких соединений, сразу же запрещают неявные транзакции, чтобы не возникали неуправляемые транзакции, способные нарушить работу приложения. Применение неявных транзакций можно сравнить с автоматическим включением блокировки дверей автомобиля после каждого их закрытия. С этим связано больше затрат, чем экономии времени, а рано или поздно владелец такого автомобиля останется на улице, поскольку забудет ключи в замке зажигания.

Определяемые пользователем транзакции

Основным средством управления транзакциями в приложениях SQL Server являются определяемые пользователем транзакции. Характерной их особенностью является то, что пользователь сам определяет, когда должна начаться и закончиться эта транзакция. Для управлениями определяемыми пользователем транзакциями служат команды `BEGIN TRAN`, `COMMIT TRAN` и `ROLLBACK TRAN`. Пример применения такой транзакции приведен в листинге 13.1.

Листинг 13.1. Определяемая пользователем транзакция

```
SELECT TOP 5 title_id, stor_id FROM sales ORDER BY
     title_id, stor_id
BEGIN TRAN
DELETE sales
SELECT TOP 5 title_id, stor_id FROM sales ORDER BY
     title_id, stor_id
GO
ROLLBACK TRAN
SELECT TOP 5 title_id, stor_id FROM sales ORDER BY
     title_id, stor_id

title_id stor_id
-------- -------
BU1032   6380
BU1032   8042
BU1032   8042
BU1111   8042
BU2075   7896

(5 row(s) affected)

(25 row(s) affected)
```

```
title_id stor_id
-------- -------

(0 row(s) affected)

title_id stor_id
-------- -------
BU1032   6380
BU1032   8042
BU1032   8042
BU1111   8042
BU2075   7896

(5 row(s) affected)
```

Распределенные транзакции

Транзакции, в которых участвует несколько серверов, называются *распределенными*. Управление такими транзакциями осуществляет центральное диспетчерское приложение, которое координирует действия участвующих серверов. Программа SQL Server может участвовать в распределенных транзакциях, координируемых диспетчерскими приложениями, которые поддерживают спецификацию X/Open XA средств распределенной обработки транзакций; в качестве примера такого приложения можно назвать программу DTC (Distributed Transaction Coordinator — координатор распределенных транзакций) компании Microsoft. Для инициализации распределенной транзакции на языке Transact-SQL используется команда `BEGIN DISTRIBUTED TRANSACTION`.

Полный отказ от использования транзакций

В действительности не существует иного способа полностью отменить ведение журнала транзакций, чем отказаться от внесения изменений в базу данных. Некоторые операции вырабатывают лишь минимальный объем информации журнала, но таких опций конфигурации, которые позволили бы полностью отменить ведение журнала, не существует.

Команды, требующие минимального объема записи в журнал

Минимальный объем записи в журнал информации о транзакциях связан с выполнением команд `CREATE INDEX`, `BULK INSERT`, `TRUNCATE TABLE`, `SELECT... INTO` и `WRITETEXT/UPDATETEXT`, поскольку при этом в журнал записываются только сведения об операциях со страницами. (Но в зависимости от

конкретных обстоятельств, обычные подробные записи журнала могут создаваться при выполнении команды BULK INSERT.) Вопреки широко распространенному ошибочному мнению, информация об указанных выше операциях в действительности записывается в журнал; просто при их выполнении не формируется подробная информация для журнала транзакций. Именно поэтому в оперативной документации Books Online указанные операции называются *не регистрируемыми в журнале операциями*, но их можно рассматривать как не регистрируемые в журнале только потому, что при выполнении этих операций не формируются записи журнала, относящиеся к уровню строки. В связи с этим автор часто называет перечисленные выше операции *регистрируемыми в журнале в минимальном объеме*.

Не регистрируемые в журнале операции обычно выполняются намного быстрее по сравнению с операциями, полностью регистрируемыми в журнале. К тому же, при выполнении этих операций вырабатываются записи журнала с информацией о распределении страниц, поэтому может осуществляться откат этих операций, как и любых других (но не накат). Применительно к не регистрируемым в журнале операциям выполняется немного другой процесс восстановления по журналу транзакций. Дело в том, что указанные операции уменьшают степень детализации информации, записываемой в журнал транзакций, поэтому изменяется и степень детализации процесса восстановления. Указанное изменение часто является вполне приемлемым, но пользователю следует знать об этом изменении характера восстановления.

Транзакции и модели восстановления

В программе SQL Server предусмотрены различные модели восстановления. Очевидно, что применяемая в данный момент модель восстановления влияет на управление транзакциями и журналом транзакций. При использовании модели восстановления `Simple` журнал транзакций, по существу, очищается после выполнения каждой контрольной точки, сгенерированной системой. Модель восстановления `Bulk-Logged` предусматривает полную запись в журнал информации обо всех операциях, кроме не регистрируемых в журнале операций. Модель восстановления `Full` предусматривает запись в журнал информации обо всех операциях, включая те, которые в других условиях не регистрируются.

Базы данных, допускающие только чтение, и однопользовательские базы данных

Одним из очевидных способов избавления от необходимости ведения журнала, а также исключения блокировок ресурсов и взаимоблокировок в базе данных является преобразование базы данных в допускающую только чтение. Безусловно, если в базу данных невозможно вносить изменения, то нет необходимости в регистрации транзакций или блокировке ресурсов. Если же база данных применяется как однопользовательская, то устраняется необходимость в использовании блокировок чтения, а это означает, что исключается возможность самоблокировки приложения.

Хотя предложение уменьшить доступность базы данных для сокращения до минимума количества проблем, связанных с управлением транзакциями, напоминает

рекомендацию не ездить на автомобиле, чтобы предотвратить его поломку, иногда указанный подход действительно применяется в реальных приложениях. Например, базы данных, допускающие только чтение, довольно часто встречаются в приложениях систем поддержки принятия решений (Decision Support System — DSS). Такие базы данных могут обновляться в нерабочие часы (например, по ночам или по выходным), затем снова переводиться в режим только чтения для эксплуатации в течение обычных рабочих часов. Очевидно, что проблемы управления транзакциями во многом теряют свою остроту, если модифицирует базу данных в один момент времени только один пользователь, а изменения вносятся крупными порциями или не вносятся вообще.

Кроме того, базы данных, допускающие только чтение, могут оказаться весьма удобными при их использовании в составе секционированных банков данных. Иногда данные, применяемые в приложении, могут быть распределены по нескольким базам данных. При этом один из них содержат статические и нормативные данные, не подверженные значительным изменениям (и поэтому допускающие перевод в режим только чтения), а другие — более часто изменяемые данные, к которым должны применяться хотя бы номинальные средства управления транзакциями.

Автоматическое управление транзакциями

В программе SQL Server предусмотрен целый ряд средств автоматического управления транзакциями. Наиболее ярким примером этих средств является средство автоматического выполнения транзакции (автоматической фиксации). Как было указано выше, инициализация и фиксация (или откат) автоматической транзакции осуществляются неявно сервером, поэтому нет необходимости явно применять операторы `BEGIN TRAN` или `COMMIT/ROLLBACK TRAN`. Сервер инициализирует транзакцию с началом выполнения команды модификации и, в зависимости от успешного или неудачного выполнения команды, в дальнейшем осуществляет фиксацию или откат. Режим автоматических транзакций применяется в программе SQL Server по умолчанию, но отменяется, если разрешены неявные или определяемые пользователем транзакции.

Неявные транзакции представляют собой разновидность средств управления автоматическими транзакциями. Транзакция инициализируется автоматически при выполнении определенных команд (`ALTER TABLE`, `FETCH`, `REVOKE`, `CREATE`, `GRANT`, `SELECT`, `DELETE`, `INSERT`, `TRUNCATE TABLE`, `DROP`, `OPEN`, `UPDATE`). В некотором смысле неявные транзакции представляют собой автоматизированный вариант явных транзакций, поэтому средства поддержки неявных транзакций по своим функциональным возможностям находятся примерно между транзакциями с автоматической фиксацией и транзакциями, определяемыми пользователем. Тем не менее неявные транзакции являются всего лишь полуавтоматическими, поскольку для их закрытия требуется явно заданный оператор `ROLLBACK TRAN` или `COMMIT TRAN`. Автоматизирована только первая часть совокупности действий, связанных с транзакцией, — инициализация транзакции. Завершение транзакции все равно должно быть обозначено явно. Для перехода в режим неявных транзакций применяется команда `SET IMPLICIT_TRANSACTIONS` языка Transact-SQL.

Разрешить или запретить аварийное завершение транзакции при возникновении ошибки этапа прогона в одной из команд транзакции можно с помощью параметра XACT_ABORT. В качестве ошибки может рассматриваться условие ошибки, выработанное системой или пользователем. По сути, контроль над ошибками эквивалентен проверке переменной @@ERROR после выполнения каждого оператора и откату транзакции в случае обнаружения ошибки. Следует отметить, что название самой команды SET XACT_ABORT (задать режим аварийного завершения транзакции) немного сбивает с толку. Дело в том, что если разрешен параметр XACT_ABORT и возникает ошибка этапа прогона, то аварийно завершается не только текущая транзакция, но и весь пакет команд, в который она входит. Например, рассмотрим код, приведенный в листинге 13.2.

Листинг 13.2. Использование параметра XACT_ABORT в ситуации, при которой выполнение определенной команды приводит к возникновению ошибки

```
SET XACT_ABORT ON
SELECT TOP 5 au_lname, au_fname FROM authors ORDER BY
     au_lname, au_fname
BEGIN TRAN
DELETE authors
DELETE sales
SELECT TOP 5 au_lname, au_fname FROM authors ORDER BY
     au_lname, au_fname
ROLLBACK TRAN
PRINT 'End of batch -- never makes it here'
GO
SELECT TOP 5 au_lname, au_fname FROM authors ORDER BY
     au_lname, au_fname
SET XACT_ABORT ON
au_lname                                 au_fname
---------------------------------------- --------------------
Bennet                                    Abraham
Blotchet-Halls                            Reginald
Carson                                    Cheryl
DeFrance                                  Michel
del Castillo                              Innes

(5 row(s) affected)

Server: Msg 547, Level 16, State 1, Line 1
DELETE statement conflicted with COLUMN REFERENCE constraint
    'FK__titleauth__au_id__164452B1'. The conflict occurred in
    database 'pubs', table 'titleauthor', column 'au_id'.
au_lname                                 au_fname
---------------------------------------- --------------------
Bennet                                    Abraham
Blotchet-Halls                            Reginald
Carson                                    Cheryl
DeFrance                                  Michel
del Castillo                              Innes

(5 row(s) affected)
```

Ход выполнения никогда не достигает оператора PRINT, поскольку нарушение ограничения, вызванное попыткой удалить все записи из таблицы authors, приводит к аварийному завершению всего пакета команд (всех операторов, предшествующих оператору GO). Эта ситуация возникает, несмотря на тот факт, что оператор ROLLBACK TRAN непосредственно предшествует оператору PRINT.

Таким образом, аварийно завершается весь пакет команд, поэтому вариант организации работы приложения, в котором выполняется проверка переменной @@ERROR после каждого внесения изменений в данные, является более предпочтительным, чем вариант, в котором разрешен параметр SET XACT_ABORT. Такое утверждение становится особенно справедливым, если в транзакции предусмотрен вызов хранимой процедуры. Дело в том, что при возникновении в процедуре ошибки этапа прогона завершаются также аварийно все операторы, которые следуют за вызовом этой процедуры в пакете команд, поэтому не остается ни малейшей возможности обработать условие ошибки.

Уровни изоляции транзакции

Программа SQL Server поддерживает четыре уровня изоляции транзакции. Как было указано выше, от уровня изоляции транзакции зависит то, в какой степени транзакция влияет и испытывает влияние других транзакций. При этом на одной чаше весов всегда находится непротиворечивость данных, а на другой — степень распараллеливания. Применение более ограничительного уровня изоляции транзакции приводит к увеличению непротиворечивости данных за счет уменьшения их доступности, а выбор менее ограничительного уровня изоляции транзакции приводит к увеличению степени распараллеливания за счет уменьшения непротиворечивости данных. Секрет успешной организации работы состоит в том, чтобы найти равновесие между этими противоположными тенденциями в целях успешного выполнения задач, стоящих перед эксплуатируемым приложением.

Для определения уровня изоляции транзакции применяется команда SET TRANSACTION ISOLATION LEVEL. В число допустимых параметров этой команды входят READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ и SERIALIZABLE.

Уровень изоляции транзакции READ UNCOMMITTED

Применение значения READ UNCOMMITTED, по существу, аналогично включению подсказки NOLOCK применительно к любой таблице, указанной в транзакции. Указанный уровень является наименее ограничительным из четырех уровней изоляции транзакции программы SQL Server. Он допускает чтение незафиксированных страниц (незафиксированных изменений, внесенных другими транзакциями) и неповторяемое чтение (чтение данных, изменяющихся от одной операции чтения, выполняемой в транзакции, до другой). Чтобы убедиться в том, что режим READ UNCOMMITTED допускает чтение незафиксированных страниц и неповторяемое чтение, одновременно выполните запросы, приведенные в листинге 13.3.

Листинг 13.3. Использование уровня изоляции транзакции `READ UNCOMMITTED`

```
-- Запрос 1

SELECT TOP 5 title_id, qty FROM sales ORDER BY title_id, stor_id
BEGIN TRAN
UPDATE sales SET qty=0
SELECT TOP 5 title_id, qty FROM sales ORDER BY title_id, stor_id
WAITFOR DELAY '00:00:05'
ROLLBACK TRAN
SELECT TOP 5 title_id, qty FROM sales ORDER BY title_id, stor_id

-- Запрос 2

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
PRINT 'Now you see it...'
SELECT TOP 5 title_id, qty FROM sales
WHERE qty=0
ORDER BY title_id, stor_id

IF @@ROWCOUNT>0 BEGIN
  WAITFOR DELAY '00:00:05'

  PRINT '...now you don''t'
  SELECT TOP 5 title_id, qty FROM sales
  WHERE qty=0
  ORDER BY title_id, stor_id
END

Now you see it...
title_id qty
-------- ------
BU1032    0
BU1032    0
BU1032    0
BU1111    0
BU2075    0

(5 row(s) affected)

...now you don't
title_id qty
-------- ------

(0 row(s) affected)
```

В то время как выполняется первый запрос, запустите на выполнение второй (у вас в запасе есть пять секунд). После этого можно будет обнаружить, что во втором запросе допускается возможность получения доступа к незафиксированным модификациям данных, внесенным в первом запросе. Затем во втором запросе происходит ожидание завершения первой транзакции, после чего снова предпринимаются попытки прочитать те же данные. А поскольку выполнен откат внесенных модификаций, то соответствующие данные уже исчезли, поэтому второй запрос оказался в ситуации неповторяемого чтения.

Уровень изоляции транзакции READ COMMITTED

Уровень изоляции транзакции READ COMMITTED применяется в программе SQL Server по умолчанию, поэтому в условиях отсутствия команды установки уровня изоляции транзакции используется READ COMMITTED. Режим READ COMMITTED позволяет исключить чтение незафиксированных страниц, поскольку обеспечивает установку разделяемых блокировок на совместно используемые данные, но позволяет вносить изменения в основополагающие данные во время транзакции, что может в принципе привести к неповторяемому чтению и/или появлению фантомных данных. Для того чтобы ознакомиться с тем, какие действия происходят при использовании данного уровня изоляции транзакции, выполните одновременно запросы, приведенные в листинге 13.4.

Листинг 13.4. Использование уровня изоляции транзакции READ COMMITTED

```
-- Запрос 1

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
BEGIN TRAN
PRINT 'Now you see it...'
SELECT TOP 5 title_id, qty FROM sales ORDER BY title_id, stor_id
WAITFOR DELAY '00:00:05'
PRINT '...now you don''t'
SELECT TOP 5 title_id, qty FROM sales ORDER BY title_id, stor_id
GO
ROLLBACK TRAN

-- Запрос 2

SET TRANSACTION ISOLATION LEVEL READ COMMITTED
UPDATE sales SET qty=6 WHERE qty=5

Now you see it...
title_id qty
-------- ------
BU1032   5
BU1032    10
BU1032    30
BU1111    25
BU2075    35

...now you don't
title_id qty
-------- ------
BU1032   6
BU1032    10
BU1032    30
BU1111    25
BU2075    35
```

Как и в предыдущем примере, запустите первый запрос, затем быстро вызовите для одновременного выполнения второй запрос (в вашем распоряжении имеется пять секунд).

В данном примере значение в столбце qty первой строки таблицы sales изменяется от одной операции чтения к другой во время выполнения первого запроса — такова классическая ситуация неповторяемого чтения.

Уровень изоляции транзакции REPEATABLE READ

В режиме REPEATABLE READ инициализируются блокировки, что позволяет исключить возможность внесения другими пользователями изменений в данные, которые используются в транзакции, но не исключена возможность вставки новых строк, что может привести к появлению фантомных строк в промежутке от одной операции чтения, выполняемой в транзакции, до другой. Соответствующий пример приведен в листинге 13.5. (Как и в других примерах, запустите первый запрос, затем одновременно вызовите на выполнение второй запрос — для запуска второго запроса у вас есть пять секунд.)

Листинг 13.5. Использование уровня изоляции транзакции REPEATABLE READ

```
-- Запрос 1

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
BEGIN TRAN
PRINT 'Nothing up my sleeve...'
SELECT TOP 5 title_id, qty FROM sales ORDER BY qty
WAITFOR DELAY '00:00:05'
PRINT '...except this rabbit'
SELECT TOP 5 title_id, qty FROM sales ORDER BY qty
GO
ROLLBACK TRAN

-- Запрос 2

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
INSERT sales VALUES
    (6380,9999999,GETDATE(),2,'USG-Whenever','PS2091')

Nothing up my sleeve...
title_id qty
-------- ------
PS2091    3
BU1032    5
PS2091    10
MC2222    10
BU1032    10

...except this rabbit
title_id qty
-------- ------
PS2091    2
PS2091    3
BU1032    5
PS2091    10
MC2222    10
```

Вполне очевидно, что в промежутке между первой и второй операциями чтения таблицы sales появилась новая строка, даже несмотря на то, что задан уровень изоляции транзакции REPEATABLE READ. Хотя в режиме REPEATABLE READ исключена возможность вносить изменения в данные, к которым уже происходит доступ, не исключается возможность добавления новых данных, что может привести к появлению фантомных строк.

Уровень изоляции транзакции SERIALIZABLE

Уровень изоляции транзакции SERIALIZABLE позволяет предотвратить чтение незафиксированных страниц и появление фантомных строк благодаря установке блокировок на всю совокупность данных, к которым применяется указанный режим доступа. Настоящий уровень изоляции транзакции является наиболее ограничительным из всех четырех уровней изоляции транзакции программы SQL Server и эквивалентным использованию подсказки HOLDLOCK для каждой таблицы, на которую имеется ссылка в транзакции. Пример применения этого уровня изоляции транзакции приведен в листинге 13.6. (Прежде чем приступить к выполнению кода этого листинга, удалите строку, введенную в предыдущем примере.)

Листинг 13.6. Использование уровня изоляции транзакции SERIALIZABLE

```
-- Запрос 1

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRAN
PRINT 'Nothing up my sleeve...'
SELECT TOP 5 title_id, qty FROM sales ORDER BY qty
WAITFOR DELAY '00:00:05'
PRINT '...or in my hat'
SELECT TOP 5 title_id, qty FROM sales ORDER BY qty
ROLLBACK TRAN

-- Запрос 2

BEGIN TRAN
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
-- Выполнение этого оператора INSERT откладывается до завершения
-- первой транзакции
INSERT sales VALUES
   (6380,9999999,GETDATE(),2,'USG-Whenever','PS2091')
ROLLBACK TRAN

Nothing up my sleeve...
title_id qty
-------- ------
PS2091   3
BU1032   5
PS2091    10
MC2222    10
BU1032    10
```

```
...or in my hat
title_id qty
-------- ------
PS2091   3
BU1032   5
PS2091    10
MC2222    10
BU1032    10
```

В этом примере блокировки, инициализированные на уровне изоляции SERIALIZABLE, исключают возможность выполнения второго запроса вплоть до завершения первого запроса. Хотя этот режим обеспечивает абсолютную непротиворечивость данных, такое свойство достигается за счет значительного снижения степени распараллеливания.

Команды управления транзакциями и синтаксическая структура этих команд

Как было указано выше, для управления транзакциями в языке Transact-SQL используются команды BEGIN TRAN, COMMIT TRAN и ROLLBACK TRAN. (Системные хранимые процедуры sp_xxxx_xact представляют собой унаследованный код, который применялся в прошлом в приложениях DB-Library, предназначенных для работы в режиме двухфазной фиксации, поэтому эти процедуры не следует использовать в современных приложениях.) Точная синтаксическая структура, применяемая для инициализации транзакций, показана ниже.

```
BEGIN TRAN[SACTION] [name|@TranNameVar]
```

Для фиксации транзакции используется следующая синтаксическая структура:

```
COMMIT TRAN[SACTION] [name|@TranNameVar]
```

А синтаксическая структура команды, предназначенной для отката транзакции, приведена ниже.

```
ROLLBACK TRAN[SACTION] [name|@TranNameVar]
```

Вместо команд COMMIT TRANSACTION и ROLLBACK TRANSACTION могут также использоваться команды COMMIT WORK и ROLLBACK WORK, но в последних не предусмотрена возможность указывать имена транзакций.

Вложенные транзакции

Язык Transact-SQL позволяет осуществлять операции с вложенными транзакциями, вызывая на выполнение вложенные команды BEGIN TRAN. Для определения уровня вложенности можно запрашивать значение автоматически сопровождаемой переменной @@TRANCOUNT; значение 0 указывает на отсутствие вложенности, 1 означает, что вложение осуществлено на один уровень, и т.д. В пакетах и хранимых процедурах, выполнение которых зависит от уровня вложенности, необходимо запрашивать значение переменной @@TRANCOUNT сразу

после вызова соответствующих программных объектов на выполнение и предусматривать соответствующие действия.

Хотя на первый взгляд создается иное впечатление, программа SQL Server не поддерживает транзакции, которые были бы действительно вложенными в полном смысле этого понятия. Команда COMMIT, выполненная применительно к любой транзакции, кроме самой внешней, не фиксирует какие-либо изменения на диске, а просто уменьшает значение автоматически сопровождаемой переменной @@TRANCOUNT. С другой стороны, команда ROLLBACK действует одинаково, вне зависимости от того, на каком уровне была вызвана, и выполняет откат всех транзакций, без учета уровня вложенности. Хотя такая организация работы на первый взгляд противоречит здравому смыслу, существуют весьма основательные причины ее применения. Дело в том, что если выполнение вложенных команд COMMIT действительно приводило бы к постоянной записи изменений на диск, то выполняемая на внешнем уровне команда ROLLBACK не давала бы возможность отменить внесенные изменения, поскольку они уже были бы записаны как предназначенные для постоянного хранения. Аналогично, если бы команда ROLLBACK не отменяла все изменения, внесенные на всех уровнях, то вызов этой команды из хранимых процедур и триггеров осуществлялся бы значительно сложнее, поскольку приходилось бы проверять возвращаемые значения и уровни вложенности транзакций при возврате из любой процедуры, для определения того, следует ли выполнить откат еще каких-то незавершенных транзакций. Ниже приведен пример, который иллюстрирует некоторые нюансы применения вложенных транзакций (листинг 13.7).

Листинг 13.7. Использование вложенных транзакций

```
SELECT 'Before BEGIN TRAN',@@TRANCOUNT
BEGIN TRAN
  SELECT 'After BEGIN TRAN',@@TRANCOUNT
  DELETE sales
  BEGIN TRAN nested
    SELECT 'After BEGIN TRAN nested',@@TRANCOUNT
    DELETE titleauthor
  COMMIT TRAN nested -- Не выполняет никаких действий, кроме
                     -- уменьшения значения @@TRANCOUNT
  SELECT 'After COMMIT TRAN nested',@@TRANCOUNT
GO -- По возможности рекомендуется помещать оператор ROLLBACK TRAN в
   -- отдельный пакет для предотвращения того, чтобы в результате
   -- возникновения ошибок в пакете не оставались открытые транзакции
ROLLBACK TRAN
SELECT 'After ROLLBACK TRAN',@@TRANCOUNT

SELECT TOP 5 au_id FROM titleauthor

----------------- -----------
Before BEGIN TRAN 0

---------------- -----------
After BEGIN TRAN 1

---------------------- -----------
After BEGIN TRAN nested 2
```

```
----------------------- -----------
After COMMIT TRAN nested 1

------------------- -----------
After ROLLBACK TRAN 0
au_id
-----------
213-46-8915
409-56-7008
267-41-2394
724-80-9391
213-46-8915
```

Этот пример показывает, что команда COMMIT TRAN является вложенной, но несмотря на это внешняя команда ROLLBACK позволяет отменить результаты выполнения команды DELETE titleauthor. В листинге 13.8 приведен еще один пример вложенной транзакции.

Листинг 13.8. Влияние использования команды ROLLBACK на вложенную транзакцию

```
SELECT 'Before BEGIN TRAN',@@TRANCOUNT
BEGIN TRAN
  SELECT 'After BEGIN TRAN',@@TRANCOUNT
  DELETE sales
  BEGIN TRAN nested
    SELECT 'After BEGIN TRAN nested',@@TRANCOUNT
    DELETE titleauthor
  ROLLBACK TRAN
  SELECT 'After ROLLBACK TRAN',@@TRANCOUNT
IF @@TRANCOUNT>0 BEGIN
  COMMIT TRAN -- Этот участок кода никогда не достигается вследствие
              -- выполнения оператора ROLLBACK
  SELECT 'After COMMIT TRAN',@@TRANCOUNT
END

SELECT TOP 5 au_id FROM titleauthor

----------------- -----------
Before BEGIN TRAN 0

---------------- -----------
After BEGIN TRAN 1

----------------------- -----------
After BEGIN TRAN nested 2
------------------- -----------
After ROLLBACK TRAN 0

au_id
-----------
213-46-8915
409-56-7008
267-41-2394
724-80-9391
213-46-8915
```

В данном примере в ходе выполнения никогда не достигается внешняя команда COMMIT TRAN, поскольку команда ROLLBACK TRAN отменяет все транзакции, выполняемые в настоящее время, и устанавливает значение переменной @@TRANCOUNT, равное 0.

Обратите внимание на то, что нельзя выполнить откат из вложенной транзакции. Команда ROLLBACK позволяет отменить указанную транзакцию, только если эта команда вызывается из внешней транзакции. Попытка выполнить откат из вложенной транзакции приводит к появлению следующего сообщения:

```
Server: Msg 6401, Level 16, State 1, Line 10
Cannot roll back nested. No transaction or savepoint of that name
was found.
```

Несмотря на само содержимое этого сообщения об ошибке, проблема состоит не в том, что не существует транзакция с указанным именем. Дело в том, что в команде ROLLBACK можно указывать транзакцию по имени, если эта транзакция является также внешней. Ниже приведен пример, который демонстрирует использование команды ROLLBACK TRAN с именами транзакций (листинг 13.9).

Листинг 13.9. Связь между именованными транзакциями и командой ROLLBACK

```
SELECT 'Before BEGIN TRAN main',@@TRANCOUNT
BEGIN TRAN main
  SELECT 'After BEGIN TRAN main',@@TRANCOUNT
  DELETE sales
  BEGIN TRAN nested
    SELECT 'After BEGIN TRAN nested',@@TRANCOUNT
    DELETE titleauthor
  ROLLBACK TRAN main
  SELECT 'After ROLLBACK TRAN main',@@TRANCOUNT
IF @@TRANCOUNT>0 BEGIN
  ROLLBACK TRAN  -- Этот участок кода никогда не достигается из-за
                 -- выполняемого ранее оператора ROLLBACK
  SELECT 'After ROLLBACK TRAN',@@TRANCOUNT
END

SELECT TOP 5 au_id FROM titleauthor

---------------------- -----------
Before BEGIN TRAN main 0

--------------------- -----------
After BEGIN TRAN main 1

----------------------- -----------
After BEGIN TRAN nested 2

------------------------ -----------
After ROLLBACK TRAN main 0

au_id
-----------
213-46-8915
409-56-7008
```

```
267-41-2394
724-80-9391
213-46-8915
```

В этом листинге показано, что внешней транзакции присвоено имя main, после чего в команду ROLLBACK TRAN введена ссылка на эту транзакцию. Обратите внимание на то, что в команде ROLLBACK TRAN никогда не требуется имя транзакции, независимо от того, была ли она инициализирована с указанием имени. По этой причине многие разработчики полностью избегают использования имен транзакций в команде ROLLBACK, поскольку такие имена не выполняют каких-либо реальных функций. Но выбор того или иного подхода представляет собой в основном проявление личного вкуса, поскольку оба варианта команды ROLLBACK действуют вполне приемлемо при условии, что разработчик понимает все нюансы ее использования. Команда ROLLBACK TRAN всегда осуществляет откат всех транзакций и устанавливает значение переменной @@TRANCOUNT, равное 0, независимо от того, в каком контексте вызвана эта команда. Исключением из данного правила является вызов команды ROLLBACK TRAN с указанием точки сохранения (как описано ниже).

Команда SAVE TRAN и точки сохранения

Для контроля над тем, какой объем работы будет отменен с помощью команды ROLLBACK, применяется команда SAVE TRAN. Команда SAVE TRAN создаст точку сохранения, к которой при желании можно выполнить откат. Для этого достаточно передать в команду ROLLBACK TRAN имя точки сохранения, в соответствии с синтаксисом этой команды. Пример определения точки сохранения показан в листинге 13.10.

Листинг 13.10. Установка точки сохранения с помощью команды SAVE TRAN

```
SELECT 'Before BEGIN TRAN main',@@TRANCOUNT
BEGIN TRAN main
  SELECT 'After BEGIN TRAN main',@@TRANCOUNT
  DELETE sales
  SAVE TRAN sales  -- Отметить точку сохранения
  SELECT 'After SAVE TRAN sales',@@TRANCOUNT
  -- Значение @@TRANCOUNT остается неизменным
  BEGIN TRAN nested
    SELECT 'After BEGIN TRAN nested',@@TRANCOUNT
    DELETE titleauthor
    SAVE TRAN titleauthor -- Отметить точку сохранения
    SELECT 'After SAVE TRAN titleauthor',@@TRANCOUNT
    -- Значение @@TRANCOUNT остается неизменным
  ROLLBACK TRAN sales
  SELECT 'After ROLLBACK TRAN sales',@@TRANCOUNT
  -- Значение @@TRANCOUNT остается неизменным
  SELECT TOP 5 au_id FROM titleauthor
IF @@TRANCOUNT>0 BEGIN
  ROLLBACK TRAN
  SELECT 'After ROLLBACK TRAN',@@TRANCOUNT
END

SELECT TOP 5 au_id FROM titleauthor
```

```
-------------------- -----------
Before BEGIN TRAN main 0

-------------------- -----------
After BEGIN TRAN main 1

-------------------- -----------
After SAVE TRAN sales 1

---------------------- -----------
After BEGIN TRAN nested 2

------------------------- -----------
After SAVE TRAN titleauthor 2

------------------------ -----------
After ROLLBACK TRAN sales 2

au_id
-----------
213-46-8915
409-56-7008
267-41-2394
724-80-9391
213-46-8915

------------------ -----------
After ROLLBACK TRAN 0

au_id
-----------
213-46-8915
409-56-7008
267-41-2394
724-80-9391
213-46-8915
```

Программа SQL Server позволяет при желании повторно использовать имя точки сохранения, но в таком случае сохраняется только последняя по времени точка сохранения с одним и тем же именем. При откате транзакции с помощью команды, в которой указано имя точки сохранения, происходит откат к команде с последним упоминанием имени этой точки сохранения.

Случайно выполняемые команды ROLLBACK

Команда ROLLBACK TRAN отменяет все транзакции, выполняемые ко времени ее вызова, поэтому необходимо следить за тем, чтобы в программе не были непреднамеренно допущены вложенные вызовы этой команды. После вызова команды ROLLBACK TRAN, выполненного впервые, нет необходимости (а также не разрешено) вызывать команду снова до инициализации новой транзакции. Например, рассмотрим код, приведенный в листинге 13.11.

Листинг 13.11. Использование команды ROLLBACK во вложенной транзакции

```
SELECT 'Before BEGIN TRAN',@@TRANCOUNT
BEGIN TRAN
  SELECT 'After BEGIN TRAN',@@TRANCOUNT
  DELETE sales
  BEGIN TRAN nested
    SELECT 'After BEGIN TRAN nested',@@TRANCOUNT
    DELETE titleauthor
  IF @@ROWCOUNT > 1000
     COMMIT TRAN nested
  ELSE BEGIN
    ROLLBACK TRAN -- Обеспечивается полный откат обеих транзакций
    SELECT 'After ROLLBACK TRAN',@@TRANCOUNT
  END
  SELECT TOP 5 au_id FROM titleauthor
ROLLBACK TRAN -- Возникает ошибка, связанная с тем, что отсутствуют
              -- транзакции, для которых мог быть выполнен откат
SELECT 'After ROLLBACK TRAN',@@TRANCOUNT

SELECT TOP 5 au_id FROM titleauthor

----------------- -----------
Before BEGIN TRAN 0

---------------- -----------
After BEGIN TRAN 1

----------------------- -----------
After BEGIN TRAN nested 2

------------------- -----------
After ROLLBACK TRAN 0

au_id
-----------
213-46-8915
409-56-7008
267-41-2394
724-80-9391
213-46-8915
Server: Msg 3903, Level 16, State 1, Line 17
The ROLLBACK TRANSACTION request has no corresponding
BEGIN TRANSACTION.

------------------- -----------
After ROLLBACK TRAN 0

au_id
-----------
213-46-8915
409-56-7008
267-41-2394
724-80-9391
213-46-8915
```

Обратите внимание на то, что после второго вызова команды ROLLBACK TRAN вырабатывается сообщение об ошибке. Дело в том, что первый вызов ROLLBACK TRAN приводит к отмене обеих транзакций, поэтому второй вызов уже не может отменить ни одной транзакции. Из этой ситуации проще всего найти выход, предусмотрев предварительную проверку значения переменной @@TRANCOUNT, как показано ниже.

```
IF @@TRANCOUNT>0 BEGIN
  ROLLBACK TRAN
  SELECT 'After ROLLBACK TRAN',@@TRANCOUNT
END
```

Синтаксические конструкции языка T-SQL, недопустимые в транзакциях

Некоторые синтаксические конструкции языка Transact-SQL, которые при обычных условиях являются допустимыми, становятся запрещенными на то время, пока является активной одна из транзакций. Например, нельзя использовать процедуру sp_dboption для изменения опций базы данных или вызывать какую-либо иную хранимую процедуру, которая модифицирует главную базу данных из транзакции. Кроме того, в транзакциях не допускается применять целый ряд команд Transact-SQL: ALTER DATABASE, DROP DATABASE, RECONFIGURE, BACKUP LOG, DUMP TRANSACTION, RESTORE DATABASE, CREATE DATABASE, LOAD DATABASE, RESTORE LOG, DISK INIT, LOAD TRANSACTION и UPDATE STATISTICS.

Отладка транзакций

При отладке транзакций и устранении связанных с ними проблем очень удобными являются две команды DBCC. Первая из них — команда DBCC OPENTRAN, которая позволяет осуществить выборку самой первой по времени (самой старой) активной транзакции в базе данных. Операции резервного копирования и удаления применяются только к той части журнала, которая содержит данные о неактивных транзакциях, поэтому транзакция, переданная на выполнение со злым умыслом или вышедшая из-под контроля, может вызвать преждевременное заполнение журнала. Для выявления транзакции, нарушающей нормальную работу, можно использовать команду DBCC OPENTRAN, чтобы иметь возможность завершить такую транзакцию в случае необходимости. Пример применения команды DBCC OPENTRAN показан в листинге 13.12.

Листинг 13.12. Использование команды DBCC OPENTRAN

```
DBCC OPENTRAN(pubs)
Transaction information for database 'pubs'.

Oldest active transaction:
   SPID (server process ID) : 15
   UID (user ID) : 1
```

```
Name           : user_transaction
LSN            : (57:376:596)
Start time     : Aug  5 1999  5:54:46:713AM
```

Еще одной удобной командой, применяемой для контроля над ошибками, связанными с транзакциями, является команда DBCC LOG, которая позволяет вывести на внешнее устройство журнал транзакций базы данных. Ее можно использовать, чтобы поднять завесу тайны и определить, какие операции выполняются над пользовательскими данными. Пример применения команды DBCC LOG показан в листинге 13.13.

Листинг 13.13. Использование команды DBCC LOG

```
CREATE TABLE #logrecs
  (CurrentLSN varchar(30),
  Operation varchar(20),
  Context varchar(20),
  TransactionID varchar(20))

INSERT #logrecs
EXEC('DBCC LOG(''pubs'')')

SELECT * FROM #logrecs
GO
DROP TABLE #logrecs
```

```
(Результаты приведены в сокращенном виде)
CurrentLSN                Operation          Context              TransactionID
------------------------- ------------------ -------------------- --------------
00000035:00000144:0001    LOP_BEGIN_CKPT     LCX_NULL             0000:00000000
00000035:00000145:0001    LOP_END_CKPT       LCX_NULL             0000:00000000
00000035:00000146:0001    LOP_MODIFY_ROW     LCX_SCHEMA_VERSION   0000:00000000
00000035:00000146:0002    LOP_BEGIN_XACT     LCX_NULL             0000:000020e0
00000035:00000146:0003    LOP_MARK_DDL       LCX_NULL             0000:000020e0
00000035:00000146:0004    LOP_COMMIT_XACT    LCX_NULL             0000:000020e0
00000035:00000147:0001    LOP_MODIFY_ROW     LCX_SCHEMA_VERSION   0000:00000000
00000035:00000147:0002    LOP_BEGIN_XACT     LCX_NULL             0000:000020e1
00000035:00000147:0003    LOP_MARK_DDL       LCX_NULL             0000:000020e1
```

Ни одно обсуждение отладки транзакции SQL Server не будет полным без упоминания автоматически сопровождаемой переменной @@TRANCOUNT. Переменная @@TRANCOUNT уже не раз упоминалась в настоящей главе, но следует также добавить, что эта переменная часто становится параметром операторов PRINT и средств контроля отладчика, поскольку данная переменная позволяет узнать уровень вложения текущей транзакции. При отладке сложных вложенных транзакций обычно принято вставлять во всем коде операторы SELECT или PRINT для определения текущего уровня вложения в различных значимых местах программы.

Наконец, не следует забывать о программе Perfmon, в которой предусмотрены многочисленные объекты и счетчики, относящиеся к управлению транзакциями и контролю производительности. В частности, значительное количество счетчиков, относящихся к транзакциям и журналу транзакций, предусмотрено в объекте SQL Server:Databases.

Оптимизация кода транзакций

Существует ряд общих рекомендаций по разработке эффективных процедур на языке T-SQL, ориентированных на работу с транзакциями. Ниже приведены некоторые из них.

- Транзакции должны быть настолько короткими, насколько это возможно. Определив, какие модификации с данными должны быть проведены, инициализируйте транзакцию, выполните эти модификации, а затем как можно быстрее завершите транзакцию. Старайтесь не инициализировать транзакции преждевременно.
- По мере возможности в транзакциях следует применять исключительно только операторы модификации данных. Не инициализируйте транзакцию во время просмотра данных, если этого можно избежать. Бесспорно, транзакции позволяют не только обеспечить корректную запись данных, но и корректное чтение (например, исключить операции чтения незафиксированных страниц и неповторяемого чтения, предотвратить появление фантомных строк и т.д.), но часто возможно ограничить транзакции лишь теми операторами, в которых происходит модификация данных, особенно если нет необходимости повторного чтения данных в транзакции.
- Не следует требовать ввода данных пользователем в течение транзакции, поскольку медленно работающий пользователь свяжет ресурсы сервера на время, которое для быстродействующей программы окажется неопределенно долгим. Кроме того, ввод данных пользователем в ходе транзакции может вызвать преждевременное заполнение журнала транзакций, поскольку из журнала невозможно будет исключать активные транзакции.
- По мере возможности следует применять оптимистическое управление распараллеливанием. Это означает, что в приложении не следует явно блокировать каждый объект, в котором могут происходить изменения, а нужно позволить серверу определять момент внесения изменений в какую-либо строку другим приложением. Может оказаться, что ситуация, в которой одновременная модификация одной и той же строки осуществляется двумя пользователями, возникает так редко (возможно, благодаря тому, что данные в приложении секционированы естественным образом, введенные строки обновляются редко и т.д.), что можно пойти на риск, отменив контроль над одновременным обновлением в целях повышения степени распараллеливания.
- Операции, не предусматривающие запись в журнал, следует использовать продуманно. Как уже было сказано выше, операции, не предусматривающие запись в журнал, влияют на процесс резервного копирования и восстановления, в котором применяется журнал транзакций. Этот фактор влияния может оказаться положительным или отрицательным, но в целом, если применение операций, не регистрируемых в журнале, является оправданным, это позволяет значительно повысить производительность приложения. Указанные операции часто способствуют сокращению времени обработки боль-

ших объемов данных на несколько порядков величины и практически устраняют целый ряд проблем, обычно связанных с управлением транзакциями. Не следует лишь забывать о том, что для такого повышения производительности иногда приходится принимать дополнительные меры.

- По возможности следует использовать более низкие (менее ограничительные) уровни изоляции транзакции. Предусмотренный по умолчанию уровень изоляции READ COMMITTED является приемлемым для большинства приложений и обеспечивает большую степень распараллеливания по сравнению с уровнем изоляции REPEATABLE READ или SERIALIZABLE.
- Следует стремиться свести к минимуму объем данных, модифицируемых в одной транзакции. Непродуманная попытка модифицировать миллионы строк в таблице не позволяет надеяться на то, что при этом проблемы распараллеливания и оптимального использования ресурсов решатся сами собой, как по волшебству. Для внесения изменений в базу данных требуются ресурсы и блокировки, а такие блокировки по определению влияют на работу других пользователей. Поэтому, если приложение не рассчитано лишь на одного пользователя, следует внимательно продумывать операции, которые способны оказать отрицательное влияние на степень распараллеливания.
- Не следует использовать неявные транзакции, за исключением тех ситуаций, когда они действительно требуются, и даже в этих случаях необходимо предусмотреть тщательный контроль над такими транзакциями. Неявные транзакции инициализируются практически всеми основными командами Transact-SQL (включая SELECT), поэтому запуск таких транзакций может происходить тогда, когда этого меньше всего следует ожидать, в результате чего снижается степень распараллеливания и возникают проблемы, связанные с обработкой журнала транзакций. Почти всегда лучше управлять транзакциями явно с помощью команд BEGIN TRAN, COMMIT TRAN и ROLLBACK TRAN, чем использовать неявные транзакции. Управляя транзакциями явно, можно точно указать, когда они должны начинаться и заканчиваться, поэтому достигается полный контроль над тем, что происходит.

Резюме

Транзакции представляют собой основную единицу работы программы SQL Server. Они гарантируют, что операции модификации данных либо выполняются полностью, либо вообще не выполняются. Транзакции SQL Server характеризуются свойствами неразрывности, непротиворечивости, изоляции и устойчивости (так называемыми свойствами ACID – Atomicity, Consistency, Isolation, Durability) и позволяют гарантировать предотвращение появления неполных данных или потерянных обновлений.

Для контроля над тем, насколько транзакции изолированы друг от друга, служит текущее значение уровня изоляции транзакции. Текущее значение уровня изоляции транзакции задается с помощью команды SET TRANSACTION ISOLATION

LEVEL. Каждый уровень изоляции транзакции представляет собой определенный компромисс между степенью распараллеливания и непротиворечивостью.

В настоящей главе даны общие сведения о транзакциях SQL Server и описаны различные команды Transact-SQL, относящиеся к управлению транзакциями. В ней представлены транзакции с автоматической фиксацией и неявные транзакции, а также кратко описаны транзакции, определяемые пользователем, и распределенные транзакции. Кроме того, в этой главе показаны некоторые распространенные проблемы, связанные с осуществлением транзакций, и описаны способы устранения этих проблем.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Модель восстановления `Simple` предусматривает полную регистрацию всех операций, кроме не регистрируемых в журнале операций, таких как `BULK INSERT`.
2. Какая процентная доля модификаций, внесенных в транзакции, записывается в журнал транзакций после вызова операции `COMMIT` на выполнение из вложенной транзакции?
3. Какая процентная доля модификаций, внесенных в транзакции, подвергается откату после вызова операции `ROLLBACK` на выполнение из вложенной транзакции?
4. Подтвердите или опровергните следующее утверждение. Программа SQL Server автоматически выполняет откат транзакции, которая была инициализирована из аварийно завершенного пакета Transact-SQL.
5. Относится ли команда `CREATE INDEX` языка Transact-SQL к числу не регистрируемых в журнале команд (точнее, команд, регистрируемых в журнале в минимальном объеме)?
6. Опишите различие между полностью регистрируемой в журнале командой и командой, регистрируемой в журнале в минимальном объеме.
7. Какая команда `DBCC` позволяет получить информацию о самой первой по времени (самой старой) активной транзакции в рассматриваемой базе данных и об идентификаторе инициализировавшего эту транзакцию серверного процесса?
8. Допустим, что разрабатывается триггер, который сохраняет значение `@@TRANCOUNT` во вторичной таблице, а активизация этого триггера осуществляется путем вызова на выполнение оператора DML вне транзакции, применяемой к основополагающей таблице триггера. Какое значение переменной `@@TRANCOUNT` этот триггер должен вставить во вторичную таблицу?
9. Опишите четыре свойства ACID, которыми может обладать транзакция.
10. Опишите четыре уровня изоляции транзакции, предусмотренные в программе SQL Server.

ГЛАВА 14

Курсоры

В этой главе приведено обновленное описание курсоров, которое впервые было опубликовано в книге автора *The Guru's Guide to Transact-SQL*. В данной главе продолжено обсуждение курсоров, начатое в указанной книге; описано использование курсоров в приложениях SQL Server; кроме того, изложенная в предыдущей книге информация обновлена с учетом текущего выпуска программного продукта SQL Server.

Краткий обзор

Курсор – это механизм доступа к строкам в таблице или в результирующем наборе на низком уровне детализации (на уровне отдельных строк). Курсоры действуют по принципу, отличному от обычно применяемого в программе SQL Server, поскольку в курсорах результирующие наборы разбиваются на отдельные строки. Выборка строки с помощью курсора аналогична получению отдельной строки с помощью однострочного оператора SELECT. Курсор обеспечивает автоматическое отслеживание его позиции и предоставляет целый ряд средств для перемещения по основополагающему результирующему набору с помощью прокрутки (а в обычном результирующем наборе это не предусмотрено). Кроме того, курсоры предоставляют удобные средства обновления основополагающего результирующего набора с учетом позиции и позволяют обращаться к указателям результирующего набора с помощью переменных.

Автор обычно рекомендует разработчикам, продумывающим возможность использования курсоров в своих приложениях, сразу отказаться от курсоров. Если поставленная задача может быть решена с помощью многочисленных инструментальных средств языка Transact-SQL, предназначенных для работы с наборами данных, так и следует сделать. Решения, основанные на использовании курсоров, редко превосходят по своей производительности решения, основанные на применении наборов данных (но такая возможность не исключена). Стандартные результирующие наборы программы SQL Server (называемые также в шутку курсорами для экстренных ситуаций) служили для решения бесчисленного множества разнообразных вычислительных задач в течение многих лет, поскольку количество типичных приложений баз данных, в которых фактически требуются курсоры, не так уж велико. Тем не менее некоторые приложения действительно больше приспособлены для использования курсоров, чем для обработки наборов данных.

Несколько слов о курсорах и базах данных ISAM

У разработчиков, перед которыми встает задача переноса приложений для СУБД ISAM или какой-либо локальной базы данных в СУБД SQL Server, часто возникает соблазн подойти к решению этой задачи поверхностно — вносить не больше изменений по сравнению с тем, что абсолютно необходимо для обеспечения работы приложения в новой СУБД. При этом обычно предусматривается использование таких упрощенных методов переноса приложения, как замена средств перебора записей СУБД ISAM (например, оператора `Recordset.MoveNext` интерфейса ADO) циклами курсоров Transact-SQL. Но записи ISAM и курсоры SQL Server не равнозначны, поэтому любая попытка рассматривать реляционную СУБД как аналогичную программному продукту ISAM чаще всего оканчивается плачевной неудачей.

Несколько лет тому назад автор имел несчастье взять на себя задачу переделки приложения базы данных ISAM в полноценное приложение SQL Server. Перед этим автор долго пытался убедить компанию заказчика перейти на клиент-серверную технологию реляционной СУБД. Заказчик несколько месяцев равнодушно выслушивал доводы автора, но наконец решил, что для проверки нового подхода нужно перевести свое ведущее приложение с программного продукта ISAM на SQL Server. К тому времени, несмотря на все усилия автора, неотразимые преимущества реляционных СУБД не были очевидны заказчику, поэтому автор решил взять на себя выполнение указанной задачи, чтобы доказать жизнеспособность реляционной технологии. Это решение было принято вопреки тому факту, что следовало лучше начать с создания нового приложения, чем с переработки жизненно важного для работы компании программного продукта.

Не прислушиваясь к предостережениям своего внутреннего голоса и не исследовав глубоко сам код, автор приступил к решению поставленной задачи, наивно полагая, что разработчики создали данное приложение, опираясь на обоснованные принципы логики и реляционного подхода. Автор не имел оснований предложить иное, поэтому допустил, что разработчики при любой возможности применяют обработку записей, представленных в виде наборов, в целях повышения производительности и уменьшения объема кода; ведь даже в той небольшой локальной СУБД, на которой было построено это приложение, поддерживался значительный объем средств доступа, ориентированных на использование наборов данных (включая собственный простой диалект SQL этой СУБД). Безусловно, автор не предполагал, что код приложения является идеальным, но считал оправданным свое предположение, что разработчики использовали находящиеся в их распоряжении инструментальные средства в основном в такой форме, для чего и были предназначены эти средства. По крайней мере такое впечатление сложилось в ходе собеседования с авторами приложения. Поэтому автор, не задумываясь, приступил к выполнению поставленной задачи, не вняв предостережениям своего внутреннего голоса.

Потратив две-три недели на анализ одного из худших приложений, которые когда-либо встречались в его практике, затем понаблюдав, как приложение само блокирует себе доступ к ресурсам сервера из-за своего ужасающего проекта,

и набив немало других шишек, автор наконец отказался от непосредственного преобразования рассматриваемого приложения в приложение SQL Server.

Разработчики приложения нарушили практически все основные принципы качественного проектирования приложений базы данных. В коде приложения применялась обработка таблиц в циклах вместо обработки строк в наборах данных. Даже те минимальные требования по поддержке реляционных операций и обеспечению целостности данных, которые были реализованы в приложении, основывались на невообразимой смеси прикладного кода и ограничений базы данных, поэтому совершенно не обеспечивали защиту данных. В приложении использовалась трудоемкая схема поддержки версий таблиц, которая так и не была закончена или полностью реализована, а также не носила ни малейшего признака соблюдения каких-либо непротиворечивых соглашений об именовании или применении прописных и строчных букв в именах, поэтому объекты базы данных носили запутанные имена, непригодные для запоминания и несовместимые друг с другом. Одни и те же атрибуты в разных таблицах часто имели различные имена, а разные атрибуты в различных таблицах часто обозначались одинаково. Таблицы во всей базе данных были денормализованными, но не в целях повышения производительности, а потому, что разработчики просто не знали, как создать лучшие таблицы. Не было и следа попыток обеспечить распараллеливание работы, поэтому приложение в соответствии с самим его проектом (или в связи с отсутствием такового) проявляло себя как предназначенное исключительно для одного пользователя. Короче говоря, с точки зрения архитектуры приложения данная разработка представляла собой полное нарушение всех принципов, и тот факт, что оно вообще работало даже на основе программного продукта ISAM, был доказательством упорства разработчиков приложения, а не надежности самого приложения.

Итак, вскоре после приобретения печального опыта работы с приложением, автор приступил к его полной переделке. Безусловно, можно было бы пойти по "простому" пути, — выполнить лишь поверхностную переделку приложения для подготовки его к эксплуатации в СУБД SQL Server; при этом современная серверная программа SQL Server, по существу, превратилась бы в почтенный сервер базы данных ISAM. Можно было бы также использовать максимально возможный объем существующего кода, не обращая внимания на то, насколько плохо он спроектирован. Все операции построчного доступа в приложении можно было бы преобразовать в эквивалентные операции с курсорами в программе SQL Server. Можно было бы использовать программу SQL Server в той форме, для которой она никогда не была предназначена, и отказаться от устранения в приложении многих реляционных и прочих проблем, грубо состыковав друг с другом всевозможные разрозненные фрагменты и создав бесформенную программу-монстра. Автор мог все это сделать (что, безусловно, позволило бы быстрее добиться кратковременного успеха и сделать приятное заказчику), но на это просто нельзя было пойти. Опыт автора свидетельствует о том, что обычно существует оптимальный способ создания программного обеспечения, но все его взгляды, навыки и знания говорили о том, что поверхностная переделка рассматриваемого приложения — далеко не оптимальный способ.

Вместо этого автору стало очевидно, что приложение необходимо полностью перепроектировать от самого основания, если оно должно стать примером качественной работы, сделанной на основе СУБД SQL Server или на основе какой-либо

другой реляционной СУБД. Острая потребность в полной переделке во многом была также прежде всего обусловлена радикальными различиями между программными продуктами ISAM и реляционными СУБД, не говоря уже о том, что этого требовало наличие некачественного проектирования и кодирования самого приложения. Тот факт, что программное обеспечение внешне кажется действующим должным образом, не говорит о том, что оно было спроектировано правильно, как и не говорит о надежности дома тот факт, что он не разрушается сразу после того, как в него въезжают жильцы. О качестве приложения следует судить, скорее, по его проекту, а не на основании того, соответствует приложение сиюминутным требованиям заказчика. Безусловно, важно, чтобы заказчики были довольны, но, стремясь к этой цели, нельзя полностью отрицать такие незыблемые требования, как расширяемость, функциональная совместимость, производительность, масштабируемость, распараллеливание работы и удобство сопровождения.

Описанный подход может показаться основанным исключительно на технологических соображениях, но заказчики, вольно или невольно, заботятся и о наличии в приложении перечисленных выше качеств. Безусловно, на эксплуатации приложения в обычных условиях показатели качества приложения отражаются не прямо, а косвенно. Но если приложение было с самого начала спроектировано неправильно, то просьба о доработке, которую заказчик может считать легко выполнимой (например, об обеспечении работы нескольких пользователей с тем приложением, с которым в настоящее время работает только один пользователь), покажется разработчику сложной или даже неосуществимой. Если разработчик приложения на этапе проектирования не задумывался над тем, чтобы обеспечить распараллеливание работы, то, скорее всего, придется полностью переделать все приложение, для того чтобы оно могло поддерживать работу нескольких пользователей. Из-за подобных переделок выпуск новых версий задерживается и заказчикам приходится долго ждать реализации средств, в которых они остро нуждаются. Проект приложения — это не абстракция, а реальность, которая затрагивает реальные интересы людей. Разработчик просто обязан создать качественный проект. Сам же заказчик может судить о качестве проекта лишь по внешним признакам.

Самой любопытной стороной урока, который был усвоен автором в описанной ситуации, оказалось то, что многие проектные решения в рассматриваемом приложении имели большую значимость для СУБД SQL Server, чем для платформы базы данных ISAM. Значительная часть дефектов первоначального проекта приложения стала очевидной только в связи с необходимостью переноса приложения в СУБД SQL Server. Именно в связи с этим возникла необходимость отказаться от существующего проекта приложения или подвергнуть проект полной переработке. В основе функционирования реляционных СУБД лежат требования по обеспечению надежности и производительности, поэтому СУБД такого типа в меньшей степени допускают применение приложений с некачественными проектами, чем программные продукты ISAM. Но автор считает такую особенность реляционных СУБД не недостатком, а достоинством. К тому же, разработчики не имеют права создавать некачественные приложения, для какой бы серверной платформы не были предназначены эти приложения.

В целом можно отметить, что перенос приложения ISAM в СУБД SQL Server — непростая задача, даже если это приложение спроектировано правильно.

Подход, предусматривающий быстрое выполнение поверхностной переделки с применением таких методов, как замена средств доступа ISAM курсорами SQL Server, почти никогда не оправдывается. Безусловно, чтобы заявить: "Для переделки этого приложения потребуется много работы; приложение должно быть полностью перепроектировано или разработано заново", необходимо иметь немалую решительность и сильный характер, но такой подход часто оказывается наилучшим. Повторное изобретение колеса становится приемлемым (или даже необходимым), если "повторно изобретаемое" колесо с самого начала было квадратным. При переносе существующих приложений в СУБД SQL Server чаще всего приходится предусматривать полную переделку; дело в том, что программу SQL Server следует рассматривать как фундамент, на котором должно строиться приложение, а не просто как одну из служб, которая приходит на смену другой. Поверхностная переделка – для тех, о ком Рон Соукап[1] сказал: "... они считают, что не нужно тратить много времени, чтобы обеспечить надлежащий перенос приложения на другую платформу, зато потом всегда находят время на переработку приложения".

Типы курсоров

Язык Transact-SQL поддерживает четыре типа курсоров: `FORWARD_ONLY`, `DYNAMIC`, `STATIC` и `KEYSET`. Основные различия между этими типами состоят в присущей им способности обнаруживать изменения в основополагающих данных во время прохождения курсора, а также в том, какие ресурсы используются в курсорах (блокировки, пространство базы данных `tempdb` и т.д.).

В зависимости от типа созданного курсора, изменения, внесенные в основополагающие данные курсора, могут обнаруживаться или не обнаруживаться во время прохождения по результирующему набору курсора. Эти изменения могут не только стать причиной появления новых значений столбцов, но и отразиться на том, какие строки возвращаются из курсора (вернее, какие строки входят в состав курсора), а также на упорядочении возвращаемых строк. Кроме того, открытие курсора может вызвать размещение всего результирующего набора курсора (или всех ключей курсора) во временной таблице, что может привести к возникновению проблем конкуренции за ресурсы базы данных `tempdb`. Общие сведения о различных типах курсоров и их характеристиках приведены в табл. 14.1.

Курсоры с перемещением только вперед

Курсор с перемещением только вперед (применяемый по умолчанию) возвращает строки из базы данных последовательно. Для этого курсора не требуется пространство в базе данных `tempdb`, а изменения, внесенные в основополагающие данные, становятся видимыми, как только соответствующие строки становятся доступными в курсоре. Пример использования курсора с перемещением только вперед приведен в листинге 14.1.

---

[1] Soukup, Ron. *Inside SQL Server 6.5*. Redmond, WA: Microsoft Press, 1998, p. 533.

Таблица 14.1. Типы курсоров, поддерживаемых в языке Transact-SQL, и характеристики этих курсоров

Тип	Прокручиваемый	Способ определения состава и порядка срок	Способ определения значений столбцов
FORWARD_ONLY (применяется по умолчанию)	Нет	Определяются динамически	Определяются динамически
DYNAMIC/ SENSITIVE	Да	Определяются динамически	Определяются динамически
STATIC/ INSENSITIVE	Да	Остаются постоянными	Остаются постоянными
KEYSET	Да	Остаются постоянными	Определяются динамически

Листинг 14.1. Использование курсора с перемещением только вперед

```
CREATE TABLE #temp (k1 int identity, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
DECLARE c CURSOR FORWARD_ONLY
FOR SELECT k1, c1 FROM #temp

OPEN c

FETCH c

UPDATE #temp
SET c1=2
WHERE k1=3

FETCH c
FETCH c

SELECT * FROM #temp

CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp

k1          c1
----------- -----------
1           NULL

k1          c1
----------- -----------
2           NULL

k1          c1
----------- -----------
```

```
3            2

k1          c1
----------- -----------
1           NULL
2           NULL
3            2
4           NULL
```

Динамические курсоры

Как и курсоры с перемещением только вперед, динамические курсоры отражают изменения в строках, на которых они основаны, сразу после того, как эти строки становятся доступными в курсоре. Дополнительное пространство в базе данных tempdb не требуется. В отличие от курсоров с перемещением только вперед, динамические курсоры по своей сути являются прокручиваемыми, поэтому можно не ограничиваться последовательным доступом к строкам курсора. Динамические курсоры иногда называют курсорами, восприимчивыми к изменениям, поскольку они позволяют обнаруживать изменения в исходных данных. Пример использования динамического курсора приведен в листинге 14.2.

Листинг 14.2. Использование динамического курсора

```
CREATE TABLE #temp (k1 int identity, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES

DECLARE c CURSOR DYNAMIC
FOR SELECT k1, c1 FROM #temp

OPEN c

FETCH c

UPDATE #temp
SET c1=2
WHERE k1=1

FETCH c
FETCH PRIOR FROM c

SELECT * FROM #temp

CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp

k1          c1
```

```
----------- -----------
1           NULL
k1          c1
----------- -----------
2            NULL

k1          c1
----------- -----------
1            2

k1          c1
----------- -----------
1            2
2            NULL
3            NULL
4            NULL
```

Как показано в данном листинге, осуществляется выборка одной строки, затем обновление этой строки, выборка другой строки, после этого повторная выборка первой строки. После того как происходит выборка первой строки во второй раз, обнаруживается изменение, внесенное с помощью операции UPDATE, даже несмотря на то, что в этой операция для внесения изменения не использовался курсор.

Статические курсоры

Статический курсор возвращает результирующий набор, допускающий только чтение, который не показывает изменения в основополагающих данных. Он обладает свойствами, противоположными свойствам динамического курсора, хотя и является полностью прокручиваемым. После открытия статического курсора изменения, внесенные в его исходные данные, не отражаются в курсоре. Это связано с тем, что весь результирующий набор курсора сразу после его открытия копируется в базу данных tempdb. Статические курсоры иногда называют курсорами-снимками или курсорами, невосприимчивыми к изменениям в данных, поскольку они не показывают изменений, внесенных в их исходные данные. Пример использования статического курсора приведен в листинге 14.3.

Листинг 14.3. Использование статического курсора

```
CREATE TABLE #temp (k1 int identity, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES

DECLARE c CURSOR STATIC
FOR SELECT k1, c1 FROM #temp
OPEN c  -- Весь результирующий набор копируется в tempdb

UPDATE #temp
SET c1=2
```

```
WHERE k1=1

FETCH c  -- Результаты этой операции не отражают изменений, внесенных
         -- операцией UPDATE

SELECT * FROM #temp  -- Но изменения действительно были внесены

CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp

k1          c1
----------- -----------
1           NULL

k1          c1
----------- -----------
1           2
2           NULL
3           NULL
4           NULL
```

В данном листинге показано, как происходит открытие курсора, после чего немедленно вносится изменение в первую строку основополагающей таблицы курсора. Это изменение не обнаруживается при выборке той же строки из курсора, поскольку эта строка фактически считывается из базы данных tempdb. Последовательное применение оператора SELECT к основополагающей таблице показывает, что изменение действительно произошло, даже несмотря на то, что оно не отражено курсором.

Курсоры ключевого набора

Открытие курсора ключевого набора приводит к получению полностью прокручиваемого результирующего набора, в котором состав строк и порядок расположения строк являются фиксированными. Как и для курсоров с перемещением только вперед и статических курсоров, изменения, внесенные в значения основополагающих данных (за исключением столбцов ключевого набора), обнаруживаются во время доступа к соответствующим строкам, но результаты вставки новых строк в курсоре не обнаруживаются. При открытии статического курсора в базу данных tempdb копируется результирующий набор, а при открытии курсора ключевого набора в базе данных tempdb создается таблица, и в нее копируется набор уникальных значений ключей для строк курсора (поэтому курсоры такого типа и называются курсорами "ключевого набора"). Именно с этим связано указанное выше свойство курсора, благодаря которому состав строк в курсоре является фиксированным. Если основополагающая таблица не имеет первичного или уникального ключа, то в таблицу ключевого набора копируется весь набор столбцов потенциального ключа. Поскольку изменения в столбцах ключевого набора не обнаруживаются в курсоре, то в случае отсутствия уникального ключа того или

иного типа для основополагающих данных создается ключевой набор, не отражающий изменения в каком-либо из столбцов потенциального ключа. Пример использования курсора ключевого набора приведен в листинге 14.4.

Листинг 14.4. Использование курсора ключевого набора

```
CREATE TABLE #temp (k1 int identity PRIMARY KEY, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES

DECLARE c CURSOR KEYSET
FOR SELECT k1, c1 FROM #temp

OPEN c  -- В tempdb копируется ключевой набор

UPDATE #temp
SET c1=2
WHERE k1=1

INSERT #temp VALUES (3) -- Результаты этой операции не будут видимыми
                        -- в курсоре (можно без опасений пропустить
                        -- столбец identity)

FETCH c -- Внесенное изменение становится видимым
FETCH LAST FROM c -- А новая строка - нет

SELECT * FROM #temp

CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp

k1          c1
----------- -----------
1           2
k1          c1
----------- -----------
4            NULL

k1          c1
----------- -----------
1            2
2            NULL
3            NULL
4            NULL
5            3
```

В данном листинге показано, что после открытия курсора ключевого набора в первую строку результирующего набора курсора вносится изменение еще до того, как в курсоре происходит выборка этой строки. Затем в основополагающую таблицу вставляется еще одна строка. С того времени, как в рассматриваемой

процедуре начинается выборка строк из курсора, первое изменение обнаруживается, а появление новой строки — нет. Это связано с тем, что данные о принадлежности строк к курсору ключевого набора не изменяются после открытия этого курсора.

Обратите внимание на то, что в рабочую таблицу включено ограничение PRIMARY KEY. При отсутствии этого ограничения изменения в столбце c1 рабочей таблицы не были бы видимыми в курсоре, даже несмотря на то, что курсор имеет столбец идентификации identity. Это связано с тем, что столбцы идентификации сами по себе не гарантируют уникальность содержащихся в них данных. Дело в том, что можно всегда воспользоваться командой SET IDENTITY_INSERT для ввода дубликатов идентификационных значений или переопределить начальное значение идентификатора для того, чтобы дубликаты идентификационных значений автоматически вводил сервер. Для обеспечения уникальности требуется ограничение PRIMARY KEY или UNIQUE KEY. При отсутствии уникального ключа сервер копирует все потенциальные ключи для каждой строки во временную таблицу курсора ключевого набора.

Рекомендации по правильному использованию курсоров

Прежде всего автор обязан дать один совет — курсоры следует использовать исключительно в случае необходимости. Такая рекомендация может показаться немного упрощенной или слишком широкой, но, по мнению автора, большинство опытных разработчиков, применяющих язык Transact-SQL, согласятся с тем, что к использованию курсоров следует прибегать лишь после того, как будут исчерпаны почти все возможности справиться с задачей с помощью других методов организации приложения. Вместо этого необходимо пытаться найти решение рассматриваемой задачи, позволяющее воспользоваться средствами языка Transact-SQL, предназначенными для работы с наборами данных. Именно этим подходом должны руководствоваться проектировщики; средства работы с наборами данных являются наиболее приемлемыми. Безусловно, начинающие разработчики способны легко усваивать понятие курсора, но чрезмерное или неправильное применение курсоров является основным источником проблем производительности в большинстве реляционных СУБД, включая SQL Server.

Сказанное выше не означает, что использовать курсоры запрещено или что все пользователи курсоров будут сталкиваться с проблемами. Любой разработчик, который достаточно долго занимался программированием на языке Transact-SQL, будет вынужден рано или поздно применить курсоры. В частности, курсоры широко используются в некоторых направлениях разработки. Как и во многом другом, успех или неудача при использовании курсоров в разработке главным образом зависит от того, приемлем ли связанный с этим подход. Применяйте курсоры, когда в этом есть смысл, но следите за тем, чтобы их использование было оправданным.

К некоторым примерам ситуаций, в которых применение курсоров вполне обосновано, относятся динамические запросы, операции, ориентированные на обработку строк, и прокручиваемые формы. В динамических запросах код Transact-SQL формируется и выполняется на этапе прогона. Операции, ориентированные

на обработку строк, представляют собой процедуры, состоящие из многочисленных операторов, которые слишком сложны, для того, чтобы их можно было выполнить с помощью операций, реализуемых с помощью одного оператора, такого как SELECT или UPDATE (необходимость в использовании подобных операций, ориентированных на обработку строк, может быть также обусловлена другими причинами). Прокручиваемые формы обычно реализуют определенное средство, позволяющее пользователям перемещаться в результирующем наборе (к тому же, иногда предусматривают вывод на внешнее устройство содержимого нескольких строк). Прокручиваемые курсоры до предела упрощают для разработчика задачу реализации подобных функциональных возможностей.

Динамические запросы

Курсоры являются удобным средством создания динамических запросов, поскольку позволяют вырабатывать исполняемый код на языке Transact-SQL с помощью результирующего набора. Например, предположим, что необходимо сформировать перекрестную таблицу (опорную таблицу) на основе ряда значений. Допустим, что этот ряд значений состоит из трех столбцов — столбца ключа, столбца подключа и столбца самих значений данных. Необходимо составить перекрестную таблицу, в которой значения ключей расположены по оси X, значения подключей — по оси Y, а значения данных приведены в точках, координатами которых являются ключи и подключи. Каждый ключ может иметь различное количество подключей, а сами подключи могут располагаться последовательно или непоследовательно. В листинге 14.5 представлена реализация подхода, в котором используется курсор для создания динамического кода T-SQL, с помощью которого формируется подобная перекрестная таблица.

Листинг 14.5. Использование курсора для формирования перекрестной таблицы

```
CREATE TABLE #series
(key1 int,
  key2 int,
  value1 decimal(6,2) DEFAULT (
    (CASE (CAST(RAND()+.5 AS int)*-1) WHEN 0 THEN 1
    ELSE -1 END)*(CONVERT(int, RAND() * 100000) % 10000)*RAND()
    )
)

INSERT #series (key1, key2) VALUES (1,1)
INSERT #series (key1, key2) VALUES (1,2)
INSERT #series (key1, key2) VALUES (1,3)
INSERT #series (key1, key2) VALUES (1,4)
INSERT #series (key1, key2) VALUES (1,5)
INSERT #series (key1, key2) VALUES (1,6)
INSERT #series (key1, key2) VALUES (2,1)
INSERT #series (key1, key2) VALUES (2,2)
INSERT #series (key1, key2) VALUES (2,3)
INSERT #series (key1, key2) VALUES (2,4)
INSERT #series (key1, key2) VALUES (2,5)
INSERT #series (key1, key2) VALUES (2,6)
```

```
INSERT #series (key1, key2) VALUES (2,7)
INSERT #series (key1, key2) VALUES (3,1)
INSERT #series (key1, key2) VALUES (3,2)
INSERT #series (key1, key2) VALUES (3,3)

DECLARE s CURSOR
FOR
SELECT DISTINCT key2 FROM #series ORDER BY key2

DECLARE @key2 int, @key2str varchar(10), @sql varchar(8000)

OPEN s
FETCH s INTO @key2

SET @sql=''
WHILE (@@FETCH_STATUS=0) BEGIN
  SET @key2str=CAST(@key2 AS varchar)
  SET @sql=@sql+',SUM(CASE WHEN key2='+@key2str+' THEN value1
  ELSE NULL END) ['+@key2str+']'
  FETCH s INTO @key2
END

SET @sql='SELECT key1'+@sql+' FROM #series GROUP BY key1'
EXEC(@sql)

CLOSE s
DEALLOCATE s
DROP TABLE #series
```

```
key1 1        2        3        4      5        6        7
---- -------- -------- -------- ------ -------- -------- -------
1    212.74    -1608.59 1825.29  690.48 1863.44  5302.54  NULL
2    -7531.42 1848.63  -3746.60 -54.37 -2263.63 -1014.01 5453.57
3    126.13   -10.41   205.35   NULL   NULL     NULL      NULL
```

Чтобы лучше понять, как организована работа приведенного в листинге кода, целесообразно вначале ознакомиться с самим динамическим запросом. Ниже показано, какой код содержится в переменной `@sql` непосредственно перед вызовом этого кода на выполнение (листинг 14.6).

Листинг 14.6. Текст динамического запроса непосредственно перед его выполнением

```
SELECT key1,SUM(CASE WHEN key2=1 THEN value1 ELSE NULL END) [1],
SUM(CASE WHEN key2=2 THEN value1 ELSE NULL END) [2],
SUM(CASE WHEN key2=3 THEN value1 ELSE NULL END) [3],
SUM(CASE WHEN key2=4 THEN value1 ELSE NULL END) [4],
SUM(CASE WHEN key2=5 THEN value1 ELSE NULL END) [5],
SUM(CASE WHEN key2=6 THEN value1 ELSE NULL END) [6],
SUM(CASE WHEN key2=7 THEN value1 ELSE NULL END) [7]
FROM #series GROUP BY key1
```

Курсор возвращает по одной строке для каждого уникального подключа в ряде подключей. Независимо от того, в каком ключе содержится подключ, если некоторый подключ появляется в таблице, происходит возврат одного экземпляра этого подключа из курсора с помощью оператора `SELECT DISTINCT`. Оператор

CASE, который формируется для каждого столбца перекрестной таблицы, возвращает столбец value1, если подключ соответствует данному столбцу, а в противном случае возвращает NULL. Конструкция GROUP BY группирует строки, возвращенные запросом, таким образом, что каждый ключ присутствует в результатах один и только один раз. Чтобы лучше понять, что при этом происходит, рассмотрим перекрестную таблицу, при формировании которой не использовалась конструкция GROUP BY (листинг 14.7).

Листинг 14.7. Результаты выполнения запроса, предназначенного для формирования перекрестной таблицы, в котором не использовалась конструкция GROUP BY

```
key1  1         2         3         4         5         6         7
----- --------- --------- --------- --------- --------- --------- -----
1     212.74    NULL      NULL      NULL      NULL      NULL      NULL
1     NULL      -1608.59  NULL      NULL      NULL      NULL      NULL
1     NULL      NULL      1825.29   NULL      NULL      NULL      NULL
1     NULL      NULL      NULL      690.48    NULL      NULL      NULL
1     NULL      NULL      NULL      NULL      5302.54   NULL      NULL
1     NULL      NULL      NULL      NULL      NULL       5302.54  NULL
2     -7531.42  NULL      NULL      NULL      NULL      NULL      NULL
2     NULL      1848.63   NULL      NULL      NULL      NULL      NULL
2     NULL      NULL      -3746.60  NULL      NULL      NULL      NULL
2     NULL      NULL      NULL      -54.37    NULL      NULL      NULL
2     NULL      NULL      NULL      NULL      -2263.63  NULL      NULL
2     NULL      NULL      NULL      NULL      NULL       -1014.01 NULL
2     NULL      NULL      NULL      NULL      NULL      NULL     5453.57
3     126.13    NULL      NULL      NULL      NULL      NULL      NULL
3     NULL      -10.41    NULL      NULL      NULL      NULL      NULL
3     NULL      NULL      205.35    NULL      NULL      NULL      NULL
```

Значение имеется только в одном столбце подключа каждой строки ключа, поскольку именно такую структуру имеют первоначальные данные рассматриваемого ряда значений. Значениям остальных столбцов в соответствующих выражениях CASE присваивается NULL. Конструкция GROUP BY сводит количество указанных значений NULL к минимуму, поскольку опорная таблица подытоживается таким образом, что в соответствующем столбце подключа каждое значение ряда появляется лишь при том условии, что это значение присутствует в наборе данных.

Операции, ориентированные на обработку строк

Еще одной подходящей областью применения курсоров являются операции, ориентированные на обработку строк. Так называются операции, превосходящие возможности обработки с помощью отдельных операторов (например, оператора SELECT). Некоторые особенности операций, ориентированных на обработку строк, таковы, что для них требуются более сложные или более гибкие средства обработки данных по сравнению с тем, что может предоставить решение на основе отдельных операторов. В листинге 14.8 приведен пример операции, ориентированной на обработку строк, которая выводит на внешнее устройство исходный код триггеров, подключенных к каждой таблице в базе данных.

Листинг 14.8. Использование курсора в операции, ориентированной на обработку строк

```
USE pubs
DECLARE objects CURSOR
FOR
SELECT name, deltrig, instrig, updtrig
FROM sysobjects WHERE type='U' AND deltrig+instrig+updtrig>0

DECLARE @objname sysname, @deltrig int, @instrig int, @updtrig int,
  @deltrigname sysname, @instrigname sysname, @updtrigname sysname

OPEN objects
FETCH objects INTO @objname, @deltrig, @instrig, @updtrig

WHILE (@@FETCH_STATUS=0) BEGIN
  PRINT 'Triggers for object: '+@objname
  SELECT @deltrigname=OBJECT_NAME(@deltrig),
    @instrigname=OBJECT_NAME(@instrig),
    @updtrigname=OBJECT_NAME(@updtrig)
IF @deltrigname IS NOT NULL BEGIN
     PRINT 'Table: '+@objname+' Delete Trigger: '+@deltrigname
     EXEC sp_helptext @deltrigname
  END
  IF @instrigname IS NOT NULL BEGIN
     PRINT 'Table: '+@objname+' Insert Trigger: '+@instrigname
     EXEC sp_helptext @instrigname
  END
  IF @updtrigname IS NOT NULL BEGIN
     PRINT 'Table: '+@objname+' Update Trigger: '+@updtrigname
     EXEC sp_helptext @updtrigname
  END
  FETCH objects INTO @objname, @deltrig, @instrig, @updtrig
END

CLOSE objects
DEALLOCATE objects

Triggers for object: employee
Table: employee Insert Trigger: employee_insupd
Text
------------------------------------------------------------------------
CREATE TRIGGER employee_insupd
ON employee
FOR insert, UPDATE
AS
--Get the range of level for this job type from the jobs table.
declare @min_lvl tinyint,
  @max_lvl tinyint,
  @emp_lvl tinyint,
  @job_id smallint
select @min_lvl = min_lvl,
  @max_lvl = max_lvl,
  @emp_lvl = i.job_lvl,
  @job_id = i.job_id
from employee e, jobs j, inserted i
where e.emp_id = i.emp_id AND i.job_id = j.job_id
```

```
IF (@job_id = 1) and (@emp_lvl <> 10)
begin
  raiserror ('Job id 1 expects the default level of 10.',16,1)
  ROLLBACK TRANSACTION
end
ELSE
IF NOT (@emp_lvl BETWEEN @min_lvl AND @max_lvl)
begin
  raiserror ('The level for job_id:%d should be
     between %d and %d.',
     16, 1, @job_id, @min_lvl, @max_lvl)
  ROLLBACK TRANSACTION
end

Table: employee Update Trigger: employee_insupd
Text
------------------------------------------------------------------------
CREATE TRIGGER employee_insupd
ON employee
FOR insert, UPDATE
AS
--Get the range of level for this job type from the jobs table.
declare @min_lvl tinyint,
  @max_lvl tinyint,
  @emp_lvl tinyint,
  @job_id smallint
select @min_lvl = min_lvl,
  @max_lvl = max_lvl,
  @emp_lvl = i.job_lvl,
  @job_id = i.job_id
from employee e, jobs j, inserted i
where e.emp_id = i.emp_id AND i.job_id = j.job_id
IF (@job_id = 1) and (@emp_lvl <> 10)
begin
  raiserror ('Job id 1 expects the default level of 10.',16,1)
  ROLLBACK TRANSACTION
end
ELSE
IF NOT (@emp_lvl BETWEEN @min_lvl AND @max_lvl)
begin
  raiserror ('The level for job_id:%d should be
     between %d and %d.',
     16, 1, @job_id, @min_lvl, @max_lvl)
  ROLLBACK TRANSACTION
end
```

Безусловно, можно было бы для получения той же информации выполнить запрос непосредственно к таблице syscomments и соединить полученные результаты с данными из таблицы sysobjects, но при этом результирующий набор не был бы отформатирован соответствующим образом. Осуществление итерации в таблице каждый раз по одной строке позволяет отформатировать вывод, относящийся к каждой таблице и ее триггерам, в любом желаемом виде.

Прокручиваемые формы

Ответ на вопрос о том, следует ли использовать курсор в качестве основы для прокручиваемой формы, в основном зависит от того, сколько данных может потребоваться для такой формы. Курсоры, оформленные на языке Transact-SQL, находятся непосредственно на сервере и возвращают только те строки, которые были в них выбраны, поэтому позволяют экономить значительный объем времени и ресурсов при обработке больших результирующих наборов. При использовании курсоров не приходится передавать в клиентское приложение по сети результирующий набор, состоящий, например, из 100 тыс. строк. С другой стороны, при наличии меньших результирующих наборов курсоры не нужны, поэтому вряд ли оправдывают усилия, связанные с их применением. Еще один фактор, который следует проанализировать при определении того, является ли курсор подходящим для некоторой прокручиваемой формы, состоит в том, обновляема ли форма и нужно ли в ней сразу же показывать изменения, внесенные другими пользователями. Если форма предназначена только для чтения или не нужно заботиться о том, чтобы в ней отображались изменения, внесенные другими пользователями, то, по-видимому, можно обойтись без использования курсора.

Синтаксические конструкции для работы с курсорами, применяемые в языке Transact-SQL

Для работы с курсорами предназначен целый ряд команд и функций, которые перечислены в табл. 14.2. Эти команды и функции более подробно рассматриваются в следующих подразделах.

Таблица 14.2. Команды и функции языка Transact-SQL для работы с курсорами

Команда или функция	Назначение
DECLARE CURSOR	Определить курсор
OPEN	Открыть курсор так, чтобы можно было осуществлять выборку из него данных
FETCH	Выбрать одну строку из курсора
CLOSE	Закрыть курсор, оставив нетронутыми внутренние структуры, которые применяются для обслуживания этого курсора
DEALLOCATE	Освободить внутренние структуры курсора
@@CURSOR_ROWS	Возвратить количество строк, доступ к которым предоставляется курсором
@@FETCH_STATUS	Показать, завершилось ли выполнение последнего оператора FETCH успехом или неудачей
CURSOR_STATUS	Сообщить информацию о состоянии, относящуюся к курсорам и переменным курсоров

Команда DECLARE CURSOR

Команда DECLARE CURSOR применяется для определения курсоров. Существуют две основные версии команды DECLARE CURSOR — команда с синтаксической структурой, совместимой с языком SQL 92 по стандарту ANSI/ISO, и команда с расширенной синтаксической структурой Transact-SQL. Синтаксическая структура ANSI/ISO выглядит следующим образом:

```
DECLARE name [INSENSITIVE][SCROLL] CURSOR
FOR select
[FOR {READ ONLY | UPDATE [OF column [,...n]]}]
```

Расширенная синтаксическая структура Transact-SQL имеет следующий вид:

```
DECLARE name CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR select
[FOR {READ ONLY | UPDATE [OF column [,...n]]}]
```

Компонент select этой команды представляет собой стандартный оператор SELECT, который определяет данные, возвращаемые курсором. В данную команду не разрешается вводить ключевые слова COMPUTE [BY], FOR BROWSE или INTO. От компонента select зависит, является ли курсор предназначенным только для чтения. Например, если в определение курсора включена конструкция FOR UPDATE, но задана команда select, которая, по существу, запрещает обновления (например, включает конструкции GROUP BY или DISTINCT), то курсор будет неявно преобразован в допускающий только чтение (или статический) курсор. Сервер преобразует в статические курсоры такие курсоры, которые по самому своему характеру не допускают обновление. Автоматические преобразования подобных типов называются неявными преобразованиями курсоров. При выполнении неявных преобразований курсоров учитывается целый ряд критериев; дополнительная информация приведена в оперативной документации Books Online.

Общий вывод из сказанного выше состоит в том, что нет необходимости задавать конструкцию FOR UPDATE, чтобы обеспечить обновление с помощью курсора, если заданный в нем оператор SELECT по существу является обновляемым. Еще раз отметим, что от характеристик оператора SELECT зависит то, является ли курсор обновляемым (если не указано иное). Соответствующий пример приведен в листинге 14.9.

Листинг 14.9. Демонстрация того, при каких условиях курсор является обновляемым

```
CREATE TABLE #temp (k1 int identity, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
```

```
DECLARE c CURSOR
FOR SELECT k1, c1 FROM #temp

OPEN c
FETCH c
UPDATE #temp
SET c1=2
WHERE CURRENT OF c

SELECT * FROM #temp
CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp

k1          c1
----------- -----------
1           NULL

k1          c1
----------- -----------
1           2
2           NULL
3           NULL
4           NULL
```

Даже несмотря на то, что приведенный в листинге 14.9 курсор не определен именно как обновляемый, он является обновляемым в силу того факта, что обновляем заданный в нем оператор SELECT; это означает, что сервер может легко преобразовать операцию обновления, применяемую к курсору, в операцию обновления какой-то конкретной строки в основополагающей таблице.

Если решено ввести конструкцию FOR UPDATE и включить в нее список столбцов, то в этом списке должен присутствовать обновляемый столбец (столбцы). При попытке обновить столбец, не заданный в списке столбцов, с помощью конструкции WHERE CURRENT OF оператора UPDATE, программа SQL Server отвергает такое изменение и вырабатывает сообщение об ошибке (листинг 14.10).

Листинг 14.10. Использование конструкции FOR UPDATE для указания на то, какие столбцы являются обновляемыми

```
CREATE TABLE #temp (k1 int identity, c1 int NULL, c2 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES

DECLARE c CURSOR
FOR SELECT k1, c1, c2 FROM #temp
FOR UPDATE OF c1

OPEN c
FETCH c
```

```
-- Ошибочный код T-SQL. В этом операторе UPDATE предпринимается попытка
-- внести изменение в столбец, не указанный в списке FOR UPDATE OF
UPDATE #temp
SET c2=2
WHERE CURRENT OF c

k1          c1          c2
----------- ----------- -----------
1           NULL        NULL

Server: Msg 16932, Level 16, State 1, Line 18
The cursor has a FOR UPDATE list and the requested column to be
updated is not in this list.
The statement has been terminated.
```

Если в операторе DECLARE CURSOR содержится оператор select, в котором имеется ссылка на переменную, то данная ссылка раскрывается во время обработки оператора объявления курсора, а не во время открытия курсора. Эту особенность важно учитывать, поскольку она говорит о том, что подобным переменным необходимо присваивать значения еще до объявления курсора, в котором используются эти переменные. Невозможно вначале объявить курсор, затем присвоить значение переменной, от которой зависит этот курсор, и надеяться, что курсор будет работать правильно. Соответствующий пример приведен в листинге 14.11.

Листинг 14.11. Использование переменных в операторе DECLARE CURSOR

```
-- Следующая операция предусмотрена на тот случай, что данный курсор
-- остался после выполнения предыдущего примера
DEALLOCATE c
DROP TABLE #temp
GO

CREATE TABLE #temp (k1 int identity, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES

DECLARE @k1 int

DECLARE c CURSOR
FOR SELECT k1, c1 FROM #temp WHERE k1<@k1 -- Операция не будет
              -- выполнена; в данной точке переменная @k1 имеет
              -- значение NULL

SET @k1=3 -- Этот указатель необходимо передвинуть до выполнения
          -- операции DECLARE CURSOR
OPEN c
FETCH c

UPDATE #temp
SET c1=2
```

```
WHERE CURRENT OF c

SELECT * FROM #temp
CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp

k1          c1
----------- -----------

Server: Msg 16930, Level 16, State 1, Line 18
The requested row is not in the fetch buffer.
The statement has been terminated.
k1          c1
----------- -----------
1            NULL
2            NULL
3            NULL
4            NULL
```

Сравнение глобальных и локальных курсоров

Глобальный курсор является видимым вне пределов пакета, хранимой процедуры или триггера, в котором он создан; глобальный курсор сохраняется до тех пор, пока не будет явно уничтожен, или не произойдет уничтожение того соединения, в котором он был определен. Локальный курсор является видимым только в пределах модуля кода, в котором он был создан (при условии, что локальный курсор не будет возвращен с помощью выходного параметра). Локальные курсоры уничтожаются неявно после их выхода из области определения.

Для совместимости с предыдущими выпусками в программе Server по умолчанию создаются глобальные курсоры, но опцию, применяемую по умолчанию, можно переопределить, явно задавая ключевое слово GLOBAL или LOCAL при объявлении курсора. Следует отметить, что можно одновременно объявлять и глобальный, и локальный курсоры с одинаковыми именами (но такой способ программирования является сомнительным). Например, код, приведенный в листинге 14.12, выполняется без ошибок.

Листинг 14.12. Применение локального и глобального курсоров с одинаковыми именами

```
DECLARE Darryl CURSOR  -- "Двоюродный брат" по имени Darryl
LOCAL
FOR SELECT stor_id, title_id, qty FROM sales

DECLARE Darryl CURSOR  -- Еще один "двоюродный брат" по имени Darryl
GLOBAL
FOR SELECT au_lname, au_fname FROM authors

OPEN GLOBAL Darryl
OPEN Darryl

FETCH GLOBAL Darryl
```

```
FETCH Darryl

CLOSE GLOBAL Darryl
CLOSE Darryl
DEALLOCATE GLOBAL Darryl
DEALLOCATE Darryl

au_lname                                 au_fname
---------------------------------------- --------------------
White                                    Johnson

stor_id title_id qty
------- -------- ------
6380    BU1032   5
```

Можно указать, должна ли программа SQL Server создавать глобальные курсоры, когда область действия не определена, с помощью системной процедуры sp_dboption (см. раздел "Настройка конфигурации курсоров" для получения дополнительной информации).

Команда OPEN

Команда OPEN обеспечивает доступ к строкам курсора с помощью оператора FETCH. Если открываемый курсор является курсором типа INSENSITIVE или STATIC, то команда OPEN копирует весь результирующий набор курсора во временную таблицу. Если же рассматриваемый курсор относится к типу KEYSET, то команда OPEN копирует набор значений уникальных ключей курсора (или все столбцы потенциального ключа, если уникальный ключ не существует) во временную таблицу. Команда OPEN позволяет указывать область действия курсора путем задания необязательного ключевого слова GLOBAL. Если имеется и локальный, и глобальный курсоры с одним и тем же именем (хотя применения одинаковых имен для двух разных курсоров следует избегать при любой возможности), то для обозначения открываемого глобального курсора используется ключевое слово GLOBAL. (Если явно не задано ни то, ни другое ключевое слово, для определения того, должен ли использоваться глобальный или локальный курсор, служит опция базы данных `default to local cursor`. См. раздел "Настройка конфигурации курсоров" для получения дополнительной информации.)

Для определения того, какое количество строк имеется в курсоре, применяется автоматически обновляемая переменная @@CURSOR_ROWS. Простой пример использования команды OPEN приведен в листинге 14.13.

Листинг 14.13. Использование команды OPEN

```
CREATE TABLE #temp (k1 int identity PRIMARY KEY, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
```

```
DECLARE GlobalCursor CURSOR STATIC  -- Объявить глобальный
                                    -- курсор GLOBAL
GLOBAL
FOR SELECT k1, c1 FROM #temp

DECLARE LocalCursor CURSOR STATIC -- Объявить локальный курсор LOCAL
LOCAL
FOR SELECT k1, c1 FROM #temp WHERE k1<4  -- Операция возвращает только
                                         -- три строки

OPEN GLOBAL GlobalCursor
SELECT @@CURSOR_ROWS AS NumberOfGLOBALCursorRows

OPEN LocalCursor
SELECT @@CURSOR_ROWS AS NumberOfLOCALCursorRows

CLOSE GLOBAL GlobalCursor
DEALLOCATE GLOBAL GlobalCursor
CLOSE LocalCursor
DEALLOCATE LocalCursor
GO
DROP TABLE #temp

NumberOfGLOBALCursorRows
------------------------
4

NumberOfLOCALCursorRows
-----------------------
3
```

Применительно к динамическим курсорам переменная @@CURSOR_ROWS возвращает значение -1, поскольку в результате ввода новых строк количество строк, возвращаемых курсором, может измениться в любое время. Если заполнение набора данных курсора осуществляется асинхронно (см. раздел "Настройка конфигурации курсоров"), то переменная @@CURSOR_ROWS возвращает отрицательное число, абсолютное значение которого показывает количество строк, находящееся в настоящее время в наборе данных курсора.

Оператор FETCH

Оператор FETCH представляет собой средство, с помощью которого осуществляется выборка данных из курсора. Этот оператор можно рассматривать как особую разновидность команды SELECT, которая возвращает только одну строку из заранее определенного результирующего набора. Как правило, оператор FETCH вызывается в цикле, в котором в качестве управляющей переменной используется переменная @@FETCH_STATUS, а каждый последующий оператор FETCH возвращает очередную строку курсора.

В прокручиваемых курсорах (в курсорах, относящихся к типу DYNAMIC, STATIC или KEYSET, либо в курсорах, объявленных с опцией SCROLL) допускается использовать оператор FETCH для выборки строк, отличных от следующей строки курсора. Прокручиваемые курсоры позволяют выбирать с помощью

оператора FETCH не только следующую, но и предыдущую строку курсора, первую строку, последнюю строку, строку, указанную по ее абсолютному номеру, или строку с номером, вычисляемым относительно номера текущей строки. Простой пример использования оператора FETCH приведен в листинге 14.14.

Листинг 14.14. Использование оператора FETCH

```
SET NOCOUNT ON
CREATE TABLE #cursortest (k1 int identity)

INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES

DECLARE c CURSOR SCROLL
FOR SELECT * FROM #cursortest

OPEN c

FETCH c -- Получить первую строку
FETCH ABSOLUTE 4 FROM c -- Получить четвертую строку
FETCH RELATIVE -1 FROM c -- Получить третью строку
FETCH LAST FROM c -- Получить последнюю строку
FETCH FIRST FROM c -- Получить первую строку

CLOSE c
DEALLOCATE c
GO
DROP TABLE #cursortest
k1
-----------
1

k1
-----------
4

k1
-----------
3

k1
-----------
10

k1
-----------
1
```

Оператор FETCH может использоваться непосредственно для получения данных из результирующего набора, но обычно с помощью этого оператора осуществляется заполнение локальных переменных данными из таблицы. Для присваивания локальным переменным значений, полученных с помощью оператора FETCH, применяется конструкция INTO, как показано в листинге 14.15.

Листинг 14.15. Пример использования конструкции INTO оператора FETCH

```
SET NOCOUNT ON
CREATE TABLE #cursortest (k1 int identity)

INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
INSERT #cursortest DEFAULT VALUES
DECLARE c CURSOR SCROLL
FOR SELECT * FROM #cursortest

DECLARE @k int

OPEN c
FETCH c INTO @k
WHILE (@@FETCH_STATUS=0) BEGIN
  SELECT @k
  FETCH c INTO @k
END

CLOSE c
DEALLOCATE c
GO
DROP TABLE #cursortest

-----------
1

-----------
2

-----------
3

-----------
4

-----------
5

-----------
```

```
6

-----------
7

-----------
8

-----------
9

-----------
10
```

По умолчанию в ходе выборки с помощью курсора применяется режим перемещения на следующую строку, определяемый с помощью опции NEXT, поэтому, если не указано, в каком режиме осуществляется выборка, то при выполнении очередной операции происходит выборка следующей строки курсора. Для выполнения операций выборки в режиме, отличном от NEXT, требуется указывать ключевое слово FROM.

Для обновления текущей записи можно использовать оператор FETCH RELATIVE 0. Этот оператор позволяет в ходе продвижения по набору данных курсора учитывать изменения, внесенные в текущую строку. Пример использования оператора FETCH RELATIVE 0 приведен в листинге 14.16.

Листинг 14.16. Использование оператора FETCH RELATIVE 0

```
USE pubs
SET CURSOR_CLOSE_ON_COMMIT OFF
  -- Предыдущая операция предусмотрена на тот случай, если данная опция
  -- была перед этим разрешена
SET NOCOUNT ON

DECLARE c CURSOR SCROLL
FOR SELECT title_id, qty FROM sales ORDER BY qty

OPEN c

BEGIN TRAN  -- Эта команда выполняется для того, чтобы можно было
            -- отменить внесенные изменения

PRINT 'Before image'

FETCH c

UPDATE sales
SET qty=4
WHERE qty=3 -- Как нам стало известно, критерию удовлетворяет только
            -- одна строка - первая
PRINT 'After image'
FETCH RELATIVE 0 FROM c
```

```
ROLLBACK TRAN -- Отменить результаты выполнения операции UPDATE

CLOSE c
DEALLOCATE c

Before image
title_id qty
-------- ------
PS2091    3

After image
title_id qty
-------- ------
PS2091    4
```

Оператор CLOSE

Оператор CLOSE удаляет текущий результирующий набор курсора и освобождает все блокировки, принадлежащие курсору. (В версиях программы SQL Server, предшествующих 7.0, все блокировки, включая блокировки курсоров, сохранялись до завершения текущей транзакции. В версии 7.0 и последующих версиях блокировки курсоров обрабатываются независимо от блокировок других типов.) Сами структуры данных курсора остаются неизменными для того, чтобы курсор мог быть открыт повторно в случае необходимости. Для указания на то, что происходит закрытие глобального курсора, необходимо задать ключевое слово GLOBAL.

Оператор DEALLOCATE

После завершения работы с курсором необходимо всегда его уничтожать. Курсор занимает место в процедурном кэше, которое можно использовать для других целей, после того как будет уничтожен курсор, который больше не требуется. Уничтожение курсора с помощью оператора DEALLOCATE приводит к его автоматическому закрытию. Тем не менее курсор следует всегда предварительно закрывать с помощью команды CLOSE, поскольку способ организации программы, в котором уничтожается открытый курсор, считается неправильным.

Настройка конфигурации курсоров

Для настройки конфигурации курсоров в языке Transact-SQL применяются не только опции объявлений, но также команды и опции настройки конфигурации, позволяющие модифицировать действия, выполняемые курсором. Для определения способа создания курсоров и регламентации действий, выполняемых курсорами после их создания, используются процедуры sp_configure и sp_dboption, а также опции команды SET.

Асинхронные курсоры

По умолчанию в программе SQL Server все ключевые наборы формируются в синхронном режиме; это означает, что из вызова команды OPEN не выполняется возврат до тех пор, пока результирующий набор курсора не будет полностью материализован (оформлен в виде таблицы). Такая организация работы может оказаться неоптимальной применительно к большим наборам данных, поэтому можно перейти к использованию другой организации работы с помощью опции настройки конфигурации cursor threshold процедуры sp_configure (опция cursor threshold относится к числу дополнительных опций; чтобы получить к ней доступ, необходимо разрешить применение дополнительных опций с помощью опции show advanced options процедуры sp_configure). Ниже приведен пример, который показывает, какое изменение в работе программы происходит после того, как для обработки курсора разрешается применять асинхронный режим (листинг 14.17).

Листинг 14.17. Использование асинхронных курсоров

```
-- Разрешить использование расширенных опций, чтобы можно было
-- выполнить настройку конфигурации с помощью параметра cursor
-- threshold
EXEC sp_configure 'show advanced options',1
RECONFIGURE WITH OVERRIDE

USE northwind

DECLARE c CURSOR STATIC -- Применить возможность скопировать строки
                        -- в tempdb
FOR SELECT OrderID, ProductID FROM [Order Details]

DECLARE @start datetime
SET @start=getdate()

-- Вначале попытаться выполнить обработку с помощью синхронного курсора
OPEN c

PRINT CHAR(13) -- Подготовить терминал к выводу данных
SELECT DATEDIFF(ms,@start,getdate()) AS
  [Milliseconds elapsed for Synchronous cursor]

SELECT @@CURSOR_ROWS AS [Number of rows in Synchronous cursor]

CLOSE c

-- Теперь изменить настройку конфигурации с помощью параметра cursor
-- threshold и обеспечить применение асинхронного курсора
EXEC sp_configure 'cursor threshold', 1000
-- Асинхронный режим работы применяется для курсоров с количеством
-- строк больше одной тысячи
RECONFIGURE WITH OVERRIDE
PRINT CHAR(13) -- Подготовить терминал к выводу данных

SET @start=getdate()
OPEN c -- Курсор открывается в асинхронном режиме, поскольку количество
```

```
      -- строк в таблице больше одной тысячи

-- Команда OPEN немедленно выполняет возврат, поскольку заполнение
-- курсора происходит в асинхронном режиме
SELECT DATEDIFF(ms,@start,getdate()) AS
  [Milliseconds elapsed for Asynchronous cursor]

SELECT @@CURSOR_ROWS AS [Number of rows in Asynchronous cursor]

CLOSE c

DEALLOCATE c
GO
EXEC sp_configure 'cursor threshold', -1 -- Возвращение в синхронный
                                         -- режим
RECONFIGURE WITH OVERRIDE

DBCC execution completed. If DBCC printed error messages, contact
your system administrator.
Configuration option changed. Run the RECONFIGURE statement
to install.

Milliseconds elapsed for Synchronous cursor
-------------------------------------------
70

Number of rows in Synchronous cursor
------------------------------------
2155

DBCC execution completed. If DBCC printed error messages, contact
your system administrator.
Configuration option changed. Run the RECONFIGURE statement
to install.

Milliseconds elapsed for Asynchronous cursor
--------------------------------------------
0
Number of rows in Asynchronous cursor
-------------------------------------
-1

DBCC execution completed. If DBCC printed error messages, contact
your system administrator.
Configuration option changed. Run the RECONFIGURE statement
to install.
```

Способ автоматического закрытия курсоров, определяемый стандартом ANSI/ISO

В спецификации языка SQL-92, определяемой стандартом ANSI/ISO, указано, что курсоры должны закрываться автоматически после завершения транзакции. Такая организация работы не совсем применима для приложений такого

типа, в которых курсоры используются наиболее часто (например, в приложениях с прокручиваемыми формами), поэтому в программе SQL Server указанный стандарт в обычных условиях не поддерживается. По умолчанию курсор SQL Server остается открытым до его явно закрытия или до разрыва соединения, в котором был создан этот курсор. Чтобы вынудить программу SQL Server закрывать курсоры после фиксации транзакции, можно применить команду SET CURSOR_CLOSE_ON_COMMIT, как показано в листинге 14.18.

Листинг 14.18. Пример организации работы, в которой курсор закрывается после фиксации транзакции

```
CREATE TABLE #temp (k1 int identity PRIMARY KEY, c1 int NULL)

INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES
INSERT #temp DEFAULT VALUES

DECLARE c CURSOR DYNAMIC
FOR SELECT k1, c1 FROM #temp

OPEN c

SET CURSOR_CLOSE_ON_COMMIT ON
BEGIN TRAN

UPDATE #temp
SET c1=2
WHERE k1=1
COMMIT TRAN

-- Эти операции FETCH окончатся неудачей, поскольку курсор был закрыт
-- оператором COMMIT
FETCH c
FETCH LAST FROM c

-- Эта операция CLOSE окончится неудачей, поскольку курсор был закрыт
-- оператором COMMIT
CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp
SET CURSOR_CLOSE_ON_COMMIT OFF

Server: Msg 16917, Level 16, State 2, Line 0
Cursor is not open.
Server: Msg 16917, Level 16, State 2, Line 26
Cursor is not open.
Server: Msg 16917, Level 16, State 1, Line 29
Cursor is not open.
```

Вопреки тому, что сказано в оперативной документации Books Online, откат транзакции не приводит к закрытию обновляемых курсоров, если отменена опция CLOSE_CURSOR_ON_COMMIT. На самом деле действия, осуществляемые вслед за выполнением команды ROLLBACK, значительно отличаются от описанных в документации и в большей степени напоминают действия, происходящие при фиксации транзакции. По сути, команда ROLLBACK не закрывает курсоры, если не разрешена опция CLOSE_CURSOR_ON_COMMIT. Соответствующий пример приведен в листинге 14.19.

Листинг 14.19. Пример использования команды ROLLBACK и закрытия курсора

```
USE pubs
SET CURSOR_CLOSE_ON_COMMIT ON
BEGIN TRAN

DECLARE c CURSOR DYNAMIC
FOR SELECT qty FROM sales

OPEN c

FETCH c

SET qty=qty+1UPDATE sales
WHERE CURRENT OF c

ROLLBACK TRAN

-- Эти операции FETCH окончатся неудачей, поскольку курсор был закрыт
-- оператором ROLLBACK
FETCH c
FETCH LAST FROM c

-- Эта операция CLOSE окончится неудачей, поскольку курсор был закрыт
-- оператором ROLLBACK
CLOSE c
DEALLOCATE c
GO
SET CURSOR_CLOSE_ON_COMMIT OFF

qty
------
5

Server: Msg 16917, Level 16, State 2, Line 21
Cursor is not open.
Server: Msg 16917, Level 16, State 2, Line 22
Cursor is not open.
Server: Msg 16917, Level 16, State 1, Line 25
Cursor is not open.
```

Теперь рассмотрим, что произойдет после отмены опции CURSOR_CLOSE_ON_COMMIT и повторного выполнения запроса (листинг 14.20).

Листинг 14.20. Еще один пример использования команды ROLLBACK и закрытия курсора

```
SET CURSOR_CLOSE_ON_COMMIT OFF
BEGIN TRAN

DECLARE c CURSOR DYNAMIC
FOR SELECT qty FROM sales FOR UPDATE OF qty

OPEN c
FETCH c

UPDATE sales
SET qty=qty+1
WHERE CURRENT OF c

ROLLBACK TRAN

-- Эти операции FETCH завершатся успешно, поскольку курсор остался
-- открытым, несмотря на выполненную операцию ROLLBACK
FETCH c
FETCH LAST FROM c

-- Эта операция CLOSE завершится успешно, поскольку курсор остался
-- открытым, несмотря на выполненную операцию ROLLBACK
CLOSE c
DEALLOCATE c

qty
------
5

qty
------
3

qty
------
30
```

Несмотря на то что при открытом динамическом курсоре произошел откат транзакции, сам курсор остался незатронутым. Следовательно, действия, выполняемые сервером, фактически происходят иначе, чем указано в документации.

Определение применяемого по умолчанию режима создания глобальных или локальных курсоров

В программе SQL Server непосредственно после ввода ее в действие по умолчанию предусматривается создание глобальных курсоров. Такой подход предусмотрен для обеспечения совместимости с предыдущими версиями сервера, в которых не поддерживались локальные курсоры. Если требуется изменить указанное значение опции конфигурации, то необходимо присвоить истинное значение опции `default to local cursor` базы данных с помощью процедуры `sp_dboption`.

Обновление курсоров

Для обновления и удаления с помощью курсора может применяться конструкция WHERE CURRENT OF операторов UPDATE и DELETE. Обновление или удаление, осуществляемое с помощью курсора, принято называть позиционированным обновлением или удалением. Соответствующие примеры приведены в листинге 14.21.

Листинг 14.21. Выполнение позиционированного обновления и удаления

```
USE pubs
SET CURSOR_CLOSE_ON_COMMIT OFF

SET NOCOUNT ON
DECLARE C CURSOR DYNAMIC
FOR SELECT * FROM sales

OPEN c

FETCH c

BEGIN TRAN -- Запустить транзакцию для того, чтобы можно было отменить
           -- внесенные изменения

-- Позиционированная операция UPDATE
UPDATE sales SET qty=qty+1 WHERE CURRENT OF c

FETCH RELATIVE 0 FROM c

FETCH c

-- Позиционированная операция DELETE
DELETE sales WHERE CURRENT OF c

SELECT * FROM sales WHERE qty=3

ROLLBACK TRAN -- Отменить внесенные изменения

SELECT * FROM sales WHERE qty=3 -- Удаленная строка восстанавливается

CLOSE c
DEALLOCATE c
stor_id ord_num  ord_date                qty    payterms  title_id
------- -------- ----------------------- ------ --------- --------
6380    6871     1994-09-14 00:00:00.000 5      Net 60    BU1032

stor_id ord_num  ord_date                qty    payterms  title_id
------- -------- ----------------------- ------ --------- --------
6380    6871     1994-09-14 00:00:00.000 6      Net 60    BU1032

stor_id ord_num  ord_date                qty    payterms  title_id
------- -------- ----------------------- ------ --------- --------
6380    722a     1994-09-13 00:00:00.000 3      Net 60    PS2091

stor_id ord_num  ord_date                qty    payterms  title_id
```

```
------- -------- ------------------------ ----- --------- --------

stor_id ord_num  ord_date                 qty    payterms  title_id
------- -------- ------------------------ ----- --------- --------
6380    722a     1994-09-13 00:00:00.000  3      Net 60    PS2091
```

Переменные курсора

Язык Transact-SQL позволяет определять переменные, которые содержат указатели на курсоры, соответствующие типу данных курсора. Ссылки на переменные курсоров, а также на имена курсоров могут быть указаны в командах OPEN, FETCH, CLOSE и DEALLOCATE. Существует также возможность задавать переменные в хранимых процедурах, содержащих определения курсоров, и возвращать курсоры, созданные в хранимой процедуре, с помощью выходного параметра. Такая возможность возврата результатов вызывающему программному объекту в эффективной и модульной форме используется в некоторых собственных процедурах программы SQL Server (например, sp_cursor_list, sp_describe_cursor, sp_fulltext_tables_cursor и т.д.). Следует отметить, что курсор нельзя передавать в процедуру с помощью какого-либо входного параметра, можно только возвращать курсоры с помощью выходных параметров. Кроме того, нельзя определять столбцы таблицы с помощью типа данных курсора (разрешается использовать только переменные), а также нельзя присваивать значение переменной курсора с помощью оператора SELECT (а также с помощью скалярных переменных); для этой цели необходимо применять команду SET.

Возможность получать доступ к курсорам с помощью выходных параметров представляет собой значительное усовершенствование по сравнению с традиционным подходом, основанным на использовании результирующего набора, поскольку выходные параметры предоставляют вызывающему программному объекту больший контроль над тем, как должны обрабатываться строки, возвращенные процедурой. При желании можно немедленно приступить к обработке курсора, рассматривая его наряду с традиционным результирующим набором, или сохранить курсор для применения в будущем. До появления переменных курсора единственный способ достижения такой же степени гибкости состоял в том, чтобы представить результирующий набор, возвращенный хранимой процедурой, в виде таблицы, а затем обрабатывать эту таблицу по мере необходимости. Такой подход вполне себя оправдывал применительно к простым и небольшим результирующим наборам, но мог стать источником проблем при увеличении объема результирующих наборов.

Для проверки выходного параметра с переменной курсора можно использовать функцию CURSOR_STATUS. Эта функция позволяет определить, ссылается ли эта переменная на открытый курсор, и узнать, к какому количеству строк предоставляет доступ рассматриваемый курсор. Ниже приведен пример, в котором применяются переменные курсора, выходные параметры и функция CURSOR_STATUS (листинг 14.22).

Листинг 14.22. Использование переменных курсора

```
CREATE PROC listsales_cur @title_id tid,
    @salescursor cursor varying OUT
AS
-- Объявить курсор LOCAL для того, чтобы он автоматически освобождался
-- после выхода из области определения
DECLARE c CURSOR DYNAMIC
LOCAL
FOR SELECT * FROM sales WHERE title_id LIKE @title_id

DECLARE @sc cursor    -- Переменная локального курсора.
SET @sc=c             -- Теперь имеются две ссылки на курсор

OPEN c

FETCH @sc

SET @salescursor=@sc -- Возвратить курсор с помощью выходного параметра
RETURN 0
GO

SET NOCOUNT ON
-- Объявить переменную локального курсора для получения выходного
-- параметра
DECLARE @mycursor cursor

EXEC listsales_cur 'BU1032', @mycursor OUT -- Вызвать процедуру

-- Убедиться в том, что возвращенный курсор открыт и содержит по
-- меньшей мере одну строку
IF (CURSOR_STATUS('variable','@mycursor')=1) BEGIN
  FETCH @mycursor
  WHILE (@@FETCH_STATUS=0) BEGIN
    FETCH @mycursor
  END
END
CLOSE @mycursor
DEALLOCATE @mycursor
```

```
stor_id ord_num  ord_date                 qty  payterms    title_id
------- -------- ----------------------- ---- ---------- --------
6380    6871     1994-09-14 00:00:00.000 5    Net 60      BU1032

stor_id ord_nu   ord_date                 qty  payterms    title_id
------- -------- ----------------------- ---- ---------- --------
8042    423LL930 1994-09-14 00:00:00.000 10    ON invoice BU1032

stor_id ord_num  ord_date                 qty  payterms    title_id
------- -------- ----------------------- ---- ---------- --------
8042    QA879.1  1999-06-24 19:13:26.230 30   Net 30      BU1032

stor_id ord_num  ord_date                 qty  payterms    title_id
------- -------- ----------------------- ---- ---------- --------
```

Обратите внимание на то, что в этом примере кода для формирования ссылок на курсор используется способ, в котором предусмотрено применение трех различных переменных, а также первоначального имени курсора. Для любой команды, кроме DEALLOCATE, ссылка на переменную курсора равнозначна указанию курсора по имени. При его открытии курсор успешно открывается независимо от того, применяется ли для ссылки на него переменная курсора или само имя курсора, после чего может осуществляться выборка строк с использованием любой переменной, которая ссылается на этот курсор. Отличительная особенность команды DEALLOCATE состоит в том, что она фактически не уничтожает курсор, если применяемая в ней ссылка на курсор не является последней. Тем не менее выполнение команды DEALLOCATE приводит к тому, что предотвращается дальнейший доступ к курсору с использованием заданного в этой команде идентификатора курсора. Поэтому, если в программе имеется курсор с именем `foo` и переменная курсора с именем `foovar`, которой присвоена ссылка на курсор `foo`, то удаление курсора `foo` с помощью команды DEALLOCATE не влечет за собой никаких последствий, кроме запрещения доступа к курсору с помощью имени `foo`, а переменная `foovar` остается неподверженной действию этой команды.

Хранимые процедуры для работы с курсорами

В программе SQL Server предусмотрен целый ряд хранимых процедур, относящихся к курсорам, с которыми читатель должен ознакомиться самостоятельно, если предполагает, что ему в дальнейшей работе придется часто сталкиваться с курсорами. Сокращенный список указанных процедур наряду с описанием каждой из них приведен в табл. 14.3.

Таблица 14.3. Хранимые процедуры, связанные с курсорами

Процедура	Назначение
`sp_cursor_list`	Возвратить список курсоров и их атрибутов, которые были открыты в соединении
`sp_describe_cursor`	Вывести список атрибутов отдельного курсора
`sp_describe_cursor_columns`	Вывести список столбцов (и атрибутов столбцов), возвращенных курсором
`sp_describe_cursor_tables`	Возвратить список таблиц, на которые указывает курсор

Каждая из этих процедур возвращает свои результаты с помощью выходного параметра курсора, поэтому, чтобы воспользоваться этими процедурами, необходимо передать в них переменную локального курсора.

Оптимизация производительности курсоров

Лучшая рекомендация по повышению производительности, касающаяся курсоров, состоит в том, что курсоры вообще не следует использовать, если можно этого избежать. Как было сказано выше, программа SQL Server работает гораздо лучше с наборами данных, чем с отдельными строками. Дело в том, что программа SQL Server – это реляционная СУБД, а однострочный доступ никогда не был сильной стороной таких СУБД. Несмотря на сказанное, в определенных обстоятельствах невозможно обойтись без использования курсора, поэтому ниже приведено несколько рекомендаций по оптимизации производительности курсоров.

- Используйте статические курсоры (т.е. курсоры, невосприимчивые к изменениям), только если они действительно необходимы. Открытие статического курсора приводит к тому, что все относящиеся к нему строки копируются во временную таблицу. Именно поэтому курсоры указанного типа невосприимчивы к изменениям – они фактически ссылаются на копию основополагающей таблицы, находящейся в базе данных `tempdb`. Разумеется, чем больше результирующий набор, тем выше вероятность, что объявление относящегося к нему статического курсора вызовет появление проблем, связанных с конкуренцией за ресурсы в базе данных `tempdb`.
- Используйте курсоры ключевого набора, только если они действительно необходимы. Как и для статических курсоров, открытие курсора ключевого набора приводит к созданию временной таблицы. Разумеется, такая таблица содержит только значения ключей из основополагающей таблицы (если не существует уникального ключа), но несмотря на это, может иметь весьма существенные размеры, если приходится работать с очень большими результирующими наборами.
- При работе по принципу перемещения только вперед с результирующими наборами, предназначенными только для чтения, используйте опцию курсора `FAST_FORWARD` вместо опции `FORWARD_ONLY`. Если при объявлении курсора применяется опция `FAST_FORWARD`, то создается курсор с опциями `FORWARD_ONLY, READ_ONLY`, в котором предусмотрен целый ряд внутренних средств оптимизации производительности.
- Объявляйте курсоры, предназначенные только для чтения, с помощью ключевого слова `READ_ONLY`. Это позволяет предотвратить непреднамеренное внесение изменений, а также дает возможность серверу определить, что курсор не изменяет строк, по которым проходит.
- Соблюдайте осторожность при модификации большого числа строк с помощью цикла, заданного на основе курсора и включенного в транзакцию. В зависимости от уровня изоляции транзакции, эти строки могут оставаться заблокированными до того, как произойдет фиксация или откат транзакции, что может вызвать конкуренцию за ресурсы в сервере.

- При работе с большими результирующими наборами продумайте вариант использования асинхронных курсоров, чтобы иметь возможность возвращать управление в вызывающий программный объект настолько быстро, насколько это возможно. Асинхронные курсоры особенно удобны, если нужно получить результирующий набор с заданными размерами, предназначенный для заполнения прокручиваемой формы, поскольку такие курсоры позволяют почти сразу же приступить к отображению строк в приложении.
- Соблюдайте осторожность при обновлении динамических курсоров, особенно если эти курсоры созданы на таблицах с неуникальными ключами кластеризованного индекса, поскольку такие операции обновления могут вызвать "проблему порочного круга" (Halloween problem) — повторного, ошибочного обновления одной и той же строки или строк. Программа SQL Server принудительно применяет внутреннее преобразование неуникальных ключей кластеризованного индекса в уникальные, добавляя к дублирующимся ключам суффикс в виде порядкового номера, поэтому существует вероятность того, что в результате обновления ключа строки будет присвоено значение, которое вынудит сервер добавить такой суффикс, под воздействием которого строка передвинется еще ниже в результирующем наборе. По мере дальнейшего продвижения по оставшейся части результирующего набора снова встретится эта строка, и описанная последовательность действий опять повторится, что приведет к возникновению бесконечного цикла. Пример проявления указанной проблемы приведен в листинге 14.23.

Листинг 14.23. Пример проблемы порочного круга

```
-- В этом коде создается курсор, при использовании которого возникает
-- проблема порочного круга.
-- Вызов этого кода на выполнение приводит к появлению бесконечного
-- цикла
SET NOCOUNT ON
CREATE TABLE #temp (k1 int identity, c1 int NULL)
CREATE CLUSTERED INDEX c1 ON #temp(c1)

INSERT #temp VALUES (8)
INSERT #temp VALUES (6)
INSERT #temp VALUES (7)
INSERT #temp VALUES (5)
INSERT #temp VALUES (3)
INSERT #temp VALUES (0)
INSERT #temp VALUES (9)
DECLARE c CURSOR DYNAMIC
FOR SELECT k1, c1 FROM #temp

OPEN c

FETCH c

WHILE (@@FETCH_STATUS=0) BEGIN
  UPDATE #temp
  SET c1=c1+1
```

```
  WHERE CURRENT OF c
  FETCH c
  SELECT * FROM #temp ORDER BY k1
END

CLOSE c
DEALLOCATE c
GO
DROP TABLE #temp
```

Резюме

Курсоры — не рекомендуемый способ решения большинства проблем выборки или обновления данных, поскольку курсоры в случае неправильного использования могут вызвать существенное снижение производительности. При выборе способа решения любой задачи обеспечения доступа к данным следует прежде всего искать возможность организовать работу приложения с помощью способа, который изначально реализован в проекте программы SQL Server, т.е. применить данные, представленные в виде наборов (прежде, чем прибегнуть к использованию языка Transact-SQL). Предусматривать использование курсоров следует лишь после того, как будет обнаружено, что все возможные исследованные варианты, основанные на наборах данных, являются менее приемлемыми по сравнению с вариантами, в которых должны использоваться курсоры.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. При переносе приложения ISAM в СУБД SQL Server следует пытаться внести в приложение минимально возможный объем изменений, особенно в части применяемого способа доступа к данным.
2. Какой элемент конструкции `WHERE` используется в команде `UPDATE` или `DELETE` для осуществления позиционированного обновления или удаления?
3. Возможно ли объявить в хранимой процедуре переменную, имеющую такой тип данных, как курсор?
4. Подтвердите или опровергните следующее утверждение. Чтобы возвратить курсор из хранимой процедуры, необходимо передать его в оператор возврата, предусмотренный в процедуре.
5. Перечислите четыре типа курсоров, поддерживаемых в программе SQL Server.
6. В чем состоит различие между командами `DEALLOCATE CURSOR` и `CLOSE` с точки зрения использования ресурса?
7. Какой механизм используется в программе SQL Server для хранения данных, возвращенных статическим курсором?

8. Какая автоматически сопровождаемая переменная обычно используется для управления циклом, в котором выполняются итерации в курсоре с использование команды FETCH?
9. Какое значение возвращает автоматически сопровождаемая переменная @@CURSOR_ROWS?
10. Какая функция может использоваться для проверки состояния курсора?

ГЛАВА 15

Средства ODSOLE

В этой главе речь идет об автоматизации COM-компонентов (т.е. об управлении COM-компонентами) с использованием средств связывания и внедрения объектов открытых служб данных (Open Data Services Object Linking and Embedding – ODSOLE) программы SQL Server. Средства ODSOLE реализованы в виде расширенных процедур sp_OA на языке Transact-SQL (таких как `sp_OACreate`, `sp_OAMethod` и т.д.). В данной главе в целом показано, как работают средства автоматизации Automation, и приведено несколько примеров реализации этих средств с помощью процедур sp_OA.

В настоящей главе приведено обновленное описание реализации средств Automation с помощью языка Transact-SQL и средств ODSOLE, которое впервые было включено в предыдущие книги автора. Как и при написании предыдущей главы, посвященной средствам SQLXML, автор решил привести в данной книге обновленное изложение информации практического назначения и вместе с тем подробнее остановиться на нюансах архитектуры программного обеспечения, которые не были отражены в книгах автора ранее. Читатели обычно стремятся найти в технической литературе информацию, позволяющую им лучше организовать свою работу; автор не считает себя вправе излагать только абстрактные сведения о деталях устройства программного обеспечения и игнорировать практические аспекты применения *самого* программного обеспечения. По мнению автора, ознакомление с тем, как проект влияет на все аспекты практического применения технологии, – превосходный способ понять все тонкости проектирования на интуитивном уровне, т.е. изучить рассматриваемую технологию буквально до мельчайших подробностей.

Краткий обзор

Как уже было сказано, первоначально аббревиатура ODSOLE расшифровывалась как Open Data Services Object Linking and Embedding (связывание и внедрение объектов открытых служб данных), но со временем смысл компонента "OLE" этого термина изменился. Вышел также из употребления сам термин "OLE Automation", и теперь применяется в основном термин "Automation".

Средства Automation представляют собой независимую от языка технологию управления и использования COM-объектов. Эта технология получила широкое распространение, поэтому во многих приложениях предусмотрен доступ к функциональным средствам этих и других приложений с помощью COM-интерфейсов. Доступ к тем или иным типам функциональных средств с помощью COM-объектов предоставляется во многих программных продуктах Microsoft, а также

во многих программных продуктах других поставщиков, которые предназначены для широкого круга пользователей. Эти объекты могут применяться для манипулирования базовым приложением через контроллер Automation, обладающий способностью взаимодействовать с COM-интерфейсом `IDispatch` приложения. К числу наиболее широко применяемых контроллеров Automation относятся Visual Basic и VBScript. Средство ODSOLE программы SQL Server представляет собой самостоятельный контроллер Automation, доступ к которому обеспечивают расширенные процедуры sp_OA, вызываемые из языка Transact-SQL.

Средства ODSOLE позволяют также пользователю создавать собственные COM-объекты и получать к ним доступ с помощью языка T-SQL. Благодаря этому можно заключать функциональные средства, не доступные в языке T-SQL, в COM-компоненты и вызывать эти средства из пакетов и хранимых процедур T-SQL.

COM-объекты и модели многопотоковой поддержки

Прежде чем приступить к изучению того, как организована работа с COM-объектами из языка Transact-SQL с помощью средств ODSOLE, рассмотрим некоторые основные сведения, касающиеся модели многопотоковой поддержки COM и организации параллельной работы. Ознакомление с тем, как организована многопотоковая поддержка применительно к средствам ODSOLE, и как осуществляется управление распараллеливанием работы объектов, позволяет лучше понять основы эффективного и безопасного применения средств ODSOLE.

COM-объекты поддерживают две основные модели многопотоковой обработки – модель однопотокового апартаментного контейнера (Single-Threaded Apartment – STA), называемую также "апартаментно-потоковой", и модель многопотокового апартаментного контейнера (MultiThreaded Apartment – MTA), называемую также "свободно-потоковой". Сам термин "апартаментный контейнер" не означает что-либо чрезвычайно сложное, а просто помогает определить концептуальную инфраструктуру, которая описывает связи между потоками, объектами и процессами. Апартаментный контейнер представляет собой именно то, что подразумевается под понятием апартаментов (или квартиры) – часть комнат в здании. В этой аналогии зданию уподобляется процесс, поэтому апартаменты – это просто логический контейнер в процессе. Апартаментный контейнер может содержать несколько потоков и/или объектов, в зависимости от его типа, а в одном процессе может находиться несколько отдельных апартаментных контейнеров.

Каждый объект и каждый поток может относиться только к одному апартаментному контейнеру. К объектам в данном конкретном апартаментном контейнере могут непосредственно получить доступ только потоки, находящиеся в том же контейнере; все прочие потоки должны обращаться к COM-посредникам того или иного типа. Поток может реализовать свое право на пребывание в некотором апартаментном контейнере (а также дополнительно указать, какая модель многопотоковой поддержки должна для него использоваться) с помощью вызова функции инициализации COM, такой как `CoInitialize`, `CoInitializeEx` или `OleInitialize`.

В модели STA в каждом апартаментном контейнере имеется единственный поток и могут содержаться несколько объектов, а в модели MTA в каждом апар-

таментном контейнере могут находиться несколько потоков и несколько объектов. В процессе можно определить несколько контейнеров STA, но только один контейнер MTA. Этот единственный контейнер MTA может сосуществовать с многочисленными контейнерами STA в одном и том же процессе.

Средства ODSOLE реализованы с использованием модели STA. Если вы подключитесь к программе SQL Server с помощью отладчика и установите точку останова на функции `OleInitialize` (которая определена в библиотеке `OLE32.DLL`), прежде чем сделать первый вызов процедуры sp_OA, то обнаружите, что в коде ODSOLE вызывается функция `OleInitialize`. В программной реализации функций `OleInitialize` предусмотрено использование модели STA, поэтому можно сделать вывод, что в средствах ODSOLE применяется модель STA. (После того как поток был инициализирован для использования в конкретной модели многопотоковой поддержки COM, с этой модели нельзя перейти на другую, не выполнив перед этим отмену инициализации COM.)

Как уже было сказано, внепроцессный COM-сервер (т.е. исполняемый файл) задает применяемую в нем модель многопотоковой поддержки с помощью вызова той или иной функции инициализации COM. Функция `CoInitializeEx` является единственным из трех основных инициализаторов COM, который позволяет указывать модель многопотоковой поддержки. Вызов функций `CoInitialize` и `OleInitialize` приводит к принудительному введению в действие модели STA. Вызов любой из функций `CoInitialize` и `OleInitialize` в конечном итоге приводит к такому вызову функции `CoInitializeEx`, в котором в качестве модели многопотоковой поддержки указана STA. Это означает, что вызов в программе SQL Server функции `OleInitialize` приводит к вызову функции `CoInitializeEx` с указанием STA в качестве модели многопотоковой поддержки.

Модель многопотоковой поддержки для внутрипроцессного COM-сервера (библиотеки DLL) не задается с помощью вызова функции `CoInitializeEx`. Вместо этого такая модель задается с помощью ключа реестра, например, следующим образом:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\InprocServer32\
↳ThreadingModel
```

Этот ключ может иметь значение `Apartment`, `Free` или `Both`. Если ключ не найден в системном реестре, то подразумевается использование модели STA.

Как уже было сказано, только потоки, находящиеся в апартаментном контейнере, где был создан COM-объект, могут непосредственно получить доступ к этому объекту. Другие потоки получают доступ к такому объекту через объекты-посредники. Учитывая то, что в каждом конкретном апартаментном контейнере STA находится только один поток, обязанности по синхронизации доступа к объекту со стороны других потоков должны взять на себя средства COM. Синхронизация осуществляется с помощью средств передачи сообщений операционной системы Windows. Средства COM создают скрытое окно для каждого апартаментного контейнера и определяют посредников, которые должны передавать сообщения в апартаментный контейнер, владеющий объектом, для того, чтобы вызвать этот объект. Управление сериализацией доступа к объекту осуществляется с помощью обычных средств ведения очереди оконных сообщений операционной системы

Windows. Методы объекта вызываются в ответ на сообщения, передаваемые в скрытое окно апартаментного контейнера. По мере того как сообщения извлекаются из очереди сообщений (с помощью функций `PeekMessage` и `GetMessage`) и доставляются в апартаментный контейнер (с помощью функции `DispatchMessage`), оконная процедура для данного потока, которая реализована с помощью средств COM, вызывает соответствующие методы объекта. После того как вызов метода завершается, и возникает необходимость возвратить результаты в вызывающий поток, указанная последовательность действий происходит в обратном порядке. В скрытое окно апартаментного контейнера вызывающего потока передаются сообщения, которые несут в себе полученный результат (результаты) и указывают на завершение вызова функции. Эти сообщения извлекаются вызывающим потоком, который, в свою очередь, выполняет возврат с помощью вызова метода объекта-посредника, завершая тем самым вызов метода в объекте, находящемся в другом апартаментном контейнере. Если настройка конфигурации некоторого компонента выполнена с расчетом на использование модели STA, то объекты, доступ к которым предоставляет этот компонент, создаются в рабочем потоке программы SQL Server, который носит имя `sp_OACreate`. Другие рабочие потоки не могут получить доступ к этим экземплярам объекта. Учитывая то, что процедура sp_OA так или иначе освобождает все созданные объекты после завершения обработки любого пакета (и тем самым гарантирует, что все вызовы данного конкретного объекта происходят в одном и том же физическом потоке), использование модели STA средствами ODSOLE осуществляется вполне успешно.

Если же настройка конфигурации компонента выполнена с учетом применения модели MTA, то средства COM автоматически запускают хост-контейнер MTA и создают в этом контейнере экземпляры объектов. А если настройка конфигурации компонента была выполнена с учетом использования модели многопотоковой поддержки, совместимой и с моделью STA, и с моделью MTA, то объект создается в контейнере STA вызывающего потока.

Главный контейнер STA

Первый поток, инициализирующий средства COM в процессе с использованием модели многопотоковой поддержки STA, становится потоком главного контейнера STA. Этот контейнер STA должен оставаться действующим до тех пор, пока не завершится вся работа, связанная с использованием средств COM, поскольку некоторые внутрипроцессные серверы всегда создаются в контексте главного контейнера STA.

В технологии OLE требуется, чтобы один поток был предназначен для формирования ответов на сообщения, относящиеся к контейнеру STA. В средствах ODSOLE это требование учитывается за счет создания специального потока, предназначенного исключительно для обработки сообщений в цикле. Этот специальный поток первым вызывает функцию `OleInitialize` и поэтому становится потоком главного контейнера STA для процесса SQL Server.

После создания этот поток главного контейнера STA почти игнорируется средствами ODSOLE. Каждый рабочий поток, который обслуживает вызовы

процедур sp_OA, выполняет свой собственный вызов функции `OleInitialize` и наращивает количество ссылок на поток обработки сообщений главного контейнера STA. Главный контейнер STA продолжает существовать до останова программы SQL Server или вызова процедуры `sp_OAStop`.

Сравнение раннего и позднего связывания

В приложении можно обеспечить использование COM-объектов с помощью двух основных способов — раннего или позднего связывания. Если в приложении применяется ссылка на объект, разрешимая на этапе компиляции, то считается, что происходит раннее связывание объекта, указанного в ссылке. Чтобы выполнить раннее связывание объекта в языке Visual Basic, необходимо на этапе разработки программы ввести ссылку на библиотеку, содержащую объект, а затем создавать конкретные экземпляры этого объекта с помощью оператора `Dim`. Для осуществления раннего связывания в таких инструментальных средствах, как Visual C++ и Delphi, необходимо осуществить импорт библиотеки типов объекта и работать с предоставляемыми объектом интерфейсами. В том и ином случае в обрабатываемой программе непосредственно используются интерфейсы, доступ к которым предоставляется объектом, как если бы это были интерфейсы, созданные в самой программе. Сам же объект может находиться совсем на другом компьютере, а доступ к нему может обеспечиваться с помощью средств DCOM (Distributed COM) или с помощью средств маршалинга такого диспетчера транзакций, как Microsoft Transaction Server или Component Services. Вообще говоря, для разработчика все это не имеет значения — он просто использует в создаваемой программе интерфейс, предоставляемый объектом.

Если же ссылки на объект не известны до этапа прогона, то происходит позднее связывание объекта. При этом обычно создается экземпляр объекта с помощью вызова функции `CreateObject`, а ссылка на объект сохраняется в переменной типа `Variant`. Поскольку компилятор на этапе компиляции не имеет информации о том, на какой объект фактически осуществляется ссылка, то на этапе прогона могут обнаруживаться неправильные вызовы методов или несуществующие свойства. В этом состоит оборотная сторона позднего связывания. Такой метод доступа к объектам, как позднее связывание, является более гибким, поскольку он позволяет решать на этапе прогона, какие экземпляры объектов должны быть созданы, и даже создавать экземпляры объектов, не существовавших в среде разработки. Но этот метод в большей степени чреват ошибками; в ходе позднего связывания объектов можно легко допустить ошибку, поскольку при этом среда разработки не может предоставить помощь на таком же уровне, который обеспечивается, когда имеется вся информация об объектах, применяемых разработчиком. Кроме того, доступ к COM-объектам с помощью средств позднего связывания происходит медленнее, чем при использовании раннего связывания, а иногда различия в быстродействии становятся весьма существенными. Несмотря на сказанное выше, позднее связывание — это единственный доступный способ использования средств ODSOLE, поэтому именно об этом методе будет в основном идти речь в данной главе.

Процедуры sp_OA

Хранимые процедуры средств Automation на языке Transact-SQL именуются с использованием соглашения sp_OA*Function*, где *Function* указывает на назначение процедуры (например, sp_OACreate создает COM-объекты, sp_OAMethod вызывает методы, sp_OAGetProperty и sp_OASetProperty позволяют, соответственно, получить и задать свойства объекта и т.д.). Каждая из процедур sp_OA, кроме sp_OACreate, принимает целочисленный параметр, содержащий указатель на ранее созданный объект. Процедура sp_OACreate, разумеется, создает объект и поэтому предусматривает передачу ей в качестве выходного параметра целочисленной переменной, в которую будет записана ссылка на создаваемый объект. Это целое число фактически ссылается на внутренний объект-оболочку, созданный средствами ODSOLE для инкапсуляции COM-объекта. Такой внутренний объект содержит ссылку на COM-объект, а также прочую вспомогательную информацию.

Некоторые процедуры sp_OA могут предоставлять возможность получения выходного параметра из метода COM или функции выборки свойства (например, sp_OAMethod или sp_OAGetProperty). Если такой параметр не задан, то возвращается одностолбцовый и однострочный результирующий набор. Если же вызов метода или функции приводит к возврату массива, то выходной параметр принимает значение NULL (если он задан), а массив преобразуется в результирующий набор. Если массив является одномерным, то возвращается одна строка, в которой элементы массива оформлены как столбцы. Если же массив является двумерным, то происходит возврат массива в виде многострочного результирующего набора. Наконец, если массив имеет больше двух измерений, то возвращается ошибка.

Процедура sp_OACreate

Как уже было сказано выше, процедура sp_OACreate используется для создания экземпляра COM-объекта. Вызов процедуры sp_OACreate приводит к возврату указателя на внутренний объект ODSOLE, который инкапсулирует ссылку на основополагающий COM-объект. После вызова процедуры sp_OACreate происходят описанные ниже действия.

1. Создается главный контейнер STA, если он еще не существует. Такое событие можно обнаружить из отладчика WinDbg, перехватывая вызовы функции OleInitialize. Если главный контейнер STA уже существует, то наращивается количество ссылок на поток этого главного контейнера.
2. Инициализируется память TLS для текущего рабочего потока. Эта память, кроме всего прочего, используется для отслеживания объектов, созданных во время выполнения пакетного задания, чтобы эти объекты можно было автоматически удалить после завершения пакетного задания.
3. Вызывается функция OleInitialize для текущего рабочего потока.
4. Выполняется настройка двух обработчиков событий ODS (одного для языковых событий, а другого для событий RPC), которые должны быть вызваны на выполнение после завершения пакетного задания. Эти обработчики

обеспечивают автоматическое удаление созданных объектов, вызывая функцию CoUninitialize для текущего рабочего потока, уменьшая количество ссылок на главный контейнер STA и выполняя другую вспомогательную работу после завершения пакетного задания.

5. Указанное имя объекта передается в функцию CLSIDFromProgID API-интерфейса OLE для преобразования его из формата ProgID в идентификатор класса COM, экземпляр которого может быть создан с помощью вызова функции CoCreateInstance. Идентификатор ProgID (Programmatic IDentifier – программный идентификатор) представляет собой строку, которая идентифицирует COM-объект и применяется для того, чтобы приложения могли обращаться к этому объекту по имени. Идентификатор ProgID не может непосредственно применяться средствами COM для создания экземпляра объекта, поэтому, чтобы создать экземпляр объекта с использованием его идентификатора ProgID, необходимо вначале преобразовать идентификатор ProgID объекта в идентификатор класса COM этого объекта с помощью функции CLSIDFromProgID. Если вызов функции CLSIDFromProgID оканчивается неудачей, то средства ODSOLE действуют на основании предположения, что переданная в эту функцию строка уже представляет собой идентификатор класса, и передают строку в функцию CLSIDFromString. Если оканчивается неудачей и вызов функции CLSIDFromString, то возвращается ошибка, и вызов процедуры sp_OACreate завершается неудачей.
6. Идентификатор класса, который преобразуется в имя указанного объекта, передается в функцию CoCreateInstance для создания экземпляра COM-объекта.
7. Вызывается предусмотренный в технологии COM метод QueryInterface вновь созданного объекта для получения ссылки на реализацию интерфейса IDispatch этого объекта. Как было сказано выше, интерфейс IDispatch лежит в основе способа позднего связывания, с помощью которого клиенты взаимодействуют с COM-компонентами. Поэтому можно утверждать, что выполняемые действия равносильны раннему связыванию объектов с интерфейсом IDispatch.
8. На этом этапе объект уже готов к использованию в других процедурах sp_OA, поэтому ссылка на объект заключается во внутренний объект, а указатель на этот объект возвращается в выходном параметре процедуры sp_OACreate.

В процедуру sp_OACreate можно передать параметр с указанием контекста. Этот параметр преобразуется в параметр dwClsContext вызова функции CoCreateInstance и определяет контекст, в котором создается объект. Экземпляр объекта может быть создан в качестве внутрипроцессного сервера (в этом случае он функционирует в том же процессе, что и вызывающий объект (т.е. программа SQL Server; отсюда и происходит термин *внутрипроцессный*), или в качестве внепроцессного сервера (в этом случае объект функционирует в своем собственном процессе). С другой стороны, параметр с указанием контекста может быть задан таким образом, чтобы поддерживался любой контекст, а фактически используемый контекст определялся в зависимости от того, находится COM-компонент в файле

EXE или DLL. Создание COM-объекта в качестве внепроцессного позволяет исключить возможность нарушения этим объектом работы процесса SQL Server или возникновения нарушений устойчивой работы каких-либо других типов.

Процедура sp_OAMethod

Процедура sp_OAMethod принимает в качестве параметра, наряду с именем метода, ссылку на ранее созданный объект, а также принимает выходной параметр и список входных параметров переменной длины. Если задан выходной параметр, а метод не возвращает значение этого параметра, то возвращается ошибка. Если задан выходной параметр, имеющий слишком малый формат представления для выходного значения, также возвращается ошибка. Если возвращается выходное значение, но выходной параметр не задан, то вырабатывается одностолбцовый и однострочный результирующий набор, при условии, что возвращаемое значение не представляет собой массив. Кроме того, как было указано выше, если возвращаемым значением является массив, то возвращается результирующий набор.

Если вызов процедуры sp_OAMethod осуществляется в обработчике событий RPC или в обработчике языковых событий, то происходят описанные ниже действия.

1. Вызывается функция GetIDsOfNames API-интерфейса COM для получения диспетчерского идентификатора вызываемого метода. Если это действие оканчивается неудачей, то вызов процедуры sp_OAMethod завершается неудачей так же.
2. Структура DISPPARAMS заполняется параметрами, которые должны быть переданы в вызываемый метод.
3. Вызывается метод IDispatch::Invoke для вызова используемого метода объекта. Если это действие оканчивается неудачей, то в качестве результата процедуры sp_OAMethod возвращается значение HRESULT.

Процедуры sp_OASetProperty и sp_OAGetProperty

Эти процедуры весьма напоминают процедуру sp_OAMethod. Но если используется позднее связывание, то, в отличие от вызова метода, операции задания или получения значения свойства, по существу, представляют собой одну и ту же операцию, поэтому оба эти действия осуществляются средствами ODSOLE почти одинаково. Опыт автора показывает, что обычно можно применять процедуру sp_OASetProperty или sp_OAGetProperty на равных с процедурой sp_OAMethod. В ходе выполнения процедур задания или получения значений свойств вызывается функция GetIDsOfNames, а также метод IDispatch::Invoke. В метод IDispatch::Invoke передается параметр в формате двоичной структуры отображения (bitmap), который указывает, осуществляется ли задание (получение) значения свойства или происходит вызов метода; средства ODSOLE передают маску, которая включает оба битовых значения, поскольку эти битовые значения фактически почти не различимы, если доступ к объекту осуществляется с помощью позднего связывания.

Процедура sp_OAGetErrorInfo

Эта процедура возвращает информацию об ошибке, относящуюся к заданному указателю на объект или к текущему рабочему потоку. Как правило, следует проверять результаты, возвращаемые после вызова одной из прочих процедур sp_OA, на наличие ненулевого значения, а затем при необходимости вызывать процедуру sp_OAGetErrorInfo для ознакомления с дополнительной информацией об ошибке.

Процедура sp_OADestroy

Эта процедура удаляет ссылку на объект, созданный с помощью процедуры sp_OACreate. После удаления ссылки этот объект больше не может использоваться в каких-либо дальнейших вызовах ODSOLE.

Процедура sp_OAStop

Эта процедура завершает обработку с помощью средств OLE и останавливает поток главного контейнера STA. Никакие дополнительные вызовы процедур sp_OA не могут выполняться до тех пор, пока не будет снова вызвана процедура sp_OACreate для создания объекта и перезапуска потока главного контейнера STA.

Переход по компонентам имени объекта

По аналогии с Visual Basic, VBScript и многими другими языками, поддерживающими средства автоматизации Automation, средства ODSOLE позволяют использовать так называемую "уточняющую запись через точку" в именах методов и свойств для быстрого перехода по иерархии объектов. С помощью такого обозначения можно указывать весь путь между родительским объектом и его дочерними объектами, дочерними объектами этих дочерних объектов и так далее, разделяя имена объектов точками в строке с именем свойства или метода. Каждый промежуточный терм имени должен ссылаться на какой-то конкретный объект, а заключительный терм может ссылаться на любое свойство или на любой метод, доступ к которому предоставляется объектом. Это означает, что можно вместо следующей совокупности команд:

```
-- Получить указатель на коллекцию Databases объекта SQLServer
EXEC @hr = sp_OAGetProperty @srvobject, 'Databases', @object OUT
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @srvobject, @hr
  GOTO FreeAll
END

-- Получить указатель из коллекции Databases на требуемую базу данных
EXEC @hr = sp_OAMethod @object, 'Item', @object OUT, @dbname
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  GOTO FreeAll
END
```

применять такие команды:

```
-- Получить указатель на требуемую базу данных
DECLARE @itemname varchar(255)
SET @itemname='Databases.Item("'+@dbname+'")'
EXEC @hr = sp_OAGetProperty @srvobject, @itemname, @object OUT
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @srvobject, @hr
  GOTO FreeAll
END
```

Использование уточняющей записи через точку в приведенном выше коде позволило избежать необходимости предварительного получения указателя на коллекцию Databases с последующим выполнением отдельного вызова для выборки конкретного элемента из этой коллекции. Такое уточняющее обозначение через точку может иметь любую необходимую длину; средства ODSOLE обеспечивают переход по термам имени метода или свойства, а также в случае необходимости переход к листовому терму.

Именованные параметры

Наряду с языком Visual Basic и другими контроллерами Automation, средства ODSOLE поддерживают понятие именованных параметров. Именованные параметры могут быть заданы в именах методов и свойств (с помощью уточняющей записи через точку), а также в командной строке вызова расширенной процедуры sp_OA. В случае использования процедур sp_OA именованные параметры должны быть заданы вслед за третьим параметром процедуры (именованные параметры, введенные в качестве параметра, предшествующего четвертому параметру, игнорируются); кроме того, именованные параметры обозначаются ведущим префиксом @, который удаляется в ходе осуществления вызова процедуры. Именованные параметры ODSOLE должны соответствовать обычным требованиям, предъявляемым к именованным параметрам Automation: безымянные параметры должны быть заданы перед именованными параметрами, а именованные параметры могут быть заданы в любом порядке.

Автоматизация с помощью средств ODSOLE

В следующих разделах приведен ряд примеров, которые показывают, как автоматизировать COM-объекты с помощью процедур sp_OA и средств ODSOLE. Мы рассмотрим некоторые простые программы, которые показывают, как получить доступ к функциональным возможностям COM, которые, скорее всего, уже имеются на компьютере читателя, а затем рассмотрим, как осуществляется автоматизация объектов управления распределенными сетями программы SQL Server (SQL-DMO, где DMO – сокращение от Distributed Management Objects, объекты управления распределенными сетями) с помощью средств ODSOLE. В конце данной главы представлены малоизвестные сведения о том, как реализуются массивы

в языке T-SQL с использованием COM-объектов, как применяются средства COM Interop и ODSOLE для доступа к объектам в инфраструктуре .NET Framework, а также некоторые другие дополнительные сведения.

Процедура sp_checkspelling

В листинге 15.1 приведена простая процедура, в которой используются процедуры sp_OA для автоматизации COM-объекта. В этой процедуре создается экземпляр объекта Microsoft Word Application, и вызывается метод CheckSpelling этого объекта для проверки правописания слова, переданного в процедуру.

Листинг 15.1. Использование процедур sp_OA для автоматизации COM-объекта

```
USE master
GO
IF (OBJECT_ID('sp_checkspelling') IS NOT NULL)
  DROP PROC sp_checkspelling
GO
CREATE PROC sp_checkspelling
  @word varchar(30),  -- Проверяемое слово
  @correct bit OUT  -- Возвратить код, указывающий, успешно ли прошло
                    -- данное слово орфографическую проверку
/*
Object: sp_checkspelling
Description: Checks the spelling of a word using the Microsoft Word
    Application Automation object
  Usage: sp_checkspelling
  @word varchar(128),    -- Проверяемое слово
@correct bit OUT  -- Возвратить код, указывающий, успешно ли прошло
                  -- данное слово орфографическую проверку
  Returns: (None)
  $Author: Ken Henderson $. Email: khen@khen.com
  Example: EXEC sp_checkspelling 'asdf', @correct OUT
  Created: 2000-10-14.  $Modtime: 2001-01-13 $.
*/
AS
IF (@word='/?') GOTO Help
DECLARE @object int,  -- Рабочая переменная, применяемая при создании
                      -- экземпляров COM-объектов
  @hr int  -- Содержит значение HRESULT, возвращаемое средствами COM

-- Создать объект Word Application
EXEC @hr=sp_OACreate 'Word.Application', @object OUT
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  RETURN
END

-- Вызвать метод CheckSpelling этого объекта
EXEC @hr = sp_OAMethod @object, 'CheckSpelling', @correct OUT,
    @word
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
```

```
  RETURN @hr
END

-- Уничтожить объект
EXEC @hr = sp_OADestroy @object
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  RETURN @hr
END

RETURN 0

Help:

EXEC sp_usage @objectname='sp_checkspelling',
@desc='Checks the spelling of a word using the Microsoft Word
   Application Automation object',
@parameters='
  @word varchar(30),   -- Проверяемое слово
  @correct bit OUT  -- Возвратить код, указывающий, успешно ли прошло
                    -- данное слово орфографическую проверку
',
@author='Ken Henderson', @email='khen@khen.com',
@datecreated='20001014',@datelastchanged='20010113',
@example='EXEC sp_checkspelling ''asdf'', @correct OUT',
@returns='(None)'
RETURN -1
GO
```

Процедура `sp_checkspelling` предоставляет доступ к двум параметрам. Одним из этих параметров является слово, правописание которого необходимо проверить, а другой представляет собой выходной параметр, который принимает значение 1 или 0 и указывает, успешно ли прошло это слово орфографическую проверку. Ниже приведен вызов этой процедуры.

```
DECLARE @cor bit
EXEC sp_checkspelling 'asdf', @cor OUT
SELECT @cor

(Результаты)

----
0
```

В ходе выполнения данной процедуры осуществляются три основных действия: создание COM-объекта, вызов его метода и удаление объекта. Начнем с вызова процедуры `sp_OACreate`. Вызов процедуры `sp_OACreate` приводит к созданию экземпляра COM-объекта. Строка `Word.Application` — это идентификатор ProgID, связанный с программой Microsoft Word. Как узнать о том, что в данном случае следует использовать идентификатор `Word.Application`? Для этого существует несколько способов. Во-первых, можно ознакомиться с объектным интерфейсом программы Word, документация которого представлена на Web-узле разработчиков Microsoft, MSDN. Во-вторых, можно активизировать среду Visual Basic и ввести в какой-то проект ссылку `Reference` на объектную библиотеку `Microsoft`

Word Object Library, после чего обратиться к технологии Intellisense программы Visual Studio для ознакомления с объектами и методами, доступ к которым можно получить из программы Word. (Той же цели можно достичь с помощью директивы #import языка Visual C++ или опции Project | Import Type Library программы Delphi.) В-третьих, можно просто открыть системный реестр и просмотреть все интерфейсы, которые относятся к программе Microsoft Word. Например, системный реестр может позволить узнать, что Word.Application — это строковое значение независимого от версии ключа VersionIndependentProgID программы Word. Это означает, что попытка создания экземпляра объекта Word.Application должна быть осуществлена успешно, независимо от того, какая версия программы Word установлена на компьютере.

Затем в этой процедуре дескриптор объекта, возвращаемый процедурой sp_OACreate, сохраняется в переменной @object. В дальнейшем этот дескриптор передается в процедуру sp_OAMethod при вызове методов интерфейса Word.Application. В данном случае вызывается только один метод, CheckSpelling, и ему передается переменная @word в качестве слова, для которого должна быть выполнена проверка правописания, и переменная @correct, в которой сохраняется значение 1 или 0, возвращаемое данным методом.

После завершения работы с объектом этот объект уничтожается с помощью вызова процедуры sp_OADestroy. И в этом случае передается дескриптор @object, полученный ранее из процедуры sp_OACreate.

Приведенный пример демонстрирует основные принципы работы с COM-объектами в языке Transact-SQL. Как и при использовании многих других языков и технологий, создается объект, с его помощью выполняются определенные действия, после чего этот объект, выполнивший всю возложенную на него работу, уничтожается.

Процедура sp_vbscript_reg_ex

В следующем примере показано, как ввести поддержку регулярных выражений в язык Transact-SQL. Регулярные выражения обеспечивают проверку согласования строк с шаблонами с помощью подстановочных символов и других средств согласования. Регулярные выражения обычно применяются в редакторах, предназначенных для программистов (например, регулярные выражения поддерживает текстовый редактор исходного кода Sequin SQL, включенный в состав компакт-диска, прилагаемого к данной книге); кроме того, доступ к этим средствам предоставляется в различных API-интерфейсах и языках программирования. Одним из средств, в котором предусмотрена превосходная поддержка регулярных выражений, является машина сценариев ActiveX компании Microsoft. В этой машине сценариев предусмотрен объект RegExp, в котором реализованы основные функциональные возможности сопоставления с регулярными выражениями, основанными на использовании шаблонов. В листинге 15.2 показано, как используется объект RegExp для согласования строки с регулярным выражением из хранимой процедуры с помощью средств Automation и ODSOLE.

Листинг 15.2. Осуществление согласования регулярного выражения из хранимой процедуры

```
USE master
GO
IF OBJECT_ID('dbo.sp_vbscript_reg_ex','P') IS NOT NULL
  DROP PROC dbo.sp_vbscript_reg_ex
GO
CREATE PROC dbo.sp_vbscript_reg_ex @pattern varchar(255),
    @matchstring varchar(8000)
AS
declare @obj int
declare @res int
declare @match bit
set @match=0
exec @res=sp_OACreate 'VBScript.RegExp',@obj OUT
IF (@res <> 0) BEGIN
  PRINT 'VBScript.RegExp Create failed'
  EXEC sp_DisplayOAErrorInfo @obj, @res
  RETURN
END
exec @res=sp_OASetProperty @obj, 'Pattern', @pattern
IF (@res <> 0) BEGIN
  PRINT 'Set Pattern failed'
  EXEC sp_DisplayOAErrorInfo @obj, @res
  RETURN
END
exec @res=sp_OASetProperty @obj, 'IgnoreCase', 1
IF (@res <> 0) BEGIN
  PRINT 'Set IgnoreCase failed'
  EXEC sp_DisplayOAErrorInfo @obj, @res
  RETURN
END
exec @res=sp_OAMethod @obj, 'Test',@match OUT, @matchstring
IF (@res <> 0) BEGIN
  PRINT 'Test call failed'
  EXEC sp_DisplayOAErrorInfo @obj, @res
  RETURN
END
exec @res=sp_OADestroy @obj
return @match
```

Вполне очевидно, что эта процедура не содержит большого объема кода. При разработке данной процедуры был принят описанный ниже основной подход.

1. Создать экземпляр объекта `VBScript.RegExp`. Как уже было сказано, объект `RegExp` инкапсулирует функциональные возможности сопоставления с регулярными выражениями сценариев ActiveX.
2. Задать свойство `Pattern` объекта `RegExp`. Тем самым задается регулярное выражение, которое предназначено для использования в процедуре.
3. Присвоить свойству `IgnoreCase` объекта `RegExp` значение `true`. Такое действие обеспечивает поиск без учета регистра. Эту команду можно закомментировать или предусмотреть управление значением свойства `IgnoreCase` с помощью параметра хранимой процедуры, если необходимо обеспечить поиск и с учетом, и без учета регистра.

4. Вызвать метод Test объекта RegExp. Метод Test проверяет указанную строку по ранее заданному шаблону для определения того, имеется ли согласование между строкой и шаблоном, после чего возвращает логическое значение, которое показывает, успешно ли выполнено согласование.
5. Уничтожить объект. Учитывая то, что средства ODSOLE, как уже было сказано, освобождают распределенные объекты автоматически, формально указанное действие не требуется, но все равно рекомендуется как способ качественного программирования.

Некоторые дополнительные примеры вызова процедуры sp_vbscript_reg_ex приведены в листинге 15.3.

Листинг 15.3. Вызов процедуры sp_vbscript_reg_ex

```
SET NOCOUNT ON
declare @res int

PRINT 'Check a basic wildcard pattern'
exec @res=sp_vbscript_reg_ex 'A.*C','AxxxxxxxxxxxxxxxxxxxxxxBC'
select @res

PRINT 'Check a word boundary (fails)'
exec @res=sp_vbscript_reg_ex 'es\b','These are the days'
select @res

PRINT 'Check a word boundary (succeeds)'
exec @res=sp_vbscript_reg_ex 'es\b','Would you like some fries
    with that?'
select @res

PRINT 'Check an either/or pattern'
exec @res=sp_vbscript_reg_ex 'good|great','Now is the time for all
    good men to come to'
select @res

PRINT 'Check an either/or pattern'
exec @res=sp_vbscript_reg_ex 'good|great','Goodness, gracious,
    great balls of fire!'
select @res

(Результаты)

Check a basic wildcard pattern

-----------
1

Check a word boundary (fails)

-----------
0
Check a word boundary (succeeds)

-----------
```

```
1

Check an either/or pattern

-----------
1

Check an either/or pattern

-----------
1
```

Автор в данной главе в основном рассматривает средства сопоставления с шаблонами, которые было бы трудно или даже невозможно реализовать с использованием стандартных подстановочных символов языка T-SQL, применяемых в конструкциях LIKE и PATINDEX. Объект RegExp поддерживает также целый ряд других условий поиска с помощью регулярных выражений. Для ознакомления с дополнительными сведениями обратитесь к документации VBScript или к материалам, представленным на узле MSDN.

Автоматизация доступа к классам инфраструктуры .NET Framework с помощью интерфейса COM Interop

Еще одним средством, предоставляющим доступ к полнофункциональной поисковой машине с помощью регулярных выражений, является инфраструктура .NET Framework. Учитывая то, что инфраструктура Framework поддерживает также создание оболочек для управляемых классов, с тем, чтобы к этим классам можно было обеспечить доступ с помощью технологии COM, эта инфраструктура фактически позволяет создавать управляемые объекты, доступ к которым предоставляется из языка T-SQL с помощью средств ODSOLE.

Следует отметить, что способы вызова управляемого кода из программы SQL Server 2000 и предыдущих версий не поддерживались компанией Microsoft. Средства ODSOLE и инфраструктура .NET Framework не проверялись на функциональную совместимость друг с другом, поэтому при использовании описанных в данном разделе способов программирования читатель может столкнуться с проблемами, для решения которых он не сможет обратиться в службы поддержки программных продуктов Microsoft Product Support Services.

Несмотря на сказанное выше, программисту трудно остаться равнодушным к тем безграничным функциональным возможностям, которые предоставляет инфраструктура .NET Framework, особенно учитывая то, что к этим функциональным возможностям так легко получить доступ с помощью средств ODSOLE. В приведенном ниже примере кода показано, как создать на языке C# управляемый класс, который инкапсулирует средства поддержки регулярных выражений инфраструктуры .NET Framework. Затем этот класс публикуется и регистрируется для использования в технологии COM с помощью технологии COM Interop из инфраструктуры .NET

Framework, чтобы можно было получить доступ к функциональным возможностям этого управляемого класса из языка T-SQL с помощью процедур sp_OA.

Начнем с листинга 15.4, в котором приведен исходный код управляемого класса. (Полный исходный код этого примера можно найти в подкаталоге SQLRegExLib каталога CH15 компакт-диска, прилагаемого к данной книге.)

Листинг 15.4. Исходный код управляемого класса SQLRegEx

```
using System;
using System.Text.RegularExpressions;

namespace SQLRegExLib
{
  public interface IRegEx
  {
    bool IsMatch(string Expression, string MatchString);
  }
  /// <summary>
  /// Общее описание класса Class1
  /// </summary>
  public class SQLRegEx : IRegEx
  {
    public SQLRegEx()
     {
     }
    public bool IsMatch(string Expression, string MatchString)
     {
      Regex regex = new Regex(Expression,RegexOptions.Compiled |
          RegexOptions.IgnoreCase);
      if (null!=regex) return regex.IsMatch(MatchString);
      else throw new Exception("Unable to create Regex object");
     }
  }
}
```

Этот класс предоставляет доступ к единственному методу, IsMatch, который принимает два параметра — строку с шаблоном и строку, в которой выполняется поиск согласования с шаблоном. В методе IsMatch создается экземпляр класса Regex инфраструктуры .NET Framework, затем вызывается метод IsMatch класса Regex для определения согласования между шаблоном и строкой.

Основное требование, которому должен соответствовать класс для того, чтобы он был доступен из среды технологии COM, состоит в том, чтобы этот класс был общедоступным. Класс должен также реализовывать применяемый по умолчанию (не принимающий параметров) конструктор. Кроме того (хотя это требование не является обязательным), в данном конкретном модуле объявляется также общедоступный интерфейс, который затем реализуется классом. Определение в коде явно заданного интерфейса и реализация интерфейса в общедоступных классах позволяет упростить доступ к этим классам из среды технологии COM.

Для получения доступа к этому классу с помощью средств ODSOLE необходимо выполнить описанные ниже действия.

1. Создать новый проект Windows Class Library и добавить к нему рассматриваемый класс.
2. Откомпилировать проект для получения библиотеки DLL.
3. Скопировать полученную библиотеку DLL в подкаталог `binn` инсталляционного каталога программы SQL Server. Поскольку мы не собираемся обозначать эту сборку (т.е. библиотеку DLL) так называемым "сильным именем" и инсталлировать ее в глобальном кэше сборок (Global Assembly Cache — GAC), то применяемая библиотека DLL должна находиться в том же каталоге, из которого происходит запуск программы, вызывающей эту библиотеку. Вызов библиотеки должен осуществляться с помощью средств ODSOLE из программы SQL Server. Поэтому каталогом, из которого запускается программа, является тот каталог, где находится файл `sqlservr.exe`.
4. Зарегистрировать саму сборку и ее библиотеку типов с помощью инструментального средства с интерфейсом командной строки `regasm.exe`, которое входит в состав инфраструктуры .NET Framework. Регистрация компонента и регистрация его библиотеки типов — это две разные операции, которые должны быть выполнены отдельно. (Чтобы получить дополнительные сведения, прочитайте справку к программе `regasm.exe`.) В результате необходимые записи будут внесены в системный реестр для того, чтобы можно было осуществить доступ к рассматриваемому классу из среды технологии COM, задавая идентификатор `ProgID` этого класса или идентификатор самого класса.
5. Вызывать процедуры sp_OA для создания экземпляров и манипуляции с объектом по такому же принципу, как и при работе с любым другим COM-объектом.

После регистрации управляемого класса с помощью инструментального средства `regasm.exe` осуществляется настройка главной библиотеки DLL инфраструктуры .NET Framework (библиотеки `mscoree.DLL`) для использования в качестве сервера для данного COM-объекта. В этом состоит отличие от неуправляемого кода, при работе с которым в качестве сервера служит библиотека DLL, в которой реализованы функции COM-объекта. При использовании управляемых классов в качестве COM-сервера служит главная библиотека DLL инфраструктуры .NET Framework, а ссылка на сборку управляемого кода осуществляется с помощью ключа `Assembly` той записи системного реестра, которая относится к соответствующему объекту.

В листинге 15.5 представлена хранимая процедура, в которой создаются оболочки для вызовов компонента управляемого кода, зарегистрированного для использования в технологии COM.

Листинг 15.5. Создание оболочек для вызовов компонента управляемого кода

```
USE master
GO
IF OBJECT_ID('dbo.sp_dotnet_reg_ex','P') IS NOT NULL
  DROP PROC dbo.sp_dotnet_reg_ex
GO
```

```
CREATE PROC dbo.sp_dotnet_reg_ex @pattern varchar(255),
    @matchstring varchar(8000)
AS
declare @obj int
declare @res int
declare @match bit
set @match=0
exec @res=sp_OACreate 'SQLRegExLib.SQLRegEx',@obj OUT
IF (@res <> 0) BEGIN
  EXEC sp_DisplayOAErrorInfo @obj, @res
  RETURN
END
exec @res=sp_OAMethod @obj, 'IsMatch',@match OUT, @pattern,
    @matchstring
IF (@res <> 0) BEGIN
  EXEC sp_DisplayOAErrorInfo @obj, @res
  RETURN
END
exec @res=sp_OADestroy @obj
return @match
```

Эта процедура действует аналогично той рассматривавшейся ранее процедуре, которая была создана для объекта RegExp, и поддерживает регулярные выражения таких же типов.

Следует отметить, что потребность в виртуальной памяти для этой версии кода поддержки регулярных выражений, скорее всего, будет намного больше по сравнению с кодом, основанным на использовании объекта RegExp, который был разработан ранее. Это связано с тем, что в последнем случае в пространство процесса SQL Server загружается не только код компонента SQLRegEx, но и весь код инфраструктуры .NET Framework, необходимый для рассматриваемого компонента. Как и при загрузке большой библиотеки DLL любого другого типа, это может привести к возникновению проблем, связанных с конкуренцией за пространство адресов виртуальной памяти, и проблем фрагментации (эта темы рассматривалась в главе 4).

Еще раз отметим, что описанный здесь способ совместного применения двух технологий не был проверен компанией Microsoft и не поддерживается этой компанией. Автор привел в этой главе описание данного способа исключительно в учебных целях.

Использование COM-объектов в определяемых пользователем функциях

Безусловно, возможность вызывать COM-объекты с помощью хранимых процедур является очень удобной, но автор уверен, что многих читателей интересует вопрос — можно ли заключить функциональные возможности COM-объекта в определяемую пользователем функцию для использования в запросах T-SQL. Не правда ли, удобно иметь возможность использовать регулярное выражение в конструкции WHERE для задания критерия выборки в операторе SELECT? Безусловно, такая возможность действительно является удобной. Ниже приведен пример функции, который показывает, как это сделать (листинг 15.6).

Листинг 15.6. Использование регулярного выражения для задания критерия выборки в операторе SELECT

```
USE master
GO
exec sp_configure 'allow updates', 1
go
reconfigure with override
go
  DROP function system_function_schema.fn_regex
GO
CREATE FUNCTION
  system_function_schema.fn_regex(@pattern varchar(255),
     @matchstring varchar(8000))
RETURNS int
AS
BEGIN
declare @obj int
declare @res int
declare @match bit
set @match=0
exec @res=sp_OACreate 'VBScript.RegExp',@obj OUT
IF (@res <> 0) BEGIN
  RETURN NULL
END
exec @res=sp_OASetProperty @obj, 'Pattern', @pattern
IF (@res <> 0) BEGIN
  RETURN NULL
END
exec @res=sp_OASetProperty @obj, 'IgnoreCase', 1
IF (@res <> 0) BEGIN
  RETURN NULL
END
exec @res=sp_OAMethod @obj, 'Test',@match OUT, @matchstring
IF (@res <> 0) BEGIN
  RETURN NULL
END
exec @res=sp_OADestroy @obj
return @match
END
GO
exec sp_configure 'allow updates', 0
go
reconfigure with override
go
```

В этом коде выполняется несколько интересных действий. Во-первых, заслуживает внимания то, что к созданию системной функции привлекается псевдопользователь system_function_schema. *Системной функцией* называется такая функция, доступ к которой предоставляется из любого контекста базы данных без необходимости указывать полностью уточненное имя. Как описал автор в своей книге *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML*, для того, чтобы преобразовать некоторую функцию в системную, необходимо выполнить два требования — создать функцию в главной базе данных (указав в качестве ее владельца

псевдопользователя system_function_schema и разрешив наряду с этим обновления) и присвоить этой функции имя с префиксом fn_. Автор решил создать рассматриваемую функцию поддержки регулярных выражений как системную функцию, поскольку предположение, что эта функция будет полезной в рамках всей системы, является вполне резонным. Она заслуживает права быть системной функцией в силу самой лишь ее применимости.

Во-вторых, заслуживает внимания тот факт, что процедуры sp_OA вызываются непосредственно из рассматриваемой функции. Многие программисты, которым довелось выполнить большой объем работ по созданию определяемых пользователем функций, знают о том, что из определяемых пользователем функций нельзя вызывать обычные хранимые процедуры. Но, к счастью, процедуры sp_OA, хотя и имеют префикс sp_ (сокращение от stored procedure — хранимая процедура), фактически представляют собой расширенные процедуры, которые можно вызывать из определяемой пользователем функции. Столь же благоприятным является тот факт, что процедуры sp_OA не относятся к категории так называемых *специальных процедур*, т.е. расширенных процедур, реализованных внутри сервера. Точки входа процедур sp_OA определены в библиотеке ODSOLE70.DLL, поэтому указанные процедуры можно вызывать из определяемых пользователем функций наряду с любыми другими обычными расширенными процедурами.

Код этой функции близок к коду хранимой процедуры, созданной ранее для доступа к объекту RegExp, который входит в состав средств VBScript. В данной функции создается объект, устанавливаются некоторые свойства, а затем вызывается метод Test для проверки того, имеет ли место согласование между строкой и шаблоном. Как показывает листинг 15.7, функциональные средства поддержки регулярных выражений, заключенные в определяемой пользователем функции, могут использоваться для определения критериев выборки в запросе.

Листинг 15.7. Применение критериев выборки в запросе

```
use pubs
go
SELECT *
FROM authors
WHERE fn_regex('G.*',au_lname)<>0

(Результаты приведены в сокращенном виде)

au_id       au_lname                                 au_fname
----------- ---------------------------------------- ------------
213-46-8915 Green                                    Marjorie
527-72-3246 Greene                                   Morningstar
472-27-2349 Gringlesby                               Burt
998-72-3567 Ringer                                   Albert
899-46-2035 Ringer                                    Anne
274-80-9391 Straight                                  Dean
724-08-9931 Stringer                                  Dirk
```

Вполне очевидно, что читатель найдет и другие COM-объекты, которые ему хотелось бы включить в определяемую пользователем функцию для использования

в запросах. Как показывает приведенный пример, совсем несложно определить системную функцию, чтобы предоставить доступ к функциональным возможностям некоторого COM-объекта во всей программе SQL Server.

Автоматизация интерфейса SQL-DMO с помощью средств ODSOLE

Нет ничего удивительного в том, что процедуры sp_OA широко применяются для автоматизации COM-объектов, доступ к которым предоставляет сама программа SQL Server. К числу распространенных вариантов указанных средств программы SQL Server относятся объектная модель DTS и интерфейс SQL-DMO. В данном разделе представлен ряд хранимых процедур, которые показывают, как автоматизировать COM-объекты интерфейса SQL-DMO с использованием средств ODSOLE программы SQL Server. Следует учитывать, что для выполнения большинства операций DMO требуется соединение с программой SQL Server, поэтому в хранимых процедурах, которые управляют объектами DMO с помощью средств ODSOLE, обычно приходится устанавливать соединение через петлю обратной связи. При создании соединений через петлю обратной связи могут возникать проблемы, если эти соединения приводят к созданию блокировок или к осуществлению попыток доступа к ресурсам, заблокированным вызывающим серверным процессом. Кроме того, можно столкнуться с ситуациями, в которых фактически происходит самоблокировка, когда вызывающий серверный процесс владеет блокировкой на ресурсе, который требуется в устанавливаемом через петлю обратной связи соединении для выполнения необходимых операций DMO. Такие ситуации встречаются не часто, но вполне возможны. При возникновении подобной ситуации обычно можно решить проблему, применяя более эффективное управление транзакциями или соединяя родительскую транзакцию и транзакцию, осуществляемую через петлю обратной связи, с помощью вызовов процедур `sp_getbindtoken` и `sp_bindsession`.

Процедура `sp_exporttable`

Процедура `sp_exporttable` создает экземпляры собственных объектов SQL-DMO программы SQL Server для экспорта таблицы, указанной по имени. Эта процедура действует аналогично встроенной команде `BULK INSERT`, предоставляя интерфейс к API-интерфейсу BCP (Bulk Copy Program —программа массового копирования) из языка Transact-SQL. Код рассматриваемой процедуры приведен в листинге 15.8.

Листинг 15.8. Код процедуры `sp_exporttable`

```
USE master
GO
IF (OBJECT_ID('sp_exporttable') IS NOT NULL)
```

```
  DROP PROC sp_exporttable
GO
CREATE PROC sp_exporttable
  @table sysname,                -- Экспортируемая таблица
  @outputpath sysname=NULL,      -- Строка с обозначением выходного
                                 -- каталога, завершающаяся символом "\"
  @outputname sysname=NULL,      -- Имя выходного файла (по умолчанию
                                 -- определяемое как @table+'.BCP')
  @server sysname='(local)',     -- Имя сервера, к которому осуществляется
                                 -- подключение
  @username sysname='sa',        -- Имя пользователя (по умолчанию
                                 -- применяется имя "sa")
  @password sysname=NULL,        -- Пароль пользователя
  @trustedconnection bit=1       -- Использовать заслуживающее доверия
                                 -- соединение для подключения к серверу
/*
Object: sp_exporttable
Description: Exports a table in a manner similar to BULK INSERT
  Usage: sp_exporttable
  @table sysname,                -- Экспортируемая таблица
  @outputpath sysname=NULL,      -- Строка с обозначением выходного
                                 -- каталога, оканчивающаяся символом "\"
  @outputname sysname=NULL,      -- Имя выходного файла (по умолчанию
                                 -- определяемое как @table+'.BCP')
  @server sysname='(local)',     -- Имя сервера, к которому осуществляется
                                 -- подключение
  @username sysname='sa',        -- Имя пользователя (по умолчанию
                                 -- применяется имя "sa")
  @password sysname=NULL,        -- Пароль пользователя
  @trustedconnection bit=1       -- Использовать заслуживающее доверия
                                 -- соединение для подключения к серверу
  Returns: Number of rows exported
  $Author: Ken Henderson $. Email: khen@khen.com
  Example: EXEC sp_exporttable 'authors', 'C:\TEMP\'
  Created: 1999-06-14.  $Modtime: 2000-12-01 $.
*/
AS
IF (@table='/?') OR (@outputpath IS NULL) GOTO Help
DECLARE @srvobject int,          -- Серверный объект
  @object int,                   -- Рабочая переменная, применяемая при
                                 -- создании экземпляров COM-объектов
  @hr int,                       -- Содержит значение HRESULT,
                                 -- возвращаемое средствами COM
  @bcobject int,                  -- Хранит указатель на объект BulkCopy
  @TAB_DELIMITED int,            -- Предназначена для хранения константы,
                                 -- применяемой при выводе строки с
                                 -- разделителями
  @logname sysname,               -- Имя файла журнала
  @errname sysname,              -- Имя файла с информацией об ошибках
  @dbname sysname,                -- Имя базы данных
  @rowsexported int              -- Количество экспортированных строк
 SET @TAB_DELIMITED=2            -- Константа SQL-DMO для экспорта строк с
                                 -- разделителями
SET @dbname=ISNULL(PARSENAME(@table,3),DB_NAME())
    -- Определить имя базы данных
SET @table=PARSENAME(@table,1)
```

```
   -- Удалить излишние элементы из имени таблицы
IF (@table IS NULL) BEGIN
  RAISERROR('Invalid table name.',16,1)
  GOTO Help
END
IF (RIGHT(@outputpath,1)<>'\')
  SET @outputpath=@outputpath+'\'       -- В случае необходимости
                                        -- добавить символ "\"
SET @logname=@outputpath+@table+'.LOG' -- Сформировать имя файла
                                        -- журнала
SET @errname=@outputpath+@table+'.ERR' -- Сформировать имя файла с
                                        -- информацией об ошибках

IF (@outputname IS NULL)
  SET @outputname=@outputpath+@table+'.BCP' -- Сформировать имя
                                            -- выходного файла
ELSE
  IF (CHARINDEX('\',@outputname)=0)
    SET @outputname=@outputpath+@outputname

-- Создать объект SQLServer
EXEC @hr=sp_OACreate 'SQLDMO.SQLServer', @srvobject OUTPUT
IF (@hr <> 0) GOTO ServerError

-- Создать объект BulkCopy
EXEC @hr=sp_OACreate 'SQLDMO.BulkCopy', @bcobject OUTPUT
IF (@hr <> 0) GOTO BCPError

-- Задать в качестве свойства DataFilePath объекта BulkCopy
-- имя выходного файла
EXEC @hr = sp_OASetProperty @bcobject, 'DataFilePath', @outputname
IF (@hr <> 0) GOTO BCPError

-- Передать объекту BulkCopy указание, что должны создаваться файлы
-- с разграничителями в виде символов табуляции
EXEC @hr = sp_OASetProperty @bcobject, 'DataFileType',
   @TAB_DELIMITED
IF (@hr <> 0) GOTO BCPError

-- Задать в качестве свойства LogFilePath объекта BulkCopy
-- имя файла журнала
EXEC @hr = sp_OASetProperty @bcobject, 'LogFilePath', @logname
IF (@hr <> 0) GOTO BCPError

-- Задать в качестве свойства ErrorFilePath объекта BulkCopy имя файла
-- с информацией об ошибках
EXEC @hr = sp_OASetProperty @bcobject, 'ErrorFilePath', @errname
IF (@hr <> 0) GOTO BCPError
-- Установить соединение с сервером
IF (@trustedconnection=1) BEGIN
  EXEC @hr = sp_OASetProperty @srvobject, 'LoginSecure', 1
    IF (@hr <> 0) GOTO ServerError
  EXEC @hr = sp_OAMethod @srvobject, 'Connect', NULL, @server
END ELSE BEGIN
  IF (@password IS NOT NULL)
  EXEC @hr =sp_OAMethod @srvobject,'Connect',NULL,@server,
```

```
      @username, @password
    ELSE
      EXEC @hr = sp_OAMethod @srvobject, 'Connect', NULL,
          @server, @username
END
IF (@hr <> 0) GOTO ServerError

-- Получить указатель на коллекцию Databases объекта SQLServer
EXEC @hr = sp_OAGetProperty @srvobject, 'Databases', @object OUT
IF (@hr <> 0) GOTO ServerError

-- Получить указатель из коллекции Databases на указанную базу данных
EXEC @hr = sp_OAMethod @object, 'Item', @object OUT, @dbname
IF (@hr <> 0) GOTO Error

-- Получить указатель из коллекции Tables объекта Databases
-- на указанную таблицу
IF (OBJECTPROPERTY(OBJECT_ID(@table),'IsTable')=1) BEGIN
EXEC @hr = sp_OAMethod @object, 'Tables', @object OUT, @table
  IF (@hr <> 0) GOTO Error
END ELSE -- Получить указатель из коллекции View объекта Databases
        -- на указанное представление
IF (OBJECTPROPERTY(OBJECT_ID(@table),'IsView')=1) BEGIN
EXEC @hr = sp_OAMethod @object, 'Views', @object OUT, @table
  IF (@hr <> 0) GOTO Error
END ELSE BEGIN
  RAISERROR('Source object must be either a table or view.',16,1)
  RETURN -1
END

-- Вызвать метод ExportData данного объекта для выполнения экспорта
-- таблицы/представления с помощью метода BulkCopy
EXEC @hr = sp_OAMethod @object, 'ExportData', @rowsexported OUT,
    @bcobject
IF (@hr <> 0) GOTO Error
EXEC sp_OADestroy @srvobject -- Уничтожить объект сервера
EXEC sp_OADestroy @bcobject  -- Уничтожить объект BCP
RETURN @rowsexported

Error:

EXEC sp_displayoaerrorinfo @object, @hr

GOTO ErrorCleanUp

BCPError:

EXEC sp_displayoaerrorinfo @bcobject, @hr

GOTO ErrorCleanUp

ServerError:

EXEC sp_displayoaerrorinfo @srvobject, @hr

GOTO ErrorCleanUp
```

```
ErrorCleanUp:

IF @srvobject IS NOT NULL
  EXEC sp_OADestroy @srvobject -- Уничтожить объект сервера
IF @bcobject IS NOT NULL
  EXEC sp_OADestroy @bcobject  -- Уничтожить объект BCP

RETURN -2

Help:

EXEC sp_usage @objectname='sp_exporttable',
@desc='Exports a table in a manner similar to BULK INSERT',
@parameters='
  @table sysname,                -- Экспортируемая таблица
  @outputpath sysname=NULL,      -- Строка с обозначением выходного
                                 -- каталога, завершающаяся символом "\"
  @outputname sysname=NULL,      -- Имя выходного файла (по умолчанию
                                 -- определяемое как @table+'.BCP')
  @server sysname=''(local)'', -- Имя сервера, к которому
                                 -- осуществляется подключение
  @username sysname=''sa'',      -- Имя пользователя (по умолчанию
                                 -- применяется имя "sa")
  @password sysname=NULL,         -- Пароль пользователя
  @trustedconnection bit=1       -- Использовать заслуживающее доверия
                                 -- соединение
',
@author='Ken Henderson', @email='khen@khen.com',
@datecreated='19990614',@datelastchanged='20001201',
@example='EXEC sp_exporttable ''authors'', ''C:\TEMP\''',
@returns='Number of rows exported'
RETURN -1
GO
```

Функционирование процедуры `sp_exporttable` организовано в соответствии с общим планом, описанным ниже.

1. Создать объект `SQLServer`. Этот объект будет использоваться для подключения к серверу. Объект `SQLServer` требуется в большинстве приложений DMO. Доступ к другим объектам сервера будет осуществляться с помощью раскрытия нижних уровней объекта `SQLServer` по такому же принципу, как и в программе Enterprise Manager.
2. Создать объект `BulkCopy`. Этот объект будет использоваться для экспорта таблицы. В конечном итоге вызывается метод `ExportData` указанной таблицы (или представления) для осуществления массового копирования содержимого таблицы (или представления) в файл операционной системы. Методу `ExportData` для осуществления его работы требуется объект `BulkCopy`.
3. Задать различные свойства объекта `BulkCopy`, которые будут использоваться для управления экспортом.
4. Подключиться к серверу с помощью объекта `SQLServer`.

5. Найти предназначенную для экспорта таблицу (или представление) с использованием вложенных коллекций объектов, доступ к которым предоставляется объектом SQLServer.
6. Вызвать метод ExportData объекта таблицы (или представления) и передать этому методу требуемый объект BulkCopy в качестве параметра.
7. После завершения экспорта уничтожить объекты SQLServer и BulkCopy.

В коде хранимой процедуры приведены комментарии, поясняющие ее работу; в целом эта процедура является довольно несложной. Для вызова процедуры sp_exporttable на выполнение используется следующий синтаксис:

```
DECLARE @rc int
EXEC @rc=pubs..sp_exporttable @table='pubs..authors',
    @outputpath='c:\temp\'
SELECT RowsExported=@rc

RowsExported
------------
23
```

Обратите внимание на то, как используется префикс pubs.. в вызове процедуры. В процедуре sp_exporttable применяется функция OBJECTPROPERTY (не позволяющая обращаться из одной базы данных в другую), поэтому для обеспечения успешной работы процедуры с объектами из других баз данных необходимо на время изменить контекст базы данных для того, чтобы он соответствовал указанному объекту. Для изменения на время контекста базы данных применяется префикс в виде имени базы данных, который предшествует вызову системной процедуры. Приведенный выше вызов эквивалентен следующему вызову:

```
USE pubs
GO
EXEC @rc=sp_exporttable @table='pubs..authors',
    @outputpath='c:\temp\'
GO
USE master -- Может быть указана какая-то другая база данных
GO
SELECT RowsExported=@rc
```

Читатель мог заметить, что в листинге 15.8 показаны вызовы системной процедуры sp_DisplayOAErrorInfo. Процедура sp_DisplayOAErrorInfo используется для отображения более подробной информации об ошибках, соответствующей кодам ошибок, возвращаемых процедурами sp_OA. В процедуре sp_DisplayOAErrorInfo вызывается процедура sp_OAGetErrorInfo для получения дополнительной информации об ошибках, относящейся к кодам ошибок объекта Automation. По умолчанию процедура sp_DisplayOAErrorInfo не создается, но если эта процедура потребуется, то ее можно найти в оперативной документации Books Online. Функционирование этой процедуры зависит от работы процедуры sp_hexadecimal (также приведенной в документации Books Online), которая применяется для преобразования двоичных значений в шестнадцатеричные строки. Исходный код обеих процедур можно найти в описании темы "*OLE Automation Return Codes and Error Information*" (Коды возврата и информация об ошибках средств OLE Automation), которая представлена в документации Books Online.

Данный пример и некоторые другие примеры, представленные в этой главе, показывают, как использовать хранимые процедуры sp_OA для автоматизации COM-объектов, доступ к которым обеспечивает сама программа SQL Server (в данном случае речь идет об объектах SQL-DMO). Объекты SQL-DMO предоставляют значительную часть функциональных возможностей, лежащих в основе функционирования программы Enterprise Manager, и позволяют воспользоваться удобными способами управления сервером с помощью программного кода.

Подробные сведения о работе процедуры `sp_exporttable` приведены в комментариях к этой процедуре. В коде процедуры выполняется целый ряд интересных действий, детальное описание которых в данном разделе потребовало бы слишком много места. Благодаря использованию средств Automation технологии COM эта процедура позволяет легко выполнять довольно трудную задачу. Объем кода Transact-SQL, требуемого для решения рассматриваемой задачи, не больше, чем требуется в сопоставимой программе Delphi или Visual Basic.

Процедура `sp_generate_script`

Процедура `sp_generate_script` позволяет создавать сценарии Transact-SQL для воссоздания объектов базы данных. В ней используются средства Automation для доступа к интерфейсу SQL-DMO и вызывается метод `ScriptTransfer` объекта `Transfer` интерфейса SQL-DMO для формирования файла сценария. Рассматриваемая процедура впервые была опубликована в книге автора *The Guru's Guide to Transact-SQL*, а обновленная версия этой процедуры приведена в изданной вслед за ней книге *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML*. Автор еще раз переработал для настоящей книги указанную процедуру, чтобы она стала еще немного более надежной и более терпимой к некоторым недостаткам интерфейса SQL-DMO.

Процедура `sp_generate_script` может применяться по принципу передачи ей имени объекта, для которого необходимо создать сценарий. При желании в процедуру можно передать маску с подстановочными символами или вообще исключить параметр имени и сформировать тем самым сценарий для всей текущей базы данных.

Как уже было сказано, процедура `sp_generate_script` для выполнения своей работы применяет API-интерфейс SQL-DMO. В процедуре создаются COM-объекты интерфейса DMO с использованием процедуры `sp_OACreate` и вызываются методы указанных объектов с помощью процедуры `sp_OAMethod`. Для интерфейса DMO требуется соединение с сервером (чтобы можно было выполнять необходимую работу по формированию сценариев), поэтому в процедуру `sp_generate_script` следует передать регистрационную информацию, позволяющую процедуре создать экземпляр соединения с сервером через петлю обратной связи, применяя вызовы методов DMO. После установления соединения процедура находит один или несколько объектов, для которых требуется сформировать сценарии, и добавляет эти объекты к объекту `Transfer` интерфейса DMO, чтобы можно было выполнить запись в дисковый файл. Если же требуется получить результирующий набор (эта опция разрешена по умолчанию), то в процедуре `sp_generate_script` вызывается расширенная процедура `xp_cmdshell` для выполнения команды `TYPE` операционной системы, с помощью

которой создается листинг файла сценария, и этот листинг возвращается в качестве результирующего набора.

Одна из особенностей, связанных с использованием рассматриваемой процедуры, состоит в том, что объект `Transfer` интерфейса DMO создает частичный результирующий набор, и это действие невозможно отменить. Как было указано выше, если некоторый метод, возвращающий выходное значение, вызывается с помощью процедуры `sp_OAMethod` без указания выходного параметра, то средства ODSOLE автоматически создают одностолбцовый и однострочный результирующий набор, содержащий требуемое значение. В данном случае происходит именно это. Автор не задает выходной параметр для вызова метода `ScriptTransfer`, поэтому интерфейс DMO создает для выходного значения, возвращаемого методом, результирующий набор. Таковым набором является текст сформированного сценария. Причина, по которой автор в данном случае не задает выходной параметр, состоит в том, что вызов метода `ScriptTransfer` завершается неудачей, если переданный этому методу параметр имеет слишком малый формат представления для того, чтобы в выходной параметр поместился весь сформированный сценарий. Учитывая то, что указанный метод не принимает параметры типа `text`, наибольший выходной параметр, который может быть передан в вызове метода, имеет длину 8 тыс. символов. Если же сценарий имеет длину больше 8 тыс. символов (что вполне вероятно даже для баз данных умеренных размеров), то вызов метода завершается неудачей, и вообще не формируется какой-либо сценарий. Поэтому автор не имеет иного выхода, кроме как пропустить выходной параметр (задавая на его месте значение `NULL`) и просто игнорировать ненужные фрагменты результирующего набора, вырабатываемые средствами ODSOLE.

Результатом применения указанной организации процедуры становится то, что после каждого вызова процедуры `sp_generate_script` формируется небольшой результирующий набор (фактически три небольших результирующих набора, по одному на каждый вызов метода `ScriptTransfer`), независимо от того, запрашивается в вызове процедуры результирующий набор или нет. Автор поместил в коде (непосредственно после последнего вызова метода `ScriptTransfer` объекта `Transfer`) оператор `PRINT` для вывода сообщения, которое указывает пользователю, что приведенные выше фиктивные выходные данные следует игнорировать, и это — все, что можно было сделать. К счастью, эти выходные данные совершенно безвредны — сценарий вырабатывается несмотря на указанное сообщение.

Код процедуры `sp_generate_script` приведен в листинге 15.9.

Листинг 15.9. Код процедуры `sp_generate_script`

```
USE master
GO
IF OBJECT_ID('sp_generate_script','P') IS NOT NULL
  DROP PROC sp_generate_script
/*
  Object: sp_generate_script
Description: Generates a creation script for an object or
    collection of objects
```

```
  Usage: sp_generate_script [@objectname='Object name or mask
    (defaults to all object in current database)']
[,@outputname='Output file name' (Default: @objectname+'.SQL', or
    GENERATED_SCRIPT.SQL for entire database)]
[,@scriptoptions=bitmask specifying script generation options]
[,@resultset=bit specifying whether to generate a result set
[,@includeheaders=bit specifying whether to generate descriptive
    headers for scripts
[,@server='server name'][, @username='user name']
    [, @password='password'][, @trustedconnection=1]
  Returns: (None)
  $Author: Ken Henderson $. Email: khen@khen.com
  $Revision: 8.0 $
  Example: sp_generate_script @objectname='authors',
    @outputname='authors.sql'
  Created: 1998-04-01.  $Modtime: 2003-04-23 $.
  */
GO
CREATE PROC sp_generate_script
  @objectname sysname=NULL,   -- Маска, обозначающая имена объектов, для
                              -- которых должен быть составлен сценарий
  @outputname sysname=NULL,   -- Имя создаваемого выходного файла (по
                              -- умолчанию - "GENERATED_SCRIPT.SQL")
  @scriptoptions int=NULL,    -- Битовая структура отображения для
                              -- объекта Transfer
  @resultset bit=1,           -- Флажок, который служит для указания
                              -- того, должен ли сценарий быть возвращен
                              -- в виде результирующего набора
  @trustedconnection bit=1,   -- Использовать заслуживающее доверия
                              -- соединение для подключения к серверу
  @IncludeHeaders bit=1,      -- Флажок, который указывает, должны ли
                              -- быть включены в сценарий описательные
                              -- заголовки
  @server sysname=@@SERVERNAME,  -- Имя сервера (по умолчанию задается
                                 -- равным значению переменной
                                 -- @@SERVERNAME)
@username sysname='sa',         -- Пользователь, от имени которого
                                -- устанавливается соединение с базой
                                -- данных (по умолчанию применяется
                                -- имя "sa")
  @password sysname=NULL          -- Пароль пользователя
AS

-- Переменные SQLDMO_SCRIPT_TYPE
DECLARE @SQLDMOScript_Default int
DECLARE @SQLDMOScript_Drops int
DECLARE @SQLDMOScript_ObjectPermissions int
DECLARE @SQLDMOScript_PrimaryObject int
DECLARE @SQLDMOScript_ClusteredIndexes int
DECLARE @SQLDMOScript_Triggers int
DECLARE @SQLDMOScript_DatabasePermissions int
DECLARE @SQLDMOScript_Permissions int
DECLARE @SQLDMOScript_ToFileOnly int
DECLARE @SQLDMOScript_Bindings int
DECLARE @SQLDMOScript_AppendToFile int
DECLARE @SQLDMOScript_NoDRI int
DECLARE @SQLDMOScript_UDDTsToBaseType int
```

```
DECLARE @SQLDMOScript_IncludeIfNotExists int
DECLARE @SQLDMOScript_NonClusteredIndexes int
DECLARE @SQLDMOScript_Indexes int
DECLARE @SQLDMOScript_Aliases int
DECLARE @SQLDMOScript_NoCommandTerm int
DECLARE @SQLDMOScript_DRIIndexes int
DECLARE @SQLDMOScript_IncludeHeaders int
DECLARE @SQLDMOScript_OwnerQualify int
DECLARE @SQLDMOScript_TimestampToBinary int
DECLARE @SQLDMOScript_SortedData int
DECLARE @SQLDMOScript_SortedDataReorg int
DECLARE @SQLDMOScript_TransferDefault int
DECLARE @SQLDMOScript_DRI_NonClustered int
DECLARE @SQLDMOScript_DRI_Clustered int
DECLARE @SQLDMOScript_DRI_Checks int
DECLARE @SQLDMOScript_DRI_Defaults int
DECLARE @SQLDMOScript_DRI_UniqueKeys int
DECLARE @SQLDMOScript_DRI_ForeignKeys int
DECLARE @SQLDMOScript_DRI_PrimaryKey int
DECLARE @SQLDMOScript_DRI_AllKeys int
DECLARE @SQLDMOScript_DRI_AllConstraints int
DECLARE @SQLDMOScript_DRI_All int
DECLARE @SQLDMOScript_DRIWithNoCheck int
DECLARE @SQLDMOScript_NoIdentity int
DECLARE @SQLDMOScript_UseQuotedIdentifiers int
-- Переменные SQLDMO_SCRIPT2_TYPE
DECLARE @SQLDMOScript2_Default int
DECLARE @SQLDMOScript2_AnsiPadding int
DECLARE @SQLDMOScript2_AnsiFile int
DECLARE @SQLDMOScript2_UnicodeFile int
DECLARE @SQLDMOScript2_NonStop int
DECLARE @SQLDMOScript2_NoFG int
DECLARE @SQLDMOScript2_MarkTriggers int
DECLARE @SQLDMOScript2_OnlyUserTriggers int
DECLARE @SQLDMOScript2_EncryptPWD int
DECLARE @SQLDMOScript2_SeparateXPs int

-- Значения SQLDMO_SCRIPT_TYPE
SET @SQLDMOScript_Default = 4
SET @SQLDMOScript_Drops = 1
SET @SQLDMOScript_ObjectPermissions = 2
SET @SQLDMOScript_PrimaryObject = 4
SET @SQLDMOScript_ClusteredIndexes = 8
SET @SQLDMOScript_Triggers = 16
SET @SQLDMOScript_DatabasePermissions = 32
SET @SQLDMOScript_Permissions = 34
SET @SQLDMOScript_ToFileOnly = 64
SET @SQLDMOScript_Bindings = 128
SET @SQLDMOScript_AppendToFile = 256
SET @SQLDMOScript_NoDRI = 512
SET @SQLDMOScript_UDDTsToBaseType = 1024
SET @SQLDMOScript_IncludeIfNotExists = 4096
SET @SQLDMOScript_NonClusteredIndexes = 8192
SET @SQLDMOScript_Indexes = 73736
SET @SQLDMOScript_Aliases = 16384
SET @SQLDMOScript_NoCommandTerm = 32768
SET @SQLDMOScript_DRIIndexes = 65536
```

```
SET @SQLDMOScript_IncludeHeaders = 131072
SET @SQLDMOScript_OwnerQualify = 262144
SET @SQLDMOScript_TimestampToBinary = 524288
SET @SQLDMOScript_SortedData = 1048576
SET @SQLDMOScript_SortedDataReorg = 2097152
SET @SQLDMOScript_TransferDefault = 422143
SET @SQLDMOScript_DRI_NonClustered = 4194304
SET @SQLDMOScript_DRI_Clustered = 8388608
SET @SQLDMOScript_DRI_Checks = 16777216
SET @SQLDMOScript_DRI_Defaults = 33554432
SET @SQLDMOScript_DRI_UniqueKeys = 67108864
SET @SQLDMOScript_DRI_ForeignKeys = 134217728
SET @SQLDMOScript_DRI_PrimaryKey = 268435456
SET @SQLDMOScript_DRI_AllKeys = 469762048
SET @SQLDMOScript_DRI_AllConstraints = 520093696
SET @SQLDMOScript_DRI_All = 532676608
SET @SQLDMOScript_DRIWithNoCheck = 536870912
SET @SQLDMOScript_NoIdentity = 1073741824
SET @SQLDMOScript_UseQuotedIdentifiers = -1

-- Значения SQLDMO_SCRIPT2_TYPE
SET @SQLDMOScript2_Default = 0
SET @SQLDMOScript2_AnsiPadding = 1
SET @SQLDMOScript2_AnsiFile = 2
SET @SQLDMOScript2_UnicodeFile = 4
SET @SQLDMOScript2_NonStop = 8
SET @SQLDMOScript2_NoFG = 16
SET @SQLDMOScript2_MarkTriggers = 32
SET @SQLDMOScript2_OnlyUserTriggers = 64
SET @SQLDMOScript2_EncryptPWD = 128
SET @SQLDMOScript2_SeparateXPs = 256

DECLARE @dbname sysname,
  @srvobject int,   -- Объект SQLServer
  @object int,      -- Рабочая переменная, применяемая для доступа
                    -- к COM-объектам
  @hr int,          -- Содержит значение HRESULT, возвращаемое
                    -- средствами COM
  @tfobject int,    -- Хранит указатель на объект Transfer
  @res int

SET @res=0

IF (@objectname IS NOT NULL) AND (CHARINDEX('%',@objectname)=0)
     AND (CHARINDEX('_',@objectname)=0) BEGIN
  SET @dbname=ISNULL(PARSENAME(@objectname,3),DB_NAME())
     -- Определить имя базы данных; по умолчанию применяется текущая
     -- база данных

  SET @objectname=PARSENAME(@objectname,1)   -- Удалить излишние
                                            -- элементы из имени
                                            -- таблицы
  IF (@objectname IS NULL) BEGIN
    RAISERROR('Invalid object name.',16,1)
    RETURN -1
  END
  IF (@outputname IS NULL)
```

```
    SET @outputname=@objectname+'.SQL'
END ELSE BEGIN
    SET @dbname=DB_NAME()
IF (@outputname IS NULL)
    SET @outputname='GENERATED_SCRIPT.SQL'
END

-- Создать объект SQLServer
EXEC @hr=sp_OACreate 'SQLDMO.SQLServer', @srvobject OUTPUT
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @srvobject, @hr
  RETURN
END

-- Установить соединение с сервером
IF (@trustedconnection=1) BEGIN
  EXEC @hr = sp_OASetProperty @srvobject, 'LoginSecure', 1
  IF (@hr <> 0) BEGIN
    EXEC sp_displayoaerrorinfo @srvobject, @hr
    GOTO ServerError
  END
  EXEC @hr = sp_OAMethod @srvobject, 'Connect', NULL, @server
END
ELSE BEGIN
  IF (@password IS NOT NULL)
  BEGIN
     EXEC @hr = sp_OAMethod @srvobject, 'Connect', NULL, @server,
        @username, @password
  END
  ELSE BEGIN
     EXEC @hr = sp_OAMethod @srvobject, 'Connect', NULL, @server,
        @username
  END
END

IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @srvobject, @hr
  GOTO ServerError
END

-- Создать объект Transfer
EXEC @hr=sp_OACreate 'SQLDMO.Transfer', @tfobject OUTPUT
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @tfobject, @hr
  GOTO FreeSrv
END
-- Задать свойство CopyData объекта Transfer
EXEC @hr = sp_OASetProperty @tfobject, 'CopyData', 0
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @tfobject, @hr
  GOTO FreeAll
END

-- Передать объекту Transfer указание, что должна быть создана
-- копия схемы
EXEC @hr = sp_OASetProperty @tfobject, 'CopySchema', 1
IF (@hr <> 0) BEGIN
```

```
  EXEC sp_displayoaerrorinfo @tfobject, @hr
  GOTO FreeAll
END

IF (@objectname IS NULL) BEGIN  -- Получить информацию обо всех
                                -- объектах в базе данных

  -- Передать объекту Transfer указание, что должны быть созданы копии
  -- всех объектов
  EXEC @hr = sp_OASetProperty @tfobject, 'CopyAllObjects', 1
  IF (@hr <> 0) BEGIN
     EXEC sp_displayoaerrorinfo @tfobject, @hr
     GOTO FreeAll
  END

  -- Передать объекту Transfer указание, что должна быть также
  -- получена информация о группах
  EXEC @hr = sp_OASetProperty @tfobject, 'IncludeGroups', 1
  IF (@hr <> 0) BEGIN
     EXEC sp_displayoaerrorinfo @tfobject, @hr
     GOTO FreeAll
  END

  -- Передать объекту Transfer указание, что должна быть включена
  -- информация о пользователях
  EXEC @hr = sp_OASetProperty @tfobject, 'IncludeUsers', 1
  IF (@hr <> 0) BEGIN
     EXEC sp_displayoaerrorinfo @tfobject, @hr
     GOTO FreeAll
  END

  -- Должна быть также включена информация о зависимостях
  EXEC @hr = sp_OASetProperty @tfobject, 'IncludeDependencies', 1
  IF (@hr <> 0) BEGIN
     EXEC sp_displayoaerrorinfo @tfobject, @hr
     GOTO FreeAll
  END
IF (@scriptoptions IS NULL) BEGIN
    SET @scriptoptions=@SQLDMOScript_OwnerQualify |
        @SQLDMOScript_Default | @SQLDMOScript_Triggers |
         @SQLDMOScript_Bindings | @SQLDMOScript_Permissions |
        @SQLDMOScript_Indexes | @SQLDMOScript_DRI_Defaults --|
        @SQLDMOScript_NoDRI
      IF @includeheaders=1 SET @scriptoptions=@scriptoptions |
        @SQLDMOScript_IncludeHeaders
  END

END -- IF (@objectname IS NULL)
ELSE BEGIN
  DECLARE @obname sysname,
    @obtype varchar(2),
    @obowner sysname,
    @OBJECT_TYPES varchar(50),
    @obcode int

  -- Приведенный ниже оператор служит для преобразования значения
  -- sysobjects.type в структуру отображения, которая требуется
```

```
  -- в объекте Transfer
  -- Не вносите изменения в приведенную ниже строку, поскольку эта
  -- строка служит в качестве таблицы преобразования
  SET @OBJECT_TYPES='T     V  U  P     D  R  TR          FN TF IF '

  -- Найти все объекты, которые соответствуют заданной маске, и ввести
  -- эти объекты в список объектов, для которых объектом Transfer
  -- составляется сценарий
  DECLARE ObjectList CURSOR FOR
  SELECT name,CASE type WHEN 'TF' THEN 'FN' WHEN 'IF' THEN 'FN'
      ELSE type END AS type,USER_NAME(uid) FROM sysobjects
  WHERE (name LIKE @objectname)
    AND (CHARINDEX(type+' ',@OBJECT_TYPES)<>0)
    AND (OBJECTPROPERTY(id,'IsSystemTable')=0)
    AND (status>0)
  UNION ALL  -- Включить определяемые пользователем типы данных
  SELECT name,'T',USER_NAME(uid)
  FROM SYSTYPES
  WHERE (usertype & 256)<>0
  AND (name LIKE @objectname)

  OPEN ObjectList

  FETCH ObjectList INTO @obname, @obtype, @obowner
  WHILE (@@FETCH_STATUS=0) BEGIN
    SET @obcode=POWER(2,(CHARINDEX(@obtype+' ',@OBJECT_TYPES)/3))
EXEC @hr = sp_OAMethod @tfobject, 'AddObjectByName', NULL,
        @obname, @obcode, @obowner
    IF (@hr <> 0) BEGIN
      EXEC sp_displayoaerrorinfo @tfobject, @hr
      GOTO FreeAll
    END

    FETCH ObjectList INTO @obname, @obtype, @obowner  END
  CLOSE ObjectList
  DEALLOCATE ObjectList

  IF (@scriptoptions IS NULL)
     SET @scriptoptions=@SQLDMOScript_Default -- Если сценарий не
                                              -- создается для всей
                                              -- базы данных, применять
                                              -- простой подход
    IF @includeheaders=1 SET @scriptoptions=@scriptoptions |
        @SQLDMOScript_IncludeHeaders
END  -- ELSE IF (@objectname IS NULL)

-- Задать свойство ScriptType объекта Transfer
EXEC @hr = sp_OASetProperty @tfobject, 'ScriptType', @scriptoptions
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @tfobject, @hr
  GOTO FreeAll
END

-- Задать свойство Script2Type объекта Transfer
EXEC @hr = sp_OASetProperty @tfobject, 'Script2Type',
    @SQLDMOScript2_NoFG
IF (@hr <> 0) BEGIN
```

```
  EXEC sp_displayoaerrorinfo @tfobject, @hr
  GOTO FreeAll
END

-- Получить указатель на базу данных
DECLARE @itemname varchar(255)
SET @itemname='Databases.Item("'+@dbname+'")'
EXEC @hr = sp_OAGetProperty @srvobject, @itemname, @object OUT
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @srvobject, @hr
  GOTO FreeAll
END
DECLARE @cmd varchar(8000)

-- Вызвать метод Transfer объекта Database для передачи информации
-- о схемах в файл
-- Вначале в сценарий вводятся команды создания объектов без
-- декларативных ограничений ссылочной целостности, затем вводятся
-- команды создания первичных ключей, а после этого - внешних ключей
EXEC @hr = sp_OAMethod @object, 'ScriptTransfer',NULL, @tfobject,
    2,@outputname
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  GOTO FreeAll
  END
-- Теперь извлекается информация о первичных ключах и уникальных ключах
-- (и команды создания этих ключей добавляются к первоначальному файлу
-- сценария)
-- Информация о первичных ключах и уникальных ключах извлекается
-- отдельно из самих таблиц, поскольку при получении информации о
-- первичных ключах иногда извлекается также информация о внешних
-- ключах, несмотря на то, что эта информация не была затребована
SET @scriptoptions=@SQLDMOScript_NoDRI |
    @SQLDMOScript_DRI_PrimaryKey | @SQLDMOScript_DRI_UniqueKeys |
    @SQLDMOScript_AppendToFile | @SQLDMOScript_OwnerQualify
IF @includeheaders=1 SET @scriptoptions=@scriptoptions |
     @SQLDMOScript_IncludeHeaders

-- Повторно задать свойство ScriptType объекта Transfer
EXEC @hr = sp_OASetProperty @tfobject, 'ScriptType', @scriptoptions
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @tfobject, @hr
  GOTO FreeAll
END

EXEC @hr = sp_OAMethod @object, 'ScriptTransfer',NULL, @tfobject,
    2,@outputname
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  GOTO FreeAll
END

-- Теперь извлекается информация о внешних ключах (и команды создания
-- этих ключей добавляются к первоначальному файлу сценария)
SET @scriptoptions=@SQLDMOScript_NoDRI |
    @SQLDMOScript_DRI_ForeignKeys | @SQLDMOScript_DRI_Checks |
    @SQLDMOScript_DRI_Defaults | @SQLDMOScript_AppendToFile |
```

```
    @SQLDMOScript_OwnerQualify
IF @includeheaders=1 SET @scriptoptions=@scriptoptions |
     @SQLDMOScript_IncludeHeaders

-- Повторно задать свойство ScriptType объекта Transfer
EXEC @hr = sp_OASetProperty @tfobject, 'ScriptType', @scriptoptions
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @tfobject, @hr
  GOTO FreeAll
END

-- Сформировать последний раздел сценария
EXEC @hr = sp_OAMethod @object, 'ScriptTransfer',NULL, @tfobject,
    2,@outputname
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  GOTO FreeAll
END

IF (@resultset=1) BEGIN
  SET @cmd='TYPE "'+@outputname+'"'
  exec master.dbo.xp_cmdshell @cmd
END

GOTO FreeAll

ServerError:
SET @res=-1
RAISERROR ('Error generating script', 16, 1)

FreeAll:
EXEC sp_OADestroy @tfobject  -- Применение этой команды - признак
                             -- правильного стиля программирования

FreeSrv:
EXEC sp_OADestroy @srvobject

RETURN @res
GO
USE Northwind
GO
EXEC sp_generate_script  'Customers', @server='khenmp\ss2000'

(Результаты приведены в сокращенном виде)

Column1
------------------------------------------------------------------
set quoted_identifier  OFF
GO
CREATE TABLE [Customers] (
  [CustomerID] [nchar] (5) COLLATE SQL_Latin1_General_CP1_CI_AS NO
  [CompanyName] [nvarchar] (40) COLLATE SQL_Latin1_General_CP1_CI_
  [ContactName] [nvarchar] (30) COLLATE SQL_Latin1_General_CP1_CI_
  [ContactTitle] [nvarchar] (30) COLLATE SQL_Latin1_General_CP1_CI
  [Address] [nvarchar] (60) COLLATE SQL_Latin1_General_CP1_CI_AS N
  [City] [nvarchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
  [Region] [nvarchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NU
```

```
  [PostalCode] [nvarchar] (10) COLLATE SQL_Latin1_General_CP1_CI_A
  [Country] [nvarchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS N
  [Phone] [nvarchar] (24) COLLATE SQL_Latin1_General_CP1_CI_AS NUL
  [Fax] [nvarchar] (24) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
  [rowguid]  uniqueidentifier ROWGUIDCOL  NOT NULL CONSTRAINT
  [DF__Customers__rowgu__0EF836A4] DEFAULT (newid()),
  CONSTRAINT [PK_Customers] PRIMARY KEY  CLUSTERED
  (
    [CustomerID]
  )  ON [PRIMARY]
) ON [PRIMARY]
GO

(1 row(s) affected)

NOTE: Ignore the code displayed above.  It's a remnant of the
SQL-DMO method used to produce the script file
line
------------------------------------------------------------------------
set quoted_identifier  OFF
GO

CREATE TABLE [Customers] (
  [CustomerID] [nchar] (5) COLLATE SQL_Latin1_General_CP1_CI_AS NO
  [CompanyName] [nvarchar] (40) COLLATE SQL_Latin1_General_CP1_CI_
  [ContactName] [nvarchar] (30) COLLATE SQL_Latin1_General_CP1_CI_
  [ContactTitle] [nvarchar] (30) COLLATE SQL_Latin1_General_CP1_CI
  [Address] [nvarchar] (60) COLLATE SQL_Latin1_General_CP1_CI_AS N
  [City] [nvarchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
  [Region] [nvarchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NU
  [PostalCode] [nvarchar] (10) COLLATE SQL_Latin1_General_CP1_CI_A
  [Country] [nvarchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS N
  [Phone] [nvarchar] (24) COLLATE SQL_Latin1_General_CP1_CI_AS NUL
  [Fax] [nvarchar] (24) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
  [rowguid]  uniqueidentifier ROWGUIDCOL  NOT NULL CONSTRAINT
  [DF__Customers__rowgu__0EF836A4] DEFAULT (newid()),
  CONSTRAINT [PK_Customers] PRIMARY KEY  CLUSTERED
  (
    [CustomerID]
  )  ON [PRIMARY]
) ON [PRIMARY]
GO
```

В разделе этого листинга, содержащем полученные результаты, все данные, предшествующие выделенному полужирным шрифтом сообщению, которое было выведено с помощью оператора PRINT, представляют собой фиктивный вывод, который можно смело игнорировать. А выходные данные, которые следуют за сообщением, представляют собой фактически сформированный сценарий. В этом случае сценарий был сформирован для таблицы Customers из базы данных Northwind. С помощью рассматриваемой процедуры можно было бы с такой же легкостью составить сценарий для всей базы данных или задать маску для формирования сценария, относящегося сразу к нескольким таблицам.

Процедура начинается с создания экземпляров объектов SQLServer и Transfer интерфейса DMO. Объект SQLServer интерфейса DMO представляет собой путь доступа этого интерфейса на уровне корневого объекта, поскольку объект SQLServer можно использовать для подключения к серверу и для доступа к другим объектам на сервере. Объект Transfer инкапсулирует объект связи между сервером и сервером, или сервером и файлом, а также средства передачи данных интерфейса DMO. Объект Transfer применяется в процедуре sp_generate_script для формирования сценариев SQL.

У читателей, причастных в какой-то степени к программированию DMO, может возникнуть вопрос – почему автор использует объект Transfer, а не вызывает метод Script отдельных объектов. Ответ состоит в том, что принятый в данном приложении подход к организации процедуры выбран в целях сохранения зависимостей между объектами в максимально возможной степени. Объект Transfer записывает в сценарий информацию схемы для объекта базы данных с учетом порядка определения зависимостей, который обозначен в таблице sysdepends. Хотя такой способ нельзя рассматривать как полностью надежный подход к определению зависимостей между объектами, лучше использовать его, чем вообще ничего, и, к сожалению, данный способ учета зависимостей – это все, что имеется в нашем распоряжении. Объект Transfer был первоначально предназначен для поддержки передачи содержимого одной базы данных в другую, поэтому в коде этого объекта разработчикам пришлось учитывать порядок создания объектов. В противном случае выполнение операторов CREATE для создания объектов базы данных, зависящих от других объектов, окончится неудачей, если требуемые для них объекты еще не были созданы. Рассмотрим ограничение внешнего ключа. Если в таблице Order Details имеется ссылка внешнего ключа на таблицу Products, то таблица Products должна существовать до выполнения операции создания таблицы Order Details; если же таблица Products не будет создана раньше, то попытка выполнить оператор CREATE TABLE для создания таблицы Order Details окончится неудачей. В объекте Transfer предпринимается попытка обеспечить успешное создание объектов путем проверки зависимостей между объектами во время составления сценариев базы данных.

На этапе выявления зависимостей между объектами, осуществляемом объектом Transfer, могут возникнуть ошибки, обусловленные неправильной или недостающей информацией в таблице sysdepends. Поэтому в процедуре sp_generate_script предпринимается дополнительный шаг – разбивка процесса формирования сценария на три этапа с учетом общих зависимостей между объектами. На первом этапе в сценарии представляются требуемые объекты без учета декларативных ограничений ссылочной целостности (Declarative Referential Integrity – DRI) какого-либо рода. На втором этапе в сценарии отображаются ограничения первичного ключа и уникального ключа для выбранных объектов. А на третьем этапе в сценарии представляются ограничения внешнего ключа для указанных объектов. Это позволяет надежно выполнять сценарий для воссоздания базы данных, даже если порядок определения зависимостей, отраженный в таблице sysdepends и используемый в интерфейсе DMO, является неправильным.

После создания объекта Transfer процедура определяет, желает ли пользователь составить сценарий для всей базы данных или только для выбранных объектов.

Это различие важно, поскольку в интерфейсе DMO при составлении сценария для всей базы данных предпринимаются попытки составить список объектов в порядке определения зависимостей, как уже было сказано. Если же сценарий должен быть составлен для подмножества объектов базы данных, то в процедуре открывается курсор на таблицах sysobjects и systypes (с помощью команды UNION ALL) и вызывается метод AddObjectByName объекта Transfer для подготовки одного за другим объектов базы данных в целях составления по ним сценария.

Затем в процедуре с помощью объекта SQLServer отыскивается база данных, в которой находятся объекты, используемые для составления сценария. Процедура находит базу данных, обращаясь к коллекции Databases объекта SQLServer. Объекты DMO часто предоставляют доступ к коллекциям других объектов. К элементам этих коллекций можно обращаться по имени или с помощью порядкового индекса. В процедуре sp_generate_script доступ к элементам коллекции всегда осуществляется по имени.

После выборки в процедуре указателя на нужную базу данных вызывается метод ScriptTransfer этой базы данных, и указанному методу передается в качестве параметра ранее созданный объект Transfer. В результате этого создается сценарий SQL, содержащий указанные объекты.

В последнем шаге этой процедуры осуществляется возврат полученного сценария в виде результирующего набора. Обычно вызывающий объект рассчитан на то, что в него поступит непосредственно сам сценарий. Если применяется значение параметра @resultset = 1 (предусмотренное по умолчанию), то в процедуре sp_generate_script вызывается расширенная процедура xp_cmdshell для вызова на выполнение команды TYPE операционной системы; эта команда создает листинг файла и возвращает его в качестве результирующего набора. Удобным вариантом способа возврата сценария мог бы служить возврат указателя курсора на сценарий, но реализацию этого варианта оставляем в качестве упражнения для читателя.

Использование средств ODSOLE для автоматизации производных объектов

В данном разделе будет показано, как автоматизировать создаваемые пользователем COM-объекты. В нем рассматривается задача автоматизации библиотеки функций Visual Basic, которая заключена автором в COM-объект, обозначенный именем VBODSOLELib. Начнем с изучения листинга 15.10 – исходного кода библиотеки VBODSOLELib на языке Visual Basic. (Полный исходный код этой библиотеки приведен в подкаталоге VBODSOLELib каталога CH15 компакт-диска, прилагаемого к данной книге.)

Листинг 15.10. Исходный код библиотеки VBODSOLELib

```
Option Explicit
Dim GlobalArray() As Variant
Dim lGlobalArraySize As Long
```

```
' Строковые функции

Public Function VBInStrRev(strCheck As String, strMatch As String)
     As String
  VBInStrRev = InStrRev(strCheck, strMatch)
End Function

Public Function VBStrReverse(strIn As String) As String
  VBStrReverse = StrReverse(strIn)
End Function

Public Function VBFormat(vExpr As Variant, strFormat As String)
     As String
  VBFormat = Format(vExpr, strFormat)
End Function

Public Function VBHex(vExpr As Variant) As String
  VBHex = Hex(vExpr)
End Function

Public Function VBOct(vExpr As Variant) As String
  VBOct = Oct(vExpr)
End Function

Public Function VBLike(strMatch As String, strExpr As String)
    As Boolean
  VBLike = (strMatch Like strExpr)
End Function

Public Function VBScriptRegEx(strPattern As String, strMatch
    As String) As Long
  Dim regEx, Match, Matches
  Set regEx = CreateObject("VBScript.RegExp")
  regEx.Pattern = strPattern
  regEx.IgnoreCase = True
  Set Matches = regEx.Execute(strMatch)
If Not IsEmpty(Matches) Then
    For Each Match In Matches
      VBScriptRegEx = Match.FirstIndex + 1 ' С отсчетом от нуля
      Exit For
    Next
  Else
    VBScriptRegEx = 0
  End If
End Function

Public Function VBScriptRegExTest(strPattern As String, strMatch
    As String) As Boolean
  Dim regEx, Match, Matches
  Set regEx = CreateObject("VBScript.RegExp")
  regEx.Pattern = strPattern
  regEx.IgnoreCase = True
  VBScriptRegExTest = regEx.Test(strMatch)
End Function

' Прочие функции
```

```
Public Function VBShell(strCommandLine As String, Optional
    iWindowStyle As Variant) As Double
  VBShell = Shell(strCommandLine, IIf(IsMissing(iWindowStyle),
    vbNormalFocus, iWindowStyle))
End Function

' Финансовые функции

Public Function VBFV(nRate As Double, nPer As Double, nPmt
    As Double, Optional vPv As Variant, Optional vType As Variant)
  VBFV = FV(nRate, nPer, nPmt, IIf(IsMissing(vPv), 0, vPv),
    IIf(IsMissing(vType), 0, vType))
End Function

Public Function VBIPmt(nRate As Double, nPer As Double,
    nPmtPeriods As Double, nPV As Double, Optional vFv As
    Variant, Optional vType As Variant)
  VBIPmt = IPmt(nRate, nPer, nPmtPeriods, nPV,
    IIf(IsMissing(vFv), 0, vFv), IIf(IsMissing(vType), 0, vType))
End Function

Public Function VBNPer(nRate As Double, nPmt As Double, nPV
End Sub
    As Double, Optional vFv As Variant, Optional vType As Variant)
  VBNPer = nPer(nRate, nPmt, nPV, IIf(IsMissing(vFv), 0, vFv),
    IIf(IsMissing(vType), 0, vType))
End Function

Public Function VBPmt(nRate As Double, nPer As Double, nPV
    As Double, Optional vFv As Variant, Optional vType As Variant)
  VBPmt = Pmt(nRate, nPer, nPV, IIf(IsMissing(vFv), 0, vFv),
    IIf(IsMissing(vType), 0, vType))
End Function

Public Function VBPPmt(nRate As Double, nPer As Double,
    nPmtPeriods As Double, nPV As Double, Optional vFv As Variant,
    Optional vType As Variant)
  VBPPmt = PPmt(nRate, nPer, nPmtPeriods, nPV,
    IIf(IsMissing(vFv), 0, vFv), IIf(IsMissing(vType), 0, vType))
End Function

Public Function VBPV(nRate As Double, nPer As Double, nPmt
    As Double, Optional vFv As Variant, Optional vType As Variant)
  VBPV = PV(nRate, nPer, nPmt, IIf(IsMissing(vFv), 0, vFv),
    IIf(IsMissing(vType), 0, vType))
End Function

' Процедуры

Public Sub VBAppActivate(strTitle As String, Optional bWait
    As Variant)
  AppActivate strTitle, IIf(IsMissing(bWait), False, bWait)
End Sub

Public Sub VBSendKeys(strKeys, Optional bWait As Variant)
  SendKeys strKeys, IIf(IsMissing(bWait), False, bWait)
```

```
End Sub

Public Sub VBAppActivateAndSendKeys(strTitle As String, strKeys
    As String, Optional bWait As Variant)
  AppActivate strTitle, IIf(IsMissing(bWait), False, bWait)
  SendKeys strKeys, IIf(IsMissing(bWait), False, bWait)
End Sub

Public Sub VBFileCopy(strSource As String, strDestination
   As String)
  FileCopy strSource, strDestination
End Sub

Public Sub VBFileErase(strFileName As String)
  Kill strFileName
End Sub

Public Sub VBMkDir(strDirName As String)
  MkDir strDirName
End Sub

Public Sub VBRmDir(strDirName As String)
  RmDir strDirName
End Sub
```

Автор не будет подробно описывать все представленные в этой библиотеке функции. Читатель уже должен иметь достаточно полное представление о том, как обращаться к этим функциям из языка T-SQL с помощью процедур sp_OA. После регистрации библиотеки DLL этого объекта с помощью вызова программы `regsvr32` доступ к методам объекта можно получить из языка T-SQL точно по такому же принципу, как и к методам других COM-объектов, причем точно так же, как было показано в других примерах данной главы.

Большинство из реализованных в этой библиотеке функций не существует в языке Transact-SQL, но можно видеть, что многие из них весьма полезны. В этой библиотеке представлены финансовые функции (например, `VBIPmt` — функция вычисления процентных платежей по ренте); функции манипулирования со строками (например, `VBInStrRev` — функция поиска в строке в обратном направлении), системные функции (например, `VBAppActivate` — функция активизации приложения), причем для использования некоторых из этих системных функций необходимо вызвать на выполнение программу SQL Server как терминальное приложение; функции для работы с регулярными выражениями (например, `VBScriptRegEx` и `VBScriptRegExTest`) и многие другие. Подкаталог `VBODSOLELib` указанного каталога компакт-диска включает несколько простых сценариев T-SQL, которые показывают, как применяются эти функции. В листинге 15.11 приведен пример, в котором вызывается функция `VBInStrRev`.

Листинг 15.11. Использование функции `VBInStrRev`

```
declare @obj int
declare @hr int
declare @songs varchar(255)
```

```
set @songs='Sister Christian, Dance, Boys of Summer, The Dance'
declare @pos int
exec @hr=sp_OACreate 'VBODSOLE.VBODSOLELib', @obj OUT
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  RETURN
END
exec @hr=sp_OAMethod @obj, 'VBInStrRev', @pos OUT, @songs, 'Dance'
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  RETURN
END

select @pos

exec @hr=sp_OADestroy @obj
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  RETURN
END

(Результаты)

-----------
46
```

Создание массивов в языке T-SQL с помощью COM-объектов

Один из разделов библиотеки VBODSOLELib будет описан более подробно. Это раздел, содержащий ряд функций обработки массивов языка Visual Basic. Такие функции позволяют получить доступ к основным службам для работы с массивами в языке Transact-SQL. В своей последней книге, *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML*, автор ввел поддержку массивов в язык Transact-SQL с использованием расширенных процедур и системных функций. А в данной книге принят другой подход. В этом случае поддержка массивов вводится в язык T-SQL с помощью описанного выше COM-объекта VBODSOLELib и некоторых системных функций. Начнем с рассмотрения кода поддержки массивов из библиотеки VBODSOLELib (листинг 15.12).

Листинг 15.12. Код поддержки массивов из библиотеки VBODSOLELib

```
Public Function VBCreateArray(lSize As Long) As Long
  Dim vArray()
  ReDim vArray(lSize)
  If IsEmpty(lGlobalArraySize) Then
    lGlobalArraySize = 0
  Else
    lGlobalArraySize = lGlobalArraySize + 1
  End If
  ReDim Preserve GlobalArray(lGlobalArraySize)
```

```
GlobalArray(lGlobalArraySize) = vArray()
  VBCreateArray = lGlobalArraySize
End Function

Public Function VBGetArray(lGlobalIndex As Long, lIndex As Long)
    As Variant
  VBGetArray = GlobalArray(lGlobalIndex)(lIndex - 1)
End Function

Public Sub VBSetArray(lGlobalIndex As Long, lIndex As Long, vVal
    As Variant)
  GlobalArray(lGlobalIndex)(lIndex - 1) = vVal
End Sub

Public Sub VBDestroyArray(lGlobalIndex As Long)
  Set GlobalArray(lGlobalIndex) = Null
End Sub

Public Function VBCreateArraySplit(strIn As String, Optional
    strDelim As Variant) As Long
  If IsEmpty(lGlobalArraySize) Then
    lGlobalArraySize = 0
  Else
    lGlobalArraySize = lGlobalArraySize + 1
  End If
  ReDim Preserve GlobalArray(lGlobalArraySize)
  GlobalArray(lGlobalArraySize) = Split(strIn,
      IIf(IsMissing(strDelim), " ", strDelim))
  VBCreateArraySplit = lGlobalArraySize
End Function

Public Function VBArrayJoin(lGlobalIndex As Long, Optional
    strDelim As Variant) As String
  VBArrayJoin = Join(GlobalArray(lGlobalIndex),
      IIf(IsMissing(strDelim), " ", strDelim))
End Function

Public Function VBListArray(lGlobalIndex As Long) As Variant
  VBListArray = GlobalArray(lGlobalIndex)
End Function

Public Function VBArrayLen(lGlobalIndex As Long) As Long
  VBArrayLen = UBound(GlobalArray(lGlobalIndex))
End Function
```

В этом коде представлены восемь описанных ниже функций.

- `VBCreateArray`. Создать массив и возвратить дескриптор этого массива.
- `VBGetArray`. Получить значение элемента массива.
- `VBSetArray`. Задать значение элемента массива.
- `VBDestroyArray`. Уничтожить массив.
- `VBCreateArraySplit`. Создать массив путем разбиения строки с разграничителями на элементы.

- VBArrayJoin. Возвратить значения элементов массива в виде строки с разграничителями.
- VBListArray. Возвратить массив в переменной типа Variant.
- VBArrayLen. Возвратить значение количества элементов в массиве.

Назначение каждой из этих функций в основном не требует пояснений. По существу, автор просто взял основные функции поддержки массивов языка Visual Basic и заключил их в COM-объект для того, чтобы эти функции стали доступными из языка T-SQL. В листинге 15.13 представлен пример сценария, который показывает, как можно вызвать эти функции из языка T-SQL.

Листинг 15.13. Вызов основных функций поддержки массивов

```
declare @obj int
declare @hr int
declare @arr int
exec @hr=sp_oacreate 'VBODSOLE.VBODSOLELib', @obj OUT
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  RETURN
END

exec @hr=sp_oamethod @obj, 'VBCreateArray', @arr OUT, 10
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  Goto Cleanup
END

exec @hr=sp_oamethod @obj, 'VBSetArray', NULL, @arr, 3, 'foo'
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  Goto Cleanup
END

declare @val varchar(30)
exec @hr=sp_oamethod @obj, 'VBGetArray', @val OUT, @arr, 3
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  Goto Cleanup
END

SELECT @val

DECLARE @len int

exec @hr=sp_oamethod @obj, 'VBArrayLen', @len OUT, @arr
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  Goto Cleanup
END

SELECT @len

DECLARE @dummy int
```

```
exec @hr=sp_oamethod @obj, 'VBListArray', @dummy OUT, @arr
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @obj, @hr
  Goto Cleanup
END

Cleanup:

exec @hr=sp_oadestroy @obj

(Результаты приведены в сокращенном виде)

------------------------------
foo

-----------
10

Column0     Column1     Column2     Column3
----------- ----------- ----------- -----------
0           0           foo         0
```

В отношении приведенного выше кода можно сделать целый ряд интересных замечаний. Прежде всего заслуживает внимания то, что в качестве дескриптора массива используется целочисленный индекс. Читатели, которые прочли последнюю книгу автора, могут задать вопрос — почему в качестве дескриптора массива предусмотрен возврат индекса, а не указателя на сам массив, как было предусмотрено в подходе к поддержке массивов на основе расширенных процедур, реализованном в предыдущей книге. Причина использования в настоящей книге указанного варианта состоит в том, что средства ODSOLE в стремлении упростить работу разработчика не позволяют возвращать результаты с типом массива из какого-либо метода Automation. При попытке возвратить массив средства ODSOLE автоматически заполняют значениями NULL выходной параметр, в который должно было быть передано полученное значение, и преобразуют массив в результирующий набор TDS. Как было указано выше, если возвращаемый массив представляет собой одномерный массив, то преобразование в результирующий набор приводит к получению одной строки, каждый столбец которой соответствует одному из элементов массива. Если массив имеет два измерения, то создается многострочный результирующий набор. А если массив имеет больше двух измерений или хранит данные сложных типов, таких как структуры, то активизируется ошибка.

Итак, из методов COM нельзя возвратить массив, не избежав преобразования этого массива средствами ODSOLE в результирующий набор, поэтому в объекте `VBODSOLELib` распределяется "массив массивов" — массив переменных типа `Variant`, каждая из которых хранит отдельный массив. Каждый раз, когда возникает необходимость распределить новый массив, указанный главный массив переменных типа `Variant` переопределяется с помощью оператора `ReDim` для включения в него нового слота, а затем новый массив распределяется в новом слоте. Этот главный массив во многом напоминает двумерный массив, который поддерживает зубчатые границы (т.е. может иметь разное количество элементов одной размерности).

После успешного определения нового массива с помощью оператора Dim возвращается индекс слота этого массива в главном массиве класса. Этот индекс служит в качестве дескриптора массива. При выполнении каждой операции доступа к массиву следует всегда указывать индекс массива в главном массиве с помощью такого дескриптора.

Обратите внимание на то, что применяемый по умолчанию в средствах ODSOLE подход, предусматривающий преобразование массивов в результирующие наборы, мы обращаем в свою пользу в методе VBListArray. Метод VBListArray фактически не формирует листинг массива, а просто возвращает переменную типа Variant, содержащую массив. Средство ODSOLE, обнаружив это значение, возвращаемое в виде массива, преобразует массив в результирующий набор и тем самым формирует требуемый листинг.

Очевидно, что достаточно перспективной является сама возможность получения доступа к описанным функциональным возможностям с помощью вызовов процедур sp_OA, но, как и в применяемом автором подходе поддержки массивов на основе расширенных процедур, который был описан в предыдущей книге, в рассматриваемом в данной книге подходе вызовы указанных процедур sp_OA заключены в системных пользовательских функциях для того, чтобы средства поддержки массивов можно было легко использовать во всем сервере. В листинге 15.14 приведен исходный код сценария, в котором создаются указанные пользовательские функции.

Листинг 15.14. Пользовательские функции поддержки массивов

```
USE master
GO
EXEC sp_configure 'allow updates',1
GO
RECONFIGURE WITH OVERRIDE
GO
DROP FUNCTION system_function_schema.fn_createobject,
              system_function_schema.fn_destroyobject,
              system_function_schema.fn_createarray,
               system_function_schema.fn_setarray,
               system_function_schema.fn_getarray,
              system_function_schema.fn_destroyarray,
               system_function_schema.fn_arraylen,
              system_function_schema.fn_listarray
GO
CREATE FUNCTION system_function_schema.fn_createobject()
RETURNS int
AS
BEGIN
  DECLARE @obj int
  DECLARE @hr int
  exec @hr=sp_OACreate 'VBODSOLE.VBODSOLELib', @obj OUT
  IF (@hr <> 0) BEGIN
    RETURN @hr
  END
  RETURN(@obj)
END
GO
CREATE FUNCTION system_function_schema.fn_destroyobject(@obj int)
```

```
RETURNS int
AS
BEGIN
  DECLARE @hr int
  exec @hr=sp_OADestroy @obj
  RETURN(@hr)
END
GO
CREATE FUNCTION system_function_schema.fn_createarray(@obj int,
    @size int)
RETURNS int
AS
BEGIN
  DECLARE @hr int
  DECLARE @hdl int
  exec @hr=sp_OAMethod @obj, 'VBCreateArray', @hdl OUT, @size
  IF (@hr <> 0) BEGIN
    RETURN @hr
  END
  RETURN(@hdl)
END
GO
CREATE FUNCTION system_function_schema.fn_destroyarray(@obj int,
     @hdl int)
RETURNS int
AS
BEGIN
DECLARE @hr int
exec @hr=sp_oamethod @obj, 'VBDestoryArray', NULL, @hdl
IF (@hr <> 0) BEGIN
  RETURN @hr
END
RETURN 0
END
GO
CREATE FUNCTION system_function_schema.fn_setarray(@obj int,
    @hdl int, @index int, @value sql_variant)
RETURNS int
AS
BEGIN
DECLARE @hr int
exec @hr=sp_OAMethod @obj, 'VBSetArray', NULL, @hdl, @index,
     @value
IF (@hr <> 0) BEGIN
  RETURN @hr
END
RETURN 0
END
GO
CREATE FUNCTION system_function_schema.fn_getarray(@obj int,
    @hdl int, @index int)
RETURNS sql_variant
AS
BEGIN
DECLARE @hr int, @valuestr varchar(8000)
exec @hr=sp_oamethod @obj, 'VBGetArray', @valuestr OUT, @hdl,
     @index
```

```
IF (@hr <> 0) BEGIN
  RETURN @hr
END
RETURN(@valuestr)
END
GO
CREATE FUNCTION system_function_schema.fn_arraylen(@obj int,
     @hdl int)
RETURNS int
AS
BEGIN
DECLARE @hr int, @len int
exec @hr=sp_oamethod @obj, 'VBArrayLen', @len OUT, @hdl
IF (@hr <> 0) BEGIN
  RETURN @hr
END
RETURN @len
END
GO
CREATE FUNCTION system_function_schema.fn_listarray(@obj int,
    @hdl int)
RETURNS @array TABLE (idx int, value sql_variant)
AS
BEGIN
  DECLARE @i int, @cnt int
  SET @cnt=fn_arraylen(@obj,@hdl)
  SET @i=1
  WHILE (@i<=@cnt) BEGIN
    INSERT @array VALUES (@i, fn_getarray(@obj,@hdl,@i))
    SET @i=@i+1
  END
  RETURN
END
GO
EXEC sp_configure 'allow updates',0
GO
RECONFIGURE WITH OVERRIDE
GO
```

Благодаря тому, что вызовы процедур sp_OA заключены в системные пользовательские функции, указанные функции поддержки массивов, основанные на использовании COM-объектов, становятся гораздо более удобными в работе. В листинге 15.15 показано, как использовать эти функции.

Листинг 15.15. Использование функций поддержки массивов

```
DECLARE @obj int, @hdl int, @siz int, @res int
SET @siz=1000

-- Создать массив и возвратить дескриптор массива и значение его длины
SET @obj=fn_createobject()
SET @hdl=fn_createarray(@obj,@siz)
SELECT @hdl, fn_arraylen(@obj,@hdl)

-- Задать значения элементов с индексами 1, 10, 998 и 1000
```

```
SELECT @res=fn_setarray(@obj,@hdl,1,'test1'),
@res=fn_setarray(@obj,@hdl,10,'test10'),
@res=fn_setarray(@obj,@hdl,998,'test998'),
@res=fn_setarray(@obj,@hdl,1000,'test1000')

-- Получить значение элемента с индексом 10
SELECT fn_getarray(@obj,@hdl,10)

-- Получить значение элемента с индексом 998
SELECT fn_getarray(@obj,@hdl,998)

-- Вывести на терминал значения элементов массива
SELECT * FROM ::fn_listarray(@obj, @hdl)
WHERE value IS NOT NULL

SET @res=fn_destroyarray(@obj,@hdl)
SET @res=fn_destroyobject(@obj)

(Результаты)

----------- -----------
1           1000

-------------------------------------------------------------------
test10

-------------------------------------------------------------------
test998

idx         value
----------- -------------------------------------------------------
1           test1
10          test10
998         test998
1000        test1000
```

Вполне очевидно, что теперь в языке T-SQL можно чрезвычайно легко вначале создать массив, затем добавлять к нему элементы и осуществлять выборку значений этих элементов. Заслуживает внимания то, что в основе рассматриваемых средств доступа лежат функции, поэтому сами средства доступа можно непосредственно использовать для обработки данных в таблицах и представлениях, как показано в листинге 15.16.

Листинг 15.16. Использование функций поддержки массивов для работы с данными таблиц и представлений

```
DECLARE @o int, @h int, @res int, @arraybase int

-- Создать объект и массив
SET @o=fn_createobject()
SELECT @h=fn_createarray(@o,1000), @arraybase=10247

-- Загрузить в массив информацию обо всех датах заказов
SELECT @res=fn_setarray(@o,@h,OrderId-@arraybase,OrderDate)
```

```
FROM Northwind..orders

-- Вывести на терминал значение одного из элементов массива
SELECT idx+@arraybase AS OrderId, value AS OrderDate
FROM ::fn_listarray(@o,@h)
WHERE idx=10249-@arraybase

-- Уничтожить массив и объект
SET @res=fn_destroyarray(@o,@h)
SET @res=fn_destroyobject(@o)

(Результаты)

OrderId     OrderDate
----------- -----------------------------------------------------
10249       NULL
```

В данном листинге показано, как загрузить в массив столбец `OrderDate` таблицы `Orders` базы данных `Northwind` с помощью оператора `SELECT` и функции `fn_setarray`. Обратите внимание на то, что можно загрузить и всю таблицу с помощью одного оператора `SELECT`. После этого выполняется запрос к массиву как к таблице с использованием функции `fn_listarray`, предназначенной для работы со значениями, представленными в виде таблицы, а критерий выборки запроса определяется с помощью индекса массива.

Безусловно, читатель уже оценил, какая широкая область применения открывается в языке Transact-SQL для функциональных средств, основанных на поддержке массивов. Дело в том, что даже при разработке программы на языке, основанном на поддержке наборов данных, иногда возникают ситуации, в которых более подходящими инструментальными средствами служат массивы. В подобных ситуациях могут использоваться средства поддержки массивов, представленные в настоящей главе.

Резюме

COM — это мощная и всеобъемлющая технология, которая позволяет обеспечить функциональную совместимость приложений с помощью самых различных способов. Благодаря наличию средств ODSOLE язык Transact-SQL позволяет обращаться к тем интерфейсам COM-объектов, доступ к которым предоставляют другие приложения и даже сама программа SQL Server. Применение возможностей реляционной базы данных в сочетании с гибкими и универсальными средствами Automation позволяет создавать действительно очень мощные приложения.

В средствах ODSOLE программы SQL Server для взаимодействия с COM-объектами применяется позднее связывание. Это означает, что в средствах ODSOLE используются вызовы COM-интерфейса `IDispatch` по аналогии с такими традиционными языками поддержки сценариев, как VBScript и JScript.

Кроме того, в средствах ODSOLE применяется модель многопотоковой поддержки STA. Сразу после инициализации средств ODSOLE создается главный

контейнер STA, если он еще не был создан. В модели STA доступ из других апартаментных контейнеров координируется с помощью сообщений Windows.

Экземпляры COM-объектов создаются из языка Transact-SQL с помощью процедуры `sp_OACreate`, а для уничтожения этих экземпляров применяется процедура `sp_OADestroy`. Любые объекты, не уничтоженные после завершения пакетного задания, автоматически освобождаются с помощью одного из двух обработчиков ситуаций завершения пакетного задания, которые устанавливаются средствами ODSOLE сразу после первого вызова любой процедуры sp_OA в рабочем потоке.

Вопросы для самопроверки

1. Какой термин преимущественно применяется в наше время для обозначения технологии OLE Automation?
2. Подтвердите или опровергните следующее утверждение. Процедуры семейства sp_OA расширенных процедур фактически представляют собой так называемые "специальные процедуры" – точки входа этих процедур не находятся в какой-либо внешней библиотеке DLL.
3. Какой известный COM-интерфейс используется процедурами sp_OA для вызова метода COM-объекта?
4. Какие действия предпринимаются средствами ODSOLE при возврате из вызова процедуры sp_OA значения типа массива?
5. Подтвердите или опровергните следующее утверждение. Нет необходимости явно уничтожать экземпляры COM-объектов, созданные в вызовах процедур sp_OA, поскольку средства ODSOLE уничтожают эти экземпляры автоматически после завершения текущего пакетного задания.
6. Опишите в общих чертах назначение главного контейнера STA процесса.
7. Подтвердите или опровергните следующее утверждение. В SQL Server 2000 и предыдущих версиях использование процедур sp_OA для создания экземпляров классов управляемого кода, опубликованных как COM-объекты, не поддерживалось компанией Microsoft.
8. Объясните, в чем состоит основное различие между функциями `CoInitialize` и `CoInitializeEx`.
9. Назовите API-интерфейс Windows, который обеспечивает доставку сообщений.
10. Какие действия должен выполнить разработчик программного кода на языке T-SQL, чтобы вынудить какой-то конкретный COM-компонент предпринять попытки осуществить внепроцессный запуск?
11. Подтвердите или опровергните следующее утверждение. Указатель, возвращаемый процедурой `sp_OACreate`, представляет собой адрес вновь созданного COM-объекта.

12. На какой объект ссылается следующая уточняющая запись через точку: `Databases.Items("pubs")`?
13. Каково максимальное количество контейнеров MTA, которые могут поддерживаться в одном процессе?
14. Какое соединение необходимо установить в выполняемой хранимой процедуре, если основное количество вызовов SQL-DMO эта процедура должна направлять на тот сервер, на которым она выполняется?
15. Допустим, что какой-то управляемый класс откомпилирован в виде сборки DLL и зарегистрирован для использования в средствах COM с помощью инструментального средства `regasm.exe`. Какая библиотека DLL фактически становится базовой для рассматриваемого класса при его использовании в рамках технологии COM?
16. Подтвердите или опровергните следующее утверждение. Средства SQL-DMO определяют зависимости между объектами путем проверки таблицы `sysdepends`, которая не всегда является надежным и точным источником информации о зависимостях.
17. Какая функция инициализации COM вызывается средствами ODSOLE?
18. Как по умолчанию задается модель многопотоковой поддержки для внутрипроцессного COM-сервера?
19. Подтвердите или опровергните следующее утверждение. В отличие от Visual Basic, применяемые в программе SQL Server средства Automation, действующие в рамках технологии COM, не поддерживают именованные параметры.
20. Возможно ли изменить модель многопотоковой поддержки COM существующего потока, не отменив предварительно инициализацию COM в этом потоке?
21. Опишите, какие действия выполняются средствами ODSOLE, если выполнение вызова процедуры `sp_OAMethod` приводит к получению выходного значения, но не определен какой-либо выходной параметр.
22. Как обрабатывался бы средствами ODSOLE массив структур, возвращенный после вызова процедуры `sp_OAMethod`?
23. Какой широко известный метод COM-интерфейса вызывается средствами ODSOLE для вызова методов COM-объекта, экземпляр которого был создан по принципу позднего связывания?
24. Подтвердите или опровергните следующее утверждение. Имена процедур sp_OA начинаются с префикса `sp_`, поэтому синтаксический анализатор Transact-SQL не позволяет вызывать эти процедуры из определяемых пользователем функций, принимая их за обычные хранимые процедуры.
25. Подтвердите или опровергните следующее утверждение. Если COM-объект создан по принципу регистрации и предоставления доступа к общедоступному управляемому классу и интерфейсу, то COM-клиент сможет использовать этот COM-объект только после того, как последний будет установлен в глобальном кэше сборок (Global Assembly Cache – GAC).

ГЛАВА 16

Полнотекстовый поиск

В данной главе рассматриваются средства полнотекстового поиска (Full-Text Search – FTS) программы SQL Server, а также машина, лежащая в основе этих средств, – служба поиска Microsoft Search. В этой главе будет описано, как работают средства полнотекстового поиска, и показано, как используются полнотекстовые запросы для выборки данных с применением сложных критериев поиска.

Поддержка полнотекстового поиска была впервые введена в версии SQL Server 7.0 и с тех пор значительно не изменялась. Применительно к данным в таблицах SQL Server средства полнотекстового поиска предоставляют во многом такие же функциональные возможности, которые обычно предоставляются автономными программными продуктами индексации, такими как служба индексации Microsoft Indexing Service (поисковая машина, основанная на использовании файлов операционной системы). Средства полнотекстового поиска обеспечивают поиск в таблицах SQL Server с помощью более сложных поисковых конструкций по сравнению с теми, которые могут применяться в стандартном языке Transact-SQL.

В настоящей главе приведено обновленное описание средств полнотекстового поиска, которое было впервые изложено в книге автора *The Guru's Guide to Transact-SQL*. Как и других главах этой книги, в которых приведено обновленное изложение того, что было написано автором по конкретной теме в изданных ранее книгах, автор пытался в равной степени описывать архитектурные особенности рассматриваемой технологии и предоставлять современные рекомендации по практическому использованию этой технологии. Понимание того, как работают конкретные программные средства, служит ключом к реализации всех возможностей рассматриваемой технологии. Несмотря на сказанное выше, в данной книге автор пытался выйти за рамки концептуального описания и показать, как заставить работать на практике приобретенные знания о проекте конкретной технологии. Автор не знает лучшего способа закрепления вновь освоенного материала, чем его практическое использование.

Краткий обзор

В мире баз данных SQL возможности поиска в символьных и текстовых полях применяются уже давно. Средства поиска подстрок в символьных строках и полях входят в состав программного обеспечения СУБД в течение многих лет. Однако обычно эти средства нельзя назвать иначе, как рудиментарными. До появления полнотекстового поиска встроенные инструментальные средства текстового поиска программы SQL Server можно было в основном рассматривать как соот-

ветствующие приведенной выше характеристике — эти средства немного выходили за рамки требований стандартов ANSI, но не отличались ничем выдающимся. С помощью этих средств можно было выполнять проверки на равенство с использованием символьных строк (наравне со всеми прочими типами данных), а также выполнять поиск в строках по шаблону (с использованием конструкций LIKE и PATINDEX), но не предоставлялась возможность выполнять какие-либо более сложные операции, такие как поиск с учетом тесного соседства слов или морфологических характеристик слов.

После введения встроенной поддержки полнотекстовой индексации указанная ситуация изменилась. До сих пор разработчикам программного обеспечения баз данных приходилось для реализации усовершенствованных средств текстового поиска использовать шлюзы баз данных, файлы операционной системы и технологии, внешние по отношению к программе SQL Server. Это положение коренным образом изменилось. Служба поиска Microsoft Search предоставляет в среде SQL Server функциональные возможности такой полноценной машины текстового поиска, как служба индексации Microsoft Indexing Service. Служба Microsoft Search используется для формирования метаданных, необходимых для поддержки полнотекстового поиска и обработки запросов полнотекстового поиска. Сама эта служба предназначена только для версий сервера, эксплуатируемых в семействе операционных систем Windows NT (Windows NT Server и Windows NT Advanced Server, Windows 2000 Server и Windows 2000 Advanced Server, Windows Server 2003 и т.д.), а доступ к этой службе может предоставляться клиентами SQL Server, работающими в версиях Windows 9x, Windows ME, Windows 2000 Professional и Windows XP.

Подробные сведения об организации полнотекстовых средств поиска SQL Server

Данные, сопровождаемые службой Microsoft Search (полнотекстовые индексы и информация каталогов, используемая для обслуживания запросов), не хранятся в обычных системных таблицах, поэтому доступ к этим данным не может быть получен непосредственно из программы SQL Server. Указанные данные хранятся в файлах операционной системы и являются доступными только для самой службы и для администраторов NT. По умолчанию файлы с указанными данными находятся в подкаталоге FTDATA корневого инсталляционного каталога SQL Server. Резервное копирование файлов службы поиска не осуществляется в ходе обычных сеансов резервного копирования баз данных, поэтому необходимо сохранять эти файлы в отдельном сеансе (и синхронизировать полученные резервные копии с резервными копиями соответствующих метаданных, хранящихся в базе данных SQL Server), чтобы защитить файлы от катастрофической потери.

Службу Microsoft Search можно рассматривать как сервер текста по аналогии с тем, как программа SQL Server рассматривается как сервер SQL или базы данных, поскольку эта служба поиска получает запросы и команды, относящиеся к полнотекстовому поиску, и возвращает требуемые результаты. Единственным

клиентом службы Microsoft Search является программа SQL Server и доступ к этой службе осуществляется именно через указанную программу.

Взаимодействие между программой SQL Server и службой Microsoft Search происходит через полнотекстовое средство доступа. Код этого средства доступа находится в библиотеке SQLFTQRY.DLL, которая хранится в подкаталоге Binn применяемого по умолчанию инсталляционного каталога SQL Server. Библиотека SQLFTQRY предоставляет программе SQL Server доступ и к службе администрирования, и к службе полнотекстовых запросов. Взаимодействие сервера с этими службами осуществляется с помощью системных процедур sp_fulltext_..., в которых используется недокументированная команда DBCC CALLFULLTEXT для выполнения административных задач, относящихся к доступу к полнотекстовым индексам и к службе Microsoft Search. Интерфейс вызова команды DBCC CALLFULLTEXT выглядит следующим образом:

```
DBCC CALLFULLTEXT(funcid[,catid][,objid][,sub])
```

Команда DBCC CALLFULLTEXT требует один обязательный параметр и поддерживает три дополнительных необязательных параметра. Параметр funcid указывает, какая функция должна быть выполнена и какие параметры являются действительными. Параметр catid обозначает идентификатор полнотекстового каталога, objid содержит объектный идентификатор затрагиваемого объекта, а sub обозначает идентификатор подфункции, если он применяется. Следует отметить, что команду CALLFULLTEXT допускается использовать только в системной хранимой процедуре. Для такой процедуры должен быть установлен бит системной принадлежности (с использованием недокументированной процедуры sp_MS_marksystemobject), а имя процедуры должно начинаться с префикса sp_fulltext_. Поддерживаемые функции перечислены в табл. 16.1.

Таблица 16.1. Функции DBCC CALLFULLTEXT

Параметр funcid	Параметр sub	Назначение	Остальные параметры
1		Создать каталог	Идентификатор каталога; путь
2		Удалить каталог	Идентификатор каталога
3		Заполнить каталог	Идентификатор каталога; режим заполнения (0 – полное заполнение, 1 – инкрементное заполнение)
4		Прекратить заполнение каталога	Идентификатор каталога
5		Добавить таблицу для полнотекстовой индексации	Идентификатор каталога; идентификатор объекта
6		Удалить таблицу из состава таблиц, предназначенных для полнотекстовой индексации	Идентификатор каталога; идентификатор объекта
7		Удалить все каталоги	Идентификатор базы данных
8		Выполнить очистку каталога	
9		Запретить автоматическое распространение полнотекстовой индексации	Идентификатор объекта

Окончание табл. 16.1

Параметр funcid	Параметр sub	Назначение	Остальные параметры
10		Разрешить автоматическое распространение полнотекстовой индексации	Идентификатор каталога; идентификатор объекта
11		Начать автоматическое распространение полнотекстовой индексации	Идентификатор каталога; идентификатор объекта
12	0	Начать полное заполнение таблицы	Идентификатор каталога, идентификатор объекта
12	1	Начать инкрементное заполнение таблицы	
12	2	Прекратить полное/инкрементное заполнение таблицы	Идентификатор каталога, идентификатор объекта
13		Определить уровень ресурсов процессора, выделенных для средств Microsoft Search	Уровень ресурса (1–5; 1 — фоновый, 5 — выделенный; значение по умолчанию — 3)
14		Установить тайм-аут соединения для полнотекстовых соединений с программой SQL Server	Значение тайм-аута соединения в секундах (1–32767)
15		Установить тайм-аут обработки данных (запроса) для полнотекстовых запросов к программе SQL Server	Значение тайм-аута данных (запроса) в секундах (1–32767)
16		Восстановить каталог (после первоначального удаления)	Идентификатор каталога; путь

В листинге 16.1 приведен пример процедуры, в которой непосредственно вызывается команда DBCC CALLFULLTEXT.

Листинг 16.1. Вызов команды **DBCC CALLFULLTEXT**

```
USE master
GO
IF OBJECT_ID('sp_fulltext_resource') IS NOT NULL
  DROP PROC sp_fulltext_resource
GO
CREATE PROC sp_fulltext_resource @value int -- Значение параметра
                                            -- resource_usage
AS
  DBCC CALLFULLTEXT(13,@value) -- FTSetResource (@value)
  IF (@@error<>0) RETURN 1
  -- Успешное завершение --
RETURN 0 -- sp_fulltext_resource
GO

EXEC sp_MS_marksystemobject 'sp_fulltext_resource'
EXEC sp_fulltext_resource 3
```

Как правило, пользователь не должен вызывать команду DBCC CALLFULLTEXT в своем собственном коде. Дело в том, что приведенные в табл. 16.1 идентификаторы и параметры функций могут изменяться от одного выпуска SQL Server к другому (что фактически и произошло между выпусками 7.0 и 2000 программы SQL Server), к тому же, удастся избежать возможности серьезного повреждения полнотекстовой инсталляции из-за непреднамеренного вызова какой-либо разрушительной функции. Нет никаких серьезных оснований для отказа от использования программы Enterprise Manager и процедур sp_fulltext_... для управления службой поиска Microsoft Search и пользовательскими полнотекстовыми индексами. Автор привел в табл. 16.1 перечень функций, чтобы можно было проще понять, как работают средства полнотекстового поиска программы SQL Server. Применять команду DBCC CALLFULLTEXT в коде производственного назначения не рекомендуется.

Независимо от количества экземпляров SQL Server, установленных на компьютере, в любой конкретный момент может функционировать только один экземпляр службы Microsoft Search. Этот единственный экземпляр осуществляет управление полнотекстовыми индексами для всех экземпляров SQL Server на компьютере.

Полнотекстовый поиск в данных, отличных от данных SQL Server

Служба Microsoft Search может осуществлять поиск только в данных SQL Server. Сформированные полнотекстовые индексы охватывают исключительно только данные SQL Server, поэтому их нельзя использовать для поиска в файлах операционной системы. Однако для осуществления поиска в файлах операционной системы может применяться служба индексации Microsoft Indexing Service и предусмотренное в ней средство доступа OLE DB. Кроме того, с помощью связанного серверного или распределенного запроса можно обращаться к этому средству доступа из языка T-SQL и даже использовать его в соединениях с обычными запросами полнотекстового поиска, применяемыми к объектам SQL Server. С использованием наборов строк полнотекстового поиска и функциональных предикатов программы SQL Server в сочетании со связанными серверными запросами или ссылками OPENQUERY/OPENROWSET, в которых применяется средство доступа Microsoft OLE DB Provider for Indexing Service (средство доступа OLE DB компании Microsoft для службы индексации), можно комбинировать результаты запросов полнотекстового поиска, полученные из программы SQL Server, с результатами полнотекстового поиска в файлах, полученными вне программы SQL Server.

Средства полнотекстового поиска в двоичных данных

Файлы данных, созданные с помощью таких программных продуктов, как Microsoft Word и Microsoft Excel, нельзя хранить в обычных символьных столбцах SQL Server, поскольку в этих файлах могут содержаться символы, неприменимые в символьных типах данных SQL Server, таких как text, char и nchar. Вместо

этого содержимое указанных файлов необходимо хранить в столбцах типа image, если нужно иметь возможность сохранить любое байтовое значение, которое могло бы присутствовать в файле, а также иметь возможность хранить файлы с объемом больше 8 Кбайт (для файлов с объемом меньше 8 Кбайт могут использоваться столбцы типа binary и varbinary).

Программа SQL Server обладает способностью создавать полнотекстовые индексы на столбцах типа image и автоматически распознавать некоторые типы внешних данных. Такая задача осуществляется с помощью так называемых "фильтров", рассчитанных на обработку файлов с конкретными расширениями имен. Фильтр просто распознает конкретный тип двоичных данных с использованием расширения имени файла. Обычно расширение имени файла является частью имени внешнего файла. Но в таблице SQL Server расширение имени файла связывается со значением конкретного столбца типа image путем применения одного из столбцов в той же таблице для хранения расширения имени файла для двоичных данных каждой строки. Затем имя столбца, в котором хранятся расширения имен файлов, обозначается в качестве имени столбца Document type при подготовке таблицы для полнотекстовой индексации в программе-мастере Full-Text Indexing Wizard (такой столбец можно также указать с помощью параметра @type_colname процедуры sp_fulltext_column, если для создания полнотекстового индекса применяются хранимые процедуры).

В настоящее время программа SQL Server поддерживает фильтры пяти типов — .DOC (Microsoft Word), .XLS (Microsoft Excel), .PPT (Microsoft PowerPoint), .TXT (текстовые файлы) и .HTM (файлы языка разметки гипертекста). Таблицы, в которых данные двоичных файлов хранятся с использованием указанного способа, позволяют хранить в разных строках данные различных форматов. Например, в одной строке может храниться документ Word, в следующей — документ Excel, а за ней может находиться строка с документом HTML. Тип данных, содержащихся в каждой рассматриваемой строке, хранится в столбце с расширением имени файла, который используется службой Microsoft Search в качестве указания, что должен быть применен соответствующий фильтр.

После того как был должным образом создан индекс с помощью соответствующего фильтра для данных типа image, можно выполнять полнотекстовые запросы к этим данным по такому же принципу, как и к данным, хранящимся в столбцах любых других типов. Полнотекстовые индексы, созданные на столбцах типа image, ничем не отличаются от индексов, созданных на столбцах обычных символьных типов.

СОВЕТ. Применение пакета DTS и преобразования Read File — превосходный способ загрузки данных в базу данных SQL Server из таких внешних источников, как документы Word и электронные таблицы Excel, чтобы в дальнейшем могла быть выполнена полнотекстовая индексация этих данных. Средства преобразования Read File позволяют загружать содержимое файлов операционной системы в столбец таблицы. Для этого создается таблица с перечнем имен загружаемых файлов, после этого выполняется настройка конфигурации средств преобразования, которые считывают файлы, перечисленные в таблице, и загружают их содержимое в столбец таблицы назначения. Подготовка к выполнению этих операций очень проста и позволяет легко загружать большое количество файлов в таблицу базы данных. Дополнительная информация о преобразованиях Read File приведена в главе 20, "Службы Data Transformation Services".

Подготовка полнотекстовых индексов

Подготовка полнотекстовых индексов осуществляется в несколько этапов. И в этом случае, как обычно, расширение возможностей приводит к повышению сложности. При условии, что осуществлен запуск службы Microsoft Search, порядок действий, требуемых для подготовки конкретного столбца таблицы, для того чтобы в нем можно было выполнять полнотекстовый поиск с помощью таких конструкций, как предикат CONTAINS и функция набора строк FREETEXTTABLE, включает шесть перечисленных ниже этапов.

1. Разрешить в применяемой базе данных полнотекстовую индексацию.
2. Создать для этой базы данных полнотекстовые каталоги.
3. Разрешить для применяемой таблицы полнотекстовую индексацию и связать эту таблицу с полнотекстовым каталогом.
4. Добавить необходимый столбец к полнотекстовому индексу таблицы.
5. Активизировать указанный полнотекстовый индекс.
6. Заполнить полнотекстовый каталог. В качестве способа заполнения может применяться полное или инкрементное заполнение. Безусловно, для начального заполнения полнотекстового индекса всегда должно осуществляться полное заполнение. Последующие операции заполнения могут быть инкрементными, если таблица содержит столбец временной отметки и если ее метаданные не изменились со времени последнего заполнения. Можно также разрешить отслеживание изменений, чтобы регистрировались изменения в индексированных столбцах, и полнотекстовый индекс обновлялся по мере возникновения этих изменений, или по требованию, или с учетом некоторого расписания. Чтобы разрешать отслеживание изменений для конкретной таблицы, щелкните правой кнопкой мыши на имени этой таблицы в программе Enterprise Manager и выберите команду Change Tracking из меню Full-Text Index Table. (Чтобы разрешить отслеживание изменений, можно также передать значение `start_change_tracking` в качестве параметра `@action` в процедуру `sp_fulltext_table`.)

Как и при выполнении большинства административных задач SQL Server, наилучшим средством создания полнотекстовых индексов является программа Enterprise Manager. Дело в том, что требуемые для этого действия слишком утомительны для того, чтобы их можно было выполнять вручную слишком часто. Несмотря на сказанное выше, в учебных целях следует выполнить самому подготовку полнотекстовых индексов с помощью языка Transact-SQL, поскольку это позволяет лучше понять, что происходит незаметно для постороннего взгляда, и как работают все эти средства, чем при прохождении по экранам программы-мастера Enterprise Manager с помощью щелчков мышью. В листинге 16.2 приведены некоторые примеры кода, которые показывают, как подготовить столбец для полнотекстового поиска с использованием исключительно средств языка SQL. (Автор пронумеровал шаги в этом коде в соответствии с этапами в приведенном выше списке.)

Листинг 16.2. Подготовка столбца для полнотекстового поиска

```
USE pubs
DECLARE @tablename sysname, @catalogname sysname,
    @indexname sysname, @columnname sysname

SET @tablename='pub_info'
SET @catalogname='pubsCatalog'
SET @indexname='UPKCL_pubinfo'
SET @columnname='pr_info'

-- Шаг 1. Разрешить использование полнотекстового поиска
--        для базы данных
EXEC sp_fulltext_database  'enable'

-- Шаг 2. Создать полнотекстовый каталог
EXEC sp_fulltext_catalog @catalogname, 'create'
-- Шаг 3. Создать полнотекстовый индекс для таблицы
EXEC sp_fulltext_table @tablename,'create',@catalogname,@indexname

-- Шаг 4. Добавить к полнотекстовому индексу столбцы
EXEC sp_fulltext_column @tablename, @columnname, 'add'

-- Шаг 5. Активизировать вновь созданный полнотекстовый индекс
EXEC sp_fulltext_table @tablename,'activate'

-- Шаг 6. Заполнить вновь созданный полнотекстовый каталог
EXEC sp_fulltext_catalog @catalogname, 'start_full'
```

В этом коде осуществляется подготовка индексов полнотекстового поиска для столбца `pr_info` таблицы `pubs..pub_info`. Это – текстовый столбец, поэтому он хорошо подходит для полнотекстовой индексации. В целях упрощения при разработке приведенной в листинге 16.2 процедуры был принят ряд допущений, которые в реальном мире могут оказаться неоправданными. Например, в коде не предусмотрена проверка, существует ли полнотекстовый каталог, перед попыткой создать этот каталог. Если же каталог уже существует, то сам оператор его создания и пакетное задание окончатся неудачей. То же касается и полнотекстового индекса на таблице `pub_info`. Каждая таблица может иметь один и только один полнотекстовый индекс. Попытка создать второй полнотекстовый индекс или воссоздать существующий приводит к ошибке. Процедура, приведенная в листинге 16.2, служит просто для демонстрации основ подготовки полнотекстового индекса с использованием языка Transact-SQL.

Основную часть информации, необходимой для проверки условий выполнения перед вызовом полнотекстовых хранимых процедур, можно получить с помощью функций доступа к метаданным. Например, для определения того, существует ли данный конкретный каталог, применяется функция `FULLTEXTCATALOGPROPERTY`. (Эта функция возвращает `NULL`, если ей передается несуществующее имя каталога.) Кроме того, с помощью функции `OBJECTPROPERTY` можно определить, имеется ли на таблице полнотекстовый индекс, а функция `COLUMNPROPERTY` позволяет узнать, был ли добавлен к полнотекстовому индексу тот или иной столбец. В листинге 16.3 представлена хранимая процедура, в которой используются

эти некоторые другие функции, позволяющие подготовить столбец для полнотекстовой индексации намного более надежным способом. Эта процедура является гораздо более отказоустойчивой по сравнению с процедурой, приведенной в листинге 16.2, поэтому ее можно использовать в производственной сфере значительно более безопасно.

Листинг 16.3. Хранимая процедура для подготовки столбца для полнотекстовой индексации

```
USE master
GO
IF OBJECT_ID('sp_enable_fulltext') IS NOT NULL
  DROP PROC sp_enable_fulltext
GO
CREATE PROC sp_enable_fulltext @tablename sysname,
     @columnname sysname=NULL, @catalogname sysname=NULL,
    @startserver varchar(3)='NO'
/*

Object: sp_enable_fulltext
Description: Enables full-text indexing for a specified column
  Usage: sp_enable_fulltext @tablename=name of host table,
    @columnname=column to set up, [,@catalogname=name of full-text
    catalog to use (Default: DB_NAME()+"Catalog")]
     [,@startsrever=YES|NO specifies whether to start the
    Microsoft Search service on this machine prior to setting up
    the column (Default: YES)]
  Returns: (None)
  Created by: Ken Henderson. Email: khen@khen.com
  Example: EXEC sp_enable_fulltext "pubs..pub_info","pr_info",
    DEFAULT,"YES"
  Created: 1999-06-14.  Last changed: 1999-07-14.

*/
AS
SET NOCOUNT ON

IF (@tablename='/?') OR (@columnname IS NULL) OR
    (OBJECT_ID(@tablename) IS NULL) GOTO Help

IF (FULLTEXTSERVICEPROPERTY('IsFulltextInstalled')=0)
    BEGIN -- Машина поиска не инсталлирована
  RAISERROR('The Microsoft Search service is not installed on
      server %s',16,10,@@SERVERNAME)
  RETURN -1
END

DECLARE @catalogstatus int, @indexname sysname

IF (UPPER(@startserver)='YES')
  EXEC master..xp_cmdshell 'NET START mssearch', no_output

IF (@catalogname IS NULL)
  SET @catalogname=DB_NAME()+'Catalog'
```

```
CREATE TABLE #indexes ( -- Таблица служит для хранения уникального
                        -- индекса, применяемого при полнотекстовом
                        -- поиске
Qualifier             sysname NULL,
Owner                 sysname NULL,
TableName             sysname NULL,
NonUnique             smallint NULL,
IndexQualifier      sysname NULL,
IndexName             sysname NULL,
Type                  smallint NULL,
PositionInIndex       smallint NULL,
ColumnName            sysname NULL,
Collation             char(1) NULL,
Cardinality          int NULL,
Pages                 int NULL,
FilterCondition       sysname NULL)

INSERT #indexes
EXEC sp_statistics @tablename

SELECT @indexname=IndexName FROM #indexes WHERE NonUnique=0
    -- Получить уникальный индекс, заданный на таблице (если задано
    -- несколько уникальных индексов, получить последний)

DROP TABLE #indexes

IF (@indexname IS NULL) BEGIN -- Если уникальные индексы отсутствуют,
                              -- выполнить аварийное завершение
  RAISERROR('No suitable unique index found on table %s',16,
      10,@tablename)
  RETURN -1
END

IF (DATABASEPROPERTY(DB_NAME(),'IsFulltextEnabled')<>1)
     -- Разрешить использование полнотекстового поиска для базы данных
  EXEC sp_fulltext_database  'enable'

SET @catalogstatus=FULLTEXTCATALOGPROPERTY(@catalogname,
    'PopulateStatus')

IF (@catalogstatus IS NULL) -- Переменная еще не существует
  EXEC sp_fulltext_catalog @catalogname, 'create'
ELSE IF (@catalogstatus IN (0,1,3,4,6,7)) -- Состояния каталога:
              -- Population in progress (Происходит заполнение),
              -- Throttled (Возникло узкое место), Recovering
              -- (Происходит восстановление), Incremental Population
              -- in Progress (Происходит инкрементное заполнение) или
              -- Updating Index (Происходит обновление индекса)
  EXEC sp_fulltext_catalog @catalogname, 'stop'

IF (OBJECTPROPERTY(OBJECT_ID(@tablename),
     'TableHasActiveFullTextIndex')=0) -- Создать полнотекстовый
                                       -- индекс, если он отсутствует
EXEC sp_fulltext_table @tablename,'create',@catalogname,
      @indexname
ELSE
```

```
  EXEC sp_fulltext_table @tablename,'deactivate'  -- Отменить
                         -- активизацию индекса, чтобы можно было
                         -- вносить в него изменения

IF (COLUMNPROPERTY(OBJECT_ID(@tablename),@columnname,
     'IsFulltextIndexed')=0) BEGIN -- Добавить столбец к индексу
  EXEC sp_fulltext_column @tablename, @columnname, 'add'
  PRINT 'Successfully added a full-text index for '+@tablename+
       '.'+@columnname+' in database '+DB_NAME()
END ELSE
  PRINT 'Column '+@columnname+' in table '+DB_NAME()+'.'+
      @tablename+' is already full-text indexed'

EXEC sp_fulltext_table @tablename,'activate'

EXEC sp_fulltext_catalog @catalogname, 'start_full'
RETURN 0

Help:
EXEC sp_usage @objectname='sp_enable_fulltext',@desc='Enables
     full-text indexing for a specified column',
@parameters='@tablename=name of host table, @columnname=column
    to set up,
[,@catalogname=name of full-text catalog to use (Default:
     DB_NAME()+"Catalog")][,@startsrever=YES|NO specifies whether
    to start the Microsoft Search service on this machine prior to
    setting up the column (Default: YES)]',
@author='Ken Henderson', @email='khen@khen.com',
@datecreated='19990614',@datelastchanged='19990714',
@example='EXEC sp_enable_fulltext "pubs..pub_info","pr_info",
    DEFAULT,"YES"'
RETURN -1

sp_enable_fulltext 'pub_info','pr_info'

Successfully added a full-text index for pub_info.pr_info
in database pubs
```

В этой процедуре выполняется целый ряд интересных действий. Процедура начинается с проверки, позволяющей определить, установлена ли служба Microsoft Search. Если данная служба не установлена, то в процедуре немедленно осуществляется аварийное завершение. Затем в этой процедуре используется расширенная процедура `xp_cmdshell` для запуска службы Microsoft Search, если это требуется для дальнейшей работы. (Соответствующая команда не оказывает никакого воздействия, если служба поиска уже функционирует.) Такое действие выполняется с помощью команды `NET START mssearch` операционной системы. Команда `NET START` применяется в операционной системе Windows NT для запуска служб, а `mssearch` — это внутреннее имя службы Microsoft Search. (Запуск службы поиска может быть также осуществлен с помощью программы Enterprise Manager, аплета Services программы Windows NT Control Panel, а также с помощью программы SQL Server Service Manager.)

Затем в процедуре осуществляется выборка индекса уникального ключа таблицы для указанной таблицы. Для добавления полнотекстового индекса к таблице требуется индекс уникального ключа. В данном случае в рассматриваемой процедуре перехватывается вывод процедуры sp_statistics (которая формирует перечень индексов таблицы) во временную таблицу с помощью команды INSERT...EXEC, затем осуществляется просмотр временной таблицы для поиска уникального индекса рассматриваемой таблицы. Если на этом этапе не будет обнаружен уникальный индекс, немедленно происходит аварийное завершение процедуры.

После этого в процедуре осуществляется проверка для определения того, разрешено ли использовать в текущей базе данных полнотекстовую индексацию. Если эта опция не разрешена, то данная процедура ее разрешает. Затем в коде происходит проверка статуса полнотекстового каталога. Если такой каталог не существует, то он создается в процедуре. А если каталог активен, то процедура его закрывает, чтобы можно было вносить в каталог изменения.

После того как найден или создан полнотекстовый каталог, процедура создает полнотекстовый индекс для таблицы после проверки с помощью функции OBJECTPROPERTY того, что полнотекстовый индекс еще не существует. Именно на этом этапе используется уникальный индекс, обнаруженный ранее в рассматриваемой процедуре.

После подготовки полнотекстового индекса процедура вводит в индекс указанный столбец. В процедуре берется переданное ей имя столбца, и указанный столбец добавляется к полнотекстовому индексу таблицы с помощью процедуры sp_fulltext_column. Вызов на выполнение этой процедуры служит для сервера указанием на то, что требуется сформировать индекс в целях отслеживания расширенной информации поиска для указанного столбца, но активизировать этот индекс и заполнять его данными фактически не требуется. Такие операции выполняются на следующем этапе.

Последовательность действий завершается путем вызова процедур sp_fulltext_table и sp_fulltext_catalog для активизации нового полнотекстового индекса и заполнения его данными. После завершения этих операций можно приступить к использованию полнотекстовых предикатов и функций набора строк, которые ссылаются на вновь индексированные столбцы.

Полнотекстовые предикаты

Предикат — это логическая конструкция, которая возвращает TRUE или NOT TRUE. (Автор в данном случае избегает использования значения FALSE из-за проблем, которые возникают при работе с трехзначной логикой.) В языке SQL такие предикаты обычно принимают форму функций и находятся в конструкции WHERE. К примерам предикатов конструкции WHERE относятся функции LIKE и EXISTS.

Если разрешен полнотекстовый поиск, то в языке Transact-SQL появляется возможность использовать два дополнительных предиката — CONTAINS и FREETEXT. Предикат CONTAINS предоставляет поддержку операций точного и неточного согласования со строками операций поиска с учетом тесного соседства слов,

морфологических признаков слов, а также весовых коэффициентов. В отличие от этого, предикат FREETEXT используется для поиска слов или словосочетаний, имеющих такой же основной смысл, как искомый термин.

Прежде чем приступить к изучению примеров кода, в котором применяются указанные функции, рассмотрим, как разрешить полнотекстовый поиск для таблицы Employees базы данных Northwind. Таблица Employees включает текстовый столбец Notes, который идеально подходит для полнотекстового поиска. Для подготовки этого столбца можно использовать описанную выше процедуру sp_enable_fulltext примерно следующим образом:

```
EXEC northwind..sp_enable_fulltext 'Employees','notes'
```

В результате будут созданы необходимые метаданные и информация индексации, позволяющие обеспечить правильную работу функций полнотекстового поиска.

Предикат CONTAINS

Предикат CONTAINS применяется для поиска строк, содержащих заданное слово или слова, а также их варианты. С его помощью можно определять местонахождения слов, характеризующихся точным и неточным соответствием критерию поиска, а также выполнять операции поиска с учетом тесного соседства слов и морфологических характеристик. Предикат CONTAINS можно рассматривать как усовершенствованный предикат LIKE. В листинге 16.4 приведен пример, в котором используется предикат CONTAINS для поиска всех сотрудников, перечисленных в таблице Employees, данные о которых содержат в поле Notes слово "English".

Листинг 16.4. Использование предиката CONTAINS и полученные результаты

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE CONTAINS(Notes,'English')

(Результаты приведены в сокращенном виде)

LastName             FirstName  Notes
-------------------- ---------- ----------------------------------
Peacock              Margaret    Margaret holds a BA in English lit
Dodsworth            Anne        Anne has a BA degree in English fr
King                 Robert      …completing his degree in English
```

Заслуживает внимания то, что в данном случае поиск осуществляется во всех столбцах полнотекстового индекса таблицы Employees (причем имеется только один такой столбец), поэтому можно было бы подставить вместо имени столбца символ "*" и получить тот же результат, как показано ниже.

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE CONTAINS(*,'English')
```

Кроме того, предикат CONTAINS поддерживает поиск с учетом тесного соседства слов. В листинге 16.5 приведен уточненный вариант последнего примера,

в котором формируется сокращенный список сотрудников, содержащий данные только о таких сотрудниках, в поле Notes которых находится слово "degree", расположенное близко от слова "English".

Листинг 16.5. Уточнение критерия поиска и полученные результаты

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE CONTAINS(*,'degree NEAR English')

(Результаты приведены в сокращенном виде)

LastName             FirstName  Notes
-------------------- ---------- ---------------------------------
Dodsworth            Anne       Anne has a BA degree in English fr
King                 Robert     …completing his degree in English
```

На этот раз в список попали только две строки, поскольку в поле Notes, относящемся к Margaret Peacock, слово "degree" вообще отсутствует. Следует отметить, что синонимом ключевого слова NEAR является знак тильды (~), поэтому можно заменить первые строки листинга 16.5 следующими строками:

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE CONTAINS(*,'degree ~ English')
```

Строка с условиями поиска поддерживает также логические выражения, как показано в листинге 16.6.

Листинг 16.6. Поиск с применением логических выражений и полученные результаты

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE CONTAINS(Notes,'English OR German')

(Результаты приведены в сокращенном виде)

LastName             FirstName  Notes
-------------------- ---------- ---------------------------------
Peacock              Margaret   Margaret holds a BA in English lit
Dodsworth            Anne       Anne has a BA degree in English fr
Fuller               Andrew     …and reads German
King                 Robert     …completing his degree in English
```

Этот запрос возвращает строки, содержащие слова "English" или "German". Точные или относительные позиции слов не имеют значения — если в столбце Notes присутствует одно из этих слов, то возвращается соответствующая строка. По принципам его использования предикат CONTAINS действует аналогично предикату LIKE, но между ними есть одно важное различие — в предикате CONTAINS учитываются границы слов, а в предикате LIKE — нет. Например, ниже приведен тот же запрос, но оформленный с использованием предиката LIKE.

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE Notes LIKE '%English%'
OR Notes LIKE '%German%'
```

Внешне эти два запроса кажутся одинаковыми, но последний запрос фактически требует предоставления другой искомой информации, чем запрос CONTAINS. Запрос LIKE приводит к поиску согласований с разновидностями искомых слов и даже со словами, которые, по случайному совпадению, содержат эти слова (например, Germantown, Englishman, Germanic и т.д.). В отличие от запроса LIKE, запрос CONTAINS позволяет распознавать отдельные слова, поэтому в нем учитывается различие между словами English и Englishman и обнаруживаются достаточные интеллектуальные возможности, позволяющие выявлять только такие варианты слов, которые интересуют пользователя.

Кроме того, предикат CONTAINS поддерживает подстановочные символы, находящиеся в позиции суффикса. К сожалению, эти подстановочные символы в большей степени напоминают подстановочные символы операционной системы, чем стандартные подстановочные символы SQL. Соответствующий пример приведен в листинге 16.7.

Листинг 16.7. Поиск с использованием подстановочных символов и полученные результаты

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE CONTAINS(*,'"psy*" OR "chem*"')

(Результаты приведены в сокращенном виде)

LastName             FirstName  Notes
-------------------- ---------- ---------------------------------
Leverling            Janet       Janet has a BS degree in chemistry
Davolio              Nancy       Education includes a BA in psychol
Callahan             Laura       Laura received a BA in psychology
```

Этот запрос позволяет найти все строки в полях Notes, в которых содержатся слова, начинающиеся с подстроки "psy" или "chem". Обратите внимание на то, что не поддерживаются отдельно взятые подстановочные символы, а также подстановочные символы, находящиеся в начале искомого терма.

Для отделения одной искомой подстроки от другой в строке условия используются кавычки. Если в строке, определяющей критерий поиска, присутствуют подстановочные символы и многочисленные термы, то кавычки являются обязательными; при их отсутствии попытка выполнить запрос завершается неудачей.

Действительно мощным средством предиката CONTAINS является поддержка в нем поиска с учетом морфологических характеристик. Возможность осуществления поиска на основе форм слов является мощным и часто очень удобным дополнением к составу средств языка Transact-SQL. В листинге 16.8 показано, как осуществляется поиск форм слова.

Листинг 16.8. Поиск форм слова и полученные результаты

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE CONTAINS(*,'FORMSOF(INFLECTIONAL,complete)')

(Результаты приведены в сокращенном виде)

LastName            FirstName  Notes
------------------- ---------- ---------------------------------
Leverling           Janet        ...completed a certificate program
Davolio             Nancy        ...She also completed "The Art of t
King                Robert       ...completing his degree in English
Buchanan            Steven      ...has completed the courses
Callahan            Laura       ...completed a course in business F
```

Для поиска различных имен глагола, а также форм существительного в единственном и множественном числе можно использовать конструкцию FORMSOF. В листинге 16.8 показано, что запрос находит пять строк, которые содержат формы слова "complete", включая "completed" и "completing".

Предикат FREETEXT

Предикат FREETEXT применяется для поиска строк, содержащих слова, которые имеют в основном такой же смысл, как и слова в строке поиска. В отличие от CONTAINS, предикат FREETEXT позволяет задавать ряды термов, которым затем в программе присваиваются веса и осуществляется согласование со значениями в полнотекстовом столбце (столбцах). В листинге 16.9 приведен пример, позволяющий найти служащих с дипломами колледжей, в частности с дипломами бакалавров.

Листинг 16.9. Использование предиката FREETEXT и полученные результаты

```
SELECT LastName, FirstName, Notes
FROM EMPLOYEES
WHERE FREETEXT(Notes,'BA BTS BS BSC degree')

(Результаты приведены в сокращенном виде)

LastName            FirstName  Notes
------------------- ---------- ---------------------------------
Leverling           Janet      Janet has a BS degree in chemistry
Davolio             Nancy      Education includes a BA in psychol
Peacock             Margaret   Margaret holds a BA in English lit
Dodsworth           Anne       Anne has a BA degree in English fr
Fuller              Andrew     Andrew received his BTS commercial
King                Robert     Robert King [completed] his degree
Buchanan            Steven     Steven Buchanan graduated with a B
Callahan            Laura      Laura received a BA in psychology
```

В данном случае запрос возвращает все строки, содержащие любые из указанных термов или аналогичные слова. Как и при использовании предиката CONTAINS, символ "*" применяется для обозначения всех столбцов таблицы с полнотекстовыми индексами.

Функции набора строк

В языке Transact-SQL определен специальный класс функций, называемый функциями набора строк, которые можно использовать вместо таблиц в конструкциях FROM запросов. Функции набора строк возвращают результирующие наборы по принципу, аналогичному производным таблицам, и могут участвовать в соединениях с настоящими таблицами, подвергаться операциям суммирования, группирования и т.д. К полнотекстовому поиску относятся две функции набора строк — CONTAINSTABLE и FREETEXTTABLE. Они представляют собой версии предикатов, описанных выше в данной главе, предназначенные для работы с наборами строк. Эти функции обычно применяются не в конструкции WHERE, а конструкции FROM оператора SELECT. Они возвращают результирующий набор, состоящий из значений ключа индекса и значений рангов строк.

Функция набора строк CONTAINSTABLE

Несмотря на то что функция CONTAINSTABLE представляет собой функцию набора строк, а не предикат, действие этой функции весьма напоминает действие предиката CONTAINS (о чем говорит и само ее имя). Функция CONTAINSTABLE поддерживает такие же критерии формирования поисковой строки, как и предикат CONTAINS, и требует только один параметр, дополнительный к тем, которые необходимы для предиката, — имя основополагающей таблицы. В листинге 16.10 приведен пример, в котором функция CONTAINSTABLE используется для формирования списка ключевых значений и значений рангов строк поиска.

Листинг 16.10. Использование функции CONTAINSTABLE и полученные результаты

```
SELECT *
FROM CONTAINSTABLE(Employees,*,'English OR French OR Italian
    OR German OR Flemish')
ORDER BY RANK DESC

(Результаты)

KEY         RANK
----------- -----------
8            64
2            64
4            48
7            48
9            48
6            32
5            32
```

Функция CONTAINSTABLE возвращает два столбца — значение ключа строки из основополагающей таблицы и значение ранга каждой строки. В этом примере для такого логического упорядочения строк, при котором строки с более высокими значениями ранга показываются в первую очередь, используется столбец RANK.

Значение ключа может применяться для формирования соединения с первоначальной таблицей, что позволяет преобразовать ключ в нечто более осмысленное, как вскоре будет показано.

Значение рангов, возвращаемых в столбце RANK, можно приспособить для своих потребностей, применяя функцию ISABOUT в строке критериев поиска, как показано в листинге 16.11.

Листинг 16.11. Использование функции ISABOUT и полученные результаты

```
SELECT *
FROM CONTAINSTABLE(Employees,*,'ISABOUT(English weight(.8),
    French weight(.1), Italian weight(.2), German weight(.4),
    Flemish weight(0.0))')
ORDER BY RANK DESC

(Результаты)

KEY         RANK
----------- -----------
9            85
2            54
4            47
7            47
8            7
6            3
5            3
```

В этом примере веса назначаются за владение каждым языком, специально обозначенным в поле Notes записи служащего, начиная от 0.0 за владение фламандским (Flemish) и заканчивая 0.8 за владение английским (English). Допустимые значения весов находятся в пределах от 0.0 до 1.0. Как и в предыдущем примере, столбец RANK используется для упорядочения строк для того, чтобы строки с более высокими значениями рангов были перечислены в первую очередь. В предикате CONTAINS также допускается применять функцию ISABOUT, но ее выполнение не дает никакого эффекта, поскольку единственное назначение этой функции состоит в изменении значения столбца RANK, который не используется в предикате.

Для выработки результатов, которые действительно имеют смысл, необходимо соединить результирующий набор, возвращенный конструкцией CONTAINSTABLE, с основополагающей таблицей, используемой в этой конструкции. Дело в том, что сами значения ключей и рангов, возвращенные функцией, не очень-то нужны, если не выполнена их привязка к оригинальным данным. Соответствующий пример приведен в листинге 16.12.

Листинг 16.12. Пример соединения результирующего набора с его основополагающей таблицей и полученные результаты

```
SELECT R.RANK, E.LastName, E.FirstName, E.Notes
FROM Employees AS E JOIN
CONTAINSTABLE(Employees,*,'ISABOUT(English weight(.8),
    French weight(.1), Italian weight(.2), German weight(.4),
    Flemish weight(0.0))') AS R ON (E.EmployeeId=R.[KEY])
```

```
ORDER BY R.RANK DESC

(Результаты приведены в сокращенном виде)

RANK        LastName   FirstName  Notes
----------- ---------- ---------- --------------------------------
85          Dodsworth  Anne        …is fluent in French and German.
54          Fuller     Andrew     …fluent in French and Italian … G
47          Peacock    Margaret    Margaret holds a BA in English 1
47          King       Robert      …before completing his degree in
7           Callahan   Laura      …reads and writes French
3           Suyama     Michael     …can read and write French, Port
3           Buchanan   Steven     …is fluent in French
```

Для того чтобы связать две таблицы, достаточно выполнить простое внутреннее соединение с использованием столбца EmployeeID таблицы Employees и столбца KEY из результирующего набора, возвращенного функцией CONTAINSTABLE. Столбец KEY содержит значения из столбца EmployeeID, соответствующие тем строкам, которые возвращены функцией CONTAINSTABLE, поэтому выполнение операции соединения имеет смысл.

Как и в предыдущих примерах, в данном запросе результирующий набор упорядочивается с использованием столбца RANK, возвращенного функцией CONTAINSTABLE. Обратите внимание на применение квадратных скобок ([]) вокруг ссылки на столбец KEY, возвращенный функцией CONTAINSTABLE. Хотя причину этого трудно объяснить, но в программе SQL Server предусмотрено использование слова KEY в качестве имени столбца, возвращенного функцией CONTAINSTABLE, даже несмотря на то, что KEY — зарезервированное слово. В связи с этим необходимо заключать слово KEY в квадратные скобки (или в двойные кавычки, если разрешен параметр QUOTED_IDENTIFIER) каждый раз, когда применяется непосредственная ссылка на указанное имя.

Чтобы определить, какое влияние оказывает присваивание весов значениям рангов, пересмотрим этот запрос таким образом, чтобы в нем использовались заданные по умолчанию значения рангов, возвращаемые службой Microsoft Search (листинг 16.13).

Листинг 16.13. Пример использования заданных по умолчанию значений рангов и полученные результаты

```
SELECT R.RANK, E.LastName, E.FirstName, E.Notes
FROM Employees AS E JOIN
CONTAINSTABLE(Employees,*,'English OR French OR Italian OR German
    OR Flemish') AS R ON (E.EmployeeId=R.[KEY])
ORDER BY R.RANK DESC

(Результаты приведены в сокращенном виде)

RANK        LastName   FirstName  Notes
----------- ---------- ---------- --------------------------------
64          Fuller     Andrew     …fluent in French and Italian … G
64          Callahan   Laura      …reads and writes French
48          Peacock    Margaret    Margaret holds a BA in English 1
48          King       Robert      …before completing his degree in
```

```
48          Dodsworth   Anne        …is fluent in French and German.
32          Suyama      Michael     …can read and write French, Port
32          Buchanan    Steven      …is fluent in French
```

Вполне очевидно, что применявшиеся перед этим уточненные значения весов привели к получению заметно отличающихся результатов. В частности, заданные пользователем значения весов во многом повлияли на порядок расположения строк.

Функция набора строк FREETEXTTABLE

Функция FREETEXTTABLE, как и соответствующий ей предикат, позволяет находить строки со словами, имеющими в основном такие же значения, как и слова, заданные в критериях поиска. Формат строки с критериями поиска в этой функции является нерегламентированным ("свободным") и не имеет конкретной синтаксической структуры. Машина поиска извлекает каждое слово из строки и назначает ей весовой коэффициент, после чего осуществляет поиск соответствующих строк. В листинге 16.14 показано, как выполнить описанную выше операцию поиска служащих со степенью бакалавра, но с помощью не предиката FREETEXT, а функции FREETEXTTABLE.

Листинг 16.14. Пример использования функции FREETEXTTABLE и полученные результаты

```
SELECT R.RANK, E.LastName, E.FirstName, E.Notes
FROM Employees AS E JOIN
FREETEXTTABLE(Employees,*,'BA BTS BS BCS degree') AS R ON
    (E.EmployeeId=R.[KEY])
ORDER BY R.RANK DESC
RANK        LastName            FirstName  Notes
----------- ------------------- ---------- ----------------------
24          Leverling           Janet       Janet has a BS degree
10          Fuller              Andrew      Andrew received his BT
16          Dodsworth           Anne        Anne has a BA degree i
8           Peacock             Margaret    Margaret holds a BA in
8           Callahan            Laura       Laura received a BA in
8           Davolio             Nancy       Education includes a B
8           King                Robert      Robert King completing
8           Buchanan            Steven      with a BSC degree in 1
```

Теперь критерии поиска в строке стали намного более широкими, поэтому запрос возвращает все строки из таблицы Employees, кроме одной. Во всех возвращенных строках присутствует в той или иной форме одно из слов, перечисленных в строке с критериями поиска.

Резюме

Средства полнотекстового поиска программы SQL Server — это мощное инструментальное средство, которое предоставляет основную часть функциональных возможностей автономных поисковых машин, предназначенных для работы

с файлами. Задача подготовки столбцов для полнотекстового поиска является сложной, поэтому для выполнения такой задачи следует использовать программу Enterprise Manager или хранимую процедуру `sp_enable_fulltext`, приведенную в настоящей главе. После подготовки столбца для осуществления операций полнотекстового поиска становятся доступными предикаты `CONTAINS` и `FREETEXT`, а также функции набора строк `CONTAINSTABLE` и `FREETEXTTABLE`. Средства полнотекстового поиска являются гораздо мощнее, чем такие широко распространенные реализации средств поиска, как предикаты `LIKE` и `PATINDEX`.

Вопросы для самопроверки

1. Какая библиотека DLL используется в программе SQL Server для обеспечения взаимодействия с машиной Microsoft Search?
2. Какая недокументированная команда DBCC используется в процедурах `sp_fulltext...` для обеспечения взаимодействия со службой Microsoft Search?
3. Подтвердите или опровергните следующее утверждение. Средство полнотекстового поиска программы SQL Server может быть установлено в версии Windows 2000 Server, но не в версии Windows 2000 Professional.
4. Назовите две новые предикативные функции, которые можно использовать на таблицах, для которых была выполнена полнотекстовая индексация.
5. Чему равно максимальное количество полнотекстовых индексов, которые могут быть определены на одной таблице?
6. Предусмотрена ли возможность получения доступа в одном и том же запросе T-SQL и к полнотекстовым индексированным данным и к файлам, которые были индексированы с помощью службы Microsoft Indexing Service?
7. Назовите две новые функции набора строк, которые можно использовать для работы с таблицами, на которых был создан полнотекстовый индекс.
8. Какую функцию T-SQL можно вызвать, чтобы определить, было ли установлено на компьютере средство полнотекстового поиска SQL Server?
9. Подтвердите или опровергните следующее утверждение. Полнотекстовые индексы хранятся в системной таблице `sysfulltextindexes`.
10. Возможно ли задать в конфигурации количество времени, в течение которого служба Microsoft Search будет ожидать установления соединения с программой SQL Server, прежде чем завершить выполнение операции подключения по тайм-ауту?

ЧАСТЬ III

Службы данных

ГЛАВА 17

Объединения серверов

Объединение серверов SQL Server — это группа серверов SQL Server, по которым распределено горизонтально секционированное представление. Каждый из серверов в объединении хранит только часть основополагающих данных представления. На каждом сервере имеется полное определение представления, поэтому сервер может использовать свои метаданные для выявления того сервера, на котором действительно хранятся физические данные, затребованные в запросе поиска к секционированному представлению. Таким образом, группа серверов действует как свободное объединение серверов SQL Server. С точки зрения масштабирования такой подход к распределению данных по серверам позволяет организовать "масштабирование" по горизонтали, а не наращивать объемы хранимых данных за счет установки более мощного серверного компьютера (или в дополнение к такой установке).

Объединение серверов SQL Server создается путем формирования распределенного секционированного представления, которое охватывает все эти серверы. В данной главе вначале речь идет о секционированных представлениях (как локальных, так и распределенных), затем обсуждаются способы создания распределенных секционированных представлений. Кроме того, в этой главе описаны проблемы производительности, связанные с использованием секционированных представлений, и подробно рассматриваются некоторые планы выполнения. Этот материал представляет собой обновленное описание секционированных представлений, приведенное в последней книге автора, *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML.*

Секционированные представления

Кратко секционированное представление можно определить как представление, объединяющее в более крупный набор данных таблицы, которые служат в качестве секций (или разделов). Например, секционированное представление с данными о результатах работы некоторого Web-узла может объединять в одно целое отдельные таблицы, относящиеся к каждому месяцу года. В таком случае каждая из этих таблиц хранит данные о результатах деятельности за конкретный месяц. Объединяя эти таблицы, секционированное представление позволяет рассматривать отдельные таблицы как одну таблицу и вместе с тем поддерживать размеры этих таблиц в разумных пределах.

Существуют два типа секционированных представлений — локальные (Local Partitioned View — LPV) и распределенные (Distributed Partitioned View — DPV).

Локальным секционированным представлением называется представление, в котором все основополагающие таблицы находятся в одном и том же экземпляре SQL Server. А распределенное секционированное представление — это представление, в котором отдельные таблицы находятся в разных экземплярах сервера. Эти экземпляры не обязательно должны находиться на разных компьютерах, но обычно так и бывает. Разбиение распределенного секционированного представления по нескольким компьютерам позволяет "масштабировать" крупные реализации программного обеспечения SQL Server. Такое масштабирование позволяет эффективно привлекать мощности и ресурсы многочисленных компьютеров к обработке одного запроса.

Секционированное представление — это обычный объект представления, который объединяет друг с другом таблицы с определенными атрибутами. Характерной особенностью такого представления является использование таблиц с одинаковой структурой, которые в сочетании друг с другом создают унифицированное представление набора данных, но сегментируются по четко определенному "столбцу секционирования". Такой столбец секционирования задается с помощью ограничения CHECK. Столбец секционирования локального или распределенного секционированного представления не только выполняет ту функцию, для которой обычно предназначены ограничения CHECK (а именно, контролирует соответствие типа данных типу столбца, предназначенного для хранения этих данных), но и предоставляет оптимизатору запросов SQL Server возможность определять при создании плана запроса, в какой секции находится данное конкретное значение столбца. Это позволяет исключить из плана запроса операции поиска в других секциях на этапе оптимизации запроса, который включает столбец секционирования.

Происходящие при этом действия проще понять на примере (листинг 17.1).

Листинг 17.1. Простое секционированное представление

```
CREATE TABLE CustomersUS (
  CustomerID nchar (5) NOT NULL,
  CompanyName nvarchar (40) NOT NULL ,
  ContactName nvarchar (30) NULL ,
  ContactTitle nvarchar (30) NULL ,
  Address nvarchar (60) NULL ,
  City nvarchar (15) NULL ,
  Region nvarchar (15) NULL ,
  PostalCode nvarchar (10) NULL ,
  Country nvarchar (15) NOT NULL CHECK (Country='US'),
  Phone nvarchar (24) NULL ,
  Fax nvarchar (24) NULL,
CONSTRAINT PK_CustUS PRIMARY KEY (Country, CustomerID)
)
CREATE TABLE CustomersUK (
  CustomerID nchar (5) NOT NULL,
  CompanyName nvarchar (40) NOT NULL ,
  ContactName nvarchar (30) NULL ,
  ContactTitle nvarchar (30) NULL ,
  Address nvarchar (60) NULL ,
  City nvarchar (15) NULL ,
  Region nvarchar (15) NULL ,
```

```
  PostalCode nvarchar (10) NULL ,
  Country nvarchar (15) NOT NULL CHECK (Country='UK'),
  Phone nvarchar (24) NULL ,
  Fax nvarchar (24) NULL,
CONSTRAINT PK_CustUK PRIMARY KEY (Country, CustomerID)
)

CREATE TABLE CustomersFrance (
  CustomerID nchar (5) NOT NULL,
  CompanyName nvarchar (40) NOT NULL ,
  ContactName nvarchar (30) NULL ,
  ContactTitle nvarchar (30) NULL ,
  Address nvarchar (60) NULL ,
  City nvarchar (15) NULL ,
  Region nvarchar (15) NULL ,
  PostalCode nvarchar (10) NULL ,
  Country nvarchar (15) NOT NULL CHECK (Country='France'),
  Phone nvarchar (24) NULL ,
  Fax nvarchar (24) NULL,
CONSTRAINT PK_CustFR PRIMARY KEY (Country, CustomerID)
)

GO

DROP VIEW CustomersV
GO

CREATE VIEW CustomersV
AS
SELECT * FROM dbo.CustomersUS
UNION ALL
SELECT * FROM dbo.CustomersUK
UNION ALL
SELECT * FROM dbo.CustomersFrance
GO
```

Вполне очевидно, что в данном случае созданы три таблицы для хранения *секций* (или горизонтальных фрагментов) исходной таблицы с данными о заказчиках. Затем эти таблицы снова объединяются с помощью секционированного представления. Какой же в этом смысл? Почему бы не хранить все эти данные в виде одной таблицы? Дело в том, что подход, основанный на использовании секционированных представлений, имеет два преимущества: во-первых, разбиение данных о заказчиках на фрагменты позволяет удерживать размеры таблиц в разумных пределах, поскольку отдельные секции имеют размер, составляющий только часть размера всей таблицы заказчиков, и, во-вторых, оптимизатор запросов SQL Server получает возможность распознавать секционированные представления и автоматически определять нужную основополагающую таблицу для запроса с учетом критериев выборки и ограничения CHECK на столбце секционирования. Например, рассмотрим план выполнения следующего запроса:

```
SELECT * FROM dbo.Customersv WHERE Country='US'

(Результаты приведены в сокращенном виде)
```

```
StmtText
------------------------------------------------------------------------
SELECT CompanyName=CompanyName FROM dbo.CustomersV WHERE Country=@
  |--Compute Scalar(DEFINE:([CustomersUS].CompanyName=CustomersUS.Co
       |--Clustered Index Scan(OBJECT:(Northwind.dbo.CustomersUS.P
```

Даже несмотря на то, что в данном запросе применяется ссылка на все представление, оптимизатор определяет, что требуемые данные могут находиться только в одной из основополагающих таблиц представления, поэтому непосредственно запрашивает эту таблицу и исключает остальные из плана запроса. Оптимизатор для принятия такого решения использует критерии поиска из конструкции WHERE и определение столбца секционирования из каждой таблицы (первичного ключа таблицы).

Но следует прежде всего отметить, что представление может быть определено как секционированное, а оптимизатор получает возможность оптимизировать применяемые к представлению запросы указанным образом лишь при условии соблюдения целого ряда требований, предъявляемых как к представлению, так и к его базовым таблицам. Фактически количество и значимость этих требований таковы, что многие пользователи не решаются воспользоваться секционированными представлениями, особенно локальными. Должны ли действительно использоваться секционированные представления, зависит от конкретных обстоятельств; с информацией о требованиях и ограничениях, связанных с секционированными представлениями, можно ознакомиться в оперативной документации Books Online. Что же касается способности оптимизатора использовать столбцы секционирования для выявления нужной базовой таблицы, в которой должен проводиться поиск данных, то опыт автора показал, что столбец секционирования должен быть крайним слева столбцом в первичном ключе. Это утверждение заслуживает дополнительного изучения. Рассмотрим секционированное представление и запрос, показанные в листинге 17.2.

Листинг 17.2. Несоответствие столбца секционирования и первичного ключа

```
CREATE TABLE Orders1996 (
  OrderID int PRIMARY KEY NOT NULL ,
CustomerID nchar (5) NULL ,
  EmployeeID int NULL ,
  OrderDate datetime NOT NULL CHECK (Year(OrderDate)=1996),
  OrderYear int NOT NULL CHECK (OrderYear=1996),
  RequiredDate datetime NULL ,
  ShippedDate datetime NULL ,
  ShipVia int NULL
)
GO

CREATE TABLE Orders1997 (
  OrderID int PRIMARY KEY NOT NULL ,
  CustomerID nchar (5) NULL ,
  EmployeeID int NULL ,
  OrderDate datetime NOT NULL CHECK (Year(OrderDate)=1997),
  OrderYear int NOT NULL CHECK (OrderYear=1997),
  RequiredDate datetime NULL ,
```

```
  ShippedDate datetime NULL ,
  ShipVia int NULL
)
GO

CREATE TABLE Orders1998 (
  OrderID int PRIMARY KEY NOT NULL ,
  CustomerID nchar (5) NULL ,
  EmployeeID int NULL ,
  OrderDate datetime NOT NULL CHECK (Year(OrderDate)=1998),
  OrderYear int NOT NULL CHECK (OrderYear=1998),
  RequiredDate datetime NULL ,
  ShippedDate datetime NULL ,
  ShipVia int NULL
)
GO

CREATE VIEW OrdersV
AS
SELECT * FROM Orders1996
UNION ALL
SELECT * FROM Orders1997
UNION ALL
SELECT * FROM Orders1998
GO

SELECT * FROM OrdersV WHERE OrderYear=1997
```

Будет ли оптимизатор способен сузить свой поиск только секцией Orders1997? Рассмотрим план выполнения, приведенный в листинге 17.3.

Листинг 17.3. План выполнения запроса, показанного в листинге 17.2

```
(Результаты приведены в сокращенном виде)

------------------------------------------------------------------
Executes    StmtText
----------- ------------------------------------------------------
1           SELECT * FROM [OrdersV] WHERE [OrderYear]=@1
1             |--Concatenation
1                  |--Filter(WHERE:(STARTUP EXPR(Convert([@1])=199
0                  |     |--Clustered Index Scan(OBJECT:([Northwind
1                  |--Filter(WHERE:(STARTUP EXPR(Convert([@1])=199
1                  |     |--Clustered Index Scan(OBJECT:([Northwind
1                  |--Filter(WHERE:(STARTUP EXPR(Convert([@1])=199
0                        |--Clustered Index Scan(OBJECT:([Northwind
```

На первый взгляд может показаться, что план запроса предусматривает поиск во всех трех секциях, даже несмотря на то, что для выполнения запроса требуются данные только из одной секции, но рассмотрим этот план запроса более внимательно. Прежде всего представляет интерес столбец Executes. Он показывает, сколько раз выполнен соответствующий этап запроса. В данном случае указанный столбец сообщает, что поиск фактически проводился только в одной из таблиц. Для двух других таблиц соответствующие значения в столбце Executes

равны 0. Это означает, что оптимизатор обнаружил свою способность обойти ненужные секции на этапе прогона, даже несмотря на то, что эти секции не удалось устранить на этапе компиляции. Устранение ненужных секций на этапе прогона — это очень удобное средство, которое позволяет использовать план с большим количеством входных параметров по сравнению с таким планом, в котором устраняются ненужные секции непосредственно во время компиляции плана. Если в плане секция устраняется на этапе компиляции, то такой план позволяет обслуживать только такие запросы пользователей, которые в конечном итоге относятся к оставшейся в плане секции. Если же устранение ненужных секций происходит на этапе прогона, то план может использоваться для любого потенциального значения параметра, независимо от того, к какой секции в конечном итоге приведет выполнение этого плана.

Несмотря на тот факт, что возможность устранения ненужных секций на этапе прогона, вообще говоря, приводит к лучшему повторному использованию плана, у читателя может возникнуть вопрос, почему же оптимизатор не устраняет ненужные секции на этапе компиляции. Для того чтобы дать возможность оптимизатору устранять ненужные секции на этапе компиляции, необходимо включить все столбцы первичного ключа в критерии выборки запроса. В листинге 17.4 приведен подобный пересмотренный вариант и формируемый в связи с этим план выполнения.

Листинг 17.4. Согласование столбца секционирования и первичного ключа для устранения секций из плана выполнения

```
SELECT * FROM OrdersV WHERE OrderYear=1997 AND OrderID=1000

(Результаты приведены в сокращенном виде)
StmtText
------------------------------------------------------------------
SELECT * FROM OrdersV WHERE OrderYear=@1 AND OrderID=@2
|-Compute Scalar(DEFINE:(Orders1997.OrderID=Orders1997.OrderID, Or
  |-Clustered Index Scan(OBJECT:(Northwind.dbo.Orders1997.PK_Order
```

В данном случае уже имеет место устранение ненужных секций на этапе компиляции. Первичный ключ базовой таблицы включает столбец секционирования, кроме того, все столбцы первичного ключа входят в состав критериев поиска запроса. В этом случае простое выполняемое слева направо согласование столбцов запроса со столбцами первичного ключа не было бы достаточным для устранения ненужных секций на этапе компиляции, которого мы добиваемся, поэтому в запрос пришлось включить все столбцы первичного ключа, так как в противном случае оптимизатор выработал бы неэффективный план. В листинге 17.5 представлены, во-первых, новый вариант секционированного представления, а во-вторых, запрос, который демонстрирует описанный выше принцип.

Листинг 17.5. Введение столбца в определение первичного ключа для предотвращения необходимости исключать ненужные секции на этапе компиляции

```
CREATE TABLE Orders1996 (
  OrderID int NOT NULL ,
  CustomerID nchar (5) NOT NULL ,
  EmployeeID int NULL ,
```

```
  OrderDate datetime NOT NULL CHECK (Year(OrderDate)=1996),
  OrderYear int NOT NULL DEFAULT 1996 CHECK (OrderYear=1996),
  RequiredDate datetime NULL ,
  ShippedDate datetime NULL ,
  ShipVia int NULL,
  CONSTRAINT PK_Orders1996
  PRIMARY KEY (OrderYear, OrderID, CustomerId)
)
GO

CREATE TABLE Orders1997 (
  OrderID int NOT NULL ,
  CustomerID nchar (5) NOT NULL ,
  EmployeeID int NULL ,
  OrderDate datetime NOT NULL CHECK (Year(OrderDate)=1997),
  OrderYear int NOT NULL DEFAULT 1997 CHECK (OrderYear=1997),
  RequiredDate datetime NULL ,
  ShippedDate datetime NULL ,
  ShipVia int NULL,
CONSTRAINT PK_Orders1997
  PRIMARY KEY (OrderYear, OrderID, CustomerId)
)
GO

CREATE TABLE Orders1998 (
  OrderID int NOT NULL ,
  CustomerID nchar (5) NOT NULL ,
  EmployeeID int NULL ,
  OrderDate datetime NOT NULL CHECK (Year(OrderDate)=1998),
  OrderYear int NOT NULL DEFAULT 1998 CHECK (OrderYear=1998),
  RequiredDate datetime NULL ,
  ShippedDate datetime NULL ,
  ShipVia int NULL,
  CONSTRAINT PK_Orders1998
  PRIMARY KEY (OrderYear, OrderID, CustomerId)
)
GO

CREATE VIEW OrdersV
AS
SELECT * FROM Orders1996
UNION ALL
SELECT * FROM Orders1997
UNION ALL
SELECT * FROM Orders1998
GO

SELECT * FROM OrdersV WHERE OrderYear=1997 AND OrderID=1000

(Результаты приведены в сокращенном виде)

------------------------------------------------------------------------
Executes    StmtText
----------- ------------------------------------------------------------
1            SELECT * FROM [OrdersV] WHERE [OrderYear]=@1 AND [Orde
1             |--Concatenation
1                  |--Filter(WHERE:(STARTUP EXPR(Convert([@1])=199
```

```
0                    |      |--Clustered Index Seek(OBJECT:([Northwind
1                    |--Filter(WHERE:(STARTUP EXPR(Convert([@1])=199
1                    |      |--Clustered Index Seek(OBJECT:([Northwind
1                    |--Filter(WHERE:(STARTUP EXPR(Convert([@1])=199
0                           |--Clustered Index Seek(OBJECT:([Northwind
```

В этом листинге показано, что в первичный ключ каждой секции введен столбец CustomerID, но запрос не изменен так, чтобы в него был включен этот новый столбец в качестве критерия поиска. В результате создается план, в котором сервер устраняет ненужные секции на этапе прогона вместо устранения этих секций оптимизатором на этапе компиляции (см. столбец Executes). Рассмотрим, что произойдет после введения столбца CustomerID в критерии поиска запроса (листинг 17.6).

Листинг 17.6. Введение столбца CustomerID в состав критериев запроса для устранения ненужных секций на этапе компиляции

```
SELECT * FROM OrdersV
WHERE OrderYear=1997 AND OrderID=1000 AND CustomerID = 'AAAAA'

StmtText
------------------------------------------------------------------
SELECT * FROM OrdersV WHERE OrderYear=@1 AND OrderID=@2 AND Custom
  |-Compute Scalar(DEFINE:(Orders1997.OrderID=Orders1997.OrderID, O
    |-Clustered Index Scan(OBJECT:(Northwind.dbo.Orders1997.PK_Order
```

В данном случае снова наблюдается ситуация с устранением ненужных секций на этапе компиляции, поскольку критерии запроса имеют взаимно-однозначное соответствие со столбцами первичного ключа секционированного представления. В этом случае устранение ненужных секцией на этапе компиляции достигнуто благодаря добавлению столбцов к критериям поиска. Можно было бы столь же просто добиться той же цели, удалив столбцы из первичного ключа каждой секции.

Как и в самом первом примере, приведенном в данном разделе, не выдвигается требование, согласно которому необходимо всегда задавать критерий выборки по всему первичному ключу для получения эффективного плана выполнения запроса к секционированному представлению. Достаточно учитывать, что такая необходимость может возникнуть, если потребуется вынудить оптимизатор удалять ненужные секции на этапе компиляции, а не на этапе прогона.

Оператор BETWEEN и запросы к секционированному представлению

При создании секционированных представлений необходимо учитывать не только связь "первичный ключ–столбец секционирования", но и обращать внимание на использование тета-операторов (операторов сравнения, отличных от операторов сравнения на равенство). Даже если имеет место полное согласование операторов и в ограничении CHECK, применяемом для секционирования, и в запросе, оптимизатор может оказаться неспособным правильно выявить секцию, в которой должен выполняться поиск, если для сравнения используются тета-операторы. Это

означает, что оптимизатор может предусмотреть поиск во всех разделах с последующим объединением результатов. Например, рассмотрим секционированное представление и запрос, показанные в листинге 17.7.

Листинг 17.7. Запрос, в котором для выборки данных из секционированного представления применяется предикат BETWEEN

```
CREATE TABLE CustomersUS (
  CustomerID nchar (5) NOT NULL,
  CompanyName nvarchar (40) NOT NULL ,
  ContactName nvarchar (30) NULL ,
  ContactTitle nvarchar (30) NULL ,
  Address nvarchar (60) NULL ,
  City nvarchar (15) NULL ,
  Region nvarchar (15) NULL ,
  PostalCode nvarchar (10) NULL ,
  Country nvarchar (15) NOT NULL CHECK (Country='US'),
  Phone nvarchar (24) NULL ,
  Fax nvarchar (24) NULL,
PRIMARY KEY (Country, CustomerID)
)

CREATE TABLE CustomersUK (
  CustomerID nchar (5) NOT NULL,
  CompanyName nvarchar (40) NOT NULL ,
  ContactName nvarchar (30) NULL ,
  ContactTitle nvarchar (30) NULL ,
  Address nvarchar (60) NULL ,
  City nvarchar (15) NULL ,
  Region nvarchar (15) NULL ,
  PostalCode nvarchar (10) NULL ,
  Country nvarchar (15) NOT NULL CHECK (Country='UK'),
  Phone nvarchar (24) NULL ,
  Fax nvarchar (24) NULL,
PRIMARY KEY (Country, CustomerID)
)

CREATE TABLE CustomersFrance (
  CustomerID nchar (5) NOT NULL,
  CompanyName nvarchar (40) NOT NULL ,
  ContactName nvarchar (30) NULL ,
  ContactTitle nvarchar (30) NULL ,
  Address nvarchar (60) NULL ,
  City nvarchar (15) NULL ,
  Region nvarchar (15) NULL ,
  PostalCode nvarchar (10) NULL ,
  Country nvarchar (15) NOT NULL CHECK (Country='France'),
  Phone nvarchar (24) NULL ,
  Fax nvarchar (24) NULL,
PRIMARY KEY (Country, CustomerID)
)

GO

DROP VIEW CustomersV
```

```
GO

CREATE VIEW CustomersV
AS
SELECT * FROM dbo.CustomersUS
UNION ALL
SELECT * FROM dbo.CustomersUK
UNION ALL
SELECT * FROM dbo.CustomersFrance
GO

SELECT * FROM dbo.CustomersV WHERE Country BETWEEN 'UK' AND 'US'
```

В данном случае столбец секционирования рассматриваемого представления совпадает с первичным ключом каждой основополагающей таблицы, и имеет место полное соответствие между критерием выборки запроса и применяемым для секционирования ограничением CHECK. Однако несмотря на это создается план выполнения, приведенный в листинге 17.8.

Листинг 17.8. План выполнения запроса, показанного в листинге 17.7

```
(Результаты приведены в сокращенном виде)

Executes  StmtText
--------- ------------------------------------------------------------
1          SELECT * FROM [dbo].[CustomersV] WHERE [Country]>=@1 AND
1           |--Concatenation
1                |--Filter(WHERE:(STARTUP EXPR(Convert([@1])<='US'
1                |     |--Clustered Index Seek(OBJECT:([Northwind].
1                |--Filter(WHERE:(STARTUP EXPR(Convert([@1])<='UK'
1                |     |--Clustered Index Seek(OBJECT:([Northwind].
1                |--Filter(WHERE:(STARTUP EXPR(Convert([@1])<='Fra
0                     |--Clustered Index Seek(OBJECT:([Northwind].
```

Базовая таблица CustomersFrance упоминается в плане, даже несмотря на то, что в принципе невозможно, чтобы таблица CustomersFrance включала искомые данные. Однако заслуживает внимания то, что поиск в таблице CustomersFrance фактически не происходит (оператору Clustered Index Seek этой таблицы соответствуют 0 случаев выполнения). И в данном случае оптимизатор выработал план, в котором ненужные секции устраняются на этапе прогона. Это — более гибкий и в целом более полезный план по сравнению с тем, в котором ненужные секции устраняются непосредственно на этапе компиляции плана.

Распределенные секционированные представления

Распределенным секционированным представлением называется секционированное представление, базовые таблицы которого разнесены по отдельным серверам, входящим в объединение (группу) серверов. Доступ к удаленным базовым таблицам осуществляется с учетом определений связанных серверов. Для создания распределенного секционированного представления необходимо выполнить описанные ниже действия.

1. Создать определения связанных серверов для тех серверов, где находятся удаленные таблицы, к которым необходимо обеспечить доступ.
2. Разрешить серверную опцию отложенной проверки по схеме для каждого из связанных серверов. Значение этой опции невозможно задать с помощью диалогового окна Linked Server Properties программы Enterprise Manager, поэтому следует использовать процедуру `sp_serveroption`.
3. Создать секционированное представление, которое ссылается на удаленные секции с использованием четырехкомпонентных имен.
4. Повторять эти действия для каждого связанного сервера, на который имеется ссылка в секционированном представлении. Это позволяет уравновешивать нагрузку в среде SQL Server, направляя пользователей к различным версиям одного и того же представления.

В листинге 17.9 показано приведенное выше секционированное представление, преобразованное в форму распределенного секционированного представления.

Листинг 17.9. Простейший вариант распределенного секционированного представления

```
CREATE VIEW OrdersV
AS
SELECT * FROM Orders1996
UNION ALL
SELECT * FROM HOMER.Northwind.dbo.Orders1997
UNION ALL
SELECT * FROM MARGE.Northwind.dbo.Orders1998
GO

SELECT CustomerID FROM OrdersV WHERE OrderYear=1997 AND
    OrderID=1000
(Результаты приведены в сокращенном виде)
StmtText
------------------------------------------------------------------------------
SELECT CustomerID=CustomerID FROM OrdersV WHERE OrderYear=@1 AND OrderID=@2
  |-Compute Scalar(DEFINE:(HOMER.Northwind.dbo.Orders1997.CustomerID=HOMER.nort
  |-Remote Query(SOURCE:(HOMER),QUERY:(SELECT Col1024 FROM (SELECT Tbl1003.
     "OrderID" Col1023,Tbl1003."CustomerID" Col1024,Tbl1003."OrderYear" Col1027
    FROM "northwind"."dbo"."Orders1997" Tbl1003) Qry1031 WHERE Col1023=(1000)))
```

Вполне очевидно, что оптимизатор имеет возможность должным образом сосредоточить поиск только на одной секции при условии, что критерии выборки запроса правильно согласованы с первичным ключом секционированного представления. Поскольку соответствующая секция находится на связанном сервере, оптимизатор вводит в план такой шаг, как `Remote Query` (удаленный запрос), и направляет этот запрос на удаленный сервер. Обратите внимание на то, что конструкция `WHERE` удаленного запроса (выделенная полужирным шрифтом) не включает столбец секционирования, даже несмотря на то, что этот столбец включен в первоначальный запрос. Это связано с тем, что указанный столбец не требуется. После выявления нужной секции необходимость в использовании самого столбца секционирования для поиска данных в удаленной таблице отпадает. Благодаря наличию ограничения `CHECK` оптимизатор легко определяет, что таблица `Orders1997` содержит данные, относящиеся только к одной секции, с данными за 1997 год.

Резюме

Объединение серверов создается путем формирования секционированного представления, которое распределяется по группе серверов. Серверы обращаются друг к другу с помощью обычных ссылок на связанные серверы.

Для использования секционированных представлений в программе SQL Server необходимо выполнить целый ряд ограничений, но после того как эти ограничения будут соблюдены, оптимизатор запросов каждого сервера объединения обеспечивает устранение ненужных секций (либо на этапе компиляции, либо на этапе прогона) для того, чтобы не происходил ненужный просмотр этих секций в ходе поиска строк, соответствующих запросу.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Если при просмотре плана выполнения запроса, применяемого к секционированному представлению, обнаруживается ненужная секция, то, по-видимому, причиной этой ситуации стала ошибка в программном обеспечении оптимизатора запросов и об этом необходимо сообщить в компанию Microsoft.
2. На что указывает значение `0` столбца `Executes`, соответствующее какому-то конкретному шагу плана запроса?
3. Подтвердите или опровергните следующее утверждение. Обеспечить максимальную производительность запроса можно лишь при том условии, что не будут перекрываться ограничения `CHECK`, которые определяют столбец секционирования для секционированного представления.
4. Опишите назначение столбца секционирования в секционированном представлении.
5. Подтвердите или опровергните следующее утверждение. Оптимизатор запросов SQL Server может устранить ненужную секцию из запроса к секционированному представлению не только на этапе компиляции, но и на этапе прогона.
6. Подтвердите или опровергните следующее утверждение. Хотя и существует возможность вынудить оптимизатор устранять ненужные секции на этапе компиляции, а не на этапе прогона, как правило, лучше предоставить оптимизатору возможность устранять ненужные секции на этапе прогона.
7. Подтвердите или опровергните следующее утверждение. Чтобы установить распределенное секционированное представление, необходимо запретить опцию отложенной проверки по схеме.

ГЛАВА 18

Технологии SQLXML

ПРИМЕЧАНИЕ. В данной главе предполагается, что читатель использует по меньшей мере версию SQL Server 2000 с установленным пакетом SQLXML 3.0. Каждый новый выпуск Web Release пакета SQLXML изменяет и расширяет функциональные возможности программы SQL Server по поддержке языка XML. Чтобы описание, приведенное в этой книге, не отставало от уровня технологий SQLXML, автор рассматривает в настоящей главе новейшую версию SQLXML, а не версию, которая поставлялась с первоначальным выпуском SQL Server 2000.

В этой главе приведено обновленное описание средств SQLXML из последней книги автора, *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML*. Указанная книга была написана до начала поставки выпуска Web Release 1 (выпуска, в котором был обновлен первоначальный вариант функциональных средств SQLXML программы SQL Server 2000). А ко времени написания настоящей книги началась поставка версии SQLXML 3.0 (которая должна была быть эквивалентна выпуску Web Release 3, если бы компания Microsoft не отказалась от прежней схемы именования своих программных продуктов). Кроме того, подготавливалась к проверке бета-версия следующего выпуска программы SQL Server, известная под названием Yukon.

К тому же, в данной главе приведен больший объем информации о том, какие проекты лежат в основе технологий SQLXML, и как эти технологии согласуются друг с другом с точки зрения архитектуры программного обеспечения. Как и во всей книге, автор при написании этой главы пытался выйти за рамки изложения рекомендаций и показать принципы, лежащие в основе функционирования технологий SQL Server.

Автор должен признать, что, приступая к написанию данной главы, он долго колебался. Выбор стоял между тем, чтобы просто обновить описание SQLXML, приведенное в его последней книге, которое было в большей степени сосредоточено на практическом применении SQLXML, но, по мнению автора, действительно нуждалось в обновлении, или написать нечто совершенно новое, посвященное только архитектурным аспектам SQLXML, вообще не затрагивая или касаясь в минимальной степени того, как эти технологии должны применяться на практике. В конечном итоге автор решил сделать и то, и другое. Согласно с основным назначением настоящей книги автор принял решение описать архитектурные аспекты SQLXML, а для обновления изложенного материала с учетом текущего состояния развития семейства технологий XML в программе SQL Server автор решил дополнить изложение тематики SQLXML из своей последней книги с точки зрения практического применения. Итак, в этой главе приведено обнов-

ленное изложение материала, посвященного SQLXML, которое было ранее опубликовано автором, а также даны подробные сведения об архитектуре SQLXML, которые до сих пор не рассматривались автором.

Краткий обзор

Язык XML стал чрезвычайно популярным и получил необычайно широкое распространение, поэтому нет ничего удивительного в том, что в программе SQL Server предусмотрена исчерпывающая поддержка средств, предназначенных для работы с этим языком. Как и в большинстве современных СУБД, в программе SQL Server регулярно возникает необходимость работать с данными и хранить данные, которые могли быть первоначально оформлены на языке XML. Без встроенной поддержки операций передачи данных XML в среду и из среды SQL Server разработчику приложений приходилось бы самому преобразовывать данные, представленные в коде XML, перед передачей их в базу данных SQL Server, а затем снова преобразовывать данные в код XML после получения из базы данных. Безусловно, подобные задачи вскоре стали бы требовать выполнения очень большого объема работы, учитывая то, насколько распространенным стал язык XML.

SQL Server – это СУБД, в которой поддерживается язык XML. Это означает, что программа SQL Server способна читать и писать данные XML. Данная программа позволяет выводить данные из баз данных в формате XML, а также читать и обновлять данные, хранящиеся в документах XML. Как показывает табл. 18.1, стандартные средства поддержки XML в программе SQL Server можно подразделить на восемь основных категорий.

Мы рассмотрим каждое из этих средств в настоящей главе и обсудим, как они работают и взаимодействуют.

Таблица 18.1. Средства XML программы SQL Server

Средство	Назначение
`FOR XML`	Расширение оператора `SELECT`, которое позволяет получать результирующие наборы, представленные в коде XML
`OPENXML`	Функция, которая позволяет выполнять чтение и запись данных, представленных в виде документов XML
Запрос XPath	Средство, позволяющее выполнять запросы к базам данных SQL Server с использованием синтаксических конструкций XPath
Схема	Средства поддержки схем отображения XSD и XDR, которые обеспечивают также выполнение запросов XPath с помощью таких схем
Средства поддержки SOAP	Средства, позволяющие клиентам получать доступ к функциональным возможностям SQL Server как к службе Web
Шаблоны обновления	Шаблоны XML, с помощью которых к базе данных могут применяться операции модификации данных

Окончание табл. 18.1

Средство	Назначение
Управляемые классы	Классы, которые предоставляют доступ к функциональным возможностям технологий SQLXML, предусмотренным в инфраструктуре .NET Framework
XML Bulk Load	Высокоскоростное программное средство, предназначенное для загрузки данных XML в базу данных SQL Server

Синтаксический анализатор MSXML

В программе SQL Server для загрузки данных XML используется синтаксический анализатор компании Microsoft, MSXML, поэтому начнем изложение темы данной главы с описания указанного синтаксического анализатора. Существует два основных способа синтаксического анализа данных XML с применением средств MSXML – синтаксический анализ с помощью объектной модели документа (Document Object Model – DOM) или синтаксический анализ с помощью простого API-интерфейса для XML (Simple API for XML – SAX). И DOM, и SAX – это стандарты консорциума W3C. Средства DOM предусматривают синтаксический анализ документа XML и загрузку документа в память в виде древовидной структуры. При осуществлении обработки таким способом весь документ материализуется и сохраняется в памяти. Результаты синтаксического анализа документа XML с помощью средств DOM принято называть *документом DOM* (или для краткости просто "DOM"). Синтаксические анализаторы XML предоставляют многочисленные способы манипулирования документами DOM. В листинге 18.1 приведено короткое приложение Visual Basic, которое показывает, как выполнить синтаксический анализ документа XML с помощью средств DOM, а затем найти с помощью запроса конкретный набор узлов. (Исходный код этого приложения приведен в подкаталоге `CH18\msxmltest` компакт-диска, прилагаемого к данной книге.)

Листинг 18.1. Пример синтаксического анализа документа XML с помощью средств DOM и выполнения запроса к документу DOM

```
Private Sub Command1_Click()

  Dim bstrDoc As String

  bstrDoc = "<Songs> " & _
                "<Song>One More Day</Song>" & _
                "<Song>Hard Habit to Break</Song>" & _
                 "<Song>Forever</Song>" & _
                "<Song>Boys of Summer</Song>" & _
                 "<Song>Cherish</Song>" & _
                "<Song>Dance</Song>" & _
                "<Song>I Will Always Love You</Song>" & _
            "</Songs>"

  Dim xmlDoc As New DOMDocument30
```

```
  If Len(Text1.Text) = 0 Then
    Text1.Text = bstrDoc
  End If
If Not xmlDoc.loadXML(Text1.Text) Then
    MsgBox "Error loading document"
  Else
    Dim oNodes As IXMLDOMNodeList
     Dim oNode As IXMLDOMNode

    If Len(Text2.Text) = 0 Then
      Text2.Text = "//Song"
    End If
    Set oNodes = xmlDoc.selectNodes(Text2.Text)

     For Each oNode In oNodes
      If Not (oNode Is Nothing) Then
        sName = oNode.nodeName
        sData = oNode.xml
        MsgBox "Node <" + sName + ">:" _
            + vbNewLine + vbTab + sData + vbNewLine
      End If
    Next

    Set xmlDoc = Nothing
  End If
End Sub
```

Работа этого приложения начинается с создания экземпляра объекта `DOMDocument`. После этого вызывается метод `loadXML` объекта для синтаксического анализа документа XML и загрузки документа в дерево DOM. Для запроса документа с помощью языка XPath вызывается метод `selectNodes` документа. Метод `selectNodes` возвращает объект списка узлов, который затем подвергается обработке в цикле с помощью конструкции `For Each`. В данном приложении с помощью функции `MsgBox` языка Visual Basic вначале отображается имя каждого узла, а затем его содержимое. Очевидно, что в этом случае существует возможность обращаться к документу и выполнять различные манипуляции так, как если бы это был объект. Но дело в том, что полученный документ и является именно таковым — синтаксический анализ документа XML с помощью средств DOM преобразует исходный документ в объект документа, находящийся в памяти, с которым затем можно работать, как с любым другим объектом.

В отличие от этого, средства SAX представляют собой API-интерфейс, управляемый событиями. Обработка документа XML с помощью средств SAX осуществляется путем настройки конфигурации приложения в целях формирования ответов на события SAX. По мере того как процессор SAX читает документ XML, активизируются события, и это происходит каждый раз, когда обнаруживаются объекты, о которых требуется передать информацию в вызывающее приложение. К числу рассматриваемых объектов относятся начальный или конечный дескриптор элемента, начало или конец определения атрибута и т.д. Средства SAX передают соответствующие данные, касающиеся рассматриваемого события, в обработчик событий, предусмотренный в приложении. Затем в приложении может быть принято решение о том, какие действия необходимо выполнить в ответ.

Приложение может сохранить данные о событии в древовидной структуре определенного типа, как и в случае обработки с помощью средств DOM. Событие может быть проигнорировано; в данных о событии может быть выполнен поиск какой-то конкретной информации; а также могут быть предприняты какие-то другие действия. После обработки данных о событии процессор SAX продолжает читать документ. В отличие от процессора DOM, процессор SAX ни при каких условиях не сохраняет документ в памяти, а фактически используется исключительно в качестве механизма синтаксического анализа, с которым приложение может соединить свои собственные функциональные средства. В действительности SAX служит в качестве основополагающего механизма синтаксического анализа для процессора DOM синтаксического анализатора MSXML. В реализации DOM компании Microsoft применяются обработчики событий SAX, которые просто сохраняют в дереве DOM данные, переданные им машиной SAX.

Читатель уже, по-видимому, понял, что процессор SAX занимает гораздо меньший объем памяти по сравнению с процессором DOM. Несмотря на такую положительную особенность, процессор SAX имеет и недостатки, связанные с тем, что его настройка и эксплуатация гораздо сложнее. Дело в том, что API-интерфейс DOM предусматривает сохранение документов в памяти, поэтому работа с документами XML становится такой же простой, как и работа с объектами любых других типов.

В программе SQL Server для обработки документов, загруженных с помощью процедуры `sp_xml_preparedocument`, используются синтаксический анализатор MSXML и процессор DOM. Но программа SQL Server ограничивает объем виртуальной памяти, который может использоваться синтаксическим анализатором MSXML для обработки документа с помощью процессора DOM, одной восьмой объема физической памяти на компьютере или объемом 500 Мбайт, в зависимости от того, какое из этих значений меньше. Тем не менее, на практике весьма маловероятно, что синтаксический анализатор MSXML будет иметь возможность обращаться к 500 Мбайт виртуальной памяти даже на компьютере с объемом физической памяти, равным 4 Гбайт. Причина этого состоит в том, что по умолчанию программа SQL Server резервирует основную часть пространства адресов непривилегированного режима для своего пула буферов. Напомним сказанное о пространстве MemToLeave в главе 11, где было, в частности, отмечено, что в версии SQL Server 2000 объем памяти, не относящейся к стекам потоков, по умолчанию составляет 256 Мбайт. Это означает, что по умолчанию синтаксический анализатор MSXML не будет иметь возможности использовать больше 256 Мбайт памяти независимо от объема физической памяти на компьютере (возможно, даже значительно меньше, поскольку эту область занимают также другие компоненты программы SQL Server).

Причина, по которой синтаксический анализатор MSXML ограничивается применением не больше чем 500 Мбайт виртуальной памяти независимо от объема памяти на компьютере, состоит в том, что для определения объема доступной физической памяти в программе SQL Server вызывается функция `GlobalMemoryStatus` API-интерфейса Win32. Функция `GlobalMemoryStatus` заполняет структуру `MEMORYSTATUS` информацией о состоянии использования памяти на компьютере. На компьютерах с объемом физической памяти больше 4 Гбайт функция `GlobalMemoryStatus` может возвращать неправильную информацию, поэтому

операционная система Windows возвращает -1 для обозначения переполнения. Безусловно, предусмотрена также функция GlobalMemoryStatusEx API-интерфейса Win32, позволяющая преодолеть это ограничение, но указанная функция не вызывается средствами SQLXML. В этом можно убедиться самостоятельно, проработав следующее упражнение.

Упражнение

Используемая система должна быть предназначена для испытания или разработки и, в идеальном случае вы должны быть единственным пользователем системы.

Упражнение 18.1. Ознакомление с тем, как синтаксический анализатор MSXML вычисляет верхний предел доступной ему памяти

1. Перезапустите программу SQL Server, лучше всего с терминала, поскольку будет осуществляться подключение к этой программе с помощью отладчика WinDbg.
2. Запустите программу Query Analyzer и подключитесь к программе SQL Server.
3. Подключитесь к программе SQL Server с помощью отладчика WinDbg. (Нажмите клавишу <F6> и выберите `sqlservr.exe` из списка выполняющихся заданий; если на компьютере работает несколько экземпляров, обязательно выберите правильный экземпляр.)
4. В приглашении к вводу команды WinDbg введите следующую точку останова:
   ```
   bp kernel32!GlobalMemoryStatus
   ```
5. После введения этой точки останова введите `g` и нажмите клавишу <Enter>, чтобы разрешить программе SQL Server продолжить работу.
6. Затем вернитесь в программу Query Analyzer и выполните запрос, приведенный в листинге 18.2.

Листинг 18.2. Запрос, применяемый для проверки работы процедуры `sp_xml_preparedocument`

```
declare @doc varchar(8000)
set @doc='
<Songs>
  <Song name="She''s Like the Wind" artist="Patrick Swayze"/>
  <Song name="Hard to Say I''m Sorry" artist="Chicago"/>
  <Song name="She Loves Me" artist="Chicago"/>
  <Song name="I Can''t Make You Love Me" artist="Bonnie Raitt"/>
  <Song name="Heart of the Matter" artist="Don Henley"/>
  <Song name="Almost Like a Song" artist="Ronnie Milsap"/>
  <Song name="I''ll Be Over You" artist="Toto"/>
</Songs>
'
declare @hDoc int
exec sp_xml_preparedocument @hDoc OUT, @doc
```

7. С началом синтаксического анализа первого документа XML с помощью процедуры `sp_xml_preparedocument` средства SQLXML вызывают функцию `GlobalMemoryStatus` для определения объема физической памяти на компьютере, затем вызывают одну недокументированную функцию, экспортируемую синтаксическим анализатором MSXML, чтобы ограничить объем виртуальной памяти, который разрешено распределять этому синтаксическому анализатору. (Автор рекомендует перезапустить сервер для того, чтобы анализируемый документ действительно был первым, и выполнение программы проходило по указанному пути в коде.) Такая недокументированная функция MSXML экспортируется из библиотеки `MSXMLn.DLL` с указанием порядкового номера, а не имени, и была предусмотрена в MSXML исключительно для того, чтобы к ней могла обратиться программа SQL Server.

8. В данный момент должно сложиться впечатление, что программа Query Analyzer зависла, поскольку в отладчике WinDbg достигнута точка останова, а программа SQL Server остановлена. Снова переключитесь на отладчик WinDbg и введите `kv` в приглашении к вводу команды, чтобы сформировать дамп стека вызовов текущего потока. Этот стек должен выглядеть примерно так, как показано ниже (из данного листинга исключено все, кроме имен функций).

```
KERNEL32!GlobalMemoryStatus (FPO: [Non-Fpo])
sqlservr!CXMLLoadLibrary::DoLoad+0x1b5
sqlservr!CXMLDocsList::Load+0x58
sqlservr!CXMLDocsList::LoadXMLDocument+0x1b
sqlservr!SpXmlPrepareDocument+0x423
sqlservr!CSpecProc::ExecuteSpecial+0x334
sqlservr!CXProc::Execute+0xa3
sqlservr!CSQLSource::Execute+0x3c0
sqlservr!CStmtExec::XretLocalExec+0x14d
sqlservr!CStmtExec::XretExecute+0x31a
sqlservr!CMsqlExecContext::ExecuteStmts+0x3b9
sqlservr!CMsqlExecContext::Execute+0x1b6
sqlservr!CSQLSource::Execute+0x357
sqlservr!language_exec+0x3e1
```

9. Как было указано в главе 3, точка входа для процедуры выполнения пакетного задания на языке T-SQL в программе SQL Server носит имя `language_exec`. В нижней части распечатки стека можно видеть вызов процедуры с точкой входа `language_exec`; эта процедура была вызвана после передачи на сервер для выполнения пакетного задания T-SQL. Продвигаясь по стеку снизу вверх, можно обнаружить вызов `SpXmlPrepareDocument` — внутренней "специальной процедуры" (расширенной процедуры, реализованной внутри самого сервера, а не во внешней библиотеке DLL), которая отвечает за реализацию расширенной процедуры `sp_xml_preparedocument`. По мере дальнейшего изучения стека можно обнаружить, что процедура `SpXmlPrepareDocument` вызывает процедуру `LoadXMLDocument`, затем процедура `LoadXMLDocument` вызывает метод `Load`, метод `Load` вызывает метод `DoLoad`, а метод `DoLoad` вызывает функцию `GlobalMemoryStatus`. Итак, мы точно определили, с помощью какой функции синтаксический анализатор MSXML вычисляет объем физической памяти на компьютере, и, зная ограничения этой функции, можем установить, какой максимальный объем виртуальной памяти может использоваться синтаксическим анализатором MSXML.

10. Введите `q` и нажмите клавишу <Enter>, чтобы выйти из отладчика WinDbg. Вам придется перезапустить применяемый экземпляр программы SQL Server.

Конструкция FOR XML

Несмотря на всю мощь и удобство использования синтаксического анализатора MSXML, в программе SQL Server не предусмотрено применение MSXML во всех средствах XML, реализованных в этой программе. Например, в сервере этот синтаксический анализатор не используется даже для реализации серверных запросов FOR XML, несмотря на то, что задача создания документа DOM программным путем и возврата этого документа в виде текста является тривиальной. Тем не менее в синтаксическом анализаторе MSXML предусмотрены средства, позволяющие выполнить указанную задачу чрезвычайно просто. Например, в листинге 18.3 представлено приложение Visual Basic, в котором с помощью интерфейса ADO выполняется запрос и динамически формируется документ DOM с учетом результатов, возвращаемых запросом.

Листинг 18.3. Приложение Visual Basic, в котором используется интерфейс ADO и формируется документ XML

```
Private Sub Command1_Click()

  Dim xmlDoc As New DOMDocument30
  Dim oRootNode As IXMLDOMNode

  Set oRootNode = xmlDoc.createElement("Root")

  Set xmlDoc.documentElement = oRootNode

  Dim oAttr As IXMLDOMAttribute
  Dim oNode As IXMLDOMNode

  Dim oConn As New ADODB.Connection
  Dim oComm As New ADODB.Command
  Dim oRs As New ADODB.Recordset

  oConn.Open (Text3.Text)
  oComm.ActiveConnection = oConn

  oComm.CommandText = Text1.Text
  Set oRs = oComm.Execute

  Dim oField As ADODB.Field

  While Not oRs.EOF
    Set oNode = xmlDoc.createElement("Row")
    For Each oField In oRs.Fields
      Set oAttr = xmlDoc.createAttribute(oField.Name)
      oAttr.Value = oField.Value
      oNode.Attributes.setNamedItem oAttr
    Next
    oRootNode.appendChild oNode
    oRs.MoveNext
  Wend

  oConn.Close
```

```
  Text2.Text = xmlDoc.xml

  Set xmlDoc = Nothing
  Set oRs = Nothing
  Set oComm = Nothing
  Set oConn = Nothing
End Sub
```

Вполне очевидно, что для преобразования результирующего набора в данные на языке XML не требуется большой объем кода. Объект `Recordset` интерфейса ADO поддерживает также потоковый вывод непосредственно в документ XML (с помощью метода `Save` этого документа), поэтому, если разработчику не требуется полный контроль над процессом преобразования, то можно обойтись еще меньшим объемом кода, чем в приведенном выше примере.

Как уже было сказано, в программе SQL Server не используется синтаксический анализатор MSXML, а также не формируется документ DOM для возврата результирующего набора в виде XML. С чем это связано? И как убедиться в том, что в программе SQL Server не используется синтаксический анализатор MSXML для обработки серверных запросов с конструкцией `FOR XML`? Ответы на оба эти вопроса приведены ниже.

Ответ на первый вопрос должен быть довольно очевидным. Для формирования документа DOM из результирующего набора перед возвратом этого документа в виде текста требовалось бы, чтобы программа SQL Server сохраняла весь результирующий набор в памяти. С учетом того, что потребность в памяти для размещения версии DOM документа XML примерно от трех до пяти раз выше по сравнению с объемом памяти, необходимым для хранения самого документа, вариант с использованием DOM не может рассматриваться как экономный способ применения ресурсов. Если бы приходилось вначале полностью сохранять документы XML в памяти перед передачей клиенту, то даже для результирующих наборов `FOR XML` средней величины потребовалось бы использовать огромные объемы виртуальной памяти (еще один вариант состоит в том, что достигался бы верхний предел объема памяти MSXML, и поэтому формируемый документ был бы слишком большим, чтобы его можно было успешно создать).

Для ответа на второй вопрос еще раз рассмотрим работу программы SQL Server с помощью отладчика.

Упражнение

Используемая система должна быть предназначена для испытания или разработки, и в идеальном случае вы должны быть единственным пользователем системы.

Упражнение 18.2. Определение того, используется ли синтаксический анализатор MSXML при выполнении серверных запросов с конструкцией `FOR XML`

1. Перезапустите программу SQL Server, лучше всего с терминала, поскольку будет осуществляться подключение к этой программе с помощью отладчика WinDbg.

2. Запустите программу Query Analyzer и подключитесь к программе SQL Server.
3. Подключитесь к программе SQL Server с помощью отладчика WinDbg. (Нажмите клавишу <F6> и выберите `sqlservr.exe` из списка выполняющихся заданий; если на компьютере работает несколько экземпляров, обязательно выберите правильный экземпляр.) После того как появится приглашение к вводу команды отладчика WinDbg, введите `g` и нажмите клавишу <Enter>, чтобы разрешить программе SQL Server продолжить работу.
4. Возвратитесь в программу Query Analyzer и вызовите на выполнение запрос с конструкцией `FOR XML`, например, такой, как показано ниже.

```
SELECT * FROM (
SELECT 'Summer Dream' as Song
UNION
SELECT 'Summer Snow'
UNION
SELECT 'Crazy For You'
) s FOR XML AUTO
```

   В этом запросе объединяются несколько операторов `SELECT`, затем с помощью конструкции `FOR XML` выполняется запрос к полученному объединению как к производной таблице.
5. После вызова на выполнение этого запроса снова переключитесь на отладчик WinDbg. По всей вероятности, вы обнаружите в командном окне WinDbg некоторые сообщения `ModLoad`. Отладчик WinDbg отображает сообщение `ModLoad` после каждой загрузки того или иного модуля в пространство отлаживаемого процесса. Если бы для обслуживания рассматриваемого запроса `FOR XML` использовалась какая-либо библиотека `MSXMLn.DLL`, то в этом окне появилось бы относящееся к этой библиотеке сообщение `ModLoad`. Но в командном окне отладчика такие сообщения не появляются; это означает, что средства MSXML не применяются для обслуживания запросов `FOR XML`.
6. Если читатель имеет большой опыт в области отладки, у него могут возникнуть сомнения на тот счет, что соответствующая библиотека DLL для MSXML уже загружена, т.е. именно поэтому при выполнении запроса с конструкцией `FOR XML` не появляется относящееся к ней сообщение `ModLoad`. Это предположение можно проверить достаточно легко. Нажмите в командном окне отладчика клавиши <Ctrl+Break>, затем введите `lm` и нажмите клавишу <Enter>. Команда `lm` позволяет получить список модулей, которые в настоящее время загружены в пространство процесса. Можете ли вы обнаружить библиотеку `MSXMLn.DLL` в этом списке? Если вы не обращались к другим средствам XML программы SQL Server со времени последней перезагрузки сервера, то указанной библиотеки в списке быть не должно. Введите `g` в командном окне и нажмите клавишу <Enter>, чтобы программа SQL Server могла продолжить работу.
7. В качестве последней проверки вызовите принудительную загрузку библиотеки `MSXMLn.DLL` путем передачи запроса на проведение синтаксического анализа документа XML. Снова загрузите в программу Query Analyzer запрос, приведенный в листинге 18.2, и вызовите его на выполнение. Вы должны обнаружить появление сообщения `ModLoad`, относящееся к библиотеке DLL средств MSXML, в командном окне WinDbg.

8. Снова нажмите клавиши <Ctrl+Break>, чтобы остановить отладчик WinDbg, затем введите q и нажмите клавишу <Enter> для прекращения отладки. Вам потребуется перезапустить программу SQL Server.

Итак, на основании всех приведенных выше сведений можно сделать вывод, что в программе SQL Server при обработке серверного запроса с конструкцией FOR XML данные XML формируются с помощью собственных средств. В составе средств MSXML не предусмотрен механизм, позволяющий выполнить эту задачу в условиях ограничений по объему памяти, поэтому указанные средства не применяются сервером.

Использование конструкции FOR XML

Как было показано в упражнении 18.2, в конце оператора SELECT можно ввести конструкцию FOR XML AUTO, чтобы предусмотреть получение результата выборки в виде фрагмента документа XML. Но синтаксическая конструкция оператора выборки с использованием FOR XML в языке Transact-SQL гораздо богаче, чем было показано в этом упражнении; конструкция FOR XML поддерживает несколько опций, которые во многом повышают удобство применения этой конструкции. В данном разделе рассматриваются некоторые из упомянутых опций и приведены примеры, которые иллюстрируют их использование.

Запрос SELECT...FOR XML (серверный)

Вполне очевидно, что читатель уже оценил возможности, которые связаны с выборкой данных XML из базы данных SQL Server с использованием опций FOR XML команды SELECT. Применение конструкции FOR XML приводит к тому, что оператор SELECT возвращает результаты выполнения запроса в виде потока вывода XML, а не в виде обычного набора строк. Этот поток вывода, формируемый запросом, который выполняется в серверном приложении, может иметь один из трех форматов – RAW, AUTO или EXPLICIT. Ниже показано основное синтаксическое определение оператора SELECT с конструкцией FOR XML.

```
SELECT column list
FROM table list
WHERE filter criteria
FOR XML RAW | AUTO | EXPLICIT [, XMLDATA] [, ELEMENTS]
    [, BINARY BASE64]
```

Формат RAW предусматривает представление возвращаемых значений столбцов в виде атрибутов и включение каждой строки в универсальный элемент строки. В формате AUTO значения столбцов представлены в виде атрибутов, а каждая строка заключена в элемент с именем, соответствующим имени таблицы, из которой поступила эта строка[1]. А формат EXPLICIT позволяет разработчику взять на себя полный контроль над форматом данных XML, возвращаемых запросом.

[1] В действительности выполняются более сложные действия, чем простое присваивание имени каждой строке с учетом имени таблицы, представления или определяемой пользователем функции, которые применялись для формирования этой строки. В программе SQL Server предусмотрен ряд эвристических алгоритмов для принятия решения о том, какие имена элементов должны фактически использоваться при выполнении конструкции FOR XML AUTO.

Применение ключевого слова XMLDATA приводит к тому, что для документа, формируемого в результате выборки, возвращается схема XML-Data. Ключевое слово ELEMENTS указывает, что столбцы в составе данных XML AUTO должны быть возвращены в виде элементов, а не атрибутов. Ключевое слово BINARY BASE64 указывает, что двоичные данные должны быть перед возвратом преобразованы с использованием кодировки BASE64.

Эти опции будут рассматриваться более подробно немного ниже. Кроме того, следует отметить, что для запросов с конструкцией FOR XML предусмотрены также опции, характерные для клиентского приложения, которые не доступны в серверных запросах. Информация об этих опциях также приведена ниже.

Режим RAW

Режим RAW является простейшим из трех основных режимов FOR XML. В этом режиме выполняется очень простое преобразование результирующего набора в данные XML. Пример применения режима RAW показан в листинге 18.4.

Листинг 18.4. Пример применения режима RAW

```
SELECT CustomerId, CompanyName
FROM Customers FOR XML RAW

(Результаты приведены в сокращенном виде)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------------------------
<row CustomerId="ALFKI" CompanyName="Alfreds Futterkiste"/><row Cu
CompanyName="Ana Trujillo Emparedados y helados"/><row CustomerId=
CompanyName="Antonio Moreno Taqueria"/><row CustomerId="AROUT" Com
Horn"/><row CustomerId="BERGS" CompanyName="Berglunds snabbkop"/><
CustomerId="BLAUS" CompanyName="Blauer See Delikatessen"/><row Cus
CompanyName="Blondesddsl p_re et fils"/><row CustomerId="WELLI"
CompanyName="Wellington Importadora"/><row CustomerId="WHITC" Comp
Clover Markets"/><row CustomerId="WILMK" CompanyName="Wilman Kala"
CustomerId="WOLZA"
CompanyName="Wolski Zajazd"/>
```

Каждый столбец становится атрибутом результирующего набора, а каждая строка становится элементом с универсальным именем row.

Как было указано в предыдущей книге автора, код XML, возвращаемый конструкцией FOR XML, не является формально правильным, поскольку в нем отсутствует корневой элемент. Формально этот код представляет собой всего лишь фрагмент XML, поэтому может стать применимым для обработки синтаксическим анализатором XML только после добавления корневого элемента. В клиентском приложении можно задать свойство xml root объекта Command интерфейса ADO для того, чтобы обеспечить автоматическое формирование корневого узла при выполнении запроса с конструкцией FOR XML.

Режим AUTO

Конструкция FOR XML AUTO предоставляет больший контроль над формированием фрагмента XML, чем конструкция режима RAW. Начнем с того, что каждая строка из результирующего набора именуется в соответствии с именем таблицы, представления или определяемой пользователем функции, возвращающей результат в виде таблицы, которые использовались для формирования этой строки. Например, простой запрос с конструкцией FOR XML AUTO приведен в листинге 18.5.

Листинг 18.5. Простой запрос с конструкцией FOR XML AUTO

```
SELECT CustomerId, CompanyName
FROM Customers FOR XML AUTO

(Результаты приведены в сокращенном виде)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
----------------------------------------------------------------
<Customers CustomerId="ALFKI" CompanyName="Alfreds Futterkiste"/><
CustomerId="ANATR" CompanyName="Ana Trujillo Emparedados y helados
CustomerId="ANTON" CompanyName="Antonio Moreno Taqueria"/><Custome
CustomerId="AROUT" CompanyName="Around the Horn"/><Customers Custo
CompanyName="Vins et alcools Chevalier"/><Customers CustomerId="WA
CompanyName="Wartian Herkku"/><Customers CustomerId="WELLI" Compan
Importadora"/><Customers CustomerId="WHITC" CompanyName="White Clo
Markets"/><Customers CustomerId="WILMK" CompanyName="Wilman Kala"/
CustomerId="WOLZA"
CompanyName="Wolski Zajazd"/>
```

Обратите внимание на то, что каждая строка получает имя таблицы, применявшейся для ее формирования – Customers. Если результаты включают больше одной строки, это приводит к появлению во фрагменте больше чем одного элемента верхнего уровня (корневого элемента), а это не разрешено в языке XML.

Одно из важных различий между режимами AUTO и RAW состоит в том, как осуществляются операции соединения. В режиме RAW происходит простое взаимно-однозначное преобразование между столбцами в результирующем наборе и атрибутами во фрагменте XML. Каждая строка становится элементом во фрагменте с именем row. Формально сами эти элементы являются пустыми, поскольку они не содержат ни значений, ни субэлементов, а включают только атрибуты. Атрибуты можно рассматривать как конструкции, определяющие характеристики элемента, тогда как данные и субэлементы образуют содержимое элемента. С другой стороны, в режиме AUTO каждая строка получает имя в соответствии с источником, из которого она была получена, а строки из соединяемых таблиц вкладываются одна в другую. Пример использования режима AUTO приведен в листинге 18.6.

Листинг 18.6. Пример использования режима AUTO

```
SELECT Customers.CustomerID, CompanyName, OrderId
FROM Customers JOIN Orders
ON (Customers.CustomerId=Orders.CustomerId)
FOR XML AUTO
```

```
(Результаты приведены в сокращенном виде и отформатированы)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------------------------
<Customers CustomerID="ALFKI" CompanyName="Alfreds Futterkiste">
  <Orders OrderId="10643"/><Orders OrderId="10692"/>
  <Orders OrderId="10702"/><Orders OrderId="10835"/>
  <Orders OrderId="10952"/><Orders OrderId="11011"/>
</Customers>
<Customers CustomerID="ANATR" CompanyName="Ana Trujillo Emparedado
  <Orders OrderId="10308"/><Orders OrderId="10625"/>
  <Orders OrderId="10759"/><Orders OrderId="10926"/></Customers>
<Customers CustomerID="FRANR" CompanyName="France restauration">
  <Orders OrderId="10671"/><Orders OrderId="10860"/>
  <Orders OrderId="10971"/>
</Customers>
```

Автор отформатировал этот фрагмент XML для того, чтобы он был более удобным для чтения, а если читатель сам выполнит данный запрос в программе Query Analyzer, то увидит неотформатированный поток вывода текста XML.

Обратите внимание на то, по какому принципу включены элементы `Orders`, относящиеся к каждому заказчику, в каждый элемент `Customer`. Как уже было сказано, в режиме AUTO предусматривается вложение строк, возвращаемых после выполнения операций соединения. Заслуживает также внимания то, что в листинге 18.6 в критериях соединения использовались полные имена, а не псевдонимы таблиц. Это связано с тем, что в режиме AUTO псевдонимы таблиц применяются для задания имен возвращаемых элементов. Если же для таблицы используются сокращенные обозначения, то и в результирующем фрагменте XML соответствующие обозначения будут присвоены элементам, сформированным по данным таблицы. Безусловно, псевдонимы удобны в обычных ситуациях использования языка Transact-SQL, а при формировании кода XML фрагмент, созданный с помощью псевдонимов, становится неудобным для чтения, если псевдонимы недостаточно описательны.

Опция ELEMENTS

Применение опции ELEMENTS конструкции FOR XML AUTO приводит к тому, что в режиме AUTO происходит возврат вложенных элементов вместо атрибутов. В зависимости от потребностей делового предприятия, отображение, предусматривающее использование элементов, может оказаться более предпочтительным по сравнению с применяемым по умолчанию отображением, предусматривающим использование атрибутов. В листинге 18.7 приведен пример запроса FOR XML, который возвращает элементы вместо атрибутов.

Обратите внимание на то, что применение опции ELEMENTS привело к возврату в виде субэлементов тех данных, которые в предыдущем примере были представлены в качестве атрибутов элемента `Customers`. Теперь каждый атрибут преобразовался в пару дескрипторов элементов, между которыми заключено значение из одного столбца таблицы.

Листинг 18.7. Запрос FOR XML, который возвращает элементы, а не атрибуты

```
SELECT CustomerID, CompanyName
FROM Customers
FOR XML AUTO, ELEMENTS

(Результаты приведены в сокращенном виде и отформатированы)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------
<Customers>
  <CustomerID>ALFKI</CustomerID>
  <CompanyName>Alfreds Futterkiste</CompanyName>
</Customers>
<Customers>
  <CustomerID>ANATR</CustomerID>
  <CompanyName>Ana Trujillo Emparedados y helados</CompanyName>
</Customers>
<Customers>
  <CustomerID>ANTON</CustomerID>
  <CompanyName>Antonio Moreno Taqueria</CompanyName>
</Customers>
<Customers>
  <CustomerID>AROUT</CustomerID>
  <CompanyName>Around the Horn</CompanyName>
</Customers>
<Customers>
  <CustomerID>WILMK</CustomerID>
  <CompanyName>Wilman Kala</CompanyName>
</Customers>
<Customers>
  <CustomerID>WOLZA</CustomerID>
  <CompanyName>Wolski  Zajazd</CompanyName>
</Customers>
```

ПРИМЕЧАНИЕ. В настоящее время в режиме AUTO не поддерживаются конструкции GROUP BY или агрегирующие функции. Дело в том, что с указанными конструкциями несовместимы эвристические алгоритмы, которые служат для определения имен элементов, поэтому и не допускается использование таких конструкций в запросах, выполняемых в режиме AUTO. Кроме того, сама конструкция FOR XML несовместима с ключевым словом COMPUTE, поэтому данное ключевое слово нельзя применять в запросах FOR XML любого типа.

Режим EXPLICIT

Режим EXPLICIT является более гибким (и поэтому более сложным в использовании) по сравнению с режимом RAW и режимом AUTO, но обеспечивает больший контроль над структурой кода XML при использовании конструкции FOR XML. В запросах режима EXPLICIT документы XML определяются в терминах так называемой "универсальной таблицы" — механизма возврата результирующего

набора из программы SQL Server, который позволяет скорее описать, как должен выглядеть документ, чем скомпоновать сам документ. Универсальная таблица представляет собой просто результирующий набор SQL Server со специальными заголовками столбцов, которые служат для сервера указанием, как должен быть сформирован документ XML на основании имеющихся данных. Формирование результатов с помощью универсальной таблицы можно рассматривать как основанный на использовании наборов способ вызова некоторой функции API-интерфейса и передачи параметров этой функции. Для осуществления подобного вызова и передачи параметров могут служить средства, доступные в языке Transact-SQL.

Универсальная таблица включает по одному столбцу для каждого столбца таблицы, который требуется возвратить во фрагменте XML, а также два дополнительных столбца – `Tag` и `Parent`. Столбец `Tag` содержит положительные целые числа, однозначно определяющие каждый дескриптор, который должен быть возвращен документом, а столбец `Parent` устанавливает родительско-дочерние связи между дескрипторами.

Остальные столбцы в универсальной таблице (соответствующие данным, которые требуется включить во фрагмент XML) имеют специальные имена, которые фактически состоят из многочисленных термов, разделенных восклицательными знаками "`!`". Такие специальные имена столбцов передают необходимую информацию для синтаксического анализатора SQL Server, а также служат в качестве конкретных указаний, касающихся формируемого фрагмента XML. Специальные имена имеют следующий формат:

```
Element!Tag!Attribute!Directive
```

Некоторые примеры применения таких специальных имен приведены ниже.

Для формирования запроса, выполняемого в режиме `EXPLICIT`, необходимо прежде всего определить компоновку документа XML, который должен быть получен в конечном итоге. После определения такой компоновки можно провести работу по формированию универсальной таблицы, которая позволит получить требуемый формат, действуя в обратном направлении, от документа XML к результирующему набору. Например, предположим, что требуется просто сформировать список заказчиков по данным таблицы `Customers` базы данных `Northwind`, в котором идентификатор заказчика должен быть представлен в виде атрибута, а имя компании – в виде элемента. Допустим также, что фрагмент XML, который должен быть сформирован, выглядит следующим образом:

```
<Customers CustomerId="ALFKI">Alfreds Futterkiste</Customers>
```

В листинге 18.8 приведен запрос Transact-SQL, который возвращает универсальную таблицу, определяющую указанную компоновку.

Листинг 18.8. Запрос Transact-SQL, возвращающий требуемую универсальную таблицу

```
SELECT 1 AS Tag,
NULL AS Parent,
CustomerId AS [Customers!1!CustomerId],
CompanyName AS [Customers!1]
FROM Customers
```

```
(Результаты приведены в сокращенном виде)
Tag    Parent   Customers!1!CustomerId Customers!1
------ -------- ---------------------- --------------------------
1      NULL     ALFKI                  Alfreds Futterkiste
1      NULL     ANATR                  Ana Trujillo Emparedados y
1      NULL     ANTON                  Antonio Moreno Taqueria
```

Первыми двумя столбцами являются дополнительные столбцы, указанные выше. Столбец Tag задает идентификатор для дескриптора, который может быть сформирован. Поскольку требуется сформировать только по одному элементу в расчете на каждую строку, то полям данного столбца присваивается в коде значение 1. Аналогичное утверждение является справедливым по отношению к столбцу Parent — существует только один элемент, а элемент верхнего уровня не имеет родительского элемента, поэтому столбец Parent в каждой строке содержит NULL.

Поскольку необходимо возвратить значение идентификатора заказчика в качестве атрибута, то имя атрибута задается в заголовке столбца 3 (выделено полужирным шрифтом). Кроме того, значение поля с названием компании CompanyName необходимо возвратить в качестве элемента, а не атрибута, поэтому в столбце 4 имя атрибута исключено.

Эта таблица, отдельно взятая, не выполняет никаких действий. Необходимо ввести в конец оператора, формирующего данную таблицу, ключевые слова FOR XML EXPLICIT для того, чтобы приведенные выше необычные имена столбцов получили определенный специальный смысл. Введем конструкцию FOR XML EXPLICIT в запрос и выполним этот запрос в программе Query Analyzer. Пример выполнения указанного запроса приведен в листинге 18.9.

Листинг 18.9. Запрос с конструкцией FOR XML EXPLICIT и полученные результаты

```
SELECT 1 AS Tag,
NULL AS Parent,
CustomerId AS [Customers!1!CustomerId],
CompanyName AS [Customers!1]
FROM Customers
FOR XML EXPLICIT

(Результаты приведены в сокращенном виде и отформатированы)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
----------------------------------------------------------------
<Customers CustomerId="ALFKI">Alfreds Futterkiste</Customers>
<Customers CustomerId="ANATR">Ana Trujillo Emparedados y helados
  </Customers>
<Customers CustomerId="WHITC">White Clover Markets</Customers>
<Customers CustomerId="WILMK">Wilman Kala</Customers>
<Customers CustomerId="WOLZA">Wolski Zajazd</Customers>
```

Вполне очевидно, что каждое значение CustomerId возвращается в виде атрибута, а каждое значение CompanyName возвращается в качестве данных, относящихся к элементу Customers, как уже было сказано.

Директивы

Четвертой частью многозначных заголовков столбцов, поддерживаемых запросами в режиме EXPLICIT, является раздел директив. Директивы могут использоваться для дополнительного контроля над тем, как должны быть представлены данные в результирующем фрагменте XML. Раздел директив поддерживает восемь значений (как показано в табл. 18.2).

Таблица 18.2. Директивы режима EXPLICIT

Директива	Назначение
element	Обеспечить кодирование данных столбца и представление данных в виде субэлемента
xml	Обеспечить представление данных столбца в виде субэлемента без кодирования данных
xmltext	Осуществить выборку данных из столбца переполнения и присоединить данные к документу
cdata	Обеспечить представление данных столбца в результирующем документе в виде секции CDATA
hide	Скрыть (исключить) в результирующем фрагменте XML столбец, который присутствует в универсальной таблице
id, idref и idrefs	Определить отношения между элементами из нескольких фрагментов XML в сочетании с опцией XMLDATA

Наиболее часто используемой из этих директив является директива element. Ее применение приводит к тому, что данные оформляются в виде субэлемента, а не атрибута. Например, предположим, что во фрагменте XML, кроме CustomerId и CompanyName, необходимо возвратить данные столбца ContactName (Имя представителя заказчика), а соответствующая информация должна быть представлена в виде субэлемента, а не атрибута. В листинге 18.10 показано, как должен выглядеть соответствующий запрос.

Листинг 18.10. Запрос с директивой element и полученные результаты

```
SELECT 1 AS Tag,
NULL AS Parent,
CustomerId AS [Customers!1!CustomerId],
CompanyName AS [Customers!1],
ContactName AS [Customers!1!ContactName!element]
FROM Customers
FOR XML EXPLICIT

(Результаты приведены в сокращенном виде и отформатированы)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
----------------------------------------------------------------
<Customers CustomerId="ALFKI">Alfreds Futterkiste
  <ContactName>Maria Anders</ContactName>
</Customers>
```

```
<Customers CustomerId="ANATR">Ana Trujillo Emparedados y
  <ContactName>Ana Trujillo</ContactName>
</Customers>
<Customers CustomerId="ANTON">Antonio Moreno Taqueria
  <ContactName>Antonio Moreno</ContactName>
</Customers>
<Customers CustomerId="AROUT">Around the Horn
  <ContactName>Thomas Hardy</ContactName>
</Customers>
<Customers CustomerId="BERGS">Berglunds snabbkop
  <ContactName>Christina Berglund</ContactName>
</Customers>
<Customers CustomerId="WILMK">Wilman Kala
  <ContactName>Matti Karttunen</ContactName>
</Customers>
<Customers CustomerId="WOLZA">Wolski  Zajazd
  <ContactName>Zbyszek Piestrzeniewicz</ContactName>
</Customers>
```

Вполне очевидно, что в каждый элемент `Customers` вложен элемент `ContactName` в виде субэлемента. Применение директивы `element` приводит к кодированию возвращаемых данных. Выборку тех же данных без кодирования можно выполнить с помощью директивы `xml`, как показано в листинге 18.11.

Листинг 18.11. Применение директивы `xml`

```
SELECT 1 AS Tag,
NULL AS Parent,
CustomerId AS [Customers!1!CustomerId],
CompanyName AS [Customers!1],
ContactName AS [Customers!1!ContactName!xml]
FROM Customers
FOR XML EXPLICIT
```

Использование директивы `xml` (выделена полужирным шрифтом) приводит к получению значения из столбца без кодирования каких-либо содержащихся в нем специальных символов.

Установление связей между данными

До сих пор рассматривались примеры выборки данных из одной таблицы, поэтому применяемые запросы `EXPLICIT` не были слишком сложными. Значительное усложнение таких запросов не происходит, даже если в запросе используется несколько таблиц, при условии, что разработчика не затрудняет требование повторять данные из каждой таблицы в каждом элементе верхнего уровня фрагмента XML. В соответствии с закономерностью, по которой значения столбцов из соединяемых таблиц часто повторяются в результирующих наборах запросов Transact-SQL, может быть легко создан фрагмент XML, который содержит данные из нескольких таблиц, повторяемые в каждом элементе. Но такой способ может оказаться не самым эффективным способом представления данных в языке XML. Напомним, что язык XML поддерживает иерархические связи между

элементами. Для установления соответствующих иерархий могут использоваться запросы режима EXPLICIT и операторы UNION языка T-SQL. Пример применения указанного подхода приведен в листинге 18.12.

Листинг 18.12. Создание иерархических связей между элементами с помощью запроса режима EXPLICIT

```
SELECT 1 AS Tag,
NULL AS Parent,
CustomerId AS [Customers!1!CustomerId],
CompanyName AS [Customers!1],
NULL AS [Orders!2!OrderId],
NULL AS [Orders!2!OrderDate!element]
FROM Customers
UNION
SELECT 2 AS Tag,
1 AS Parent,
CustomerId,
NULL,
OrderId,
OrderDate
FROM Orders
ORDER BY [Customers!1!CustomerId], [Orders!2!OrderDate!element]
FOR XML EXPLICIT
```

В этом запросе выполняется несколько интересных действий. Прежде всего устанавливается связь между таблицами Customers и Orders с использованием общего для них столбца CustomerId. Обратите внимание на то, что в каждом операторе SELECT упоминается третий столбец, который возвращает столбец CustomerId из каждой таблицы. Столбцы Tag и Parent определяют дополнительные подробности установления связи между двумя таблицами. Значения Tag и Parent второго запроса связывают второй запрос с первым. Эти значения определяют, что записи Order являются дочерними по отношению к записям Customer. Наконец, заслуживает внимания конструкция ORDER BY. Эта конструкция обеспечивает удобное упорядочение элементов в таблице; вначале выполняется сортировка по значениям столбца CustomerId, а затем — по значениям OrderDate каждого заказа Order. Результаты, полученные при выполнении данного запроса, приведены в листинге 18.13.

Листинг 18.13. Результаты выполнения запроса, в котором устанавливаются иерархические связи между элементами

```
(Результаты приведены в сокращенном виде и отформатированы)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------
<Customers CustomerId="ALFKI">Alfreds Futterkiste
  <Orders OrderId="10643">
     <OrderDate>1997-08-25T00:00:00</OrderDate>
  </Orders>
  <Orders OrderId="10692">
     <OrderDate>1997-10-03T00:00:00</OrderDate>
```

```
  </Orders>
  <Orders OrderId="10702">
     <OrderDate>1997-10-13T00:00:00</OrderDate>
  </Orders>
  <Orders OrderId="10835">
     <OrderDate>1998-01-15T00:00:00</OrderDate>
  </Orders>
  <Orders OrderId="10952">
     <OrderDate>1998-03-16T00:00:00</OrderDate>
  </Orders>
  <Orders OrderId="11011">
     <OrderDate>1998-04-09T00:00:00</OrderDate>
  </Orders>
</Customers>
<Customers CustomerId="ANATR">Ana Trujillo Emparedados y helados
  <Orders OrderId="10308">
     <OrderDate>1996-09-18T00:00:00</OrderDate>
  </Orders>
  <Orders OrderId="10625">
     <OrderDate>1997-08-08T00:00:00</OrderDate>
</Orders>
  <Orders OrderId="10759">
     <OrderDate>1997-11-28T00:00:00</OrderDate>
  </Orders>
  <Orders OrderId="10926">
     <OrderDate>1998-03-04T00:00:00</OrderDate>
  </Orders>
</Customers>
```

Вполне очевидно, что элементы с данными о заказах каждого заказчика вложены в элемент с данными о заказчике.

Директива `hide`

Директива `hide` позволяет исключить столбцы, введенные в универсальную таблицу из результирующего документа XML. Одним из способов использования такой функциональной возможности является упорядочение результатов по значениям столбца, который не нужно включать во фрагмент XML. Дело в том, что если для слияния таблиц не используется оператор `UNION`, то проблема удаления ненужного столбца не возникает, поскольку упорядочение может выполняться по значениям любого выбранного столбца. А если в запросе присутствует оператор `UNION`, то упорядочение должно проводиться по значениям столбцов, входящих в результирующий набор. Директива `hide` предоставляет возможность соблюсти указанное требование к составу столбцов и вместе с тем избавиться от необходимости возвращать данные, которые не требуются в результатах. Пример применения директивы `hide` приведен в листинге 18.14.

Обратите внимание на то, что директива `hide` (выделенная полужирным шрифтом) включена в заголовок столбца 5. Эта директива позволяет задавать в конструкции `ORDER BY` такие столбцы, которые фактически не появятся в результирующем фрагменте XML.

Листинг 18.14. Использование директивы hide

```
SELECT 1 AS Tag,
NULL AS Parent,
CustomerId AS [Customers!1!CustomerId],
CompanyName AS [Customers!1],
PostalCode AS [Customers!1!PostalCode!hide],
NULL AS [Orders!2!OrderId],
NULL AS [Orders!2!OrderDate!element]
FROM Customers
UNION
SELECT 2 AS Tag,
1 AS Parent,
CustomerId,
NULL,
NULL,
OrderId,
OrderDate
FROM Orders
ORDER BY [Customers!1!CustomerId], [Orders!2!OrderDate!element],
[Customers!1!PostalCode!hide]
FOR XML EXPLICIT
```

Директива cdata

Секции CDATA могут присутствовать в документе XML везде, где допускается появление символьных данных. Секция CDATA используется для маскировки символов, которые в противном случае распознавались бы как символы разметки (например, "<", ">", "/" и т.д.). Поэтому секции CDATA позволяют включать в документ XML такие фрагменты символьных данных, которые при отсутствии подобной секции могли бы нарушить работу синтаксического анализатора. Для того чтобы сформировать секцию CDATA в запросе режима EXPLICIT, необходимо включить в этот запрос директиву cdata, как показано в листинге 18.15.

Листинг 18.15. Запрос с директивой cdata и полученные результаты

```
SELECT 1 AS Tag,
NULL AS Parent,
CustomerId AS [Customers!1!CustomerId],
CompanyName AS [Customers!1],
Fax AS [Customers!1!!cdata]
FROM Customers
FOR XML EXPLICIT

(Результаты приведены в сокращенном виде и отформатированы)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
------------------------------------------------------------------
<Customers CustomerId="ALFKI">Alfreds Futterkiste
  <![CDATA[030-0076545]]>
</Customers>
<Customers CustomerId="ANATR">Ana Trujillo Emparedados y helados
```

```
  <![CDATA[(5) 555-3745]]>
</Customers>
<Customers CustomerId="ANTON">Antonio Moreno Taqueria
</Customers>
<Customers CustomerId="AROUT">Around the Horn
  <![CDATA[(171) 555-6750]]>
</Customers>
<Customers CustomerId="BERGS">Berglunds snabbkop
  <![CDATA[0921-12 34 67]]>
</Customers>
```

Как показано в листинге 18.15, каждое значение из столбца Fax возвращается во фрагменте XML в виде секции CDATA. Заслуживает также внимания то, что имя атрибута в заголовке столбца cdata (выделенного полужирным шрифтом) отсутствует. Это связано с тем, что для секций CDATA не разрешается применять имена атрибутов. Еще раз отметим, что указанные секции представляют собой замаскированные секции документа, поэтому синтаксический анализатор XML не обрабатывает какие-либо имена атрибутов или элементов, которые могут содержаться в подобных секциях.

Директивы id, idref и idrefs

Типы данных ID, IDREF и IDFREFS могут использоваться для представления в документе XML реляционных данных. Такие типы данных, заданные в определении DTD или в схеме XML-Data, устанавливают связи между элементами. Это удобно в тех ситуациях, когда требуется организовать обмен сложными данными и вместе с тем свести к минимуму дублирование данных в документе.

В запросах режима EXPLICIT директивы id, idref и idrefs могут использоваться для определения реляционных полей в документе XML. Очевидно, что связанный с этим подход может применяться, только если для определения документа и идентификации столбцов, используемых для установления связей между сущностями, применяется схема. Средства формирования встроенной схемы для фрагмента XML предоставляются опцией XMLDATA конструкции FOR XML. В сочетании с директивами id эта опция позволяет обозначать реляционные поля во фрагменте XML. Соответствующий пример приведен в листинге 18.16.

Листинг 18.16. Использование директив id и idref в запросе

```
SELECT 1 AS Tag,
       NULL AS Parent,
       CustomerId AS [Customers!1!CustomerId!id],
       CompanyName AS [Customers!1!CompanyName],
       NULL AS [Orders!2!OrderID],
       NULL AS [Orders!2!CustomerId!idref]
FROM Customers
UNION
SELECT 2,
        NULL,
        NULL,
        NULL,
```

```
OrderID,
      CustomerId
FROM Orders
ORDER BY [Orders!2!OrderID]
FOR XML EXPLICIT, XMLDATA

(Результаты приведены в сокращенном виде и отформатированы)

XML_F52E2B61-18A1-11d1-B105-00805F49916B
--------------------------------------------------------------------
<Schema name="Schema2" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Customers" content="mixed" model="open">
    <AttributeType name="CustomerId" dt:type="id"/>
    <AttributeType name="CompanyName" dt:type="string"/>
     <attribute type="CustomerId"/>
    <attribute type="CompanyName"/>
  </ElementType>
  <ElementType name="Orders" content="mixed" model="open">
    <AttributeType name="OrderID" dt:type="i4"/>
    <AttributeType name="CustomerId" dt:type="idref"/>
    <attribute type="OrderID"/>
     <attribute type="CustomerId"/>
  </ElementType>
</Schema>
<Customers xmlns="x-schema:#Schema2" CustomerId="ALFKI"
  CompanyName="Alfreds Futterkiste"/>
<Customers xmlns="x-schema:#Schema2" CustomerId="ANATR"
  CompanyName="Ana Trujillo Emparedados y helados"/>
<Customers xmlns="x-schema:#Schema2" CustomerId="ANTON"
  CompanyName="Antonio Moreno Taqueria"/>
<Customers xmlns="x-schema:#Schema2" CustomerId="AROUT"
  CompanyName="Around the Horn"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10248"
     CustomerId="VINET"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10249"
     CustomerId="TOMSP"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10250"
     CustomerId="HANAR"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10251"
     CustomerId="VICTE"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10252"
     CustomerId="SUPRD"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10253"
     CustomerId="HANAR"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10254"
     CustomerId="CHOPS"/>
<Orders xmlns="x-schema:#Schema2" OrderID="10255"
     CustomerId="RICSU"/>
```

Обратите внимание на то, как используются директивы `id` и `idref` в столбцах `CustomerId` таблиц `Customers` и `Orders` (эти директивы выделены полужирным шрифтом). Указанные директивы связывают две таблицы с помощью столбца `CustomerId`, общего для этих таблиц.

Изучение фрагмента XML, возвращенного приведенным выше запросом, показывает, что выполнение запроса начинается со схемы XML-Data, создаваемой директивой XMLDATA. Затем на эту схему сделана ссылка в следующем за ней фрагменте XML.

Запрос SELECT...FOR XML (клиентский)

Средства SQLXML обеспечивают также возможность передачи клиентскому приложению обязанностей по преобразованию результирующего набора в код XML. К этим функциональным средствам можно получить доступ с помощью управляемых классов SQLXML, шаблонов XML, параметра настройки конфигурации виртуального каталога и средства доступа SQLXMLOLEDB. В настоящем разделе рассматривается способ применения конструкции FOR XML в клиентском запросе, основанный на использовании средства доступа SQLXMLOLEDB, поскольку этот способ требует наименьших трудозатрат по настройке. Соответствующая технология остается одной и той же, независимо от используемого механизма.

Средства доступа SQLXMLOLEDB служат в качестве промежуточного уровня между клиентским приложением (или приложением среднего яруса) и собственным средством доступа SQLOLEDB программы SQL Server. Свойство Data Source средства доступа SQLXMLOLEDB определяет то средство доступа OLE DB, с помощью которого будут выполняться запросы; в настоящее время разрешается использовать для этой цели только SQLOLEDB.

Средство доступа SQLXMLOLEDB не предназначено для работы с наборами строк. Для того чтобы обеспечить применение этого средства доступа под управлением интерфейса ADO, необходимо получить к нему доступ с помощью потокового режима ADO. Примеры кода, которые показывают, как выполнить эту задачу, приведены ниже.

Для обработки клиентских запросов с конструкцией FOR XML на основе средств доступа SQLXMLOLEDB, следует выполнить такие действия.

1. Подключиться к серверу с использованием строки соединения ADO, которая определяет SQLXMLOLEDB в качестве средства доступа.
2. Присвоить значение True свойству ClientSideXML объекта Command интерфейса ADO.
3. Создать и открыть объект потока ADO и связать его со свойством Output Stream объекта Command.
4. Выполнить запрос FOR XML EXPLICIT, FOR XML RAW или FOR XML NESTED языка Transact-SQL с помощью объекта Command, задавая в вызове метода Execute опцию adExecuteStream.

Пример выполнения указанной последовательности действий приведен в листинге 18.17 (исходный код данного приложения можно найти в подкаталоге CH18\forxml_clientside компакт-диска, прилагаемого к этой книге).

Листинг 18.17. Приложение, в котором осуществляется обработка клиентского запроса FOR XML

```
Private Sub Command1_Click()
  Dim oConn As New ADODB.Connection
  Dim oComm As New ADODB.Command

  Dim stOutput As New ADODB.Stream
  stOutput.Open

  oConn.Open (Text3.Text)
  oComm.ActiveConnection = oConn
  oComm.Properties("ClientSideXML") = "True"
  If Len(Text1.Text) = 0 Then
    Text1.Text = _
      "select * from pubs..authors FOR XML NESTED"
  End If
  oComm.CommandText = Text1.Text
  oComm.Properties("Output Stream") = stOutput
  oComm.Properties("xml root") = "Root"
  oComm.Execute , , adExecuteStream

  Text2.Text = stOutput.ReadText(adReadAll)

  stOutput.Close
  oConn.Close

  Set oComm = Nothing
  Set oConn = Nothing
End Sub
```

Вполне очевидно, что основная часть действий, выполняемых в рассматриваемом приложении, связана с использованием объекта Command интерфейса ADO. Свойству ClientSideXML этого объекта присваивается значение True, а свойству Output Stream — дескриптор объекта потока ADO, который был создан до вызова метода Execute объекта Command.

Обратите внимание на то, как используется конструкция FOR XML NESTED. Опция NESTED применима только для обработки клиентских запросов с конструкцией FOR XML, поэтому ее нельзя использовать в серверных запросах. Конструкция FOR XML NESTED весьма напоминает конструкцию FOR XML AUTO, но имеет некоторые небольшие отличия. Например, если запрос в конструкции FOR XML NESTED ссылается на представление, то в сформированном коде XML применяются имена основополагающих базовых таблиц представления. Аналогичное утверждение является справедливым по отношению к псевдонимам таблиц; в сформированном коде XML используются имена базовых таблиц. Если в клиентском запросе с конструкцией FOR XML применяется опция FOR XML AUTO, это приводит к обработке запроса в серверной, а не в клиентской программе, поэтому, чтобы воспользоваться в клиентской программе функциональными возможностями, аналогичными тем, которые предоставляет опция FOR XML AUTO, следует применять опцию NESTED.

Напомним, что выше в данной главе были приведены результаты исследования того, используется ли синтаксический анализатор MSXML при выработке

кода XML в серверной программе (см. упр. 18.2). В связи с этим у читателя может возникнуть вопрос, используется ли синтаксический анализатор MSXML средствами SQLXML при обработке клиентских запросов с конструкцией FOR XML. Ответ на этот вопрос является отрицательным. И в данном случае можно подключить отладчик (к приложению forxml_clientside), чтобы убедиться в этом самостоятельно. В действительности должно быть обнаружено, что первом вызове запроса на выполнение в пространство адресов процесса, относящегося к указанному приложению, загружается библиотека SQLXMLn.DLL. Именно в этой библиотеке находятся функции средства доступа SQLXMLOLEDB, а обработка клиентских запросов с конструкцией FOR XML средствами SQLXML осуществляется именно с помощью этих функций.

Функция OPENXML

OPENXML — это встроенная функция языка Transact-SQL, которая способна возвращать документ XML в виде набора строк. При использовании в сочетании с процедурами sp_xml_preparedocument и sp_xml_removedocument функция OPENXML позволяет разбивать (или, как принято называть эту операцию, *разделять*) нереляционные документы XML на реляционные фрагменты, которые могут быть вставлены в таблицы.

По мнению автора, изучение принципов работы функции OPENXML следует начать с исследования того, где она реализована. Прежде всего необходимо получить ответ на вопрос о том, находится ли код реализации функции OPENXML в отдельной библиотеке DLL (возможно, в библиотеке SQLXMLn.DLL) или эта функция полностью реализована в исполняемом файле SQL Server.

Наиболее удобным способом получения ответа на этот вопрос является вызов программы SQL Server на выполнение под управлением отладчика, останов этой программы в ходе выполнения вызова OPENXML и проверка стека вызовов. Такой подход позволяет узнать, в каком модуле реализована рассматриваемая функция. Но имя класса (или функции), реализующего OPENXML, неизвестно, поэтому невозможно просто установить точку останова, чтобы выполнить поставленную задачу. Таким образом, нам остается только рассчитывать на свою реакцию или удачу, которая позволит остановить отладчик в нужном месте и воспользоваться выбранным подходом к выявлению модуля, в котором реализована функция OPENXML. Но в действительности это проще сказать, чем сделать. Даже если функция OPENXML применяется для обработки сложных документов, ее работа завершается довольно быстро, поэтому попытка прервать работу функции с помощью отладчика может оказаться практически не выполнимой.

Еще один способ достижения указанной цели может состоять в том, чтобы вынудить функцию OPENXML активизировать ошибку и заранее установить точку останова, чтобы остановить выполнение в стандартной процедуре SQL Server, формирующей сообщение об ошибке. Автор много лет работал с программой SQL Server, наблюдал немало ситуаций нарушения доступа и изучил значительное количество дампов стека, поэтому знает, что основной процедурой формирования

сообщений об ошибках для данного сервера является процедура ex_raise. Хотя через процедуру ex_raise проходят не все ошибки программы SQL Server, а лишь значительная их часть, имеет смысл установить точку останова на процедуре ex_raise и вынудить функцию OPENXML активизировать ошибку, чтобы узнать, сможем ли мы получить распечатку стека вызовов и установить, где реализована функция OPENXML. В упражнении 18.3 рассматривается порядок действий, позволяющий решить именно эту задачу.

Упражнение

Используемая система должна быть предназначена для испытания или разработки, и в идеальном случае вы должны быть единственным пользователем системы.

Упражнение 18.3. Определение того, где реализована функция OPENXML

1. Перезапустите программу SQL Server, лучше всего с терминала, поскольку будет осуществляться подключение к этой программе с помощью отладчика WinDbg.
2. Запустите программу Query Analyzer и подключитесь к программе SQL Server.
3. Подключитесь к программе SQL Server с помощью отладчика WinDbg. (Нажмите клавишу <F6> и выберите sqlservr.exe из списка выполняющихся заданий; если на компьютере работает несколько экземпляров, обязательно выберите правильный экземпляр.)
4. После появления приглашения к вводу команды WinDbg установите точку останова на процедуре ex_raise следующим образом:

```
bp sqlservr!ex_raise
```

5. Введите g и нажмите клавишу <Enter>, чтобы программа SQL Server могла продолжить работу.
6. Возвратитесь в программу Query Analyzer и выполните следующий запрос:

```
declare @hDoc int
set @hdoc=8675309 -- Принудительно задать фиктивный дескриптор
select * from openxml(@hdoc,'/',1)
```

7. Возникнет впечатление, что программа Query Analyzer зависла, поскольку достигнута точка останова, установленная в отладчике WinDbg. Снова переключитесь на программу WinDbg, введите kv в приглашении к вводу команды и нажмите клавишу <Enter>. В результате будет сформирован дамп стека вызовов. Полученная распечатка стека должна выглядеть примерно так, как показано ниже (автор удалил все, кроме имен функций).

```
sqlservr!ex_raise
sqlservr!CXMLDocsList::XMLMapFromHandle+0x3f
sqlservr!COpenXMLRange::GetRowset+0x14d
sqlservr!CQScanRmtScan::OpenConnection+0x141
sqlservr!CQScanRmtBase::Open+0x18
sqlservr!CQueryScan::Startup+0x10d
sqlservr!CStmtQuery::ErsqExecuteQuery+0x26b
sqlservr!CStmtSelect::XretExecute+0x229
```

```
sqlservr!CMsqlExecContext::ExecuteStmts+0x3b9
sqlservr!CMsqlExecContext::Execute+0x1b6
sqlservr!CSQLSource::Execute+0x357
sqlservr!language_exec+0x3e1
sqlservr!process_commands+0x10e
UMS!ProcessWorkRequests+0x272
UMS!ThreadStartRoutine+0x98 (FPO: [EBP 0x00bd6878] [1,0,4])
MSVCRT!_beginthread+0xce
KERNEL32!BaseThreadStart+0x52 (FPO: [Non-Fpo])
```

8. Этот стек вызовов позволяет узнать о многом. Во-первых, он сообщает, что функция OPENXML реализована непосредственно в самом сервере. Код этой функции находится в файле `sqlservr.exe` — в исполняемом файле программы SQL Server. Во-вторых, с помощью стека можно узнать, что за выработку набора строк, возвращаемого функцией OPENXML языка T-SQL, отвечает класс `COpenXMLRange`.
9. Введите `q` и нажмите клавишу <Enter>, чтобы прекратить отладку. После этого необходимо перезапустить программу SQL Server.

Рассматривая стек вызовов, можно также определить, как работает функция OPENXML. Вызов этой функции поступает на сервер в виде языкового события или события RPC (выполняемый в данном примере код, безусловно, был передан на сервер в виде языкового события; об этом сообщает запись `language_exec` в стеке вызовов) и в конечном итоге приводит к вызову метода `GetRowset` класса `COpenXMLRange`. Можно предположить, что метод `GetRowset` получает доступ к документу DOM, созданному ранее с помощью вызова процедуры `sp_xml_preparedocument`, и превращает этот документ в двумерную матрицу, которая может быть возвращена в виде набора строк, завершая тем самым работу функции OPENXML.

Теперь, зная имя класса и метода, лежащих в основе функции OPENXML, можно установить новую точку останова (на вызове `COpenXMLRange::GetRowset`), передать в функцию OPENXML дескриптор допустимого документа и изучить всю организацию выполнения указанного метода после достижения точки останова. Тем не менее, выполнив данное упражнение, мы уже получили достаточное представление о том, как работает функция OPENXML, поэтому вряд ли нам удастся узнать еще многое об архитектуре OPENXML, изучая после этого структуру стека вызовов.

Использование функции OPENXML

В оперативной документации Books Online достаточно подробно описано, как используется функция OPENXML, поэтому автор не будет повторять здесь приведенную в документации информацию. В листинге 18.18 показан простой пример того, как применяется функция OPENXML.

Листинг 18.18. Пример использования функции OPENXML и полученные результаты

```
DECLARE @hDoc int
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
  <song><name>Somebody to Love</name></song>
```

```
<song><name>These Are the Days of Our Lives</name></song>
  <song><name>Bicycle Race</name></song>
  <song><name>Who Wants to Live Forever</name></song>
  <song><name>I Want to Break Free</name></song>
  <song><name>Friends Will Be Friends</name></song>
</songs>'
SELECT * FROM OPENXML(@hdoc, '/songs/song', 2) WITH
    (name varchar(80))
EXEC sp_xml_removedocument @hDoc

(Результаты)

name
----------------------------------------------------------------
Somebody to Love
These Are the Days of Our Lives
Bicycle Race
Who Wants to Live Forever
I Want to Break Free
Friends Will Be Friends
```

Для использования функции OPENXML нужно выполнить перечисленные ниже простые действия.

1. Вызвать процедуру sp_xml_preparedocument, чтобы загрузить документ XML в память. Для преобразования документа в дерево узлов, к которому затем может быть получен доступ с помощью запроса XPath, вызывается синтаксический анализатор DOM из состава средств MSXML. Указатель на это дерево узлов возвращается процедурой в виде целого числа.
2. Вызвать оператор SELECT из функции OPENXML, передав в эту функцию дескриптор, полученный в п. 1.
3. Включить в вызов OPENXML синтаксическую конструкцию XPath, чтобы точно указать, к каким узлам необходимо получить доступ.
4. При желании включить конструкцию WITH, которая отображает документ XML в конкретную схему таблицы. Это может быть полная схема таблицы, а также ссылка на саму таблицу.

Функция OPENXML является чрезвычайно гибкой, поэтому некоторые из перечисленных выше действий имеют варианты и альтернативы. Это означает, что выше описан лишь основной ход действий, позволяющий разделять и применять в дальнейшей работе любой документ XML, обрабатываемый с помощью функции OPENXML.

В листинге 18.19 приведен вариант предыдущего запроса, в котором используется таблица для определения схемы, предназначенной для преобразования документа.

Листинг 18.19. Запрос, в котором определяется схема, и полученные результаты

```
USE tempdb
GO
create table songs (name varchar(80))
go
DECLARE @hDoc int
```

```
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
  <song><name>Somebody to Love</name></song>
  <song><name>These Are the Days of Our Lives</name></song>
  <song><name>Bicycle Race</name></song>
  <song><name>Who Wants to Live Forever</name></song>
  <song><name>I Want to Break Free</name></song>
  <song><name>Friends Will Be Friends</name></song>
</songs>'
SELECT * FROM OPENXML(@hdoc, '/songs/song', 2) WITH songs
EXEC sp_xml_removedocument @hDoc
GO
DROP TABLE songs

(Результаты)

name
-----------------------------------------------------------------
Somebody to Love
These Are the Days of Our Lives
Bicycle Race
Who Wants to Live Forever
I Want to Break Free
Friends Will Be Friends
```

Для определения подробностей преобразования документа XML в таблицы базы данных может также использоваться конструкция WITH, как показано в листинге 18.20.

Листинг 18.20. Применение в запросе конструкции `WITH` и полученные результаты

```
DECLARE @hDoc int
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
  <artist name="Johnny Hartman">
  <song> <name>It Was Almost Like a Song</name></song>
<song> <name>I See Your Face Before Me</name></song>
  <song> <name>For All We Know</name></song>
  <song> <name>Easy Living</name></song>
  </artist>
  <artist name="Harry Connick, Jr.">
  <song> <name>Sonny Cried</name></song>
  <song> <name>A Nightingale Sang in Berkeley Square</name></song>
  <song> <name>Heavenly</name></song>
  <song> <name>You Didn''t Know Me When</name></song>
  </artist>
</songs>'
SELECT * FROM OPENXML(@hdoc, '/songs/artist/song', 2)
WITH (artist varchar(30) '../@name',
      song varchar(50) 'name')
EXEC sp_xml_removedocument @hDoc

(Результаты)

artist                         song
------------------------------ ---------------------------------------
```

```
Johnny Hartman              It Was Almost Like a Song
Johnny Hartman              I See Your Face Before Me
Johnny Hartman              For All We Know
Johnny Hartman              Easy Living
Harry Connick, Jr.          Sonny Cried
Harry Connick, Jr.          A Nightingale Sang in Berkeley Square
Harry Connick, Jr.          Heavenly
Harry Connick, Jr.          You Didn't Know Me When
```

Обратите внимание на то, что перед ссылками на атрибуты применяется префикс в виде символа @. В листинге 18.20 задается конструкция XPath, с помощью которой осуществляется прохождение по дереву вниз до элемента song, затем формируется ссылка на атрибут name элемента artist — родительского элемента по отношению к элементу song. Для получения данных второго столбца осуществляется выборка дочернего по отношению к song элемента, который также носит имя name.

В листинге 18.21 приведен еще один пример.

Листинг 18.21. Еще один запрос с конструкцией WITH

```
DECLARE @hDoc int
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
<artist> <name>Johnny Hartman</name>
  <song> <name>It Was Almost Like a Song</name></song>
  <song> <name>I See Your Face Before Me</name></song>
  <song> <name>For All We Know</name></song>
  <song> <name>Easy Living</name></song>
  </artist>
  <artist> <name>Harry Connick, Jr.</name>
  <song> <name>Sonny Cried</name></song>
  <song> <name>A Nightingale Sang in Berkeley Square</name></song>
  <song> <name>Heavenly</name></song>
  <song> <name>You Didn''t Know Me When</name></song>
  </artist>
</songs>'
SELECT * FROM OPENXML(@hdoc, '/songs/artist/name', 2)
WITH (artist varchar(30) '.',
      song varchar(50) '../song/name')
EXEC sp_xml_removedocument @hDoc

(Результаты)

artist                      song
--------------------------- ---------------------------------------
Johnny Hartman              It Was Almost Like a Song
Harry Connick, Jr.          Sonny Cried
```

Обратите внимание на то, что при выполнении данного запроса получено всего две строки. С чем это связано? Причина состоит в том, что применяемый шаблон XPath предусматривает переход к узлу artist/name, а таких узлов имеется всего два. Кроме получения элемента name, относящегося к каждому элементу

artist, была также выполнена выборка элемента name из первого элемента song, относящегося к элементу artist. В предыдущем запросе шаблон XPath приводил к элементу song (а общее количество таких элементов равно восьми), затем ссылался на родительский узел каждого элемента song (на соответствующий ему элемент artist) с помощью обозначения ".." языка XPath.

Обратите внимание на то, что в листинге 18.21 используется уточнитель "." языка XPath. Этот уточнитель просто ссылается на текущий элемент и действительно здесь требуется, поскольку имя текущего элемента изменяется с name на artist. Рекомендуем применять такой прием, когда требуется переименовать элемент, возвращаемый с помощью функции OPENXML.

Параметр flags

Параметр flags функции OPENXML позволяет указать, в какой форме должен обрабатываться документ: предусматривающей использование атрибутов; предусматривающей использование элементов, или в форме, которая представляет собой некоторое сочетание этих двух подходов. До сих пор в данной главе в качестве параметра flags задавалось значение 2, которое определяет отображение, предусматривающее использование элементов. В листинге 18.22 приведен пример отображения, предусматривающего применение атрибутов.

Листинг 18.22. Отображение, предусматривающее использование атрибутов, и полученные результаты

```
DECLARE @hDoc int
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
  <artist name="Johnny Hartman">
  <song name="It Was Almost Like a Song"/>
  <song name="I See Your Face Before Me"/>
  <song name="For All We Know"/>
  <song name="Easy Living"/>
  </artist>
  <artist name="Harry Connick, Jr.">
  <song name="Sonny Cried"/>
  <song name="A Nightingale Sang in Berkeley Square"/>
  <song name="Heavenly"/>
  <song name="You Didn''t Know Me When"/>
  </artist>
</songs>'
SELECT * FROM OPENXML(@hdoc, '/songs/artist/song', 1)
WITH (artist varchar(30) '../@name',
      song varchar(50) '@name')
EXEC sp_xml_removedocument @hDoc

(Результаты)

artist                         song
------------------------------ ---------------------------------------
Johnny Hartman                 It Was Almost Like a Song
Johnny Hartman                 I See Your Face Before Me
```

```
Johnny Hartman                For All We Know
Johnny Hartman                 Easy Living
Harry Connick, Jr.             Sonny Cried
Harry Connick, Jr.            A Nightingale Sang in Berkeley Square
Harry Connick, Jr.            Heavenly
Harry Connick, Jr.            You Didn't Know Me When
```

Формат краевой таблицы

В операторе SELECT с функцией OPENXML можно полностью исключить конструкцию WITH, чтобы выполнить выборку части документа XML в так называемом "формате краевой таблицы" (edge table), которая по существу является двумерным представлением дерева XML. Соответствующий пример приведен в листинге 18.23.

Листинг 18.23. Оператор SELECT с функцией OPENXML, в котором исключена конструкция WITH, и полученные результаты

```
DECLARE @hDoc int
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
  <artist name="Johnny Hartman">
  <song> <name>It Was Almost Like a Song</name></song>
  <song> <name>I See Your Face Before Me</name></song>
  <song> <name>For All We Know</name></song>
  <song> <name>Easy Living</name></song>
  </artist>
  <artist name="Harry Connick, Jr.">
  <song> <name>Sonny Cried</name></song>
  <song> <name>A Nightingale Sang in Berkeley Square</name></song>
  <song> <name>Heavenly</name></song>
  <song> <name>You Didn''t Know Me When</name></song>
  </artist>
</songs>'
SELECT * FROM OPENXML(@hdoc, '/songs/artist/song', 2)
EXEC sp_xml_removedocument @hDoc

(Результаты приведены в сокращенном виде)

id                   parentid             nodetype    localname
-------------------- -------------------- ----------- -----------
4                    2                    1            song
5                    4                    1            name
22                   5                    3            #text
6                    2                    1            song
7                    6                    1            name
23                   7                    3            #text
8                    2                    1            song
9                    8                    1            name
24                   9                    3            #text
10                   2                    1            song
11                   10                   1            name
25                   11                   3            #text
```

```
14          12          1           song
15          14          1           name
26          15          3           #text
16          12          1           song
17          16          1           name
27          17          3           #text
18          12          1           song
19          18          1           name
28          19          3           #text
20          12          1           song
21          20          1           name
29          21          3           #text
```

Вставка данных с помощью функции OPENXML

С учетом того, что OPENXML — функция набора строк, вполне естественным является подход, предусматривающий вставку результатов выполнения оператора SELECT с функцией OPENXML в другую таблицу. Существует целый ряд способов реализации такого подхода. Прежде всего можно выполнить отдельный проход по документу XML для извлечения каждого требуемого фрагмента данных. При этом осуществляется вставка данных с помощью оператора INSERT...SELECT FROM OPENXML для каждой таблицы, в которую необходимо вставить строки, с выборкой отдельных частей документа XML при каждом проходе, как показано в листинге 18.24.

Листинг 18.24. Выполнение выборки с помощью функции OPENXML и последующей вставки, а также полученные результаты

```
USE tempdb
GO
CREATE TABLE Artists
(ArtistId varchar(5),
  Name varchar(30))
GO
CREATE TABLE Songs
(ArtistId varchar(5),
  SongId int,
  Name varchar(50))
GO

DECLARE @hDoc int
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
  <artist id="JHART" name="Johnny Hartman">
  <song id="1" name="It Was Almost Like a Song"/>
  <song id="2" name="I See Your Face Before Me"/>
  <song id="3" name="For All We Know"/>
<song id="4" name="Easy Living"/>
  </artist>
  <artist id="HCONN" name="Harry Connick, Jr.">
  <song id="1" name="Sonny Cried"/>
  <song id="2" name="A Nightingale Sang in Berkeley Square"/>
```

```
  <song id="3" name="Heavenly"/>
  <song id="4" name="You Didn''t Know Me When"/>
  </artist>
</songs>'
INSERT Artists (ArtistId, Name)
SELECT id,name
FROM OPENXML(@hdoc, '/songs/artist', 1)
WITH (id varchar(5) '@id',
      name varchar(30) '@name')

INSERT Songs (ArtistId, SongId, Name)
SELECT artistid, id,name
FROM OPENXML(@hdoc, '/songs/artist/song', 1)
WITH (artistid varchar(5) '../@id',
      id int '@id',
      name varchar(50) '@name')
EXEC sp_xml_removedocument @hDoc
GO
SELECT * FROM Artists
SELECT * FROM Songs
GO
DROP TABLE Artists, Songs

(Результаты)

ArtistId Name
-------- ------------------------------
JHART    Johnny Hartman
HCONN    Harry Connick, Jr.

ArtistId SongId      Name
-------- ----------- ---------------------------------------------
JHART    1            It Was Almost Like a Song
JHART    2            I See Your Face Before Me
JHART    3           For All We Know
JHART    4           Easy Living
HCONN    1           Sonny Cried
HCONN    2           A Nightingale Sang in Berkeley Square
HCONN    3           Heavenly
HCONN    4           You Didn't Know Me When
```

Вполне очевидно, что выполняется отдельный вызов OPENXML применительно к каждой таблице. Таблицы нормализованы, а документ XML нет, поэтому содержимое документа XML разделяется на несколько таблиц. В листинге 18.25 показан еще один способ выполнения той же задачи, который не требует применения нескольких вызовов функции OPENXML.

Листинг 18.25. Еще один способ выполнения выборки с помощью функции OPENXML и последующей вставки, а также полученные результаты

```
USE tempdb
GO
CREATE TABLE Artists
(ArtistId varchar(5),
  Name varchar(30))
```

```
GO
CREATE TABLE Songs
(ArtistId varchar(5),
  SongId int,
  Name varchar(50))
GO
CREATE VIEW ArtistSongs AS
SELECT a.ArtistId,
       a.Name AS ArtistName,
       s.SongId,
       s.Name as SongName
FROM Artists a JOIN Songs s
ON (a.ArtistId=s.ArtistId)
GO
CREATE TRIGGER ArtistSongsInsert ON ArtistSongs INSTEAD OF
    INSERT AS
INSERT Artists
SELECT DISTINCT ArtistId, ArtistName FROM inserted
INSERT Songs
SELECT ArtistId, SongId, SongName FROM inserted
GO

DECLARE @hDoc int
EXEC sp_xml_preparedocument @hDoc output,
'<songs>
  <artist id="JHART" name="Johnny Hartman">
  <song id="1" name="It Was Almost Like a Song"/>
  <song id="2" name="I See Your Face Before Me"/>
  <song id="3" name="For All We Know"/>
  <song id="4" name="Easy Living"/>
  </artist>
  <artist id="HCONN" name="Harry Connick, Jr.">
<song id="1" name="Sonny Cried"/>
  <song id="2" name="A Nightingale Sang in Berkeley Square"/>
  <song id="3" name="Heavenly"/>
  <song id="4" name="You Didn''t Know Me When"/>
  </artist>
</songs>'
INSERT ArtistSongs (ArtistId, ArtistName, SongId, SongName)
SELECT artistid, artistname, songid, songname
FROM OPENXML(@hdoc, '/songs/artist/song', 1)
WITH (artistid varchar(5) '../@id',
      artistname varchar(30) '../@name',
       songid int '@id',
      songname varchar(50) '@name')

EXEC sp_xml_removedocument @hDoc
GO
SELECT * FROM Artists
SELECT * FROM Songs
GO
DROP VIEW ArtistSongs
GO
DROP TABLE Artists, Songs

(Результаты)

ArtistId Name
```

```
-------- ------------------------------
HCONN    Harry Connick, Jr.
JHART     Johnny Hartman

ArtistId SongId      Name
-------- ----------- --------------------------------------------
JHART    1           It Was Almost Like a Song
JHART    2           I See Your Face Before Me
JHART    3           For All We Know
JHART    4           Easy Living
HCONN    1           Sonny Cried
HCONN    2           A Nightingale Sang in Berkeley Square
HCONN    3           Heavenly
HCONN    4           You Didn't Know Me When
```

Данный способ предусматривает использование представления и триггера INSTEAD OF для устранения необходимости двух проходов по документу с помощью функции OPENXML. Для моделирования денормализованной компоновки документа XML используется представление, затем устанавливается триггер INSTEAD OF, позволяющий вставлять данные документа XML в таблицы, лежащие в основе представления. В действительности все операции по разбиению документа XML выполняет триггер, но эти действия осуществляются гораздо более эффективно по сравнению с двукратным вызовом функции OPENXML. При использовании рассматриваемого способа выполняются два прохода по логической вставленной таблице, и содержащиеся в ней столбцы (которые отображаются на представление) разбиваются по двум отдельным таблицам.

Доступ к программе SQL Server по протоколу HTTP

Для того чтобы обеспечить доступ к программе SQL Server по протоколу HTTP, необходимо выполнить настройку виртуального каталога сервера IIS с помощью команды меню Configure IIS Support в каталоге программы SQLXML. Безусловно, можно осуществлять выборку данных XML из программы SQL Server и без установки виртуального каталога (например, с использованием средств ADO или OLE DB), но в данном разделе рассматривается только способ получения данных XML из программы SQL Server по протоколу HTTP.

Настройка конфигурации виртуального каталога позволяет работать со средствами XML программы SQL Server по протоколу HTTP. Виртуальный каталог используется для установления связи между базой данных SQL Server и сегментом URL. Кроме того, виртуальный каталог предоставляет путь перехода от корневого каталога Web-сервера к базе данных, которая определена в программе SQL Server.

Способность программы SQL Server предоставлять доступ к данным по протоколу HTTP (или публиковать данные) обеспечивается благодаря использованию средств SQLISAPI, т.е. с помощью расширения API-интерфейса сервера Internet (Internet Server API — ISAPI), которое входит в поставку программного продукта SQL Server. В состав средств SQLISAPI входят собственные средства доступа OLE DB

программы SQL Server (SQLOLEDB), которые обеспечивают доступ к базе данных, связанной с виртуальным каталогом, и возвращают результаты клиенту.

В клиентских приложениях могут применяться четыре метода запроса данных из программы SQL Server по протоколу HTTP. Эти методы подразделяются на два основных типа – методы, которые в большей степени подходят для обеспечения доступа по закрытой внутренней сети, поскольку характеризуются низким уровнем защиты, и методы, допустимые для использования в открытой сети Internet, поскольку являются безопасными.

Методы, применимые в закрытой внутренней сети

1. Передать шаблон запроса XML в интерфейс SQLISAPI.
2. Отправить в URL строку запроса `SELECT...FOR XML`.

Методы, применимые в открытой сети Internet

3. Определить серверную схему XML в корневом виртуальном каталоге.
4. Определить серверный шаблон запроса XML в корневом виртуальном каталоге.

Методы 1 и 2 предоставляют открытый доступ к базе данных, поэтому могут создавать предпосылки нарушения защиты доступа через общедоступную сеть Internet, но идеально подходят для корпоративных или закрытых внутренних сетей. Как правило, в Web-приложениях используются серверные схемы и шаблоны запросов для предоставления доступа к данным XML для внешнего мира в контролируемой форме.

Настройка виртуального каталога

Перейдя в каталог SQLXML, загрузите утилиту Configure IIS Support с помощью меню Start | Programs. На экране должен появиться список серверов IIS, настройка конфигурации которых выполнена на текущем компьютере. Щелкните на знаке "плюс" слева от имени используемого сервера, чтобы развернуть относящийся к этому серверу узел. (Если требуемый сервер не указан в списке, например, если это – удаленный сервер, щелкните правой кнопкой мыши на узле IIS Virtual Directory Manager и выберите команду Connect, чтобы подключиться к серверу.) Для добавления нового виртуального каталога щелкните правой кнопкой мыши на узле Default Web Site и выберите команду New | Virtual Directory. После этого на экране должно появиться диалоговое окно New Virtual Directory Properties.

Определение имени и пути доступа к виртуальному каталогу

Имя нового виртуального каталога задается в поле ввода Virtual Directory Name. Именно это имя пользователи должны включать в URL, чтобы обратиться к данным, доступ к которым предоставляет виртуальный каталог, поэтому важно, чтобы имя виртуального каталога было описательным. Обычно применяется соглашение, состоящее в том, что виртуальному каталогу присваивается имя,

соответствующее базе данных, на которую ссылается этот каталог. Подготавливаясь к выполнению упражнений, приведенных в оставшейся части данной главы, укажите Northwind в качестве имени нового виртуального каталога.

Параметр Local Path (локальный путь) иногда не используется; тем не менее, этот параметр является обязательным. В обычных приложениях ASP или HTML *локальным* называется путь, в котором находятся сами исходные файлы. В приложениях SQLISAPI соблюдается условие, что каталог, на который указывает локальный путь, не обязательно должен что-либо содержать, но все равно должен существовать на компьютере. Если в операционной системе определены разделы NTFS, необходимо также проследить за тем, чтобы пользователи имели по меньшей мере права доступа для чтения к каталогу, на который указывает локальный путь. Определение в конфигурации того, какая учетная запись пользователя должна применяться для доступа к приложению (в связи с чем для соответствующего пользователя потребуется доступ к локальному каталогу), задается на странице Security указанного выше диалогового окна.

Щелкните на вкладке Security, чтобы выбрать режим аутентификации, который желательно использовать. На этой вкладке можно указать конкретную учетную запись пользователя и определить способ аутентификации Windows Integrated Authentication или Basic (clear text) Authentication. Выберите вариант аутентификации, который в большей степени подходит для используемого способа организации доступа к данным, а для выполнения упражнений, приведенных в данной главе, лучше всего подходит вариант Windows Integrated Authentication.

Затем щелкните на вкладке страницы Data Source. Именно на этой вкладке задается имя экземпляра SQL Server и имя базы данных, на которую ссылается виртуальный каталог. Выберите имя экземпляра SQL Server из списка и укажите Northwind в качестве имени базы данных.

Перейдите к таблице Virtual Names и задайте два виртуальных имени, templates и schemas. Создайте под каталогом Northwind два каталога, с именами Templates и Schemas, чтобы каждому из указанных двух виртуальных каталогов соответствовал собственный локальный каталог. Укажите тип schema в качестве типа для виртуального имени schemas и тип template в качестве типа для виртуального имени templates. Каждое из указанных имен предоставляет путь перехода от URL к файлам в локальном каталоге, соответствующем тому или другому имени. Такие пути перехода будут применяться ниже.

Последняя интересующая нас страница диалогового окна – это страница Settings. Щелкните на вкладке этой страницы и убедитесь в том, что на ней отмечен каждый флажок. Все опции, заданные на странице, должны быть разрешены, чтобы можно было выполнить их проверку в ходе дальнейшего изложения данной главы. Ниже приведены подразделы, в которых содержится краткое описание каждой из опций страницы Settings.

Опция Allow sql=... or template=... or URL queries

Если опция Allow sql=... or template=... or URL queries (Разрешить использование шаблонов в формате sql=... или template=..., а также запросов URL) разрешена, то можно выполнять запросы, передаваемые с помощью URL (в формате

команды GET или POST протокола HTTP) в виде параметров sql= или template=. Запросы URL дают возможность пользователю задавать полный запрос Transact-SQL с помощью URL. Специальные символы заменяются символьными сущностями, но фактически запрос передается на сервер в своем непосредственном виде, а его результаты возвращаются по протоколу HTTP. Обратите внимание на то, что эта опция позволяет пользователям выполнять произвольные запросы по отношению к виртуальному корневому каталогу и базе данных, поэтому не следует разрешать ее использование для какой-либо иной сети, кроме внутренней. А теперь читатель должен выполнить приведенные выше указания и разрешить применение рассматриваемой опции, чтобы можно было ее проверить при выполнении приведенных ниже упражнений.

Выбор указанной выше опции запрещает опцию Allow template=... containing updategrams only (Разрешить использование шаблонов в формате template=..., содержащих только шаблоны обновления), поскольку, если разрешена опция Allow sql=... or template=... or URL queries, то можно всегда отправлять на сервер шаблоны XML с шаблонами обновления. С другой стороны, опция Allow template=... containing updategrams only позволяет передавать с помощью URL шаблоны XML (которые содержат только шаблоны обновления). Поскольку такая опция запрещает задавать запросы SQL и XPath вместе с шаблоном, то при ее использовании степень защищенности данных немного повышается.

Запросы с шаблонами представляют собой метод выборки данных XML из базы данных SQL Server по протоколу HTTP, который намного превосходит другие методы по своей распространенности. Документы XML, в которых хранятся шаблоны запросов (универсальные параметризованные запросы с метками-заполнителями для параметров), находятся на сервере и предоставляют управляемый доступ к основополагающим данным. Результаты выполнения запросов с шаблонами возвращаются пользователю по протоколу HTTP.

Опция Allow XPath

Если опция Allow XPath разрешена, то пользователи могут осуществлять выборку данных из базы данных SQL Server на основе некоторой аннотированной схемы с помощью подмножества языка XPath. Аннотированные схемы хранятся на Web-сервере в виде документов XML и отображают элементы и атрибуты XML на те данные в базе данных, на которые указывает виртуальный каталог. Запросы XPath позволяют пользователю указывать, какие данные, определенные в аннотированной схеме, должны быть ему возвращены.

Опция Allow POST

Протокол HTTP предоставляет возможность передавать данные на Web-сервер с помощью команды POST этого протокола. Если опция Allow POST разрешена, то пользователь может передать шаблон запроса (обычно реализованный в виде скрытого поля формы на Web-странице) на Web-сервер с помощью протокола HTTP. Передача такого запроса влечет за собой его выполнение и возврат результатов клиенту.

Как было указано выше, рассматриваемые методы доступа к данным являются незащищенными, и поэтому обычно применение этих методов ограничивается закрытыми внутренними сетями. Дело в том, что при использовании этих методов злонамеренные пользователи могут формировать свои собственные шаблоны и передавать их по протоколу HTTP для выборки данных, к которым они, как предполагается, не должны иметь доступа, или, что еще хуже, могут вносить изменения в эти данные.

Опция Run on the client

Опция Run on the client определяет, что форматирование XML (например, с помощью средств `FOR XML`) должно осуществляться в одном из клиентских приложений. Разрешив эту опцию, можно передать в одну из клиентских программ функции по преобразованию в код XML наборов строк, соответствующих запросам HTTP.

Опция Expose runtime errors as HTTP error

Опция Expose runtime errors as HTTP error управляет тем, должна ли информация об ошибках при выполнении запроса, заданного в одном из шаблоном XML, возвращаться в заголовке HTTP или в составе сформированного документа XML. Если эта опция разрешена, и попытка выполнения запроса, заданного в одном из шаблонов, завершается неудачей, то возвращается ошибка с номером 512, определяемая протоколом HTTP, а в заголовке HTTP передаются описания ошибок. Если же эта опция запрещена, и выполнение запроса с шаблоном завершается неудачей, то возвращается код 200 успешного завершения HTTP, а описание ошибки возвращается в виде команд обработки внутри самого документа XML.

Разрешите все опции на странице Settings, кроме последних двух, описанных выше, и щелкните на кнопке OK, чтобы создать новый виртуальный каталог.

СОВЕТ. Одной из удобных опций на вкладке Advanced является опция Disable caching of mapping schemas (Отменить кэширование схем преобразования). При обычных условиях выполняется кэширование схем преобразования в памяти после первого обращения к этим схемам, после чего схемы извлекаются из кэша. Но в коде разработки схемы преобразования обычно следует отменять эту опцию, чтобы схема перезагружалась перед каждой ее проверкой.

Запросы URL

Средства, позволяющие запрашивать базу данных SQL Server по протоколу HTTP, реализованы в библиотеке `SQLISn.DLL`, которая представляет собой библиотеку расширения DLL интерфейса ISAPI средств SQLXML. Это расширение DLL известно под общим названием SQLISAPI. В инструментальном средстве Configure IIS Support требуемая библиотека DLL задана по умолчанию, но при настройке конфигурации, на этапе определения виртуального каталога, применяемого для запросов HTTP, можно указать точное имя библиотеки расширения DLL.

Подключившись к серверу IIS (исполняемый файл этого сервера имеет имя `inetinfo.exe`) с помощью отладчика WinDbg до вызова на выполнение каких-либо запросов HTTP, можно обнаружить сообщения `ModLoad`, относящиеся к библиотеке `SQLISn.DLL`, а также к одной или двум библиотекам DLL. Библиотека расширения DLL интерфейса ISAPI не загружается до тех пор, пока не будет выполнен первый вызов одной из функций этой библиотеки.

При выполнении простого запроса URL происходят описанные ниже действия, затрагивающие различные компоненты средств доступа к базе данных по протоколу HTTP.

1. Запрос передается в виде URL из Web-браузера.
2. Запрос проходит в виде запроса `GET` протокола HTTP из браузера в Web-сервер.
3. Виртуальный каталог, указанный в запросе, позволяет определить, какая библиотека расширения DLL должна быть вызвана для обработки полученного URL. Сервер IIS загружает соответствующую библиотеку расширения и передает запрос на обработку в функции этой библиотеки.
4. Функции библиотеки `SQLISn.DLL` (библиотеки расширения DLL интерфейса SQLISAPI) собирают информацию о соединении, идентификации и базе данных из указанной записи виртуального каталога, подключаются к соответствующей программе SQL Server и к базе данных, затем выполняют заданный запрос. Если запрос был передан в виде простого запроса T-SQL, то поступает на сервер в виде языкового события. Если же запрос был передан в виде запроса с шаблоном, то поступает на сервер в качестве события RPC.
5. Сервер собирает затребованные данные и возвращает их в функции библиотеки `SQLISn.DLL`.
6. Библиотека расширения ISAPI возвращает полученные данные на Web-сервер, который, в свою очередь, передает данные в браузер клиента, затребовавший эти данные. Таким образом, выполнение первоначального запроса `GET` протокола HTTP заканчивается.

Использование запросов URL

Простейший способ проверки виртуального каталога, сформированного ранее, состоит в передаче запроса URL с использованием этого виртуального каталога из браузера с поддержкой XML, такого как Internet Explorer. Запросы URL имеют следующую форму:

```
http://localhost/Northwind?sql=SELECT+*+FROM+Customers+FOR+XML+AUTO
↳&root=Customers
```

ПРИМЕЧАНИЕ. Как и все прочие, приведенный выше URL должен быть введен на одной строке. Из-за ограничений по ширине страницы некоторые URL, приведенные в настоящей книге, могут оказаться разбитыми на несколько строк, но в действительности URL всегда следует вводить в виде одной строки.

В данном случае localhost — это имя Web-сервера. Вместо данного имени можно столь же просто задать полностью уточненное по правилам DNS доменное имя, такое как http://www.khen.com. Кроме того, отметим, что Northwind — это имя виртуального каталога, созданного ранее.

Вопросительный знак отделяет URL от его параметров. Многочисленные параметры отделяются друг от друга символами амперсанда. Первый параметр в приведенном выше URL имеет имя sql и задает предназначенный для выполнения запрос. Второй параметр определяет имя корневого элемента документа XML, который должен быть возвращен пользователю. По определению в каждом документе имеется один и только один корневой элемент. Упущение, обусловленное тем, что корневой элемент не задан, приводит к ошибке, если запрос возвращает больше одного элемента верхнего уровня.

Чтобы наглядно ознакомиться с тем, как работают рассматриваемые средства, укажите URL, приведенный в листинге 18.26, в Web-браузере. (Обязательно замените имя localhost правильным именем Web-сервера, если сервер находится на другом компьютере.)

Листинг 18.26. Пример URL и результаты, полученные с его помощью

```
http://localhost/Northwind?sql=SELECT+*+FROM+Customers+WHERE+CustomerId
↳='ALFKI'+FOR+XML+AUTO

(Результаты)

<Customers CustomerID="ALFKI" CompanyName="Alfreds Futterkiste"
ContactName="Maria Anders" ContactTitle="Sales Representative"
Address="Obere Str. 57" City="Berlin" PostalCode="12209 "
Country="Germany" Phone="030-0074321"  Fax="030-0076545" />
```

Обратите внимание на то, что в данном случае спецификация корневого элемента исключена. Рассмотрим, что происходит, если запрос возвращает больше одной строки (листинг 18.27).

Листинг 18.27. URL запроса без указания корневого элемента, который возвращает больше одной строки, и полученные результаты

```
http://localhost/Northwind?sql=SELECT+*+FROM+Customers+WHERE+CustomerId
↳='ALFKI'+OR+CustomerId='ANATR'+FOR+XML+AUTO

(Результаты приведены в сокращенном виде)

The XML page cannot be displayed
Only one top level element is allowed in an XML document.
Line 1, Position 243
```

В данном случае происходит возврат нескольких элементов верхнего уровня (точнее, двух элементов), поэтому в полученном документе XML есть два корневых элемента с именем Customers, а это, безусловно, не разрешено, поскольку не приводит к получению формально правильного кода XML. Чтобы исправить

ситуацию, необходимо задать корневой элемент. Этот элемент может иметь произвольное имя (отличное от имен всех прочих элементов в документе), поскольку он должен заключать в себе только строки, возвращенные конструкцией FOR XML, для того, чтобы полученный документ стал формально правильным. Соответствующий пример приведен в листинге 18.28.

Листинг 18.28. Запрос с указанием корневого элемента и полученные результаты

```
http://localhost/Northwind?sql=SELECT+*+FROM+Customers+WHERE+CustomerId
↳='ALFKI'+OR+CustomerId='ANATR'+FOR+XML+AUTO&root=CustomerList

(Результаты)

<?xml version="1.0" encoding="utf-8" ?>
<CustomerList>
  <Customers CustomerID="ALFKI" CompanyName="Alfreds Futterkiste"
    ContactName="Maria Anders" ContactTitle="Sales Representative"
    Address="Obere Str. 57" City="Berlin" PostalCode="12209"
    Country="Germany" Phone="030-0074321" Fax="030-0076545" />
  <Customers CustomerID="ANATR" CompanyName=
    "Ana Trujillo Emparedados y helados" ContactName="Ana Trujillo"
    ContactTitle="Owner" Address="Avda. de la Constitucion 2222"
    City="Mexico D.F." PostalCode="05021" Country="Mexico"
    Phone="(5) 555-4729" Fax="(5) 555-3745" />
</CustomerList>
```

Можно также задать имя корневого элемента непосредственно в составе параметра sql, как показано в листинге 18.29.

Листинг 18.29. Способ задания имени корневого элемента в составе параметра sql и полученные результаты

```
http://localhost/Northwind?sql=SELECT+'<CustomerList>';SELECT+*+FROM
↳+Customers+WHERE+CustomerId='ALFKI'+OR+CustomerId='ANATR'+FOR+XML
↳+AUTO;SELECT+'</CustomerList>';

(Результаты отформатированы)
<CustomerList>
  <Customers CustomerID="ALFKI" CompanyName="Alfreds Futterkiste"
     ContactName="Maria Anders" ContactTitle="Sales Representative"
    Address="Obere Str. 57" City="Berlin" PostalCode="12209"
    Country="Germany" Phone="030-0074321" Fax="030-0076545" />
  <Customers CustomerID="ANATR" CompanyName=
     "Ana Trujillo Emparedados y helados" ContactName="Ana Trujillo"
    ContactTitle="Owner" Address="Avda. de la Constitucion 2222"
    City="Mexico D.F." PostalCode="05021" Country="Mexico"
    Phone="(5) 555-4729" Fax="(5) 555-3745" />
</CustomerList>
```

Параметр sql этого URL фактически содержит три запроса. Первый запрос формирует открывающий дескриптор для корневого элемента. Второй представляет собой сам запрос, а третий вырабатывает закрывающий дескриптор для корневого элемента. Отдельные запросы разделены точками с запятой.

Вполне очевидно, что конструкция FOR XML возвращает фрагменты документа XML, поэтому для создания формально правильного документа необходимо предоставить корневой элемент.

Специальные символы

Некоторые символы, которые являются вполне допустимыми в языке Transact-SQL, могут вызвать проблемы в запросах URL, поскольку имеют в URL специальные значения. Читатель должен был уже заметить, что для обозначения символа пробела в URL используется знак плюса "+". Очевидно, что тем самым исключается возможность применять сам знак "+" в запросе. Вместо этого необходимо закодировать в запросе URL символы, которые имеют особый смысл, для того, чтобы средства SQLISAPI могли правильно преобразовать их перед передачей запроса в программу SQL Server. Кодирование с помощью специального символа сводится к заданию знака процента "%", за которым следует значение кода ASCII символа в шестнадцатеричном представлении. В табл. 18.3 перечислены специальные символы, распознаваемые средствами SQLISAPI, и соответствующие им значения.

Таблица 18.3. Специальные символы и их шестнадцатеричные значения

Символ	Шестнадцатеричное значение
+	2B
&	26
?	3F
%	25
/	2F
#	23

Ниже приведен запрос URL, который показывает, как закодировать специальные символы.

```
http://localhost/Northwind?sql=SELECT+'<CustomerList>';SELECT+*+FROM
↳+Customers+WHERE+CustomerId+LIKE+'A%25'+FOR+XML+AUTO;SELECT
↳+'</CustomerList>';
```

В этом запросе задан предикат LIKE, который включает закодированный знак процента "%", представляющий собой подстановочный символ языка Transact-SQL. Знак процента имеет в коде ASCII шестнадцатеричное значение 25 (или десятичное значение 37), поэтому знак процента закодирован как %25.

Таблицы стилей

Кроме параметров sql и root, запрос URL может также включать параметр xsl, позволяющий определить таблицу стилей XML, применяемую для преобразования документа XML, возвращенного запросом, в другой формат. Наиболее широко применяемое назначение этого средства состоит в преобразовании документа в код HTML. Это позволяет просматривать документ с использованием

браузеров, не поддерживающих язык XML, и предоставляет пользователю больший контроль над отображением документа в тех браузерах, которые поддерживают XML. Ниже приведен запрос URL, включающий параметр xsl.

```
http://localhost/Northwind?sql=SELECT+CustomerId,+CompanyName+FROM
+Customers+FOR+XML+AUTO&root=CustomerList&xsl=CustomerList.xsl
```

В листинге 18.30 показана таблица стилей XSL, на которую ссылается документ, а в табл. 18.4 приведены выходные данные, сформированные с помощью этой таблицы стилей.

Листинг 18.30. Таблица стилей XSL

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE border="1">
          <TR>
            <TD><B>Customer ID</B></TD>
            <TD><B>Company Name</B></TD>
          </TR>
          <xsl:for-each select="CustomerList/Customers">
             <TR>
               <TD>
              <xsl:value-of select="@CustomerId"/>
               </TD>
               <TD>
              <xsl:value-of select="@CompanyName"/>
               </TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Таблица 18.4. Выходные данные, полученные с помощью таблицы стилей, приведенной в листинге 18.30

Поле Customer ID	Поле Company Name
ALFKI	Alfreds Futterkiste
ANATR	Ana Trujillo Emparedados y helados
ANTON	Antonio Moreno TaquerAa
AROUT	Around the Horn
BERGS	Berglunds snabbkÃ¶p

Окончание табл. 18.4

Поле Customer ID	Поле Company Name
BLAUS	Blauer See Delikatessen
BLONP	Blondesddsl pÃ··re et fils
WARTH	Wartian Herkku
WELLI	Wellington Importadora
WHITC	White Clover Markets
WILMK	Wilman Kala
WOLZA	Wolski Zajazd

Тип информационного наполнения

По умолчанию средства SQLISAPI возвращают результаты запроса URL с указанием соответствующего типа данных в заголовке документа, для того чтобы браузер правильно отформатировал данные документа перед выводом на экран. Если в запросе используется конструкция FOR XML, то в качестве типа данных должен быть указан тип `text/xml`, если только не задан атрибут `xsl`, который определяет таблицу стилей, применяемую для преобразования документа XML в код HTML. В последнем случае возвращается тип `text/html`.

Кроме того, тип данных может быть задан принудительно с использованием параметра `contenttype` запроса URL, как показано ниже.

```
http://localhost/Northwind?sql=SELECT+CustomerId,+CompanyName+FROM
↳+Customers+FOR+XML+AUTO&root=CustomerList&xsl=CustomerList.xsl
↳&contenttype=text/xml
```

В данном случае указана таблица стилей из предыдущего примера, применяемая для преобразования информационного наполнения в тип, который по умолчанию обозначается как `text/html`. Затем это заданное по умолчанию значение переопределяется путем указания параметра `contenttype` со значением `text/xml`. В результате формируется документ XML, содержащий преобразованный результирующий набор, как показано в листинге 18.31.

Листинг 18.31. Преобразованный результирующий набор

```
<HTML>
  <BODY>
    <TABLE border="1">
       <TR>
         <TD>
           <B>Customer ID</B>
         </TD>
         <TD>
          <B>Company Name</B>
         </TD>
       </TR>
       <TR>
         <TD>ALFKI</TD>
```

```
        <TD>Alfreds Futterkiste</TD>
       </TR>
       <TR>
         <TD>ANATR</TD>
        <TD>Ana Trujillo Emparedados y helados</TD>
       </TR>
       <TR>
         <TD>WILMK</TD>
         <TD>Wilman Kala</TD>
       </TR>
       <TR>
         <TD>WOLZA</TD>
        <TD>Wolski Zajazd</TD>
       </TR>
     </TABLE>
  </BODY>
</HTML>
```

При этом, даже если документ состоит из формально правильного кода HTML, то выводится на экран как документ XML, поскольку значение типа информационного наполнения этого документа принудительно задано как `text/xml`.

Получение результатов в коде, отличном от XML

Возможность указывать тип информационного наполнения становится особенно удобной при работе с фрагментами XML в браузере, поддерживающем XML. Как было указано выше, вызов на выполнение запроса `FOR XML` без определения корневого элемента `root` приводит к возникновению ошибки. Тем не менее такую ситуацию можно обойти, предусмотрев принудительное введение типа информационного наполнения в код HTML следующим образом:

```
http://localhost/Northwind?sql=SELECT+*+FROM+Customers+WHERE+CustomerId
↳='ALFKI'+OR+CustomerId='ANATR'+FOR+XML+AUTO&contenttype=text/html
```

После развертывания в браузере данных, полученных с помощью этого URL, на экране, скорее всего, появится пустая страница, поскольку большинство браузеров игнорирует дескрипторы, которые не предусмотрены для использования в этих браузерах. Однако после просмотра исходного кода Web-страницы обнаружится, что возвращен именно тот фрагмент XML, которого и следовало ожидать. Такой подход может оказаться удобным в тех ситуациях, когда обмен данными со средствами SQLISAPI по протоколу HTTP осуществляется вне программы браузера, например, в каком-то приложении. Это позволяет передать фрагмент XML в клиентскую программу, затем с помощью алгоритмов, реализованных в клиентской программе, дополнить фрагмент корневым элементом и/или выполнить дальнейшую обработку XML.

Средства SQLISAPI предоставляют также возможность исключить конструкцию `FOR XML`, чтобы получить один столбец с данными из таблицы, представления или функции со значением в виде таблицы в формате простого текстового потока, как показано в листинге 18.32.

Листинг 18.32. Запрос без конструкции FOR XML и полученные результаты

```
http://localhost/Northwind?sql=SELECT+CAST(CustomerId+AS+char(10))+AS
↳+CustomerId+FROM+Customers+ORDER+BY+CustomerId&contenttype=text/html

(Результаты)
ALFKI ANATR ANTON AROUT BERGS BLAUS BLONP BOLID BONAP BOTTM BSBEV
CACTU CENTC CHOPS COMMI CONSH DRACD DUMON EASTC ERNSH FAMIA FISSA
FOLIG FOLKO FRANK FRANR FRANS FURIB GALED GODOS GOURL GREAL GROSR
HANAR HILAA HUNGC HUNGO ISLAT KOENE LACOR LAMAI LAUGB LAZYK LEHMS
LETSS LILAS LINOD LONEP MAGAA MAISD MEREP MORGK NORTS OCEAN OLDWO
OTTIK PARIS PERIC PICCO PRINI QUEDE QUEEN QUICK RANCH RATTC REGGC
RICAR RICSU ROMEY SANTG SAVEA SEVES SIMOB SPECD SPLIR SUPRD THEBI
THECR TOMSP TORTU TRADH TRAIH VAFFE VICTE VINET WANDK WARTH WELLI
WHITC WILMK WOLZA
```

Обратите внимание на то, что средства SQLISAPI не поддерживают возможность получения таким образом многостолбцовых результатов. Несмотря на сказанное выше, следует знать, что запросы без конструкции FOR XML представляют собой удобный способ быстрого формирования простых списков данных.

Хранимые процедуры

Запросы URL могут использоваться для вызова на выполнение хранимых процедур, наряду с запросами Transact-SQL других типов. Безусловно, если результаты, возвращаемые процедурой, предназначены для обработки в качестве кода XML в браузере или клиентском коде, то процедура должна возвращать свои результаты с помощью конструкции FOR XML. Примером хранимой процедуры такого типа может служить процедура, приведенная в листинге 18.33.

Листинг 18.33. Хранимая процедура с конструкцией FOR XML

```
CREATE PROC ListCustomersXML
@CustomerId varchar(10)='%',
@CompanyName varchar(80)='%'
AS
SELECT CustomerId, CompanyName
FROM Customers
WHERE CustomerId LIKE @CustomerId
AND CompanyName LIKE @CompanyName
FOR XML AUTO
```

После создания процедуры, которая правильно возвращает результаты в формате XML, появляется возможность вызывать эту процедуру из запроса URL с помощью команды EXEC языка Transact-SQL. В листинге 18.34 приведен пример запроса URL, в котором вызывается хранимая процедура с использованием команды EXEC.

Листинг 18.34. Вызов хранимой процедуры с помощью команды EXEC и полученные результаты

```
http://localhost/Northwind?sql=EXEC+ListCustomersXML+@CustomerId
↳='A%25',@CompanyName='An%25'&root=CustomerList
```

```
(Результаты)

<?xml version="1.0" encoding="utf-8" ?>
<CustomerList>
  <Customers CustomerId="ANATR" CompanyName="Ana Trujillo
      Emparedados y helados" />
  <Customers CustomerId="ANTON" CompanyName="Antonio Moreno
      Taqueria" />
</CustomerList>
```

Обратите внимание на то, что в данном коде подстановочный символ "%" языка Transact-SQL задается с помощью его закодированного эквивалента, %25. Такая замена необходима, как уже было сказано выше, поскольку символ "%" имеет в запросах URL особый смысл.

СОВЕТ. Для вызова хранимой процедуры из запроса URL можно также использовать синтаксическую конструкцию `CALL` интерфейса ODBC. При этом процедуры вызываются на выполнение на сервере с помощью события RPC; такой способ вызова процедур обычно является более быстрым и эффективным по сравнению с обычными языковыми событиями T-SQL. Отметим, что на Web-узлах с большим объемом обрабатываемых запросов становится весьма заметным даже небольшое повышение производительности выполнения отдельных запросов.

Ниже приведен ряд запросов URL, в которых используется синтаксическая конструкция `CALL` интерфейса ODBC.

```
http://localhost/Northwind?sql={CALL+ListCustomersXML}+&root=
=CustomerList
```

```
http://localhost/Northwind?sql={CALL+ListCustomersXML('ALFKI')}+&root=
=CustomerList
```

Если один из этих URL будет передан из Web-браузера в то время, как работает трассировка Profiler, которая включает событие `RPC:Starting`, должна быть обнаружена активизация события `RPC:Starting`, относящегося к соответствующей процедуре. Это означает, что вызов процедуры происходит с помощью механизма обработки RPC, более эффективного по сравнению с механизмом обработки языковых событий.

Дополнительная информация о том, как выполняются вызовы RPC из службы SQLXML, приведена ниже, в разделе "Запросы с шаблонами".

Запросы с шаблонами

Более безопасный и широко используемый способ выборки данных по протоколу HTTP состоит в применении серверных шаблонов XML, которые инкапсулируют запросы Transact-SQL. Такие шаблоны хранятся на Web-сервере, а ссылка на них осуществляется с помощью виртуального имени, поэтому конечные пользователи не имеют возможности видеть исходный код. Шаблоны представляют собой документы XML, основанные на использовании пространства имен XML-SQL,

и действуют как механизм преобразования URL в запрос, который может обрабатываться программой SQL Server. Как и при применении простых запросов URL, результаты запросов с шаблонами возвращаются либо в коде XML, либо в коде HTML.

Простой пример запроса XML с шаблоном приведен в листинге 18.35.

Листинг 18.35. Запрос XML с шаблоном

```
<?xml version='1.0' ?>
<CustomerList xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    SELECT CustomerId, CompanyName
    FROM Customers
    FOR XML AUTO
  </sql:query>
</CustomerList>
```

Обратите внимание на использование префикса пространства имен `sql` в дескрипторе `query` самого запроса. Возможность применения этого пространства имен обеспечивается благодаря наличию ссылки на данное пространство имен во второй строке шаблона (выделена полужирным шрифтом).

В данном случае осуществляется просто возврат двух столбцов из таблицы `Customers` базы данных `Northwind`, как уже было сделано в нескольких примерах настоящей главы. Для получения данных в формате XML включена конструкция `FOR XML AUTO`. Соответствующий URL, в котором используется рассматриваемый шаблон запроса, наряду с возвращаемыми данными, показан в листинге 18.36.

Листинг 18.36. Вызов шаблона запроса и полученные результаты

```
http://localhost/Northwind/templates/CustomerList.XML

(Результаты приведены в сокращенном виде)

<?xml version="1.0" ?>
<CustomerList xmlns:sql="urn:schemas-microsoft-com:xml-sql">
<Customers CustomerId="ALFKI" CompanyName=
    "Alfreds Futterkiste" />
  <Customers CustomerId="VAFFE" CompanyName="Vaffeljernet" />
  <Customers CustomerId="VICTE" CompanyName=
     "Victuailles en stock" />
  <Customers CustomerId="VINET" CompanyName=
    "Vins et alcools Chevalier" />
  <Customers CustomerId="WARTH" CompanyName="Wartian Herkku" />
  <Customers CustomerId="WELLI" CompanyName=
    "Wellington Importadora" />
  <Customers CustomerId="WHITC" CompanyName=
     "White Clover Markets" />
  <Customers CustomerId="WILMK" CompanyName="Wilman Kala" />
  <Customers CustomerId="WOLZA" CompanyName="Wolski Zajazd" />
</CustomerList>
```

Обратите внимание на то, что в листинге 18.36 используется имя виртуального каталога `templates`, созданного ранее в виртуальном каталоге `Northwind`.

Параметризованные шаблоны

Предусмотрена также возможность создавать параметризованные шаблоны XML запросов, которые позволяют пользователю задавать параметры запроса во время вызова запроса на выполнение. Параметры определяются в заголовке шаблона, содержащемся в элементе шаблона `sql:header`. Каждый параметр определяется с помощью дескриптора `sql:param` и может включать необязательное значение, применяемое по умолчанию. Соответствующий пример параметризованного шаблона приведен в листинге 18.37.

Листинг 18.37. Пример параметризованного шаблона

```
<?xml version='1.0' ?>
<CustomerList xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:header>
    <sql:param name='CustomerId'>%</sql:param>
  </sql:header>
  <sql:query>
    SELECT CustomerId, CompanyName
    FROM Customers
    WHERE CustomerId LIKE @CustomerId
    FOR XML AUTO
  </sql:query>
</CustomerList>
```

Обратите внимание на то, как используется элемент `sql:param` для определения параметра. В данном случае параметру присваивается заданное по умолчанию значение "%", поскольку параметр применяется в предикате `LIKE` запроса. Это означает, что если значение параметра не будет задано, то сформируется список всех заказчиков.

Следует отметить, что средства SQLISAPI способны самостоятельно принять решение о передаче запроса с шаблоном на сервер в виде вызова RPC, если заданы параметры запроса. Средствами SQLISAPI выполняется привязка заданных в шаблоне параметров к параметрам RPC и передача запроса в программу SQL Server с помощью вызовов API-интерфейса RPC. Такой способ выполнения запроса является более эффективным по сравнению с использованием языковых событий T-SQL и должен привести к повышению производительности, особенно в системах с высокой пропускной способностью.

Пример URL, в котором задан запрос с параметризованным шаблоном, наряду с полученными результатами, приведен в листинге 18.38.

Листинг 18.38. Вызов запроса с параметризованным шаблоном и полученные результаты

```
http://localhost/Northwind/Templates/CustomerList2.XML?CustomerId=A%25

(Результаты)

<?xml version="1.0" ?>
<CustomerList xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <Customers CustomerId="ALFKI" CompanyName=
```

```
    "Alfreds Futterkiste" />
  <Customers CustomerId="ANATR" CompanyName=
    "Ana Trujillo Emparedados y helados" />
  <Customers CustomerId="ANTON" CompanyName=
    "Antonio Moreno Taqueria" />
  <Customers CustomerId="AROUT" CompanyName="Around the Horn" />
</CustomerList>
```

Таблицы стилей

Как и при использовании обычных запросов URL, может быть задана таблица стилей, предназначенная для применения к запросу с шаблоном. Таблица стилей может быть указана в самом шаблоне или в URL, который обращается к этому шаблону. Ниже приведен пример URL, в котором предусматривается применение таблицы стилей к запросу с шаблоном.

```
http://localhost/Northwind/Templates/CustomerList3.XML?xsl
=Templates/CustomerList3.xsl&contenttype=text/html
```

Обратите внимание на использование параметра `contenttype`, который принудительно определяет требование, что выходные данные должны рассматриваться как код HTML (параметр выделен полужирным шрифтом). Параметр `contenttype` применяется в связи с тем, что данная таблица стилей (о чем известно пользователю) преобразует код XML, возвращаемый программой SQL Server, в формат таблицы HTML.

Кроме того, в URL задано обозначение относительного пути от виртуального каталога к таблице стилей, поскольку при выполнении подобных запросов не осуществляется автоматический поиск таблиц стилей в каталоге `Templates`, даже если соответствующий документ XML находится в этом каталоге. Спецификации пути для запроса с шаблоном и его параметров отделены друг от друга.

Как уже было сказано выше, в пространстве имен XML-SQL предоставляется также возможность задавать таблицу стилей в самом шаблоне. В листинге 18.39 приведен шаблон, в котором задана таблица стилей.

Листинг 18.39. Шаблон с таблицей стилей

```
<?xml version='1.0' ?>
<CustomerList xmlns:sql='urn:schemas-microsoft-com:xml-sql'
    sql:xsl='CustomerList3.xsl'>
  <sql:query>
    SELECT CustomerId, CompanyName
    FROM Customers
    FOR XML AUTO
  </sql:query>
</CustomerList>
```

Таблица стилей, указанная в рассматриваемом шаблоне (листинг 18.39), приведена в листинге 18.40.

Листинг 18.40. Таблица стилей шаблона, приведенного в листинге 18.39

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <TABLE border="1">
          <TR>
            <TD><I>Customer ID</I></TD>
            <TD><I>Company Name</I></TD>
          </TR>
          <xsl:for-each select="CustomerList/Customers">
            <TR>
              <TD><B>
              <xsl:value-of select="@CustomerId"/>
              </B></TD>
              <TD>
              <xsl:value-of select="@CompanyName"/>
              </TD>
            </TR>
          </xsl:for-each>
        </TABLE>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

В листинге 18.41 показан URL, в котором используются шаблон и таблица стилей, приведенные в листингах 18.39 и 18.40, а в табл. 18.5 показаны полученные результаты.

Листинг 18.41. URL, в котором используются шаблон и таблица стилей

```
http://localhost/Northwind/Templates/CustomerList4.XML?contenttype=
↳text/html
```

Таблица 18.5. Результаты, полученные с помощью URL, приведенного в листинге 18.41 (в сокращенном виде)

Поле Customer ID	Поле Company Name
ALFKI	Alfreds Futterkiste
ANATR	Ana Trujillo Emparedados y helados
ANTON	Antonio Moreno TaquerAa
AROUT	Around the Horn
VICTE	Victuailles en stock
VINET	Vins et alcools Chevalier
WARTH	Wartian Herkku

Окончание табл. 18.5

Поле Customer ID	Поле Company Name
WELLI	Wellington Importadora
WHITC	White Clover Markets
WILMK	Wilman Kala
WOLZA	Wolski Zajazd

Обратите внимание на то, что в данном случае снова задан параметр `contenttype`, позволяющий принудительно указать, что выходные данные должны рассматриваться как код HTML. Такое требование является обязательным, поскольку браузеры с поддержкой XML, такие как Internet Explorer, автоматически обрабатывают выходные данные, возвращаемые шаблонами XML, как данные в формате `text/xml`. Поскольку возвращаемый шаблоном код HTML вполне может рассматриваться как формально правильный код XML, браузер не сможет определить, что этот код должен использоваться для вывода данных на экран в качестве кода HTML, если не получит указание об этом. Именно для этой цели предназначена спецификация `contenttype`. Такая спецификация вынуждает браузер подготавливать для вывода на экран результаты выполнения запроса с шаблоном так, как если бы это был обычный документ HTML.

СОВЕТ. При разработке шаблонов XML и аналогичных документов, которые затем необходимо проверять в Web-браузере, иногда приходится сталкиваться с проблемами, обусловленными тем, что браузер использует не новую версию документа, а извлекает из кэша старую версию даже после щелчка на кнопке Refresh или нажатия клавиши обновления (<F5>). В программе Internet Explorer можно нажать клавиши <Ctrl+F5>, чтобы вынудить браузер выполнить полную перезагрузку документа, даже если сам браузер не имеет информации о том, что такая перезагрузка действительно требуется. Обычно этот прием позволяет устранить проблемы, связанные с тем, что старая версия продолжает храниться в памяти браузера, после того как на диск записана новая версия документа.

Кроме того, можно отменить кэширование шаблонов для данного конкретного виртуального каталога, выбрав опцию Disable caching of templates на странице Advanced диалогового окна Properties, соответствующего рассматриваемому виртуальному каталогу. Автор почти всегда отменяет все кэширование, проводя разработку шаблонов и других документов XML.

Применение таблиц стилей в клиентской программе

Если в клиентской программе предусмотрена поддержка XML, то применение таблиц стилей к результатам запросов с шаблонами может быть также предусмотрено в клиентской программе. Это позволяет передать определенный объем работы из серверной программы в клиентскую. Но при такой организации работы требуется отдельный цикл обмена данными с сервером для загрузки таблицы стилей

в клиентскую программу. Если же клиентская программа не поддерживает XML, то таблица стилей будет проигнорирована. Это означает, что указанный подход является более приемлемым в тех ситуациях, когда точно известно, что клиентские программы поддерживают XML, например, в условиях эксплуатации приложений для закрытой внутренней сети или корпоративных приложений. Преобразование с помощью клиентской таблицы стилей предусмотрено в шаблоне, приведенном в листинге 18.42.

Листинг 18.42. Пример применения таблицы стилей в клиентской программе

```
<?xml version='1.0' ?>
<?xml-stylesheet type='text/xsl' href='CustomerList3.xsl'?>
<CustomerList xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:query>
    SELECT CustomerId, CompanyName
    FROM Customers
    FOR XML AUTO
  </sql:query>
</CustomerList>
```

Здесь заслуживает внимания спецификация `xml-stylesheet` в верхней части документа (выделена полужирным шрифтом). Эта спецификация служит для процессора XML клиентской программы указанием на то, что необходимо загрузить таблицу стилей, указанную в атрибуте `href`, и применить ее к документу XML, выводимому с помощью шаблона. Отметим, что URL, в котором вызывается этот шаблон, показан в листинге 18.43, а полученные результаты — в табл. 18.6.

Листинг 18.43. Пример использования URL с шаблоном, приведенным в листинге 18.42

```
http://localhost/Northwind/Templates/CustomerList5.XML?contenttype=
↳text/html
```

Таблица 18.6. Результаты, полученные с помощью URL, приведенного в листинге 18.43 (в сокращенном виде)

Поле Customer ID	Поле Company Name
ALFKI	Alfreds Futterkiste
ANATR	Ana Trujillo Emparedados y helados
ANTON	Antonio Moreno TaquerAa
AROUT	Around the Horn
VICTE	Victuailles en stock
VINET	Vins et alcools Chevalier
WARTH	Wartian Herkku
WELLI	Wellington Importadora
WHITC	White Clover Markets
WILMK	Wilman Kala
WOLZA	Wolski Zajazd

Клиентские шаблоны

Как уже было указано выше, гораздо более широко распространенный (и безопасный) способ применения шаблонов состоит в том, чтобы хранить шаблоны на Web-сервере и предоставлять пользователям возможность вызывать эти шаблоны с помощью виртуальных имен. Несмотря на сказанное выше, при определенных обстоятельствах очень перспективным становится подход, позволяющий пользователям задавать шаблоны в клиентской программе. Если предусмотрен способ организации работы, при использовании которого шаблоны задаются в коде HTML, передаваемом из клиентского приложения, или определяются в самом клиентском приложении, то устраняется необходимость заранее создавать шаблоны или определять виртуальные имена, которые ссылаются на шаблоны. Такой подход, безусловно, проще с точки зрения административной организации работы, но может стать причиной нарушения защиты при использовании в общедоступной сети Internet, поскольку позволяет клиентам самостоятельно задавать код, предназначенный для выполнения в программе SQL Server. По-видимому, следует ограничивать рамки использования такого метода закрытыми внутренними и корпоративными сетями.

В листинге 18.44 приведена Web-страница, в которой внедрен клиентский шаблон.

Листинг 18.44. Web-страница с клиентским шаблоном

```
<HTML>
  <HEAD>
    <TITLE>Customer List</TITLE>
  </HEAD>
  <BODY>
    <FORM action='http://localhost/Northwind' method='POST'>
      <B>Customer ID Number</B>
      <INPUT type=text name=CustomerId value='AAAAA'>
      <INPUT type=hidden name=xsl value=Templates/CustomerList2.xsl>
      <INPUT type=hidden name=template value='
      <CustomerList xmlns:sql="urn:schemas-microsoft-com:xml-sql">
         <sql:header>
          <sql:param name="CustomerId">%</sql:param>
         </sql:header>
        <sql:query>
          SELECT CompanyName, ContactName
          FROM Customers
          WHERE CustomerId LIKE @CustomerId
          FOR XML AUTO
         </sql:query>
       </CustomerList>
      '>
      <P><input type='submit'>
    </FORM>
  </BODY>
</HTML>
```

Клиентский шаблон (выделен полужирным шрифтом) внедрен в Web-страницу как скрытое поле. После того как эта страница откроется в Web-браузере, должны появиться поле ввода идентификатора заказчика Customer ID и кнопка передачи запроса на сервер (кнопка Submit Query). После ввода идентификатора заказчика (или маски) и щелчка на кнопке Submit Query на Web-сервер передается шаблон. Затем средства SQLISAPI извлекают запрос, содержащийся в шаблоне, и выполняют его применительно к базе данных `Northwind` программы SQL Server (поскольку в запросе указан виртуальный каталог, соответствующий этой базе данных). После этого к полученным результатам применяется таблица стилей `CustomerList2.xsl`, которая преобразует в код HTML документ XML, возвращенный программой SQL Server, и полученные результаты возвращаются в клиентскую программу. Форма ввода запроса, которая отображается в клиентской программе, приведена на рис. 18.1, а полученные результаты показаны в табл. 18.7.

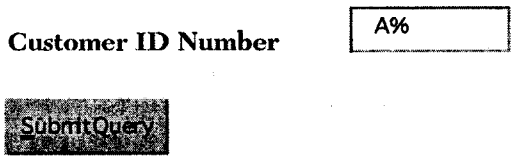


Рис. 18.1. Форма ввода запроса, полученная после выполнения кода, приведенного в листинге 18.44

Таблица 18.7. Результаты, полученные после отправки на сервер запроса с помощью формы, показанной на рис. 18.1

Поле `Company Name`	Поле `Contact Name`
Alfreds Futterkiste	Maria Anders
Ana Trujillo Emparedados y helados	Ana Trujillo
Antonio Moreno TaquerAa	Antonio Moreno
Around the Horn	Thomas Hardy

Как и в при использовании серверных шаблонов, клиентские шаблоны передаются в программу SQL Server с помощью интерфейса RPC.

Схемы отображения

Схемы XML представляют собой документы XML, которые определяют типы данных, разрешенные для использования в других документах XML. Схемы XML предназначены для замены определений DTD, которые первоначально предназначались для этой цели; такие схемы являются более удобными в использовании и более гибкими, поскольку сами представляют собой документы XML.

Кроме того, по своему характеру схемы определяют форматы обмена документами. Эти схемы позволяют указать, что может и не может содержать тот или

иной документ, поэтому компании, желающие обмениваться данными в формате XML, должны предварительно согласовать друг с другом какое-то общее определение схемы. Схемы XML позволяют обеспечить бесперебойный обмен данными друг с другом даже компаниям, характеризующимся разными деловыми потребностями и использующим разный подход к организации работы.

Схема отображения представляет собой схему специального типа, предназначенную для прямого и обратного преобразования данных из формата документа XML в формат реляционной таблицы. Схема отображения может использоваться для создания в коде XML представления таблицы SQL Server. В этом смысле схема отображения аналогична объекту представления SQL Server, который возвращает представление основополагающего объекта таблицы или представления SQL Server в коде XML.

К тому времени, как началась поставка версии SQL Server 2000, работа над окончательной редакцией стандарта XML Schema все еще продолжалась. Поэтому в тот период компания Microsoft, наряду с несколькими другими компаниями, предложила использовать подмножество синтаксиса XML-Data консорциума W3C для определения схем, предназначенных для обмена документами. Первоначально поддержка схем XML в программе SQL Server была основана на спецификации XDR (XML-Data Reduced – сокращенная версия XML-Data), которая представляет собой подмножество спецификации XML-Data, применимое для определения схем. В дальнейшем состоялся выпуск окончательной редакции стандарта XML Schema, и средства SQLXML были доработаны в целях поддержки этого стандарта. Теперь преимущественным способом создания схем, предназначенных для использования в сочетании со средствами SQLXML, является способ, основанный на стандарте XML Schema. Этот способ является более гибким и предоставляет больший спектр возможностей по сравнению с первоначальной поддержкой схем XDR средствами SQLXML. Описание поддержки спецификаций XDR и XML Schema в средствах SQLXML приведено в следующих двух разделах.

Схемы отображения XDR

Начнем изучение схем отображения XDR с примера (листинг 18.45).

Листинг 18.45. Пример применения схемы отображения XDR

```
<?xml version="1.0"?>
<Schema name="NorthwindProducts"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">

  <ElementType name="Description" dt:type="string"/>
  <ElementType name="Price" dt:type="fixed.19.4"/>

  <ElementType name="Product" model="closed">
    <AttributeType name="ProductCode" dt:type="string"/>
    <attribute type="ProductCode" required="yes"/>
    <element type="Description" minOccurs="1" maxOccurs="1"/>
     <element type="Price" minOccurs="1" maxOccurs="1"/>
```

```
  </ElementType>

  <ElementType name="Category" model="closed">
     <AttributeType name="CategoryID" dt:type="string"/>
    <AttributeType name="CategoryName" dt:type="string"/>
     <attribute type="CategoryID" required="yes"/>
     <attribute type="CategoryName" required="yes"/>
    <element type="Product" minOccurs="1" maxOccurs="*"/>
  </ElementType>

  <ElementType name="Catalog" model="closed">
    <element type="Category" minOccurs="1" maxOccurs="1"/>
  </ElementType>

</Schema>
```

Эта схема определяет, как должен выглядеть каталог продукции. (В данной главе используются примеры таблиц и данных из базы данных Northwind.) В рассматриваемой схеме для определения допустимых типов данных элементов и атрибутов в документе применяется пространство имен datatypes (обозначено полужирным шрифтом). Каждый префикс dt: в листинге 18.45 представляет собой ссылку на пространство имен datatypes. Таким образом, в рассматриваемой схеме используется закрытая модель данных, а это гарантирует, что в любом документе, основанном на данной схеме, будет разрешено применять только элементы, заданные в ней.

В листинге 18.46 показан документ XML, в котором используется схема ProductCat.xdr.

Листинг 18.46. Документ XML, в котором используется схема ProductCat.xdr

```
<?xml version="1.0"?>
<Catalog xmlns=
  "x-schema:http://localhost/ProductsCat.xdr">
  <Category CategoryID="1" CategoryName="Beverages">
     <Product ProductCode="1">
      <Description>Chai</Description>
       <Price>18</Price>
     </Product>
     <Product ProductCode="2">
      <Description>Chang</Description>
       <Price>19</Price>
     </Product>
  </Category>
  <Category CategoryID="2" CategoryName="Condiments">
     <Product ProductCode="3">
      <Description>Aniseed Syrup</Description>
       <Price>10</Price>
     </Product>
  </Category>
</Catalog>
```

Если вы скопируете оба эти файла (см. листинги 18.45 и 18.46) в корневой каталог Web-сервера и введете в браузере следующий URL:

```
http://localhost/ProductsCat.xml
```

то на экране браузера появятся такие выходные данные:

```
<?xml version="1.0" ?>
<Catalog xmlns="x-schema:http://localhost/ProductsCat.xdr">
<Category CategoryID="1" CategoryName="Beverages">
  <Product ProductCode="1">
    <Description>Chai</Description>
    <Price>18</Price>
  </Product>
  <Product ProductCode="2">
    <Description>Chang</Description>
    <Price>19</Price>
  </Product>
</Category>
<Category CategoryID="2" CategoryName="Condiments">
  <Product ProductCode="3">
    <Description>Aniseed Syrup</Description>
    <Price>10</Price>
  </Product>
</Category>
</Catalog>
```

Выше в данной книге уже было показано, что для извлечения и форматирования данных XML может применяться целый ряд способов. Одна из трудностей, с которыми приходится сталкиваться при обмене данными с помощью языка XML, обусловлена широким разнообразием возникающих при этом возможностей. Схемы отображения позволяют преодолеть указанную трудность, поскольку дают возможность извлекать данные из базы данных в строго определенном формате. Схемы отображения указывают, как связаны друг с другом столбцы и таблицы, атрибуты и элементы.

Наиболее простой способ использования схемы XDR для преобразования в элементы и атрибуты XML данных, возвращаемых программой SQL Server, состоит в принятии на вооружение заданного по умолчанию отображения, которое предусмотрено в программе SQL Server. В соответствии с этим отображением каждая таблица преобразуется в элемент, а каждый столбец становится атрибутом. В листинге 18.47 приведена схема XDR, которая выполняет указанную задачу.

Листинг 18.47. Схема XDR, в которой предусмотрено преобразование, применяемое по умолчанию в программе SQL Server

```
<?xml version="1.0"?>
<Schema name="customers"
  xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="Customers">
    <AttributeType name="CustomerId"/>
    <AttributeType name="CompanyName"/>
  </ElementType>
</Schema>
```

В данном случае осуществляется выборка только двух столбцов, каждый из которых берется из таблицы Customers. После сохранения этой схемы XDR в виртуальном каталоге Web-сервера и выборки с ее помощью данных через URL сформируется

простой документ XML с данными из таблицы Customers базы данных Northwind, в котором применяется отображение, предусматривающее использование атрибутов.

Как показано в листинге 18.48, для преобразования столбца таблицы в элемент результирующего документа XML может применяться дескриптор ElementType из спецификации XML-Data.

Листинг 18.48. Использование дескриптора ElementType из спецификации XML-Data

```
<?xml version="1.0"?>
<Schema name="customers"
  xmlns="urn:schemas-microsoft-com:xml-data">
  <ElementType name="Customers">
    <ElementType name="CustomerId" content="textOnly"/>
    <ElementType name="CompanyName" content="textOnly"/>
  </ElementType>
</Schema>
```

Обратите внимание на то, что в каждом элементе задан атрибут content="textOnly". В сочетании с элементом ElementType этот элемент позволяет отобразить столбец на результирующий документ XML. Следует отметить, что элементы, соответствующие каждому столбцу, фактически остаются пустыми; эти элементы содержат только атрибуты, но не содержат данных.

Аннотированные схемы XDR

Аннотированная схема представляет собой схему отображения со специальными аннотациями (из пространства имен XML-SQL), которые связывают элементы и атрибуты с таблицами и столбцами. В листинге 18.49 показан документ, в котором в качестве примера используется уже знакомый нам список заказчиков Customer.

Листинг 18.49. Пример использования списка заказчиков Customer

```
<?xml version="1.0"?>
<Schema name="customers"
xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="Customer" sql:relation="Customers">
    <AttributeType name="CustomerNumber" sql:field="CustomerId"/>
    <AttributeType name="Name" sql:field="CompanyName"/>
  </ElementType>
</Schema>
```

Прежде всего заслуживает внимания ссылка на пространство имен XML-SQL в верхней части схемы. Такая ссылка должна быть задана в самом начале схемы, чтобы при последующих ссылках на пространство имен XML-SQL в самой схеме можно было использовать сокращенное обозначение sql: для этого пространства имен. Затем следует отметить, что в первом элементе ElementType применяется атрибут sql:relation, который указывает, что элемент Customer в результирующем документе связан с таблицей Customers базы данных, на которую указывает виртуальный каталог. Это позволяет вызывать в схеме любой элемент.

Наконец, обратите внимание на ссылки `sql:field`. Эти ссылки определяют, например, что элемент `CustomerNumber` относится к столбцу `CustomerId` в таблице, указанной в ссылке. Если приходится иметь дело с несколькими таблицами, то задача становится более сложной, но листинг 18.49 все равно остается весьма наглядной иллюстрацией – он показывает, что аннотированная схема позволяет определять детализированные отображения между сущностями документа и сущностями базы данных.

Схемы отображения XSD

По аналогии со схемами XDR, представления XML можно также формировать с использованием аннотированных схем на языке XSD (XML Schema Definition – определение схемы XML). В действительности в этом состоит предпочтительный способ формирования аннотированных схем, поскольку, как уже было сказано выше, XDR – это технология переходного периода, которая предшествовала окончательному утверждению стандарта XML Schema. В данном разделе показаны различные способы формирования аннотированных схем отображения XSD и приведено несколько примеров.

Как и при описании схем отображения XDR, начнем изучение схем отображения XSD с примера (листинг 18.50).

Листинг 18.50. Схема отображения XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
  <xsd:element name="Customers" >
    <xsd:complexType>
      <xsd:attribute name="CustomerID" type="xsd:string" />
      <xsd:attribute name="CompanyName" type="xsd:string" />
      <xsd:attribute name="ContactName" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Обратите внимание на наличие ссылки на пространство имен XSD, `http://www.w3.org/2001/XMLSchema`. Для обозначения этого пространства имен в листинге 18.50 используется псевдоним `xsd` (вместо этого псевдонима можно было применить и другой псевдоним, поскольку в данном случае псевдоним служит просто в качестве сокращения, позволяющего отличать элементы и атрибуты XSD от элементов и атрибутов, относящихся к другим пространствам имен). А в дальнейшем элементы и атрибуты XSD в схеме обозначаются префиксом `xsd:`.

Пространство имен схемы отображения SQLXML определяется в элементе `urn:schemas-microsoft-com:mapping-schema`. Это пространство имен используется для определения соответствия между элементами и атрибутами схемы, а также таблицами и столбцами базы данных. В определении этого пространства имен указан псевдоним `sql`, поэтому при использовании ссылок на элементы и атрибуты в пространстве имен схемы отображения SQLXML применяется префикс `sql:`.

Отображение, применяемое по умолчанию

В схеме, приведенной в листинге 18.50, для установления соответствия между сложными типами XSD и таблицами/представлениями с тем же именем, а также атрибутов со столбцами с тем же именем применяется заданное по умолчанию отображение. Обратите внимание на отсутствие каких-либо ссылок на пространство имен `sql` (после того как оно было определено). Пространство имен `sql` не используется потому, что в указанной схеме не предусмотрено явное отображение каких-либо элементов или атрибутов на таблицы или столбцы. Чтобы выполнить запрос к полученному представлению XML с помощью выражения XPath, можно сформировать шаблон, подобный приведенному ниже.

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:xpath-query mapping-schema="Customers.xsd">
    /Customers
  </sql:xpath-query>
</ROOT>
```

Чтобы передать запрос к представлению XML, создаваемому с помощью листинга 18.50, применяя в браузере приведенный выше шаблон, необходимо выполнить следующие действия.

1. Сохранить представление XML под именем `Customers.XSD` в созданном ранее подкаталоге `templates` виртуального каталога `Northwind`.
2. Сохранить приведенный выше шаблон под именем `CustomersT.XML` в том же каталоге.
3. Перейти в браузере к следующему URL:
   `http://localhost/Northwind/templates/CustomersT.XML`

Явное отображение

К тому же, схема отображения может явно задавать связи между элементами и атрибутами XSD, а также между таблицами и столбцами SQL Server. Такой подход осуществляется с использованием пространства имен схем отображения SQLXML, упомянутого выше. В частности, для установления указанных связей могут применяться элементы `sql:field` и `sql:relation`, как показано в листинге 18.51.

Листинг 18.51. Пример явного определения связей с помощью схемы отображения

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
  <xsd:element name="Cust" sql:relation="Customers" >
     <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CustNo"
                     sql:field="CustomerId"
                     type="xsd:integer" />
        <xsd:element name="Contact"
                      sql:field="ContactName"
                      type="xsd:string" />
        <xsd:element name="Company"
```

```
                    sql:field="CcompanyName"
                    type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Обратите внимание на то, что для определения отображения между элементом документа `Cust` и таблицей базы данных `Customers` используется обозначение `sql:relation`, а для определения отображений между элементами документа и столбцами таблицы применяются обозначения `sql:field`. Поскольку каждый столбец таблицы аннотируется как элемент, каждый столбец в таблице `Customers` становится отдельным элементом в результирующем документе XML. Столбцы таблицы могут также отображаться на атрибуты, как показано в листинге 18.52.

Листинг 18.52. Способ отображения столбцов таблицы на атрибуты

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
  <xsd:element name="Cust" sql:relation="Customers" >
  <xsd:complexType>
      <xsd:attribute name="CustNo" sql:field="CustomerId"
          type="xsd:integer" />
      <xsd:attribute name="Contact" sql:field="ContactName"
          type="xsd:string" />
      <xsd:attribute name="Company" sql:field="CompanyName"
          type="xsd:string" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

В листинге 18.52 элемент `complexType` не используется (поскольку он не требуется — нам не приходится определять вложенные элементы), а просто осуществляется отображение каждого столбца таблицы на атрибут в схеме XSD с помощью обозначения `sql:field`.

Установление связей

Для установления связи между двумя элементами может использоваться аннотация `sql:relationship`. Таким образом, можно определить пустой элемент `sql:relationship` и включить атрибуты `parent`, `parent-key`, `child` и `child-key` для определения связи между двумя элементами. Связи, определенные таким образом, могут быть именованными или неименованными. Применительно к элементам, отображаемым на таблицы и столбцы в базе данных SQL Server, создание связей аналогично выполнению операций соединения таблиц. При этом критерии соединения задаются в виде согласований элементов `parent/child` и `parent-key/child-key`. В листинге 18.53 представлен соответствующий пример (который взят из файла `EmpOrders.XSD` подкаталога `CH18` компакт-диска, прилагаемого к данной книге).

Листинг 18.53. Пример создания связей между элементами

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">

  <xsd:element name="Employee" sql:relation="Employees"
      type="EmployeeType" />
    <xsd:complexType name="EmployeeType" >
      <xsd:sequence>
         <xsd:element name="Order"
                     sql:relation="Orders">
          <xsd:annotation>
            <xsd:appinfo>
              <sql:relationship
                 parent="Employees"
                 parent-key="EmployeeID"
                child="Orders"
                child-key="EmployeeID" />
            </xsd:appinfo>
           </xsd:annotation>
           <xsd:complexType>
            <xsd:attribute name="OrderID" type="xsd:integer" />
            <xsd:attribute name="EmployeeID" type="xsd:integer" />
          </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
        <xsd:attribute name="EmployeeID"   type="xsd:integer" />
        <xsd:attribute name="LastName"  type="xsd:string" />
    </xsd:complexType>
</xsd:schema>
```

В схеме, приведенной в листинге 18.53, создается связь между элементами `Employee` и `Order` с помощью атрибута `EmployeeID`. И эта задача решается с помощью атрибутов, применяемых в качестве обозначений, которые предусмотрены в пространстве имен `mapping-schema` компании Microsoft.

Аннотация `sql:inverse`

Аннотацию `sql:inverse` можно использовать для инвертирования связей, установленных с помощью аннотации `sql:relationship`. Такая необходимость может быть связана с применением шаблонов обновления. Дело в том, что средства поддержки шаблонов обновления SQLXML интерпретируют схему для определения таблиц, обновляемых с помощью некоторого шаблона обновления (шаблоны обновления рассматриваются в следующем разделе). Родительско-дочерние связи, установленные с помощью аннотации `sql:relationship`, определяют порядок, в котором происходят операции удаления и вставки строк. Если обозначение `sql:relationship` будет задано таким образом, что родительско-дочерняя связь между таблицами станет обратной по отношению к основополагающей связи первичного ключа и внешнего ключа, то намеченная попытка выполнить вставку или удаление окончится неудачей из-за нарушения ограничений внешнего ключа. Атрибуту `sql:inverse` в элементе `sql:relationship` может быть присвоено

значение 1 (или истина), чтобы в результате инвертирования связей не возникала указанная ситуация.

Область применения обозначения `sql:inverse` ограничивается шаблонами обновления, поскольку в обычной схеме отображения использовать инверсию не имеет смысла. В листинге 18.54 приведен пример схемы отображения, в котором аннотационный атрибут `sql:inverse` используется по назначению. (Файл, соответствующий этому листингу, можно найти под именем `OrderDetails.XSD` в каталоге `CH18` компакт-диска, прилагаемого к данной книге.)

Листинг 18.54. Пример применения атрибута `sql:inverse`

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">

  <xsd:element name="OrderDetails" sql:relation="[Order Details]"
       type="OrderDetailsType" />
    <xsd:complexType name="OrderDetailsType" >
      <xsd:sequence>
        <xsd:element name="Order"
                     sql:relation="Orders">
           <xsd:annotation>
            <xsd:appinfo>
               <sql:relationship
                parent="[Order Details]"
                 parent-key="OrderID"
                child="Orders"
                 child-key="OrderID"
              inverse="true" />
             </xsd:appinfo>
           </xsd:annotation>
           <xsd:complexType>
            <xsd:attribute name="OrderID" type="xsd:integer" />
            <xsd:attribute name="EmployeeID" type="xsd:integer" />
          </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="ProductID"   type="xsd:integer" />
      <xsd:attribute name="Qty" sql:field="Quantity" type=
"xsd:integer" />
     </xsd:complexType>
</xsd:schema>
```

Обратите внимание на то, что имя таблицы `Order Details` заключено в квадратные скобки. В этом состоит одно из требований по использованию в схеме отображения имен таблиц SQL Server, содержащих пробелы.

Аннотация `sql:mapped`

Аннотация `sql:mapped` может использоваться для контроля над тем, должен ли атрибут или элемент отображаться на объект базы данных. Если применяется отображение, заданное по умолчанию, то каждому элементу и атрибуту в схеме отображения ставится в соответствие объект базы данных. А если используется

схема, в которой имеются элементы или атрибуты, не требующие отображения на объекты базы данных, то в спецификации элемента или атрибута XSD может быть задана аннотация sql:mapped, равная 0 (или ложь). Аннотация sql:mapped особенно полезна тогда, когда схема не может быть изменена или же схема используется для проверки допустимости данных XML из других источников и содержит элементы или атрибуты, не имеющие аналогов в локальной базе данных. В листинге 18.55 показана схема отображения, в которой аннотация sql:mapped применяется для обозначения включенного в схему элемента, который не отображается на объект базы данных.

Листинг 18.55. Пример использования аннотации sql:mapped в схеме отображения

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">

  <xsd:element name="Employee" sql:relation="Employees"
       type="EmployeeType" />
    <xsd:complexType name="EmployeeType" >
      <xsd:sequence>
         <xsd:element name="Order"
                     sql:relation="Orders">
           <xsd:annotation>
            <xsd:appinfo>
              <sql:relationship
                 parent="Employees"
                parent-key="EmployeeID"
                child="Orders"
                child-key="EmployeeID" />
             </xsd:appinfo>
           </xsd:annotation>
           <xsd:complexType>
             <xsd:attribute name="OrderID" type="xsd:integer" />
            <xsd:attribute name="EmployeeID" type="xsd:integer" />
           </xsd:complexType>
         </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="EmployeeID"    type="xsd:integer" />
      <xsd:attribute name="LastName"  type="xsd:string" />
      <xsd:attribute name="Level" type="xsd:integer"
           sql:mapped="0" />
    </xsd:complexType>
</xsd:schema>
```

Обратите внимание на то, что в элемент Employee включен атрибут Level. Поскольку определение этого атрибута включает аннотацию sql:mapped, которой присвоено значение ложь, атрибут Level не отображается на какой-либо объект базы данных.

Аннотации sql:limit-field и sql:limit-value

Аналогично способу, с помощью которого можно задавать критерии выборки из представлений XML с использованием выражений XPath, критерии выборки

можно также задавать с учетом значений, возвращаемых из базы данных, с помощью аннотаций sql:limit-field и sql:limit-value. Аннотация sql:limit-field определяет столбец критерия выборки из базы данных, а аннотация sql:limit-value определяет значение, которое должно служить в качестве критерия выборки. Обратите внимание на то, что аннотация sql:limit-value фактически является необязательной; если эта аннотация не задана, то предполагается использование в качестве нее значения NULL. В листинге 18.56 приведен пример схемы отображения, в которой предусматривается применение в качестве критерия выборки значения столбца из базы данных.

Листинг 18.56. Пример схемы отображения, в которой предусматривается применение значения столбца из базы данных в качестве критерия выборки

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">

  <xsd:element name="Employee" sql:relation="Employees"
       type="EmployeeType" />
    <xsd:complexType name="EmployeeType" >
      <xsd:sequence>
         <xsd:element name="Order"
                      sql:relation="Orders">
           <xsd:annotation>
            <xsd:appinfo>
              <sql:relationship
                 parent="Employees"
                 parent-key="EmployeeID"
                child="Orders"
                 child-key="EmployeeID" />
             </xsd:appinfo>
           </xsd:annotation>
           <xsd:complexType>
            <xsd:attribute name="OrderID" type="xsd:integer" />
            <xsd:attribute name="EmployeeID" type="xsd:integer" />
           </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
         <xsd:attribute name="EmployeeID"
          type="xsd:integer"
           sql:limit-field="EmployeeID"
           sql:limit-value="3"/>
        <xsd:attribute name="LastName"  type="xsd:string" />
    </xsd:complexType>

</xsd:schema>
```

Схема, приведенная в листинге 18.56, обеспечивает выборку данных из документа XML с учетом значений столбца EmployeeID из базы данных. Вместе с формируемым документом возвращаются только те строки, в которых значение в столбце EmployeeID равно 3. После передачи запроса URL к этой схеме отображения с использованием следующего шаблона:

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:xpath-query mapping-schema="EmpOrders_Filtered.XSD">
    /Employee
  </sql:xpath-query>
</ROOT>
```

формируется документ, который разворачивается в браузере примерно так, как показано ниже (результаты приведены в сокращенном виде).

```
<ROOT xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <Employee EmployeeID="3" LastName="Leverling">
  <Order EmployeeID="3" OrderID="10251" />
  <Order EmployeeID="3" OrderID="10253" />
  <Order EmployeeID="3" OrderID="10256" />
  <Order EmployeeID="3" OrderID="10266" />
  <Order EmployeeID="3" OrderID="10273" />
  <Order EmployeeID="3" OrderID="10283" />
  <Order EmployeeID="3" OrderID="10309" />
  <Order EmployeeID="3" OrderID="10321" />
  <Order EmployeeID="3" OrderID="10330" />
  <Order EmployeeID="3" OrderID="10332" />
  <Order EmployeeID="3" OrderID="10346" />
  <Order EmployeeID="3" OrderID="10352" />
...
</ROOT>
```

Аннотация sql:key-fields

Аннотация sql:key-fields используется для обозначения ключевых столбцов в таблице, на которую отображается представление XML. Аннотация sql:key-fields обычно требуется в схемах отображения для обеспечения того, чтобы в результирующем документе XML гарантировалось правильное вложение элементов. Необходимость в этом связана с тем, что для создания иерархии вложения в документе применяются ключевые столбцы основополагающей таблицы. В результате этого процесс выработки кода XML становится зависимым от порядка расположения основополагающих данных. Если невозможно определить ключевые столбцы основополагающих данных, то выработанный код XML может быть сформирован неправильно. Поэтому следует всегда либо задавать аннотацию sql:key-fields, либо использовать элементы, которые отображаются непосредственно на таблицы базы данных. В листинге 18.57 приведен пример схемы отображения, в которой применяется аннотация sql:key-fields (пример взят из файла EmpOrders_KeyFields.XSD, который находится в каталоге CH18 компакт-диска, прилагаемого к данной книге).

Листинг 18.57. Пример схемы отображения, в которой применяется аннотация sql:key-fields

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">

  <xsd:element name="Employee"
  sql:relation="Employees"
  type="EmployeeType"
  sql:key-fields="EmployeeID"/>
```

```
    <xsd:complexType name="EmployeeType" >
      <xsd:sequence>
         <xsd:element name="Order"
                      sql:relation="Orders">
          <xsd:annotation>
           <xsd:appinfo>
             <sql:relationship
                parent="Employees"
               parent-key="EmployeeID"
               child="Orders"
               child-key="EmployeeID" />
            </xsd:appinfo>
          </xsd:annotation>
          <xsd:complexType>
           <xsd:attribute name="OrderID" type="xsd:integer" />
           <xsd:attribute name="EmployeeID" type="xsd:integer" />
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
        <xsd:attribute name="LastName"  type="xsd:string" />
        <xsd:attribute name="FirstName"  type="xsd:string" />
    </xsd:complexType>
</xsd:schema>
```

Обратите внимание на то, что в листинге 18.57 отображение для столбца `EmployeeID` таблицы `Employees` не предусмотрено. Но в связи с отсутствием этого столбца в отображении отсутствует и столбец, с помощью которого можно сформировать соединение таблицы `Employees` с таблицей `Orders`. Включение столбца `EmployeeID` в аннотацию `sql:key-fields` позволяет оставить этот столбец вне отображения, но тем не менее определить связь между двумя таблицами.

Шаблоны обновления

До сих пор в данной главе рассматривались способы выборки данных из базы данных SQL Server в формате XML, но еще не были описаны способы обновления данных SQL Server с помощью языка XML. Шаблоны обновления лежат в основе одного из способов обновления данных в базе данных SQL Server с помощью языка XML. По сути, шаблоны обновления представляют собой шаблоны со специальными атрибутами и элементами, позволяющие задавать данные, которые необходимо обновить и указать способ их обновления. Шаблон обновления содержит начальный образ и конечный образ данных, в которые необходимо внести изменения. Для передачи шаблонов обновления в программу SQL Server применяется в основном такой же способ, как и для передачи шаблонов выборки. Все механизмы выполнения, доступные для использования с шаблонами выборки, действуют столь же успешно применительно к шаблонам обновления. Шаблоны обновления можно передавать с помощью команды `POST` по протоколу HTTP, сохранять шаблоны обновления в файлах и выполнять их с помощью URL, а также вызывать шаблоны обновления непосредственно на выполнение с использованием интерфейсов ADO и OLE DB.

Принципы действия шаблонов обновления

В основе создания шаблонов обновления лежит пространство имен xml-updategram. Для ссылки на это пространство имен применяется уточнитель xmlns:updg. Каждый шаблон обновления содержит по меньшей мере один элемент sync. Элемент sync содержит информацию о тех изменениях, которые требуется внести в данные, представленную в форме элементов before и after. Элемент before содержит начальный образ данных, которые необходимо модифицировать. В обычных обстоятельствах элемент before содержит также ссылку на первичный или потенциальный ключ, которая позволяет программе SQL Server находить строку, подлежащую изменению. Обратите внимание на то, что для обновления с помощью элемента before может быть выбрана только одна строка. Если элементы и атрибуты, включенные в элемент before, идентифицируют больше одной строки, то вырабатывается сообщение об ошибке.

Шаблон обновления, предназначенный для удаления строк, содержит начальный образ, но не включает конечный образ, а шаблон обновления, предназначенный для вставки строк, имеет конечный образ, но не содержит начальный образ. Кроме того, безусловно, шаблон обновления, предназначенный для внесения изменений в данные, содержит и начальный, и конечный образы. Пример шаблона обновления последнего типа приведен в листинге 18.58.

Листинг 18.58. Шаблон обновления, предназначенный для внесения изменений в данные

```
<?xml version="1.0"?>
<employeeupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync>
    <updg:before>
      <Employees EmployeeID="4"/>
    </updg:before>
    <updg:after>
      <Employees City="Scotts Valley" Region="CA"/>
    </updg:after>
  </updg:sync>
</employeeupdate>
```

В примере, показанном в листинге 18.58, происходит внесение изменений в столбцы City и Region таблицы Employees базы данных Northwind, которые относятся к полю EmployeeID со значением 4. Атрибут EmployeeID в элементе before указывает строку, в которую должны быть внесены изменения, а атрибуты City и Region элемента after указывают, какие столбцы должны быть изменены и какие значения должны быть им присвоены.

Каждый пакет обновлений в элементе sync рассматривается как отдельная транзакция. Могут либо завершиться успешно все обновления в элементе sync, либо не будет внесено в базу данных ни одно из этих обновлений. В шаблоны обновления можно включать несколько элементов sync для разбиения обновлений на несколько транзакций.

Отображение данных

Безусловно, при передаче данных на сервер для осуществления операций обновления, удаления и вставки с помощью средств XML необходимо иметь возможность связывать значения в применяемом для этого документе XML со столбцами в целевой таблице базы данных. В программе SQL Server предусмотрены два средства достижения указанной цели: применяемое по умолчанию отображение и схемы отображения.

Применяемое по умолчанию отображение

Естественно, что простейший способ отображения данных, представленных в шаблоне обновления, на столбцы в целевой таблице состоит в использовании применяемого по умолчанию отображения (называемого также *встроенным отображением*). При использовании применяемого по умолчанию отображения предполагается, что дескриптор верхнего уровня элемента `before` или `after` ссылается на целевую таблицу базы данных, а каждый субэлемент либо атрибут элемента `before` или `after` ссылается на столбец с тем же именем из целевой таблицы.

Ниже приведен пример, который показывает, как отобразить дескриптор документа XML на столбец `OrderID` из таблицы `Orders`.

```
<Orders OrderID="10248"/>
```

Этот пример демонстрирует способ, с помощью которого на столбцы таблицы можно отображать атрибуты XML. Кроме того, на столбцы таблицы можно отображать субэлементы следующим образом:

```
<Orders>
  <OrderID>10248</OrderID>
</Orders>
```

При составлении шаблона отображения не обязательно требуется строго придерживаться либо способа отображения, предусматривающего использование атрибутов, либо способа отображения, предусматривающего использование элементов. Эти два способа отображения можно произвольно сочетать в каждом конкретном элементе `before` или `after`, как показано ниже.

```
<Orders OrderID="10248">
  <ShipCity>Reims</ShipCity>
</Orders>
```

Для замены символов в именах таблицы, которые не разрешено использовать в элементах XML (например, пробелов), применяется четырехпозиционный шестнадцатеричный код UCS-2. Например, для формирования ссылки на таблицу `Order Details` базы данных `Northwind` можно применить следующую конструкцию:

```
<Order_x0020_Details OrderID="10248"/>
```

Схемы отображения

Для отображения данных, представленных в шаблоне обновления, на таблицы и столбцы в базе данных, можно также использовать схемы отображения XDR и XSD. Чтобы указать схему отображения для шаблона обновления, можно использовать

атрибут updg:mapping-schema элемента sync. В листинге 18.59 приведен пример, который задает шаблон обновления для таблицы Orders.

Листинг 18.59. Шаблон обновления для таблицы Orders

```
<?xml version="1.0"?>
<orderupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync updg:mapping-schema="OrderSchema.xml">
    <updg:before>
      <Order OID="10248"/>
    </updg:before>
    <updg:after>
      <Order City="Reims"/>
    </updg:after>
  </updg:sync>
</orderupdate>
```

В листинге 18.60 приведена схема отображения XDR для шаблона обновления, показанного в листинге 18.59.

Листинг 18.60. Схема отображения XDR для шаблона обновления, показанного в листинге 18.59

```
<?xml version="1.0"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="Order" sql:relation="Orders">
     <AttributeType name="OID"/>
    <AttributeType name="City"/>
    <attribute type="OID" sql:field="OrderID"/>
    <attribute type="City" sql:field="ShipCity"/>
  </ElementType>
</Schema>
```

В листинге 18.61 приведена схема отображения XSD для шаблона обновления, показанного в листинге 18.59.

Листинг 18.61. Схема отображения XSD для шаблона обновления, показанного в листинге 18.59

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
  <xsd:element name="Order" sql:relation="Orders" >
    <xsd:complexType>
      <xsd:attribute name="OID" sql:field="OrderId"
           type="xsd:integer" />
       <xsd:attribute name="City" sql:field="ShipCity"
          type="xsd:string" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Вполне очевидно, что схема отображения позволяет установить соответствие между компоновкой документа XML и структурой таблицы базы данных (в приведенных выше примерах применяется таблица Orders базы данных Northwind). Дополнительная информация о том, как создаются схемы отображения XML, приведена выше, в разделе "Схемы отображения".

Значения NULL

Обычно принято представлять в базе данных недостающие данные или данные, неприменимые в определенных контекстах, в виде значений NULL. Для представления или выборки данных NULL в шаблоне обновления используется атрибут nullvalue элемента sync, который задает метку-заполнитель для значения NULL. Эта метка-заполнитель затем применяется в шаблоне обновления во всех тех местах, где необходимо указать значение NULL (как показано в листинге 18.62).

Листинг 18.62. Способ задания значения NULL в шаблоне обновления

```
<?xml version="1.0"?>
<employeeupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync updg:nullvalue="NONE">
    <updg:before>
      <Orders OrderID="10248"/>
    </updg:before>
    <updg:after>
      <Orders ShipCity="Reims" ShipRegion="NONE"
        ShipName="NONE"/>
    </updg:after>
  </updg:sync>
</employeeupdate>
```

Вполне очевидно, что в листинге 18.62 определена метка-заполнитель с именем NONE для значения NULL. Затем эта метка-заполнитель используется для присваивания значения NULL столбцам ShipRegion и ShipName.

Параметры

Любопытно отметить, что параметры в шаблонах обновления задаются немного иначе по сравнению с шаблонами выборки. Для обозначения параметров в шаблоне обновления используются не символы коммерческого at (@), а символы доллара ($), как показано в листинге 18.63.

Листинг 18.63. Пример обозначения параметров в шаблоне обновления

```
<?xml version="1.0"?>
<orderupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:header>
    <updg:param name="OrderID"/>
```

```
    <updg:param name="ShipCity"/>
  </updg:header>
  <updg:sync>
     <updg:before>
      <Orders OrderID="$OrderID"/>
    </updg:before>
    <updg:after>
      <Orders ShipCity="$ShipCity"/>
     </updg:after>
  </updg:sync>
</orderupdate>
```

В связи с этим нюансом возникают особые требования к шаблонам обновления, касающиеся передачи долларовых денежных значений в качестве параметров. Чтобы передать значение параметра, выраженное в долларовых денежных единицах, в столбец таблицы (например, в столбец `Freight` таблицы `Orders`), необходимо отобразить соответствующие данные с помощью схем отображения.

Параметры со значением NULL

Для того чтобы передать с помощью шаблона обновления параметр со значением NULL, необходимо включить в элемент `header` шаблона обновления атрибут `nullvalue` с меткой-заполнителем. Затем это значение метки-заполнителя передается в шаблон обновления для задания параметра со значением NULL. Указанный способ аналогичен способу, с помощью которого задается значение NULL для столбца в шаблоне обновления; различие между этими двумя способами состоит в том, что атрибут `nullvalue` для значений столбцов задается в элементе `sync`, а для значений параметров — в элементе `header`. Пример применения последнего способа показан в листинге 18.64.

Листинг 18.64. Способ задания параметра со значением NULL

```
<?xml version="1.0"?>
<orderupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:header nullvalue="NONE">
    <updg:param name="OrderID"/>
<updg:param name="ShipCity"/>
</updg:header>
  <updg:sync>
     <updg:before>
      <Orders OrderID="$OrderID"/>
    </updg:before>
    <updg:after>
      <Orders ShipCity="$ShipCity"/>
     </updg:after>
  </updg:sync>
</orderupdate>
```

Шаблон обновления, приведенный в листинге 18.64, принимает два параметра. Передача значения NONE влечет за собой присваивание значения NULL столбцу `ShipCity` соответствующего заказа.

Обратите внимание на то, что при определении метки-заполнителя nullvalue для параметров в элементе header шаблона обновления не применяется уточнитель xml-updategram (updg:).

Способ обновления нескольких строк

Как было указано выше, каждый элемент before позволяет идентифицировать не больше одной строки. Это означает, что для обновления нескольких строк необходимо включить по одному элементу для каждой строки, которую требуется обновить.

Атрибут id

Если в элементах before и after задается несколько субэлементов. то программа SQL Server предъявляет требование, чтобы был предусмотрен определенный способ согласования каждого элемента before с соответствующим ему элементом after. Один из таких способов состоит в использовании атрибута id. Атрибут id позволяет задать уникальное строковое значение, которое может использоваться для согласования элемента before с элементом after. Пример применения атрибута id приведен в листинге 18.65.

Листинг 18.65. Пример применения атрибута id

```
<?xml version="1.0"?>
<orderupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync>
    <updg:before>
       <Orders updg:id="ID1" OrderID="10248"/>
       <Orders updg:id="ID2" OrderID="10249"/>
    </updg:before>
    <updg:after>
      <Orders updg:id="ID2" ShipCity="Munster"/>
      <Orders updg:id="ID1" ShipCity="Reims"/>
    </updg:after>
  </updg:sync>
</orderupdate>
```

В листинге 18.65 показано, как используется атрибут updg:id для согласования субэлементов в элементах before и after. Даже несмотря на то, что субэлементы в элементах before и after заданы в разном порядке, программа SQL Server имеет возможность применить обновления к нужным строкам.

Использование многочисленных элементов before и after

Еще один способ обновления нескольких строк состоит в задании нескольких элементов before и after, а не нескольких субэлементов элементов before и after. Для каждой строки, которую необходимо модифицировать, задается отдельная пара элементов before/after, как показано в листинге 18.66.

Листинг 18.66. Применение нескольких элементов `before` и `after`

```
<?xml version="1.0"?>
<orderupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync>
     <updg:before>
       <Orders OrderID="10248"/>
     </updg:before>
     <updg:after>
      <Orders ShipCity="Reims"/>
     </updg:after>
     <updg:before>
       <Orders OrderID="10249"/>
     </updg:before>
     <updg:after>
      <Orders ShipCity="Munster"/>
     </updg:after>
  </updg:sync>
</orderupdate>
```

Вполне очевидно, что в шаблоне обновления, показанном в листинге 18.66, обновляются две строки. Данный шаблон включает отдельную пару элементов `before/after` для каждого обновления.

Полученные результаты

Результат, возвращаемый в клиентское приложение, в котором вызван на выполнение шаблон обновления, обычно представляет собой документ XML, содержащий пустой корневой элемент, заданный в шаблоне обновления. Например, можно рассчитывать на получение таких результатов после вызова на выполнение следующего шаблона обновления `orderupdate`:

```
<?xml version="1.0"?>
<orderupdate xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
</orderupdate>
```

Любые ошибки, возникающие во время выполнения шаблона обновления, возвращаются в виде элементов `<?MSSQLError>` в корневом элементе шаблона обновления.

Значения столбца `identity`

В реальных приложениях часто возникает необходимость осуществлять выборку значения идентификатора `identity`, выработанное программой SQL Server для одной таблицы, после чего вставлять это значение в другую таблицу. Необходимость в этом особенно часто возникает, если требуется вставлять данные в таблицу, первичным ключом которой является столбец `identity`, и в таблицу, которая ссылается на этот первичный ключ через ограничение внешнего ключа. Рассмотрим в качестве примера операцию вставки строк с данными о заказах

в таблицы Orders и Order Details базы данных Northwind. Как показывает само имя этой таблицы, Order Details хранит информацию расшифровки для заказов из таблицы Orders. Частью первичного ключа таблицы Order Details является столбец OrderID таблицы Orders. После вставки новой строки в таблицу Orders необходимо иметь возможность выполнить выборку значения столбца OrderID и вставить его в таблицу Order Details.

В языке Transact-SQL такое действие обычно выполняется с помощью триггера вставки INSTEAD OF или хранимой процедуры. А для выполнения указанного действия с помощью шаблона обновления используется атрибут at-identity. Атрибут at-identity, как и атрибут id, служит в качестве метки-заполнителя; везде, где в шаблоне обновления применяется значение этого атрибута, программа SQL Server подставляет значение identity для соответствующей таблицы (каждая таблица может иметь только один столбец identity). Соответствующий пример приведен в листинге 18.67.

Листинг 18.67. Пример использования атрибута at-identity

```
<?xml version="1.0"?>
<orderinsert xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync>
    <updg:before>
    </updg:before>
    <updg:after>
      <Orders updg:at-identity="ID" ShipCity="Reims"/>
      <Order_x0020_Details OrderID="ID" ProductID="11"
       UnitPrice="$16.00" Quantity="12"/>
      <Order_x0020_Details OrderID="ID" ProductID="42"
       UnitPrice="$9.80" Quantity="10"/>
    </updg:after>
  </updg:sync>
</orderinsert>
```

В листинге 18.67 для обозначения столбца identity таблицы Orders используется строка "ID". После присваивания значения этой строке ее можно использовать в операциях вставки, применяемых в таблице Order Details.

Вполне вероятно, что в приложении может потребоваться не только применить значение столбца identity где-либо в шаблоне обновления, но и возвратить это значение в клиентскую программу. Для этого достаточно предусмотреть в шаблоне атрибут returnid элемента after и присвоить этому атрибуту значение метки-заполнителя at-identity, как показано в листинге 18.68.

Листинг 18.68. Пример присваивания атрибуту returnid элемента after значения метки-заполнителя at-identity

```
<?xml version="1.0"?>
<orderinsert xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <updg:sync>
    <updg:before>
```

```
    </updg:before>
    <updg:after updg:returnid="ID">
       <Orders updg:at-identity="ID" ShipCity="Reims"/>
       <Order_x0020_Details OrderID="ID" ProductID="11"
        UnitPrice="$16.00" Quantity="12"/>
       <Order_x0020_Details OrderID="ID" ProductID="42"
        UnitPrice="$9.80" Quantity="10"/>
     </updg:after>
  </updg:sync>
</orderinsert>
```

Вызов этого шаблона обновления на выполнение приводит к возврату документа XML, который выглядит примерно так:

```
<?xml version="1.0"?>
<orderinsert xmlns:updg=
    "urn:schemas-microsoft-com:xml-updategram">
  <returnid>
    <ID>10248</ID>
  </returnid>
</orderinsert>
```

Глобально уникальные идентификаторы

На практике нередко приходится сталкиваться с такой ситуацией, когда в качестве значения ключа во всем секционированном представлении или в другом объекте распределенной системы используются глобально уникальные идентификаторы (Globally Unique Identifier – GUID) (такие глобально уникальные идентификаторы хранятся в столбцах типа `uniqueidentifier`). При обычных обстоятельствах для выработки новых значений типа `uniqueidentifier` применяется функция `NEWID()` языка Transact-SQL. Эквивалентом функции `NEWID()` в шаблоне обновления является атрибут `guid`. Атрибут `guid` может быть задан в элементе `sync` в целях выработки идентификатора GUID в любом нужном месте. Как и атрибуты `id`, `nullvalue` и другие атрибуты, представленные в данном разделе, атрибут `guid` устанавливает метку-заполнитель, которую можно затем передавать в другие элементы и атрибуты шаблона обновления для использования выработанного значения идентификатора GUID. Соответствующий пример приведен в листинге 18.69.

Листинг 18.69. Пример использования атрибута `guid`

```
<orderinsert>
  xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
  <updg:sync>
     <updg:before>
    </updg:before>
    <updg:after>
       <Orders updg:guid="GUID">
         <OrderID>GUID</OrderID>
        <ShipCity>Reims</ShipCity>
      </Orders>
```

```
      <Order_x0020_Details OrderID="GUID" ProductID="11"
        UnitPrice="$16.00" Quantity="12"/>
      <Order_x0020_Details OrderID="GUID" ProductID="42"
        UnitPrice="$9.80" Quantity="10"/>
    </updg:after>
  </updg:sync>
</orderinsert>
```

Компонент XML Bulk Load

Как показано в приведенном выше описании шаблонов обновления и средств OPENXML, задача вставки данных XML в базу данных SQL Server является относительно простой. Однако оба эти способа загрузки данных имеют один серьезный недостаток – они не применимы для загрузки больших объемов данных. Способы применения шаблонов обновления и средств OPENXML для загрузки больших объемов данных XML в базу данных SQL Server являются малопроизводительными и требуют большого расхода ресурсов.

В составе средств SQLXML предусмотрено одно средство, предназначенное специально для решения указанной проблемы, которое основано на использовании компонента XML Bulk Load. Указанный компонент представляет собой COM-компонент, который можно вызывать из языков и инструментальных средств, поддерживающих технологию OLE Automation, таких как Visual Basic, Delphi и даже Transact-SQL. Компонент XML Bulk Load предоставляет объектно-ориентированный интерфейс для массовой загрузки данных XML в форме, аналогичной команде BULK INSERT языка Transact-SQL.

С точки зрения архитектуры приложения компонент XML Bulk Load представляет собой внутрипроцессный COM-компонент с именем SQLXMLBulkLoad, который находится в библиотеке DLL, имя которой представлено в форме XBLKLDn.DLL. Массовая загрузка данных в базу данных SQL Server с помощью этого компонента осуществляется с использованием интерфейса массовой загрузки собственного средства доступа SQLOLEDB служб OLE DB программы SQL Server. Запустив на выполнение трассировку Profiler одновременно с тем, как происходит массовая загрузка, можно обнаружить, что в результатах трассировки появляется информация об языковом событии INSERT BULK. Событие INSERT BULK является признаком применения пакета TDS специального типа, который предназначен именно для массовой загрузки данных. Это событие не относится ни к типу истинного языкового события, ни к типу события RPC; вместо этого оно представляет собой отдельный тип пакета данных, передаваемый средствами массовой загрузки на сервер, когда возникает необходимость инициировать операцию массового копирования.

Использование компонента XML Bulk Load

Первым шагом в ходе использования компонента XML Bulk Load является определение схемы отображения, которая устанавливает соответствие между импортируемыми данными XML, а также таблицами и столбцами базы данных. Во

время загрузки данных XML компонент XML Bulk Load считывает данные в виде потока и использует схему отображения для принятия решений о том, в какие объекты базы данных должны поступать эти данные.

Схема отображения определяет, в какую часть каждой строки будут введены данные, обрабатываемые компонентом Bulk Load. После считывания закрывающего дескриптора каждой строки соответствующие данные записываются в базу данных.

Доступ к самому компоненту Bulk Load предоставляется с помощью интерфейса SQLXMLBulkLoad COM-объекта SQLXMLBulkLoad. Первый шаг состоит в использовании указанного COM-объекта для подключения к базе данных с применением строки соединения OLE DB; еще один вариант подключения состоит в том, что в качестве свойства ConnectionCommand используемого COM-объекта задается существующий объект Command интерфейса ADO. Второй шаг состоит в вызове метода Execute применяемого COM-объекта. В качестве примера выполнения двух указанных шагов может служить код VBScript, приведенный в листинге 18.70.

Листинг 18.70. Пример определения строки соединения и вызова метода Execute

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

В качестве источника для загрузки может быть также указан поток ввода-вывода XML (а не файл); благодаря наличию такой возможности передача данных между СУБД становится весьма несложной (при условии, что компьютерные платформы, участвующие в передаче данных, обладают средствами поддержки XML).

Фрагменты XML

Присваивание свойству XMLFragment значения True позволяет использовать компонент Bulk Load для загрузки данных из фрагмента XML (т.е. документа XML без корневого элемента, который по своему типу напоминает документ, возвращаемый средствами расширения FOR XML языка Transact-SQL). Пример применения свойства XMLFragment приведен в листинге 18.71.

Листинг 18.71. Пример использования свойства XMLFragment

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.XMLFragment = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Принудительное применение ограничений проверки и целостности

По умолчанию использование компонента XML Bulk Load не связано с принудительным применением ограничений проверки и целостности. Принудительное применение ограничений по мере загрузки данных существенно замедляет процесс загрузки, поэтому указанный компонент не должен налагать эти ограничения, если ему не дано на это разрешение. Но принудительное применение ограничений может потребоваться, если загрузка данных осуществляется непосредственно в производственные таблицы и необходимо обеспечить, чтобы целостность данных не была нарушена. Чтобы вынудить компонент принудительно применять заданные ограничения по мере загрузки данных с его помощью, необходимо присвоить свойству `CheckConstraints` значение `True`, как показано в листинге 18.72.

Листинг 18.72. Использование свойства `CheckConstraints`

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.CheckConstraints = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Дублирующиеся ключи

Обычно при обнаружении дублирующегося ключа следует останавливать процесс массовой загрузки. Как правило, появление дублирующегося ключа означает, что получены непредусмотренные значения данных или возникло какое-то искажение данных, поэтому необходимо проверить источник данных, прежде чем продолжить работу. Но из этого правила есть исключения. Например, допустим, что в базу данных ежедневно поступают данные из внешнего источника, содержащие всю таблицу. Каждый день в этих данных появляется несколько новых строк, но в основном данные, представленные в виде документа XML, уже существуют в рассматриваемой таблице. В задачу пользователя входит загрузка только новых строк, но в программе, являющейся внешним источником, который предоставляет пользователю данные, нет информации о том, какие данные уже есть у пользователя, а каких данных у него нет. К тому же, один источник данных может служить для предоставления данных многочисленным компаниям-получателям, и данные, которые находятся в базе данных одного пользователя, могут отсутствовать в базе данных другого.

В такой ситуации можно перед загрузкой присвоить свойству `IgnoreDuplicateKeys` значение `True`, после чего компонент массовой загрузки будет игнорировать встретившиеся значения дублирующихся ключей. Массовая загрузка после обнаружения дублирующегося ключа не будет прекращаться; такой вариант загрузки предусматривает просто игнорирование строк, содержащих дублирующиеся

ключи, и загрузку строк с уникальными ключами, в соответствии с поставленной задачей. Пример использования свойства IgnoreDuplicateKeys приведен в листинге 18.73.

Листинг 18.73. Пример использования свойства IgnoreDuplicateKeys

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.IgnoreDuplicateKeys = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Если свойству IgnoreDuplicateKeys присвоено значение True, то операции вставки, которые могут вызвать появление в таблице дублирующегося ключа, все равно завершаются неудачей, но процесс массовой загрузки не останавливается. Строки, отличные от строк с дублирующимися ключами, обрабатываются так, как если бы в процессе загрузки вообще не возникали какие-либо ошибки.

Столбцы IDENTITY

Свойству KeepIdentity объекта SQLXMLBulkLoad значение True присвоено по умолчанию. Благодаря этому значения столбцов идентификаторов identity, представленные в данных XML, непосредственно загружаются в базу данных, а не генерируются программой SQL Server динамически. При обычных обстоятельствах требуется именно такая организация загрузки данных, но если есть необходимость в том, чтобы значения identity вместо этого вырабатывала программа SQL Server, то можно присвоить свойству KeepIdentity значение False.

При использовании свойства KeepIdentity необходимо учитывать целый ряд предостережений. Прежде всего, если свойству KeepIdentity присвоено значение True, то программа SQL Server выполняет команду SET IDENTITY_INSERT, чтобы разрешить вставку значений identity в целевую таблицу. Выполнение команды SET IDENTITY_INSERT связано с соблюдением определенных требований по применению прав доступа, а право на выполнение команд по умолчанию присваивается роли sysadmin (системный администратор), постоянным ролям базы данных db_owner (владелец базы данных) и db_ddladmin (администратор DDL базы данных), а также владельцу таблицы. Это означает, что пользователь, который не является владельцем целевой таблицы, а также не обладает ролью sysadmin, db_owner или db_ddladmin, скорее всего, столкнется с затруднениями, попытавшись выполнить загрузку данных с помощью компонента XML Bulk Load. Пользователю для этого недостаточно просто иметь права роли bulkadmin (администратор массовой загрузки).

Следующее предостережение состоит в том, что при обычных условиях желательно сохранять значения identity при массовой загрузке данных в таблицу с зависимыми таблицами. Решение о предоставлении серверу права переопределять значения identity может оказаться катастрофическим, поскольку может

привести к разрушению родительско-дочерних связей между таблицами без надежды на их восстановление. Если первичным ключом родительской таблицы является столбец identity, а при загрузке данных в эту таблицу свойству KeepIdentity присвоено значение False, то может оказаться, что синхронизация данных родительской таблицы с данными, загружаемыми в дочернюю таблицу, стала невозможной. К счастью, свойство KeepIdentity разрешено по умолчанию, поэтому обычно возможность возникновения указанной ситуации не должна быть поводом для беспокойства. Но если вы решили присвоить свойству KeepIdentity значение False, то должны знать, что делаете.

В листинге 18.74 приведен пример присваивания значения свойству KeepIdentity.

Листинг 18.74. Пример присваивания значения свойству KeepIdentity

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.KeepIdentity = False
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Наконец, еще одно предостережение, которое следует учитывать, состоит в том, что свойство KeepIdentity представляет собой чисто бинарную опцию: оно может быть либо задано, либо не задано. Значение, присвоенное указанному свойству, влияет на все объекты, в которые осуществляется вставка строк с помощью компонента XML Bulk Load в данном конкретном сеансе массовой загрузки. Это означает, что невозможно сохранить значения identity для одних таблиц и позволить программе SQL Server вырабатывать эти значения для других таблиц.

Значения NULL

Если для какого-то столбца не предусмотрено отображение в схеме, то происходит вставка значения столбца, применяемого по умолчанию. С другой стороны, если для столбца не предусмотрено значение по умолчанию, то происходит вставка NULL. Если же в столбце не поддерживаются значения NULL, то массовая загрузка останавливается и формируется сообщение об ошибке.

Свойство KeepNulls позволяет передать средствам массовой загрузки указание, что в столбец следует вставлять значения NULL, а не значения, предусмотренные для столбца по умолчанию, если столбец не отображен в схеме. Пример использования свойства KeepNulls приведен в листинге 18.75.

Листинг 18.75. Пример использования свойства KeepNulls

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
```

```
  "Integrated Security=SSPI;"
objBulkLoad.KeepNulls = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Блокировка таблиц

Как и при использовании других средств массовой загрузки программы SQL Server, можно выполнить настройку конфигурации объекта SQLXMLBulkLoad таким образом, чтобы этот объект блокировал всю целевую таблицу, прежде чем начать загрузку в нее данных. Такой способ загрузки является более эффективным и быстрым по сравнению со способом, в котором применяются более детализированные блокировки, но имеет один недостаток, связанный с тем, что исключается доступ к таблице другим пользователям на протяжении всего сеанса массовой загрузки. Чтобы вынудить сервер заблокировать таблицу во время массовой загрузки XML, необходимо присвоить свойству ForceTableLock значение True, как показано в листинге 18.76.

Листинг 18.76. Пример использования свойства ForceTableLock

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.ForceTableLock = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Транзакции

По умолчанию операции массовой загрузки XML не являются транзакционными; это означает, что при возникновении ошибки во время процесса загрузки строки, загруженные вплоть до этого момента, остаются в базе данных. Такой способ организации работы обеспечивает самое высокое быстродействие, но имеет один недостаток, связанный с тем, что в случае ошибки таблица может остаться в частично загруженном состоянии. Чтобы вынудить сервер выполнять операцию массовой загрузки в виде одной транзакции, необходимо присвоить свойству Transaction объекта SQLXMLBulkLoad значение True, прежде чем вызвать метод Execute.

Если свойство Transaction равно True, то все данные, подготовленные для вставки, кэшируются во временном файле, прежде чем произойдет их загрузка в базу данных SQL Server. Для контроля над тем, в каком месте должен быть записан этот файл, можно задавать значение свойства TempFilePath. Свойство TempFilePath имеет смысл, только если свойство Transaction равно True. Если же свойство Transaction равно True, а свойству TempFilePath не присвоено значение,

то по умолчанию в качестве каталога для размещения временного файла используется каталог, указанный с помощью переменной среды TEMP, заданной на сервере.

Следует отметить, что массовая загрузка данных в составе транзакции происходит гораздо медленнее по сравнению с загрузкой вне транзакции. Именно поэтому в компоненте массовой загрузки не предусмотрено по умолчанию использование транзакции для загрузки данных. Необходимо также учитывать, что в составе транзакции нельзя выполнять массовую загрузку двоичных данных XML.

Пример выполнения массовой загрузки с помощью транзакции приведен в листинге 18.77.

Листинг 18.77. Пример выполнения массовой загрузки с помощью транзакции

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.Transaction = True
objBulkLoad.TempFilePath = "c:\temp\xmlswap"
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

В данном примере объект SQLXMLBulkLoad устанавливает свое собственное соединение с сервером с помощью интерфейса OLE DB, поэтому действует в рамках своего собственного контекста транзакции. Если в ходе массовой загрузки происходит ошибка, то компонент массовой загрузки выполняет откат своей собственной транзакции.

Если же объект SQLXMLBulkLoad подключается к существующему соединению OLE DB с помощью своего свойства ConnectionCommand, то контекст транзакции принадлежит этому соединению и управляется клиентским приложением. В таком случае после завершения массовой загрузки в клиентском приложении необходимо явно выполнить фиксацию или откат транзакции. Пример использования объектом SQLXMLBulkLoad существующего соединения OLE DB приведен в листинге 18.78.

Листинг 18.78. Пример использования объектом SQLXMLBulkLoad существующего соединения OLE DB

```
On Error Resume Next
Err.Clear
Set objCmd = CreateObject("ADODB.Command")
objCmd.ActiveConnection= _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.Transaction = True
objBulkLoad.ConnectionCommand = objCmd
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
If Err.Number = 0 Then
  objCmd.ActiveConnection.CommitTrans
Else
```

```
  objCmd.ActiveConnection.RollbackTrans
End If
Set objBulkLoad = Nothing
Set objCmd = Nothing
```

Обратите внимание на то, что при использовании свойства ConnectionCommand обязательным является также применение свойства Transaction — этому свойству должно быть присвоено значение True.

Ошибки

Компонент XML Bulk Copy поддерживает регистрацию сообщений об ошибках в файле, указанном с помощью свойства ErrorLogFile объекта массового копирования. Указанный файл сам является документом XML, в котором регистрируются все ошибки, происшедшие во время массовой загрузки. Пример применения свойства ErrorLogFile показан в листинге 18.79.

Листинг 18.79. Пример применения свойства ErrorLogFile

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.ErrorLogFile = "c:\temp\xmlswap\errors.xml"
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Указанный файл после его записи будет содержать по одному элементу Record для каждой ошибки, которая возникла в ходе последнего сеанса массовой загрузки. Последнее по времени сообщение об ошибке будет указано первым.

Создание объектов схем базы данных

Компонент XML Bulk Copy способен не только загружать данные в существующие таблицы, но и автоматически создавать целевые таблицы, если они еще не существуют, либо удалять и вновь создавать, если они существуют. Для создания несуществующих таблиц необходимо присвоить свойству SchemaGen экземпляра компонента значение True, как показано в листинге 18.80.

Листинг 18.80. Пример использования свойства SchemaGen

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.SchemaGen = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

После того как свойству SchemaGen присваивается значение True, все таблицы в схеме, которые еще не существуют, будут созданы с началом массовой загрузки. Если же требуемые таблицы уже существуют, данные просто загружаются в них, как и при обычных условиях.

С другой стороны, если свойству BulkLoad экземпляра компонента присвоено значение False, то данные не загружаются. Итак, если свойство SchemaGen равно True, а BulkLoad равно False, то в базе данных создаются пустые таблицы, соответствующие тем таблицам схемы отображения, которые еще не существуют, но данные в созданные таблицы не загружаются. Пример использования свойств SchemaGen и BulkLoad для создания в базе данных пустых таблиц приведен в листинге 18.81.

Листинг 18.81. Пример использования свойств SchemaGen и BulkLoad для создания в базе данных пустых таблиц

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.SchemaGen = True
objBulkLoad.BulkLoad = False
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

При создании таблиц с помощью экземпляра компонента XML Bulk Load для определения столбцов в каждой таблице используется информация схемы отображения. Аннотация sql:datatype определяет типы данных столбцов, а атрибут dt:type дополнительно уточняет информацию о типе столбца. Для того чтобы определить в схеме отображения первичный ключ, необходимо присвоить атрибуту dt:type столбца значение id и присвоить свойству SGUseID экземпляра компонента XML Bulk Load значение True. Пример использования атрибута dt:type приведен в схеме отображения, показанной в листинге 18.82.

Листинг 18.82. Пример использования атрибута dt:type

```
<ElementType name="Orders" sql:relation="Orders">
  <AttributeType name="OrderID" sql:datatype="int" dt:type="id"/>
  <AttributeType name="ShipCity" sql:datatype="nvarchar(30)"/>

  <attribute type="OrderID" sql:field="OrderID"/>
  <attribute type="ShipCity" sql:field="ShipCity"/>
</ElementType>
```

В листинге 18.83 показан код VBScript, в котором задается свойство SGUseID, позволяющее автоматически определить первичный ключ для таблицы, которая создается на сервере.

Листинг 18.83. Пример использования свойства SGUseID

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.SchemaGen = True
objBulkLoad.SGUseID = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Ниже приведен код Transact-SQL, который формируется при выполнении сеанса массовой загрузки.

```
CREATE TABLE Orders
(
  OrderID int NOT NULL,
  ShipCity nvarchar(30) NULL,
  PRIMARY KEY CLUSTERED (OrderID)
)
```

Объект SQLXMLBulkLoad способен не только создавать новые таблицы по определениям, содержащимся в схеме отображения, но также удалять и снова создавать таблицы. Для того чтобы экземпляр компонента массовой загрузки удалял и вновь создавал таблицы, определения которых отображены в схеме, необходимо присвоить свойству SGDropTables значение True, как показано в листинге 18.84.

Листинг 18.84. Пример использования свойства SGDropTables

```
Set objBulkLoad = CreateObject("SQLXMLBulkLoad.SQLXMLBulkLoad")
objBulkLoad.ConnectionString = _
  "provider=SQLOLEDB;data source=KUFNATHE;database=Northwind;" & _
  "Integrated Security=SSPI;"
objBulkLoad.SchemaGen = True
objBulkLoad.SGDropTables = True
objBulkLoad.Execute "d:\xml\OrdersSchema.xml",
    "d:\xml\OrdersData.xml"
Set objBulkLoad = Nothing
```

Управляемые классы

Средства SQLXML предоставляют классы управляемого кода, которые позволяют осуществлять выборку данных XML из базы данных SQL Server (преобразование данных в код XML может выполняться на серверном или на клиентском компьютере). Эти классы имеют аналогии в самой инфраструктуре .NET Framework, но в большей степени пригодны для использования со средствами SQLXML и предоставляют доступ к своим уникальным функциональным возможностям в рамках приложений с управляемым кодом. Классы SQLXML находятся в сборке Microsoft.Data.SqlXml и доступ к ним, как и при использовании любой другой сборки с управляемым кодом, может быть получен из приложений, написанных

на любом языке, совместимом с CLR (Common Language Runtime – общая среда выполнения), включая C#, Visual Basic.NET, Delphi.NET и другие.

Основными классами управляемого кода в сборке SqlXml являются классы SqlXmlCommand, SqlXmlParameter и SqlXmlAdapter. Как было указано выше, эти классы подобны своим аналогам в инфраструктуре .NET Framework, имеющим похожие имена. Класс SqlXmlCommand используется для выполнения команд T-SQL или вызова на выполнение процедурных объектов SQL Server и предусматривает возможность возвращать полученные результаты выполнения в виде кода XML. Класс SqlXmlParameter применяется для подготовки к выполнению параметризованных запросов. Класс SqlXmlAdapter служит для обработки результатов выполнения, полученных с помощью класса SqlXmlCommand. Если основополагающий источник данных поддерживает возможность внесения модификаций, то изменения можно вносить в клиентской программе и снова передавать на сервер, используя для инкапсуляции операций модификации данных так называемые шаблоны определения различий (diffgram), которые представляют собой специализированные шаблоны, подобные шаблонам обновления и применяемые в инфраструктуре .NET Framework.

Наилучший способ понять, как осуществляется взаимодействие этих классов в реальном приложении, состоит в создании такого приложения. В следующем примере приведен код C#, который демонстрирует использование каждого из основных управляемых классов SQLXML для выполнения хранимой процедуры и обработки полученного с ее помощью результирующего набора. Начнем с изучения исходного кода хранимой процедуры (листинг 18.85).

Листинг 18.85. Исходный код хранимой процедуры ListCustomers

```
USE Northwind
GO
DROP PROC ListCustomers
GO
CREATE PROC ListCustomers @CustomerID nvarchar(10)='%'
AS
PRINT '@CustomerID = ' +@CustomerID

SELECT *
FROM Customers
WHERE CustomerID LIKE @CustomerID

RAISERROR('%d Customers', 1,1, @@ROWCOUNT)
GO
EXEC ListCustomers N'ALFKI'
```

Хранимая процедура, приведенная в листинге 18.85, принимает один параметр (маску идентификатора заказчика) и выводит все строки таблицы Customers базы данных Northwind, которые соответствуют этой маске. В листинге 18.86 показан код C#, в котором используются управляемые классы SQLXML для выполнения указанной хранимой процедуры (этот код можно найти в подкаталоге CH18\managed_classes компакт-диска, прилагаемого к данной книге).

Листинг 18.86. Пример использования хранимой процедуры, приведенной в листинге 18.85

```
using System;
using Microsoft.Data.SqlXml;
using System.IO;
using System.Xml;
class CmdExample
{
  static string strConn = "Provider=SQLOLEDB;Data Source='(local)';
      database=Northwind; Integrated Security=SSPI";
  public static int CmdExampleWriteXML()
  {
    XmlReader Reader;
    SqlXmlParameter Param;
    XmlTextWriter TxtWriter;

    // Создать новый экземпляр объекта SqlXmlCommand
    SqlXmlCommand Cmd = new SqlXmlCommand(strConn);
    // Выполнить настройку конфигурации объекта, чтобы с его помощью
    // можно было вызвать хранимую процедуру
    Cmd.CommandText = "EXEC ListCustomersXML ?";

    // Создать параметр и присвоить ему значение
    Param = Cmd.CreateParameter();
    Param.Value = "ALFKI";

     // Выполнить процедуру
    Reader = Cmd.ExecuteXmlReader();

    // Создать новый экземпляр объекта XmlTextWriter для вывода данных
    // на терминал
    TxtWriter = new XmlTextWriter(Console.Out);

    // Перейти к корневому элементу
    Reader.MoveToContent();

    // Вывести документ на терминал
    TxtWriter.WriteNode(Reader, false);

    // Выполнить сброс данных на внешнее устройство в объекте записи и
    // закрыть объект чтения
     TxtWriter.Flush();
    Reader.Close();

     return 0;
  }
  public static int Main(String[] args)
  {
    CmdExampleWriteXML();
     return 0;
  }
}
```

Обратите внимание на то, как используется ссылка на сборку `Microsoft.Data.SqlXml`. Для того чтобы иметь возможность выполнить компиляцию и редактирование связей для рассматриваемого кода, необходимо добавить

ссылку на указанную сборку в интегрированной среде разработки Visual Studio .NET (или указать ее в командной строке `csc.exe`).

Рассмотрим, как работает код, приведенный в листинге 18.86. Выполнение этого кода начинается с создания нового экземпляра `SqlXmlCommand` и передачи ему заданной строки соединения. Затем свойству `CommandText` объекта `SqlXmlCommand` присваивается значение, обеспечивающее вызов хранимой процедуры с помощью заменяемого параметра. После этого создается экземпляр объекта `SqlXmlParameter` и присваивается значение свойству `Value` этого объекта для задания значения, используемого в качестве параметра хранимой процедуры.

После выполнения подготовки объекта `SqlXmlCommand` должным образом вызывается метод `ExecuteXmlReader` этого объекта. Метод возвращает экземпляр объекта `XmlReader`, который может использоваться для обработки результатов хранимой процедуры. После этого создается объект `XmlTextWriter`, позволяющий выводить данные XML, возвращаемые объектом `SqlXmlCommand`. Дальнейшие действия выполняются путем перемещения в начало самого документа (с помощью вызова метода `MoveToContent`) с последующим выводом всего документа на терминал с помощью вызова `TxtWriter.WriteNode`. Наконец, выполняются заключительные действия; при этом происходит вывод оставшихся данных из объекта `XmlTextWriter` и закрытие объекта `XmlReader`, который был первоначально возвращен в результате вызова метода `SqlXmlCommand.ExecuteXmlReader`.

Для тех разработчиков, которые выполнили большой объем программирования с использованием классов ADO.NET и XML инфраструктуры .NET Framework, рассматриваемый код может показаться весьма знакомым. Все три управляемых класса SQLXML имеют аналоги в самой инфраструктуре .NET Framework. Остаются также аналогичными принципы использования этих классов. Управляемые классы SQLXML там, где это оправдано, возвращают данные, типы которых являются совместимыми с базовыми классами инфраструктуры .NET Framework, поэтому могут применяться как взаимозаменяемые с классами .NET. Но назначение управляемых классов SQLXML состоит в дополнении классов ADO.NET в целях предоставления функциональных возможностей, характерных для SQLXML, а не в замене классов .NET, а также не в создании альтернатив этим классам.

Поддержка службы Web SQLXML (SOAP)

Средства поддержки службы Web SQLXML позволяют предоставлять доступ к программе SQL Server как к службе Web. В результате этого появляется возможность вызывать на выполнение хранимые процедуры и другие процедурные объекты, а также шаблоны запросов, как если бы они представляли собой методы, доступ к которым предоставляется с помощью традиционной службы Web, основанной на протоколе SOAP. Средства SQLXML включают все необходимое для обеспечения доступа к данным SQL Server по протоколу SOAP из любой компьютерной платформы или клиентской программы, позволяющей передавать запросы SOAP.

Преимущество такой организации работы, безусловно, состоит в том, что для выполнения запросов и доступа к объектам SQL Server не требуется клиентское программное обеспечение SQL Server. Это означает, что возможность передачи

запросов и получения результатов из программы SQL Server обеспечивается и для приложений на таких клиентских платформах, которые непосредственно не поддерживаются программой SQL Server (например, Linux), с помощью средств SQLXML и входящего в их состав средства SOAP.

Настройка программы SQL Server для того, чтобы она выглядела как служба Web, осуществляется путем введения в конфигурацию виртуального имени SOAP с помощью инструментального средства IIS Virtual Directory Management (это инструментальное средство можно найти с помощью команды меню SQLXML | Configure IIS, раскрыв меню Start | Programs). Виртуальное имя SOAP — это просто каталог, связанный с именем виртуального каталога IIS, в качестве типа которого задано значение soap. Желаемое имя службы может быть указано в текстовом поле Web Service Name; обычно принято задавать имя soap. После задания этого виртуального имени выполняется настройка конфигурации конкретных объектов SQL Server, доступ к которым должен предоставляться службой Web. Для этого нужно щелкнуть на кнопке Configure вкладки Virtual Names и выбрать имя объекта, затем формат кода XML, который должен вырабатываться объектами среднего яруса (с помощью интерфейса SQLISAPI), наконец, задать способ предоставления доступа к объекту. В качестве такого способа можно указать коллекцию элементов XML, один объект `Dataset` или коллекцию объектов `Datasets`. Как будет показано в упражнении, к выполнению которого мы вскоре приступим, доступ к любому серверному объекту может предоставляться и с помощью многочисленных способов, если клиентское приложение поддерживает широкий набор способов взаимодействия с программой SQL Server по протоколу SOAP.

С точки зрения архитектуры программного обеспечения следует отметить, что функциональные возможности SOAP в составе средств SQLXML предоставляются с помощью расширения ISAPI этих средств — интерфейса SQLISAPI. Функциональные возможности поддержки протокола SOAP представляют собой расширение понятия виртуального каталога, конфигурацию которого можно настраивать для получения доступа к серверу с помощью запросов URL и шаблонов. Виртуальное имя SOAP, подготовка которого осуществляется в процессе настройки, предоставляет доступ к службе Web средств SQLXML с помощью URL. Поэтому виртуальное имя SOAP позволяет любому клиентскому приложению, способному к взаимодействию по протоколу SOAP, обращаться с помощью соответствующего URL к объектам SQL Server, как если бы эти объекты относились к любой другой службе Web. К процедурным объектам и шаблонам XML программы SQL Server могут получать доступ приложения Java, традиционные приложения ADO и, безусловно, приложения .NET без необходимости использовать традиционное клиентское программное обеспечение SQL Server или взаимодействовать с помощью службы TDS.

В следующем упражнении рассматривается порядок действий по предоставлению доступа к программе SQL Server как к службе Web, а затем показано, каким образом воспользоваться услугами этой службы в приложении C#. Вначале подготавливается виртуальное имя SOAP, затем выполняется настройка конфигурации процедурного объекта SQL Server в целях последующего предоставления доступа к методам этого объекта как коллекции методов службы Web. Наконец, будет создано небольшое приложение, которое пользуется услугами указанной службы и демонстрирует способы взаимодействия с этой службой.

Упражнение

Упражнение 18.4. Создание службы Web SQLXML и использование ее услуг

1. В созданном ранее каталоге `\inetpub\wwwroot\Northwind` создайте подкаталог `Soap`.
2. Вызовите на выполнение инструментальное средство IIS Virtual Directory Management for SQLXML, которое уже применялось в данной книге для настройки виртуального каталога `Northwind`.
3. Перейдите на вкладку Virtual Names и введите новое виртуальное имя, указав `soap` в качестве значений полей Name, Type и Web Service. Задайте в качестве значения пути имя каталога, созданного в п. 1.
4. Сохраните конфигурацию виртуального имени. К этому моменту кнопка Configure должна быть уже доступна. Щелкните на этой кнопке, чтобы приступить к предоставлению доступа к конкретным процедурным объектам и шаблонам с помощью службы Web.
5. Щелкните на кнопке со знаком троеточия справа от текстового поля SP/Template и выберите из раскрывшегося списка хранимую процедуру `ListCustomers`.
6. Присвойте предоставляемому методу имя `ListCustomers`, установите для него формат строки `Raw` и формат вывода `XML objects`, затем щелкните на кнопке OK.
7. Повторите процесс и назовите новый метод `ListCustomersAsDataset` (в качестве ссылки будет применяться хранимая процедура `ListCustomers`). Укажите в качестве типа выходных данных метода значение `Single dataset`, затем щелкните на кнопке OK.
8. Еще раз повторите этот процесс и назовите новый метод `ListCustomersAsDatasets`. Укажите в качестве типа выходных данных метода значение `Dataset`, затем щелкните на кнопке OK. Выполнение указанных выше действий равносильно предоставлению доступа к хранимой процедуре `ListCustomers` как к трем различным методам службы Web с использованием трех разных форматов вывода. Обратите внимание на то, что процедурные объекты, настройка которых выполняется таким образом, сами не должны возвращать код XML (т.е. в этих объектах не должна применяться опция `FOR XML` языка Transact-SQL), поскольку форматирование XML осуществляется исключительно средствами среднего яруса на основе интерфейса SQLISAPI, если используется средство службы Web SQLXML.
9. Создайте новый проект приложения Windows на языке C# в среде Visual Studio .NET. Создаваемое приложение должно дать возможность вызывать средство службы Web SQLXML для выполнения хранимой процедуры `ListCustomers` с использованием заданной маски идентификатора заказчика `CustomerID`.
10. Введите один элемент управления `TextBox` в левом верхнем углу применяемой по умолчанию формы, который будет использоваться в качестве поля ввода для маски `CustomerID`.

11. Введите в форму элемент управления `Button` справа от элемента управления `TextBox` для вызова на выполнение метода службы Web.

12. Справа от кнопки `Button` введите три элемента управления `RadioButton`, с помощью которых можно будет указывать, какой метод Web требуется вызвать на выполнение. Назовите первый элемент управления `rbXMLElements`, второй — `rbDataset` и третий — `rbDatasetObjects`. Введите в качестве свойства `Text` каждого элемента управления краткое описание соответствующего этому элементу управления метода Web (например, свойство `Text` элемента управления `rbXMLElements` должно представлять собой нечто подобное текстовому описанию "`XML Elements`" — элементы XML).

13. Введите в форму элемент управления `ListBox`, который будет находиться ниже всех остальных элементов управления. Этот элемент управления будет использоваться для отображения выходных данных, полученных после вызова методов службы Web. Выполните привязку элемента управления `ListBox` к нижнему краю формы и обязательно откорректируйте его размер таким образом, чтобы он занимал большую часть формы.

14. Убедитесь в том, что используемый экземпляр сервера IIS является функционирующим и доступным. Как и в других примерах данной главы, предназначенных для использования в Web, предполагается, что читатель имеет собственный экземпляр сервера IIS, и этот экземпляр работает на локальном компьютере.

15. Щелкните правой кнопкой мыши на разрабатываемом решении в программе Solution Explorer и выберите команду Add Web Reference (Добавить ссылку Web). В URL, относящемся к ссылке Web, введите следующее:

    ```
    http://localhost/Northwind/soap?wsdl
    ```

    Этот URL ссылается по имени на виртуальный каталог, созданный ранее, затем ссылается на имя виртуального каталога `soap`, созданного под указанным виртуальным каталогом, и, наконец, на функциональные средства языка WSDL (Web Services Description Language — язык описания служб Web), предоставляемые с помощью интерфейса SQLISAPI. Как было указано выше, вопросительный знак в URL обозначает начало параметров URL, поэтому слово `wsdl` передается в расширение DLL интерфейса SQLISAPI в виде параметра. По аналогии с языком XML и протоколом SOAP язык WSDL определяется отдельным стандартом консорциума W3C и описывает службы Web с помощью языка XML как множество конечных точек соединений, оперирующих сообщениями, которые содержат либо процедурную, либо документальную информацию. С дополнительными сведениями о языке WSDL можно ознакомиться, открыв на Web-узле W3C следующую ссылку:

    ```
    http://www.w3.org/TR/wsdl
    ```

16. После добавления указанной ссылки Web на виртуальный каталог `soap` служба Web хоста `localhost` становится доступной для использования в приложении. В каталоге приложения создается класс-посредник, который имеет информацию о том, как должно осуществляться взаимодействие со службой Web, указанной в ссылке. В коде приложения этот класс-посредник выглядит как идентичный действительной службе Web. После выполнения вызовов к методам этого класса прозрачно для пользователя осуществляется маршалинг этих вызовов и передача в саму службу Web, которая может быть

развернута на каком-то другом компьютере, находящемся в любом месте локальной внутренней сети или общедоступной сети Internet. Как было сказано в главе 6, средство RPC операционной системы Windows действует во многом по такому же принципу. Службы Web, по существу, являются просто расширенной реализацией подхода, основанного на использовании дистанционного вызова процедур (RPC). Пользовательское приложение работает и взаимодействует с локальными классами и методами, а средства, прозрачные для приложения, обеспечивают обмен данными с действительной реализацией службы так, что в приложении вообще не учитывается тот факт, что в нем используется определенный тип удаленного ресурса.

17. Дважды щелкните на элементе управления `Button`, введенном ранее, и введите в определение этого элемента управления код, представленный в листинге 18.87.

Листинг 18.87. Код элемента управления `Button`

```
int iReturn = 0;
object result;
object[] results;
System.Xml.XmlElement resultElement;
System.Data.DataSet resultDS;
localhost.soap proxy = new localhost.soap();
proxy.Credentials=System.Net.CredentialCache.DefaultCredentials;

// Выполнить возврат объектов ListCustomers после приведения к типу
// данных XMLElements
if (rbXMLElements.Checked)
{
  listBox1.Items.Add("Executing ListCustomers...");
  listBox1.Items.Add("");

  results = proxy.ListCustomers(textBox1.Text);

  for (int j=0; j<results.Length; j++)
  {
    localhost.SqlMessage errorMessage;
    result= results[j];
    if (result.GetType().IsPrimitive)
     {
      listBox1.Items.Add(
         string.Format("ListCustomers return value: {0}", result));
     }
    if (result is System.Xml.XmlElement)
     {
      resultElement = (System.Xml.XmlElement) results[j];
      listBox1.Items.Add(resultElement.OuterXml);
     }
     else if (result is localhost.SqlMessage) {
       errorMessage = (localhost.SqlMessage) results[j];
      listBox1.Items.Add(errorMessage.Message);
       listBox1.Items.Add(errorMessage.Source);
     }
  }
  listBox1.Items.Add("");
}
```

```
// Выполнить возврат объектов ListCustomers в качестве объектов Dataset
else if (rbDatasetObjects.Checked)
{
  listBox1.Items.Add("Executing ListCustomersAsDatasets...");
  listBox1.Items.Add("");
  results = proxy.ListCustomersAsDatasets(textBox1.Text);

  for (int j=0; j<results.Length; j++)
  {
    localhost.SqlMessage errorMessage;
    result= results[j];

     if (result.GetType().IsPrimitive)
     {
      listBox1.Items.Add(
         string.Format("ListCustomers return value: {0}", result));
     }
    if (result is System.Data.DataSet)
     {
      resultDS = (System.Data.DataSet) results[j];
      listBox1.Items.Add("DataSet " +resultDS.GetXml());
     }
    else if (result is localhost.SqlMessage)
     {
       errorMessage = (localhost.SqlMessage) results[j];
      listBox1.Items.Add("Message " +errorMessage.Message);
      listBox1.Items.Add(errorMessage.Source);
    }
  }
  listBox1.Items.Add("");
}
// Выполнить возврат объектов ListCustomers в виде объекта Dataset
else if (rbDataset.Checked)
{
  listBox1.Items.Add("Executing ListCustomersAsDataset...");
  listBox1.Items.Add("");
  resultDS = proxy.ListCustomersAsDataset(textBox1.Text,
      out iReturn);
  listBox1.Items.Add(resultDS.GetXml());
  listBox1.Items.Add(
    string.Format("ListCustomers return value: {0}", iReturn));
  listBox1.Items.Add("");
}
```

Этот код можно разделить на три основные процедуры — по одной на каждый из трех вызываемых методов службы Web. Изучите код, относящийся к формату вывода каждого типа, затем сравните и сопоставьте аналоги и различия в этом коде. Обратите внимание на то, что в этом коде используется принцип взаимно-однозначного соответствия для определения того, какого типа объекты будут получены в ответ на вызовы службы Web в тех ситуациях, когда возможно применение нескольких типов.

18. Откомпилируйте приложение и вызовите его на выполнение. Опробуйте все три формата вывода и попытайтесь применить различные маски `CustomerID`. После каждого щелчка на элементе управления `Button` происходят описанные ниже действия.

18.1. В коде выполняется вызов метода из класса-посредника Visual Studio .NET, добавленного к проекту после введения ссылки Web на службу Web SQLXML, действующую по протоколу SOAP, которая была подготовлена для работы с базой данных `Northwind`.

18.2. В коде службы Web инфраструктуры .NET вызов метода преобразуется в вызов SOAP и передается по сети на указанный хост-компьютер. В данном случае хост службы Web, по-видимому, находится на том же компьютере, на котором функционирует клиентское приложение, но рассматриваемая архитектура позволяет разместить хост на любом узле локальной внутренней сети или общедоступной сети Internet.

18.3. Расширение ISAPI средств SQLXML получает вызов SOAP и преобразует его в вызов хранимой процедуры `ListCustomers`, которая находится в базе данных `Northwind`, указанной с помощью виртуального каталога IIS.

18.4. Программа SQL Server вызывает процедуру на выполнение и возвращает ее результаты в виде набора строк в интерфейс SQLISAPI.

18.5. Интерфейс SQLISAPI преобразует набор строк в соответствующий формат XML или в формат объекта, в зависимости от того способа, который был задан при настройке конфигурации метода службы Web, затем возвращает по протоколу SOAP в код службы Web инфраструктуры .NET Framework, работающий на клиентском компьютере.

18.6. Код службы Web инфраструктуры .NET Framework преобразует полученное сообщение SOAP в соответствующие объекты и коды результатов, которые затем возвращает в приложение.

18.7. После этого в приложении используются дополнительные вызовы методов для извлечения возвращенной информации в виде текста, и полученный текст выводится в элемент управления `ListBox`.

Таким образом, выше в общих чертах описано, как используются средства SOAP в составе средств SQLXML для доступа к программе SQL Server по протоколу SOAP. Как уже было сказано, одной из очевидных областей применения этой технологии является обеспечение возможности использовать программу SQL Server в качестве одного из представителей служб Web. При этом сервер взаимодействует с другими службами Web, не требуя установки фирменного клиентского программного обеспечения или использования лишь поддерживаемых операционных систем. Благодаря средству поддержки службы Web из состава средств SQLXML любое приложение, способное к обмену данными по протоколу SOAP, получает возможность обращаться к программе SQL Server. Поддержка службы Web в составе средств SQLXML — это давно ожидаемое и очень мощное дополнение к семейству технологий программы SQL Server.

Ограничения средств SQLXML

Поддержка XML в программе SQL Server характеризуется некоторыми фундаментальными ограничениями, которые затрудняют использование этих средств в определенных ситуациях. В настоящем разделе рассматривается ряд указанных ограничений и описаны способы их преодоления.

Процедура sp_xml_concat

Исходя из того, что процедура sp_xml_preparedocument принимает для обработки текст документов фактически любой длины (вплоть до 2 Гбайт), можно предположить, что средства XML программы SQL Server также вполне должны обладать способностью обрабатывать длинные документы, но это предположение не оправдывается. Хотя формальный параметр xmltext процедуры sp_xml_preparedocument принимает фактические параметры не только типа text, но и типа varchar, язык Transact-SQL не поддерживает локальные текстовые переменные. В наибольшей степени для поддержки локальных текстовых переменных в языке Transact-SQL подходит способ, предусматривающий применение процедуры с параметром text. Тем не менее этот параметр не может присваиваться и не может служить для присваивания ему текстовых данных, возвращенных командой READTEXT. Почти единственный выход из этой ситуации состоит в том, чтобы выполнить вставку полученных данных в таблицу.

Указанная проблема особенно заметно проявляется при попытке сохранить большой документ XML в таблице и обработать его с помощью процедуры sp_xml_preparedocument. Но после загрузки документа в таблицу возникает вопрос — как извлечь его, чтобы передать в процедуру sp_xml_preparedocument. К сожалению, не существует легкого способа решения этой задачи. Поскольку мы не имеем возможности объявлять локальные текстовые переменные, почти единственное решение, которое может быть принято, состоит в разбивке текста документа на фрагменты с длиной 8 тысяч байтов и присваивания этих фрагментов нескольким переменным varchar, с последующим использованием конкатенации параметров перед вызовом процедуры sp_xml_preparedocument. Эта задача является на удивление сложной, поэтому автор написал хранимую процедуру, избавляющую пользователя от необходимости решать ее самому. Такая процедура называется sp_xml_concat и может использоваться для обработки больших документов XML, хранящихся в столбцах типа text, varchar или char таблицы.

Процедура sp_xml_concat принимает три параметра: имена таблицы и столбца, в которых находится документ, а также выходной параметр, который возвращает дескриптор документа, сформированный процедурой sp_xml_preparedocument. Дескриптор, возвращенный процедурой sp_xml_concat, можно использовать для работы с функцией OPENXML и процедурой sp_xml_unpreparedocument.

В качестве параметра table с обозначением имени таблицы может быть задано имя действительной таблицы или представления либо запрос Transact-SQL, заключенный в круглые скобки, который рассматривается как производная таблица. Возможность задавать производную таблицу позволяет применять критерии выборки к таблице, передаваемой в эту процедуру. Поэтому, если есть необходимость обработать конкретную строку в таблице или иным образом лимитировать представление таблицы, передаваемой в процедуру, это можно сделать с помощью выражения производной таблицы.

В листинге 18.88 показан полный исходный код процедуры sp_xml_concat.

Листинг 18.88. Полный исходный код процедуры sp_xml_concat

```
USE master
GO
IF OBJECT_ID('sp_xml_concat','P') IS NOT NULL
  DROP PROC sp_xml_concat
GO
CREATE PROC sp_xml_concat
  @hdl int OUT,
  @table sysname,
  @column sysname
AS
EXEC('
SET TEXTSIZE 4000
DECLARE
  @cnt int,
  @c nvarchar(4000)
DECLARE
  @declare varchar(8000),
  @assign varchar(8000),
  @concat varchar(8000)

SELECT @c = CONVERT(nvarchar(4000),'+@column+') FROM '+@table+'

SELECT @declare = ''DECLARE'',
       @concat = '''',
       @assign = '''',
       @cnt = 0
WHILE (LEN(@c) > 0) BEGIN
  SELECT @declare = @declare + '' @c''+CAST(@cnt as nvarchar(15))
      +''nvarchar(4000),'',
    @assign = @assign + ''SELECT @c''+CONVERT(nvarchar(15),@cnt)
       +''= SUBSTRING(' + @column+','' + CONVERT(nvarchar(15),
       1+@cnt*4000)+ '', 4000) FROM '+@table+' '',
    @concat = @concat + ''+@c''+CONVERT(nvarchar(15),@cnt)
  SET @cnt = @cnt+1
  SELECT @c = CONVERT(nvarchar(4000),SUBSTRING('+@column+',
      1+@cnt*4000,4000)) FROM '+@table+'
END

IF (@cnt = 0) SET @declare = ''''
ELSE SET @declare = SUBSTRING(@declare,1,LEN(@declare)-1)

SET @concat = @concat + ''+''''''''''''''

EXEC(@declare+'' ''+@assign+'' ''+
''EXEC(
''''DECLARE @hdl_doc int
  EXEC sp_xml_preparedocument @hdl_doc OUT, ''+@concat+''
    DECLARE hdlcursor CURSOR GLOBAL FOR SELECT @hdl_doc AS
       DocHandle''''')''
)
')
OPEN hdlcursor
FETCH hdlcursor INTO @hdl
DEALLOCATE hdlcursor
GO
```

В этой процедуре динамически вырабатываются операторы DECLARE И SELECT, необходимые для разбивки большого текстового столбца на фрагменты типа nvarchar(4000), например, DECLARE @c1 nvarchar(4000) SELECT @c1= .... По мере выполнения этих действий процедура вырабатывает также выражение конкатенации, которое включает все эти переменные (например, @c1+@c2+@c3, ...). Поскольку функция EXEC() поддерживает операцию конкатенации строк с размером вплоть до 2 Гбайт, это выражение конкатенации динамически передается в функцию EXEC(), после чего данная функция оперативно выполняет конкатенацию. При этом, по существу, реконструируется документ, извлеченный из таблицы. Затем конкатенированная строка передается в процедуру sp_xml_preparedocument на обработку. Конечным результатом становится дескриптор документа, который можно использовать в функции OPENXML. Соответствующий пример приведен в листинге 18.89 (полный текст этого проверочного запроса приведен в подкаталоге CH18 компакт-диска, прилагаемого к данной книге).

Листинг 18.89. Запрос, в котором используется процедура sp_xml_concat (в сокращенном виде)

```
USE Northwind
GO
CREATE TABLE xmldoc
(id int identity,
  doc text)
INSERT xmldoc VALUES('<Customers>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
  <Order CustomerID="VINET" EmployeeID="5" OrderDate=
       "1996-07-04T00:00:00">
     <OrderDetail OrderID="10248" ProductID="11" Quantity="12"/>
     <OrderDetail OrderID="10248" ProductID="42" Quantity="10"/>
// Здесь изъята часть кода ...
  </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos GOnzlez">
  <Order CustomerID="LILAS" EmployeeID="3" OrderDate=
       "1996-08-16T00:00:00">
    <OrderDetail OrderID="10283" ProductID="72" Quantity="3"/>
  </Order>
</Customer>
</Customers>')

DECLARE @hdl int
EXEC sp_xml_concat @hdl OUT, '(SELECT doc FROM xmldoc WHERE id=1)
    a', 'doc'

SELECT * FROM OPENXML(@hdl, '/Customers/Customer') WITH
    (CustomerID nvarchar(50))

EXEC sp_xml_removedocument @hdl
SELECT DATALENGTH(doc) from xmldoc
GO
DROP TABLE xmldoc

(Результаты)
CustomerID
```

```
-----------------------------------------------
VINET
LILAS

-----------
36061
```

Автор сократил документ XML, приведенный в проверочном запросе, который применяется в листинге 18.89. А запрос, находящийся на компакт-диске, превышает по размерам 36 тысяч байтов, как показывает результат запроса `DATALENGTH()` в конце этого листинга.

В процедуру `sp_xml_concat` передается выражение производной таблицы наряду с именем столбца, который необходимо извлечь, а эта процедура выполняет все остальное. Она способна извлекать искомые узлы, даже если некоторые из таких узлов находятся в конце довольно большого документа.

Процедура sp_run_xml_proc

Еще одно ограничение поддержки XML в программе SQL Server обусловлено тем, что результаты XML не возвращаются в виде традиционных наборов строк. Возврат результатов XML в виде потоков ввода-вывода дает много преимуществ, но связан с некоторыми недостатками. Один из этих недостатков состоит в том, что невозможно вызвать хранимую процедуру, возвращающую результат XML, с помощью четырехкомпонентного имени или функции `OPENQUERY()` и получить результаты в удобном формате. Полученный набор будет представлять собой двоичный результирующий набор в нераспознаваемой форме, поскольку архитектура связанных серверов SQL Server не поддерживает потоки XML.

С аналогичными ограничениями приходится сталкиваться при попытке вставить результат запроса с конструкцией `FOR XML` в таблицу или заключить этот результат в переменную, поскольку программа SQL Server не позволяет сделать ни того, ни другого. Это связано с тем, что документы XML, возвращенные программой SQL Server, не являются традиционными наборами строк.

Для преодоления этого ограничения автор написал хранимую процедуру `sp_run_xml_proc`. Ее можно использовать для вызова хранимых процедур связанного сервера (такие процедуры должны находиться на связанном сервере), которые возвращают документы XML, а также локальные процедуры XML, результаты которых требуется сохранить в таблице или заключить в переменную. Процедура `sp_run_xml_proc` выполняет необходимые действия, открывая собственное соединение с сервером (в этой процедуре предполагается использование способа аутентификации Windows Authentication) и вызывая на выполнение переданную ей процедуру. После завершения работы пользовательской процедуры процедура `sp_run_xml_proc` обрабатывает возвращаемый пользовательской процедурой поток XML с помощью вызовов SQL-DMO, затем преобразует поток XML в традиционный набор строк и возвращает полученный набор строк. После этого результирующий набор можно вставлять в таблицу или подвергать дальнейшей обработке, наряду с любым другим результирующим набором. Исходный код процедуры `sp_run_xml_proc` приведен в листинге 18.90.

Листинг 18.90. Исходный код процедуры sp_run_xml_proc

```
USE master
GO
IF OBJECT_ID('sp_run_xml_proc','P') IS NOT NULL
  DROP PROC sp_run_xml_proc
GO
CREATE PROC sp_run_xml_proc
  @procname sysname  -- Процедура, вызываемая на выполнение
AS

DECLARE @dbname sysname,
  @sqlobject int,     -- Объект SQLServer
  @object int,    -- Вспомогательная переменная для доступа к
                  -- COM-объектам
  @hr int,    -- Содержит результат HRESULT, возвращенный COM-объектом
  @results int,     -- Объект QueryResults
  @msgs varchar(8000)    -- Сообщения Query

IF (@procname='/?') GOTO Help

-- Создать объект SQLServer
EXEC @hr=sp_OACreate 'SQLDMO.SQLServer', @sqlobject OUT
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @sqlobject, @hr
  RETURN
END

-- Выполнить настройку конфигурации объекта SQLServer на использование
-- соединения, заслуживающего доверия
EXEC @hr = sp_OASetProperty @sqlobject, 'LoginSecure', 1
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @sqlobject, @hr
  RETURN
END

-- Отменить режим использования префиксов ODBC в сообщениях
EXEC @hr = sp_OASetProperty @sqlobject, 'ODBCPrefix', 0
IF (@hr <> 0) BEGIN
  EXEC sp_displayoaerrorinfo @sqlobject, @hr
  RETURN
END

-- Открыть новое соединение (под этим подразумевается соединение,
-- заслуживающее доверия)
EXEC @hr = sp_OAMethod @sqlobject, 'Connect', NULL, @@SERVERNAME
IF (@hr <> 0) BEGIN
EXEC sp_displayoaerrorinfo @sqlobject, @hr
  RETURN
END

-- Получить указатель на коллекцию Databases объекта SQLServer
EXEC @hr = sp_OAGetProperty @sqlobject, 'Databases', @object OUT
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @sqlobject, @hr
  RETURN
```

```
END

-- Получить указатель на текущую базу данных из коллекции Databases
SET @dbname=DB_NAME()
EXEC @hr = sp_OAMethod @object, 'Item', @object OUT, @dbname
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  RETURN
END

-- Вызвать метод ExecuteWithResultsAndMessages2 объекта Database, чтобы
-- выполнить запуск процедуры
EXEC @hr = sp_OAMethod @object, 'ExecuteWithResultsAndMessages2',
    @results OUT, @procname, @msgs OUT
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  RETURN
END

-- Вывести на экран все сообщения, возвращенные процедурой
PRINT @msgs

DECLARE @rows int, @cols int, @x int, @y int, @col varchar(8000),
    @row varchar(8000)

-- Вызвать метод Rows объекта QueryResult, чтобы определить количество
-- строк в результирующем наборе
EXEC @hr = sp_OAMethod @results, 'Rows',@rows OUT
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  RETURN
END

-- Вызвать метод Columns объекта QueryResult, чтобы определить
-- количество столбцов в результирующем наборе
EXEC @hr = sp_OAMethod @results, 'Columns',@cols OUT
IF @hr <> 0 BEGIN
  EXEC sp_displayoaerrorinfo @object, @hr
  RETURN
END

DECLARE @table TABLE (XMLText varchar(8000))

-- Выполнять выборку одного за другим столбцов результирующего набора
-- с использованием метода GetColumnString
SET @y=1
WHILE (@y<=@rows) BEGIN
  SET @x=1
  SET @row=''
  WHILE (@x<=@cols) BEGIN
      EXEC @hr = sp_OAMethod @results, 'GetColumnString',
          @col OUT, @y, @x
      IF @hr <> 0 BEGIN
        EXEC sp_displayoaerrorinfo @object, @hr
        RETURN
      END
```

```
    SET @row=@row+@col+' '
    SET @x=@x+1
  END
  INSERT @table VALUES (@row)
  SET @y=@y+1
END

SELECT * FROM @table

EXEC sp_OADestroy @sqlobject     -- Применение этой команды - признак
                                 -- правильного стиля программирования

RETURN 0

Help:
PRINT 'You must specify a procedure name to run'
RETURN -1

GO
```

Хотя необходимость в создании отдельного соединения с сервером для преобразования документа нельзя назвать особенно приятной, к сожалению, в этом состоит единственный способ преодоления рассматриваемого в данном разделе ограничения без обращения к клиентской обработке (по крайней мере так обстоят дела в настоящее время). В листинге 18.91 приведен проверочный код, который показывает, как используется процедура `sp_run_xml_proc`.

Листинг 18.91. Пример использования процедуры `sp_run_xml_proc`

```
USE pubs
GO
DROP PROC testxml
GO
CREATE PROC testxml as
PRINT 'a message here'
SELECT * FROM pubs..authors FOR XML AUTO
GO
EXEC [TUK\PHRIP].pubs.dbo.sp_run_xml_proc 'testxml'

(Результаты приведены в сокращенном виде)

a message here
XMLText
----------------------------------------------------------------
<pubs..authors au_id="172-32-1176" au_lname="White" au_fname="John
<pubs..authors au_id="672-71-3249" au_lname="Yokomoto" au_fname="A
```

Автор значительно сократил в листинге 18.91 результирующий документ. Но после выполнения приведенного в этом листинге кода в программе Query Analyzer (и замены ссылки на связанный сервер, указанной в качестве примера, своей собственной ссылкой) будет обнаружено, что в качестве результирующего набора возвращается весь документ. Затем результирующий набор можно вставить в таблицу с помощью оператора `INSERT...EXEC` для дальнейшей обработки. В частности,

указанный метод можно использовать для присваивания возвращаемого документа переменной (в объеме вплоть до первых 8 тысяч байтов) или модифицировать его каким-то образом с помощью языка Transact-SQL. А после внесения в документ требуемых изменений можно вызвать процедуру sp_xml_concat (приведенную выше в данной главе), чтобы получить дескриптор документа, позволяющий выполнять запросы к этому документу с помощью функции OPENXML. В примере, приведенном в листинге 18.92, выполняются именно эти действия.

Листинг 18.92. Пример применения процедуры sp_run_xml_proc, функции OPENXML и процедуры sp_xml_concat

```
SET NOCOUNT ON
GO
USE pubs
GO
DROP PROC testxml
GO
CREATE PROC testxml as
SELECT au_lname, au_fname FROM authors FOR XML AUTO
GO

CREATE TABLE #XMLText1
(XMLText varchar(8000))
GO

-- Вставить документ XML в таблицу с использованием процедуры
-- sp_run_xml_proc
INSERT #XMLText1
EXEC sp_run_xml_proc 'testxml'

-- Присвоить содержимое документа переменной и добавить
-- корневой элемент
DECLARE @doc varchar(8000)
SET @doc=''
SELECT @doc=@doc+XMLText FROM #XMLText1
SET @doc='<root>'+@doc+'</root>'

-- Снова поместить документ в таблицу для того, чтобы можно было
-- передать документ в процедуру sp_xml_concat
SELECT @doc AS XMLText INTO #XMLText2

GO
DECLARE @hdl int
EXEC sp_xml_concat @hdl OUT, '#XMLText2', 'XMLText'
SELECT * FROM OPENXML(@hdl, '/root/authors') WITH
    (au_lname nvarchar(40))
EXEC sp_xml_removedocument @hdl
GO
DROP TABLE #XMLText1, #XMLText2
```

Как показано в листинге 18.92, после получения документа из процедуры sp_run_xml_proc и сохранения его в таблице документ загружается в переменную, заключается в корневой элемент и сохраняется во второй таблице для того, чтобы его

можно было передать в процедуру sp_xml_concat. После возврата управления из процедуры sp_xml_concat полученный дескриптор документа передается в функцию OPENXML и возвращается часть документа, как показано в листинге 18.93.

Листинг 18.93. Результаты выполнения кода, приведенного в листинге 18.92 (в сокращенном виде)

```
au_lname
----------------------------------------
Bennet
Blotchet-Halls
Carson
DeFrance
...
Ringer
Ringer
Smith
Straight
Stringer
White
Yokomoto
```

Итак, использование процедур sp_xml_concat и sp_run_xml_proc в сочетании со встроенными инструментальными средствами XML программы SQL Server позволяет применять в программе всю гамму средств обработки XML. При этом можно начать с получения фрагмента XML с помощью конструкции FOR XML AUTO, сохранить этот фрагмент в таблице, осуществить выборку его из таблицы, заключить в корневой элемент и, наконец, передать в функцию OPENXML для извлечения небольшой части первоначального документа в виде набора строк. Вполне очевидно, что описанные в данном разделе две процедуры значительно расширяют собственные возможности программы SQL Server по обработке кода XML.

Резюме

Средства SQLXML предоставляют пользователю исключительно широкий набор возможностей представления и обработки кода XML в программе SQL Server. С помощью этих средств можно осуществлять синтаксический анализ и загрузку документов XML, выполнять запросы к ним с использованием синтаксических конструкций XPath, применять запросы к объектам базы данных с помощью языка XPath, а также конструировать шаблоны и схемы отображения для запроса данных. Загрузка данных в базу данных SQL Server средствами XML может осуществляться с помощью функции OPENXML, шаблонов обновления и компонента XML Bulk Load, а для выборки данных из базы данных в виде кода XML можно использовать конструкцию FOR XML. Доступ к программе SQL Server выполняется с помощью протоколов HTTP и SOAP, а для передачи данных XML клиенту применяются средства и SQLOLEDB, и SQLXMLOLEDB. Набор строк может быть преобразован в код XML и на серверном компьютере, и на клиентском

компьютере, а для управления форматом создаваемого кода XML предусмотрен еще целый ряд механизмов. Столкнувшись с некоторыми более существенными ограничениями технологий SQLXML, можно использовать для устранения этих ограничений хранимые процедуры `sp_xml_concat` и `sp_run_xml_proc`, представленные в данной главе.

Вопросы для самопроверки

1. Назовите синтаксический анализатор XML, используемый в составе средств XML программы SQL Server.
2. Подтвердите или опровергните следующее утверждение. Опция `NESTED` может использоваться только в конструкции `FOR XML`, применяемой в клиентской программе.
3. Какая расширенная хранимая процедура используется для подготовки документа XML к использованию в функции `OPENXML`?
4. Каково теоретическое значение максимального объема памяти, который разрешается использовать синтаксическому анализатору MSXML в составе средств SQLXML, использующих пространство памяти процесса SQL Server?
5. Подтвердите или опровергните следующее утверждение. В настоящее время нет никакого способа, с помощью которого можно было бы запретить кэширование шаблонов для данного конкретного виртуального каталога SQLISAPI.
6. Опишите назначение атрибута `sql:mapping` из пространства имен `mapping-schema` компании Microsoft.
7. Почему максимум, упомянутый в вопросе 4, назван только теоретическим максимумом? Какие другие факторы могут помешать синтаксическому анализатору MSXML достичь верхнего предела допустимого объема распределяемой памяти?
8. Какой вспомогательный файл XML необходимо определить прежде, чем приступать к массовой загрузке документа XML в базу данных SQL Server?
9. Что определяется для двух таблиц с помощью аннотации `sql:relationship`?
10. Возможно ли изменить название библиотеки расширения DLL интерфейса ISAPI, связанной с данным конкретным виртуальным каталогом, или одна и та же библиотека расширения ISAPI должна применяться во всех виртуальных каталогах, конфигурация которых настроена на использование SQLISAPI?
11. Опишите способ обработки запросов URL средствами SQLXML.
12. Подтвердите или опровергните следующее утверждение. Интерфейс SQLXMLOLEDB, как и любое другое средство доступа OLE DB, обеспечивает возврат традиционных наборов строк.
13. Какая функция API-интерфейса Win32 вызывается средствами SQLXML для вычисления объема физической памяти на компьютере?

14. Назовите два главных API-интерфейса, предоставляемых средствами MSXML, с помощью которых может осуществляться синтаксический анализ документов XML.
15. Укажите приблизительно, насколько больший объем памяти занимает документ DOM по сравнению с основополагающим документом XML?
16. Объясните, что такое "специальная процедура".
17. Какая внутренняя специальная процедура обеспечивает реализацию расширенной процедуры `sp_xml_preparedocument`?
18. Какие два свойства объекта `Command` интерфейса ADO должны быть заданы для того, чтобы обеспечивалась обработка конструкции `FOR XML` в клиентской программе?
19. Какой метод объекта `Recordset` интерфейса ADO позволяет сохранить набор записей в формате кода XML?
20. Как расшифровывается аббревиатура "SAX", относящаяся к терминологии XML?
21. Информация о событии какого типа передается в программу SQL Server при выполнении стандартного запроса Transact-SQL в виде запроса URL?
22. Как называется средство доступа OLE DB, которое реализует клиентские функциональные возможности `FOR XML`, и в какой библиотеке DLL находится код этого средства доступа?
23. Используется ли средствами SQLXML синтаксический анализатор MSXML для форматирования в виде кода XML результатов серверных запросов `FOR XML`?
24. Подтвердите или опровергните следующее утверждение. Схемы XDR больше не поддерживаются средствами SQLXML.
25. Какой компонент следует использовать для обеспечения загрузки данных XML в базу данных SQL Server самым быстрым способом из всех возможных?
26. Подтвердите или опровергните следующее утверждение. Интерфейс SQLISAPI не предоставляет возможность выполнить возврат данных, отличных от XML, из базы данных SQL Server.
27. Возможно ли осуществить настройку виртуального каталога таким образом, чтобы запросы `FOR XML` по умолчанию обрабатывались в клиентской программе?
28. Укажите приблизительно, насколько больший объем занимает в оперативной памяти представление документа XML, сформированное средствами SQLXML и записанное в память для использования с функцией `OPENXML`, по сравнению с самим документом?
29. Подтвердите или опровергните следующее утверждение. Средства SQLXML не поддерживают вставку новых данных с помощью функции `OPENXML`, поскольку функция `OPENXML` всегда возвращает набор строк, допускающий только чтение.

30. Какой атрибут пространства имен `mapping-schema` следует использовать с атрибутом `xsd:relationship`, если схема отображения используется с шаблоном обновления, причем схема отображения связывает две таблицы в обратном порядке?
31. Назовите основную процедуру формирования сообщений об ошибках программы SQL Server, на вызове которой в данной главе была установлена точка останова.
32. Опишите ситуацию, в которой имело бы смысл использовать схему отображения с шаблоном обновления.
33. Какое единственное значение может иметь параметр `Data Source` интерфейса SQLXMLOLEDB?
34. Подтвердите или опровергните следующее утверждение. В основе работы синтаксического анализатора SAX лежит принцип сохранения всего документа в памяти в виде древовидной структуры, что позволяет легко обеспечить доступ к документу для остальной части приложения.

ГЛАВА 19

Службы рассылки извещений

В последние годы явно наметилась тенденция к расширению использования средств доставки данных на основе извещений. Уже сейчас существует возможность переходить на Web-узлы и оформлять подписку для получения сообщений о погоде, сообщений о ситуации на дорогах, результатов спортивных соревнований и многих других данных, рассылаемых в виде извещений. А после оформления подписки данные могут повсюду "следовать" за пользователем и доставляться одновременно с помощью электронной почты, сообщений на пейджер, текстовых сообщений для сотовых телефонов, средств мгновенного обмена сообщениями и т.д. Приложения такого типа постепенно завоевывают весь мир и находятся в главном фокусе внимания инициативы .NET компании Microsoft.

В целях предоставления платформы уровня предприятия для создания надежных, масштабируемых и полнофункциональных приложений рассылки извещений, которые используют программу SQL Server в качестве хранилища данных, были разработаны службы Notification Services (Службы рассылки извещений). В этих службах применяются возможности инфраструктуры .NET Framework (большинство служб Notification Services написано с помощью управляемого кода). Они представляют собой первоклассный набор инструментальных средств разработки определяемых пользователем приложений рассылки извещений, для которых характерны масштабируемость и надежность программы SQL Server, а также удобство настройки конфигурации и гибкость языка XML. Сами службы Notification Services рассматриваются в первой части этой главы, а последняя часть главы содержит описание процесса создания приложения для рассылки извещений.

Принципы работы служб Notification Services

Прежде всего необходимо четко определить терминологию: как показывает само название рассматриваемых средств, Notification Services – это не просто отдельная служба, а целая платформа и набор вспомогательных инструментальных средств, которые позволяют создавать приложения для рассылки извещений. Ключевым компонентом любого приложения для рассылки извещений является одна из служб Windows, предоставляемая пользователю службами Notification Services, но все приложение не сводится лишь к этой службе.

В отличие от самой программы SQL Server, невозможно просто установить службы Notification Services, а затем создать приложения для рассылки извещений, которые подключаются к этим службам с помощью клиентского программного обеспечения. Вместо этого вначале создается проект приложения для рассылки извещений с помощью инструментальных средств, предоставляемых службами Notification Services, а эти службы, в свою очередь, создают требуемое приложение. Поэтому сам термин "Notification Services" может оказаться немного расплывчатым, особенно если к анализу работы соответствующих служб подходить с точки зрения того, как работает программа SQL Server. По мнению автора, наиболее точным определением понятия "службы Notification Services" является следующее: коллекция служб и инструментальных средств, предназначенных для создания развитых и масштабируемых приложений для рассылки извещений.

Приложение Notification Services не функционирует в рамках процесса SQL Server; к такому приложению даже не предъявляется требование, чтобы оно работало на том же компьютере, где эксплуатируется программа SQL Server. Формально подобное приложение представляет собой просто клиентское приложение SQL Server. Приложения Notification Services используют базу данных SQL Server в качестве хранилища данных во многом на основе способа, который применяется в таких программных продуктах, как Systems Management Server и Microsoft Exchange.

Приложение Notification Services обычно имеет три компонента: компонент Windows Service, набор баз данных SQL Server, который служит в качестве хранилища данных, и приложение управления подпиской. В ходе создания приложения Notification Services службы Notification Services формируют первые два компонента без участия пользователя и предоставляют в качестве третьего компонента пример приложения, который может использоваться для того, чтобы пользхователь мог иметь по меньшей мере начальную заготовку для требуемого приложения управления подпиской.

Каждое приложение Notification Services базируется в одном из экземпляров служб Notification Services. Экземпляр Notification Services включает три составляющих: запись в системном реестре под ключом `HKLM\Software\Microsoft\Notification Services\Instances`, база данных SQL Server, которая хранит данные, относящиеся к экземпляру (например, таблицу подписчиков), и служба Windows, которая выполняет фактическую работу по согласованию событий с подписками и по выработке извещений. На одном экземпляре может базироваться множество приложений, но обычно принято соблюдать взаимнооднозначное соответствие между экземплярами и приложениями.

Примером ситуации, в которой может потребоваться несколько приложений, базирующихся на одном экземпляре, является необходимость иметь несколько взаимосвязанных приложений для рассылки извещений, которые должны совместно использовать общий список подписчиков. Такая организация работы, в которой все приложения базируются на одном экземпляре, позволяет пользователю единожды выполнять настройку для обеспечения подписки на извещения из любого приложения, а не создавать несколько экземпляров, при использовании которых приходилось бы отдельно регистрироваться в качестве подписчика в каждом отдельном приложении.

Утилита NSControl

Точкой входа в процедуру создания и управления приложениями Notification Services является утилита NSControl. Она представляет собой приложение с интерфейсом командной строки, которое выполняет применительно к службам Notification Services такую же общую роль, какую программа Enterprise Manager выполняет применительно к программе SQL Server, поскольку с помощью утилиты NSControl осуществляется создание, администрирование и управление приложениями Notification Services. Например, чтобы создать базы данных SQL Server, предназначенные для использования в приложении для рассылки извещений, необходимо вызвать функцию `Create` утилиты NSControl, а для регистрации нового экземпляра Notification Services вызывается функция `Register` утилиты NSControl. Эти темы будут рассматриваться более подробно ниже в данной главе.

Базы данных экземпляра и приложения

Работа каждого приложения Notification Services зависит по меньшей мере от двух баз данных SQL Server. Эти базы данных создаются от имени пользователя в ходе того, как происходит создание нового приложения для рассылки извещений с помощью функции `Create` утилиты NSControl. Одна из этих баз данных резервируется для использования экземпляром, а другая применяется самим приложением. Если экземпляр поддерживает несколько приложений, то должна быть предусмотрена только одна база данных экземпляра, а баз данных приложения может быть несколько. В приложении могут использоваться другие базы данных, но в типичной конфигурации существует только одна база данных, относящаяся к приложению, и одна база данных экземпляра.

Имя базы данных экземпляра всегда оканчивается суффиксом `NSMain`, а имя базы данных приложения всегда должно иметь и в качестве префикса, и в качестве суффикса имя, заданное для приложения в файле конфигурации приложения. Более подробные сведения о файле конфигурации приложения приведены ниже в данной главе.

Указанные базы данных содержат множество хранимых процедур, системных таблиц и других вспомогательных объектов. Имена этих объектов всегда обозначаются префиксом `NS` (автор знает только два исключения из этого правила). В качестве примеров исключений укажем, что представление с данными о подписках (subscription) в базе данных приложения имеет имя `AppName Subscriptions`, где `AppName` — имя приложения, а определяемая пользователем функция рассылки извещений (notification) в базе данных экземпляра имеет имя `ClassName Notify`, где `ClassName` — имя класса извещений, которому соответствует определяемая пользователем функция. Все другие объекты Notification Services в базах данных экземпляра и приложения имеют имя с префиксом `NS`.

Файлы конфигурации

Выше в данной главе упоминались файлы конфигурации, применяемые для создания приложения Notification Services. Безусловно, читателя не может не

интересовать вопрос, какие именно файлы служат для этой цели и как они используются. Вначале в данном разделе будет приведен краткий обзор особенностей каждого из этих файлов, а более подробное описание файлов конфигурации будет приведено в той части главы, в которой рассматривается процесс создания собственного приложения для рассылки извещений.

Для определения приложения Notification Services используется два файла конфигурации — файл конфигурации экземпляра и файл определения приложения. Файл конфигурации экземпляра определяет экземпляр и указывает те приложения, которые базируются на этом экземпляре, а файл определения приложения характеризует отдельное приложение. Если на одном экземпляре базируется несколько приложений, то должен быть предусмотрен только один файл конфигурации экземпляра и несколько файлов определения приложения.

Файл конфигурации экземпляра обычно носит имя `appConfig.XML` (например, такое имя ему присваивается в примерах приложений управления подпиской, `Sample`), но фактически может иметь любое имя. Автор предпочитает использовать имя `instConfig.XML`, поскольку файл конфигурации экземпляра определяет не одно приложение (на что указывает префикс `app` — приложение), а относится ко всему экземпляру (отсюда и префикс `inst` — экземпляр). Этот файл представляет собой документ XML, который должен соответствовать схеме XML с именем `ConfigurationFileSchema.XSD` (файл с указанной схемой можно найти в подкаталоге `XML Schemas` корневого инсталляционного каталога Notification Services). Файл конфигурации экземпляра содержит данные о конфигурации экземпляра и определяет, как работает служба Windows этого экземпляра.

Файл конфигурации экземпляра содержит по одной записи для каждого приложения, базирующегося на этом экземпляре; такая запись содержит ссылку на файл определения приложения. Кроме того, файл конфигурации экземпляра включает узлы, указывающие экземпляр программы SQL Server, на котором базируется экземпляр приложения, а также протоколы и каналы доставки, поддерживаемые экземпляром. Приступая к выполнению любой задачи, применительно к которой утилита NSControl требует указания имени файла конфигурации, необходимо указывать только имя файла конфигурации экземпляра. Поскольку этот файл содержит ссылки, которые указывают на файлы определений для приложений, базирующихся на данном экземпляре, утилита NSControl имеет возможность получить доступ к файлам определения приложения, не требуя указывать их имена в командной строке.

Файл определения приложения также представляет собой документ XML. Этот документ соответствует схеме XML, заданной в файле `ApplicationDefinitionFileSchema.XSD`, также находящемся в подкаталоге `XML Schemas` корневого инсталляционного каталога Notification Services. Файл определения приложения описывает отдельное приложение Notification Services и регламентирует структуру событий и подписок, принимаемых приложением в качестве входных данных, а также извещений, вырабатываемых приложением в качестве выходных данных. Пример файла определения приложения будет рассматриваться при создании примера приложения ниже в этой главе.

Следует учитывать, что указанные файлы конфигурации используются только утилитой NSControl. Файлы конфигурации предоставляют сведения о конфигурации

для конкретных функций утилиты NSControl, выполняющих такие задачи, как создание баз данных экземпляра и приложения или обновление конфигурации экземпляра. После создания службы экземпляра обращение к этим файлам больше не происходит. Информация о конфигурации, представленная в этих файлах, после ее передачи в утилиту NSControl материализуется в виде объектов и записей таблиц в базах данных экземпляра и приложения, поэтому в дальнейшем к данным файлам обращение не происходит ни в процессе сбора информации о событиях, ни во время вызова генератора, ни в ходе распространения извещений. Но, безусловно, удалять файлы конфигурации не следует, поскольку может быть принято решение уточнить детали конфигурации после создания и регистрации приложения (в таком случае соответствующий файл конфигурации передается в функцию `Update` утилиты NSControl). Тем не менее читатель должен учитывать, что указанные файлы служат исключительно в качестве входных данных для утилиты NSControl и не имеют больше какого-либо иного назначения.

Исполняемый файл `NSService.exe`

Файл `NSService.exe` — это исполняемый файл Notification Services, который функционирует в качестве службы Windows в приложении для рассылки извещений. В каждом приложении Notification Services, относящемся к какому-то конкретному выпуску этого программного продукта, используется одна и та же копия программы `NSService.exe` (в этом смысле экземпляр Notification Services можно рассматривать как экземпляр указанного исполняемого файла). Если экземпляр зарегистрирован в качестве службы, то в системном реестре в качестве исполняемого файла указано имя `NSService.exe`, а имя экземпляра передается в командной строке. После запуска экземпляр `NSService.exe` принимает имя экземпляра Notification Services и осуществляет поиск базового приложения SQL Server с помощью ключа реестра. А после того как экземпляр `NSService.exe` определит имя экземпляра базы данных с помощью имени экземпляра Notification Services, реализуемая программой `NSService.exe` служба Windows получает все необходимое для доступа к объектам SQL Server, которые требуются для текущего контроля над событиями и преобразования информации об этих событиях в извещения.

Следует отметить, что требование по эксплуатации программы `NSService.exe` в качестве службы не является обязательным. Так же, как и агенты репликации SQL Server, программа `NSService.exe` может быть вызвана на выполнение в качестве утилиты с интерфейсом командной строки. Но весомых оснований для такой организации работы не существует; автор упоминает этот вариант только для полноты и предоставления читателю возможности лучше понять, как взаимодействуют друг с другом все фрагменты служб рассылки извещений. Программа `NSService.exe` может быть вызвана на выполнение из командной строки примерно таким образом:

```
nsservice InstanceName -a
```

Здесь *`InstanceName`* — имя используемого экземпляра. Безусловно, рассматриваемый исполняемый файл должен либо находиться в каталоге, указанном в переменной среды `PATH`, которая обозначает пути к каталогам, применяемым по

умолчанию, либо перед вызовом на выполнение указанной выше команды должен быть выполнен переход в тот каталог, где находится соответствующий исполняемый файл. (По умолчанию файл `NSService.exe` находится в каталоге `\Program Files\Microsoft SQL Server Notification Services\v2.0.2114.0\Bin`, где `v2.0.2114.0` — установленная версия служб Notification Services.)

ПРЕДОСТЕРЕЖЕНИЕ. Еще раз отметим, что информация о том, как вызвать на выполнение программу `NSService.exe` из командной строки, приведена только для полноты. Автор не рекомендует вызывать эту программу на выполнение таким образом в любом сценарии эксплуатации служб рассылки извещений на производстве. Дело в том, что несложно предугадать, какие проблемы возникнут после запуска на выполнение программы `NSService.exe` из командной строки одновременно с тем, как эта программа уже будет работать в качестве службы, а оба процесса станут ссылаться на один и тот же экземпляр Notification Services.

Основное назначение программы `NSService.exe` состоит в том, что она согласует данные о событиях с подписками и вырабатывает извещения. Программа `NSService.exe` представляет собой машину выработки извещений, т.е. средство, способное собирать информацию о событиях, согласовывать эти события с подписками и преобразовывать результаты выполненных согласований в извещения, передаваемые подписчикам. Привязка пользовательского кода к функциям этой машины осуществляется с помощью описанных ниже объектов.

- Файлы конфигурации экземпляра и файлы определения приложения.
- Триггеры, запросы и хранимые процедуры на языке Transact-SQL.
- Компоненты управляемого кода, такие как определяемые пользователем средства доступа к информации о событиях и определяемые пользователем протоколы доставки.
- Таблицы стилей XML.

Каждый из этих объектов будет рассматриваться более подробно ниже в данной главе.

Службы Notification Services предоставляют расширяемый фундамент, на котором может формироваться богатый набор определяемых пользователем функциональных возможностей. Эти службы предоставляют машину формирования извещений и объекты, предназначенные для преобразования информации о событиях и подписках в извещения. А пользователь преобразует эту универсальную машину формирования извещений в специализированное приложение, выполняя привязку к этой машине определяемых им функциональных средств в соответствующих местах. Для этого используются тщательно определенные процедуры и стандартные API-интерфейсы.

Компоненты приложения для рассылки извещений

Функционирование приложения Notification Services основано на использовании трех основных компонентов: средств доступа к данным о событиях, генератора и распределительного сервера. Средства доступа к данным о событиях

собирают информацию о событиях. Генератор согласовывает эти события с подписками и вырабатывает неотформатированные извещения. Распределительный процесс принимает неотформатированные извещения и преобразует их в извещения, применимые для доставки подписчикам.

Средства доступа к данным о событиях

Как уже было сказано, средства доступа к данным о событиях собирают информацию о событиях, представляющих интерес для пользователей приложения Notification Services. Например, одно из средств доступа к данным о событиях может собирать информацию о результатах спортивных соревнований для использования в приложении рассылки извещений со спортивной информацией или об измеренных значениях температуры, необходимых для приложения рассылки извещений с информацией о погоде. С каждым отдельным приложением для рассылки извещений может быть связано несколько средств доступа к данным о событиях.

Средство доступа к данным о событиях может либо базироваться, либо не базироваться на другом процессе. Базирующееся средство доступа к данным о событиях представляет собой библиотеку DLL или сборку, которая реализует интерфейсы `IEventProvider` или `ISchedulEdEventProvider`. Такое средство доступа функционирует в рамках процесса `NSService.exe`.

Базирующиеся средства доступа к данным о событиях подразделяются на две разновидности — действующее по графику и действующее непрерывно. Средство доступа к данным о событиях, действующее по графику, вызывается в соответствии с графиком, который задается при создании приложения с помощью функции `Create` утилиты NSControl. Средство доступа к данным о событиях, действующее непрерывно, запускается с момента запуска приложения для рассылки извещений и функционирует до тех пор, пока не происходит останов этого приложения.

Небазирующееся (или независимое) средство доступа к данным о событиях действует вне пределов приложения Notification Services. Оно может быть оформлено в виде отдельно действующего исполняемого файла или может функционировать в составе другого процесса, такого как процесс сервера IIS. В небазирующемся средстве доступа к данным о событиях используется один из описанных ниже API-интерфейсов для передачи данных о событиях в систему.

1. Применительно к данным XML независимое средство доступа может создавать объекты `EventLoader` и записывать данные о событиях, представленные в формате XML, в базу данных приложения.
2. Для передачи данных о событиях средство доступа может также вызывать специальные хранимые процедуры, предусмотренные службами Notification Services в базе данных приложения. В приложении, пример которого приведен ниже в данной главе, этот метод, основанный на триггере T-SQL, используется для выработки информации о событиях Notification Services.
3. Средство доступа с управляемым кодом может непосредственно создавать объекты `Event`, добавлять их к объекту `EventCollector` и передавать в виде пакета событий в систему.

4. Для передачи данных о событиях в средстве доступа могут применяться объекты и методы COM. В службах Notification Services для предоставления доступа к классам управляемого кода, представленным в виде COM-интерфейсов, используется технология COM Interop.

Генератор

В отличие от многочисленных средств доступа к данным о событиях, в каждом приложении может быть предусмотрен только один генератор. Генератор согласовывает подписки с событиями и вырабатывает извещения. Правила, по которым проводится согласование между подпиской и событием, называются *правилами согласования* и состоят из операторов Transact-SQL, которые задаются в файле определения приложения.

Процесс, в котором обнаруживаются согласования и вырабатываются извещения, может быть более сложным, чем простое согласование подписок с событиями. Извещения могут вырабатываться на основе графика, определенного в подписке, кроме того, при их составлении могут использоваться исторические данные.

Генератор вырабатывает "неотформатированные" извещения, которые должны быть обработаны в распределительном процессе для их преобразования в формат, подходящий для доставки подписчикам. У автора часто возникает желание найти какой-то другой термин для обозначения формируемых генератором исходных элементов извещений, вместо термина "извещение", который используется в документации служб Notification Services. Возможно, лучше было бы называть такие элементарные информационные единицы "предызвещениями", "неотформатированными извещениями" или даже "сообщениями". Но как бы то ни было, достаточно помнить, что выходные данные процесса генерации все еще должны быть обработаны распределительным сервером, прежде чем их можно будет отправить подписчикам.

Если приложение поддерживает подписки, обслуживаемые по графику, и генератор их обрабатывает, то обработка применяется только к тем подпискам, которые подлежат обработке в данный конкретный момент времени. Такая организация работы исключает необходимость для генератора выполнять бесполезные действия по исполнению подписок, требования которых невозможно удовлетворить, поскольку не прошел обусловленный графиком этих подписок временной интервал, или не наступило время вызова их на выполнение.

Если в приложении требуется использовать исторические данные для определения того, согласуются ли некоторая подписка и некоторое событие, такая задача может быть решена с помощью дополнительных таблиц, называемых *хронологическими*. По существу, хронологическая таблица представляет собой специальную таблицу, определяемую приложением, в которой регистрируются исторические данные (т.е. ведется хронология таких данных) для дальнейшего использования в процессе выработки извещений. Например, хронологическая таблица может применяться для регистрации колебаний во времени цен на акции определенного выпуска, чтобы исключить необходимость повторной выработки извещений в течение одного и того же операционного дня, после того как цена несколько раз выйдет за пределы верхнего или нижнего порогового значения, обусловленного в подписке.

Правила согласования

Как уже было сказано, данные о событиях, преобразуемые генератором в неотформатированные извещения, согласуются с подписками с помощью правил согласования. Правила согласования представляют собой операторы SELECT языка T-SQL, заданные в файле определения приложения. Генератор поддерживает правила трех описанных ниже типов.

1. Правила хроники событий предусматривают регистрацию исторической информации о событиях во вспомогательных таблицах. Как уже было сказано выше, хронологические таблицы позволяют воспользоваться в процессе выработки событий историческим контекстом. Правила хроники событий активизируются генератором в первую очередь.
2. Правила события подписки вырабатывают извещения, относящиеся к подпискам, основанным на данных о событиях. Эти правила активизируются после правил хроники событий (если имеется относящийся к ним пакет событий) и также могут взаимодействовать с хронологическими таблицами.
3. Проверяемые по графику правила подписки применяются при выработке извещений для подписок, выполняемых по графику. Эти правила активизируются после правил хроники событий для соответствующих подписок и также могут взаимодействовать с хронологическими таблицами.

Кванты времени

От значения продолжительности кванта времени, которое определяется в узле `QuantumDuration` файла определения приложения, зависит то, насколько часто генератор выходит из состояния ожидания и активизирует правила. Это значение продолжительности времени задается в формате времени XML Schema, обусловленном стандартом ISO 8601, и может находиться в пределах от нескольких секунд до любого необходимого значения времени. Безусловно, чем короче продолжительность кванта времени, тем чаще активизируется генератор и тем больше нагрузка на систему, связанная с работой генератора. С другой стороны, увеличение продолжительности кванта времени приводит к уменьшению нагрузки на систему, но служит также причиной того, что выработка извещений занимает больше времени, и поэтому доставка извещений подписчикам больше запаздывает. Если продолжительность кванта времени не задана, то применяется значение, равное по умолчанию одной минуте.

Функция рассылки извещений

Извещения обрабатываются с помощью функции рассылки извещений, которая представляет собой определяемую пользователем функцию SQL Server, создаваемую от имени пользователя службой Notification Services в базе данных приложения для каждого класса извещения (типа извещения), который задан в файле определения приложения. Если определен только один класс извещения, то создается только одна функция рассылки извещений. Каждая определяемая

пользователем функция рассылки извещений получает имя, которое происходит от имени соответствующего ей класса извещений и имеет в своем составе суффикс Notify. Такая функция используется в операторе SELECT правила согласования для выработки извещений. Извещения вырабатываются путем вызова расширенной процедуры, которая открывает соединение через петлю обратной связи с хостом SQL Server и вставляет данные извещения с помощью вызова хранимой процедуры NSInsertNotificationn. (Соединение через петлю обратной связи является необходимым, поскольку не существует иного способа выполнения запросов к программе SQL Server с помощью расширенных процедур.) Указанная расширенная процедура имеет имя xp_NSNotify_*version*, где *version* – версия служб Notification Services.

Столбцы, указанные в том операторе SELECT языка T-SQL, который задан в правиле согласования, передаются в определяемую пользователем функцию, поэтому такая функция вызывается для обработки каждой строки в результирующем наборе. (Если в определяемую пользователем функцию, которая используется в операторе SELECT, не передается ни один столбец, то эта функция вызывается только один раз, применительно ко всему оператору.) Взаимодействие между определяемой пользователем функцией и процессом выработки извещений проще всего понять с помощью примера. Правила согласования описаны более подробно ниже в данной главе, а в настоящем разделе приведен пример правила, который демонстрирует использование определяемой пользователем функции для выработки извещений (листинг 19.1).

Листинг 19.1. Пример использования определяемой пользователем функции для выработки извещений

```
SELECT dbo.BNSInfoNotify(s.SubscriberId,
                         s.DeviceName,
                         s.SubscriberLocale,
                         p.ID,
                         p.Product,
                         p.OpenedBy)
FROM BNSEvents e, BNSSubscriptions s, BNS..Bugs p
WHERE e.ID = p.ID
AND p.Product = s.Product
```

Обратите внимание на то, как применяется функция BNSInfoNotify. Эта функция вызывает расширенную процедуру рассылки извещений. Для оператора SELECT не требуется возвращаемое значение функции, поскольку это значение относится к одному из побочных эффектов применения функции. Определяемая пользователем функция вызывает расширенную процедуру, поэтому обладает способностью модифицировать другие таблицы на сервере в контексте оператора SELECT. При обычных условиях выполнение таких действий в определяемой пользователем функции не разрешено. По сути, определяемая пользователем функция позволяет избежать необходимости открывать при обработке правила согласования курсор, заданный в операторе SELECT, который согласовывает события с подписками и вызывает функцию NSInsertNotificationn для каждого отдельного

события. Благодаря использованию такого подхода в службах Notification Services объем нагрузки по кодированию правил согласования существенно уменьшается.

Именно такой же подход применялся при создании расширенной процедуры xp_exec, которая была описана в последней книге автора, *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML*. Как указано в упомянутой книге, действия, которые могут быть выполнены в определяемой пользователем функции SQL Server, подвержены многочисленным ограничениям. Поэтому вызов расширенной процедуры, в которой используется соединение через петлю обратной связи для выполнения определяемого пользователем кода T-SQL, является почти единственным способом осуществления многих полезных действий из определяемой пользователем функции, включая применение оператора SELECT для реализации параметризованных вызовов хранимых процедур. В приведенной выше определяемой пользователем функции рассылки извещений выполняются именно эти действия. В указанной определяемой пользователем функции рассылки извещений принят такой же подход, который продемонстрирован в примере кода xp_exec – в этой функции используется расширенная процедура для открытия соединения через петлю обратной связи с сервером, и выполняется код, который при обычных условиях не было бы разрешено выполнять из определяемой пользователем функции (т.е. код, содержащий вызовы хранимых процедур) с учетом значений, переданных в функцию из оператора SELECT.

Данный способ напоминает способ, предусмотренный в задачах управляемого данными запроса Data Driven Query (DDQ) средств DTS. В задаче DDQ выполняются параметризованные запросы с учетом данных, переданных в эту задачу другой задачей DTS (а последняя задача вполне может представлять собой вызов оператора SELECT или хранимой процедуры).

Но в отличие от расширенной процедуры xp_exec, расширенная процедура формирования извещения не поддерживает понятие соединения с транзакционным контекстом вызывающего объекта. Рассмотрим, с чем это связано и почему в данной расширенной процедуре не предусмотрено соединение с транзакционным контекстом вызывающего объекта для обеспечения того, чтобы не происходила блокировка необходимых ресурсов вызывающим объектом. Причина, по которой в расширенной процедуре формирования извещения не применяется такое соединение, проста – указанная операция не поддерживается программой SQL Server. Как было указано автором при описании расширенной процедуры xp_exec в его предыдущей книге, расширенная процедура не может соединить свой контекст с транзакционным контекстом вызывающего ее объекта, если вызов происходит из определяемой пользователем функции. Учитывая то, что расширенная процедура xp_NSNotify не была предназначена для вызова из какого-либо иного объекта, кроме определяемой пользователем функции, наверное, имело бы смысл предусмотреть в этой расширенной процедуре какую-либо поддержку, позволяющую выполнить привязку к транзакционному пространству вызывающего эту процедуру серверного процесса.

Несмотря на то, что в этом состоит единственно возможный способ действий, который как раз и стремились осуществить разработчики служб Notification Services, из определяемой пользователем функции инициализируется отдельное соединение, а это приводит к возникновению некоторых интересных вопросов. Прежде всего,

учитывая то, что теоретически отдельным серверным процессам могут потребоваться одни и те же ресурсы, существуют предпосылки того, что расширенная процедура формирования извещения может быть заблокирована вызывающим ее процессом. Но в связи с тем, что транзакционный контекст, в котором выполняются правила согласования, контролируется исключительно процессом `NSService.exe`, такая ситуация маловероятна. Пользователю пришлось бы ввести в правило согласования какие-то собственные приемы создания определяемых пользователем функций, чтобы изменить транзакционную среду до такой степени, чтобы можно было заблокировать расширенную процедуру. Пользователю пришлось бы искать способ выйти за рамки пользовательской функции и установить блокировки на тех же таблицах, к которым обращается хранимая процедура `NSInsertNotificationn` для вставки новых извещений. Автор не может представить себе, при каких условиях это могло бы произойти, не считая весьма надуманных ситуаций.

Гораздо более вероятный сценарий состоит в том, что соединение через петлю обратной связи может быть закрыто из-за возникновения аварийной ситуации. Автор действительно наблюдал такие ситуации при выполнении расширенной процедуры `xp_exec`, а также недокументированной расширенной процедуры `xp_execresultset`, которая входит в поставку программы SQL Server (после выпуска служебного пакета SQL Server 2000 Service Pack 3 указанная расширенная процедура уже таковой не является). Если в клиентской программе используется заслуживающее доверия соединение для подключения к программе SQL Server, то при отсутствии доступа к контроллеру домена и при отсутствии в кэше на локальном компьютере достаточного объема информации мандата защиты клиентская программа, пытающаяся установить соединение, получает печально знаменитое сообщение об ошибке "cannot generate SSPI context" (невозможно сформировать контекст SSPI). В этом сообщении SSPI обозначает интерфейс провайдера поддержки защиты (Security Support Provider Interface), а сама ошибка означает, что невозможно выполнить необходимые операции защиты, чтобы успешно делегировать программе SQL Server лексему защиты клиента. Такая ситуация часто обусловлена тем, что клиент не может обратиться к контроллеру домена. Благодаря тому, что предусмотрена запись информации мандата защиты в кэш, иногда указанную проблему можно устранить (по меньшей мере до следующей перезагрузки), повторно подключаясь к сети; еще один способ устранения этой ошибки состоит в обеспечении доступа к контроллеру домена.

Если доступ к контроллеру домена является затруднительным, то вызовы расширенной процедуры формирования извещения могут завершаться неудачей, даже если программа `NSService.exe` подключена к программе SQL Server и продолжает функционировать без каких-либо проблем. В программе `NSService.exe` используется средство создания пула соединений, поэтому вероятность того, что в этой программе потребуется инициализировать новое соединение с сервером в ходе ее функционирования, является достаточно низкой. С другой стороны, в расширенной процедуре формирования извещения может действительно потребоваться инициализировать новое соединение, и эта попытка может окончиться неудачей, если контроллер домена недоступен. Обычная ситуация, в которой чаще всего наблюдается подобное развитие событий, связана

с эксплуатацией служб Notification Services на портативном компьютере. Если портативный компьютер обычно регистрируется в сети с помощью контроллера домена, а после отключения от сети перезагружается, то могут возникать проблемы при формировании извещений из-за ошибок, связанных с формированием контекста SSPI. В таком случае системный журнал регистрации событий будет показывать неудавшуюся попытку вызова и причину возникшей неудачи.

Учитывая то, что хранимая процедура `NSInsertNotificationn` передает на сервер простой оператор `INSERT` для вставки данных о новом извещении, может быть предусмотрен еще один способ осуществления такого же действия, которое выполняется с помощью создания соединения через петлю обратной связи для расширенной процедуры. Этот способ состоит в создании триггера `INSTEAD OF` на представлении или на вспомогательной таблице. При этом в представление или во вспомогательную таблицу должно быть вставлено правило согласования, а триггер должен обеспечивать преобразование строк, вставленных с помощью оператора (операторов) `INSERT`, в строки соответствующих основополагающих таблиц. Именно этот способ описан в предыдущей книге автора, где представлен альтернативный подход к разделению документов XML на основе использования триггера `INSTEAD OF`.

Безусловно, способ, основанный на применении определяемой пользователем функции/расширенной процедуры, который реализован в службах Notification Services, уменьшает нагрузку по разработке кода, возложенную на разработчика, создающего правила согласования для приложения рассылки извещений. Однако автор не уверен в том, что такой подход позволяет добиться большего масштабирования служб Notification Services по сравнению, скажем, с подходом, предусматривающим использование курсора, определенного на результатах применения правила согласования. Как уже было сказано в главе 10, вызов расширенной процедуры любого рода (независимо от того, инициализирует эта процедура соединение через петлю обратной связи или нет) приводит к тому, что рабочий поток, связанный с вызывающей процедурой, "уходит" из-под управления планировщика. Иными словами, данный поток становится для планировщика неприменимым при обработке рабочих запросов, поступающих от других серверных процессов. Дело в том, что планировщик не может определить, будет ли данная расширенная процедура возвращать управление достаточно часто, поэтому должен исходить из предположения, что на это не следует рассчитывать, т.е. должен игнорировать базовый рабочий поток этой расширенной процедуры с точки зрения возможности применения этого рабочего потока для выполнения других рабочих запросов до тех пор, пока расширенная процедура не завершит свою работу.

Если вы воспользуетесь утилитой `STRESS.CMD`, описанной выше в этой книге, для создания экземпляров, скажем, 255 соединений, чтобы выполнить расширенную процедуру, в которой просто вызывается функция `Sleep` API-интерфейса Windows для установки паузы в вызывающем потоке на 30 секунд, то не сможете создать новое соединение с сервером до тех пор, пока работает эта расширенная процедура. Более того, произойдет исчерпание ресурсов во всех других соединениях, занятых в настоящее время выполнением запросов, поскольку в каждом из этих соединений в какой-то момент происходит передача управления, а рабочий

поток, который использовался в реальных запросах, захватывает другая расширенная процедура. Поэтому сам подход, предусматривающий вызов расширенной процедуры, оказывает отрицательное влияние на масштабируемость программы SQL Server, поскольку требует принудительного выделения рабочего потока какому-то конкретному серверному процессу вместо совместного использования этого потока многими различными процессами, как и должно быть при обычных условиях.

Еще одним фактором уменьшения масштабируемости, возникающим при использовании рассматриваемого подхода, кроме описанного выше, является то, что расширенная процедура инициализирует новое соединение с программой SQL Server в расчете на каждую строку в согласованном результирующем наборе. Система может обрабатывать тысячи или даже миллионы извещений с помощью единственного вызова правила согласования, поэтому может оказаться, что в результате одной активизации экземпляра генератора создаются и уничтожаются тысячи или миллионы новых соединений с программой SQL Server. Еще худшая ситуация может возникнуть, если система чрезвычайно загружена или имеет ограниченное количество доступных рабочих потоков (например, из-за того, что рабочие потоки захвачены расширенными процедурами). В такой ситуации может оказаться, что расширенная процедура из функции формирования извещения не сможет немедленно установить соединение через петлю обратной связи и заблокируется на время ожидания открытия такого соединения. Безусловно, отрицательное влияние на работу системы, вызванное подобными соединениями через петлю обратной связи, может оказаться весьма значительным, в зависимости от количества и частоты поступления запросов на установление таких соединений.

Подход, основанный на использовании триггера `INSTEAD OF`, о котором упоминалось выше в данной главе, не связан с возникновением подобных проблем, а также не требует простого открытия курсора на результирующем наборе, полученном в ходе согласования, и вызова процедуры хранимой процедуры `NSInsertNotificationn` применительно к каждой строке. Безусловно, ввод в действие курсоров связан с собственными проблемами производительности и масштабируемости, но применение курсоров по крайней мере не приводит к тому, что серверный процесс захватывает рабочий поток, а также не вызывает пробуксовку соединений.

Распределительные серверы

Как было указано выше, извещения, вырабатываемые генератором, создаются в неотформатированном виде, поэтому должны быть отформатированы и упакованы перед их распределением в виде извещений для передачи конечному пользователю. Именно эту задачу и выполняет распределительный сервер. Распределительный сервер принимает неотформатированные извещения, выработанные генератором, и преобразует их в отформатированные извещения, которые могут передаваться пользователям по протоколу доставки. Для преобразования неотформатированных извещений в файлы, применимые для доставки подписчика, обычно используются таблицы стилей XML.

После того как генератор заканчивает создание пакета неотформатированных извещений, распределительный сервер считывает данные о подписчиках из па-

кета извещений и определяет, какого типа форматирование требуется в каждом случае. Затем распределительный сервер обрабатывает каждое неотформатированное извещение и передает его в службу доставки (например, на сервер SMTP) с помощью канала доставки, который определен в файле конфигурации экземпляра.

Форматирование сообщений

При подготовке информации о классе извещений в файле определения приложения должно быть указано, как осуществляется форматирование извещений этого класса. Выше уже было сказано, что общепринятый метод преобразования неотформатированных извещений в такие текстовые данные, которые могут быть отправлены подписчикам, состоит в использовании таблицы стилей XML. Чтобы воспользоваться встроенным средством форматирования служб Notification Services на основе XSL, необходимо указать `XsltFormatter` в качестве опции `ClassName` узла `ContentFormatter` в файле определения приложения. Необходимо также создать таблицу стилей и указать местонахождение и имя файла в узле `ContentFormatter`.

Определение средства форматирования информационного наполнения `ContentFormatter` должно быть задано для каждой комбинации устройства и региональной установки. Если приложение поддерживает несколько региональных установок и несколько устройств доставки, то средство форматирования информационного наполнения должно быть предусмотрено для каждого сочетания того и другого. Службы Notification Services предоставляют пользователю одно готовое к работе средство форматирования информационного наполнения — `XsltFormatter`. Поскольку язык таблиц стилей XML обладает столь высокой гибкостью и мощью, чаще всего в приложениях не возникает необходимость в использовании какого-либо иного средства форматирования информационного наполнения. Как было указано в главе 8, таблицы стилей XML позволяют формировать широкий перечень форматов выходных данных, поступающих из единственного источника данных XML. Таблицы стилей дают возможность применять циклические конструкции, операторы управления ходом выполнения по условию, а также определяемые пользователем функции, поэтому с помощью таблиц стилей можно выполнять практически любые необходимые действия.

Несмотря на сказанное выше, в некоторых ситуациях пользователю может потребоваться или он может пожелать создать собственное средство форматирования информационного наполнения. В таком случае службы Notification Services предусматривают возможность создавать определяемые пользователем средства форматирования информационного наполнения путем реализации интерфейса `IContentFormatter`. Такое средство форматирования информационного наполнения и все необходимые для него параметры задаются в секции `NotificationClasses` файла определения приложения.

Доставка извещений

Программа `NSService.exe` не осуществляет доставку извещений в конечную точку назначения. Соответствующие обязанности возлагаются на такие службы доставки, как сервер SMTP. Программа `NSService.exe` передает извещения в эти службы с помощью каналов доставки, определения которых находятся в файле

конфигурации экземпляра. В канале доставки полученные извещения оформляются в виде пакета используемого протокола и передаются в указанную службу доставки. Например, при использовании сервера SMTP извещения оформляются в канале доставки с помощью протокола доставки SMTP и передаются на указанный сервер SMTP или Exchange, а сервер, в свою очередь, доставляет эти извещения подписчикам. Службы Notification Services поддерживают перечисленные ниже стандартные протоколы доставки.

- **Протокол SMTP.** Обеспечивает передачу извещений на серверы SMTP или Exchange.
- **Файловый протокол.** Предоставляет возможность помещать извещения в файлы операционной системы, которые затем могут быть считаны другими процессами для дальнейшего распределения. Этот протокол также удобен для отладки приложений рассылки извещений.

Следует отметить, что пользователю предоставляется также возможность создавать собственные протоколы доставки. Для создания определяемого пользователем протокола доставки необходимо использовать класс, который реализует интерфейс `IDeliveryProtocol` или интерфейс `IHttpProtocolProvider`. Интерфейс `IDeliveryProtocol` — это основной интерфейс определяемого пользователем протокола доставки. В службах Notification Services предусмотрен интерфейс `IHttpProtocolProvider`, позволяющий упростить создание определяемых пользователем протоколов доставки на основе протокола HTTP. При этом пользователю предоставляется основная часть вспомогательных средств, необходимых для работы с протоколом HTTP, а пользователь просто предусматривает код, необходимый для форматирования конверта и обработки ответа. Кроме того, пользователь может использовать интерфейс `IHttpProtocolProvider` для доставки извещений с помощью таких протоколов на основе HTTP, как SOAP, SMS и .NET Alerts.

Резюмированные и многоадресатные извещения

Кроме стандартных извещений на основе сообщений, службы Notification Services позволяют также группировать извещения, предназначенные для какого-то конкретного подписчика, и передавать их в виде одного извещения. Для этого используются так называемые *резюмированные извещения*. Кроме того, указанные службы позволяют передавать единственное извещение многочисленным подписчикам с помощью средства, известного под названием *многоадресатной доставки*. Резюмированные извещения позволяют исключить ситуации, в которых подписчиков буквально накрывают лавины извещений, а многоадресатная доставка позволяет добиться увеличения масштабируемости, поскольку одно-единственное извещение доставляется сразу многим подписчикам, вместо отправки отдельного извещения каждому из них.

Подписчики и подписки

Как было отмечено в начале этой главы, приложения Notification Services состоят из трех основных компонентов: исполняемый файл службы Windows, хранилище данных SQL Server и приложение управления подпиской. Первые два из этих трех

компонентов либо предоставляются пользователю службами Notification Services, либо формируются этими службами от имени пользователя. Третий компонент, приложение управления подпиской, пользователь должен создать самостоятельно.

Приложение управления подпиской в составе приложения рассылки извещений обеспечивает подготовку подписчиков, устройств подписчиков, а также подписок. Это может быть приложение Web или стандартное приложение Windows, обладающее способностью использовать объектную модель служб Notification Services для управления подписками и связанной с этим информацией. Безусловно, основная часть работы объектной модели, относящейся к управлению подпиской, фактически выполняется хранимыми процедурами либо в базе данных экземпляра, либо в базе данных приложения, но автор не рекомендует вызывать эти хранимые процедуры непосредственно, поскольку они являются недокументированными.

Как уже упоминалось выше, информация, относящаяся к подпискам, хранится в базе данных приложения, а информация, относящаяся к подписчикам, находится в базе данных экземпляра. Такая организация работы позволяет любому подписчику подписываться на извещения, поступающие из многих приложений, не будучи вынужденным отдельно задавать сведения о себе в каждом приложении. Подписчик, работая с объектной моделью Notification Services, не обязан также указывать конкретные имена баз данных. Службы рассылки извещений сами определяют базу данных, в которой находится объект каждого конкретного типа с учетом указанных пользователем имен экземпляра и приложения.

Ниже в этой главе показано, как создать полностью готовое к работе приложение Notification Services, включая приложение управления подпиской. Но вначале необходимо ознакомиться с тем, как типичное приложение управления подпиской взаимодействует с объектной моделью Notification Services.

Пример приложения

В поставку Notification Services входит ряд примеров приложений, которые можно копировать и приспосабливать для использования с собственными приложениями рассылки извещений. Для копирования примера приложения применяется утилита `CopySample`. Такая операция копирования позволяет создать копию нового приложения, наряду с собственным виртуальным каталогом IIS и переменными среды, относящимися к данному конкретному проекту.

Существующие примеры приложений показывают, как следует использовать проект Makefile в среде Visual Studio .NET для реализации определяемого пользователем процесса создания приложения в интегрированной среде разработки Visual Studio. Каждый пример готового решения состоит из двух проектов. Один из этих проектов относится к приложению управления подпиской, которое представляет собой обычное приложение ASP.NET, а другим проектом является проект VC++ Makefile с именем `AppDefinition`, который охватывает коллекцию файлов конфигурации, файлов CMD, таблиц стилей XSLT и других вспомогательных файлов, которые служат для создания и регистрации рассматриваемого приложения рассылки извещений.

Проект VC++ Makefile позволяет привязывать определяемые пользователем команды к командам Build, Clean и Rebuild в интегрированной среде разработки Visual Studio. В проектах Notification Services, создаваемых пользователем на основе образцов, указанные пользовательские команды служат для вызова специализированных файлов CMD, выполняющих операции, соответствующие операциям Build, Clean и Rebuild, для проекта AppDefinition. Результатом становится созданное пользователем готовое решение, которое довольно хорошо интегрируется со средой разработки Visual Studio. Щелкнув правой кнопкой мыши на обозначении проекта AppDefinition в программе Visual Studio Solution Explorer и выбрав команду Build, пользователь фактически вызывает файл CMD, в котором выполняется утилита NSControl. Выходные данные этой утилиты накапливаются в выходном окне, которое так или иначе является частью самой интегрированной среды разработки.

Интересные синтаксические конструкции файлов CMD

Поскольку речь идет о файлах CMD и примерах приложений, автор должен упомянуть, что файлы CMD, которые входят в состав примеров Notification Services, предоставляют превосходную возможность многое узнать, в частности, об утилите NSControl и, в целом, о процессе создания приложения Notification Services. В некоторых из этих файлов CMD встречаются интересные синтаксические конструкции, позволяющие весьма успешно использовать основные возможности синтаксиса файлов CMD операционной системы Windows. Автор считает, что об этих синтаксических конструкциях следует упомянуть, поскольку они позволяют лучше понять, как устроены рассматриваемые примеры приложений, а также косвенно проиллюстрировать принципы работы утилиты NSControl. Если читатель имеет возможность ознакомиться с примерами файлов CMD, которые входят в поставку программного продукта Notification Services, то сможет многое узнать о том, как используется утилита NSControl для создания и управления приложением Notification Services. В табл. 19.1 перечислены некоторые из наиболее интересных синтаксических конструкций файлов CMD, обнаруженных автором в этих файлах, и описано, какие действия выполняют эти конструкции.

Таблица 19.1. Наиболее интересные команды из файлов CMD, применяемых в примерах приложений Notification Services

Команда	Назначение
`if %ERRORLEVEL% GEQ 1 popd & goto Error`	Проверить значение переменной среды ERRORLEVEL; если это значение больше или равно 1, происходит возврат в предыдущий каталог (помещенный в "стек" каталогов с помощью команды pushd) и переход к метке Error
`exit /B 1`	Завершить выполнение текущего командного файла и в ходе завершения задать значение переменной среды ERRORLEVEL

Окончание табл. 19.1

Команда	Назначение
`del %TEMP%\GrantPermissions.out > nul 2>&1`	Удалить указанный файл и перенаправить вывод команды удаления на устройство `nul`, чтобы отменить отображение вывода на экране. Кроме того, в этой команде вывод на устройство `stderr` (системное устройство, в которое по умолчанию записываются ошибки канала операционной системы) также перенаправляется на устройство `stdout`, чтобы на устройство `nul` были перенаправлены и сообщения об ошибках
`start /wait cmd.exe /c call GrantPermissions.cmd`	Открыть новое окно (видимое пользователю) для того, чтобы в утилите `osql` (которая вызывается командой `GrantPermissions.cmd`) можно было запросить любые требуемые пароли
`@if "%_echo%" == "" echo off`	Проверить, задано ли значение переменной среды `_echo`, и в случае отрицательного ответа отменить отображение вывода команды на экране. Такая команда весьма удобна при отладке, поскольку позволяет разрешать или запрещать отображение вывода команд на экране, не редактируя сам командный файл
`setlocal`	Определить условия, согласно которым изменения значений переменных среды остаются локальными по отношению к текущему командному файлу. Такие условия остаются в силе до тех пор, пока не будет выполнена команда `endlocal`. При обычных условиях изменения значений переменных среды в командном файле обнаруживаются во всем текущем сеансе `CMD.EXE`
`echo.`	Вывести на терминал пустую строку. Эта команда позволяет повысить удобочитаемость выходных данных, выводимых на экран командным файлом
`set SqlServer=%SqlServer:"=%`	Удалить из строкового значения перемен-ной среды символ, который предшествует знаку равенства "=" (в данном случае двойную кавычку). Вместо двойной кавычки можно задать любой символ, и программа `CMD.EXE` удалит этот символ из значения переменной среды перед выполнением операции присваивания. Следует отметить, что такая операция удаления символа применима не только в командах `set`, но и в любых других конструкциях командного файла, в которых используется значение переменной среды

Создание собственного приложения рассылки извещений

В данном разделе показано, как создать приложение Notification Services с нуля. Разрабатываемое приложение представляет собой пример службы рассылки извещений о программных ошибках (Bug Notification Service – BNS). Эта служба должна передавать подписчикам извещения о том, что в некоторую таблицу базу данных SQL Server внесена информация о вновь обнаруженных ошибках в интересующих их программных продуктах, и о\том, какие изменения внесены в эти программные продукты. События генерируются с помощью триггера, заданного на таблице Bugs в определяемой пользователем базе данных (находящейся вне приложения Notification Services), а доставка извещений осуществляется по протоколу SMTP. В данном разделе будут сформированы все фрагменты приложения, включая приложение управления подпиской. При подготовке данного упражнения автор исходил из предположения, что читатель имеет в своем распоряжении программные продукты SQL Server, Notification Services и сервер SMTP (сервер SMTP входит в поставку семейства программных продуктов Windows NT), которые установлены на одном и том же компьютере. Если ваш сервер SMTP в данное время работает, остановите его на период разработки приложения, поскольку по результатам работы рассматриваемого примера приложения фактически не предусмотрена отправка какой-либо электронной почты.

Процесс разработки приложения Notification Services

Типичный цикл разработки приложения Notification Services состоит из перечисленных ниже этапов.

1. Создать файл конфигурации экземпляра и файл определения приложения.
2. Воспользоваться командой `Create` утилиты NSControl для создания баз данных экземпляра и приложения с помощью файлов конфигурации.
3. Предоставить соответствующим пользователям права доступа к новым базам данных.
4. Воспользоваться командой `Register` утилиты NSControl для регистрации нового экземпляра и соответствующей этому экземпляру службы Windows.
5. Воспользоваться командой `Enable` утилиты NSControl, чтобы разрешить работу с новым экземпляром и приложением.
6. Выполнить запуск новой службы с помощью команды `NET START` или приложения Services. К этому моменту должен существовать работающий экземпляр серверной части разрабатываемого приложения.
7. Создать приложение управления подпиской и приступить к вводу данных о подписках и событиях для проверки нового приложения рассылки извещений.

Создание файлов конфигурации

Начнем разработку приложения с создания файла конфигурации экземпляра. Для этого будет использоваться самая простая конфигурация, чтобы можно было объяснить наиболее важные понятия, не вдаваясь в мелкие подробности. В листинге 19.2 показан файл конфигурации экземпляра, который будет использоваться в разрабатываемом приложении BNS (код этого файла можно найти в файле `instConfig.XML` подкаталога `CH19\bns\svc` компакт-диска, прилагаемого к данной книге.)

Листинг 19.2. Файл конфигурации экземпляра для приложения BNS

```
<?xml version="1.0" encoding="utf-8"?>
<NotificationServicesInstance xmlns:xsd=
    "http://www.w3.org/2001/XMLSchema" xmlns:xsi=
     "http://www.w3.org/2001/XMLSchema-instance" xmlns=
    "http://www.microsoft.com/MicrosoftNotificationServices/
    ConfigurationFileSchema">
  <InstanceName>BNSInstance</InstanceName>
  <SqlServerSystem>%Server_Instance%</SqlServerSystem>
  <Applications>
    <Application>
      <ApplicationName>BNS</ApplicationName>
      <BaseDirectoryPath>%BasePath%</BaseDirectoryPath>
      <ApplicationDefinitionFilePath>appADF.xml
          </ApplicationDefinitionFilePath>
      <Parameters>
         <Parameter>
          <Name>Server_Instance</Name>
          <Value>%Server_Instance%</Value>
        </Parameter>
         <Parameter>
          <Name>SystemName</Name>
          <Value>%COMPUTERNAME%</Value>
         </Parameter>
         <Parameter>
          <Name>BasePath</Name>
          <Value>%BasePath%</Value>
        </Parameter>
       </Parameters>
    </Application>
  </Applications>
  <DeliveryChannels>
    <DeliveryChannel>
      <DeliveryChannelName>EmailChannel</DeliveryChannelName>
      <ProtocolName>SMTP</ProtocolName>
    </DeliveryChannel>
  </DeliveryChannels>
</NotificationServicesInstance>
```

В файле, приведенном в листинге 19.2, выделены полужирным шрифтом два значения, которые читателю нужно будет откорректировать в соответствии с применяемой им конкретной конфигурацией. Значения указанных параметров можно задать, устанавливая значения переменных среды перед вызовом утилиты

NSControl или передавая их в командной строке NSControl. В данном разделе предусмотрена передача указанных значений в командной строке во время вызова утилиты NSControl, как будет описано ниже.

Первый параметр, которому соответствует узел `SqlServerSystem`, представляет собой имя компьютера и имя экземпляра SQL Server (разделенные обратной косой чертой), на которых должно базироваться разрабатываемое приложение для рассылки извещений. Вторая запись, `BaseDirectoryPath`, представляет собой исходное обозначение пути для файлов конфигурации, таблиц стилей XML и других вспомогательных файлов, которые будут использоваться при создании приложения.

Приведенный файл конфигурации ссылается на одно приложение, BNS, и один канал доставки, `EmailChannel`, в котором используется встроенный протокол доставки SMTP. Обратите внимание на узел `Parameters` в узле `BNS Application`. Узел `Parameters` определяет параметры, передаваемые в файл определения приложения. В отличие от файла конфигурации экземпляра, в файле определения приложения невозможно осуществить выборку параметров из переменных среды или из командной строки NSControl. Поэтому в данном случае просто передаются параметры, первоначально переданные в файл конфигурации экземпляра (наряду с переменной среды `COMPUTERNAME`). В других сценариях могло бы потребоваться использование более сложного способа передачи параметров.

Теперь перейдем к изучению файла определения приложения, `appADF.XML`, показанного в листинге 19.3 (этот файл можно найти в подкаталоге `CH19\bns\svc` компакт-диска, прилагаемого к данной книге).

Листинг 19.3. Файл определения приложения `appADF.XML`

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.microsoft.com/MicrosoftNotificationServices/
↳ApplicationDefinitionFileSchema">
  <EventClasses>
    <EventClass>
      <EventClassName>BNSEvents</EventClassName>
       <Schema>
         <Field>
           <FieldName>ID</FieldName>
           <FieldType>integer</FieldType>
           <FieldTypeMods>not null</FieldTypeMods>
        </Field>
       </Schema>
    </EventClass>
  </EventClasses>
  <SubscriptionClasses>
    <SubscriptionClass>
       <SubscriptionClassName>BNSSubscriptions
          </SubscriptionClassName>
       <Schema>
         <Field>
          <FieldName>DeviceName</FieldName>
          <FieldType>nvarchar(255)</FieldType>
           <FieldTypeMods>not null</FieldTypeMods>
```

```
      </Field>
       <Field>
         <FieldName>SubscriberLocale</FieldName>
         <FieldType>nvarchar(10)</FieldType>
         <FieldTypeMods>not null</FieldTypeMods>
       </Field>
       <Field>
         <FieldName>Product</FieldName>
         <FieldType>nvarchar(30)</FieldType>
         <FieldTypeMods>not null</FieldTypeMods>
       </Field>
       <Field>
         <FieldName>ID</FieldName>
         <FieldType>nvarchar(15)</FieldType>
         <FieldTypeMods>not null</FieldTypeMods>
       </Field>
       <Field>
        <FieldName>OpenedBy</FieldName>
         <FieldType>nvarchar(30)</FieldType>
         <FieldTypeMods>not null</FieldTypeMods>
       </Field>
       <Field>
         <FieldName>AssignedTo</FieldName>
         <FieldType>nvarchar(30)</FieldType>
         <FieldTypeMods>not null</FieldTypeMods>
       </Field>
     </Schema>
     <EventRules>
       <EventRule>
         <RuleName>BNSSubscriptionsRule</RuleName>
         <Action>
          SELECT dbo.BNSInfoNotify(s.SubscriberId,
                                   s.DeviceName,
                                   s.SubscriberLocale,
                                   p.ID,
                                   p.Product,
                                   p.OpenedBy,
                                   p.AssignedTo,
                                   p.Description,
                                   p.DateChanged,
                                   p.Pri,
                                   p.Sev)
          FROM BNSEvents e, BNSSubscriptions s, BNS..Bugs p
          WHERE e.ID = p.ID
            AND p.Product = s.Product
             AND p.OpenedBy LIKE s.OpenedBy
             AND p.AssignedTo LIKE s.AssignedTo
            AND p.ID LIKE s.ID
         </Action>
         <EventClassName>BNSEvents</EventClassName>
      </EventRule>
     </EventRules>
</SubscriptionClass>
</SubscriptionClasses>
<NotificationClasses>
<NotificationClass>
```

```
      <NotificationClassName>BNSInfo</NotificationClassName>
      <Schema>
         <Fields>
    <Field>
      <FieldName>ID</FieldName>
      <FieldType>int</FieldType>
    </Field>
    <Field>
      <FieldName>Product</FieldName>
      <FieldType>nvarchar(30)</FieldType>
    </Field>
    <Field>
      <FieldName>OpenedBy</FieldName>
      <FieldType>nvarchar(30)</FieldType>
    </Field>
    <Field>
      <FieldName>AssignedTo</FieldName>
      <FieldType>nvarchar(30)</FieldType>
    </Field>
    <Field>
      <FieldName>Description</FieldName>
      <FieldType>nvarchar(80)</FieldType>
    </Field>
    <Field>
      <FieldName>DateChanged</FieldName>
      <FieldType>datetime</FieldType>
    </Field>
    <Field>
      <FieldName>Pri</FieldName>
      <FieldType>int</FieldType>
    </Field>
    <Field>
      <FieldName>Sev</FieldName>
      <FieldType>integer</FieldType>
    </Field>
    </Fields>
</Schema>
<ContentFormatter>
    <ClassName>XsltFormatter</ClassName>
    <Arguments>
    <Argument>
      <Name>XsltBaseDirectoryPath</Name>
      <Value>%BasePath%</Value>
    </Argument>
    <Argument>
      <Name>XsltFileName</Name>
      <Value>BNSInfo.xslt</Value>
</Argument>
         </Arguments>
      </ContentFormatter>
       <Protocols>
         <Protocol>
           <ProtocolName>SMTP</ProtocolName>
           <Fields>
             <Field>
              <FieldName>Subject</FieldName>
```

```
                <SqlExpression>'Bug Change Notification'
                    </SqlExpression>
                </Field>
                <Field>
                 <FieldName>BodyFormat</FieldName>
                 <SqlExpression>'html'</SqlExpression>
                </Field>
                <Field>
                  <FieldName>From</FieldName>
                 <SqlExpression>'bns@yourcompany.com'</SqlExpression>
                </Field>
                <Field>
                 <FieldName>Priority</FieldName>
                 <SqlExpression>'Normal'</SqlExpression>
                </Field>
                <Field>
                 <FieldName>To</FieldName>
                  <SqlExpression>DeviceAddress</SqlExpression>
                </Field>
              </Fields>
            </Protocol>
          </Protocols>
      </NotificationClass>
    </NotificationClasses>
    <Providers>
      <NonHostedProvider>
         <ProviderName>SQLTriggerEventProvider</ProviderName>
      </NonHostedProvider>
    </Providers>
    <Generator>
      <SystemName>%SystemName%</SystemName>
    </Generator>
    <Distributors>
       <Distributor>
        <SystemName>%SystemName%</SystemName>
<QuantumDuration>PT5S</QuantumDuration>
       </Distributor>
    </Distributors>
    <ApplicationExecutionSettings>
      <QuantumDuration>PT5S</QuantumDuration>
      <Vacuum>
        <RetentionAge>P3DT00H00M00S</RetentionAge>
        <VacuumSchedule>
           <Schedule>
             <StartTime>23:00:00</StartTime>
             <Duration>P0DT02H00M00S</Duration>
           </Schedule>
           <Schedule>
             <StartTime>03:00:00</StartTime>
             <Duration>P0DT02H00M00S</Duration>
           </Schedule>
        </VacuumSchedule>
      </Vacuum>
    </ApplicationExecutionSettings>
</Application>
```

Автор выделил в листинге 19.3 полужирным шрифтом параметры, передаваемые из файла конфигурации экземпляра. Как было указано выше, файл определения приложения никогда не передается непосредственно в утилиту NSControl. Если при вызове утилиты NSControl необходимо указать файл конфигурации экземпляра, то данной утилите передается файл конфигурации экземпляра, который, в свою очередь, ссылается на файлы определения приложения, которые относятся к приложениям, базирующимся на данном экземпляре.

Узлы `SystemName` в секциях `Generator` и `Distributor` ссылаются на имя компьютера, на котором базируется рассматриваемое приложение рассылки извещений. Узел `XsltBaseDirectoryPath` указывает путь, содержащий таблицы стилей XML, которые предназначены для использования средством форматирования информационного наполнения XSLT.

Как указывает сам термин "файл определения приложения", этот файл определяет способ использования приложения для рассылки извещений. Узлы классов событий указывают, каковыми являются события, а узлы классов подписок и извещений определяют, каковыми являются подписки и извещения. Секции с данными о генераторе и распределительном сервере содержат информацию о том, как функционируют и какие действия выполняют генератор и распределительный сервер. Секция с определениями средств доступа задает тип используемого средства доступа к данным о событиях. В рассматриваемом примере приложения доступ к информации о событиях предоставляется с помощью триггеров SQL Server, как было отмечено выше. Секция с определением средств очистки указывает, насколько часто происходит удаление из системы устаревших данных (например, данных о доставленных извещениях). Поскольку данные о событиях и извещениях продолжают накапливаться в базах данных, и вследствие этого объем данных со временем значительно возрастает, необходимо регулярно проводить очистку.

ПРЕДОСТЕРЕЖЕНИЕ. Примеры приложений, которые входят в поставку программного продукта Notification Services, не содержат секции с данными о средствах очистки в своих файлах определения приложения. Таким образом, после применения утилиты `CopySample` для копирования одного из таких файлов в целях последующей корректировки и использования в конкретном приложении обнаруживается, что новое приложение не имеет заданной по умолчанию секции с описанием средств очистки в своем файле определения приложения, и поэтому в приложении не происходит очистка устаревших данных. Секцию с описанием средств очистки можно ввести в файл определения приложения перед созданием приложения с помощью функции `Create` утилиты NSControl или ввести эту секцию впоследствии и воспользоваться функцией `Update` утилиты NSControl для ввода изменений в действие. Кроме того, операцию очистки можно вызвать на выполнение вручную, выполнив хранимую процедуру `NSVacuum`. Пользователям, занимающимся очисткой устаревших данных, должна быть назначена роль `NSVacuum`.

Обратите внимание на то, какой способ формирования правила согласования применяется в секции `EventRules`. Конструкция этого правила, обеспечивающая соединение таблиц, выглядит примерно следующим образом:

```
WHERE e.ID = p.ID
    AND p.Product = s.Product
    AND p.OpenedBy LIKE s.OpenedBy
    AND p.AssignedTo LIKE s.AssignedTo
    AND p.ID LIKE s.ID
```

В предикатах этого запроса используется ключевое слово LIKE, которое позволяет применять в определении подписки подстановочные символы вместо фактических значений. Поэтому столбцы OpenedBy, AssignedTo и ID (идентификатор ошибки), по существу, являются необязательными. Если эти столбцы не заданы в приложении управления подпиской, то в графическом пользовательском интерфейсе вместо обозначений этих столбцов может быть вставлено значение "%", что позволяет фактически игнорировать значения этих столбцов при определении критериев выборки запроса. Такой подход нельзя назвать наиболее эффективным с точки зрения использования индексов, но в определенной степени он обеспечивает динамическое формирование правил согласования. Еще один способ динамического создания правил согласования рассматривается ниже в данной главе.

Создание определяемой пользователем базы данных

Еще одно действие, которое должно быть осуществлено перед вызовом на выполнение утилиты NSControl для создания, регистрации и разрешения доступа к приложению рассылки извещений, состоит в создании определяемой пользователем базы данных, в которой хранится таблица Bugs. Как было указано выше, на этой таблице должен быть задан триггер, который вырабатывает извещение после каждого добавления информации о новой обнаруженной программной ошибке или корректировки существующей. В листинге 19.4 представлен сценарий T-SQL, предназначенный для создания базы данных BNS. (Код этого сценария можно найти в файле CreateBNS.SQL подкаталога \CH19\bns\svc компакт-диска, прилагаемого к данной книге.)

Листинг 19.4. Код сценария CreateBNS.SQL

```
USE master
GO
IF (DB_ID('BNS') IS NOT NULL)
  DROP DATABASE BNS
GO

CREATE DATABASE BNS
GO

USE BNS
GO

IF ('BNS'<>DB_NAME()) RAISERROR('Database create failed.
    Aborting.',25,1) WITH LOG
GO
```

```
CREATE TABLE Bugs (
  ID             int,
  Product       nvarchar(30),
  OpenedBy      nvarchar(30),
  AssignedTo    nvarchar(30),
  Description   nvarchar(80),
  DateChanged   datetime,
  Pri           int,
  Sev           int,
  Status        int,
  BugText        text,
  Repro         text
primary key (ID))
GO
CREATE TRIGGER BugTrigger ON Bugs
FOR INSERT, UPDATE
AS
BEGIN
  DECLARE @EventBatchId bigint

  -- Открыть пакет событий
  EXEC BNSInstanceBNS..NSEventBeginBatchBNSEvents
    @ProviderName = N'SQLTriggerEventProvider',
    @EventBatchId = @EventBatchId OUTPUT

  DECLARE @BugID integer
DECLARE NewBugCursor CURSOR
  LOCAL FAST_FORWARD
  FOR Select ID from inserted

  OPEN NewBugCursor

  FETCH NEXT FROM NewBugCursor INTO @BugId
  WHILE @@FETCH_STATUS = 0
  BEGIN
    -- Записать событие в пакет
    EXEC BNSInstanceBNS..NSEventWriteBNSEvents
      @EventBatchId,
      @BugId

    FETCH NEXT FROM NewBugCursor INTO @BugId
  END

  CLOSE NewBugCursor
  DEALLOCATE NewBugCursor

  -- Закрыть пакет событий
  EXEC BNSInstanceBNS..NSEventFlushBatchBNSEvents @EventBatchId

END
GO

EXECUTE sp_grantdbaccess guest
GO
```

В сценарии, приведенном в листинге 19.4, создается новая база данных, BNS, а затем в этой базе данных формируется таблица Bugs для хранения записей с данными о программных ошибках. После этого на таблице создается триггер, который активизируется после применения операции INSERT или UPDATE к таблице и формирует соответствующие извещения. Вызовите этот сценарий на выполнение, чтобы создать базу данных BNS, после чего можно будет приступить к созданию приложения рассылки извещений. (Обнаружив предупреждающие сообщения, которые касаются недостающих объектов и невозможности задать информацию о зависимостях в таблице sysdepends, игнорируйте эти сообщения.)

Создание баз данных экземпляра и приложения

После создания базы данных BNS добавьте каталог bin службы Notification Services к значению системной переменной, которая показывает применяемые по умолчанию пути доступа к файлам, чтобы можно было вызывать утилиту NSControl, не уточняя полностью ее местонахождение в файловой системе. Этот каталог должен иметь обозначение пути доступа \Program Files\Microsoft SQL Server Notification Services\vN.N.N.N\bin на том диске, на котором установлены службы Notification Services. После добавления каталога bin к системной переменной с обозначением пути доступа откройте командное окно и перейдите в подкаталог CH19\bns\svc компакт-диска, прилагаемого к данной книге. После этого вызовите на выполнение утилиту NSControl для создания баз данных экземпляра и приложения с использованием следующей командной строки (она должна быть введена в виде одной строки):

```
nscontrol create -in instConfig.xml Server_Instance=
↳YourServer\YourSSInstance BasePath=D:\CH19\bns\svc
```

Параметры Server_Instance и BasePath передаются в файл конфигурации экземпляра. Как было указано выше, Server_Instance — это имя компьютера *YourServer* и имя экземпляра SQL Server *YourSSInstance*, разделенные обратной косой чертой. Параметр BasePath обозначает каталог, в котором находятся файлы конфигурации, таблицы стилей XML и другие вспомогательные файлы, относящиеся к экземпляру. Перед вызовом на выполнение утилиты NSControl замените каждое из указанных значений соответствующими значениями, относящимися к конкретной системе.

Регистрация экземпляра

После создания баз данных экземпляра и приложения можно приступить к регистрации экземпляра. Безусловно, экземпляр можно зарегистрировать, не регистрируя при этом также его службу Windows, но в рассматриваемом сценарии не имеет смысла этого делать, поэтому регистрация экземпляра и его службы будет проводиться одновременно. Для регистрации экземпляра используется утилита NSControl, как показано ниже (введите приведенную ниже команду на одной строке).

```
nscontrol register -name BNSInstance -server YourServer\YourSSInstance
↳ -service -serviceusername YourSvcUser -servicepassword YourSvcPwd
```

Укажите вместо параметров *YourServer\YourSSInstance* имя компьютера и имя экземпляра SQL Server, на котором базируется рассматриваемое приложение рассылки извещений. Настоящее упражнение подготовлено на основе того предположения, что и программа SQL Server, и приложение Notification Services должны находиться на одном и том же компьютере. Укажите соответственно вместо параметров *YourSvcUser* и *YourSvcPwd* имя пользователя и пароль учетной записи NT, от лица которой должна работать служба `NSService.exe`. Обычно принято, что пользователи создают специальную учетную запись исключительно для этой цели. Если вы также будете придерживаться этого правила, то обязательно предоставьте данной учетной записи соответствующие права доступа, которые изложены в инструкциях по теме "`NS$instance_name` *Service Account Security*" (Защита учетной записи службы `NS$instance_name`) в оперативной документации Books Online, относящейся к службам Notification Services.

Разрешение на использование экземпляра

После регистрации экземпляра необходимо разрешить его использование, чтобы иметь возможность приступить с помощью экземпляра к выработке извещений. При этом можно избирательно разрешать или запрещать использование отдельных частей приложения, например, можно разрешить сбор информации о событиях, но запретить применение генератора. В данном случае необходимо разрешить работу всего приложения, поэтому требуется просто передать имя экземпляра *BNSInstance* утилите NSControl в командной строке следующим образом:

```
nscontrol enable -n BNSInstance
```

(Опыт автора показывает, что параметр `-name` обычно можно сокращенно задавать как `-n`.)

Разрешить использование экземпляра можно либо до, либо после запуска службы Windows. Если использование экземпляра разрешается до запуска службы (что и было сделано в приведенной выше команде), то зачастую обнаруживается, что некоторые из компонентов приложения выполнения `Enable` утилиты NSControl переходят в приостановленное состояние. Но такая ситуация не свидетельствует о каких-либо нарушениях в работе. Приостановленные компоненты перейдут в оперативный режим после запуска службы. Этап запуска рассматривается в следующем разделе.

Запуск службы

Запуск новой службы рассылки извещений может быть выполнен с помощью приложения Services или команды `NET START` операционной системы Windows. Сам автор относится к старшему поколению разработчиков, поэтому обычно использует команду `NET START` примерно таким образом:

```
NET START NS$BNSInstance
```

После запуска служба полностью готова и может приступать к доставке извещений. Теперь достаточно только создать приложение управления подпиской и начать ввод подписок и событий для проверки нового приложения рассылки извещений.

Создание приложения управления подпиской

Как было указано выше, для создания копии одного из примеров приложений, поставляемых в составе служб Notification Services, чтобы можно было затем приспосабливать это приложение для собственных нужд, следует использовать утилиту `CopySample`. Такая операция в данном упражнении не предусматривается. Вместо этого новое приложение управления подпиской будет создано с нуля в среде Visual Basic .NET. Это приложение должно представлять собой, скорее, традиционное приложение Windows, чем более широко распространенное приложение ASP.NET, поскольку автор стремится показать, какие задачи должны быть решены для обеспечения взаимодействия с объектной моделью Notification Services с наименьшими сложностями. Подход, предусматривающий создание приложения управления подпиской в виде традиционного приложения Windows, не совсем оправдан на практике, поскольку по умолчанию это приложение должно работать на том же компьютере, что и приложение рассылки извещений. Если приложение управления подпиской представляет собой приложение ASP.NET, то указанная проблема не столь велика, ведь независимо от того, за какими компьютерами работают пользователи приложения, само приложение выполняется на сервере IIS, а этот сервер способен вполне разделять свой базовый компьютер с приложением рассылки извещений. Но если приложение управления подпиской представляет собой приложение Windows, то оно работает в контексте того пользовательского компьютера, на котором выполняется. Поэтому по умолчанию пользователю приходится эксплуатировать приложение управления подпиской на том же компьютере, на котором базируется приложение рассылки извещений, для того, чтобы управление подпиской осуществлялось должным образом.

Существует возможность обеспечить дистанционный доступ к объектной модели Notification Services с использованием технологий COM Interop и DCOM. Но, как правило, следует создавать приложения управления подпиской как приложения ASP.NET, чтобы упростить организацию программного обеспечения и исключить ситуации, при которых в приложении смешиваются COM-объекты и классы управляемого кода. Приложение, представленное в данной главе, не предназначено для использования на производстве.

Как уже было сказано, автор отказался от создания приложения управления подпиской, рассматриваемого в данном примере, в качестве приложения ASP.NET, поскольку стремился добиться максимального упрощения представленного примера. Дело в том, что при создании приложений, действующих без поддержки логических соединений (основанных на технологии Web), приходится учитывать многие нюансы и преодолевать значительные сложности, с которыми не приходится сталкиваться при создании традиционных приложений Windows. А цель автора, которую он преследует в данном разделе, состоит не в подготовке приложения управления подпиской, которое можно было бы немедленно установить в производственной системе, а, скорее, в демонстрации того, насколько легко можно обеспечить использование объектной модели Notification Services и взаимодействие с этой объектной моделью.

Автор должен также упомянуть, что в данном примере используется язык Visual Basic .NET, а не C#, поскольку, по мнению автора, значительная часть

квалифицированных пользователей программы SQL Server имеют подготовку в области языка Visual Basic, а не занимаются разработкой на языках C, C++ или C#. (Сам автор во многом предпочитает язык C# и использует его гораздо чаще.) Но с учетом того, что все примеры приложений Notification Services, за исключением примера `Flight`, написаны на языке C#, а также зная о том, что большинство квалифицированных пользователей программы SQL Server с большей вероятностью предпочитают Visual Basic, автор разработал описанное ниже приложение управления подпиской на языке Visual Basic .NET. Основную часть кода данного приложения можно легко применить повторно в приложении ASP.NET, написанном на языке Visual Basic .NET без изменений или с небольшими изменениями. Все примеры приложений управления подпиской, которые входят в поставку служб Notification Services, представляют собой приложения ASP.NET, поэтому читатель может ознакомиться с ними самостоятельно, чтобы понять, как можно реализовать собственное приложение управления подпиской на основе ASP.NET.

В примерах приложений Notification Services используется один общий вспомогательный класс, который хранится в файле `NSUtility.cs`. Этот класс обеспечивает взаимодействие с объектной моделью Notification Services с помощью еще более простых средств по сравнению с уже имеющимися, и широко используется в примерах приложений. Учитывая то, что рассматриваемый в данном разделе пример приложения управления подпиской разработан на языке Visual Basic, а также то, что модуль C# нельзя использовать в проекте Visual Basic .NET без его предварительной компиляции в виде сборки, автор перевел значительную часть функций, реализованных в классе `NSUtility.cs`, на язык Visual Basic .NET. Читатель вправе извлечь эти функции из исходного кода, который вскоре будет рассматриваться в данном разделе, и применить их в собственных приложениях управления подпиской на основе языка Visual Basic .NET.

Прежде всего для обеспечения взаимодействия приложения с управляемым кодом и объектной моделью Notification Services необходимо ввести в свой проект ссылку на сборку `Microsoft.SqlServer.NotificationServices`. После загрузки готового решения `BNSSubscribe` в среду Visual Studio .NET можно обнаружить, что ссылка на эту сборку действительно применяется.

Итак, без дальнейших комментариев рассмотрим исходный код приложения `BNSSubscribe` — приложения управления подпиской для `BNS` (листинг 19.5). Учитывая то, что большая часть исходного кода этого приложения была сгенерирована средой Visual Studio .NET, автор не будет подробно рассматривать все это приложение. Поэтому ниже описан только тот код, который был написан автором для реализации данного приложения, и затронуты наиболее важные вопросы, связанные с его функционированием. (Этот код можно найти в файле `Form1.vb` подкаталога `\CH19\bns\BNSSubscribe` компакт-диска, прилагаемого к данной книге. Язык VB с самого начала разрабатывался с учетом построчного размещения операторов программы, поэтому часть кода этого приложения не удалось удобно отформатировать для его представления на печатной странице; чтобы ознакомиться с листингом, более удобным для чтения, откройте сам файл исходного кода.)

Листинг 19.5. Файл `Form1.vb`

```
Private Function CreateDataSource
         (ByVal subscriptionEnumeration As _SubscriptionEnumeration)
        As DataSet
      Dim ds As DataSet = New DataSet()
      Dim dt As DataTable = New DataTable("Subscriptions")
       ds.Tables.Add(dt)

       Dim dr As DataRow

      dt.Columns.Add(New DataColumn("SubscriptionId",
          System.Type.GetType("System.Int32")))
      dt.Columns.Add(New DataColumn("Product", System.Type.GetType
            ("System.String")))
      dt.Columns.Add(New DataColumn("ID", System.Type.GetType
            ("System.String")))
       dt.Columns.Add(New DataColumn("OpenedBy", System.Type.GetType
            ("System.String")))
      dt.Columns.Add(New DataColumn("AssignedTo",
          System.Type.GetType("System.String")))

      Dim subscription As Subscription
      For Each subscription In subscriptionEnumeration
        dr = dt.NewRow()
        Dim i As Integer
        Dim name As String
         For i = 0 To dt.Columns.Count - 1
           name = dt.Columns(i).ColumnName

          If (0 = String.Compare(name, "SubscriptionId", True))
              Then
            dr(name) = subscription.SubscriptionId
           Else
             dr(name) = subscription(name)
          End If
         Next
        dt.Rows.Add(dr)
         Next

        Return ds
    End Function

    Public Sub UpdateGrid(ByVal userName As String)
      Dim subscriptionEnumeration As SubscriptionEnumeration
      subscriptionEnumeration = New SubscriptionEnumeration
          (application, subscriptionClassName, userName)

      dgSubscriptions.SetDataBinding(CreateDataSource
          (subscriptionEnumeration), "Subscriptions")
    End Sub

    Public Function GetDeliveryChannel(ByVal protocolName As
         String) As String
      Dim deliveryChannelEnumeration As DeliveryChannelEnumeration
           = New DeliveryChannelEnumeration(instance)
```

```
      Dim deliveryChannel As IDeliveryChannel
      For Each deliveryChannel In deliveryChannelEnumeration
        If deliveryChannel.ProtocolName = protocolName Then
           Return deliveryChannel.DeliveryChannelName
         End If
      Next
      Return Nothing
    End Function

    Public Function GetSubscriberDeviceName(ByVal subscriberId As
         String) As String
      Dim subscriberDeviceName As String = Nothing
      Dim subDeviceEnumeration As SubscriberDeviceEnumeration
      subDeviceEnumeration = New SubscriberDeviceEnumeration
      (instance, subscriberId)

    If Not subDeviceEnumeration Is Nothing Then
    Dim subscriberDevice As SubscriberDevice
    For Each subscriberDevice In subDeviceEnumeration
      subscriberDeviceName = subscriberDevice.DeviceName
    Next
    End If

    Return subscriberDeviceName
End Function

Private Sub AddSubscriber(ByVal subscriberId As String, ByVal
    protocolName As String, ByVal emailaddress As String)
    Try
    Dim subscriber As Subscriber = New Subscriber(instance)

    subscriber.SubscriberId = subscriberId

    subscriber.Add()

    Dim subscriberDevice As SubscriberDevice =
        New SubscriberDevice(instance)

    Dim deliveryChannelName As String =
        GetDeliveryChannel(protocolName)

    subscriberDevice.SubscriberId = subscriberId
    subscriberDevice.DeviceTypeName = protocolName
    subscriberDevice.DeviceName = "myDevice"
    subscriberDevice.DeviceAddress = emailaddress
    subscriberDevice.DeliveryChannelName = deliveryChannelName

    subscriberDevice.Add()
    Catch ex As NSException
    If (NSEventEnum.DuplicateSubscriber <> ex.ErrorCode) Then
      Throw (ex)
    End If
    End Try
End Sub
Public Function AddSubscription(ByVal subscriberId As String, _
                          ByVal subscriptionClassName As String, _
                          ByVal subscriptionFields As Hashtable, _
```

```
                        ByVal dateTimeStart As String, _
                        ByVal recurrence As String)

    Dim subscription As Subscription = New Subscription
        (application, subscriptionClassName)
    subscription.SubscriberId = subscriberId

    Dim entry As DictionaryEntry
    For Each entry In subscriptionFields
      Dim fieldName As String = entry.Key
      Dim fieldValue As Object = entry.Value
      subscription(fieldName) = fieldValue
    Next

    If subscription.HasTimedRule Then
      subscription.ScheduleStart = dateTimeStart
      subscription.ScheduleRecurrence = recurrence
    End If

    Return subscription.Add()
  End Function

  Private Sub DeleteSubscription(ByVal subscriberId As String,
      ByVal subscriptionClassName As String,
      ByVal subscriptionIdString As String)
    Dim subscriptionEnumeration As SubscriptionEnumeration =
        New SubscriptionEnumeration(application,
        subscriptionClassName, subscriberId)

    Dim subscription As Subscription = subscriptionEnumeration
        (subscriptionIdString)

    subscription.Delete()
  End Sub

  Private Sub AddSub(ByVal Product As String,
      ByVal Bug As String, ByVal OpenedBy As String,
      ByVal AssignedTo As String)
    Dim subscriberDeviceName As String = Nothing
    Dim subscriptionId As String = Nothing
    Dim subscriptionFields As Hashtable = Nothing
    Dim bugmask As String = "%"
    Dim openedbymask As String = "%"
    Dim assignedtomask As String = "%"

    Try
      Try
        If 0 <> OpenedBy.Length Then openedbymask =
            tbOpenedBy.Text
        If 0 <> AssignedTo.Length Then assignedtomask =
            tbAssignedTo.Text
        If 0 <> Bug.Length Then bugmask =
            Int32.Parse(tbBug.Text).ToString()
      Catch ex As Exception
        Throw New Exception("Invalid Bug ID specified.", ex)
      End Try
```

```
        AddSubscriber(userName, "SMTP", tbEmail.Text)

        subscriberDeviceName = GetSubscriberDeviceName(userName)

        subscriptionFields = New Hashtable()

        subscriptionFields.Add("DeviceName", subscriberDeviceName)
        subscriptionFields.Add("SubscriberLocale", "en-US")
        subscriptionFields.Add("Product", Product)
        subscriptionFields.Add("ID", bugmask)
        subscriptionFields.Add("OpenedBy", openedbymask)
        subscriptionFields.Add("AssignedTo", assignedtomask)

        subscriptionId = AddSubscription(userName,
            subscriptionClassName, subscriptionFields,
            Nothing, Nothing)

        UpdateGrid(userName)

        sbMsg.Text = "Subscription successfully added."
      Catch ex As Exception
        sbMsg.Text = "Cannot add subscription: " + ex.Message
      End Try
    End Sub

    Private Sub btAdd_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles btAdd.Click
      AddSub(cbProduct.SelectedItem, tbBug.Text, tbOpenedBy.Text,
          .tbAssignedTo.Text)
    End Sub
Private Sub btDelete_Click(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles btDelete.Click
      If (-1 = dgSubscriptions.CurrentRowIndex) Then Exit Sub
       Dim subscriptionIdString As String =
           dgSubscriptions.Item(dgSubscriptions.CurrentCell)

      Try
        DeleteSubscription(userName, subscriptionClassName,
            subscriptionIdString)

         sbMsg.Text = "Subscription deleted"

        UpdateGrid(userName)
      Catch ex As Exception
        sbMsg.Text = "Cannot delete the subscription: " +
            ex.Message
      End Try
    End Sub
End Class
```

Рассмотрим основные элементы этого приложения. Прежде всего в конструкторе класса формы создаются экземпляры объектов `NSInstance` и `NSApplication`. В данный листинг не включен код конструктора, поскольку основная часть этого кода вырабатывается в среде Visual Studio .NET, но ниже приведены конкретные операторы, с помощью которых создаются экземпляры указанных объектов.

```
instance = New NSInstance(instanceName)
application = New NSApplication(instance, applicationName)
```

Переменные instance и application являются переменными экземпляра класса формы, поэтому доступны из всех методов класса. После создания экземпляров объектов их можно использовать для выполнения других вызовов к объектной модели Notification Services.

Вслед за этапом создания указанных объектов осуществляется выборка данных о подписках, относящихся к текущему пользователю, и обновляется таблица в нижней части формы, содержащая перечень подписок. Эти действия выполняются с помощью метода UpdateGrid. Как показывает исходный код, приведенный в листинге 19.5, при этом создается новый объект SubscriptionEnumeration и дескриптор этого объекта передается в метод CreateDataSource для создания источника данных (в данном случае объекта DataSet), применимого для использования с элементом управления DataGrid. В метод UpdateGrid передается имя текущего пользователя (выборка которого осуществляется при запуске приложения с помощью метода System.Security.Principal.WindowsIdentity.GetCurrent.Name), которое применяется в качестве критерия выборки данных о подписках для подготовки перечня подписок, отображаемого в таблице формы.

Кроме того, интерес представляет код методов btAdd_Click и AddSub. Этот код вызывается на выполнение после того, как пользователь щелкает на кнопке Add в графическом интерфейсе пользователя для добавления подписки. Метод btAdd_Click вызывает метод AddSub, а метод AddSub выполняет реальную работу по добавлению подписки. Эта работа начинается с ввода данных о подписчике (в том случае, если эти данные еще не введены), затем происходят операции выборки информации об устройстве доставки подписчика и добавления подписки. Как было указано выше, в графическом интерфейсе пользователя предусмотрены значения подстановочных символов для полей подписки, которые остались незаполненными. Это позволяет, по существу, определять значения этих полей как необязательные при вызове на выполнение правил согласования.

Последняя часть кода, которая будет рассматриваться в данном разделе, относится к методу btDelete_Click. Очевидно, что этот код вызывается на выполнение после того, как пользователь выбирает некоторое определение подписки в элементе управления DataGrid и щелкает кнопкой Delete, чтобы удалить соответствующую подписку. В методе btDelete_Click вызывается метод DeleteSubscription, в основе которого лежит код, взятый автором из файла исходного кода NSUtility.cs на языке C#, который входит в поставку примеров приложений служб Notification Services, и переведенный на язык VB.

Остальная часть кода в модуле исходного кода, приведенном в листинге 19.5, представляет собой вспомогательный код для двух основных функций приложения – добавления и удаления подписок. Безусловно, приложение управления подпиской должно предоставлять гораздо более широкий перечень функциональных возможностей по сравнению с реализованными в данном приложении, но нельзя и отрицать, что в любом создаваемом приложении управления подпиской необходимо иметь по меньшей мере функции добавления и удаления подписок.

Проверка приложения рассылки извещений

Чтобы иметь возможность проверить приложение рассылки извещений, необходимо воспользоваться методом BNSSubscribe для добавления подписки. Вызовите приложение на выполнение из подкаталога \CH19\bns\BNSSubscribe\bin и добавьте подписку на информацию об обнаруженных программных ошибках, относящуюся к программному продукту Sequin. (Если бы потребовалось обеспечить фактическую доставку извещений, то нужно было бы откорректировать значение в поле адреса электронной почты, указав действительный адрес; пока об этом не следует беспокоиться.) Оставьте остальные поля ввода пустыми и щелкните на кнопке Add, чтобы добавить подписку. После успешного добавления подписки запустите программу Query Analyzer и выполните следующий запрос:

```
USE BNS
GO
INSERT INTO Bugs
VALUES(1, 'Sequin', '','','', getdate(), 1, 1, 1, '','')
```

После выполнения этого запроса откройте программу Windows Explorer и перейдите в каталог сбора писем сервера SMTP. В операционных системах семейства Windows NT этот каталог имеет имя \Inetpub\mailroot\pickup. Через несколько секунд вы должны обнаружить появление в данном каталоге файла с расширением .eml. Если при этом на вашем компьютере работает сервер SMTP, то он должен забрать этот файл и предпринять попытку отправить содержащееся в файле письмо получателю. Примите наши поздравления – вы только что сформировали свое первое извещение с помощью служб Notification Services!

Возможные усовершенствования

Читатель вполне может представить себе, что описанное в данной главе приложение можно усовершенствовать во многих отношениях и сделать его более подходящим для использования на производстве. Прежде всего, как уже было сказано, приложение управления подпиской должно быть приложением Web, если это программное обеспечение предназначено для применения на производстве. Еще одно замечание состоит в том, что можно разрешить использование резюмированных извещений, для того чтобы подписчиков не захлестывали лавины извещений в тех случаях, когда неожиданно происходит добавление или изменение огромного количества записей с информацией об ошибках, передаваемой в виде извещений. Переход к использованию режима доставки резюмированных извещений позволяет группировать извещения в пакеты извещений, передаваемые подписчикам в виде одного письма по электронной почте, что может способствовать существенному сокращению количества писем с извещениями, доставляемых по электронной почте.

Динамически определяемые правила согласования

Еще одним полезным усовершенствованием данного приложения могло бы стать введение открытой спецификации правил согласования, позволяющей

пользователю задавать более сложные критерии подписки. Как было указано выше, применение подстановочных символов для назначения необязательных столбцов в запросе с правилом согласования — это не самый эффективный подход с точки зрения использования индексов. Если в запросе применяется простой подстановочный символ "%", это фактически гарантирует, что выборка данных из таблицы будет осуществляться с помощью полного просмотра (а если на таблице задан кластеризованный индекс, то план выполнения запроса покажет, что используется просмотр кластеризованного индекса, но полный просмотр и просмотр кластеризованного индекса семантически и функционально эквивалентны). Лучший подход мог бы предусматривать полное удаление из запроса упоминаний о столбцах, которые пользователь не желает применять в качестве критерия выборки данных, относящихся к подписке. Но с учетом того, что для всех подписчиков используется одно и то же правило согласования, реализация такого подхода — нелегкая задача.

Один из методов осуществления подхода, предусматривающего полное удаление столбцов из запроса, состоит в создании динамически определяемого правила согласования. Для этого могут использоваться некоторые приемы работы в языке T-SQL, которые были показаны автором в его предыдущих книгах. По существу, способ применения динамически определяемого правила согласования предусматривает проведение описанных ниже действий.

1. Графический интерфейс пользователя должен представлять собой интерфейс наподобие формирователя запросов, который не просто позволяет задавать значения для фиксированного набора полей с критериями выборки данных о подписке, но и дает возможность подписчику создавать запрос любого типа к основополагающим данным, который ему потребуется (в рассматриваемом случае — запрос к таблице Bugs). Применяемое средство форматирования запроса должно предоставлять возможность использования логических операторов (AND, OR, NOT) и позволять указывать в составе критерия выборки данных о подписке любое поле из основополагающей таблицы.
2. После того как пользователь щелкнет на кнопке Add в графическом интерфейсе пользователя для добавления подписки в базу данных приложения, графический интерфейс пользователя создает объект представления SQL Server (незаметно для пользователя), который инкапсулирует заданные критерии выборки данных о подписке. Каждое подобное представление возвращает из основополагающей таблицы все столбцы и все строки, которые соответствуют конкретным критериям, указанным пользователем. Затем средствами графического интерфейса пользователя информация о подписке добавляется в базу данных приложения с использованием имени указанного представления в качестве единственного критерия.
3. После вызова на выполнение правило согласования динамически вырабатывает запрос T-SQL, в котором берутся имена представлений, ранее созданных с помощью графического интерфейса пользователя, и осуществляется конкатенация представлений в большие запросы с оператором UNION ALL, количество которых сводится к минимуму. Текст каждого из динами-

чески определяемых запросов имеет длину меньше 8 тысяч байтов в связи с ограничениями, налагаемыми на данные типа varchar. При этом каждый из запросов используется для выработки извещений так же, как применяется оператор SELECT обычного правила согласования. Конечным результатом становится то, что определяемые пользователем критерии выборки данных о подписке, заданные в каждой подписке, используются для управления процессом рассылки извещений, не требуя применения курсоров или каких-то других менее эффективных механизмов.

Описанный выше подход можно проще всего понять на примере, подобном приведенному в листинге 19.6 (код этого примера можно найти в сценарии dynamic_filter.sql подкаталога \CH19\bns\svc компакт-диска, прилагаемого к данной книге).

Листинг 19.6. Код сценария dynamic_filter.sql

```
DECLARE @strQry varchar(8000)  -- Строка динамического запроса
DECLARE @strLastView varchar(128)  -- Имя последнего обработанного
                                   -- представления
DECLARE @nViewNameLen int  -- Максимальная длина имени представления
DECLARE @nMaxViewsPerQuery int  -- Общее количество представлений,
                           -- которое может быть обработано в расчете
                           -- на каждый динамический запрос
DECLARE @nViewQryLen int   -- Общая длина строки в расчете на каждое
                           -- представление
DECLARE @strSelectFrag varchar(30)  -- Часть запроса, относящаяся к
                                    -- оператору SELECT
DECLARE @strUnionFrag varchar(30)   -- Часть запроса, относящаяся к
                                    -- оператору UNION
DECLARE @nNumViews int  -- Общее количество запросов, подлежащих
                        -- обработке

SET @strSelectFrag='SELECT * FROM '  -- Часть динамического запроса с
                                     -- оператором SELECT
SET @strUnionFrag=' UNION ALL '  -- Часть динамического запроса с
                                 -- оператором UNION
SET @nViewNameLen=7  -- Имена имеют формат V######, что позволяет
                     -- задавать до 999 999 представлений

SET @nViewQryLen=@nViewNameLen+
  DATALENGTH(@strSelectFrag)+DATALENGTH(@strUnionFrag)

SET @nMaxViewsPerQuery=8000 / @nViewQryLen
SET @strQry=''

SELECT @nNumViews=COUNT(*)
FROM sysobjects
WHERE type='V'
AND name LIKE 'V%'

SET @strLastView=''

SET ROWCOUNT @nMaxViewsPerQuery
```

```
-- Удалить таблицу matchtable
-- select * into matchtable from authors where 0=1

WHILE (@nNumViews>0) BEGIN

  SELECT @strQry=@strQry+
         @strSelectFrag+
         cast(name as varchar(128))+
         @strUnionFrag,
         @strLastView=cast(name as varchar(128))
  FROM sysobjects
  WHERE type='V'
  AND name LIKE 'V%'
  AND name > @strLastView
  ORDER BY name

  -- Удалить последний оператор UNION ALL
  SET @strQry=LEFT(@strQry,DATALENGTH(@strQry)-
       DATALENGTH(@strUnionFrag))

SET ROWCOUNT 0
  INSERT matchtable
  EXEC(@strQry)
SET ROWCOUNT @nMaxViewsPerQuery

  SET @nNumViews=@nNumViews-@nMaxViewsPerQuery
  SET @strQry=''

END

SET ROWCOUNT 0
```

В листинге 19.6 наибольший интерес представляет код, находящийся в цикле WHILE. В этом коде используется ряд интересных способов, позволяющих избежать применения курсора и охватить одновременно с помощью запроса максимально возможное количество представлений с критериями выборки информации о подписке. Во-первых, в коде используется прием с оператором конкатенации (впервые продемонстрированный автором в его книге *The Guru's Guide to Transact-SQL*) для динамического формирования строки кода T-SQL, в котором применяются операторы SELECT для одновременной обработки максимально возможного количества представлений с критериями выборки. Во-вторых, переменной @strLastView присваивается имя последнего представления, обработанного в каждой итерации, для контроля над тем, какие имена представлений остались еще не обработанными. С учетом того, что длина строки кода динамически определяемого запроса ограничивается значением 8 тыс. символов, в сценарии вычисляется количество представлений с критериями выборки, которые могут быть обработаны в каждой итерации, затем используется команда SET ROWCOUNT для задания предельного значения количества имен представлений, доступных в цикле, равного вычисленному значению количества. Для этой цели применяется переменная @MaxViewsPerQuery. Поскольку известно, что переменная, которой присвоено значение из столбца результирующего набора с несколькими строками, сохраняет

значение, полученное из последней строки, значение переменной @strLastView можно присвоить в том же операторе SELECT, который использовался для динамического формирования строки запроса, чтобы значение этой переменной можно было применить в следующих итерациях цикла для возобновления работы с того места, где она была прервана в предыдущей итерации.

Приведенный выше оператор INSERT замещает оператор SELECT, обычно используемый в правиле согласования. Автор предусмотрел тем самым вставку результатов применения динамически определяемых критериев запроса в таблицу для упрощения проверки кода. Очевидно, что желательно было бы также предусмотреть вызов функции формирования извещений для выработки неотформатированных извещений с помощью рассматриваемых представлений с динамически определяемыми критериями выборки, если бы цель состояла в том, чтобы применить эти представления в приложении, которое описано в данном разделе. Например, можно было бы изменить используемый запрос с правилом согласования, чтобы он выглядел примерно таким образом:

```
EXEC('SELECT dbo.BNSInfoNotify(b.SubscriberId,
     s.DeviceName,
     s.SubscriberLocale,
     b.ID,
     b.Product,
     b.OpenedBy,
     b.AssignedTo,
     b.Description,
     b.DateChanged,
     b.Pri,
     b.Sev)
FROM BNSSubscriptions e JOIN ('+@strQry+') b ON
    (e.SubscriberId=b.SubscriberId)')
```

Затем можно было бы вызывать на выполнение этот запрос в каждой итерации цикла WHILE, приведенного в листинге 19.6. Это позволило бы обеспечить выработку извещений с учетом критериев, заданных в представлениях.

В настоящем подразделе описан всего лишь прототип, который показывает, как можно реализовать подход к использованию динамически определяемых правил согласования в службах Notification Services. Проверка, проведенная автором, показала, что создаваемое приложение оказывается весьма быстродействующим и производит впечатление полностью функционально пригодного. Для проведения экспериментов с помощью описанного способа читатель может воспользоваться сценариями create_views.sql и dynamic_filter.sql, которые находятся в подкаталоге CH19\bns\svc компакт-диска, прилагаемого к данной книге.

Безусловно, применение подхода, основанного на механизме такого типа, вряд ли оправдано, если основополагающие данные состоят всего лишь из нескольких столбцов, как в рассматриваемой таблице Bugs. Но представьте себе ситуацию, в которой основополагающие данные включают сотни столбцов. Использование приема с подстановочными символами для исключения столбцов из критериев выборки данных о подписке может оказаться неоправданным с точки зрения производительности. К тому же, указанный прием может стать слишком ограничительным с точки зрения используемых критериев поиска, поскольку

пользователи могут пожелать, чтобы им предоставлялись более широкие возможности по сравнению с обычным заданием простых критериев поиска для фиксированного множества полей. В подобной ситуации механизм динамического определения правил согласования, подобный описанному в настоящем подразделе, был бы не только более эффективным по сравнению с подходом, основанным на использовании подстановочных символов, но и гораздо более гибким. Разработчик мог бы даже создать вспомогательную таблицу, позволяющую пользователям задавать имена (или описания) для своих представлений с критериями поиска, чтобы можно было в дальнейшем использовать такие имена для ссылки на нужные представления. Иными словами, пользователям может быть предоставлена возможность создавать именованные, повторно применимые запросы, оформленные как объекты представлений и способные вырабатывать извещения на основе модифицированной версии стандартного правила согласования служб Notification Services, как показано в настоящем подразделе.

Резюме

Службы Notification Services предоставляют пользователю гибкую, расширяемую и масштабируемую платформу для создания приложений рассылки извещений. Эти службы включают в себя большинство вспомогательных средств, необходимых для создания приложений такого типа; пользователю остается только вводить по мере необходимости специализированный код для реализации требуемых ему функциональных возможностей.

Любое приложение Notification Services состоит из трех основных компонентов: служба Windows, хранилище данных SQL Server и приложение управления подпиской. Службы Notification Services предоставляют инструментальные средства для автоматического создания хранилища данных SQL Server и службы Windows, а также позволяют воспользоваться примерами приложений, которые можно копировать и приспосабливать к своим нуждам для создания собственных приложений управления подпиской.

В службе Windows, `NSService.exe`, выполняются три основные задачи для выработки извещений, передаваемых подписчикам. Процесс сбора данных о событиях обеспечивает сбор информации о событиях, представляющих интерес для приложения. Генератор обеспечивает согласование этой информации о событиях с определениями подписок и выработку неотформатированных извещений. Распределительный сервер обеспечивает преобразование таких неотформатированных извещений (которые, по существу, представляют собой просто сообщения) в полноценные извещения и передачу таких извещений подписчикам по каналам доставки.

В основе служб Notification Services лежит развитая инфраструктура объектов, с помощью которой могут создаваться приложения рассылки извещений. В службах рассылки извещений предусмотрены интерфейсы для реализации определяемых пользователем средств доступа к данным о событиях, каналов доставки и средств форматирования информационного наполнения. Предусмотрены объекты для управления подписчиками, устройствами подписчика, подписками и прочими элементами

Notification Services. Кроме указанной инфраструктуры, файлов конфигурации XML и таблиц стилей, поддерживаемых программным продуктом Notification Services, предоставляется полностью готовое программное решение, которое может быть легко модифицировано в соответствии с требованиями пользователя.

Вопросы для самопроверки

1. Какой компонент в приложении службы рассылки извещений обеспечивает преобразование неотформатированных извещений в уведомительные сообщения, применимые для отправки подписчикам?
2. Чему равно максимальное количество процессов генератора, поддержку которых способен обеспечить каждый экземпляр приложения Notification Services?
3. Подтвердите или опровергните следующее утверждение. Служба Windows приложения Notification Services реализована в виде исполняемого файла с управляемым кодом, поэтому определяемые пользователем средства сбора информации о событиях обязательно должны быть реализованы в виде программных сборок с управляемым кодом, функционирующих в контексте процесса этого исполняемого файла.
4. Назовите имя исполняемого файла, который используется в приложении Notification Services в качестве службы Windows.
5. Подтвердите или опровергните следующее утверждение. В отличие от файлов конфигурации экземпляра, файлы определения приложения не позволяют непосредственно получить доступ к переменным среды для использования значений этих переменных в качестве параметров.
6. Какая задача обычно стоит на первом месте в цикле разработки приложения Notification Services: регистрация службы Windows или создание баз данных экземпляра и приложения?
7. Опишите назначение функции `Enable` утилиты NSControl.
8. Назовите единственное средство форматирования информационного наполнения, которые входит в поставку программного продукта Notification Services.
9. Какие действия осуществляются незаметно для пользователя после того, как из соответствующего правила вызывается функция формирования извещения?
10. Почему рекомендуется по возможности реализовывать приложение управления подпиской как приложение Web, а не как приложение Windows?
11. Подтвердите или опровергните следующее утверждение. После регистрации экземпляра файл конфигурации экземпляра непосредственно не используется службой Windows этого экземпляра.
12. Подтвердите или опровергните следующее утверждение. Для регистрации нового экземпляра приложения службы рассылки извещений используется программа `regsvr32.exe`.

13. В каком файле конфигурации XML (в файле конфигурации экземпляра или файле определения приложения) перечислены каналы доставки, доступные для использования в приложении?
14. Существует ли возможность эксплуатировать исполняемый файл службы Windows как обычное терминальное приложение?
15. Опишите назначение хронологических таблиц.
16. Назовите процесс, применяемый в рамках приложения Notification Services для удаления устаревших данных о событиях и извещениях.
17. Назовите компоненты Notification Services трех типов, для которых может быть разработан определяемый пользователем код путем реализации заранее определенных интерфейсов.
18. На что влияет величина продолжительности кванта времени, указанная в файле определения приложения?
19. Опишите, каким образом многоадресатная доставка способствует повышению масштабируемости приложения.
20. Подтвердите или опровергните следующее утверждение. Приложения управления подпиской Notification Services должны разрабатываться на языке C#, поскольку в них должна учитываться зависимость от файла класса `NSUtility.cs`, который пока еще не был переведен на какой-либо другой язык, совместимый с CLR.

ГЛАВА 20

Службы Data Transformation Services

Службы преобразования данных (Data Transformation Services — DTS) представляют собой одну из наиболее выдающихся технологий, которые были реализованы в последние годы на основе программы SQL Server. В прошлом, чтобы воспользоваться функциональными возможностями, предоставляемыми службами DTS, которые теперь непосредственно входят в состав средств SQL Server, приходилось обращаться к программным продуктам сторонних поставщиков (и, возможно, нести весьма значительные дополнительные затраты). Опытные программисты и практики с большим стажем работы с программой SQL Server оценивают DTS как превосходно спроектированное и мощное дополнение к семейству средств SQL Server, обладающее удобным интерфейсом, который имеет широкие функциональные возможности и не требует от разработчика больших трудозатрат. Технология работы пользователя со средствами DTS является наглядной и хорошо продуманной; в интерфейсе DTS воплощено все, чем должны обладать приложения, предназначенные для разработчика, работающего в визуальной среде, — удобство использования, расширяемость и полная функциональная пригодность.

Средства DTS опираются на такие ключевые технологии Microsoft, как COM, ATL, OLE DB и, разумеется, SQL Server. Работа приложения DTS Designer основана на применении расширяемой объектной модели, благодаря которой появляется возможность дополнять состав средств этого приложения компонентами COM. Доступ к самой объектной модели DTS может быть получен с помощью средств COM Automation, поэтому пользователь имеет возможность управлять пакетами и преобразованиями DTS программным путем на любом языке, поддерживающем COM Automation (например, Visual Basic).

Нет ничего удивительного в том, что службы DTS позволяют успешно передавать данные из входного набора в выходной. А если учесть такую возможность и тот факт, что основополагающая архитектура данных этих служб основана на интерфейсе OLE DB, то становится вполне очевидно, что эти службы позволяют передавать данные от одного средства доступа OLE DB к другому. В качестве входного и выходного наборов могут служить плоские файлы, базы данных в архитектуре "клиент-сервер", базы данных на мэйнфреймах и т.д. Основное назначение служб DTS состоит в том, чтобы обеспечить перемещение данных из пункта А в пункт Б настолько просто, насколько это возможно, поэтому в составе этих служб предусмотрено бесчисленное множество возможностей, позволяющих решать любые задачи подобного типа.

Автор не собирается загромождать эту главу поэтапными инструкциями по созданию множества различных типов пакетов DTS или по передаче данных из одного средства доступа в другое. Программы Import/Export Wizard и DTS Designer имеют весьма наглядный интерфейс, чтобы можно было воспользоваться ими, не изучая перед этим целую книгу. Достаточно просто перенести некоторые объекты соединений в рабочее пространство DTS Designer, связать эти соединения с задачами преобразования и ограничениями предшествования, после чего приступить к работе. Создание несложного пакета выполняется настолько просто, что даже не требуется изучать оперативную документацию Books Online, чтобы узнать, как это сделать, не говоря уже об изучении какой-либо дополнительной литературы. Чтобы научиться работать с программой DTS Designer, лучше всего просто открыть эту программу и самостоятельно создать несколько пакетов. Превосходный способ сразу же освоить всю эту тематику состоит в том, чтобы вначале воспользоваться программой DTS Import/Export Wizard и составить с ее помощью какой-то пакет, а затем загрузить полученный пакет в программу DTS Designer.

Кроме того, автор не собирается заполнять страницы данной главы, показывая снимки с экрана программы DTS Designer один за другим. Сам автор однажды допустил ошибку, приобретя книгу по программированию для SQL Server только для того, чтобы узнать, что в этой книге для описания многих тем, включая DTS, используются в основном снимки с экрана. Зачастую в той книге на одной странице было помещено по два снимка с экрана и лишь одна или две строки пояснительного текста между ними. Но автор считает, что читателю не нужна книга, которая показывает, как выглядит интерфейс программы DTS Designer, поскольку каждый может сам запустить эту программу на выполнение из меню Enterprise Manager.

Поэтому в данной главе представлены основные элементы пакетов DTS, описано, из каких компонентов состоят службы DTS с точки зрения архитектуры программного обеспечения, рассматриваются некоторые элементы программирования и создания приложений DTS, достаточно важные, но относящиеся к этой тематике лишь косвенно, а затем приведен пример приложения DTS. Кроме того, читатель узнает о некоторых сильных и слабых сторонах служб DTS, и, как мы надеемся, сможет непредвзято оценить важность некоторых новаторских способов, позволяющих воспользоваться возможностями средств DTS.

Краткий обзор

С точки зрения архитектуры программного обеспечения службы DTS состоят из описанных ниже частей.

- **Средства доступа OLE DB.** (С формальной точки зрения не являются частью DTS, но представляют собой необходимый компонент, без которого невозможно эксплуатировать эти службы. В службах DTS с помощью средств доступа OLE DB осуществляется выборка данных и запись данных на внешние устройства.)
- **Средство многофазной перекачки данных.** Это средство используется для обеспечения сложных преобразований данных.

- **Средство проектирования с графическим интерфейсом.** Это средство основано на расширяемой объектной модели, которая базируется на технологии COM. Основными элементами этой модели являются пакеты и задачи. Компонент COM, предназначенный для использования в качестве пользовательской задачи DTS, должен реализовывать интерфейс `CustomTask`, который определен в объектной библиотеке `DTSPackage Object Library` компании Microsoft. Указанный компонент должен эксплуатироваться как внутрипроцессный, если он предназначен для использования в программе DTS Designer, а если он применяется в составе отдельного программного обеспечения, то может эксплуатироваться как внепроцессный.
- **Набор программируемых COM-объектов.** Такие объекты предоставляют средства создания, манипулирования и выполнения пакетов DTS. Кроме того, предусмотрены также объекты для реализации средств перекачки данных DTS и выполнения управляемых данными запросов, а также объект `Application`, предназначенный для управления параметрами настройки конфигурации DTS на уровне приложения.
- **Коллекция инструментальных средств.** В эту коллекцию входят такие программы, как DTS Import/Export Wizard, `dtsrun` и `dtsrunui`, предназначенные для создания и выполнения пакетов.
- **Программа DTS Query Designer.** Эта программа представляет собой графический генератор запросов на языке T-SQL.

Чтобы воспользоваться средствами DTS для передачи или преобразования данных, необходимо создать *пакет*, состоящий из соединений, этапов выполнения задачи и ограничений предшествования. Задачи DTS подразделяются на несколько типов, начиная от простых преобразований данных и заканчивая действиями, основанными на использовании сложных сценариев ActiveX и T-SQL. А поскольку из сценария ActiveX можно получить доступ к технологии COM, то любой пакет DTS позволяет выполнить весьма широкий перечень задач.

Пакеты

Фундаментальным элементом приложения DTS является пакет. Для создания пакета может применяться один из двух способов, которые основаны на использовании программы DTS Import/Export Wizard или DTS Designer. Пакет можно сохранить в базе данных SQL Server; в репозитарии метаданных Metadata Repository в виде кода на языке Visual Basic (запись пакета в репозитарий – необратимая операция, поскольку пакеты, хранящиеся в виде кода VB, нельзя загружать в программу DTS Designer); а также в структурированном файле хранения COM.

ПРИМЕЧАНИЕ. Ко времени написания этой главы, чтобы получить возможность сохранять пакеты в репозитарии, необходимо было щелкнуть правой кнопкой мыши на узле Data Transformation Services в программе Enterprise Manager, выбрать опцию Properties и отметить флажок Enable Save To Meta Data Services. Если на указанном флажке нет отметки, то в диалоговом окне сохранения пакета отсутствует опция Meta Data Services.

Если выбран вариант, предусматривающий сохранение пакета в базе данных SQL Server, то в программе DTS Designer вызывается недокументированная хранимая процедура `sp_add_dtspackage` для записи пакета в базу данных `msdb`. Пакет передается в качестве данных типа `image` в эту процедуру и записывается в таблицу `sysdtspackages`. Со списком пакетов в таблице `sysdtspackages` можно ознакомиться, перейдя к узлу **Local Packages**, находящемуся под узлом **Data Transformation Services** в программе Enterprise Manager.

Структурированные файлы хранения COM по своей организации во многом напоминают файловую систему. Они предоставляют возможность сохранения COM-объектов на диске. Благодаря тому, что пакет DTS может быть записан на диске в виде структурированного файла хранения COM, появляется возможность легко передавать пакет другому пользователю с помощью таких носителей данных, как электронная почта и компакт-диск.

Пакет DTS состоит из объектов `Connection` (ссылок на средства доступа OLE DB), задач, преобразований, этапов и ограничений предшествования. Ограничения предшествования управляют потоком данных пакета, т.е указывают, как осуществляется перемещение данных, обрабатываемых в пакете. (Как вскоре будет описано, управление потоком данных пакета можно также осуществлять с помощью сценариев ActiveX.) Преобразования указывают, как данные должны изменяться, форматироваться и передаваться из входного набора в выходной. Объекты `Connection` обозначают начальные и конечные точки процесса преобразования – отправителя и получателя данных (преобразуемых и передаваемых из одного места в другое).

Соединения

Соединения, поддерживаемые средствами DTS, состоят из источников данных OLE DB. Поскольку имеются средства доступа OLE DB, которые поддерживают источники данных ODBC, средства DTS позволяют также использовать источники данных ODBC. Ниже перечислены некоторые примеры того, к данным какого типа можно получить доступ в службах DTS с помощью средств доступа OLE DB.

- Базы данных SQL Server (доступ осуществляется с помощью SQLOLEDB – встроенного средства доступа OLE DB программы SQL Server).
- Другие серверы реляционных СУБД, такие как Oracle, Sybase и DB2.
- Нереляционные хранилища данных, такие как Active Directory, Indexing Service, Site Server и Exchange Server.
- Текстовые файлы в формате с разграничителями и в фиксированном формате.
- Код HTML.
- Файлы Access, Excel и Visual FoxPro.
- Файлы dBase и Paradox.
- Другие серверы баз данных, доступные с помощью интерфейса ODBC.

Задачи

Задача – это неразрывная единица работы. Каждая задача в пакете DTS определяет часть процесса преобразования данных и выполняется в виде одного этапа. В поставку программного обеспечения средств DTS входит целый ряд объектов задач, которые могут использоваться для создания сложных приложений преобразования данных. Ниже приведены примеры того, какие действия могут выполняться с помощью объектов задач.

- Копирование данных из одного средства доступа OLE DB в другое. Операции копирования данных позволяют загружать данные из источника данных одного типа в выходной набор другого типа. Например, можно выполнить копирование данных из базы данных Oracle в базу данных SQL Server или наоборот. Можно также передавать не только реляционные, но и нереляционные данные и использовать функциональные средства массовой загрузки `Bulk Load` служб DTS для обеспечения загрузки данных из текстовых файлов в базы данных SQL Server с высокой скоростью.
- Сложные преобразования данных. С помощью задач можно отображать столбцы из входного набора данных в выходной набор данных и указывать, какие преобразования должны применяться к данным в процессе копирования. Для выполнения манипуляций с данными в ходе их передачи можно использовать сценарии ActiveX, а также определять между данными входного и выходного наборов связи "один ко многим", "многие к одному" и другие необычные связи.
- Вложенные пакеты. С помощью задачи Execute Package можно настроить один пакет на осуществление вызова другого пакета, установить значения глобальных переменных пакета и т.д.
- Передача сообщений. Службы DTS предоставляют средства для передачи сообщений электронной почты с учетом состояния завершения этапов пакета и для организации взаимодействия с очередью сообщений Message Queue в целях обеспечения обмена сообщениями между пакетами.
- Выполнение сценариев T-SQL и ActiveX. Выбранные пользователем сценарии T-SQL и ActiveX можно выполнять как этапы пакета. В выполняемых сценариях T-SQL могут применяться вызовы произвольных запросов, а также вызовы хранимых процедур, а для создания запросов может использоваться программа DTS Query Designer. Для формирования сценариев ActiveX может служить любой язык сценариев ActiveX. По умолчанию в любой инсталляции SQL Server доступны языки VBScript и JScript.
- Дублирование объектов базы данных. Из одной базы данных SQL Server в другую можно передавать таблицы, представления, хранимые процедуры, определяемые пользователем функции, заданные по умолчанию значения, правила и определяемые пользователем типы данных. Службы DTS позволяют даже в случае необходимости автоматически запротоколировать все выполняемые пользователем операции в виде ряда сценариев T-SQL и файлов данных BCP.

Средство многофазной перекачки данных

Основная часть реально действующих функциональных средств DTS реализована в виде средства многофазной перекачки данных. Это средство представляет собой машину, лежащую в основе задач Transform Data, Data Driven Query и Parallel Data Pump. Вполне очевидно, что основное назначение средства перекачки состоит в перемещении и преобразовании данных из входного набора в выходной.

При передаче и преобразовании данных из одного источника в другой с помощью средства перекачки данных выполняются шесть основных этапов, называемых также *фазами*. Доступ к этим фазам может быть предоставлен с помощью событий, а в обработчики событий можно включить код сценариев, позволяющих реализовать в процессе перекачки данных функции, определяемые пользователем. Следует отметить, что по умолчанию пользователь может ввести дополнительный код только в одной фазе перекачки данных — фазе преобразования строки. Для того чтобы предоставить доступ в коде пакета и сценариев другим фазам, необходимо разрешить отображение многофазного средства перекачки в программе DTS Designer, щелкнув правой кнопкой мыши на узле Data Transformation Services в программе Enterprise Manager и отметив опцию Show multi-phase pump in DTS Designer. После того как в программе DTS Designer будет предоставлен полный доступ к многофазному процессу перекачки, появляется возможность подключать код сценариев ActiveX к событиям отдельных фаз для введения определяемых пользователем функций в процесс перекачки. Как уже было сказано, при преобразовании данных с помощью средства перекачки данных выполняются шесть основных фаз, которые описаны ниже.

1. Фаза, предшествующая чтению входного набора. Эта фаза выполняется перед тем, как фактически начинается чтение строк из входного набора данных. Именно в это время следует выполнять такие операции, как создание строк заголовков, или осуществлять другую подготовительную работу, предшествующую началу процесса преобразования строк.
2. Фаза преобразования строк. Это — фаза перекачки данных, предусмотренная по умолчанию; данная фаза представляет собой этап, на котором каждая строка считывается из входного набора данных и в качестве дополнительной возможности подвергается преобразованию.
3. Фаза, следующая за фазой преобразования строк. Данный этап наступает по завершении фазы преобразования строк и сам состоит из трех подэтапов, которые описаны ниже.
   - 3.1. Этап On Transform Failure. Этот этап выполняется, если во время преобразования строк возникает ошибка. Следует отметить, что любые ошибки, вырабатываемые на этом этапе, не учитываются при подсчете максимально допустимого количества ошибок, заданного на странице Options для текущего преобразования. Чтобы обнаруживать ошибки на рассматриваемом этапе и реагировать на них соответствующим образом, можно использовать код сценария. Затем с учетом того, что произошло на данном этапе, можно выполнить один из описанных ниже двух подэтапов.

3.2. Этап On Insert Success. Этот этап наступает после успешной вставки строки в выходной набор строк. (Следует помнить, что фактически в выходной набор в данный момент не записываются какие-либо данные, поскольку операции записи применяются к выходному набору строк, т.е. к кэшу, содержимое которое будет записано в выходной набор лишь в дальнейшем.)

3.3. Этап On Insert Failure. Этот этап выполняется, если вставка преобразованной строки в выходной набор строк завершается неудачей. Такая ошибка не учитывается при подсчете максимально допустимого количества ошибок, заданного на странице Options для текущего преобразования.

4. Фаза завершения пакетного задания. Эта фаза наступает после того, как количество строк, вставленных в выходной набор строк, становится равным размеру пакетного задания, указанному на странице Options для текущего преобразования. Кроме того, эта фаза наступает после того, как обработаны все строки во входном наборе данных, а в выходном наборе строк имеется по меньшей мере одна строка. После наступления данной фазы средство перекачки записывает строки из выходного набора строк в таблицу выходного набора. В зависимости от того, была ли выполнена настройка конфигурации пакета таким образом, чтобы в нем использовались транзакции, появление ошибки при выполнении этой фазы может привести к тому, что в выходной набор будут записаны только некоторые строки. Эта фаза выполняется после завершения любого пакетного задания, независимо от того, была ли успешно осуществлена запись данных этого задания в выходной набор. Следует отметить, что при определенных обстоятельствах данная фаза может быть не выполнена, например, если завершилась неудачей вставка всех входных строк в выходной набор строк. При таком развитии ситуации отсутствует пакетное задание, данные которого можно было бы записать в выходной набор, поскольку выходной набор строк пуст.

5. Фаза, следующая за фазой чтения из входного набора. Эта фаза подытоживает фазу, предшествующую чтению из входного набора, и позволяет пользователю ввести в действие код сценария, выполняющий определенные задачи после преобразования всех строк. В это время все еще предоставляется доступ к данным выходного набора, поэтому в данной фазе существует возможность, например, записывать итоговые строки или выполнять какие-то другие операции с данными выходного набора.

6. Фаза завершения перекачки данных. Эта фаза наступает после завершения всей обработки, непосредственно перед тем, как прекращается работа средства перекачки. В это время пользователь больше не имеет доступа ни к данным входного набора, ни к данным выходного набора, но все еще имеет в своем распоряжении полный набор возможностей среды поддержки сценариев ActiveX, чтобы можно было выполнять такие операции, как вывод записей контрольного или текущего журналов, взаимодействие с файловой системой и т.д.

Подключение кода сценариев к другим фазам процесса преобразования данных с помощью средств перекачки может быть выполнено точно так же, как при обычном

подключении кода сценариев к фазе преобразования строк. Для этого достаточно щелкнуть на вкладке Transformations, затем выбрать в поле со списком Phases filter ту фазу, к которой нужно подключить код, и щелкнуть кнопкой New, чтобы приступить к подключению кода. Затем необходимо выбрать опцию ActiveX Script в диалоговом окне Create New Transformation и щелкнуть на кнопке Properties в диалоговом окне Transformation Options, чтобы ввести код сценария. После этого необходимо выполнить соответствующим образом настройку конфигурации столбцов входного набора и выходного набора и, наконец, сохранить это преобразование. После вызова пакета на выполнение код, подключенный к указанной фазе перекачки данных, будет выполняться в соответствии с предусмотренными требованиями.

Чтобы провести некоторые эксперименты с использованием описанных выше возможностей, загрузите пакет `MultiphaseDataPumpExample.DTS` из каталога `CH20` компакт-диска, прилагаемого к данной книге, в программу DTS Designer. Это – несложный пакет, который просто копирует таблицу из базы данных `pubs` в базу данных `Northwind`. Автор закрепил код сценария ActiveX за событиями каждой фазы перекачки, чтобы можно было продемонстрировать, как действуют обработчики этих событий. Выполните должным образом настройку свойств применяемых вами соединений, затем вызовите пакет на выполнение. Вы должны обнаружить, что в связи с возникновением каждого события появляется диалоговое окно. Обратите внимание на то, что некоторые события происходят чаще, чем другие, например, вы должны обнаружить около двух десятков событий `Insert Success`, по одному на каждую операцию копирования строки из входного набора в выходной набор.

ПРИМЕЧАНИЕ. Во всех примерах пакетов, приведенных в данной главе, при указании на СУБД SQL Server используется имя "`(local)`". Если сервер, на котором проверяются эти пакеты, представляет собой именованный экземпляр, то перед вызовом примеров пакетов на выполнение можно создать псевдоним конфигурации клиента "`(local)`", чтобы исключить необходимость вносить изменения в заданные примеры.

В диалоговых окнах с определениями опций, относящихся к каждому из преобразований ActiveX, имеется вкладка Phases, которая позволяет определить, с какой фазой (фазами) должен быть связан код ActiveX. По умолчанию такая фаза совпадает с фазой, выбранной в поле со списком Phase filter при первоначальном создании преобразования. Обратите внимание на то, что последние два элемента в указанных диалоговых окнах расположены в порядке, обратном последовательности их выполнения, поскольку событие завершения перекачки фактически происходит после события, наступающего вслед за завершением обработки входного набора.

Задача Bulk Insert

Еще одним важным механизмом DTS, относящимся к загрузке данных, кроме средства многофазной перекачки данных, является задача Bulk Insert. Задача Bulk Insert фактически предоставляет доступ с помощью графического интерфейса к функциональным возможностям команды `BULK INSERT` языка T-SQL, причем вызов этой команды осуществляется в самой задаче для загрузки данных на сервер.

Из того факта, что механизмом, действительно применяемым для загрузки данных на сервер, является команда BULK INSERT, следует целый ряд важных соображений. Во-первых, путь к входному файлу должен быть задан относительно целевого сервера SQL Server. Поэтому, если задача Bulk Insert используется для копирования файла с клиентского компьютера на удаленный сервер SQL Server, необходимо указать путь к файлу в формате UNC, а учетная запись, в которой работает программа SQL Server, должна иметь доступ к этому файлу (например, для копирования файла с клиентского компьютера на сервер не подходит учетная запись LocalSystem, поскольку LocalSystem не имеет сетевых прав). Во-вторых, производительность загрузки становится намного выше, если загружаемый файл находится на том же компьютере, на котором работает экземпляр SQL Server. По существу, команда BULK INSERT языка T-SQL реализована в виде COM-объекта, который функционирует внутри программы SQL Server. Если загружаемый файл находится на том же компьютере, на котором работает экземпляр SQL Server, то COM-объект просто открывает и читает загружаемый файл как локальный файл. Если же файл находится на другом компьютере в сети, то может оказаться, что ограничения, связанные с недостаточной пропускной способностью сети, снижают скорость массовой загрузки точно так же, как при использовании любого другого средства массовой загрузки, такого как программа bcp.exe или сама команда BULK INSERT.

Следует отметить, что задача Bulk Insert не может применяться для загрузки данных непосредственно из одной таблицы базы данных SQL Server в другую. Входным набором для задачи Bulk Insert должен быть файл операционной системы. Но несмотря на сказанное выше, можно обеспечить высокоскоростную обработку данных, подлежащих массовому копированию с помощью обычных задач Transform Data, используя опцию задачи Use fast load (которая разрешена по умолчанию).

Кроме того, следует отметить, что задачи Bulk Insert не поддерживают преобразования с помощью сценариев ActiveX, но такие преобразования поддерживаются более универсальной задачей Transform Data. В задаче Bulk Insert вызывается команда BULK INSERT языка T-SQL, поэтому функциональные возможности этой задачи более ограничены по сравнению с задачей Transform Data.

Для проведения экспериментов с задачами Bulk Insert можно загрузить пример пакета BulkInsertExample.DTS из каталога CH20 компакт-диска, прилагаемого к данной книге. Автор рекомендует запустить трассировку Profiler до начала выполнения пакета, чтобы можно было видеть код T-SQL, передаваемый на сервер.

Задача Data Driven Query

Если бы мир был проще, то для передачи данных из одного источника данных в другой достаточно было бы использовать элементарные команды языка DML. Например, язык DML вполне применим, если преобразования данных состоят в основном из операций взаимно-однозначного копирования столбцов, а необходимость в использовании каких-либо сложных логических алгоритмов для определения того, как и когда данные должны быть скопированы из одного места в другое, возникает редко. Но в реальном мире в основе способов, применяемых

для перемещения данных из одного места в другое, часто лежат сложные логические алгоритмы, а для выполнения работы нередко приходится использовать хранимые процедуры и определяемые пользователем запросы. Для осуществления действий в ситуациях подобного типа можно применить задачу Data Driven Query. Например, задача Data Driven Query позволяет значительно упростить работу, если требуется осуществить преобразование данных, для которого необходим более широкий перечень функциональных возможностей по сравнению с простой вставкой строк в целевую таблицу. Тем не менее для выполнения не столь сложных операций загрузки данных задачи Transform Data и Bulk Insert являются более предпочтительными, чем задача Data Driven Query, поскольку указанные задачи в значительной степени оптимизированы для выполнения операций вставки. Поэтому задачу Data Driven Query следует использовать вместо одной из указанных задач, если требования к выполняемому преобразованию данных превосходят возможности этих задач.

Задача Data Driven Query организована по принципу предоставления пользователю возможности задавать альтернативные запросы вставки, обновления и удаления, предназначенные для выполнения применительно к каждой строке из входного набора данных. В качестве таких запросов могут применяться простые запросы SQL, вызовы хранимых процедур или сложные пакеты заданий на языке SQL. Каждый запрос может иметь подставляемые параметры, которым в задаче Data Driven Query могут присваиваться значения из входного набора данных после чтения каждой строки набора.

В ходе подготовки задачи Data Driven Query к выполнению осуществляется настройка конфигурации таблицы связывания. Важно понять назначение таблицы связывания и способ ее использования в указанной задаче. По существу, *таблица связывания* представляет собой просто заготовку — механизм для определения выходного набора строк, в который должны быть помещены данные из входного набора строк. Фактические пункты назначения передаваемых данных определяют подготавливаемые пользователем запросы. В запросах может быть указана таблица связывания или совсем другая таблица (таблицы).

Следует также учитывать, что и эти запросы представляют собой просто заготовки, поэтому из запроса на вставку может быть выполнена команда UPDATE, из запроса на обновление — команда DELETE и т.д. Пример типичного запроса на обновление приведен в листинге 20.1.

Листинг 20.1. Пример типичного запроса на обновление

```
UPDATE authors_new
SET au_lname=?,
au_fname =?,
phone=?,
address=?,
city=?,
state=?,
zip=?,
contract=1
WHERE au_id=?
```

Запрос, приведенный в листинге 20.1, применяется для обновления столбцов в копии таблицы pubs..authors с заменой значениями из входного набора данных во всех столбцах, кроме столбца contract, которому при выполнении операций обновления принудительно присваивается значение 1.

После определения конфигурации запросов на вставку, обновление и удаление задается сценарий ActiveX, который указывает, какой из этих запросов должен вызываться в процессе обработки данных. Значение, возвращаемое из сценария, позволяет определить, какой именно запрос был выполнен. В сценарии можно выполнять проверку на наличие (отсутствие) строк, использовать оператор switch для возврата другого результата сценария с учетом значения того или иного столбца во входном наборе данных, осуществлять переходы с учетом значений глобальных переменных, а также использовать другие программные конструкции, поэтому сценарий, по существу, предоставляет пользователю полный контроль над тем, какой из запросов должен быть выполнен. В листинге 20.2 показано, как может выглядеть сценарий ActiveX для задачи Data Driven Query.

Листинг 20.2. Пример сценария ActiveX для задачи Data Driven Query

```
Function Main()

  DTSDestination("au_id") = DTSSource("au_id")
  DTSDestination("au_lname") = DTSSource("au_lname")
  DTSDestination("au_fname") = DTSSource("au_fname")
  DTSDestination("phone") = DTSSource("phone")
  DTSDestination("address") = DTSSource("address")
  DTSDestination("city") = DTSSource("city")
  DTSDestination("state") = DTSSource("state")
  DTSDestination("zip") = DTSSource("zip")
  DTSDestination("contract") = DTSSource("contract")

  Select Case Trim(DTSSource("state"))
  Case "CA"
     Main = DTSTransformstat_InsertQuery
  Case "OR"
     Main = DTSTransformstat_UpdateQuery
  Case "KS"
     Main = DTSTransformstat_DeleteQuery
  Case Else
    Main = DTSTransformstat_SkipRow
  End Select
End Function
```

В сценарии, приведенном в листинге 20.2, переход осуществляется с учетом значения столбца state во входном наборе данных. Применительно к заказчикам из штата Калифорния ("CA") всегда выполняется запрос на вставку. Применительно к заказчикам из штата Орегон ("OR") всегда выполняется запрос на обновление. Применительно к заказчикам из штата Канзас ("KS") всегда выполняется запрос на удаление. А применительно к заказчикам из всех остальных штатов никакие действия не выполняются — строка из входного набора просто пропускается. Как уже было сказано, заготовки запросов на вставку, обновление, удаление и выборку являются именно таковыми — заготовками. Фактически

выполняемые запросы могут осуществлять любые действия, которые требуются пользователю, например, во всех запросах могут осуществляться операции вставки в выходной набор строк, хотя и разными способами. Четыре варианта выбора типа запроса, предоставляемые в задаче Data Driven Query, можно использовать для разбиения запросов, необходимых для выполнения предстоящей работы, на четыре группы. Кроме того, безусловно, вызываемый код SQL или вызываемая хранимая процедура позволяют дополнительно управлять ходом выполнения преобразования данных с учетом значений, полученных из входного набора данных.

Чтобы ближе ознакомиться с задачей Data Driven Query, загрузите пакет `DataDrivenQueryExample.DTS` из каталога `CH20` компакт-диска, прилагаемого к данной книге. И в этом случае сбор данных трассировки Profiler в ходе эксплуатации пакета может стать удобным способом изучения того, что происходит на сервере незаметно для пользователя в процессе преобразования каждой строки из входного набора данных.

Преобразования ActiveX

Выше в данной главе было сказано, что службы DTS предоставляют несколько механизмов, позволяющих использовать код сценариев ActiveX для расширения функциональных возможностей пакета. Один из способов расширения функциональных возможностей любого пакета DTS с помощью кода ActiveX состоит в применении преобразований ActiveX. Преобразования ActiveX предоставляют пользователю полный контроль над отображением столбцов входного набора на столбцы выходного набора в процессе преобразования данных. Благодаря этому появляется возможность соединять несколько столбцов входного набора в один столбец выходного набора и разбивать один столбец входного набора на несколько столбцов выходного. Кроме того, значения столбцам выходного набора можно присваивать вообще без использования столбцов входного набора (при этом обычно входные данные берутся из глобальных переменных или результатов поиска); преобразование столбца входного набора может осуществляться без использования столбца выходного набора (в таком случае назначением для преобразования обычно становится глобальная переменная); кроме того, могут выполняться преобразования, в которых не участвуют ни столбцы входного набора, ни столбцы выходного набора (например, для проведения манипуляций с глобальными переменными или внешними объектами с помощью средств FileSystemObject или ADO).

Описанное выше можно лучше всего проиллюстрировать с помощью примера. В сценарии ActiveX, приведенном в листинге 20.3, столбцы `address`, `city`, `state` и `zip` таблицы `pubs..authors` объединяются в один столбец выходного набора.

Листинг 20.3. Пример сценария ActiveX

```
Function Main()
  DTSDestination("address") = DTSSource("address") & " " _
    & DTSSource("city") _
    & ", "+DTSSource("state") & " " &DTSSource("zip")
  Main = DTSTransformStat_OK
End Function
```

Обратите внимание на то, как в листинге 20.3 используются встроенные объекты DTSSource и DTSDestination. Поскольку сценарий преобразования ActiveX выполняется для каждой строки во входном наборе данных, объект DTSSource всегда указывает на ту строку, которая является текущей строкой во входном наборе данных, а объект DTSDestination — на текущую строку в выходном наборе данных. Исходный код рассматриваемого пакета можно найти в файле ActiveXTransformationExample.DTS каталога CH20 компакт-диска, прилагаемого к данной книге.

Преобразования других типов

В начале данной главы было указано, что в ней не будут рассматриваться каждое отдельное средство DTS и каждый нюанс использования служб DTS. Дело в том, что эту информацию можно легко усвоить самостоятельно, прочитав оперативную документацию Books Online и сформировав несколько пакетов. Но несмотря на сказанное выше, в этой книге заслуживают упоминания некоторые средства, более сложные по сравнению с другими. Одним из них является преобразование WriteFile. Преобразование с помощью задачи WriteFile позволяет взять два столбца входного набора (в одном из них должны быть указаны имена некоторых файлов, а в другом приведено содержимое этих файлов) и преобразовать эти столбцы во внешние текстовые файлы, соответствующие каждой строке исходной таблицы. Текстовый файл может быть записан в коде ANSI, UNICODE или в формате, определяемом пользователем. Для ознакомления с тем, как использовать задачу WriteFile для преобразования таблицы pubs..pub_info в ряд текстовых файлов, можно загрузить пакет WriteFileExample.DTS из каталога CH20 компакт-диска, прилагаемого к данной книге.

Аналогично задача Read File может применяться для загрузки содержимого ряда текстовых файлов с диска в столбец выходного набора. В данном случае имеется только один столбец входного набора и один столбец выходного. Столбец входного набора задает имя загружаемого файла, а столбец выходного набора определяет целевой столбец для содержимого файла. Способ использования задачи Read File для загрузки текстовых файлов, созданных с помощью пакета WriteFileExample.DTS, в копию таблицы pubs..pub_info показан в пакете ReadFileExample.DTS, который находится в каталоге CH20 компакт-диска, прилагаемого к данной книге.

В обоих примерах для указания имен входных или выходных файлов применяется столбец pub_id таблицы pub_info. Поскольку первичным ключом для данной таблицы является pub_id, это гарантирует, что при использовании указанных пакетов не будут возникать коллизии имен файлов в файловой системе.

Поисковые запросы

Хотя автор не рекомендует широко использовать поисковые запросы, он считает себя обязанным описать эти запросы и объяснить принципы их работы. По существу, *поисковый запрос* — это параметризованный подзапрос, который можно

подготавливать для применения при поиске значений данных. (Но фактически поисковый запрос позволяет осуществлять другие действия, кроме поиска значений данных, и выполнять другие операции применительно к каждому значению во входном наборе данных.) По своему принципу использование поискового запроса аналогично открытию курсора на таблице в сценарии T-SQL и вызову хранимой процедуры или выполнению подзапроса для каждой строки в курсоре. Поисковый запрос обычно выполняется при преобразовании некоторого типа. Поисковый запрос может представлять собой вызов хранимой процедуры или простой запрос SQL. Типичный поисковый запрос может выглядеть следующим образом:

```
SELECT      phone
FROM        authors
WHERE       (au_id = ?)
```

Подготовка поисковых запросов осуществляется с помощью вкладки Lookups используемой задачи преобразования. Вкладка Lookups появляется в диалоговом окне настройки конфигурации задачи, если задача поддерживает поисковые запросы. Каждый поисковый запрос имеет имя, соединение с входным набором данных, параметры настройки кэша и связанный с ним запрос. Параметры настройки кэша позволяют задавать в конфигурации количество значений, возвращаемых поисковым запросом, которые сохраняются в кэше для повторного использования. Возможность записи данных в кэш особенно удобна, если выполняется преобразование относительно большого количества строк, а количество строк в поисковой таблице относительно мало.

Ссылка на поисковые запросы осуществляется с использованием глобальной функции DTSLookups. Типичная ссылка на поисковый запрос в преобразовании ActiveX может выглядеть примерно таким образом:

```
DTSDestination("phone") =
  DTSLookups("phone").Execute(DTSSource("au_id"))
```

В приведенном выше коде функции DTSLookups передается имя требуемого поискового запроса, затем вызывается метод Execute этого поискового объекта и в этот метод передается параметр, требуемый для параметризованного запроса поискового объекта.

Следует учитывать, что не исключена вероятность возврата поисковым запросом нулевого количества строк. Если это происходит, то метод Execute возвращает пустое значение типа Variant. Для проверки такой ситуации в сценарии можно использовать функцию IsEmpty языка VBScript, как показано в листинге 20.4.

Листинг 20.4. Пример использования в сценарии функции IsEmpty языка VBScript

```
Dim Phone
Phone = DTSLookups("phone").Execute(DTSSource("au_id"))
If IsEmpty(Phone) Then
  DTSDestination("phone")="None"
Else
  DTSDestination("phone")=Phone
End If
```

Не исключена также возможность, что поисковый запрос возвратит несколько строк. Доступ предоставляется только к первой, возвращаемой строке, но пользователь может включить конструкцию ORDER BY в поисковый запрос для обеспечения того, чтобы первая возвращаемая строка была именно той, которая ему требуется. Кроме того, обнаружить ситуацию, в которой возвращено несколько строк, можно с помощью проверки свойства LastRowCount поискового объекта, возвращенного функцией DTSLookups, как показано в листинге 20.5.

Листинг 20.5. Пример использования свойства LastRowCount поискового объекта

```
Dim c
c=DTSLookups("phone").LastRowCount
If  c > 1 Then
  MsgBox "Warning: " & c & " lookup matches found"
End If
```

С поисковыми запросами можно дополнительно ознакомиться, загрузив в программу DTS Designer пакет LookupQueryExample.DTS из каталога CH20 компакт-диска, прилагаемого к данной книге. Дважды щелкните на задаче Transform Data и ознакомьтесь с вкладкой Lookups открывшегося окна. В данном примере список авторов, перечисленных в таблице pubs..titleauthor, копируется в новую таблицу базы данных Northwind. При этом берется столбец au_id из таблицы titleauthor и передается в ряд поисковых запросов для выборки различных столбцов из таблицы authors. В выходной набор фактически не копируется ни один столбец из таблицы titleauthor, кроме столбца au_id.

Свойства потока данных

Для управления преобразованиями в каждой части потока данных пакета может осуществляться корректировка свойств этого потока данных. В данном разделе не будут рассматриваться все свойства, но некоторые из свойств потока данных являются очень удобными, поэтому о них следует знать, особенно если приходится создавать более сложные пакеты, чем обычно. Для вывода на экран диалогового окна Workflow Properties необходимо щелкнуть правой кнопкой мыши на обозначении рассматриваемого этапа пакета и выбрать команду Workflow Properties из всплывающего меню.

Свойство Close connection on completion

Опция Close connection on completion служит для средств DTS указанием, что требуется закрыть соединение, связанное с задачей, после завершения выполнения этой задачи. Такая возможность является удобной в описанных ниже обстоятельствах.

- Количество соединений, которые могут быть установлены с источником данных, является ограниченным, поэтому необходимо рационально подходить к использованию этих соединений.

- Стоимость поддержки соединений в открытом виде неприемлемо высока.
- Необходимо как можно скорее освобождать блокировки, устанавливаемые на ресурсах (например, на локальных дисковых файлах) в открытом соединении.
- Необходимо обновить метаданные, относящиеся к соединению, которые были записаны в кэш во время открытия соединения.

Свойство Execute on main package thread

По умолчанию средства DTS являются многопотоковыми и обеспечивают параллельное выполнение задач. Для контроля над тем, какое максимальное количество задач может выполняться параллельно, используется диалоговое окно Package Properties. Но некоторые компоненты COM не являются свободнопотоковыми и не поддерживают параллельное выполнение задач (например, определяемые пользователем задачи, созданные с помощью языка VB, являются исключительно апартаментно-потоковыми, как и некоторые средства доступа OLE DB).

Программа DTS Designer автоматически задает опцию Execute on main package thread от имени пользователя для задач, распознаваемых этой программой как требующие применения указанной опции, но пользователь может задать ее самостоятельно для сопутствующих задач. Дело в том, что задачи, характеризующиеся зависимостью от апартаментно-потоковой задачи, также должны выполняться в главном потоке пакета (в качестве примера можно указать задачу ActiveX Script или задачу Dynamic Properties, которая вносит изменения в апартаментно-потоковую задачу, а также в задачу Execute Package, которая вызывает на выполнение пакет, содержащий апартаментно-потоковую задачу). Если возникает ситуация, в которой необходимо использовать в пакете DTS подобный компонент, можно разрешить опцию потока данных Execute on main package thread, чтобы гарантировать для себя возможность безопасного доступа к этому компоненту.

Свойство `Step priority`

Каждый поток, создаваемый службами DTS для выполнения задач, имеет по умолчанию приоритет потока `Normal`. Это значение можно откорректировать для какой-то конкретной задачи, чтобы увеличить или уменьшить приоритет этой задачи по отношению к другим задачам пакета. Такое изменение приоритета вынуждает средства DTS вызвать функцию `SetThreadPriority` API-интерфейса Win32 для корректировки приоритета соответствующего потока по отношению к другим рабочим потокам (дополнительные сведения о функции `SetThreadPriority` приведены в главе 3). При обычных условиях необходимость в корректировке приоритета задачи возникать не должна, но свойство Step priority может оказаться удобным, если приходится использовать задачи, требующие больших затрат процессорного времени, которые должны быть выполнены как можно быстрее, даже за счет других работ, выполняемых в пакете.

Службы DTS и транзакции

При обычных обстоятельствах каждая модификация, внесенная при выполнении пакета DTS, фиксируется сразу после ее выполнения. При выполнении пакета могут также инициализироваться транзакции в зависимости от значения свойства пакета Use transactions. И хотя это свойство по умолчанию имеет истинное значение, необходимо также разрешить использование свойства потока данных Join transaction if present (Присоединиться к транзакции, если она задана) для конкретного этапа пакета, чтобы данные о проводимых изменениях записывались в очередь в составе отдельной транзакции. Поскольку это свойство потока данных по умолчанию имеет ложное значение, изменения, внесенные в ходе выполнения пакета, обычно фиксируются сразу после их осуществления.

Итак, настройка конфигурации пакета может быть выполнена так, чтобы изменения, внесенные в ходе его выполнения, регистрировались в виде отдельной транзакции. Для этого необходимо выполнить описанные ниже действия.

- Разрешить использование свойства пакета Use transactions (которое имеет по умолчанию истинное значение).
- Разрешить использование свойства потока данных Join transaction if present для каждого этапа, который должен стать частью транзакции.

Для того чтобы запуск транзакции прошел успешно, должны быть соблюдены описанные ниже условия.

- На компьютере должен быть обеспечен доступ к службе MSDTC (Microsoft Distributed Transaction Coordinator — координатор распределенных транзакций Microsoft). Дело в том, что для участия в нескольких соединениях требуется создать распределенную транзакцию, поэтому службами DTS всегда предпринимается попытка запустить распределенную транзакцию независимо от того, какое количество соединений определено в пакете.
- Используемые источники данных должны поддерживать транзакции (например, базы данных SQL Server и Oracle поддерживают транзакции, а драйверы доступа к файлам dBase — нет).
- Этапы пакета, которые требуется включить в состав транзакции, должны представлять собой поддерживаемые типы задач (например, задача Bulk Insert поддерживается, а задача Execute Process — нет).

Следует отметить, что в результате настройки этапа пакета для присоединения к транзакции происходит то, что в состав транзакции включаются и соединения, относящиеся к данному этапу, поэтому в состав транзакции включаются и все другие задачи, в которых применяются те же соединения, даже если для них не разрешена опция Join transaction if present. Чтобы предотвратить такую ситуацию, следует использовать отдельные соединения для тех задач, которые необходимо включать в транзакции, и для тех, которым это не требуется.

Для контроля над тем, может ли на каком-то конкретном этапе выполнения пакета осуществляться фиксация текущей транзакции в случае успеха или откат транзакции в случае неудачи, используются соответственно опции потока данных Commit transaction on successful completion of this step (Зафиксировать транзакцию после

успешного завершения данного этапа) и Rollback transaction on failure (Выполнить откат транзакции в случае неудачи). Для контроля над тем, должна ли выполняться фиксация открытой транзакции после успешного завершения всего пакета в целом, может также применяться свойство пакета Commit on successful package completion (Зафиксировать транзакцию после успешного выполнения пакета).

Пакет, вызываемый на выполнение из другого пакета с помощью задачи Execute Package, наследует транзакционный контекст своего вызывающего пакета, если вызвавшая дочерний пакет задача Execute Package присоединилась к транзакции родительского пакета. В результате такого присоединения задачи Execute Package к транзакции родительского пакета транзакционная семантика дочернего пакета существенно изменяется. Во-первых, в дочернем пакете не инициализируется транзакция, даже если в этом пакете разрешена опция Use transactions и имеются этапы с разрешенной опцией Join transaction if present. Во-вторых, игнорируются опции Commit transaction on successful completion of this step и Commit on successful package completion. В дочернем пакете, независимо от значений указанных опций, фиксация не выполняется. Но следует учитывать, что осуществление отката в дочернем пакете все же приводит к откату транзакции родительского пакета.

Для проведения экспериментов со службой DTS и транзакциями можно загрузить пакет `TransactionExample.DTS`, находящийся в каталоге `CH20` компакт-диска, прилагаемого к данной книге. В этом пакете происходят два преобразования данных, входными и выходными наборами для которых являются базы данных `pubs` и `Northwind`, создаются две новые таблицы, и каждая из этих таблиц заполняется данными.

В частности, при проведении экспериментов рекомендуем остановить и запретить использование службы Microsoft Distributed Transaction Coordinator для ознакомления с тем, как эти действия повлияют на способность пакета к выполнению.

Управление потоком данных пакета с помощью средств поддержки сценариев

Как уже было сказано, потоком данных пакета можно управлять с использованием кода сценариев ActiveX. Одним из случаев, в которых, безусловно, возникает необходимость в использовании функциональных возможностей такого типа, является реализация циклов в пакете. Для достижения указанной цели можно применить несколько способов. В данном разделе описан ряд таких способов, а на основании этого описания читатель сможет определить, какая организация цикла является для него наиболее подходящей.

Организация цикла с использованием отдельного пакета

Простейшим способом обеспечения циклической организации работы в пакете DTS является оформление этапов, которые требуется выполнять повторно в виде отдельного пакета, и использование простого сценария ActiveX для вызова

этого пакета на выполнение такое количество раз, какое потребуется. В листинге 20.6 приведен пример задачи ActiveX Script, в которой осуществляется проход по циклу заданное количество раз, и в каждой итерации цикла вызывается на выполнение другой пакет. (Указанный в данном разделе код можно найти в примере пакета `outer.dts` в каталоге `CH20` компакт-диска, прилагаемого к данной книге.)

Листинг 20.6. Пример кода задачи ActiveX Script

```
Function Main()
  Dim oPkg
  Set oPkg=CreateObject("DTS.Package")
  oPkg.LoadFromStorageFile "inner.dts", ""
  For x=1 TO 5
    oPkg.Execute
  Next
  Main = DTSTaskExecResult_Success
End Function
```

Следует отметить, что код, приведенный в листинге 20.6, не опирается на среду DTS этапа прогона, поэтому может быть выполнен как автономный сценарий VBScript (хотя, безусловно, результат из рассматриваемой функции не будет получен). В этом листинге просто создается объект пакета DTS, пакет загружается из структурированного файла хранения и пять раз вызывается на выполнение. При этом явное освобождение объекта (путем присваивания переменной `oPkg` значения `Nothing`) не осуществляется, поскольку такое действие производится от имени пользователя средой этапа прогона сценария. Пример автономного сценария VBScript, в котором пакет повторно вызывается на выполнение, приведен в файле `loop.vbs` на языке VBScript, который находится в каталоге `CH20` компакт-диска, прилагаемого к данной книге.

Организация цикла с использованием свойства `ExecutionStatus`

Еще один способ, с помощью которого можно управлять потоком данных пакета из сценария ActiveX и влиять на ход выполнения действий в цикле, состоит в том, чтобы снова изменять значение свойства `ExecutionStatus` уже выполненного этапа, присваивая ему прежнее значение — `DTSStepExecStat_Waiting`. Такая операция приводит к повторному выполнению этапа, в результате чего в пакете создается своего рода повторяющийся цикл. Пользователь может принимать решение об изменении значения свойства `ExecutionStatus`, руководствуясь выбранными им логическими условиями, поэтому в распоряжении пользователя имеются любые инструментальные средства, необходимые для создания логического цикла, не уступающие тем, которые обычно используются в общепринятой среде программирования.

При использовании свойства `ExecutionStatus` для реализации цикла в пакете DTS применяется целый ряд различных подходов. Наиболее удобным и всесторонним из них является связывание некоторого сценария ActiveX с определенным этапом пакета с помощью диалогового окна Workflow Properties. При

этом для определения повторяемого этапа применяется коллекция Steps текущего пакета, после чего свойству ExecutionStatus этого этапа присваивается значение DTSStepExecStat_Waiting.

Такой подход лучше по сравнению с подходом, который будет описан в следующем подразделе, поскольку в нем не требуется предусматривать отдельные задачи ActiveX Script для достижения единственной цели — обеспечения циклической организации работы. В данном подходе сценарий ActiveX закрепляется за уже существующим этапом пакета с помощью ассоциации Workflow Properties, поэтому устраняется необходимость в использовании отдельных компонентов лишь для циклической организации работы, а также существует возможность применять каждый этап пакета в качестве инициализатора цикла, тела цикла или оператора управления циклом.

Еще одно преимущество подхода, рассматриваемого в настоящем разделе, состоит в том, что с его помощью можно получить контроль над тем, будет ли выполняться этап, с которым связан применяемый сценарий ActiveX, с учетом условия организации цикла. Например, может возникнуть необходимость в том, чтобы этап не выполнялся до тех пор, пока не завершится цикл. А поскольку с потоком данных этапа связан сценарий ActiveX, то из этого сценария ActiveX можно возвращать значение DTSStepScriptResult_DontExecuteTask свойства ExecutionStatus, чтобы исключить возможность выполнения данного этапа.

Пример цикла в пакете DTS, реализованного с помощью привязки сценариев ActiveX к этапам потока данных пакета, приведен в листинге 20.7. (Код этого пакета можно найти на компакт-диске, прилагаемом к данной книге, в файле CH20\LoopExample.dts — см. код примера Loop1.)

Листинг 20.7. Пример цикла в пакете DTS, реализованного с помощью привязки сценариев ActiveX к этапам потока данных пакета

```
'   Этап 1

'********************************************************************
'  Инициализировать переменную управления циклом
'********************************************************************

Function Main()
  DTSGlobalVariables("foo").Value=0
  MsgBox "Loop1:  Initialize"
  Main = DTSTaskExecResult_Success
End Function

'   Этап 2

'********************************************************************
' Выполнить обработку в цикле
'********************************************************************

Function Main()
  MsgBox "Loop1:  Work"
  Main = DTSTaskExecResult_Success
End Function

'   Этап 3
```

```
'**********************************************************************
'  Проверить значение переменной управления циклом и в случае
'  необходимости повторить текущий этап
'**********************************************************************

Function Main()
  Dim oPkg
  DTSGlobalVariables("foo").Value = _
  DTSGlobalVariables("foo").Value + 1

  If DTSGlobalVariables("foo").Value < 5 Then
    MsgBox "Loop1:  " & DTSGlobalVariables("foo").Value
    Set oPkg = DTSGlobalVariables.Parent

    oPkg.Steps("DTSStep_DTSActiveScriptTask_6").ExecutionStatus = _
      DTSStepExecStat_Waiting

    Main = DTSStepScriptResult_DontExecuteTask
  Else
    Main = DTSStepScriptResult_ExecuteTask
  End If

End Function
```

Как уже было сказано, предусмотрен целый ряд способов использования свойства `ExecutionStatus` для создания механизма организации цикла в пакете DTS. Еще один способ достижения этой цели состоит в применении задач ActiveX Script для реализации кода организации цикла. Этот способ действует по принципу, описанному ниже.

1. Создается пакет, в состав которого входит этап, выполняемый неоднократно.
2. Осуществляется привязка этапа задачи ActiveX Script к рассматриваемому этапу с помощью ограничения предшествования. Таковым ограничением может служить любое из трех стандартных ограничений потока данных (On Success, On Completion или On Failure), в зависимости от того, какой цели фактически пытается достичь разработчик. При реализации простого цикла, скорее всего, следует использовать первые два из указанных ограничений (On Success или On Completion), а при реализации логического алгоритма, предусматривающего проведение повторных попыток, может использоваться ограничение On Failure.
3. В коде сценария ActiveX проверяется логическое условие, позволяющее определить, следует ли повторить начальный этап (такая проверка необходима при том условии, что цикл не должен быть бесконечным), после чего свойству `ExecutionStatus` данного этапа в случае положительно ответа присваивается значение `DTSStepExecStat_Waiting`.

Приведенное выше описание можно проще всего понять, рассматривая сам код, поэтому в листинге 20.8 приведен исходный код ActiveX из пакета, в котором демонстрируется рассматриваемый метод. (Этот код можно найти на компакт-диске, прилагаемом к данной книге, в файле `CH20\LoopExample.dts` — см. пример кода Loop2.)

Листинг 20.8. Исходный код ActiveX

```
'    Этап 1

'*********************************************************************
'  Инициализировать переменную управления циклом
'*********************************************************************

Function Main()
  DTSGlobalVariables("bar").Value=0
  MsgBox "Loop2:  Initialize"
  Main = DTSTaskExecResult_Success
End Function
'    Этап 2

'*********************************************************************
' Выполнить обработку в цикле
'*********************************************************************

Function Main()
  MsgBox "Loop2:  Work"
  Main = DTSTaskExecResult_Success
End Function

'    Этап 3

'*********************************************************************
'  Проверить значение переменной управления циклом и в случае
'  необходимости повторить текущий этап
'*********************************************************************

Function Main()

  DTSGlobalVariables("bar").Value=DTSGlobalVariables("bar").Value+1
  If DTSGlobalVariables("bar").Value<5 Then
    MsgBox "Loop2:  " & DTSGlobalVariables("bar").Value
    Dim oPkg
    Set oPkg = DTSGlobalVariables.Parent
    oPkg.Steps("DTSStep_DTSActiveScriptTask_1").ExecutionStatus _
        = DTSStepExecStat_Waiting
  End If
  Main = DTSTaskExecResult_Success
End Function
```

В листинге 20.8 приведен код пакета, состоящего из трех этапов. На этапе 1 сценарий ActiveX используется для инициализации переменной управления циклом — глобальной переменной bar. На этапе 2 проводится работа, которая должна выполняться неоднократно. На этапе 3 увеличивается значение управляющей переменной и проводится проверка для определения того, остается ли это значение меньшим 5; в случае положительного ответа свойству ExecutionStatus задачи ActiveX этапа 2 присваивается значение DTSStepExecStat_Waiting. Это действие приводит к тому, что этап 2 выполняется повторно, а это приводит к этапу 3, причем на этот раз снова увеличивается и проверяется значение переменной управления циклом для определения того, продолжить ли выполнение цикла или перейти к оставшейся части пакета.

Некоторые разработчики предпочитают данный подход тому подходу, который был представлен в этом разделе первым, поскольку код ActiveX, в котором реализуется цикл, является более легко доступным в программе DTS Designer. А в первом подходе доступ к коду в большей степени затруднен из-за использования связей со свойствами потока данных.

Обратите внимание на то, что для получения ссылки на выполняемый в настоящее время пакет применяется переменная экземпляра `DTSGlobalVariables.Parent`. Такая переменная концептуально эквивалентна переменной "`this`" в языке C++ или C#, переменной "`Me`" в языке VB и переменной "`Self`" в языке Object Pascal. В пакете, приведенном в листинге 20.8, демонстрируется удобный способ получения ссылки на пакет, выполняемый в настоящее время, из сценария ActiveX.

Заслуживает также внимания то, что в качестве индекса для коллекции `Steps` используется имя этапа. Чтобы определить имя этапа, можно щелкнуть правой кнопкой мыши на узле с обозначением этапа, выбрать команду Workflow Properties и щелкнуть на странице Options (имя этапа указано в верхней части этой страницы). Доступ к коллекции объектов в коде VB обычно осуществляется с применением либо порядкового индекса, либо имени элемента коллекции. В рассматриваемом листинге для удобства чтения используется имя.

Подход к реализации цикла с применением любого из способов, основанных на присваивании значений свойству `ExecutionStatus`, является более удобным по сравнению с подходом, основанным на выполнении в цикле целого пакета, по очевидной причине — в подходе с использованием свойства `ExecutionStatus` не требуется предварительно помещать этапы, которые должны выполняться повторно, в отдельный пакет. Это означает, что для определения необходимости продолжать выполнение цикла могут применяться проверки сложных условных выражений, к тому же, такие проверки могут проводиться на нескольких разных этапах. Примером той ситуации, в которой может потребоваться проверка условия цикла на нескольких этапах и присваивание свойству `ExecutionStatus` соответствующего значения, может служить реализация логического алгоритма осуществления повторных попыток. Например, допустим, что необходимо выполнить целый ряд этапов для определения допустимости данных в процессе загрузки, причем пакет должен быть организован таким образом, чтобы в случае неудачного завершения любого из таких этапов процесс обработки перезапускался. В такой ситуации можно использовать сценарий ActiveX для проверки условий завершения в случае неудачи на нескольких этапах пакета и присваивать свойству `ExecutionStatus` соответствующее значение при неудачном завершении любых из этих этапов.

Реализация способа выполнения по условию

Как уже мог предположить читатель, ассоциации со сценариями ActiveX можно также использовать для реализации способа выполнения по условию. В первом примере организации цикла с помощью свойства `ExecutionStatus`, приведенном выше, из сценария ActiveX, связанного с потоком данных этапа, возвращается значение `DTSStepScriptResult_DontExecuteTask`, если выполнять этот этап не требуется, и значение `DTSStepScriptResult_ExecuteTask`, если тре-

буется. Тот же метод можно применить для выполнения этапа по условию с учетом любых критериев, которые могут быть проверены из сценария ActiveX.

Следует отметить, что ассоциации со сценариями ActiveX можно также использовать для реализации логических алгоритмов повторного осуществления попыток. Из сценария ActiveX потока данных можно возвращать не только значения DTSStepScriptResult_ExecuteTask и DTSStepScriptResult_DontExecuteTask, но и значение DTSStepScriptResult_RetryLater, чтобы обеспечить повторное выполнение данного этапа в какой-то последующий момент в процессе выполнения всего пакета.

Организация выполнения по условию с помощью свойства ExecutionStatus и задач ActiveX Script

Для реализации способов выполнения этапов пакета по условию может использоваться разновидность способа организации цикла с помощью задачи ActiveX Script. Допустим, например, что нужно, чтобы был выполнен указанный ряд этапов пакета, если данное конкретное условие является истинным, а в противном случае этот ряд этапов должен быть пропущен. Одним из способов достижения этой цели является проверка условия в задаче ActiveX Script, которая предшествует рассматриваемому ряду этапов, и присваивание свойству ExecutionStatus первого этапа в этом ряде значения DTSStepExecStat_Inactive, если этот этап не требуется выполнять. Перевод некоторого этапа в неактивное состояние приводит к тому, что отменяется выполнение и этого этапа, и тех этапов, которые следуют за ним. Код VBScript, который иллюстрирует описанный метод, приведен в листинге 20.9.

Листинг 20.9. Код VBScript, котором используются свойства ExecutionStatus и задачи ActiveX Script

```
Function Main()
  Dim oPkg
  Set oPkg = DTSGlobalVariables.Parent
  If DTSGlobalVariables("bFoo") Then
     oPkg.Steps("DTSStep_DTSCreateProcessTask_1") _
        .ExecutionStatus = DTSStepExecStat_Waiting
  Else
     oPkg.Steps("DTSStep_DTSCreateProcessTask_1") _
        .ExecutionStatus = DTSStepExecStat_Inactive
  End If
  Main = DTSTaskExecResult_Success
End Function
```

Как показано в листинге 20.9, этап DTSStep_DTSCreateProcessTask_1 следует за задачей ActiveX Script, в которой выполняется данный код, поэтому указанный этап (и этапы, следующие за ним) будут выполнены, только если глобальной переменной bFoo присвоено истинное значение на входе в этот этап.

Способ выполнения по условию с учетом успешного или неудачного завершения задачи

Еще один способ организации выполнения по условию состоит в том, чтобы предусматривалась проверка значения управляющей переменной и возврат значения DTSTaskExecResult_Failure, если требуется предотвратить выполнение этапов, которые следуют за конкретной задачей ActiveX Script (при условии, что эти этапы связаны с ограничениями предшествования On Success). Пример сценария ActiveX, который возвращает значение, определяющее успешное или неудачное завершение задачи, приведен в листинге 20.10.

Листинг 20.10. Пример сценария ActiveX, который возвращает значение, определяющее успешное или неудачное завершение задачи

```
Function Main()
  If DTSGlobalVariables("bFoo") Then
    Main = DTSTaskExecResult_Success
  Else
    Main = DTSTaskExecResult_Failure
  End If
End Function
```

Как показано в листинге 20.10, проверяется значение глобальной переменной, и возвращаемому результату задачи присваивается значение с учетом значения этой глобальной переменной. Описанный в данном разделе способ, в отличие от остальных, не обеспечивает бесперебойное функционирование приложения, поскольку его применение может привести к появлению сообщений об ошибках, вырабатываемых машиной выполнения пакета DTS. Безусловно, можно игнорировать эти сообщения об ошибках, но использование одного из остальных способов вместо указанного в настоящем разделе позволяет полностью избежать появления сообщений об ошибках.

Параметризованные пакеты DTS

При использовании пакета DTS возникает вполне обоснованное стремление ввести в него параметры для того, чтобы можно было, например, применять этот пакет для работы с входными и выходными наборами данных, отличными от тех, которые были первоначально заданы при создании этого пакета. В ходе подобной параметризации чаще всего предпринимаются попытки использовать в качестве параметров значения глобальных переменных, свойства соединений, имена объектов и тому подобные значения.

Для параметризации пакетов DTS может использоваться целый ряд способов. Один из основных способов состоит в применении задачи Dynamic Properties. Задача Dynamic Properties позволяет присвоить значение любому свойству в пакете с использованием одного из шести описанных ниже источников данных.

1. Глобальная переменная из пакета.
2. Значение из файла INI.
3. Переменная среды.
4. Запрос T-SQL (используется только первый столбец первой строки в результирующем наборе запроса).
5. Файл данных.
6. Константа.

Значения глобальным переменным могут присваиваться при выполнении пакета (например, с помощью утилиты `dtsrunui` или задачи Execute Package другого пакета), поэтому чаще всего применяется способ, предусматривающий присваивание значений глобальных переменных свойствам пакета с помощью задачи Dynamic Properties с последующим присваиванием значений этим переменным на этапе прогона. Благодаря тому, что глобальные переменные используются таким образом, создается своего рода уровень абстракции между механизмом, формирующим значения параметров, и динамическим способом, применяемым для присваивания значений свойствам в пакете. Такой подход позволяет легко изменять способ выполнения пакета, не нарушая динамического характера пакета.

Еще один способ параметризации пакета DTS состоит в использовании кода сценария ActiveX. На практике часто бывает проще модифицировать свойство пакета с помощью кода сценария, чем специально подготавливать пакет для выполнения под управлением задачи Dynamic Properties. Кроме того, следует учитывать, что с помощью задачи Dynamic Properties нельзя корректировать только часть многозначного значения свойства с помощью глобальных переменных, поскольку в данном случае изменяется либо все значение свойства, либо это значение вообще не изменяется. Использование кода сценария или кода Automation за пределами пакета обеспечивает больший контроль над процессом изменения значений свойств на этапе прогона и позволяет применять при этом глобальные переменные с помощью любого подходящего для этого способа.

Упражнение

Чтобы лучше разобраться в том, как может быть параметризован пакет, загрузите пакет `ParamExample.DTS` из каталога `CH20` компакт-диска, прилагаемого к данной книге, в программу DTS Designer. Этот пакет позволяет скопировать выбранную таблицу из указанной исходной базы данных одной СУБД в целевую базу данных другой СУБД.

Упражнение 20.1. Пример параметризованного пакета

1. Дважды щелкните кнопкой мыши на задаче Dynamic Properties с именем Get Params.
2. Дважды щелкните на каждом из элементов в списке Change, чтобы узнать, для присваивания каких свойств используются эти элементы. При этом необходимо убедиться в том, что глобальные переменные применяются для присваивания значений свойствам основных соединений для задачи TransferObject.

3. Выйдите из диалогового окна Dynamic Properties и щелкните правой кнопкой мыши на канве программы DTS Designer. Выберите команду Package Properties из всплывающего меню.
4. Щелкните на вкладке Global Variables в диалоговом окне свойств. В этом окне должны быть показаны шесть определяемых глобальных переменных, которые относятся к исходной СУБД, базе данных и таблице, а также к целевой СУБД, базе данных и таблице.
5. Запустите утилиту `dtsrunui`. В диалоговом окне DTS Run измените значение поля Location, выбрав Structured Storage File (Структурированный файл хранения), затем выберите пакет `ParamExample.DTS`, который перед этим рассматривался в программе DTS Designer, щелкнув на кнопке со знаком троеточия справа от текстового поля File name:.
6. Введите в качестве имени пакета `ParamExample`, затем щелкните на кнопке Advanced, чтобы вывести на экран диалоговое окно, которое позволит задавать значения глобальных переменных для пакета перед его выполнением.
7. Задайте значения для исходной СУБД, базы данных и таблицы, а также для целевой СУБД, базы данных и таблицы. Для этого упражнения превосходно подходят базы данных `pubs` и `Northwind`.
8. Щелкните на кнопке OK, чтобы закрыть диалоговое окно Advanced DTS Run, затем щелкните на кнопке Run, чтобы выполнить пакет с указанными параметризованными значениями. После этого пакет должен быть выполнен успешно, в результате чего указанный объект должен быть скопирован из исходной базы данных в целевую.

Средство доступа к набору строк DSO

Пакеты DTS позволяют весьма эффективно обрабатывать данные. Один из самых интересных способов состоит в использовании средства доступа OLE DB для формирования запроса к пакету DTS. Дело в том, что отдельный этап пакета можно обозначить как средство доступа к набору строк DSO, а затем передать запрос к пакету из сценария T-SQL с помощью команды `OPENROWSET` и средства доступа OLE DB объекта `DTSPackageDSO`. Это позволяет, например, предоставить доступ пользователю к результатам одного из преобразований как к набору строк, который можно запрашивать с помощью языка T-SQL. Безусловно, это означает, что один пакет может служить источником данных для другого пакета, поскольку, разумеется, существует возможность определять запросы T-SQL в качестве источника данных для преобразования в составе пакета. Это также означает, что предусмотрена возможность передавать функции сложной обработки данных в пакет DTS, затем вызывать соответствующее преобразование из запроса T-SQL или из хранимой процедуры. Ниже приведен пример запроса T-SQL, в котором происходит выборка результатов задачи преобразования в виде результирующего набора.

```
SELECT *
FROM OPENROWSET('DTSPackageDSO', '/FD:\CH20\DSORowsetExample.dts',
'SELECT * ')
```

Средства доступа DTSPackageDSO поддерживают два параметра — набор параметров, подобный параметрам командной строки утилиты dtsrun, и текст запроса. Параметры командной строки dtsrun можно использовать для указания входного набора данных и местонахождения пакета. В приведенном выше примере дана ссылка на пакет, хранящийся в виде структурированного файла хранения COM.

Обратите внимание на то, что в тексте запроса, передаваемого в функцию OPENROWSET, отсутствует что-либо, напоминающее имя таблицы. Это связано с тем, что пакет, указанный в ссылке, имеет только один этап, который обозначен как средство доступа к набору строк DSO. Если бы таким образом было обозначено несколько этапов, то нужно было бы указать имя этапа в тексте запроса для обозначения того этапа, к которому требуется применить запрос, как показано ниже.

```
SELECT *
FROM OPENROWSET('DTSPackageDSO',
'/FD: \CH20\DSORowsetExample.dts',
'SELECT * FROM DTSStep_DTSDataPumpTask_1')
```

Следует отметить, что во время прогона пакета этап, обозначенный как средство доступа к набору строк DSO, фактически не выполняется. Дело в том, что такой этап резервируется строго для предоставления данных и игнорируется машиной выполнения пакета. Для проведения экспериментов с задачей средства доступа к набору строк DSO путем выполнения запросов к пакету DSORowsetExample.DTS (находящемуся в каталоге CH20 компакт-диска, прилагаемого к данной книге) из программы Query Analyzer можно воспользоваться примером сценария DSORowsetExample.SQL на языке T-SQL. Рекомендуем вначале загрузить указанный пакет в программу DTS Designer, чтобы можно было выполнить настройку конфигурации свойств соединения и провести другую подготовительную работу.

Использование средств DTS для преобразования данных, репликация которых осуществляется с помощью подписок

Средства репликации программы SQL Server будут рассматриваться более подробно в главах 21–23, а в этом разделе показан способ использования пакета DTS для преобразования опубликованных данных в ходе их доставки подписчикам. Читатель, по-видимому, уже знаком с определением понятия *репликационного сервера публикации* — сервера (или другого средства), который предоставляет данные подписчикам; если доставка данных осуществляется по принципу репликации, то подписчики рассматриваются как потребители данных. Для манипулирования опубликованными данными в ходе доставки данных подписчикам можно использовать язык T-SQL, но еще одна возможность состоит в создании пакетов DTS, выполняющих сложные преобразования данных в ходе их доставки. Пакет, используемый для преобразования данных подписки, может находиться на том же компьютере, где размещен распределительный сервер, или на компьютерах

отдельных подписчиков. Кроме того, такой пакет может применяться не только для преобразования опубликованных данных, но и для разбиения таблиц с данными на отдельные секции.

Для создания подписки, поддерживающей преобразование, необходимо начать с создания преобразуемой публикации, как описано ниже.

1. Создайте новый снимок или транзакционную публикацию с использованием программы-мастера Create Publication. При этом в окне программы-мастера необходимо отметить флажок Show advanced options (Показывать дополнительные опции).
2. Исключите возможность обновления публикации в программе-мастере (немедленного обновления или обновления по мере обработки очереди). Обновляемые публикации и преобразуемые подписки являются взаимоисключающими.
3. Щелкните кнопкой Yes на странице Transform Published Data в программе-мастере Create Publication.
4. Завершите создание публикации, выбрав статьи, опции снимка и прочие параметры в программе-мастере Create Publication.

После создания преобразуемой публикации можно приступить к определению самого преобразования. Для этого используется программа-мастер Transform Published Data, как описано ниже.

1. Щелкните правой кнопкой мыши на преобразуемой публикации и выберите команду Properties. На странице Subscriptions диалогового окна Publication Properties щелкните на кнопке Transformations.
2. Появится приглашение к вводу имени преобразуемой публикации. Оставьте неизменным значение, предложенное по умолчанию, и щелкните на кнопке Next.
3. Затем требуется выбрать выходной набор для преобразования. Если необходимо предусмотреть использование данной публикации несколькими подписчиками, выполните настройку конфигурации с учетом такого места назначения, которое является наиболее представительным для всех намеченных подписчиков в целом, затем щелкните на кнопке Next.
4. В следующем диалоговом окне определите требуемые преобразования для рассматриваемых данных. Щелкните на кнопке со знаком троеточия справа от каждой статьи, чтобы вывести на экран диалоговое окно Column Mappings and Transformations (Отображения и преобразования столбцов). В этом окне можно выполнить простые присваивания для взаимно-однозначного отображения столбцов или определить более сложные преобразования ActiveX.
5. Затем в конфигурацию вводится информация о том, где физически должен находиться пакет DTS. Этот пакет можно разместить на том же компьютере, где размещен распределительный сервер, или на компьютере подписчика.
6. Настройка конфигурации завершается путем присваивания имени пакета. Пакет будет храниться в таблице `msdb..sysdtspackages` на сервере, вы-

бранном в предыдущем пункте. Пакеты преобразования данных подписки нельзя хранить в файлах с форматом COM Structured Storage File (Файл структурированного хранения COM) или в репозитариях Meta Data Services (Службы метаданных).

7. Щелкните на кнопке Finish, чтобы создать пакет преобразования.
8. Щелкните на кнопке OK, чтобы выйти из диалогового окна Publication Properties. При оформлении подписки на преобразуемую публикацию пользователю поступит приглашение к вводу имени пакета, применяемого для преобразования данных подписки. Следует отметить, что с каждой публикацией может быть связано несколько пакетов преобразования. Такая особенность может использоваться для преобразования и разбиения данных по секциям различным образом для каждого подписчика.

Принципы работы пакета DTS, используемого для преобразования данных подписки

Пакет DTS, созданный программой-мастером Transform Published Data, состоит по меньшей мере из четырех объектов: три объекта связаны с каждой статьей — объект `Connection`, задача Execute SQL и задача Data Driven Query, а один объект `Connection` используется всеми статьями для предоставления данных подписчикам. Задача Execute SQL обеспечивает выборку строк из исходной статьи для предоставления данных, применяемых в задаче Data Driven Query. При создании преобразуемых подписок вместо задачи Transform Data или Bulk Insert всегда используется задача Data Driven Query, которая позволяет выполнять операции прямого копирования из столбца в столбец или более сложные преобразования с помощью предоставленного пользователем кода сценариев ActiveX, как уже было сказано.

Пакеты преобразования можно открывать в программе DTS Designer точно так же, как и любые другие локальные пакеты SQL Server. Для этого необходимо подключиться к компьютеру, на котором находится распределительный сервер или программа подписчика (в зависимости от того, на каком компьютере хранится пакет), из программы Enterprise Manager, затем щелкнуть на узле Local Packages, находящемся под узлом Data Transformation Services. В правом списке следует дважды щелкнуть на пакете, который был определен в программе-мастере Transform Published Data, чтобы его открыть.

Чтобы можно было осуществлять выборку данных из публикации, пакет преобразования DTS должен содержать объект `Connection`, который ссылается на объект SQL Server Replication OLE DB Provider for DTS (Средство доступа OLE DB репликации SQL Server для служб DTS). После вывода на экран диалогового окна Properties, относящегося к одному из таких объектов `Connection`, и щелчка на кнопке Properties можно получить список столбцов, опубликованных с помощью рассматриваемой статьи, на вкладке All диалогового окна Data Link Properties. Указанное выше средство доступа спроектировано для исключительного использования в пакетах преобразуемых подписок и не доступно из применяемой по умолчанию палитры Connection (Соединение) программы DTS Designer.

Определяемые пользователем задачи

Как уже было сказано, функционирование служб DTS основано на расширяемой модели COM. Одним из преимуществ такой организации работы является то, что пользователь имеет возможность создавать свои собственные задачи как COM-объекты и устанавливать эти задачи в программе DTS Designer. Для достижения указанной цели можно воспользоваться одним из многочисленных способов, в частности, объект определяемой пользователем задачи может быть создан с нуля на языке, позволяющем реализовывать интерфейсы COM; можно также откорректировать в соответствии с требованиями пользователя один из образцов компонентов определяемой пользователем задачи, которые входят в поставку программы SQL Server. Любой из указанных выше подходов может оказаться практически применимым способом расширения служб DTS, в зависимости от потребностей пользователя, поэтому автор покажет, как осуществляется и тот, и другой способ.

Создание новой определяемой пользователем задачи

Все объекты задач DTS реализуют интерфейс DTS `CustomTask`. Этот интерфейс должна также реализовывать любая определяемая пользователем задача. С точки зрения реализации интерфейса встроенные и определяемые пользователем задачи не различаются, поскольку встроенные задачи службы DTS представляют собой просто компоненты COM, которые как минимум реализуют интерфейс `CustomTask`. В определяемой пользователем задаче могут быть также реализованы другие интерфейсы, такие как `CustomTaskUI` (для компонентов, которые определяют свои собственные интерфейсы пользователя), но в данном разделе речь в основном пойдет об интерфейсе `CustomTask`, поскольку именно этот интерфейс позволяет использовать компонент COM в качестве определяемой пользователем задачи DTS.

В следующем упражнении показано, как создать на языке Visual Basic определяемую пользователем задачу, которая способна выполнять сценарии T-SQL. У читателя может возникнуть вопрос, почему рассматривается пример создания такого компонента, учитывая то, что уже существует встроенная задача Execute SQL. Причина этого проста – задача Execute SQL действительно позволяет выполнять сценарии T-SQL, но не имеет механизма, позволяющего справляться с большими объемами разнообразного вывода сценариев. Это означает, что, если выполняется сценарий, возвращающий несколько результирующих наборов, чередующихся с выводом операторов `PRINT` и `RAISERROR`, то пользователь не имеет возможности осуществить выборку подобных выходных данных из задачи Execute SQL, а такие инструментальные средства, как Query Analyzer и `osql`, предоставляют указанные возможности.

В действительности в новой определяемой пользователем задаче просто вызывается утилита исполнения сценария, которая указана с помощью свойства. В данном случае удобный вариант состоит в использовании утилиты `osql`, но может быть также вызвана программа ISQL или какая-то другая утилита исполнения сценария. Итак, приступим к созданию указанного компонента, рассматривая при этом, что и как требуется для этого делать.

Упражнение

В приведенных ниже шагах предполагается использование версии VB6; должна также оказаться применимой версия VB.NET, хотя содержание указанных шагов может немного измениться.

Упражнение 20.2. Создание определяемой пользователем задачи DTS на языке Visual Basic

1. Откройте среду разработки Visual Basic и запустите новый проект ActiveX DLL из диалогового окна New Project. Безусловно, среда этапа прогона DTS (например, `dtsrun`) позволяет эксплуатировать определяемые пользователем задачи в виде внепроцессных компонентов (например, файлов EXE), но если определяемая пользователем задача решается в сочетании с программой DTS Designer, то задача должна быть определена как внутрипроцессный компонент.
2. Из меню Project вызовите на экран диалоговое окно Project Properties и измените значение поля Project Name, введя `ExecuteSQLScript`. На вкладке Component задайте в поле Startup Object значение "`(None)`", в качестве значения Threading Model задайте "`Apartment Threaded`" и выберите опцию Project Compatibility, затем щелкните на кнопке OK.
3. В среде VB вновь созданному классу будет присвоено по умолчанию имя `Class1`. В окне Properties введите вместо этого имени имя `clsExecuteSQLScript`. Кроме того, обязательно задайте в качестве значения свойства `Instancing` строку "`5 - Multiuse`".
4. Щелкните на названии опции References в меню Project и выберите из списка доступных ссылок `Microsoft DTSPackage Object Library`. Эта ссылка обеспечивает импорт в создаваемый проект библиотеки типов COM с именем `DTSPackage` и позволяет обращаться к интерфейсам, доступ к которым предоставляется этой библиотекой.
5. В окне для ввода кода обязательно выберите в двух полях со списком значения "`(General)`" и "`(Declarations)`", затем введите в окне редактора следующую строку:

   ```
   Implements DTS.CustomTask
   ```

   Эта команда сообщает среде VB, что вновь создаваемый компонент будет реализовывать интерфейс `CustomTask` служб DTS (который определен в библиотеке `DTSPackage Object Library`). После этого интегрированная среда разработки VB позволит обращаться в окне редактора кода к методам, доступ к которым предоставляется интерфейсом `CustomTask`.
6. Измените значение в левом поле со списком в окне редактора кода, чтобы в этом окне появилась ссылка на интерфейс `CustomTask`. Редактор должен немедленно вставить пустой метод `Get` свойств `CustomTask_Properties`.
7. Введите следующую строку в текст метода `Get` свойств `CustomTask_Properties`:

   ```
   Set CustomTask_Properties = Nothing
   ```

   Безусловно, во вновь создаваемом компоненте может быть реализован собственный редактор свойств, но в данном случае указанному свойству присваивается

значение Nothing, которое указывает службам DTS, что от имени пользователя должен быть предусмотрен применяемый по умолчанию редактор свойств.

8. В правом поле со списком выберите каждый из остальных методов, доступ к которым предоставляется интерфейсом CustomTask. Редактор VB вставит пустые заготовки методов для реализации каждого из них.
9. Введите следующие две строки в начале создаваемого файла исходного кода:

   ```
   Dim m_bstrName As String
   Dim m_bstrDescription As String
   ```

   Эти две переменные экземпляра будут использоваться для кэширования имени и описания вновь создаваемого компонента.
10. Введите следующую строку в функцию Let, относящуюся к свойству CustomTask_Description:

    ```
    m_bstrDescription = RHS
    ```

    Этот оператор обеспечивает присваивание значения параметра RHS (которое передается в функцию Let при присваивании значения свойству Description) переменной экземпляра, объявленной ранее.
11. Введите следующую строку в функцию Get, относящуюся к свойству CustomTask_Description:

    ```
    CustomTask_Description = m_bstrDescription
    ```

    Выполнение этого оператора приводит к тому, что при каждой выборке значения свойства Description происходит возврат значения переменной экземпляра m_bstrDescription.
12. Выполните подготовку свойства Name по аналогии со свойством Description — введите в функции Let и Get, относящиеся к этим свойствам, такой код, чтобы эти функции присваивали и осуществляли выборку значения переменной экземпляра m_bstrName, объявленной ранее.
13. Следует отметить, что методы Let и Get свойства, определяемые с помощью интерфейса CustomTask, по умолчанию имеют атрибут private. Это связано с тем, что указанные методы ссылаются на свойства базовой определяемой пользователем задачи DTS, а не на саму вновь создаваемую задачу, определяемую пользователем. Для того чтобы эти методы были доступными в пакете DTS, необходимо определить свойства с атрибутом public. Создайте два новых метода (как показано в листинге 20.11), чтобы предоставить доступ к свойству Description из определяемой пользователем задачи.

Листинг 20.11. Код методов Get и Let

```
Public Property Get Description() As String
  Description = m_bstrDescription
End Property
Public Property Let Description(ByVal RHS As String)
  m_bstrDescription = RHS
End Property
```

Не создавайте аналогичные методы для свойства Name, поскольку нет необходимости в том, чтобы в программе DTS Designer предоставлялась возможность

вносить изменения в это свойство. Программа DTS Designer автоматически вырабатывает имя для задачи после перетаскивания объекта задачи на лист проекта этой программы. Мы не нарушаем те принципы, по которым действует программа DTS Designer, но не должны позволить, чтобы программа присвоила другое имя разрабатываемой задаче, поэтому нет смысла предоставлять доступ к такому имени с помощью свойства с атрибутом `public`.

14. К этому времени полностью создан определяемый пользователем объект задачи DTS, поэтому при желании можно откомпилировать этот объект для получения библиотеки DLL и зарегистрировать эту библиотеку в программе DTS Designer. Тем не менее соответствующий компонент еще не выполняет каких-либо действий, поэтому в данном шаге действия по компиляции и регистрации еще не будут выполняться.
15. Введите объявления переменных экземпляра, приведенные в листинге 20.12, в верхней части разрабатываемого файла исходного кода.

Листинг 20.12. Объявления переменных экземпляра

```
Private m_bstrScriptUtility As String
Private m_bstrServerInstance As String
Private m_bstrAuthString As String
Private m_bstrScriptToExecute As String
Private m_bstrOutputFileName As String
Private m_lTimeout As Long
Private m_bTerminateOnTimeout As Boolean
```

Назначение каждой из этих переменных экземпляра должно быть вполне очевидно, но автор все равно приведет краткое описание большинства из них. Переменная экземпляра `m_bstrScriptUtility` должна хранить командную строку, которая была передана в утилиту сценария. Как будет показано ниже, эта переменная экземпляра обеспечивает использование заменяемых параметров, которые фактически определены с помощью других свойств. Переменная экземпляра `m_bstrServerInstance` хранит обозначения имени и экземпляра SQL Server (разделенные обратной косой чертой), к которым должно быть выполнено подключение. Переменная экземпляра `m_bstrAuthString` хранит строку аутентификации, которую требуется передать в утилиту сценария. Для утилиты `osql` эта строка может иметь вид "`-E`" (использовать заслуживающее доверие соединение) или "`-U user -P password`" (применять аутентификацию SQL Server). Переменная экземпляра `m_bstrScriptToExecute` хранит имя сценария, который должен быть выполнен. Следует отметить, что вместо имени сценария может быть также введена строка запроса SQL, если утилита сценария поддерживает возможность передачи строки запроса непосредственно в командной строке этой утилиты (например, как утилита `osql`). Переменная экземпляра `m_bstrOutputFileName` хранит имя выходного файла, который должен быть создан утилитой сценария. Если утилита непосредственно поддерживает возможность перехвата ее вывода и записи в файл, а также позволяет передавать имя этого файла в командной строке, то указанная переменная экземпляра может использоваться для хранения имени файла. Если же утилита непосредственно не поддерживает возможность задавать имя выходного файла, а направляет свой вывод на терминал, то можно вместо

утилиты сценария вызвать на выполнение процессор команд (например, в семействе операционных систем Windows NT — программу CMD.EXE) и применить перенаправление для передачи выходных данных в такой файл. В указанной ситуации придется вызвать утилиту сценария на выполнение с помощью процессора команд, поскольку именно процессор команд служит тем средством, с помощью которого осуществляется перенаправление терминального вывода в файл. Пример применения описанного выше способа приведен в пакете CustomTaskVB_TimeoutExample.DTS в подкаталоге CH20\CustomTaskVB компакт-диска, прилагаемого к данной книге.

16. Подготовьте для каждого из объявленных в предыдущем пункте переменных экземпляра методы Let и Get с атрибутом public, тщательно следя за тем, чтобы при этом правильно использовались типы данных. Чтобы ускорить работу, широко применяйте операции копирования и вставки кода некоторых существующих методов Let и Get.

17. Ниже будет разработан метод Execute определяемой пользователем задачи и все необходимые для него программные средства. В процессе работы потребуется выйти в интерпретатор утилиты сценариев и приостановить выполнение до тех пор, пока эта утилита не завершит свою работу или не выполнит выход по тайм-ауту, поэтому функция Shell языка VB не может использоваться (поскольку она выполняет вызовы асинхронно), и вместо нее должна вызываться функция CreateProcess API-интерфейса Win32. Функция CreateProcess рассматривалась в главе 3; эта функция представляет собой функцию API-интерфейса Win32, с помощью которой из одного процесса может быть запущен другой. Для импортирования функции, находящейся в библиотеке DLL (а в библиотеке DLL находятся все функции API-интерфейса Win32), в программу на языке VB используется команда Declare Function. В разрабатываемой задаче требуется выполнить импорт нескольких подобных функций, поэтому в редакторе кода должно быть введено несколько операторов Declare Function. Введите код, показанный в листинге 20.13, в верхней части разрабатываемого модуля исходного кода.

Листинг 20.13. Код, обеспечивающий импорт необходимых функций API-интерфейса Win32

```
Private Type PROCESS_INFORMATION
  hProcess As Long
  hThread As Long
  dwProcessId As Long
  dwThreadId As Long
End Type

Private Type STARTUPINFO
  cb As Long
  lpReserved As String
  lpDesktop As String
  lpTitle As String
  dwX As Long
  dwY As Long
  dwXSize As Long
  dwYSize As Long
  dwXCountChars As Long
  dwYCountChars As Long
```

```
  dwFillAttribute As Long
  dwFlags As Long
  wShowWindow As Integer
  cbReserved2 As Integer
  lpReserved2 As Long
  hStdInput As Long
  hStdOutput As Long
  hStdError As Long
End Type

Private Declare Function CreateProcess Lib "kernel32" _
  Alias "CreateProcessA" _
  (ByVal lpApplicationName As String, _
  ByVal lpCommandLine As String, _
  lpProcessAttributes As Any, _
  lpThreadAttributes As Any, _
  ByVal bInheritHandles As Long, _
  ByVal dwCreationFlags As Long, _
  lpEnvironment As Any, _
  ByVal lpCurrentDirectory As String, _
  lpStartupInfo As STARTUPINFO, _
  lpProcessInformation As PROCESS_INFORMATION) As Long

Private Declare Function TerminateProcess Lib "kernel32" _
  (ByVal hProcess As Long, _
  ByVal uExitCode As Long) As Long

Private Declare Function WaitForSingleObject Lib "kernel32" _
  (ByVal hHandle As Long, ByVal dwMilliseconds As Long) As Long

Private Declare Function CloseHandle Lib "kernel32" (ByVal _
  hObject As Long) As Long

Private Declare Function GetLastError Lib "kernel32" () As Long

Private Const CREATE_DEFAULT_ERROR_MODE = &H4000000
Private Const WAIT_TIMEOUT = &H102&
Private Const INFINITE = -1&
```

В коде, приведенном в листинге 20.13, обеспечивается импорт функций `CreateProcess`, `TerminateProcess`, `WaitForSingleObject`, `CloseHandle` и `GetLastError` API-интерфейса Win32, наряду со структурами и константами, которые требуются для этих функций.

18. После импорта необходимых функций Win32 можно приступить к созданию самого метода `Execute`. Введите код, приведенный в листинге 20.14, в определение метода `CustomTask_Execute`.

Листинг 20.14. Код метода `Execute`

```
Dim pi As PROCESS_INFORMATION
    Dim si As STARTUPINFO
    Dim lRes As Long

    Dim bstrCmdLine As String
```

```
    ' Выполнить подстановку параметров командной строки вызова сценария
    bstrCmdLine = m_bstrScriptUtility
    bstrCmdLine = Replace(bstrCmdLine, "%server_instance%", _
      m_bstrServerInstance)
    bstrCmdLine = Replace(bstrCmdLine, "%auth_string%", _
      m_bstrAuthString)
    bstrCmdLine = Replace(bstrCmdLine, "%script%", _
       m_bstrScriptToExecute)
    bstrCmdLine = Replace(bstrCmdLine, "%output%", _
      m_bstrOutputFileName)

    ' Инициализировать структуру STARTUPINFO
    si.cb = Len(si)

    ' Осуществить запуск процесса
    Dim bstrNull As String

    lRes = CreateProcess(bstrNull, _
                         bstrCmdLine, _
                         ByVal 0&, _
                         ByVal 0&, _
                          0&, _
                         CREATE_DEFAULT_ERROR_MODE, _
                         ByVal 0&, _
                         bstrNull, _
                          si, _
                         pi)
If lRes <> 0 Then

    Dim lTimeout As Long
    If 0 = m_lTimeout Then
      lTimeout = INFINITE
    Else
      lTimeout = m_lTimeout * 1000
    End If

    ' Ожидать завершения процесса или остановить процесс по тайм-ауту
    lRes = WaitForSingleObject(pi.hProcess, lTimeout)

    ' Если выполнен останов процесса по тайм-ауту, записать в журнал
    ' сообщение
    If WAIT_TIMEOUT = lRes Then
      If Not pPackageLog Is Nothing Then
        pPackageLog.WriteTaskRecord 50001, Me.Description & ": _
            Script execution timed out"
       End If
      If m_bTerminateOnTimeout Then
        lRes = TerminateProcess(pi.hProcess, -1&)
        If 0 = lRes Then
          pPackageLog.WriteTaskRecord 50002,Me.Description& _
              ": Script termination failed"
        End If
       End If
    End If

    Call CloseHandle(pi.hThread)
```

```
    Call CloseHandle(pi.hProcess)

    pTaskResult = DTSTaskExecResult_Success

    Else
    Dim lLastError As Long

    ' Получить последнее сообщение об ошибке из операционной
    ' системы Windows
    lLastError = GetLastError()

    Dim oPkgEvent As DTS.PackageEvents
    Set oPkgEvent = pPackageEvents

    Dim bCancel As Boolean
    bCancel = True
    ' Вызвать обработчик ошибок OnError
    oPkgEvent.OnError Me.Description, lLastError, bstrCmdLine, _
        "CreateProcess failed", "", 0, "", bCancel
    If bCancel Then
         pTaskResult = DTSTaskExecResult_Failure
     Else
         pTaskResult = DTSTaskExecResult_Success
    End If

End If
```

19. В коде, приведенном в листинге 20.14, осуществляются действия, фактически обеспечивающие вызов на выполнение утилиты сценариев. Работа данного кода начинается с объявления структур `PROCESS_INFORMATION` и `STARTUPINFO`, которые требуются для функции `CreateProcess`. Структура `STARTUPINFO` предоставляет параметры, необходимые для создания процесса, а структура `PROCESS_INFORMATION` заполняется дескрипторами, а также идентификаторами процесса и потоков, относящимися к новому процессу, после создания процесса.

20. Затем в этом коде переопределяются значения нескольких параметров, применяемых по умолчанию в командной строке утилиты сценариев. Разрабатываемый компонент обеспечивает хранение основных частей командной строки утилиты сценариев в отдельных свойствах, которые затем используются для составления командной строки, вызываемой на выполнение, путем замены указанных параметров предусмотренными для них значениями при вызове метода `Execute`. Это позволяет легко настраивать конфигурацию параметров командной строки для утилиты исполнения сценариев с помощью глобальных переменных DTS, задач Dynamic Property, кода Automation и тому подобного. Например, предположим, что имеется следующая командная строка утилиты сценариев:

    ```
    OSQL %auth_string% -S%server_instance% -i%script% -o%output%
    ```

    При выполнении данного этапа метод `Execute` заменяет параметр `%auth_string%` значением переменной экземпляра `m_bstrAuthString`, параметр `%server_instance%` — значением переменной экземпляра `m_bstrServerInstance`, параметр `%script%` — значением переменной экземпляра `m_bstrScriptToExecute` и параметр `%output%` — значением переменной экземпляра `m_bstrOutputFileName`. Благодаря тому, что метод

Execute подготовлен для работы в таком режиме, появляется возможность размещать параметры в любых нужных местах в командной строке и легко модифицировать их значения с помощью глобальных переменных, задач Dynamic Property и подобных механизмов. При этом пользователь не обязан каждый раз переформировывать текст командной строки для утилиты сценариев из базового пакета (например, с помощью сценария ActiveX).

21. Затем в методе Execute вызывается функция CreateProcess для запуска утилиты сценариев. А после этого вызывается функция WaitForSingleObject для перехода в состояние ожидания завершения работы утилиты сценариев. Если свойство Timeout имеет ненулевое значение, то метод Execute передает значение тайм-аута в функцию WaitForSingleObject для того, чтобы можно было прекратить ожидание по тайм-ауту; в противном случае ожидание завершения работы утилиты сценариев в методе Execute продолжается неопределенно долгое время.
22. Если свойству TerminateOnTimeout присвоено значение true и функция WaitForSingleObject прекращает ожидание завершения работы утилиты сценариев по тайм-ауту, то в методе Execute вызывается функция TerminateProcess для останова работы этой утилиты.
23. Обратите внимание на то, что в объекте PackageLog предусмотрен вызов метода WriteTaskRecord. Объект PackageLog входит в состав объектной модели DTS и предоставляет средства ведения журнала в некотором пакете. В данном случае без дополнительных последствий происходит запись информации о ситуации, которая не должна вызвать аварийное завершение задачи и не требует немедленной передачи отчета пользователю.
24. Если вызов функции CreateProcess оканчивается неудачей, то функция возвращает 0. Это значение проверяется, и в случае неудачного завершения активизируется событие пакета OnError. Обратите внимание на то, что для ссылки на текущий объект задачи служит идентификатор Me.
25. После вызова функции CreateProcess и получения кода завершения, свидетельствующего об успехе или неудаче, из определяемой пользователем задачи возвращается соответствующий результат текущего этапа. Это позволяет службам DTS определить, успешно ли был выполнен этап прогона утилиты сценариев.
26. Завершив разработку кода задачи ExecuteSQLScript, необходимо установить новый компонент в программе DTS Designer, чтобы данный компонент можно было использовать в пакете. Для этого вначале следует откомпилировать компонент и создать файл библиотеки DLL с помощью команды меню File | Make.
27. Затем создайте новый пакет DTS, щелкнув правой кнопкой мыши на узле Data Transformation Services в программе Enterprise Manager и выбрав команду New Package. Выберите команду меню Task | Register Custom Task в программе DTS Designer. Щелкните на кнопке со знаком троеточия рядом с текстовым полем Task location и найдите файл библиотеки DLL для данной определяемой пользователем задачи.
28. Введите содержательное описание для создаваемого компонента в текстовом поле Task description. Программа DTS Designer всегда добавляет строку ": undefined" (не определено) к любому введенному в этом поле тексту при создании заданного по умолчанию описания для нового экземпляра объекта

задачи из пакета, поэтому следует ввести текст, позволяющий легко определить назначение компонента. Сам автор вводит для этой задачи описание "Execute SQL Script Task" (Задача выполнения сценария SQL).

29. Щелкните на кнопке OK, чтобы закрыть диалоговое окно Register Custom Task, и зарегистрируйте вновь разработанный компонент в программе DTS Designer. Вы должны обнаружить появление компонента в палитре задач.
30. Теперь вновь созданный компонент можно применить в пакете DTS, поэтому перейдите к осуществлению оставленного задания и перетащите компонент на лист проекта. Вы должны обнаружить, что в программе DTS Designer отображается диалоговое окно заданных по умолчанию свойств, которое формируется службами DTS от имени пользователя на основе данных о свойствах с атрибутом `public`, доступ к которым предоставляется рассматриваемым компонентом.
31. Для проведения с вновь созданной определяемой пользователем задачей можно загрузить пакеты `CustomTaskVBExample.DTS` и `CustomTaskVB_Timeout-Example.DTS` из подкаталога `CH20\CustomTaskVB` компакт-диска, прилагаемого к данной книге. В пакете `CustomTaskVBExample` выполняется простой сценарий с помощью утилиты `OSQL.EXE`, и для этого используется определяемая пользователем задача ExecuteSQLScript. В пакете `CustomTaskVB_Timeout-Example` выполняется строка запроса (а не сценарий) и ожидание завершения выполнения запроса прекращается по тайм-ауту, если запрос выполняется дольше, чем пять секунд. Кроме того, в последнем пакете вызов на выполнение утилиты сценариев происходит с помощью процессора команд `CMD.EXE`, а для передачи выходных данных в выходной файл служат средства перенаправления.

Создание новой определяемой пользователем задачи на основе примера задачи

Еще один способ создания определяемой пользователем задачи DTS состоит в доработке одного из примеров задач, которые входят в поставку программы SQL Server. В следующем упражнении описан порядок действий, связанных с доработкой определяемой пользователем задачи DTSSampleTask, которая включена в состав примеров кода DTS, поставляемых вместе с программой SQL Server. Код указанной задачи можно найти в подкаталоге `...\80\Tools\DevTools\Samples\dts\CustomTasks\DTSTask` инсталляционного каталога SQL Server. (Этот каталог применяется по умолчанию, но вам может потребоваться разархивировать примеры DTS, чтобы найти этот каталог.) В рассматриваемом примере задачи реализуются такие же существенно важные функциональные возможности, как и в задаче Execute Process: пользователь предоставляет командную строку для вызываемого на выполнение процесса, а задача вызывает функцию `CreateProcess` API-интерфейса Win32, чтобы выполнить указанное задание. При этом предусматривается возможность перехода в состояние завершения выполняемой программы и/или выхода по тайм-ауту и принудительного завершения, если это потребуется.

ПРИМЕЧАНИЕ. Рассматриваемая задача DTSSampleTask представляет собой компонент COM, написанный на языке C++. Для внесения изменений в код этой задачи необходимо откорректировать определение на языке IDL (Interface Definition Language — язык определения интерфейса), а также код на языке C++. Если читатель недостаточно хорошо владеет обоими указанными языками, то ему лучше пропустить это упражнение.

Упражнение

В приведенных ниже указаниях принято предположение, что используется версия VC6, должна также оказаться применимой версия VC7, хотя содержание указанных шагов может немного измениться.

Упражнение 20.3. Создание новой определяемой пользователем задачи на основе примера задачи

1. Загрузите проект `dtstask` (из подкаталога `DTSTask`, указанного выше) в интегрированную среду разработки Visual C++. Рекомендуется скопировать содержимое указанного подкаталога в другое место, чтобы исключить необходимость внесения изменений в оригинальный код примера задачи.
2. Начнем с описания файла `dtstask.idl`. Этот файл содержит код IDL, который описывает интерфейс COM компонента определяемой пользователем задачи. Файл IDL используется компилятором MIDL для формирования других файлов, которые в конечном итоге компилируются наряду с другими файлами исходного кода компонента для создания библиотеки DLL определяемой пользователем задачи.
3. Найдите спецификации атрибута свойства для свойства `ProcessCommandLine` примерно в середине файла `dtstask.idl`. Удалите четыре строки, которые определяют свойство `ProcessCommandLine`.
4. Вставьте строки, приведенные в листинге 20.15, вместо удаленных строк и перенумеруйте соответствующим образом свойства, которые следуют за приведенными в этом листинге определениями свойств.

Листинг 20.15. Определения дополнительных свойств на языке IDL

```
[id(15), propget, helpstring("Command line of script _
    execution utility (e.g., OSQL.EXE). Specify %script% for _
    script file name placeholder and %output% for output _
    file name.")]
HRESULT ScriptExecutionUtility([out, retval] BSTR *pRetVal);
[id(15), propput]
HRESULT ScriptExecutionUtility([in] BSTR NewValue);

[id(16), propget, helpstring("UNC File name of the script _
    to execute")]
HRESULT ScriptToExecute([out, retval] BSTR *pRetVal);
[id(16), propput]
```

```
HRESULT ScriptToExecute([in] BSTR NewValue);

[id(17), propget, helpstring("UNC File name of the output _
    file to create")]
HRESULT OutputFileToCreate([out, retval] BSTR *pRetVal);
[id(17), propput]
HRESULT OutputFileToCreate([in] BSTR NewValue);
```

В листинге 20.15 определены три новых свойства: `ScriptExecutionUtility`, `ScriptToExecute` и `OutputFileToCreate`. Обязательно перенумеруйте свойства, которые следуют за указанными свойствами, для того, чтобы в конечном итоге каждое свойство имело уникальный идентификатор.

5. Откорректируйте строку справки в нижней части файла `dtstask.idl`, относящуюся к определяемой пользователем задачи, которая в настоящее время содержит текст "`DTS Sample Task: Create process`" и вместо этого текста введите "`DTS ExecuteScript Task`" (Задача ExecuteScript служб DTS).
6. Следующим файлом, в который требуется внести изменения, является файл заголовка `task.h`. Этот файл описывает интерфейс к классу, который реализует объект определяемой пользователем задачи. Удалите из этого файла все ссылки на переменную экземпляра `m_bstrCommandLine` и свойство `ProcessCommandLine`. Обязательно удалите все объявления методов `get` и `put` в определении `IDTSSampleTask`, относящемся к свойству `ProcessCommandLine`.
7. Введите следующие строки в конструктор `CTask`:

```
m_bstrScriptExecutionUtility = SysAllocString(L"");
m_bstrScriptToExecute = SysAllocString(L"");
m_bstrOutputFileToCreate = SysAllocString(L"");
```

8. Введите следующие строки в деструктор `CTask`:

```
if (m_bstrScriptExecutionUtility)
  SysFreeString(m_bstrScriptExecutionUtility);
if (m_bstrScriptToExecute)
  SysFreeString(m_bstrScriptToExecute);
if (m_bstrOutputFileToCreate)
  SysFreeString(m_bstrOutputFileToCreate);
```

9. Введите объявления методов, приведенные в листинге 20.16, в определение `IDTSSampleTask`.

Листинг 20.16. Объявления методов

```
STDMETHOD(get_ScriptExecutionUtility)(
    /* [retval][out] */ BSTR  *pRetVal);

STDMETHOD(put_ScriptExecutionUtility)(
    /* [in] */ BSTR NewValue);

STDMETHOD(get_ScriptToExecute)(
    /* [retval][out] */ BSTR  *pRetVal);

STDMETHOD(put_ScriptToExecute)(
    /* [in] */ BSTR NewValue);

STDMETHOD(get_OutputFileToCreate)(
```

```
    /* [retval][out] */ BSTR  *pRetVal);

STDMETHOD(put_OutputFileToCreate)(
    /* [in] */ BSTR NewValue);
```

10. Откорректируйте объявления переменных экземпляра BSTR с атрибутом private в определении IDTSSampleTask, чтобы они выглядели следующим образом:

    ```
    BSTR m_bstrName, m_bstrDescription, m_bstrScriptExecutionUtility,
    ↳ m_bstrScriptToExecute, m_bstrOutputFileToCreate;
    ```

    Тем самым определяются переменные экземпляра с атрибутом private, которые будут использоваться для кэширования значений новых свойств.

11. Последним файлом, в который должны быть внесены изменения, является файл task.cpp. Этот файл представляет собой модуль, обеспечивающий реализацию определяемой пользователем задачи. Найдите код методов get и put, относящийся к определению свойства ProcessCommandLine, и удалите этот код. Вместо удаленного кода введите код, приведенный в листинге 20.17.

Листинг 20.17. Новый код методов get и put

```
STDMETHODIMP CTask::get_ScriptExecutionUtility(
      /* [retval][out] */ BSTR  *pRetVal)
{
  if (!pRetVal)
    return E_POINTER;
  *pRetVal = SysAllocString(m_bstrScriptExecutionUtility);
  if (!*pRetVal)
return E_OUTOFMEMORY;
  return NOERROR;
}

STDMETHODIMP CTask::put_ScriptExecutionUtility(
       /* [in] */ BSTR NewValue)
{
  if (m_bstrScriptExecutionUtility)
    SysFreeString(m_bstrScriptExecutionUtility);
  m_bstrScriptExecutionUtility = SysAllocString(NewValue);
  if (!m_bstrScriptExecutionUtility)
     return E_OUTOFMEMORY;
  return NOERROR;
}

STDMETHODIMP CTask::get_ScriptToExecute(
      /* [retval][out] */ BSTR  *pRetVal)
{
  if (!pRetVal)
     return E_POINTER;
  *pRetVal = SysAllocString(m_bstrScriptToExecute);
  if (!*pRetVal)
     return E_OUTOFMEMORY;
  return NOERROR;
}

STDMETHODIMP CTask::put_ScriptToExecute(
       /* [in] */ BSTR NewValue)
```

```
{
  if (m_bstrScriptToExecute)
    SysFreeString(m_bstrScriptToExecute);
  m_bstrScriptToExecute = SysAllocString(NewValue);
  if (!m_bstrScriptToExecute)
     return E_OUTOFMEMORY;
  return NOERROR;
}

STDMETHODIMP CTask::get_OutputFileToCreate(
      /* [retval][out] */ BSTR  *pRetVal)
{
  if (!pRetVal)
     return E_POINTER;
*pRetVal = SysAllocString(m_bstrOutputFileToCreate);
  if (!*pRetVal)
     return E_OUTOFMEMORY;
  return NOERROR;
}

STDMETHODIMP CTask::put_OutputFileToCreate(
      /* [in] */ BSTR NewValue)
{
  if (m_bstrOutputFileToCreate)
    SysFreeString(m_bstrOutputFileToCreate);
  m_bstrOutputFileToCreate = SysAllocString(NewValue);
  if (!m_bstrOutputFileToCreate)
     return E_OUTOFMEMORY;
  return NOERROR;
}
```

Код методов, приведенный в листинге 20.17, предоставляет вспомогательные средства, необходимые для задания и получения значений новых свойств.

12. Последнее действие, которое необходимо выполнить, чтобы приспособить рассматриваемую задачу для использования в собственных целях, состоит в том, чтобы откорректировать метод `Execute`, предназначенный для вызова утилиты исполнения сценария. Автор просто представит без объяснений требуемый для этого код, с которым можно ознакомиться самостоятельно, чтобы узнать, как он работает. Основное различие между этим кодом и оригинальным кодом метода `Execute` состоит в том, что в текущем варианте предусмотрена замена заранее заданных строковых параметров в командной строке утилиты сценариев значениями переменных экземпляра `m_bstrScriptToExecute` и `m_bstrOutputFileToCreate`, если в этом есть необходимость. Как и в примере `CustomTaskVB`, такая замена расширяет возможности настройки командной строки утилиты сценариев и позволяет задавать имя выполняемого сценария и имя создаваемого выходного файла. Для этого могут использоваться глобальные переменные, задачи Dynamic Properties и аналогичные средства пакетов. Кроме того, в варианте, приведенном в данном упражнении, исправлена ошибка, допущенная в оригинальном коде примера задачи, который входит в поставку программы SQL Server. Из-за этой ошибки терялась память, занимаемая двумя дескрипторами (дескриптором потока и дескриптором процесса), после каждого вызова метода `Execute`. Замените оригинальный код метода `Execute` кодом, приведенным в листинге 20.18.

Листинг 20.18. Код метода Execute, предназначенный для замены оригинального кода

```
STDMETHODIMP CTask::Execute(
      /* [in] */ IDispatch  *pPackage,
      /* [in] */ IDispatch  *pPackageEvents,
      /* [in] */ IDispatch  *pPackageLog,
      /* [out][in] */ LONG  *pTaskResult)
{
  //*************************************************************
  // Примечание. В данном примере метод SetErrorInfo не определен
  // должным образом. Пользователь должен реализовать метод
  // SetErrorInfo и правильно вызывать при возникновении ошибок
  //*************************************************************
  USES_CONVERSION;

  HRESULT hr = NOERROR;

  // Проверить и инициализировать указатели на возвращаемое значение и
  // выходной параметр
  *pTaskResult = DTSTaskExecResult_Failure; // Предполагается наличие
                                            // ошибки

  //*** Начало кода метода ***

  PROCESS_INFORMATION     procInfo;
  STARTUPINFO    startupInfo;
  DWORD   dwTimeout;
  LPTSTR   szScriptExecutionUtility;
  LPTSTR   szScriptToExecute;
  LPTSTR   szOutputFileToCreate;
  TCHAR szCmd[0x1000];
  TCHAR szCmd2[0x1000];
  const TCHAR *SCRIPT = _T("%script%");
  const TCHAR *OUTPUT = _T("%output%");

  memset(&startupInfo, 0, sizeof(startupInfo));
  memset(&procInfo, 0, sizeof(procInfo));
  startupInfo.cb = sizeof(STARTUPINFO);

  szScriptExecutionUtility = OLE2T(m_bstrScriptExecutionUtility);
  szScriptToExecute = OLE2T(m_bstrScriptToExecute);
  szOutputFileToCreate = OLE2T(m_bstrOutputFileToCreate);

  // Выполнить подстановку параметров SCRIPT и OUTPUT
  TCHAR *p=_tcsstr(szScriptExecutionUtility,SCRIPT);
  TCHAR *q=_tcsstr(szScriptExecutionUtility,OUTPUT);
  if ((p) || (q)) {
    if ((q) && (p) && (p<q)) {  // Получены значения параметров SCRIPT
                                // и OUTPUT; на первом месте находится
                                // параметр SCRIPT
      // Выполнить подстановку параметра SCRIPT
       _tcsncpy(szCmd,szScriptExecutionUtility,
       p-szScriptExecutionUtility);
       szCmd[p-szScriptExecutionUtility]=_T('\0');
      _tcscat(szCmd,szScriptToExecute);
      _tcscat(szCmd,p+_tcslen(SCRIPT));
```

```
      // Выполнить подстановку параметра OUTPUT
      q=_tcsstr(szCmd,OUTPUT);
      _tcsncpy(szCmd2,szCmd,q-szCmd);
      szCmd2[q-szCmd]=_T('\0');
      _tcscat(szCmd2,szOutputFileToCreate);
      _tcscat(szCmd2,q+_tcslen(OUTPUT));

      _tcscpy(szCmd,szCmd2);
    }
    else if ((q) && (p) && (p>q)) {  // Получены значения параметров
                                     // SCRIPT и OUTPUT; на первом
                                     // месте находится параметр
OUTPUT
      // Выполнить подстановку параметра OUTPUT
      _tcsncpy(szCmd,szScriptExecutionUtility,
          q-szScriptExecutionUtility);
      szCmd[q-szScriptExecutionUtility]=_T('\0');
      _tcscat(szCmd,szOutputFileToCreate);
      _tcscat(szCmd,q+_tcslen(OUTPUT));

      // Выполнить подстановку параметра SCRIPT
      p=_tcsstr(szCmd,OUTPUT);
      _tcsncpy(szCmd2,szCmd,p-szCmd);
      szCmd2[p-szCmd]=_T('\0');
      _tcscat(szCmd2,szScriptToExecute);
      _tcscat(szCmd2,p+_tcslen(SCRIPT));

      _tcscpy(szCmd,szCmd2);
    }
    else if ((q) && (!p)) { // Получено значение параметра OUTPUT, но
                            // не SCRIPT
      // Выполнить подстановку параметра OUTPUT
      _tcsncpy(szCmd,szScriptExecutionUtility,q-
↳szScriptExecutionUtility);
      szCmd[q-szScriptExecutionUtility]=_T('\0');
      _tcscat(szCmd,szOutputFileToCreate);
      _tcscat(szCmd,q+_tcslen(OUTPUT));

    }
    else {  // Получено значение параметра SCRIPT, но не OUTPUT
      // Выполнить подстановку параметра SCRIPT
      _tcsncpy(szCmd,szScriptExecutionUtility,
      p-szScriptExecutionUtility);
szCmd[p-szScriptExecutionUtility]=_T('\0');
       _tcscat(szCmd,szScriptToExecute);
       _tcscat(szCmd,p+_tcslen(SCRIPT));

    }

  }
  else
    _tcscpy(szCmd,szScriptExecutionUtility);

  // Создать процесс с помощью вызова из командной строки
  if (!CreateProcess(NULL, szCmd, NULL, NULL, FALSE,
      CREATE_DEFAULT_ERROR_MODE, NULL, NULL, &startupInfo,
```

```
      &procInfo))
     return E_UNEXPECTED;

  if (m_lTimeout == 0)  // Тайм-аут не задан
    dwTimeout = INFINITE;
  else
    dwTimeout = 1000 * m_lTimeout;

  if (WAIT_TIMEOUT != WaitForSingleObject
       (procInfo.hProcess, dwTimeout))
  {
     // Процесс завершен
    DWORD dwExitCode;
    if (!GetExitCodeProcess(procInfo.hProcess, &dwExitCode))
      return E_UNEXPECTED;

     if (dwExitCode == (DWORD)m_lSuccessReturnCode)  // Совпадение кода
                                                     // вызова с желаемым значением
                                                     // свидетельствует об успехе
       *pTaskResult = DTSTaskExecResult_Success; // Предполагается
                                                 // наличие ошибки
  }
  else
  {
     if (m_bTerminateProcessAfterTimeout)  // Завершить процесс, если в
                                // этом состоит требование пользователя.
                                // По умолчанию подразумевается обратное
      TerminateProcess(procInfo.hProcess, (UINT)-1);
     if (m_bFailPackageOnTimeout) {  // В этом методе ::Execute по
                                // истечении тайм-аута формируется
                                // сообщение об ошибке, а не просто
                                // происходит отказ, если в этом состоит
                                // требование пользователя
       CloseHandle(procInfo.hProcess);
      CloseHandle(procInfo.hThread);
return HRESULT_FROM_WIN32(ERROR_TIMEOUT);  // Такая ситуация напоминает
                                           // отмену
     }
  }

  CloseHandle(procInfo.hProcess);
  CloseHandle(procInfo.hThread);
  return hr;
}
```

13. Теперь можно приступить к компиляции разрабатываемого компонента и созданию библиотеки DLL, а также к регистрации этой библиотеки в программе DTS Designer. Нажмите клавишу <F7>, чтобы сформировать проект. После создания библиотеки `dtstask.DLL` зарегистрируйте ее в программе DTS Designer с помощью команды меню Task | Register Custom Task, руководствуясь такими же указаниями, которые касались регистрации библиотеки DLL, относящейся к описанной выше определяемой пользователем задаче, `ExecuteSQLScript.DLL` (по умолчанию библиотека `dtstask.DLL` создается в каталоге `ReleaseUMinDependency`). Как и в предыдущем упражнении, можно

ввести описание для определяемой пользователем задачи при ее регистрации. Сам автор в качестве описания задачи использует текст "Execute Script Task" (Задача исполнения сценария).

14. После регистрации нового компонента в программе DTS Designer можно перетащить этот компонент на лист проекта и выполнить настройку конфигурации для использования в пакете. Как и для задачи Execute SQL Script Task, созданной в предыдущем упражнении, можно задать строки определения параметров в командной строке утилиты сценариев, чтобы указать, в каком месте необходимо вставить значения свойств `ScriptToExecute` и `OutputFileToCreate`. Например, свойству `ScriptExecutionUtility` может быть присвоено примерно такое значение:

```
OSQL -S. -E -i%script% -o%output%
```

После запуска задачи на выполнение и вызова метода `Execute` этой задачи параметр `%script%` заменяется текущим значением свойства `ScriptToExecute`, а параметр `%output%` — текущим значением свойства `OutputFileToCreate`. Поскольку значения указанным свойствам могут присваиваться с помощью глобальных переменных, задач Dynamic Properties и аналогичных механизмов, такая возможность позволяет динамически заменять части командной строки утилиты сценариев без необходимости полностью переформировывать командную строку с помощью кода сценариев или чего-то подобного.

15. Дополнительные эксперименты с определяемой пользователем задачей ExecuteScript можно провести, загружая пакет `CustomTaskExample.DTS` из подкаталога `CH20\CustomTask` компакт-диска, прилагаемого к данной книге, в программу DTS Designer, вызывая этот пакет на выполнение и внося различные изменения.

Отладка компонентов определяемой пользователем задачи

Процесс отладки компонента определяемой пользователем задачи во многом аналогичен отладке библиотеки DLL любого другого типа (автор еще раз рекомендует создавать компоненты определяемой пользователем задачи в виде библиотек DLL, чтобы можно было использовать эти компоненты в программе DTS Designer), но можно привести ряд рекомендаций, позволяющих упростить отладку, если когда-либо потребуется заниматься этой работой. Автору встречались публикации в телеконференциях, где предлагалось применять целый ряд сложных операций (зачастую ненужных) для отладки определяемой пользователем задачи DTS. А ниже приведены некоторые общие указания, позволяющие значительно упростить такую работу, если когда-либо придется ею заниматься.

1. Следует учитывать, что непосредственная отладка библиотеки DLL не производится, поскольку отлаживается только базовый исполняемый файл, в котором вызывается эта библиотека. Автору приходилось сталкиваться с тем, как специалисты советуют неопытным пользователям попытаться провести отладку непосредственно из среды Visual Basic или Visual C++, определив в качестве "базового приложения" программу Enterprise Manager, по крайней

мере в той ситуации, когда применяется отладчик интегрированной среды разработки. Безусловно, такая организация отладки возможна, но является слишком сложной. Есть более простой способ – достаточно, как обычно, запустить базовый процесс, а затем подключить отладчик к этому процессу. Для программы Enterprise Manager таковым является процесс исполнения программы `MMC.EXE` – приложения, под управлением которого функционирует программа SEM. Для утилиты `dtsrun` таковой программой, безусловно, является `dtsrun.exe`. А для автономного приложения, которое обращается к объектной модели DTS с помощью средств COM Automation, отладчик может быть подключен к самому приложению. Возможность подключения к работающему процессу предоставляют и отладчик VC++, и программа WinDbg – автономный отладчик, используемый во всей этой книге.

2. По возможности следует использовать отладочную версию рассматриваемого компонента. Отладка с применением оптимизированного кода может оказаться сложной или даже невозможной, поскольку оптимизирующие компиляторы способны переупорядочивать и даже удалять машинный код, который соответствует отдельным строкам исходного кода. Определяемая пользователем задача, только что созданная в настоящей главе, поддерживает две разные конфигурации отладочных версий. Доступ к этим конфигурациям можно получить из интегрированной среды разработки VC6 с помощью команды меню **Build | Set Active Configuration**.
3. Следует обязательно подготавливать отладочную символическую информацию для разрабатываемого компонента. Такая информация создается по умолчанию для отладочных версий в среде VC++, а в проектах VB можно разрешить подготовку этой информации, отметив флажок **Create Symbolic Debug Info** в диалоговом окне **Project Properties | Compile**.
4. Прежде чем подключиться к базовому приложению с помощью отладчика, следует правильно задать в отладчике пути к отладочной информации и пути к файлам исходного кода. При этом, вероятно, потребуется добавить к обозначению пути к отладочной информации имя каталога, в котором находится файл PDB, сформированный в среде VB или VC++, а к обозначению пути к файлам исходного кода в отладчике – имя каталога, в котором расположен файл исходного кода рассматриваемого компонента.
5. После подключения к базовому приложению необходимо провести проверку того, загружена ли библиотека DLL рассматриваемого компонента и найден ли отладчиком файл отладочной информации для этого компонента. В отладчике WinDbg такую проверку можно провести с помощью команды `lm`, которая выводит список загруженных в настоящее время модулей с указанием для каждого модуля пути к файлам с отладочной информацией, если файлы с отладочной информацией найдены и загружены. Если же компонент еще не загружен, то следует сделать все необходимое, чтобы вынудить базовое приложение загрузить этот компонент, поскольку обращаться к отладочной информации библиотеки DLL (например, задавать точки останова) невозможно до тех пор, пока эта библиотека не будет

загружена. Если вы обнаружите, что библиотека DLL компонента загружена, а доступ к отладочной информации компонента еще не получен, добейтесь того, чтобы имя каталога с файлом отладочной информации вошло в обозначение пути к отладочной информации отладчика, затем передайте отладчику команду на осуществление еще одной попытки загрузить отладочную информацию для рассматриваемого компонента (невозможно проводить полную отладку компонента до тех пор, пока успешно не загружена его отладочная информация). В различных отладчиках для выполнения загрузки отладочной информации используются разные команды, а в отладчике WinDbg для принудительной перезагрузки отладочной информации применяется команда `.reload -f`.

6. Теперь вы должны получить возможность обращаться к методам рассматриваемого компонента с помощью обычных синтаксических конструкций `Class::method`, даже если компонент создан на языке VB. Например, если класс компонента на языке VB носит имя `clsExecuteSQLScript`, то можно установить точку останова на методе `Execute` этого класса, сославшись на символическое имя `clsExecuteSQLScript::CustomTask_Execute`.

При условии, что обозначения путей к отладочной информации и файлам исходного кода заданы правильно, вы должны получить возможность отлаживать рассматриваемый компонент наряду с любой другой библиотекой DLL. Рассмотрим простое упражнение, в котором используется автономный отладчик WinDbg для отладки определяемой пользователем задачи, которая была описана выше в этой главе.

Упражнение

Чтобы разрешить создание отладочной информации в среде Visual Basic, необходимо выбрать опцию Create Symbolic Debug Info на вкладке Compile диалогового окна Project Properties.

Упражнение 20.4. Отладка определяемой пользователем задачи

1. Откомпилируйте определяемую пользователем задачу в виде файла библиотеки DLL и убедитесь в том, что создана символическая отладочная информация, в результате чего сформирован файл PDB.
2. Запустите программу DTS Designer и установите в ней определяемую пользователем задачу, если это еще не сделано.
3. Запустите отладчик WinDbg и добавьте к заданным в нем обозначениям путей к отладочной информации (с помощью меню File) имя каталога, содержащего файл PDB для рассматриваемой библиотеки DLL, и имя каталога, содержащего исходный код компонента.
4. Из отладчика WinDbg подключитесь к базовому приложению программы Enterprise Manager — `MMC.EXE`. (Нажмите клавишу <F6>, чтобы ознакомиться со списком работающих процессов.)

5. Установите точку останова на методе `Execute` класса определяемой пользователем задачи. Например, если класс имеет имя `clsExecuteSQLScript`, а метод `Execute` — имя `CustomTask_Execute`, то можно ввести следующую команду в отладчике WinDbg, чтобы установить указанную точку останова:
   ```
   bp clsExecuteSQLScript::CustomTask_Execute
   ```
   Эта точка останова вызовет останов в ходе выполнения задачи.
6. Введите `g` и нажмите клавишу <Enter>, чтобы разрешить приложению MMC продолжить работу.
7. Снова переключитесь в программу DTS Designer и перетащите экземпляр определяемой пользователем задачи на лист проекта.
8. Щелкните правой кнопкой мыши на рассматриваемом объекте определяемой пользователем задачи и выберите команду Execute. В результате выполнения этой команды должна активизироваться точка останова в отладчике WinDbg и прекратиться работа. (Вам придется воспользоваться комбинацией клавиш <Alt+Tab>, чтобы перейти к отладчику WinDbg, поскольку создастся впечатление, что программа DTS Designer зависла.)
9. Теперь вы должны обнаружить, что в отладчик WinDbg загружен модуль исходного кода, который содержит метод `Execute` определяемой пользователем задачи, и должны получить возможность осуществлять пошаговое выполнение кода этого модуля с помощью нажатия клавиши <F10>. По мере пошагового выполнения кода вы сможете устанавливать просматриваемые выражения, проверять стек вызовов и регистры, контролировать значения локальных переменных и осуществлять другие действия. Такая возможность становится чрезвычайно полезной при выявлении не поддающихся анализу программных ошибок в определяемых пользователем задачах DTS.
10. После того как вы будете готовы предоставить программе DTS Designer возможность продолжить выполнение, введите `g` и нажмите <Enter> в командном окне отладчика WinDbg. После отключения от приложения `MMC.EXE` вы, скорее всего, обнаружите, что это приложение полностью исчезло из состава работающих процессов. Если вы не работаете в версии Windows XP или более поздней версии, то отключение от процесса, подключение к которому происходило "насильственным путем" ("насильственное" подключение применяется по умолчанию и является необходимым для задания точек останова), автоматически приводит к завершению процесса, в котором проводилась отладка.

Настройка испытательного приложения

Еще одним методом отладки определяемых пользователем задач является разработка простого приложения VB, которое создает программным путем пакет, содержащий определяемую пользователем задачу (или открывает пакет, содержащий ссылку на такую задачу), и вызывает пакет на выполнение. В таком случае существует возможность ввести в приложение код, который затрагивает определенные части пакета, вызывает на выполнение указанные этапы, для которых необходимо провести проверку, и т.д. Само испытательное приложение может подвергаться пошаговому выполнению под управлением отладчика, что позволяет проверять в интерактивном режиме значения свойств пакета, применять режим пошагового выполнения кода определяемой пользователем задачи и т.д. Благодаря этому появ-

ляется возможность осуществлять многие действия, которые обычно выполнимы только при использовании автономного отладчика и подключения к приложению MMC или `dtsrun` (хотя при этом не требуется даже выходить за пределы среды VB).

Управление службами DTS с помощью средств Automation

По мнению автора, истинная мощь служб DTS состоит в том, что возможно организовать эксплуатацию этих служб программным путем. Программа DTS Designer настолько проста в эксплуатации, что практически любой пользователь способен успешно и продуктивно выполнять несложные преобразования данных. Но для выполнения более сложных преобразований данных часто требуется непосредственно обращаться к объектной модели DTS из определяемых пользователем программ. Для этого требуется осуществлять настройку и сохранять контроль над тем, что и когда происходит, с помощью определяемого пользователем прикладного кода, как и при эксплуатации приложений SQL Server других типов.

Подробное описание объектной модели DTS выходит за рамки данной книги. На эту тему уже написано несколько книг, поэтому автор не имеет ни времени, ни места, чтобы повторно приводить такие сведения в данной главе. Как было сказано во вступительной части этой главы, автор не ставил перед собой цель рассмотреть все особенности и нюансы технологии DTS программы SQL Server. Автор намеревался лишь объяснить, какие функциональные возможности предоставляют средства DTS и как организована работа этих средств с точки зрения архитектуры программного обеспечения, но не имел возможности сделать это описание настолько подробным, насколько ему хотелось бы. Но несмотря на сказанное выше, следует отметить, что ни одно описание служб DTS не будет полным, если в нем хотя бы вскользь не затрагиваются возможности управления технологией DTS программным путем, поэтому данная тема кратко рассматривается в следующем разделе.

В завершении данной главы будет представлено несколько приложений DTS, которые показывают, как обращаться к объектной модели DTS программным путем в целях обеспечения управления пакетами DTS и потоком данных из автономного приложения. Такое описание является удобным способом завершения всего изложения тематики DTS, поскольку позволяет проиллюстрировать с помощью действующего кода многие понятия, приведенные выше.

Простое приложение служб DTS, действующее на основе средств Automation

Начнем с изучения исходного кода первого приложения, показанного в листинге 20.19. (Полный исходный код этого приложения можно найти в подкаталоге `CH20\Automation` компакт-диска, прилагаемого к данной книге.)

Листинг 20.19. Исходный код первого приложения

```
Dim g_oPkg As DTS.Package2
Dim g_bInited

Function TaskName(Descrip As String) As String
  Dim oTask As DTS.Task
  TaskName = "NotFound"
  For Each oTask In g_oPkg.Tasks
    If oTask.Description = Descrip Then
      TaskName = oTask.Name
      Exit For
    End If
  Next
End Function
Private Sub Command1_Click()
  Dim oPkg As New DTS.Package2
  If (g_bInited) Then
    g_oPkg.UnInitialize
    Set g_oPkg = Nothing
    g_bInited = False
  End If
  Set g_oPkg = oPkg
  oPkg.LoadFromStorageFile App.Path & "\" & Text1.Text, ""

  lbGlobals.Clear
  Dim oGlobal As DTS.GlobalVariable2
  For Each oGlobal In oPkg.GlobalVariables
    lbGlobals.AddItem (oGlobal.Name & "=" & oGlobal.Value)
  Next

  lbSteps.Clear
  Dim oStep As DTS.Step2
  For Each oStep In oPkg.Steps
    lbSteps.AddItem (oStep.Name)
  Next

  lbTasks.Clear
  Dim oTask As DTS.Task
  For Each oTask In oPkg.Tasks
    lbTasks.AddItem (oTask.Description)
  Next

  g_bInited = True

End Sub

Private Sub Command2_Click()
  If (g_bInited) And (Len(Text2.Text) <> 0) Then
    g_oPkg.SaveToStorageFile App.Path & "\" & Text2.Text
  End If
End Sub

Private Sub Command3_Click()
  If (g_bInited) And (lbSteps.ListIndex <> -1) Then
    Dim oStep As DTS.Step2
```

```
    Set oStep = g_oPkg.Steps(lbSteps.List(lbSteps.ListIndex))
    oStep.Execute
  End If
End Sub

Private Sub Form_Load()
  g_bInited = False
End Sub
```

В приложении, приведенном в листинге 20.19, выполняется несколько интересных действий. Работа приложения начинается с предоставления пользователю возможности загрузить в память пакет DTS, который хранится на диске в структурированном формате COM. Пользователю разрешается загрузить любой выбранный им пакет; автор для начала специально предусмотрел пример такого пакета. После загрузки пакета приложение VB перебирает в цикле и вносит в списки глобальные переменные, этапы и задачи пакета, а затем выводит полученную информацию в элементах управления `ListBox` формы. Каждый из полученных списков хранится в виде коллекции, поэтому для обработки списков достаточно воспользоваться простым циклом `For Each`.

Приложение позволяет также выбрать отдельный этап в поле со списком **Step** и выполнить этот этап. При этом осуществляется поиск объекта `Step` по его имени, затем просто вызывается метод `Execute` этого объекта.

Кроме того, пакет можно сохранить в структурированном файле COM. Пользователь имеет возможности сохранить пакет в том же файле, из которого пакет был загружен, или выбрать другой файл.

Как показывает рассматриваемый листинг, основной прием, обеспечивающий возможность манипулирования пакетом DTS с помощью кода Automation, состоит в создании объекта `Package` или `Package2`. После создания такого объекта все остальные действия сводятся к получению доступа к свойствам, методам и событиям этого объекта.

Приложение DTSPkgGuru

Последнее приложение, которое будет рассматриваться в данной главе, немного сложнее по сравнению с большинством других описанных приложений. Прежде всего, это приложение разработано с использованием управляемого кода, точнее, кода на языке C#. Автор создал данное приложение с помощью управляемого кода, чтобы показать, как можно обеспечить автоматизацию объектной модели DTS из управляемого кода. При этом значительных отличий от неуправляемого кода не наблюдается, но приходится учитывать некоторые нюансы и особенности, о которых следует знать.

Рассматриваемое приложение имеет имя DTSPkgGuru и предназначено для использования в качестве средства редактирования пакетов за пределами программы DTS Designer. В частности, назначение этого приложения состоит в том, чтобы предоставить пользователю возможность автоматизировать внесение ряда изменений в пакет без потери той компоновки пакета, которую он имел на листе проектирования.

Необходимость внести ряд изменений в пакет с помощью кода Automation возникает достаточно часто. Такой подход к модификации пакета может оказаться намного более быстродействующим и точным по сравнению с организацией работы, в которой изменения вносятся вручную в программе DTS Designer. Например, если есть необходимость изменить продолжительность тайм-аута для сотни задач Execute Process (а разработчик не проявил достаточную предусмотрительность, чтобы обеспечить присваивание значения тайм-аута с помощью задачи Dynamic Properties), то гораздо быстрее было бы открыть пакет с помощью кода Automation и обработать в цикле объекты `Task` пакета, внося при этом необходимые изменения. А после завершения этой работы достаточно вызвать один из методов `Save` объекта `DTS.Package` для повторного сохранения уже модифицированного пакета.

Но с описанным подходом связано одно неприятное ограничение, состоящее в том, что после сохранения пакета с помощью кода Automation информация о компоновке пакета теряется. Даже если не происходит добавление каких-либо объектов и модифицируются свойства только существующих объектов, сохранение пакета с помощью кода Automation приводит к переопределению компоновки листа проекта. После повторного открытия модифицированного пакета в программе DTS Designer, скорее всего, будет обнаружена компоновка, весьма отличная от той, на разработку которой могли быть потрачены многие часы. Единственным средством, с помощью которого можно модифицировать пакет и сохранить его, не теряя компоновку на листе проекта, является сама программа DTS Designer. Поэтому все усилия, потраченные разработчиком на создание визуальной компоновки в том виде, какой ему требовался, перечеркиваются после сохранения пакета с помощью кода Automation.

Автор пытался обойти это ограничение, используя самые различные методы (включая передачу параметра `pVarPersistStgOfHost` в методы `Save` объекта `DTS.Package`), но не добился успеха. Насколько известно автору, не существует способа сохранить пакет с помощью кода Automation без потери компоновки пакета на листе проекта.

Поэтому для устранения указанного ограничения автор написал приложение DTSPkgGuru. Это приложение открывает пакет с помощью кода Automation, позволяет пользователю составить очередь из изменений, вносимых в пакет, затем отправить эти изменения в программу DTS Designer. Поскольку не существует документированных способов автоматизации программы DTS Designer с помощью стандартных средств COM Automation, приложение DTSPkgGuru управляет программой DTS Designer, передавая в нее последовательности кодов комбинаций клавиш для внесения в пакет заданных изменений.

Приложение DTSPkgGuru позволяет также сохранить набор изменений как текстовый файл с разделителями в виде символов табуляции и загрузить этот файл позднее. Две указанные особенности рассматриваемого приложения позволяют легко выполнять операции глобального поиска и замены или вносить в пакет другие сложные изменения с помощью любого выбранного пользователем текстового редактора, затем загружать текстовый файл в приложение DTSPkgGuru и применять эти изменения к пакету в программе DTS Designer.

Прежде чем пользователь получит возможность автоматизировать внесение изменений в пакет с помощью приложения DTSPkgGuru, он должен выполнить два описанных ниже основных требования.

1. Сохранить рассматриваемый пакет в виде структурированного файла хранения COM. Скорее всего, автор однажды предусмотрит возможность загружать пакеты из таблицы `msdb..sysdtspackages` или из репозитария (читатель вправе внести соответствующие изменения в код самостоятельно, если этого пожелает), но на данный момент приложение DTSPkgGuru позволяет загружать только пакеты, записанные на диск в виде структурированных файлов хранения COM. Это не означает, что тем самым исключается возможность использовать данное инструментальное средство для модификации пакета, хранящегося в другом виде. Например, если требуется автоматизировать внесение изменений в пакет, который хранится в репозитарии, то достаточно загрузить этот пакет в программу DTS Designer и записать на диск в виде структурированного файла хранения COM. Приложение DTSPkgGuru требует, чтобы загружаемый пакет был представлен в виде структурированного файла хранения COM, но само это приложение способно автоматизировать внесение изменений в любой пакет, который может быть загружен в программу DTS Designer.
2. Читатель мог сам уже догадаться, что второе требование состоит в следующем, — рассматриваемый пакет перед вызовом приложения DTSPkgGuru уже должен быть загружен в программу DTS Designer. Поскольку приложение DTSPkgGuru управляет программой DTS Designer с помощью кодов клавиатуры, для работы приложения требуется, чтобы пакет был уже загружен. Приложение DTSPkgGuru может помочь пользователю автоматизировать внесение изменений в пакет лишь при том условии, что модифицируемый пакет был загружен в программу DTS Designer.

Исполняемый файл и исходный код приложения DTSPkgGuru расположены в подкаталоге `CH20\DTSPkgGuru` компакт-диска, прилагаемого к данной книге. Прежде чем перейти к изучению исходного кода, кратко рассмотрим, как должно эксплуатироваться данное инструментальное средство. Автор преодолел искушение привести в данной главе снимок с экрана этого инструментального средства, поскольку стремился не использовать какие-либо снимки с экрана во всей данной главе (а это весьма нелегко, особенно если учесть, что в главе рассматривается технология, основанная на применении визуального средства проектирования) и не хотел изменять этому принципу даже в этом разделе. Несмотря на сказанное выше, читатель может сам загрузить любой пакет в программу DTS Designer (сделав это, необходимо сохранить пакет в виде файла структурированного хранения COM, если указанный файл еще не создан), затем вызвать на выполнение приложение DTSPkgGuru, чтобы следить за описанием. При этом требуется учитывать, что в системе должна быть только одна работающая копия программы Enterprise Manager. В приложении DTSPkgGuru при подготовке к получению контроля над клавиатурой используется название окна для поиска экземпляра программы Enterprise Manager, поэтому при наличии нескольких работающих копий указанной

программы становится трудно определить, какое именно приложение должно применяться для автоматизации. Ниже описан рекомендуемый порядок действий

1. Прежде всего необходимо щелкнуть на кнопке Open, чтобы загрузить пакет в приложение DTSPkgGuru. Еще раз отметим, что пакет должен быть предварительно записан на диск в структурированном формате хранения COM.
2. В графическом интерфейсе пользователя DTSPkgGuru должны появиться все компоненты пакета и свойства этих компонентов. Компоненты пакета представлены в древовидной форме слева, а свойства – в сетке справа.
3. Изменения в пакете задаются путем внесения этих изменений в графическом интерфейсе пользователя и добавления к списку изменений в нижней части главной формы DTSPkgGuru. Для этого необходимо щелкнуть на свойстве в сетке, находящейся в правой части формы. В текстовом поле в верхней части окна будет отображено значение свойства. Внесите требуемые изменения в текстовое поле, затем щелкните на кнопке Add change. В результате этого изменение будет добавлено к списку изменений в нижней части формы. Повторите указанный процесс для каждого изменения, которое требуется внести в проект.
4. После того как вы будете готовы отправить намеченные изменения в программу DTS Designer, щелкните на кнопке Send changes. В результате контекст активного окна переключится на программу Enterprise Manager (в которой уже должна работать программа DTS Designer с загруженным пакетом), затем произойдет автоматическая передача кодов комбинаций клавиш, необходимых для внесения изменений, оформленных пользователем в виде очереди.
5. После завершения операции модификации пакета можно возвратиться в приложение DTSPkgGuru, нажав клавиши <Alt+Tab>, или продолжить работу с пакетом в программе DTS Designer.
6. Список изменений можно сохранить для последующей выборки как текстовый файл с разграничителями в виде символов табуляции, щелкнув правой кнопкой мыши на этом списке и выбрав команду Save. Кроме того, можно сохранить как текстовый файл с разграничителями в виде символов табуляции значения всех свойств в пакете, щелкнув на кнопке Save all. И в том, и в другом случае используется одинаковый формат файла, поэтому можно загрузить текстовый файл, созданный с помощью любой из этих опций, в список изменений, находящийся в нижней части формы, щелкнув на этом списке правой кнопкой мыши и выбрав команду Load. Поскольку указанные файлы представляют собой обычные текстовые файлы с разграничительными символами табуляции, эти файлы можно загружать в любой текстовый редактор и вносить все необходимые изменения, затем снова загружать в графический интерфейс пользователя DTSPkgGuru и передавать содержимое файлов в программу DTS Designer.

Приложение DTSPkgGuru способно вносить требуемые изменения в пакет благодаря наличию средства Disconnected Edit. Но при использовании этого

средства необходимо учитывать несколько предостережений. Во-первых, во время работы программы DTS Designer в режиме Disconnected Edit не применяется значительная часть обычного кода проверки этой программы. Присваивание свойству недействительного или недопустимого значения с помощью режима Disconnected Edit может иметь катастрофические последствия. Во-вторых, свойства, отображаемые на листе проекта (например, описания задач), модифицированные с помощью режима Disconnected Edit, не будут сразу же показывать свои новые значения в отображении в виде листа проекта после прогона приложения DTSPkgGuru. Тем не менее это не означает, что изменения не внесены, а тот факт, что новые значения не отображаются на листе проекта, обусловлен тем, как организована работа самой программы DTS Designer. Эта программа не обновляет изображение на экране после внесения изменений с помощью режима Disconnected Edit, независимо от того, внесены эти изменения самим пользователем, или такую операцию разрешено провести приложению DTSPkgGuru.

При использовании приложения DTSPkgGuru следует также учитывать такой нюанс, что некоторые свойства не могут быть модифицированы. Например, идентификатор пакета и идентификатор версии пакета — это свойства, допускающие только чтение, и не могут быть изменены. В приложении DTSPkgGuru предпринимается попытка помочь пользователю избежать ошибки, связанной с модификацией неизменяемых свойств. Это выражается в том, что приложение не позволяет вводить изменения, относящиеся к допускающим только чтение свойствам, а также обозначает допускающие только чтение свойства символом комментария при потоковом выводе значений всех свойств в текстовый файл (с помощью кнопки Save all). Тем не менее указанное приложение не может запретить пользователю внести допускающие только чтение свойства в список изменений вручную путем редактирования текстового файла и загрузки этого файла в графический интерфейс пользователя DTSPkgGuru. Учитывая то, что пользователь может вывести в виде текстовых файлов не только списки отдельных изменений, но и весь список значений свойств, появляется возможность внести с помощью текстового редактора в текстовый файл такие изменения, которые касаются любых свойств. Необходимо учитывать, что после добавления к списку изменений свойства, допускающего только чтение, попытка внести соответствующие изменения окончится неудачей, а изменения, которые следуют в списке за тем изменением, которое не удалось внести, также, скорее всего, не удастся выполнить, поскольку программа DTS Designer будет находиться не в том состоянии, на которое рассчитывает приложение DTSPkgGuru.

Итак, несмотря на сказанное выше, рассмотрим код приложения DTSPkgGuru. Этот код также можно найти в подкаталоге `CH20\DTSPkgGuru` компакт-диска, прилагаемого к данной книге. Автор не включил в эту главу весь исходный код приложения DTSPkgGuru, поскольку значительная часть этого кода вырабатывается автоматически интегрированной средой разработки Visual Studio .NET. Вместо этого рассмотрим те части кода, которые были написаны самим автором (листинг 20.20).

Листинг 20.20. Часть исходного кода приложения DTSPkgGuru

```
DTS.Package2Class pkg;

private void btOpen_Click(object sender, System.EventArgs e)
{
    if (DialogResult.OK!=od_dts.ShowDialog()) return;
    tbFileName.Text=od_dts.FileName;
    pkg = new DTS.Package2Class();
    object dummy=new object();
    pkg.LoadFromStorageFile(tbFileName.Text,"",null,null,null,
      ref dummy);
    TreeNode noderoot;
    TreeNode nodeparent;
    TreeNode nodechild;
    btAddChange.Enabled=false;
    btSave.Enabled=false;
    btSend.Enabled=false;
    tvItems.BeginUpdate();
    lvProperties.BeginUpdate();
    lvChanges.BeginUpdate();
    try
    {
    tvItems.Nodes.Clear();
    lvProperties.Items.Clear();
    lvChanges.Items.Clear();
    noderoot=tvItems.Nodes.Add(pkg.Name);
    noderoot.Tag=pkg;
    nodeparent=noderoot.Nodes.Add("Connections");
    foreach (DTS.Connection conn  in pkg.Connections)
    {
      nodechild=nodeparent.Nodes.Add(conn.Name);
      nodechild.Tag=conn;
    }
    nodeparent=noderoot.Nodes.Add("Tasks");
    foreach (DTS.Task task in pkg.Tasks)
    {
nodechild=nodeparent.Nodes.Add(task.Name);
        nodechild.Tag=task;
      }
      nodeparent=noderoot.Nodes.Add("Steps");
      foreach (DTS.Step step in pkg.Steps)
      {
        nodechild=nodeparent.Nodes.Add(step.Name);
        nodechild.Tag=step;
      }
      nodeparent=noderoot.Nodes.Add("Global Variables");
      foreach (DTS.GlobalVariable var in pkg.GlobalVariables)
      {
        nodechild=nodeparent.Nodes.Add(var.Name);
        nodechild.Tag=var;
      }
      tvItems.ExpandAll();
      btSave.Enabled=true;
    }
    finally
```

```
        {
          tvItems.EndUpdate();
          lvProperties.EndUpdate();
          lvChanges.EndUpdate();
        }
      }

      private void tvItems_AfterSelect(object sender,
          System.Windows.Forms.TreeViewEventArgs e)
      {
        tbChange.Text="";
        btAddChange.Enabled=false;
        lvProperties.BeginUpdate();
        try
        {
          lvProperties.Items.Clear();
          if ((null==pkg) ||
            (null==tvItems.SelectedNode) ||
            (null==tvItems.SelectedNode.Tag))
            return;
          DTS.Properties props=null;
          if (tvItems.SelectedNode.Tag is DTS.Package)
            props = (tvItems.SelectedNode.Tag as DTS.Package).
                Properties;
          else if (tvItems.SelectedNode.Tag is DTS.Connection)
props = (tvItems.SelectedNode.Tag as DTS.Connection).
                Properties;
          else if (tvItems.SelectedNode.Tag is DTS.Task)
            props = (tvItems.SelectedNode.Tag as DTS.Task).
                Properties;
          else if (tvItems.SelectedNode.Tag is DTS.Step)
            props = (tvItems.SelectedNode.Tag as DTS.Step).
                Properties;
          else if (tvItems.SelectedNode.Tag is DTS.GlobalVariable)
            props = (tvItems.SelectedNode.Tag as DTS.GlobalVariable).
                Properties;

          foreach (DTS.Property prop in props)
          {
            ListViewItem lvitem = lvProperties.Items.Add(prop.Name);
            lvitem.SubItems.Add(prop.Value as string);
            lvitem.Tag=tvItems.SelectedNode.Tag;
            if (!prop.Set)
              lvitem.ImageIndex=1;  // Свойство, допускающее только
                                    // чтение
          }
        }
        finally
        {
          lvProperties.EndUpdate();
        }
      }

      private void lvProperties_SelectedIndexChanged(object sender,
          System.EventArgs e)
      {
```

```
      if ((null==lvProperties.SelectedItems) ||
        (0==lvProperties.SelectedItems.Count))
        return;
      if (1!=lvProperties.SelectedItems[0].ImageIndex)
      {
        btAddChange.Enabled=true;
        tbChange.ReadOnly=false;
      }
      else
      {
        btAddChange.Enabled=false;
        tbChange.ReadOnly=true;
      }
      // Проверить, находится ли уже данный элемент в списке изменений
      foreach (ListViewItem lvitem in lvChanges.Items)
      {
        if (lvitem.Tag.Equals(lvProperties.SelectedItems[0]))
        {
          tbChange.Text=lvitem.SubItems[3].Text;
          return;
        }
      }
      tbChange.Text=lvProperties.SelectedItems[0].SubItems[1].Text;
    }

    private void btAddChange_Click(object sender,
        System.EventArgs e)
    {
      if ((null==lvProperties.SelectedItems) ||
        (0==lvProperties.SelectedItems.Count)
        )
        return;
      foreach (ListViewItem lvi in lvChanges.Items)
      {
        if (lvi.Tag.Equals(lvProperties.SelectedItems[0]))
        {
          lvChanges.Items.Remove(lvi);
          break;
        }
      }

      ListViewItem lvitem = lvChanges.Items.Add(tvItems.
          SelectedNode.Text);
      lvitem.SubItems.Add(lvProperties.SelectedItems[0].
           SubItems[0].Text);
      lvitem.SubItems.Add(lvProperties.SelectedItems[0].
           SubItems[1].Text);
      lvitem.SubItems.Add(tbChange.Text);
      lvitem.Tag=lvProperties.SelectedItems[0];
      btSend.Enabled=true;
    }
    // Удалить данные об изменениях
    private void menuItem1_Click(object sender, System.
        EventArgs e)
    {
lvChanges.Items.Clear();
```

```
    btSend.Enabled=false;
  }

  // Преобразовать коды нажатий клавиш, которые нарушают работу
  // функции SendKeys
  string ReplaceSpecialKeys(string instring)
  {
    if (null==instring) return string.Empty;
    instring=instring.Replace("{","\\{").Replace("}","\\}").
   Replace("%","{%}").Replace("^","{^}").Replace("+","{+}");
    return instring.Replace("\\{","{{}").Replace("\\}","{}}");
  }

  private void btSend_Click(object sender, System.EventArgs e)
  {
    foreach (ListViewItem lvitem in lvChanges.Items)
    {
      lvitem.Checked=false;
    }

    int hWnd=Win32.FindWindow(null,
        "SQL Server Enterprise Manager");
    if (Win32.SetForegroundWindow(hWnd))
    {
      lvChanges.BeginUpdate();
      try
      {
        string Keys;
        Keys="%pd";
        SendKeys.SendWait(Keys);
        foreach (ListViewItem lvitem in lvChanges.Items)
        {
          Keys="";
          if ((lvitem.Tag as ListViewItem).Tag is DTS.Connection)
             Keys+="c";
           else if ((lvitem.Tag as ListViewItem).Tag is DTS.Task)
             Keys+="t";
           else if ((lvitem.Tag as ListViewItem).Tag is DTS.Step)
             Keys+="s";
          else if ((lvitem.Tag as ListViewItem).Tag is
              DTS.GlobalVariable)
             Keys+="g";
          SendKeys.SendWait(Keys);
          if (0!=Keys.Length) // Свойства, отличные от свойств пакета
{
             Keys="{RIGHT}{DOWN}";
             SendKeys.SendWait(Keys);
            Keys=lvitem.Text;  // Имя элемента
             SendKeys.SendWait(Keys);
           }
          Keys="{TAB}{DOWN}";
           SendKeys.SendWait(Keys);
          Keys=lvitem.SubItems[1].Text;  // Имя свойства
           SendKeys.SendWait(Keys);
           Keys="{ENTER}";
           SendKeys.SendWait(Keys);
```

```
            Keys="{TAB}";
             SendKeys.SendWait(Keys);
            Keys=ReplaceSpecialKeys(lvitem.SubItems[3].Text);
             SendKeys.SendWait(Keys);
             Keys="{ENTER}";
             SendKeys.SendWait(Keys);
             Keys="+{TAB}{HOME}";
             SendKeys.SendWait(Keys);
             lvitem.Checked=true;
          }
          Keys="%c"; // Закрыть диалоговое окно Disconnected Edit
          SendKeys.SendWait(Keys);
        }
       finally
        {
          lvChanges.EndUpdate();
        }
      }
    else MessageBox.Show("Could not find Enterprise Manager
        window");
  }

  TreeNode FindNode(TreeNodeCollection nodes, string
      searchstring)
  {
     TreeNode result=null;
    foreach (TreeNode tvitem in nodes)
     {
       if (0!=tvitem.Nodes.Count)
        result=FindNode(tvitem.Nodes,searchstring);
      if (null==result)
       {
if (tvitem.Text.ToLower()==searchstring.ToLower())
    {
      result = tvitem;
      break;
    }
  }
  else break;
  }
  return result;
}

// Преобразовать коды символов в эквивалентные управляющие коды (для
// сохранения в текстовом файле)
string NoEscapeChars(object inobj)
{
  string instring = inobj as string;
  if (null==instring) return string.Empty;
  return instring.Replace("\n\r","\\n\\r").Replace("\n","\\n").
    Replace("\r","\\r").Replace("\t","\\t");
}

// Преобразовать управляющие коды в эквивалентные коды символов (для
// загрузки из текстового файла)
string ReplaceEscapeChars(object inobj)
{
```

```
    string instring = inobj as string;
    if (null==instring) return string.Empty;
    return instring.Replace("\\n\\r","\n\r").Replace("\\n","\n").
      Replace("\\r","\r").Replace("\\t","\t");
}
// Загрузить список изменений
private void menuItem2_Click(object sender,
    System.EventArgs e)
{
    if (DialogResult.OK!=od_TXT.ShowDialog()) return;
    lvChanges.Items.Clear();
    StreamReader f = File.OpenText(od_TXT.FileName);
    tvItems.BeginUpdate();
    lvProperties.BeginUpdate();
    lvChanges.BeginUpdate();
    try
    {
    string linein;
    while (null!=(linein = f.ReadLine()))
    {
if (linein[0]==';') // Комментарий
    continue;
string[] args=linein.Split('\t');
ListViewItem lvitem = new ListViewItem();
// Выполнить разбивку входной строки в местах вхождения знаков
// табуляции
foreach (string arg in args)
{
    if (0==lvitem.Text.Length)
    lvitem.Text=arg;
    else
    lvitem.SubItems.Add(ReplaceEscapeChars(arg));
}

// Найти родительский элемент (задача, этап и т.д.)
TreeNode parentnode=null;
parentnode=FindNode(tvItems.Nodes,lvitem.SubItems[0].
    Text);
if (null==parentnode)
{
    MessageBox.Show("Unable to locate item " +
      lvitem.SubItems[0].Text);
    return;
}
tvItems.SelectedNode=parentnode;

// Найти родительское свойство
ListViewItem propitem=null;
foreach (ListViewItem lvi in lvProperties.Items)
{
    if (lvi.Text.ToLower()==lvitem.SubItems[1].
      Text.ToLower())
    {
    propitem = lvi;
    break;
    }
```

```
}
if (null==propitem)
{
    MessageBox.Show("Unable to locate property " +
      lvitem.SubItems[1].Text);
    return;
}
lvitem.Tag=propitem;
lvChanges.Items.Add(lvitem);
          btSend.Enabled=true;

        }
      }
      finally
      {
        f.Close();
        tvItems.EndUpdate();
        lvProperties.EndUpdate();
        lvChanges.EndUpdate();
      }

    }
    // Сохранить список изменений
    private void menuItem3_Click(object sender,
        System.EventArgs e)
    {
      if (DialogResult.OK!=sd_TXT.ShowDialog()) return;
      StreamWriter f = File.CreateText(sd_TXT.FileName);
      try
      {
        f.WriteLine(";ItemName\tPropertyName\tOldValue\tNewValue");
        foreach (ListViewItem lvitem in lvChanges.Items)
        {
          f.WriteLine("{0}\t{1}\t{2}\t{3}",lvitem.Text,lvitem.
              SubItems[1].Text,lvitem.SubItems[2].Text,lvitem.
              SubItems[3].Text);
        }
        od_TXT.FileName=sd_TXT.FileName;
      }
      finally
      {
        f.Close();
      }
    }

    private void btSave_Click(object sender,
        System.EventArgs e)
    {
      if (DialogResult.OK!=sd_TXT.ShowDialog()) return;
      StreamWriter f = File.CreateText(sd_TXT.FileName);
      try
{
   f.WriteLine(";ItemName\tPropertyName\tOldValue\tNewValue");
   foreach (DTS.Property prop in pkg.Properties)
   {
    f.WriteLine("{0}{1}\t{2}\t{3}\t{4}",prop.Set?"":";",
```

```
            pkg.Name,prop.Name,NoEscapeChars(prop.Value),
            NoEscapeChars(prop.Value));
        }
        foreach (DTS.Connection conn in pkg.Connections)
        {
         foreach (DTS.Property prop in conn.Properties)
         {
           f.WriteLine("{0}{1}\t{2}\t{3}\t{4}",prop.Set?"":";",
               conn.Name,prop.Name,NoEscapeChars(prop.Value),
               NoEscapeChars(prop.Value));
         }
        }
        foreach (DTS.Task task in pkg.Tasks)
        {
         foreach (DTS.Property prop in task.Properties)
         {
           f.WriteLine("{0}{1}\t{2}\t{3}\t{4}",prop.Set?"":";",
               task.Name,prop.Name,NoEscapeChars(prop.Value),
               NoEscapeChars(prop.Value));
         }
        }
        foreach (DTS.Step step in pkg.Steps)
        {
         foreach (DTS.Property prop in step.Properties)
         {
           f.WriteLine("{0}{1}\t{2}\t{3}\t{4}",prop.Set?"":";",
               step.Name,prop.Name,NoEscapeChars(prop.Value),
               NoEscapeChars(prop.Value));
         }
        }
        foreach (DTS.GlobalVariable var in pkg.GlobalVariables)
        {
         foreach (DTS.Property prop in var.Properties)
         {
           f.WriteLine("{0}{1}\t{2}\t{3}\t{4}",prop.Set?"":";",
               var.Name,prop.Name,NoEscapeChars(prop.Value),
               NoEscapeChars(prop.Value));
}
         }
         }
         finally
         {
         f.Close();
         }
}
```

Ниже в данном разделе приведено поэтапное описание работы кода, приведенного в листинге 20.20, и указаны те элементы, которые автор счел в наибольшей степени заслуживающими внимания. При этом затронуты лишь наиболее важные особенности, а не приведены утомительные построчные пояснения. В листинге 20.20 определения описанных ниже методов выделены полужирным шрифтом.

1. Для того чтобы в приложении DTSPkgGuru можно было применять вызовы к объектной модели DTS Package, в проект необходимо вначале ввести ссылку на объектную библиотеку COM — библиотеку DTS Package. Такую операцию

можно выполнить, выбрав в интегрированной среде разработки Visual Studio .NET команду Add Reference и перейдя на вкладку COM в диалоговом окне. В приложениях с управляемым кодом можно обеспечить использование COM-объектов с помощью средств COM Interop – набора служб и API-интерфейсов, которые обеспечивают работу с COM-объектами и упрощают для управляемых классов задачу получения доступа к COM-объектами. (Обратите внимание на то, что в верхней части файла исходного кода `DTSPkgGuru.cs` имеется ссылка на сборку `System.Runtime.InteropServices`.) Объектная библиотека DTS Package указана на вкладке COM так же, как должна быть указана в среде Visual Basic 6.

2. Далее отметим, что код обработчика событий нажатия кнопки `btOpen` обеспечивает открытие пакета и заполнение графического интерфейса пользователя значениями элементов и свойств пакета. Этот код действует во многом аналогично коду VB, который рассматривался в последнем примере приложения, поскольку просто осуществляется обработка в цикле соответствующих коллекций и заполнение связанных с ними элементов графического интерфейса пользователя. Заслуживает также внимания то, как используется свойство `Tag` для слежения за элементами пакета DTS, к которым относятся конкретные узлы дерева. Это позволяет применять связи, ориентированные в противоположном направлении, после того как пользователь подготавливает все необходимое для передачи списка изменений в программу DTS Designer, чтобы определить, какую ветвь древовидного представления Disconnected Edit следует выбрать. Например, появляется возможность выбирать разные ветви с учетом того, происходит ли внесение изменений в объект `Connection` или в объект `Task`.
3. Код обработчика событий `AfterSelect` древовидного представления позволяет заполнять сетку свойств с учетом того, какой компонент пакета выбран в древовидном представлении. А после каждого изменения состава выбранных компонентов происходит переформирование списка свойств.
4. Код обработчика событий `SelectedIndexChanged` представления списка свойств позволяет, во-первых, определить, доступно ли свойство для редактирования, и изменить соответствующие элементы графического интерфейса пользователя должным образом, и, во-вторых, отобразить модифицированное значение свойства (если таковое имеется) в текстовом поле в верхней части формы.
5. Код обработчика событий щелчка на кнопке `btAddChange` позволяет добавить запись к представлению списка изменений в нижней части формы. Запись в списке изменений состоит из имени объекта, имени свойства, старого и нового значений свойства. Старое значение фактически не используется для чего бы то ни было и включается в список только для вывода на экран.
6. Код обработчика событий щелчка на кнопке `btSend` обеспечивает переключение фокуса ввода на программу Enterprise Manager и передачу необходимых кодов комбинаций клавиш для внесения изменений, содержащихся в списке изменений. В этом приложении используется класс `SendKeys`

инфраструктуры .NET Framework, а коды комбинаций клавиш, соответствующие каждому изменению, передаются с помощью нескольких отдельных вызовов. При этом прежде всего отображается диалоговое окно Disconnected Edit, затем передаются коды комбинаций клавиш, соответствующие каждому изменению. После завершения передачи диалоговое окно Disconnected Edit закрывается. Еще раз отметим, что автоматизация внесения изменений обеспечивается исключительно с применением кодов комбинаций клавиш, поэтому приложение DTSPkgGuru может помочь пользователю модифицировать любой пакет, который может загружен в программу DTS Designer, независимо от того, где хранится этот пакет. В приложении DTSPkgGuru пакет в формате структурированного файла COM требуется для упрощения присваивания порядковых номеров собственным элементам графического интерфейса пользователя этого приложения, но само приложение позволяет вносить изменения в пакеты любых типов.

7. Код обработчика событий щелчка на кнопке `btSave` записывает информацию обо всех компонентах пакета и свойствах этих компонентов в текстовый файл с разграничительными символами табуляции. Поскольку формат этого файла специально принят аналогичным формату списка изменений, в коде рассматриваемого обработчика событий текущее значение каждого свойства записывается в файл дважды – в первый раз записывается старое значение, а во второй раз – новое. В файле можно откорректировать новое значение с помощью текстового редактора и снова загрузить файл в графический интерфейс пользователя DTSPkgGuru, чтобы использовать полученный список в качестве списка изменений. Указанный список является также удобным инструментом для сохранения пакета на диске в виде текстового файла. Итак, любой пакет можно не только сохранить в виде кода VB в программе DTS Designer, но и воспользоваться указанной выше возможностью сохранить пакет на диске в виде текстового файла, чтобы получить полное представление и о пакете, и о его содержимом. Поскольку файл изменений представляет собой стандартный файл с разграничительными символами табуляции, можно даже импортировать этот файл в таблицу с помощью служб DTS для дальнейшего анализа или сохранить в системе контроля версий, чтобы иметь возможность следить за тем, какие изменения вносились в пакет со временем, и сравнивать одну версию с другой с применением таких инструментальных средств поиска различий в текстовых файлах, как утилита WinDiff и инструмент поиска различий diff пакета Visual SourceSafe.

Итак, на этом описание приложения DTSPkgGuru завершено. Автор рекомендует изучить исходный код этого приложения, вызвать саму утилиту на выполнение и даже применить приложение для внесения некоторых изменений в пакеты, загруженные в программу DTS Designer. Если читателю когда-либо придется решать проблему автоматизации и внесения многочисленных изменений в пакеты, то он вполне может согласиться с автором, что приложение DTSPkgGuru окажется для этого весьма удобным.

Резюме

Службы DTS являются долгожданным и мощным дополнением к семейству технологий SQL Server. Эти службы предоставляют пользователю широкий набор средств преобразования данных и управления потоком данных, способных соперничать на равных с многими программными продуктами независимых поставщиков. Во многих отношениях службы DTS представляют собой самодостаточную среду визуального программирования, поскольку технологию DTS можно использовать для создания целого ряда приложений различных типов, даже не относящихся непосредственно к преобразованию данных.

Основным средством транспортировки данных в пакете DTS является компонент многофазной перекачки данных. Этот компонент позволяет передавать данные из одного средства доступа OLE DB в другое. К тому же, данный компонент представляет собой машину преобразования данных, лежащую в основе задач Transform Data, Data Driven Query и Parallel Data Pump. Кроме того, для перемещения данных из текстовых файлов в базы данных SQL Server могут использоваться задачи Bulk Insert. Службы DTS предоставляют целый ряд механизмов, позволяющих создавать гибкие и быстродействующие средства перемещения и преобразования данных.

Любой пакет DTS допускает возможность применять в нем широкий набор средств программирования. Практически на любом этапе преобразования могут выполняться сценарии ActiveX. За конкретными фазами перекачки данных могут быть закреплены конкретные сценарии, кроме того, сценарии могут быть связаны с отдельными элементами потока данных. Учитывая то, что с помощью средств поддержки сценариев ActiveX можно получить доступ практически к любым объектам (например, к COM-объектам, ADO, к файловой системе, к API-интерфейсу Windows и т.д.), способы применения возможностей, предоставляемых пользователю службами DTS, становятся практически безграничными.

Вопросы для самопроверки

1. В каком формате следует сохранить пакет DTS для того, чтобы передать, допустим, с помощью электронной почты?
2. Какой единственный компонент DTS представляет собой машину, лежащую в основе задачи Transform Data, задачи Data Driven Query и задачи Parallel Data Pump?
3. Подтвердите или опровергните следующее утверждение. Для использования подписки на основе преобразуемой репликации необходимо определить пакет с помощью программы DTS Designer, а для получения доступа к опубликованным статьям использовать задачу Data Driven Query.
4. Какую задачу DTS можно использовать для выборки значений из файла INI и присваивания этих значений глобальным переменным?

5. Подтвердите или опровергните следующее утверждение. Для условного выполнения какой-то конкретной строки задачи в пакете DTS необходимо использовать задачу ActiveX Script.
6. Какой интерфейс должен реализовывать COM-объект, чтобы этот объект мог служить в качестве определяемой пользователем задачи DTS?
7. Опишите назначение ограничений предшествования в пакете DTS.
8. Каким образом можно определить, что поисковый запрос возвратил несколько строки?
9. Какую задачу DTS необходимо использовать в процессе для вызова на выполнение другого процесса?
10. Опишите обстоятельства, в которых задача Data Driven Query была бы более предпочтительной по сравнению с задачей Transform Data или с задачей Bulk Insert.
11. Какое количество готовых запросов может быть связано с единственной задачей Data Driven Query?
12. Допустим, что вложенный пакет присоединился к контексту транзакции вызывающего пакета. Что происходит, если в дочернем пакете выполняется фиксация транзакции?
13. Можно ли выполнить настройку конфигурации задачи Bulk Insert так, чтобы в ней применялись преобразования ActiveX?
14. Какие действия необходимо выполнить в первую очередь, чтобы иметь возможность сохранять пакеты DTS в архиве Meta Data Services?
15. В какой базе данных и таблице находится пакет DTS, хранящийся локально в СУБД SQL Server?
16. Какое средство доступа OLE DB используется для выполнения запроса к пакету DTS из сценария на языке Transact-SQL?
17. Какое средство доступа OLE DB используется для выполнения запроса к данным публикации, которые относятся к преобразуемой подписке?
18. Какой этап перекачки данных осуществляется в первую очередь: этап, следующий за чтением из источника, или этап завершения перекачки?
19. Подтвердите или опровергните следующее утверждение. Чтобы можно было использовать высокоскоростные средства массового копирования, предоставляемые источником данных OLE DB, необходимо разрешить опцию Use fast load в задаче Transform Data, поскольку эта опция не разрешена по умолчанию.
20. Подтвердите или опровергните следующее утверждение. В задаче Execute SQL, безусловно, предоставляется возможность выполнять обычные сценарии T-SQL, включающие признаки конца пакета, но не предусмотрен механизм выборки выходных данных с нерегулярной структурой или многочисленных наборов строк.
21. Почему способ изменения значений свойств в пакете DTS с использованием средства Disconnected Edit является потенциально опасным?

22. Какая команда T-SQL вызывается в задаче Bulk Insert?
23. Какую опцию необходимо разрешить, чтобы получить возможность ознакомиться с опциями многофазной перекачки данных в программе DTS Designer?
24. Какой неприятный побочный эффект возникает при сохранении пакета DTS с использованием кода Automation?
25. Подтвердите или опровергните следующее утверждение. Средства DTS поддерживают прямые соединения с источниками данных HTML.
26. Объясните назначение опции потока данных Close connection on completion.
27. Какой метод объекта `DTS Package` можно вызвать в приложении для загрузки пакета непосредственно из структурированного файла хранения?
28. Подтвердите или опровергните следующее утверждение. Объектная библиотека `DTS Package` — это библиотека COM, поэтому в таких языках определения управляемого кода, как C# или VB.NET, для перебора в цикле элементов коллекции, доступ к которой предоставляет указанная библиотека, нельзя использовать конструкцию `foreach`.
29. Подтвердите или опровергните следующее утверждение. Даже если в пакете имеется только один объект `Connection`, необходимо иметь доступ к средствам Microsoft Distributed Transaction Coordinator, чтобы можно было обеспечить ведение очереди модифицированных данных в транзакции при эксплуатации пакета DTS.
30. Подтвердите или опровергните следующее утверждение. Разумеется, на том компьютере, на котором эксплуатируется программа DTS Designer, могут быть инсталлированы многие языки сценариев ActiveX, но при подготовке кода сценариев ActiveX для пакетов DTS допускается использовать только языки VBScript и JScript.
31. Что необходимо сделать сразу после создания нового проекта, чтобы можно было использовать объектную библиотеку `DTS Package` в приложении с управляемым кодом?
32. Опишите ситуацию, в которой было бы целесообразно разрешить опцию потока данных Execute on main package thread.
33. Допустим, что дочерний пакет, в котором разрешено свойство Use transactions, вызывается на выполнение с помощью задачи Execute Package из пакета, в котором уже инициирована транзакция, притом что в транзакцию включена задача Execute Package. Будет ли дочерний пакет инициализировать вложенную транзакцию?
34. Подтвердите или опровергните следующее утверждение. Средства COM Interop, предоставляемые в инфраструктуре .NET Framework, не обеспечивают поддержки интерфейса `IDispatch`, поэтому в приложении с управляемым кодом невозможно открыть пакет DTS с использованием объектной библиотеки `DTS Package`.
35. Опишите, каким образом осуществляется преобразование `WriteFile`.

ГЛАВА 21

Репликация снимка

В этой и в следующих двух главах приведены сведения о том, как функционируют средства репликации SQL Server. Настоящая книга не посвящена репликации как таковой, поэтому автор не имеет ни времени, ни места для того, чтобы раскрыть указанную тему с той широтой и полнотой, которой она заслуживает. Главы этой книги, относящиеся к репликации, не содержат информации о подготовке к работе или по администрированию средств репликации, поэтому на данных страницах вы не найдете снимков с экрана программ-мастеров Enterprise Manager Replication. Дело в том, что сами программы-мастера Replication достаточно просты, чтобы их можно было использовать, не прибегая к помощи книги, подобной этой. Более того, основные вопросы настройки конфигурации и управления репликацией весьма неплохо освещены в оперативной документации Books Online, поэтому ту информацию, которую нельзя получить с помощью программ-мастеров Enterprise Manager, вполне можно найти в указанной документации.

Кроме того, в настоящей книге не приведены сведения о многих сложных требованиях, связанных с репликацией, а также не перечислены предостережения и исключения, которые неразрывно связаны с каждым отдельным аспектом репликации. Еще раз отметим, что вся эта информация весьма подробно представлена в оперативной документации Books Online, поэтому в настоящей книге нет смысла просто цитировать документацию.

При написании данной главы предполагалось, что читатель изучил в оперативной документации Books Online все, что касается репликации, и знаком с такими основными понятиями репликации, как распределительный сервер (distributor), сервер публикаций (publisher), агент (agent), снимок (snapshot), статья (article) и т.д. Если вы еще не ознакомились с тем, что сказано о репликации в оперативной документации Books Online, рекомендуем сделать это сейчас.

Как и в отношении других тем, рассматриваемых в настоящей книге, автор при написании данной главы поставил перед собой цель представить репликацию с точки зрения архитектуры программного обеспечения и ответить на фундаментальные вопросы о том, как работают средства репликации и каким образом организовано функционирование этих средств. Многие пользователи имеют элементарное представление о характере действий, происходящих при репликации снимка, но как именно осуществляются эти действия? Раскрытию этой темы как раз и посвящена данная глава.

Краткий обзор

Репликация снимка сводится к тому, что снимается копия некоторого объекта на сервере публикаций, после чего эта копия передается подписчику. Передача реплицированных данных осуществляется по принципу "все или ничего". Это означает, что при репликации снимка не обеспечивается распространение инкрементных изменений, хотя и можно ограничить диапазон выборки в снимке как по горизонтали (по строкам), так и по вертикали (по столбцам), чтобы уменьшить объем данных, попадающих в снимок. С другой стороны, для распространения инкрементных изменений можно использовать транзакционную репликацию или репликацию слиянием.

В программе SQL Server средства репликации снимка реализованы на основе программ Snapshot Agent и Distribution Agent. Программа Snapshot Agent обеспечивает создание снимков данных и снимков схемы, которые распространяются по подписчикам, а программа Distribution Agent обеспечивает получение и применение снимков к базе данных подписчика.

Программа Snapshot Agent

Как и все другие агенты репликации, программа Snapshot Agent представляет собой приложение для работы в терминальном режиме, в котором используется интерфейс ODBC для организации взаимодействия с программой SQL Server. Исполняемые файлы для агентов репликации SQL Server находятся в подкаталоге `80\COM` каталога `Microsoft SQL Server`. Во время настройки конфигурации средств публикации снимка программа SQL Server создает задание SQL Server Agent, в котором эксплуатируется программа Snapshot Agent. С указанным заданием можно ознакомиться, раскрыв соответствующую запись под узлом Agents\Snapshot Agents в программе Replication Monitor на компьютере распределительного сервера (или перейдя к этой записи с помощью узла Jobs, находящегося под узлом Management\SQL Server Agent) в программе Enterprise Manager. Каждое задание на репликацию снимка состоит из нескольких этапов, одним из которых является вызов на выполнение исполняемого файла `snapshot.exe` программы Snapshot Agent. Во время настройки конфигурации этап задания, в котором фактически эксплуатируется программа Snapshot Agent, определяется как относящийся к типу Replication Snapshot, и служит для задания SQL Server Agent указанием, что при выполнении данного этапа необходимо вызвать исполняемый файл `snapshot.exe`.

Для определения того, какие параметры передаются при вызове исполняемого файла `snapshot.exe` по умолчанию, можно ознакомиться с этапом типа Replication Snapshot рассматриваемого задания. В зависимости от того, как была выполнена настройка публикации снимка, рассматриваемые параметры должны выглядеть примерно следующим образом:

```
-Publisher [TUK\PHRIP] -PublisherDB [pubs] -Distributor
    [TUK\PHRIP] -Publication [pubs_sales] -DistributorSecurityMode 1
```

Следует отметить, что исполняемый файл snapshot.exe может быть вызван на выполнение независимо от задания SQL Server Agent. В упражнении 21.1 показано, как сделать именно это.

Упражнение

Упражнение 21.1. Вызов программы Replication Agent на выполнение из командной строки

1. Запустите программу Enterprise Manager на том же компьютере, на котором эксплуатируется распределительный сервер, и воспользуйтесь программой-мастером Create Publication для создания публикации снимка, если это еще не сделано. В ходе определения конфигурации средств публикации снимка с помощью программы-мастера Create Publication программа Enterprise Manager вызывает хранимые процедуры sp_addpublication и sp_addpublication_snapshot для создания требуемой публикации, как показано в листинге 21.1.

Листинг 21.1. Вызовы хранимых процедур sp_addpublication и sp_addpublication_snapshot

```
exec sp_addpublication @publication = N'pubs_titles',
    @restricted = N'false', @sync_method = N'native',
    @repl_freq = N'snapshot', @description = N'Snapshot
    publication of pubs database from Publisher TUK\PHRIP.',
     @status = N'inactive', @allow_push = N'true', @allow_pull =
    N'true', @allow_anonymous = N'true', @enabled_for_internet =
    N'false', @independent_agent = N'true', @immediate_sync =
     N'true', @allow_sync_tran = N'false', @autogen_sync_procs =
    N'true', @retention = 336, @allow_queued_tran = N'false',
    @snapshot_in_defaultfolder = N'true', @compress_snapshot =
    N'false', @ftp_port = 21, @allow_dts = N'false',
    @allow_subscription_copy = N'false', @add_to_active_directory =
    N'false'
exec sp_addpublication_snapshot @publication = N'pubs_titles',
    @frequency_type = 8, @frequency_interval = 64,
     @frequency_relative_interval = 0, @frequency_recurrence_factor
     = 1, @frequency_subday = 1, @frequency_subday_interval = 0,
    @active_start_date = 0, @active_end_date = 99991231,
     @active_start_time_of_day = 10600, @active_end_time_of_day = 0
```

2. Откройте запись программы Snapshot Agent, соответствующую созданной публикации, которая находится под узлом Agents\Snapshot Agents окна Replication Monitor программы Enterprise Manager.
3. Дважды щелкните на обозначении этапа задания с именем Run Agent.
4. Выберите текст в окне Command и скопируйте его в буфер обмена (с помощью комбинации клавиш <Ctrl+C>).
5. Закройте редактор этапа задания и диалоговое окно свойств агента в программе Enterprise Manager.

6. Откройте командное окно на компьютере с распределительным сервером и перейдите в подкаталог `80\COM` каталога `Microsoft SQL Server`.
7. Введите в командной строке слово `snapshot` и пробел, затем вставьте в командную строку содержимое буфера обмена (например, с помощью комбинации клавиш <Alt+Space, E, P>).
8. Нажмите клавишу <Enter>, чтобы запустить на выполнение программу агента из командной строки. Должен появиться примерно такой вывод, который представлен в листинге 21.2.

Листинг 21.2. Результаты выполнения команды, сформированной в командной строке

```
Microsoft SQL Server Snapshot Agent 8.00.194
Copyright (c) 2000 Microsoft Corporation

Generating schema script for article 'titles'
Bulk copying snapshot data for article 'titles'
Bulk copied snapshot data for article 'titles' (18 rows).
Inserted schema command for article 'titles' into the
distribution database
Inserted index creation command for article 'titles' into the
distribution database.
Inserted bcp command for article 'titles' into the distribution
database.
A snapshot of 1 article(s) was generated.

The process finished. Use CTRL+C to close this window.
```

9. Как показывает листинг 21.2, с помощью команды, сформированной в командной строке, был создан такой же снимок, который обычно создается из задания SQL Server Agent. Указанная возможность является удобной, если приходится сталкиваться с затруднениями при получении снимка с помощью обычных средств. Приведенную выше команду можно вызвать на выполнение из командной строки, чтобы ознакомиться с выводом команды, появившимся в окне терминала, а не быть вынужденным просматривать системные таблицы в базе данных распределительного сервера.
10. Следует отметить, что вывод, сформированный агентом, может быть направлен в текстовый файл с помощью параметра `-Output filename`. Для этого просто введите в командной строке агента параметр `-Output`, за которым укажите имя выходного файла, который должен быть создан агентом, после чего выполните перезапуск агента, чтобы опробовать такую возможность. Если указанный файл уже существует, то выходные данные будут добавлены к этому файлу, если же его не существует, он будет создан.

Файлы, содержащие данные и схему для статей публикации, создаются программой Snapshot Agent в каталоге снимка. По умолчанию каталог снимка является подкаталогом каталога `ReplData`, который, в свою очередь, является подкаталогом корневого каталога инсталляции распределительного сервера программы SQL Server, но вместо него можно указать другой каталог с помощью диалогового окна Publication Properties программы Enterprise Manager. Для каждой публикации создается отдельный каталог снимка. Имя этого каталога состоит из имени сервера,

экземпляра и базы данных, из которых публикуются данные рассматриваемой публикации, а также из имени самой публикации. В каждом каталоге публикуемого снимка имеется два подкаталога — unc и FTP. Подкаталог unc представляет собой контейнер для публикуемых снимков, которые распространяются в рамках универсального соглашения об именовании UNC, а подкаталог FTP — это контейнер публикуемых снимков, которые распространяются с помощью протокола FTP. Если явно не указано, что публикация должна распространяться с помощью протокола FTP, то снимки этой публикации должны находиться в подкаталоге, соответствующем подкаталогу unc.

Для каждого сформированного снимка создается отдельный подкаталог либо в каталоге unc, либо в каталоге FTP. Имя созданного подкаталога состоит из значений даты и времени формирования снимка. Наконец, в этом подкаталоге фактически находятся сами файлы снимков.

Если настройка конфигурации публикации будет выполнена таким образом, чтобы снимки публикации формировались в другом месте (с помощью диалогового окна Publication Properties), но при этом формирование снимков в обычном каталоге снимка не будет отменено, то снимок фактически будет записан в оба каталога. А если разрешено распространение публикации по протоколу FTP, то может быть задан относительный путь, который может использоваться FTP-клиентом для доступа к файлам снимков.

Сами файлы снимков представляют собой смесь данных BCP и сценариев Transact-SQL. Данные хранятся в формате BCP и имеют расширение файла .bcp. Если настройка конфигурации публикации была выполнена для распределения только в другие программы SQL Server и не разрешено преобразование публикаций с помощью средств DTS, то файл записывается в собственном формате BCP (поскольку такая операция записи обычно выполняется быстрее); в противном случае используется символьный формат. Остальные файлы (например, с расширениями .sch, .dri, .trg, .idx и т.д.) содержат код Transact-SQL, необходимый для создания объекта каждой статьи, декларативных ограничений ссылочной целостности, триггеров, индексов и других необходимых компонентов статьи.

Файлы BCP, созданные в программе Snapshot Agent, позволяют хранить не только данные, соответствующие таблице, но и данные, возвращенные объектами представления. По аналогии с тем, что утилиту BCP можно применять для записи результатов запроса к представлению или к файлу, программа Snapshot Agent позволяет создать не только снимок данных, возвращенных представлением, но и представление, содержащее сами данные.

После запуска на выполнение программа Snapshot Agent определяет, появились ли новые подписки со времени последнего запуска этой программы. Если новые подписки не были созданы, то агенту не требуется создавать новые сценарии или файлы данных. Но если была создана публикация с опцией, предусматривающей немедленное формирование снимка публикации, то программа Snapshot Agent создает новый снимок для публикации после каждого запуска на выполнение этой программы агента.

Следует отметить, что программа Snapshot Agent используется при формировании снимков не только для репликации снимка, но также для транзакционной

репликации и репликации слиянием. Но независимо от типа репликации необходим начальный снимок, чтобы у подписчика был исходный образ данных.

После создания снимка, относящегося к публикации, этот снимок забирает либо программа Distribution Agent (если используется репликация снимка и транзакционная репликация), либо программа Merge Agent (если используется репликация слиянием), после чего снимок распространяется по подписчикам. Кроме того, пользователь может сам взять необходимые файлы из каталога снимка и передать их подписчику вручную. Такая операция часто выполняется на практике, когда впервые осуществляется настройка конфигурации удаленного подписчика, который имеет соединение с распределительным сервером, характеризующееся низкой пропускной способностью. Поместив начальный снимок, скажем, на компакт-диск, можно обеспечить гораздо более быструю настройку конфигурации удаленного узла, чем при передаче снимка через медленный канал глобальной сети, когда приходится очень долго ожидать окончания передачи.

Функции программ Snapshot Agent и Distribution Agent

Как было указано выше, работа по репликации снимка распределяется между программами Snapshot Agent и Distribution Agent. В следующем разделе функционирование каждой из этих программ рассматривается отдельно.

Задачи программы Snapshot Agent

Работа программы Snapshot Agent начинается с установления соединения между распределительным сервером и сервером публикаций и установки разделяемой блокировки на каждой из таблиц, включенных в статьи публикации. Такие блокировки гарантируют получение единообразного представления данных, поскольку исключают возможность внесения изменений в данные до освобождения блокировок. Безусловно, это также означает, что программа Snapshot Agent должна функционировать в те периоды, когда запрещение вносить изменения в опубликованные статьи не становится источником затруднений для пользователей.

В процессе своей работы программа Snapshot Agent ведет запись всех выполняемых действий в таблице `MSsnapshot_history`. Ознакомившись с таблицей `MSsnapshot_history`, можно обнаружить записи журнала с данными, аналогичными тем, которые обнаруживаются после запуска агента в приглашении к вводу команд. Степень детализации сообщений агента можно откорректировать с помощью параметра настройки конфигурации агента `OutputVerboseLevel`. От этого параметра зависит то, насколько подробными становятся записи в таблице `MSsnapshot_history` (а также на терминале, если запуск агента осуществляется из командной строки). Таблица `MSsnapshot_history` является удобным источником информации, с которого можно начать, если во время работы со снимками возникают проблемы и требуется точно определить, что именно является причиной возникновения этих проблем. Доступ к информации из таблицы `MSsnapshot_history`

можно получить, просто выполнив запрос к таблице; еще один вариант состоит в том, что в программе Enterprise Manager можно выбрать публикацию под узлом Replication Monitor, затем дважды щелкнуть на соответствующем этой публикации узле Snapshot Agent (каждая публикация снимка имеет свою собственную программу Snapshot Agent) и выбрать команду Session Details в диалоговом окне Snapshot Agent History. В листинге 21.3 приведен пример той информации, которую можно найти в таблице MSsnapshot_history.

Листинг 21.3. Пример содержимого таблицы MSsnapshot_history

```
(Приведена в сокращенном виде)

runs start_time              dur comments
---- ----------------------- --- ----------------------------
1    2003-03-25 15:39:57.977 0   Starting agent.
1    2003-03-25 15:39:59.870 0   Initializing
3    2003-03-25 15:39:59.870 1    Connecting to Publisher 'TUK
3    2003-03-25 15:39:59.870 3    Generating schema script for
3    2003-03-25 15:39:59.870 4    Locking published tables whi
3    2003-03-25 15:39:59.870 4    Bulk copying snapshot data f
3    2003-03-25 15:39:59.870 5    Bulk copied snapshot data fo
3    2003-03-25 15:39:59.870 6    Posting snapshot commands in
2    2003-03-25 15:39:59.870 7    A snapshot of 1 article(s) w
```

Затем, после блокировки объектов, которые требуются для получения снимка, агент устанавливает соединение по сети между сервером публикаций и распределительным сервером (если эти серверы эксплуатируются на разных компьютерах) и сохраняет схему для каждой статьи публикации в своем собственном файле, который находится в каталоге снимка на компьютере распределительного сервера (как уже было сказано, местонахождение каталога снимка может быть изменено). Каждый файл схемы имеет расширение .sch и содержит операторы T-SQL, необходимые для создания объекта статьи. Если при добавлении статьи к публикации было решено включить индексы, триггеры или ограничения ссылочной целостности (кластеризованные индексы должны быть включены по умолчанию в программе-мастере Create Publication), то программа Snapshot Agent сохраняет команды T-SQL, необходимые для создания этих объектов, в отдельных файлах, относящихся к каждой статье, и присваивает каждому файлу собственное расширение имени файла.

Затем программа Snapshot Agent записывает в каталог снимка сами данные. Как было указано выше, каждый файл данных записывается в формате BCP (либо в собственной, либо в символьной кодировке, в зависимости от того, предусмотрено ли использование данной публикации для разнотипных подписчиков и предусмотрено ли преобразование публикации с помощью средств DTS) и имеет расширение .bcp. Указанные файлы данных и схемы представляют собой *синхронизационный набор* (так называются файлы, которые регистрируют состояние объекта в том виде, каким он был в данный конкретный момент времени) для каждой статьи в публикации.

Затем агент вводит в таблицу MSrepl_commands распределительной базы данных строки, указывающие местонахождение синхронизационного набора

и задающие все необходимые сценарии, которые должны применяться перед прогоном приложения или после него.

При настройке конфигурации публикации снимка можно задавать определяемые пользователем сценарии T-SQL, предназначенные для прогона до и/или после того, как снимок применяется к некоторому подписчику. Если задается какой-либо из этих сценариев, то программа Snapshot Agent копирует его в каталог снимка, чтобы программа Distribution Agent могла вызвать этот сценарий на выполнение во время применения снимка, и вводит в таблицу MSrepl_commands строку, относящуюся к данному сценарию.

Таблица MSrepl_commands играет ключевую роль и в репликации снимка, и в транзакционной репликации. Эта таблица имеет структуру, показанную в табл. 21.1.

Таблица 21.1. Структура таблицы MSrepl_commands

Имя столбца	Тип столбца
publisher_database_id	int
xact_seqno	varbinary(16)
type	int
article_id	int
originator_id	int
command_id	int
partial_command	bit
command	varbinary(1024)

Для вывода на экран содержимого таблицы MSrepl_commands можно воспользоваться хранимой процедурой sp_browsereplcmds. Эта процедура формирует динамический запрос T-SQL с учетом переданных параметров и вызывает недокументированную хранимую процедуру xp_printstatements для вызова запроса на выполнение (с помощью соединения через петлю обратной связи), затем возвращает таблицу в формате, удобном для чтения.

Наиболее эффективный способ вывода на экран команд, относящихся к снимку, из таблицы MSrepl_commands состоит просто в том, что пользователь сам формирует запрос к данной таблице на языке T-SQL и преобразует при этом тип данных столбца command к требуемому типу. (Но следует учитывать, что такой подход не всегда приемлем, если необходимо вывести на экран команды репликации других типов, например, команды, относящиеся к транзакционной репликации или к репликации слиянием, поэтому, если по указанному поводу возникают какие-либо сомнения, следует использовать хранимую процедуру sp_browsereplcmds). Хотя столбец command определен как относящийся к типу varbinary(1024), этот тип данных можно привести к типу nvarchar(512), что позволяет ознакомиться с содержимым многих команд в формате, удобном для чтения, не прибегая к использованию процедур sp_browsereplcmds и xp_printstatements. Пример запроса, позволяющего ознакомиться с содержимым таблицы MSrepl_commands, приведен в листинге 21.4.

Листинг 21.4. Запрос, позволяющий ознакомиться с содержимым таблицы MSrepl_commands, и полученные результаты

```
USE distribution
GO
SELECT
publisher_database_id,
xact_seqno,
type,
article_id,
originator_id,
command_id,
partial_command,
CAST(command AS nvarchar(512)) AS command
FROM msrepl_commands

(Результаты приведены в сокращенном виде)

pub xact_seqno      type         art orig comm part command
--- -------------- ------------ --- ---- ---- ---- ----------------
1   0x000000070000 -2147483598 1   0    1    0     \\TUK\C$\Program
1   0x000000070000 -2147483597 1   0    2    0
1   0x000000070000 -2147483641 0   0    3    0     \\TUK\C$\Program
1   0x000000070000 -2147483646 1   0    4    0     \\TUK\C$\Program
1   0x000000070000 -2147483646 1   0    5    0     \\TUK\C$\Program
1   0x000000070000 -2147483645 1   0    6    0     sync -t"titles"
1   0x000000070000 -2147483596 1   0    7    0
```

Обычно в таблице MSrepl_commands к каждому снимку относится не одна строка, а несколько. Следует отметить, что в таблице MSrepl_commands каждому снимку присвоено отдельное значение порядкового номера xact_seqno, поэтому такое значение можно использовать, чтобы отличить команды, относящиеся к одному снимку, от команд другого снимка.

Кроме того, программа Snapshot Agent вводит строки в таблицу MSrepl_transactions. Создаваемые при этом записи указывают на синхронизационную задачу подписчика.

После завершения формирования снимка и обновления соответствующих репликационных системных таблиц программа Snapshot Agent освобождает разделяемые блокировки, если таковые были перед этим установлены, и вводит последние записи журнала в таблицу MSsnapshot_history.

С созданными снимками можно ознакомиться в программе Enterprise Manager, щелкнув правой кнопкой мыши на публикации снимка в каталоге Replication\Publications и выбрав в меню команду Explore the Latest Snapshot Folder (Показать последний каталог снимка). Программа Enterprise Manager откроет последний каталог снимка, относящийся к рассматриваемой публикации, в окне Windows Explorer, чтобы пользователь мог ознакомиться с файлами, которые в нем находятся. Такая возможность может оказаться полезной, например, если пользователь намеревается скопировать файлы на другой носитель, например, на компакт-диск, чтобы отправить подписчику.

Задачи программы Distribution Agent

Программа Distribution Agent выполняет задачу по перемещению файлов схемы и файлов данных из каталога снимка в каталог на компьютере подписчика. Выполнение этой задачи начинается с установления соединения между сервером, под управлением которого функционирует эта задача, и распределительным сервером. Если используются подписки, в которых инициатором доставки является распределительный сервер, то программа Distribution Agent обычно работает под управлением распределительного сервера, а если применяются подписки, в которых инициатором доставки является подписчик, то эта программа обычно работает под управлением сервера подписчика.

Затем агент читает содержимое таблиц `MSrepl_commands` и `MSrepl_transactions` для выборки данных о местонахождении синхронизационных наборов, которые должны быть переданы с помощью этого агента, а также команд синхронизации подписчика. Строки, считываемые программой Distribution Agent из указанных таблиц, были введены ранее программой Snapshot Agent.

Наконец, программа Distribution Agent применяет снимок к базе данных подписчика, создавая необходимые объекты и загружая в эти объекты данные, содержащиеся в каждом синхронизационном наборе снимка. В случае необходимости эта программа осуществляет преобразования типов данных для баз данных, отличных от SQL Server, и для подписчиков нижнего уровня. Кроме того, агент синхронизирует все статьи публикации и сохраняет ограничения транзакционной и ссылочной целостности для затронутых объектов в базе данных подписки, при условии, что сервер подписчика обладает способностью выполнять соответствующие операции.

Программа Distribution Agent может применять снимок или при первоначальном создании подписки, или в соответствии с расписанием, определяемым при создании публикации. Если настройка конфигурации снимка выполнена для его дальнейшего применения по расписанию, то следует помнить, что в расписании учитывается системное время того компьютера, на котором работает программа Distribution Agent. Если же агент работает под управлением распределительного сервера, то в расписании учитывается системное время компьютера распределительного сервера. А если агент работает под управлением сервера подписчика (например, как в случае организации работы, когда инициатором доставки данных подписки является сервер подписчика), то в расписании учитывается системное время компьютера подписчика.

Обновляемые подписки

По умолчанию данные, распространяемые с помощью репликации снимка, не подлежат обновлению с помощью базы данных подписчика. Иными словами, по умолчанию нельзя вносить изменения в базу данных подписчика в расчете на то, что эти изменения затем распространятся на базу данных сервера публикаций или на базы данных других подписчиков. Тем не менее публикация снимка может использоваться в таком режиме, чтобы можно было распространять обновления, внесенные

подписчиком. Для этой цели могут применяться следующие три опции: базы данных подписчика с немедленным обновлением, базы данных подписчика с обновлением в порядке очереди и базы данных подписчика с немедленным обновлением и использованием обновления в порядке очереди в качестве резервного режима.

Немедленное обновление

Серверы подписчиков с немедленным обновлением действуют по принципу инициализации распределенной транзакции (охватывающей сервер публикаций) с помощью служб Microsoft Distributed Transaction Coordinator. Обновление осуществляется с применением протокола двухфазной фиксации, согласно которому изменения должны быть либо внесены в базу данных подписчика и в базу данных сервера публикаций, либо не внесены ни в одну из этих баз данных.

Безусловно, для осуществления этого варианта требуется, чтобы подписчик мог получить доступ к серверу публикаций, если возникает необходимость внести какие-либо изменения. Двухфазная фиксация транзакции, охватывающей базу данных сервера публикаций, происходит автоматически (благодаря наличию триггера на реплицируемой таблице), поэтому подписчик вносит изменения в свою таблицу, не выполняя каких-либо специальных подготовительных действий, обусловленных тем фактом, что в действительности незаметно для пользователя инициализируется распределенная транзакция.

После того как пользователь публикует таблицу с помощью средств репликации снимка и разрешает вносить в эту таблицу изменения, программа SQL Server добавляет к таблице столбец уникального идентификатора с именем `msrepl_tran_version`. Этот столбец используется в критериях выборки для последующих операций обновления и удаления. Поскольку данный столбец добавлен к таблице (и при обычных обстоятельствах в него не происходит вставка значений), операторы вставки, применяемые к базе данных сервера публикаций или к базе данных подписчика, должны включать список столбцов.

Программа SQL Server налагает ограничения на то, какие таблицы могут публиковаться с помощью репликации снимка и обновляться в базе данных подписчика. Это должны быть таблицы, на которых уже задан по меньшей мере один уникальный ключ. Безусловно, с помощью средств репликации снимка может быть опубликована таблица, не имеющая первичного ключа, но такая таблица не может быть частью публикации, в которой допускается обновление таблицы подписчиком.

В базе данных подписчика на обновляемых таблицах, полученных с помощью репликации снимка, автоматически создаются три триггера, относящиеся соответственно к операциям вставки, обновления и удаления. Обязательным условием создания каждого из этих триггеров является то, что это должен быть первый триггер, выполняемый применительно к заданной в нем операции. Соблюдение указанного требования обеспечивается благодаря использованию недокументированных строк `OBJECTPROPERTY` для свойств `TriggerInsertOrder`, `TriggerUpdateOrder` и `TriggerDeleteOrder`. Автор не может объяснить, почему для этой цели в триггерах не применяются документированные свойства `ExecIsFirstInsertTrigger`, `ExecIsFirstUpdateTrigger` и `ExecIsFirstDeleteTrigger`, но создается

впечатление, что указанные недокументированные свойства действуют так же, как и документированные.

После проверки в каждом триггере того, что вызов на выполнение произошел в правильной последовательности, триггер вызывает хранимую процедуру для осуществления соответствующего обновления в базе данных сервера публикаций. Триггер передает этой процедуре значения, относящиеся к каждому столбцу в таблице, и структуру отображения, указывающую, какие из этих столбцов были фактически модифицированы. (Такая структура отображения создается с помощью функции COLUMNS_UPDATED.) Если изменения, внесенные той операцией DML, которая активизировала триггер, затронули несколько строк, то триггер открывает курсор на соответствующей таблице, подвергшейся воздействию операций удаления и/или вставки, и вызывает хранимую процедуру по одному разу для каждой модифицированной строки.

В вызываемой хранимой процедуре используются первичный ключ таблицы и глобально уникальный идентификатор, хранящийся в столбце уникального идентификатора, для обеспечения того, чтобы операция обновления была применена к правильно выбранной строке. Кроме того, в хранимой процедуре используется переданная ей структура отображения для определения того, какой столбец (столбцы) подлежит изменению. Для упрощения организации работы предусмотрено, что процедура снова присваивает столбцу прежнее значение, если столбец не подвергался изменению в первоначальной операции DML, выполненной в базе данных подписчика. К сожалению, применение такого подхода приводит к тому, что другие триггеры, заданные на таблице, в которых применяются синтаксические конструкции COLUMNS_UPDATED и IF UPDATE, получают ложную информацию о том, будто изменились значения в каждом столбце таблицы, независимо от того, происходит ли это в действительности.

Информация о модификации данных передается на сервер публикаций с помощью обычных средств связанного сервера SQL Server; при этом в случае необходимости происходит обращение к службе Microsoft Distributed Transaction Coordinator. Если операции внесения изменений в базе данных сервера публикаций завершаются успешно, то триггер разрешает зафиксировать изменения в базе данных подписчика; в противном случае осуществляется откат этих изменений.

В приложениях подписчика необходимо учитывать вероятность того, что операции обновления базы данных сервера публикаций могут окончиться неудачей. Такая ситуация может возникать по многим причинам, включая возможный конфликт с обновлениями, которые вносят другие подписчики или сам сервер публикаций. Но обычно для выхода из этой ситуации достаточно просто подождать несколько секунд и попытаться снова выполнить обновление.

После успешного применения обновления к базе данных сервера публикаций другие подписчики получат данные этого обновления ко времени следующего обновления снимка. Следует отметить, что вариант, рассматриваемый в данном разделе, требует, чтобы в транзакции участвовали только базы данных подписчика, вносящего обновление, и сервера публикаций. Поэтому описанный подход требует меньше ресурсов по сравнению с типичным подходом, основанным на использовании распределенной транзакции, согласно которому все получатели данных должны участвовать в одной и той же распределенной транзакции.

Обновление в порядке очереди

Публикация, для которой разрешено обновление в порядке очереди, применяется во многом аналогично той публикации, для которой разрешено немедленное обновление – в базе данных подписчика активизируются триггеры, которые обеспечивают распространение изменений на базу данных сервера публикаций, а затем эти изменения распространяются по другим подписчикам. Но, безусловно, имеются и различия, определяемые тем, что изменения хранятся в очереди до тех пор, пока не появится возможность отправить данные об изменения в базу данных сервера публикаций. По умолчанию в качестве очереди используется таблица SQL Server, имеющая выразительное имя `MSreplication_queue`, но очередь может быть также реализована с помощью средств MSMQ (Microsoft Message Queuing – средства ведения очередей сообщений Microsoft). Если публикация первоначально была подготовлена для использования таблицы `MSreplication_queue`, то ее можно преобразовать так, чтобы в ней применялись средства MSMQ. Для этого необходимо внести изменения на вкладке **Updatable** диалогового окна **Publication Properties** программы Enterprise Manager.

После того как подписчик вносит изменения в таблицу репликации снимка, которая предназначена для проведения обновлений в порядке очереди, активизируется триггер и вводит запись в очередь, сообщая тем самым об обновлении. Если для ведения очереди используется таблица `MSreplication_queue`, то введение записи в очередь сводится к тому, что в ответ на изменение каждой строки в реплицируемой таблице вводится одна строка в таблицу `MSreplication_queue`. Если же очередь реализована с применением средств MSMQ, то данные об обновлениях сохраняются в очереди на распределительном сервере. А если распределительный сервер не доступен, то средства MSMQ ставят данные об обновлениях в очередь на компьютере подписчика до тех пор, пока не появится возможность установить связь с распределительным сервером.

Чтение данных из очереди и применение хранящихся в очереди данных об изменениях в базе данных сервера публикаций обеспечивает программа Queue Reader Agent. Если используется таблица `MSreplication_queue`, то указанная программа-агент считывает хранящиеся в очереди данные об изменениях непосредственно из таблицы. Если применяются средства MSMQ, то агент считывает изменения из очереди, которая ведется на компьютере распределительного сервера.

В случае обнаружения конфликтов разрешение конфликтов осуществляется в соответствии с правилами разрешения конфликтов, установленными при первоначальном создании публикации. В связи с этим могут создаваться компенсационные команды для отката транзакции в базе данных подписчика, но такие команды передаются только в базу данных подписчика, который был инициатором внесения изменений, но не передаются всем другим подписчикам на данную публикацию.

Немедленное обновление с обновлением в порядке очереди в качестве резервного варианта

Режим обновления, имеющий столь длинное название (немедленное обновление с обновлением в порядке очереди в качестве резервного варианта), фактически

осуществляется не по тому принципу, что обновление в порядке очереди начинает применяться автоматически, если попытка немедленного обновления оканчивается неудачей (например, из-за того, что не удается установить соединение между базой данных подписчика и базой данных сервера публикаций). Дело в том, что пользователь должен разрешить переход на резервный вариант (вариант с использованием очередей) вручную. После этого пользователь сможет переключиться на режим немедленного обновления только после того, как удастся установить соединение между базой данных подписчика и базой данных сервера публикаций и программа Queue Reader Agent применит все хранящиеся в очереди данные об обновлении.

В рассматриваемом варианте обновления не применяется автоматический переход в режим обновления в порядке очереди, поскольку чаще всего нарушения связи между сервером подписчика и сервером публикаций могут быть легко устранены. Поэтому обычно лучше сразу устранить возникшие проблемы связи, чем автоматически переходить в режим записи данных об обновлениях в очередь, организованную в виде таблицы или хранилища MSMQ.

Активизация удаленного агента

Как было указано выше, программа Distribution Agent может эксплуатироваться на сервере, отличном от выбранного в качестве используемого по умолчанию. Такой режим работы может быть осуществлен с помощью средств активизации удаленного агента. Эксплуатация программы Distribution Agent на удаленном компьютере может осуществляться в одном из двух вариантов. Если эта программа эксплуатируется на том же компьютере, что и сервер подписчика, то осуществляет доставку данных подписки, инициируемую подписчиком, а если эксплуатируется на том же компьютере, что и распределительный сервер, то осуществляет доставку данных подписки, инициируемую распределительным сервером. Вместо заданного по умолчанию местонахождения программы Distribution Agent можно указать другое при настройке конфигурации подписки. Дело в том, что иногда возникает необходимость передать часть функций по обработке подписок, доставка которых инициируется распределительным сервером, от распределительного сервера серверу подписчика, особенно если количество доставляемых подписок велико. При этом необходимо учитывать, что по умолчанию, если используется подписка с инициируемой подписчиком доставкой, то программа Distribution Agent работает на компьютере подписчика, а если используется подписка с доставкой, инициируемой распределительным сервером, то по умолчанию предусматривается эксплуатация агента на компьютере распределительного сервера. Это означает, что для уменьшения нагрузки распределительного сервера можно выполнить настройку конфигурации агентов так, чтобы они работали на компьютерах подписчиков.

Для обеспечения эксплуатации агента на другом компьютере с помощью средств активизации удаленного агента используется технология DCOM. Чтобы иметь возможность осуществлять активизацию удаленного агента, пользователь должен иметь правильно заданную конфигурацию прав доступа DCOM. Несоблюдение этого требования приводит к тому, что нарушается синхронизация между

распределительным сервером и сервером подписчика. Следует отметить, что в той конфигурации службы репликации, в которой применяются серверы подписчика на компьютерах с операционной системой Win9x, средства активизации удаленного агента не могут использоваться.

Удаление устаревших данных репликации

Одной из важных задач, которые приходится решать в вычислительной среде, рассчитанной на продолжительную эксплуатацию, является своевременное удаление устаревших данных. Если архитектура программного обеспечения не предоставляет возможности систематически осуществлять удаление из системы устаревших данных и выполнять максимально возможный объем операций обслуживания системы с помощью самой системы, то нагрузка по администрированию такой системы становится весьма значительной. При первоначальном введении в действие средств репликации в программе SQL Server создаются пять стандартных заданий SQL Server Agent, которые обеспечивают бесперебойное проведение репликации, а также позволяют использовать саму систему для удаления накопившихся в ней устаревших данных. Итоговые сведения об указанных заданиях приведены в табл. 21.2.

Таблица 21.2. Задания по сопровождению средств репликации

Задание	Назначение
Удаление устаревших данных из хронологических таблиц агента	Найти и удалить устаревшие данные хронологических таблиц агента репликации базы данных распределительного сервера
Удаление устаревших данных из базы данных распределительного сервера	Удалить устаревшие данные реплицированных транзакций из распределительной базы данных
Удаление устаревших данных подписок с истекшим сроком давности	Найти и удалить устаревшие данные подписок с истекшим сроком давности из опубликованных баз данных
Повторная инициализация подписок, при проверке которых обнаруживались ошибки в данных	Повторно инициализировать все подписки, при проверке которых были обнаружены ошибки в данных
Проверка агентов репликации	Поиск агентов репликации, неудачное завершение работы которых осталось незамеченным (которые перестали активно вносить в журнал хронологические данные)

Каждое из этих заданий технического обслуживания обеспечивает долговременную эксплуатацию средств репликации и позволяет снизить нагрузку по администрированию, возложенную на специалистов, которые управляют репликационной инсталляцией. Например, задание по удалению данных дистрибутивного сервера предусматривает удаление файлов, относящихся к снимку, после применения снимка ко всем подписчикам. Разумеется, если публикация снимка поддерживает анонимных подписчиков или была определена с опцией немедленного создания первого снимка, то должна быть сохранена по меньшей мере одна копия файлов снимка.

Резюме

Средства репликации снимка используются для копирования целых объектов из базы данных сервера публикаций в базу данных сервера подписчика. Безусловно, к снимку могут применяться критерии выборки, но сам принцип репликации снимка не предоставляет возможности отправлять подписчикам инкрементные изменения.

В программе SQL Server репликация снимка реализуется с помощью программ Snapshot Agent и Distribution Agent. Программа Snapshot Agent обеспечивает создание снимка. Снимок состоит из файлов данных BCP и сценариев T-SQL. Программа Distribution Agent берет готовый снимок и применяет его к подписчикам. Функционирование обеих программ-агентов основано на том, что эти программы обращаются к системным таблицам репликации в распределительной базе данных и обновляют эти таблицы.

Вопросы для самопроверки

1. В каком формате программа Snapshot Agent сохраняет данные таблиц, опубликованных как статьи в составе публикации снимка?
2. Подтвердите или опровергните следующее утверждение. Безусловно, программа Snapshot Agent аналогична программе SQL Server Agent, но если разрешена репликация, устанавливается как собственная служба Windows и не использует SQL Server Agent.
3. Какая программа-агент обеспечивает доставку подписчикам снимков, созданных программой Snapshot Agent?
4. Если в публикацию снимка включен объект представления, программа Snapshot Agent выводит данные, возвращаемые представлением, или только схему представления?
5. Какая системная хранимая процедура может использоваться для вывода на внешнее устройство содержимого таблицы `MSrepl_commands`?
6. Подтвердите или опровергните следующее утверждение. Безусловно, агент репликации может быть вызван как терминальное приложение, но фактически после вызова из командной строки не выполняет никакой работы.
7. Назовите одну из пяти задач, устанавливаемых программой SQL Server после первоначального ввода в действие средств репликации, которая позволяет упростить сопровождение системы за счет использования средств самой системы.
8. Назовите два варианта организации работы, в которых программа Snapshot Agent автоматически записывает файлы данных в символьном формате BCP, а не в собственном формате.
9. Подтвердите или опровергните следующее утверждение. Безусловно, можно модифицировать в базе данных подписчика таблицы, которые были получены в составе публикации снимка, но не существует никакого

поддерживаемого системой способа, позволяющего распространять внесенные при этом изменения на базу данных сервера публикаций для последующей передачи другим подписчикам.

10. Столбец `command` таблицы `MSrepl_commands` имеет тип данных `varbinary(1024)`. Что нужно сделать, чтобы преобразовать хранящиеся в этом столбце команды, которые относятся к снимкам, в удобный для чтения текст?
11. Какой столбец добавляется к таблицам репликации снимка, входящим в состав публикации, для которых разрешено выполнять немедленное обновление?
12. Назовите имя таблицы, в которую программа Snapshot Agent вносит информацию о том, какие действия выполняются по мере создания снимка.
13. Дайте определение термина "синхронизационный набор".
14. Таблица, которая используется для ведения очереди обновлений, если в программе SQL Server применяется режим хранения обновлений, выполняемых в порядке очереди, носит содержательное имя. Назовите это имя.
15. Подтвердите или опровергните следующее утверждение. Программа Snapshot Agent используется также в транзакционной репликации. С помощью этой программы создается начальный снимок данных, который публикуется для транзакционный репликации.
16. Какой параметр командной строки программы-агента позволяет регламентировать степень детализации вывода программы Snapshot Agent?
17. Где находится очередь, создаваемая средствами MSMQ, если эти средства используются для ведения очереди обновлений?
18. Подтвердите или опровергните следующее утверждение. Для каждой публикации снимка предусмотрено использование отдельного экземпляра программы Snapshot Agent.
19. Подтвердите или опровергните следующее утверждение. Если настройка конфигурации средств публикации снимка выполнена в целях распространения подписки по расписанию, то дата и время применения снимка всегда определяются с учетом системного времени компьютера распределительного сервера, а не компьютера подписчика.
20. Подтвердите или опровергните следующее утверждение. Если публикация снимка применяется в сочетании с опцией немедленного обновления, то с помощью триггеров инициализируются операции двухфазной фиксации.

ГЛАВА 22

Транзакционная репликация

Транзакционная репликация представляет собой такую разновидность средств репликации SQL Server, которая находит гораздо более широкое распространение по сравнению с другими, поскольку предоставляет богатый выбор функциональных возможностей в сочетании с достаточно удобной настройкой конфигурации и простым администрированием. Транзакционная репликация обладает более развитыми функциональными средствами по сравнению с репликацией снимков и вместе с тем обеспечивает более легкую настройку конфигурации и более простое сопровождение, чем репликация путем слияния.

Автор часто замечал, что некоторые пользователи, принимая решение о том, какой тип репликации ввести в действие, допускают ошибку, не учитывая то, что немедленное обновление подписок и обновление подписок в порядке очереди можно обеспечить и с помощью репликации снимка, и с помощью транзакционной репликации. Дело в том, что многие считают, будто репликация путем слияния представляет собой единственный тип репликации, который поддерживает двунаправленную репликацию данных. Но это утверждение далеко от истины, что подтверждают приведенные в главе 21 сведения об обновлении баз данных подписчиков и о репликации снимков. Транзакционная репликация предоставляет такие же возможности по обновлению, что и репликация снимков. Кроме того, транзакционная репликация позволяет воспользоваться моделью публикации, применимой в таких ситуациях, когда необходимо регулярно использовать инкрементные обновления, передаваемые в базы данных подписчиков из базы данных сервера публикаций.

Краткий обзор

В программе SQL Server транзакционная репликация реализована с помощью программ Snapshot Agent, Log Reader Agent и Distribution Agent. Как и при репликации снимка, программа Snapshot Agent подготавливает первоначальную версию снимка транзакционной публикации. Программа Log Reader Agent просматривает журнал транзакций сервера публикаций и обнаруживает изменения, внесенные в данные после получения снимка, а затем регистрирует эти изменения в базе данных распределительного сервера. (Как будет описано ниже, при использовании параллельной обработки снимка программа Log Reader Agent может фактически сохранять в базе данных распределительного сервера изменения, внесенные

в ходе создания снимка.) Программа Distribution Agent считывает изменения, зарегистрированные в базе данных распределительного сервера и применяет их к базам данных подписчиков.

Таблица MSrepl_commands

Внесение каждого изменения в опубликованную таблицу вынуждает программу Log Reader Agent записать по меньшей мере одну строку в таблицу MSrepl_commands. В отличие от репликации снимка, чтобы ознакомиться с содержимым таблицы MSrepl_commands, невозможно просто привести тип данных столбца command этой таблицы к типу nvarchar(512) и получить удобный для чтения текст команд транзакционной репликации, поэтому для этой цели необходимо использовать хранимую процедуру sp_browsereplcmds.

По аналогии с тем, что выполнение одной команды T-SQL может повлечь за собой внесение нескольких записей в журнал транзакций, единственная команда T-SQL может вынудить программу Log Reader Agent внести несколько записей в таблицу MSrepl_commands, и это следует учитывать в своей работе. Если, допустим, выполняется оператор UPDATE языка T-SQL, который воздействует на 10 тыс. строк, то в таблице MSrepl_commands появляется по меньшей мере 10 тыс. записей. Такое же утверждение является истинным применительно к командам DELETE — если в одном операторе DELETE языка T-SQL обусловлено удаление 10 тыс. строк, то программа Log Reader Agent вносит по меньшей мере 10 тыс. строк в таблицу MSrepl_commands. Модификация любой строки в статье таблицы транзакционной публикации с помощью команды T-SQL приводит к появлению не менее одной новой записи в таблице MSrepl_commands.

Если же в статье таблицы транзакционной публикации происходит модификация данных в столбце с ограничением уникальности, это приводит к записи в таблицу MSrepl_commands по меньшей мере двух строк в расчете на каждую модифицированную строку: в одной строке содержится команда DELETE или вызов хранимой процедуры, а во второй строке — команда INSERT или вызов хранимой процедуры. Применительно к транзакционной репликации в качестве столбца с ограничением уникальности рассматривается любой столбец, который входит в состав ключа уникального индекса или ключа кластеризованного индекса, даже если кластеризованный индекс не является уникальным кластеризованным индексом. (Программа SQL Server добавляет специальный "унификатор" к ключам неуникального кластеризованного индекса, чтобы сделать эти ключи уникальными и позволить использовать в качестве локаторов строк в некластеризованных индексах.) К записи в таблицу MSrepl_commands не меньше двух строк в расчете на каждую модифицированную строку приводит также обновление индексированного представления или обновление таблицы, на которой основано индексированное представление.

Исходя из описанного выше, можно сделать вывод, что даже единственный оператор DML, применяемый к таблице, которая была опубликована с помощью средств транзакционной репликации, может привести к выполнению значительного объема операций с журналом не только в первоначальной базе данных,

но и в базе данных распределительного сервера, а также в целевых базах данных подписчиков. По этой причине транзакционную репликацию не следует использовать, если требуется репликация всей базы данных, а основной объем операций модификации данных осуществляется на другом компьютере. В подобном сценарии лучше использовать какой-то другой способ репликации, например, доставку журнала.

ПРИМЕЧАНИЕ. Настройку конфигурации программного обеспечения можно выполнить таким образом, чтобы средства доставки журнала и средства репликации действовали совместно. В частности, настройка конфигурации средств транзакционной репликации может быть осуществлена так, чтобы эти средства взаимодействовали со средствами доставки журнала в целях предоставления доступа к серверу горячего резервирования в случае отказа сервера публикаций.

Для интеграции средств доставки журнала и средств транзакционной репликации могут использоваться два варианта: синхронный и полусинхронный режим. Чтобы ввести в действие синхронный режим, необходимо обеспечить синхронизацию с базой данных публикаций с помощью хранимой процедуры `sp_replicationdboption`, предусмотрев применение резервного копирования. После перехода в синхронный режим программа Log Reader Agent игнорирует модифицированные записи в журнале транзакций до тех пор, пока не будет выполнено резервное копирование этих записей. Подобная организация работы гарантирует, что ни один сервер подписчика не будет опережать распределительный сервер по темпам внесения изменений. Благодаря этому в случае отказа сервера публикаций можно быть уверенным в том, что ни на одном сервере подписчика нет данных, отсутствующих на сервере резервного копирования. Естественно, что при этом увеличивается задержка, которая отсчитывается с момента внесения изменений на сервере публикаций и до того момента, как это изменение распространяется по серверам подписчиков, и вместо обычного значения, составляющего несколько секунд, может возрасти до нескольких минут.

Чтобы ввести в действие полусинхронный режим, достаточно просто выполнить настройку конфигурации средств доставки журнала, как и в обычных условиях, а также предоставить средствам транзакционной репликации возможность действовать по таким же принципам, которые предусмотрены по умолчанию. В указанном режиме допускается выход сервера резервного копирования и серверов подписчиков из синхронизации, но задержка между внесением изменений в базу данных сервера публикаций и распространением этих изменений на базы данных подписчиков перестает быть связанной с периодичностью доставки журнала и обычно составляет примерно от двух до десяти минут.

Процедура `sp_replcmds`

Расширенная процедура `sp_replcmds` (реализованная внутри программы SQL Server) используется в программе Log Reader Agent для выборки записей журнала, сформированных в результате выполнения операторов DML и записанных в журнал транзакций опубликованной базы данных. Для каждой базы данных сервера публикаций, участвующей в транзакционной репликации, должна быть предусмотрена одна и только одна программа Log Reader Agent, независимо от того, какое количество транзакционных публикаций содержится в этой базе данных.

Первый клиент, вызывающий процедуру sp_replcmds применительно к некоторой базе данных, рассматривается как *средство чтения журнала* для этой базы данных до тех пор, пока не произойдет отключение этого клиента. Все другие клиенты, предпринимающие попытку вызвать на выполнение процедуру sp_replcmds до того, как произойдет отключение первого клиента, получают сообщение об ошибке, формулирующееся следующим образом: "Another log reader is replicating the database" (В этой базе данных для репликации уже применяется другое средство чтения журнала).

Одна из причин, по которым для каждой базы данных разрешено использовать только одну программу Log Reader Agent, состоит в том, что выполнение операций просмотра журнала для поиска изменений может отрицательно повлиять на производительность. Каждый вызов процедуры sp_replcmds в программе Log Reader Agent приводит к тому, что код средства чтения журнала, функционирующий в рамках процесса SQL Server, применяется для просмотра журнала транзакций опубликованной базы данных и выявления изменений, которые требуют репликации. В ходе выполнения этой операции в программе Log Reader Agent вместо последовательного способа чтения, который обычно используется в программе SQL Server для доступа к журналу, происходит в большей степени случайный доступ. Безусловно, сервер вносит новые записи в журнал транзакций по мере выполнения изменений в базе данных в конец журнала, но в программе Log Reader Agent может осуществляться чтение из другого участка журнала для последующей записи команд репликации в таблицы MSrepl_commands и MSrepl_transactions. Из-за того что две программы одновременно обращаются к разным частям журнала транзакций, может возникать конкуренция за ресурсы и в целом происходит уменьшение производительности сервера.

Кэш статей

В программе SQL Server ведется глобальный кэш метаданных статей, называемый *кэшем статей*. В этом кэше хранятся метаданные из таблиц sysarticles и syscolumns, относящиеся к каждой реплицируемой статье. Каждый раз, когда при выполнении кода средства чтения журнала в сервере требуются метаданные, относящиеся к конкретной статье, происходит обращение к указанному кэшу. Если метаданные статьи уже находятся в кэше, то в коде средства чтения журнала осуществляется выборка требуемой информации из кэша. Если же метаданные статьи в кэше отсутствуют, то в коде происходит непосредственное обращение к таблицам sysarticles и syscolumns, выборка необходимой информации, а затем добавление этой информации к кэшу.

Для проверки наличия кэша статей можно воспользоваться недокументированной командой DBCC RESOURCE. Дополнительная информация о команде DBCC RESOURCE приведена в предыдущих книгах автора, *The Guru's Guide to Transact-SQL* и *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML*. Эта команда возвращает адрес в пространстве серверного процесса, по которому находится кэш статей. Чтобы самостоятельно узнать этот адрес, выполните следующие команды из программы Query Analyzer:

```
DBCC TRACEON(3604) -- Перенаправить выходные данные в клиентскую
                   -- программу
DBCC RESOURCE
DBCC TRACEOFF(3604)
```

После возврата управления из команды DBCC RESOURCE выполните поиск в ее выходных данных строки article_cache. Это — имя элемента глобальной структуры resource, в котором содержится адрес кэша репликационной статьи.

Элемент article_cache глобальной структуры resource содержит указатель на связный список реплицированных объектов базы данных. В каждом элементе этого списка имеется переменная, указывающая на то, какая структура SRV_PROC относится к текущему средству чтения журнала для соответствующей базы данных. После вызова в некотором соединении процедуры sp_replcmds сервер присваивает указатель SRV_PROC на средство чтения журнала соответствующему элементу в списке объектов базы данных (при условии, что это еще не сделано применительно к другому соединению). До тех пор пока значение указателя SRV_PROC остается ненулевым, в другом соединении невозможно успешно выполнить процедуру sp_replcmds по отношению к конкретной базе данных. А после закрытия соединения, которое в настоящее время обозначено как средство чтения журнала базы данных, обработчик события, предшествующего закрытию соединения, службы ODS очищает значение указателя SRV_PROC в соответствующем объекте базы данных, что позволяет приступить к выполнению кода средства чтения журнала в другом соединении.

Параметры процедуры sp_replcmds

Параметр @maxtrans указывает количество транзакций, информация о которых должна быть возвращена процедурой sp_replcmds. Чтобы откорректировать значение, передаваемое программой Log Reader Agent в параметре @maxtrans, требуется создать определяемый пользователем профиль агента. Для этого необходимо выполнить описанные ниже действия.

1. Запустите программу Enterprise Manager и разверните узел Replication Monitor в окне распределительного сервера репликации.
2. Разверните узел Agents и щелкните на обозначении Log Reader Agents.
3. Найдите обозначение требуемой программы Log Reader Agent в списке, находящемся справа, и дважды щелкните на этом обозначении.
4. Щелкните на кнопке Agent Profile, а затем — на обозначении New Profile.
5. Присвойте имя новому профилю, затем задайте в качестве параметра ReadBatchSize значение, которое требуется передать в параметре @maxtrans. В частности, одним из типичных этапов процедуры поиска неисправностей является присваивание этому параметру значения 1 (если параметр @maxtrans не задан, то фактически по умолчанию принимает значение 1).
6. Возвратитесь к диалоговому окну Agent Profile, щелкните на кнопке переключателя рядом с новым профилем, чтобы его выбрать, затем щелкните на кнопке OK.

Следует отметить, что в оперативной документации Books Online указан только один параметр для процедуры sp_replcmds, но фактически эта процедура поддерживает несколько параметров. Перехват трассировки Profiler в ходе выполнения программы Log Reader Agent показывает, что фактически кроме документированного параметра @maxtrans передаются два дополнительных целочисленных параметра. Поверхностная проверка, проведенная автором, показывает, что значения этих параметров, по-видимому, никогда не изменяются (всегда остаются равными соответственно 0 и -1), но важно то, что результаты трассировки содержат и более значимые сведения.

В частности, заслуживает внимания то, что значения параметров, наблюдаемые в трассировке Profiler, отличаются от значений, показанных в выводе программы Log Reader Agent. Чтобы убедиться в этом самостоятельно, создайте публикацию транзакционной репликации и разрешите использование выходного файла для программы Log Reader Agent. (Задайте параметр командной строки -Output *filename*, чтобы указать выходной файл агента, как было описано в главе 21.) В выходном файле должны присутствовать примерно такие записи, относящиеся к вызовам процедуры sp_replcmds:

```
Publisher: {call sp_replcmds (500, 0)}
```

А в трассировке Profiler тот же вызов приводит к получению следующих данных:

```
RPC:Starting exec sp_replcmds 500, 0, -1
```

На этом основании, безусловно, возникает вопрос, какой смысл имеет значение -1, передаваемое в качестве третьего параметра. Проверка, проведенная автором, показала, что значение этого параметра изменяется в зависимости от установленного выпуска программы SQL Server. В выпусках, предшествующих SQL Server 2000 со служебным пакетом Service Pack 1, этот параметр не передавался, поэтому, учитывая то, что фактическое значение параметра, скорее всего, не изменялось, можно сделать вывод, что параметр -1 указывает на применение выпуска программы SQL Server, который по меньшей мере относится к выпуску SQL Server 2000 Service Pack 1. Но независимо от того, показывает ли трассировка Profiler, что передается этот параметр, в выводе агента он регистрируется со значением 0 или вообще не регистрируется.

В процедуру sp_replcmds могут также передаваться другие параметры в зависимости от обстоятельств. Но значения этих параметров не столь важны, если используется транзакционная репликация, поскольку при этом не приходится непосредственно вызывать процедуру sp_replcmds.

Процедура sp_repldone

После завершения в программе Log Reader Agent вызова процедуры sp_replcmds и внесения новых записей в таблицы MSrepl_commands и MSrepl_transactions эта программа вызывает процедуру sp_repldone для указания на, что успешно выполнена репликация заданных записей журнала (такая информация требуется распределительному серверу). В результате вызова

процедуры sp_repldone программа SQL Server получает возможность удалять записи журнала по мере необходимости. (Записи журнала, относящиеся к статьям транзакционной репликации, не могут быть удалены до тех пор, пока не будет выполнена успешная репликация статей на распределительном сервере.)

Необходимо учитывать, что процедуру sp_repldone нельзя вызывать на выполнение вручную. Попытка применить такой вызов может привести к нарушению последовательности и непротиворечивости реплицируемых транзакций. Попытка вызова процедуры sp_repldone вручную может быть оправдана только в случае возникновения аварийной ситуации, связанной с переполнением журнала транзакций из-за прекращения процесса удаления реплицируемых транзакций (если достоверно известно, что данные этих транзакций уже отправлены на распределительный сервер), но предпринимать такое действие следует только после получения соответствующих указаний от компании Microsoft или от компании-партнера Microsoft, которая занимается технической поддержкой. Вместо этого право принимать решение о том, когда следует вызывать на выполнение процедуру sp_repldone, необходимо предоставить программе Log Reader Agent. Автор упоминает в данной главе процедуру sp_repldone, чтобы можно было понять, почему сведения об этой процедуре иногда обнаруживаются в выводе программы Log Reader Agent и в трассировках Profiler.

Хранимые процедуры обновления

Формат команд, фактически записываемых программой Log Reader Agent в таблицу MSrepl_commands, зависит от того, как осуществлена настройка конфигурации конкретной статьи. Предусмотрена возможность определять способ передачи команд DML подписчикам отдельно для каждой статьи таблицы и публикации. Для этого необходимо щелкнуть на кнопке со знаком троеточия рядом с обозначением статьи таблицы на вкладке Articles диалогового окна Publication Properties. Для настройки конфигурации формата команд, передаваемых подписчику, выберите вкладку Commands в диалоговом окне Table Article Properties. При этом можно воспользоваться одним из четырех описанных ниже вариантов.

1. Можно очистить флажки Replace..., чтобы вынудить программу Log Reader записывать в таблицу MSrepl_commands исключительно операторы INSERT, UPDATE и DELETE. Но если публикация предназначена только для подписчиков, использующих программу SQL Server, то лучший и более эффективный способ состоит в применении хранимых процедур вместо указанных операторов. Для этого в программе SQL Server предусмотрены три следующих варианта (варианты 2–4).
2. Если публикация рассчитана исключительно на подписчиков SQL Server, то можно заметить, что флажки, регламентирующие использование хранимых процедур вместо команда INSERT, UPDATE и DELETE, отмечены по умолчанию. Для указания того, как эти процедуры будут вызываться программой Distribution Agent, в сервере подписчика предоставляются три возможности,

связанные с применением синтаксических конструкций CALL, MCALL и XCALL. Каждая из этих конструкций обусловливает использование отдельного соглашения о вызовах, поддерживающего немного отличный от других механизм вызова хранимых процедур DML, что позволяет взвешенно подходить к проектированию соответствующих топологий репликации. По умолчанию процедуры вставки и удаления вызываются с помощью синтаксических конструкций CALL, а процедуры обновления — с помощью синтаксических конструкций MCALL. Что касается вставки, то в синтаксической конструкции CALL может быть определено, что процедура принимает значения для каждого из столбцов таблицы в качестве параметров. А что касается удаления, то вызов CALL указывает на то, что процедура принимает в качестве параметров значения для столбца (столбцов) первичного ключа таблицы. Если речь идет об обновлении, то в вызове MCALL предусмотрено, что процедура должна принимать новые значения для всех столбцов статьи, а за ними следуют первоначальные значения для столбца (столбцов) первичного ключа таблицы.

3. Как было указано выше, в процедуре обновления для публикаций, относящихся только к подписчикам SQL Server, по умолчанию используется синтаксическая конструкция MCALL. В синтаксической конструкции MCALL предусмотрено, что процедуре обновления передаются обновляемые значения для всех столбцов в статье, за которыми следуют первоначальные значения столбца (столбцов) первичного ключа таблицы, и структура отображения, указывающая, какие столбцы были фактически изменены. Поскольку указаны изменяющиеся столбцы, программа SQL Server позволяет исключить необходимость снова записывать в хранимой процедуре неизменившиеся значения в базу данных, что могло бы привести к ненужной выработке записей журнала, к вызову кода ограничений и активизации триггеров.
4. Последний вариант вызова указанных процедур состоит в использовании синтаксической конструкции XCALL, которая определяет, что в процедуру обновления передается первоначальное значение каждого столбца в статье, за которым следует новое значение для каждого столбца. Поскольку имеются первоначальные значения столбцов, то появляется возможность более легко реализовать оптимистическое управление распараллеливанием, в котором обнаруживаются изменения в статье, передаваемой подписчику, внесенные другими пользователями.

ПРИМЕЧАНИЕ. При использовании вариантов, предусматривающих вызов процедур, необходимо учитывать, что в вызовах хранимых процедур, вырабатываемых программой Log Reader Agent, могут встретиться неправильно сформированные имена процедур. Ко времени написания данной книги попытки разрешить программе Log Reader Agent формировать синтаксические конструкции CALL, MCALL или XCALL для статей таблиц, имена которых содержат пробелы, приводили к появлению неправильно сформированных команд в таблице MSrepl_commands. Это связано с тем, что имена процедур обновления создаются на основе имен статей таблицы, а программа Log Reader Agent не заключает должным образом имена процедур

в кавычки или в квадратные скобки, поэтому в таблице MSrepl_commands появляются примерно такие синтаксические конструкции:

```
{CALL sp_MSupd_Order Details (NULL,NULL,NULL,24,NULL,10248,11,0x08)}
```

Обратите внимание на наличие пробела между фрагментами имени sp_MSupd_Order и Details. Правильной синтаксической конструкцией является следующая:

```
{CALL [sp_MSupd_Order Details] (NULL,NULL,NULL,48,NULL,10248,11,0x08)}
```

Чтобы устранить указанные проблемы, следует вводить квадратные скобки или кавычки вокруг имен хранимых процедур в диалоговом окне Table Article Properties при определении используемых соглашений о вызове.

Независимо от того, используется ли синтаксическая конструкция CALL, MCALL или XCALL, программа Distribution Agent передает соответствующие вызовы хранимых процедур каждому подписчику в виде события RPC. Соглашения о вызовах MCALL и XCALL, по суги, представляют собой просто варианты синтаксической конструкции CALL интерфейса ODBC, а это позволяет вызывать хранимые процедуры с помощью средств RPC программы SQL Server (описание средств RPC приведено в главе 6). В вызовах MCALL и XCALL просто предусмотрены определения того, какие типы параметров передаются в процедуру DML; в конечном итоге оба эти вызова приводят к тому, что процедура вызывается с использованием синтаксической конструкции средств CALL RPC интерфейса ODBC.

Параллельная обработка снимка

По умолчанию на то время, как осуществляется создание первоначального снимка, программа SQL Server устанавливает разделяемые блокировки на статьях таблицы в транзакционной публикации. Благодаря этому гарантируется транзакционная непротиворечивость снимка, но исключается возможность вносить обновления в таблицы в течение времени создания снимка.

Транзакционная репликация предоставляет возможность обойти это ограничение благодаря использованию так называемой *параллельной обработки снимка*. Следует отметить, что эта опция доступна, только если публикация распространяется по подписчикам, использующим SQL Server 7.0 и более поздние версии. Если для транзакционной публикации разрешена параллельная обработка снимка, то программ Snapshot Agent, как всегда, устанавливает разделяемую блокировку на статьях таблицы в публикации. Затем эта программа вносит в журнал транзакций запись, указывающую, что началось создание транзакционного снимка, и освобождает разделяемые блокировки, установленные ранее на статьях таблицы. С этого момента продолжается формирование снимка и появляется возможность вносить изменения в опубликованные таблицы. В обычных условиях разделяемые блокировки, устанавливаемые во время параллельной обработки снимка, являются весьма непродолжительными и, как правило, сохраняются в течение всего лишь нескольких секунд.

После завершения формирования снимка программа Snapshot Agent вносит в журнал транзакций вторую запись, которая указывает, что текущая операция закончена. После этого программа Log Reader Agent читает из журнала данные

о транзакциях, которые были выполнены с того момента, как началось формирование снимка, и до того момента, как формирование закончилось, и записывает в базу данных распределительного сервера такие команды, с помощью которых сформированный снимок можно согласовать с предыдущим состоянием таблицы.

Естественно, для этого требуется, чтобы программа Log Reader Agent участвовала в завершении процесса формирования снимка, поскольку эта программа обеспечивает сбор данных об изменениях, происходящих на протяжении формирования снимка, и вносит сведения об этих изменениях в базу данных распределительного сервера. Для того чтобы снимок оставался непротиворечивым с транзакционной точки зрения, программа Log Reader Agent должна обнаруживать изменения, происходящие в период от начала и до конца формирования снимка, а также записывать эти сведения в базу данных распределительного сервера. В действительности, если программа Log Reader Agent не может эксплуатироваться, то программа Distribution Agent теряет способность применять снимок к подписчикам и возвращает сообщение об ошибке, указывающее, что снимок недоступен.

В то время как программа Distribution Agent применяет созданный параллельно обрабатываемый снимок к подписчику, происходит ввод в действие не только первоначальных файлов снимка, сформированных программой Snapshot Agent, но и команд, записанных программой Log Reader Agent в базу данных распределительного сервера в целях согласования снимка с реальным состоянием реплицируемых данных на момент начала формирования снимка. На то время как программа Distribution Agent выполняет указанные действия, устанавливаются блокировки на таблицах, входящих в состав публикации, которая хранится в базе данных подписчика, чтобы можно было обеспечить транзакционную непротиворечивость этих таблиц в том состоянии, какое они примут после применения снимка. Происходящий при этом процесс согласования весьма напоминает процесс восстановления, через который проходит база данных сразу после запуска программы SQL Server.

Следует отметить, что нельзя выполнять команду UPDATETEXT на столбце в таблице, для которой формируется параллельно обрабатываемый снимок. При попытке выполнить указанную команду активизируется ошибка 7137, сообщение о которой гласит: "UPDATETEXT is not allowed because the column is being processed by a concurrent snapshot..." (Операция UPDATETEXT не разрешена, поскольку для обработки данного столбца применяется параллельно обрабатываемый снимок...). После завершения создания снимка снова появляется возможность выполнять операторы UPDATETEXT применительно к такому столбцу.

Параллельно обрабатываемый снимок состоит из ряда файлов BCP, за которыми следуют сценарии с операторами INSERT и DELETE, поэтому в ходе применения снимка база данных подписчика может находиться в несогласованном состоянии. Если на таблицах базы данных подписчика, обновляемых с помощью снимка, определены ограничения, то указанные ограничения могут ошибочно активизировать сообщения о нарушении ограничений целостности или бизнес-правил в ходе применения снимка. Хотя текущий снимок применяется как отдельная транзакция (что исключает для пользователей вероятность получения данных, находящихся в несогласованном состоянии), код алгоритмов проверки бизнес-правил все еще может обнаруживать фиктивные нарушения правил. Чтобы

исключить возможность возникновения подобной ситуации, необходимо задавать опцию NOT FOR REPLICATION для всех ограничений и столбцов идентификаторов в базе данных подписчика, если на эти столбцы может оказывать воздействие применение текущего снимка. Следует отметить, что опцию NOT FOR REPLICATION не нужно задавать для ограничений внешнего ключа, ограничений проверки и триггеров, поскольку действие этих объектов отменяется на время применения параллельно обрабатываемого снимка и возобновляется после этого.

Кроме того, во время формирования параллельно обрабатываемого снимка следует ожидать, что производительность сервера публикаций несколько снизится. Дело в том, что заметное влияние на производительность оказывают не только сами издержки формирования снимка, но и издержки, связанные с чтением в программе Log Reader Agent данных об изменениях из журнала транзакций и записью этих данных в базу данных распределительного сервера, которые возникают, даже если разрешены обновления опубликованных таблиц, особенно если сервер публикаций и распределительный сервер эксплуатируются на одном и том же компьютере. Задания по получению параллельно обрабатываемых снимков следует планировать на выполнение в периоды низкой активности системы (например, ночью или в течение нерабочих часов).

Следует отметить, что репликация может окончиться неудачей, если разрешена параллельная обработка снимка для публикации, содержащей таблицу с первичным ключом или ограничением уникальности, не содержащимся в кластеризованном индексе, а во время обработки снимка происходят изменения в ключе кластеризации. Для предотвращения подобной ситуации не следует разрешать использование параллельной обработки снимка в публикациях, содержащих таблицу с первичным ключом или ограничением уникальности, не включенным в кластеризованный индекс таблицы, если нет полной уверенности в том, что столбцы кластеризованного индекса не будут подвергаться модификациям во время обработки снимка.

Если публикация распространяется по подписчикам, эксплуатирующим версию SQL Server 7.0, то распределительный сервер должен работать под управлением SQL Server 2000 или более поздней версии, а для использования параллельно обрабатываемых снимков требуется, чтобы в подписках с доставкой, которая активизируется сервером, доставка данных осуществлялась по инициативе распределительного сервера. Если применяется подписка с доставкой, активизируемой сервером, то программа Distribution Agent работает на том же компьютере, где эксплуатируется распределительный сервер, а использование подписки с доставкой, активизируемой подписчиком, приводит к тому, что программа Distribution Agent работает под управлением подписчика, эксплуатирующего версию SQL Server 7.0, в которой параллельная обработка снимка не предусмотрена.

Очевидно, что при использовании параллельной обработки снимка приходится учитывать множество ограничений и предостережений, поэтому данный режим не разрешен по умолчанию. Однако пользователь имеет возможность отредактировать свойства транзакционной публикации после ее создания и разрешить применение параллельной обработки снимка с помощью вкладки Snapshot диалогового окна Publication Properties.

Обновляемые подписки

По умолчанию данные, распространяемые с помощью транзакционной репликации, не подлежат обновлению с помощью базы данных подписчика. Иными словами, по умолчанию нельзя вносить изменения в базу данных подписчика с тем, чтобы эти изменения затем распространялись на базу данных сервера публикаций или на базы данных других подписчиков. Тем не менее транзакционная публикация может использоваться в таком режиме, чтобы можно было распространять обновления, внесенные подписчиком. Для этой цели могут применяться следующие три опции: базы данных подписчика, обеспечивающие немедленное обновление, базы данных подписчика с обновлением в порядке очереди и базы данных подписчика с немедленным обновлением и использованием обновления в порядке очереди в качестве резервного режима.

Немедленное обновление

Серверы подписчиков с немедленным обновлением действуют по принципу инициализации распределенной транзакции, охватывающей сервер публикаций, с помощью служб Microsoft Distributed Transaction Coordinator. Обновление осуществляется с применением протокола двухфазной фиксации, согласно которому изменения должны быть либо внесены и в базу данных подписчика, и в базу данных сервера публикаций, либо не внесены ни в одну из этих баз данных.

Безусловно, для осуществления этого варианта требуется, чтобы подписчик мог получить доступ к серверу публикаций, если возникает необходимость внести какие-либо изменения. Двухфазная фиксация транзакции, охватывающей базу данных сервера публикаций, происходит автоматически (благодаря наличию триггера на реплицируемой таблице), поэтому подписчик вносит изменения в свою таблицу, не выполняя каких-либо специальных подготовительных действий, обусловленных тем фактом, что в действительности незаметно для пользователя инициализируется распределенная транзакция.

После того как пользователь публикует таблицу с помощью средств транзакционной репликации и разрешает вносить в эту таблицу изменения, программа SQL Server добавляет к таблице столбец уникального идентификатора с именем `msrepl_tran_version`. Этот столбец используется в критериях выборки для последующих операций обновления и удаления. Поскольку данный столбец добавлен к таблице (и при обычных обстоятельствах в него не происходит вставка значений), операторы вставки, применяемые к базе данных сервера публикаций или к базе данных подписчика, должны включать список столбцов.

В базе данных подписчика на обновляемых таблицах, полученных с помощью транзакционной репликации, автоматически создаются три триггера, относящиеся соответственно к операциям вставки, обновления и удаления. Обязательным условием создания каждого из этих триггеров является то, что это должен быть первый триггер, выполняемый применительно к заданной в нем операции. Соблюдение указанного требования обеспечивается благодаря использованию

недокументированных имен TriggerInsertOrder, TriggerUpdateOrder и TriggerDeleteOrder для свойства OBJECTPROPERTY. Автор не может объяснить, почему для этой цели в триггерах не применяются документированные свойства ExecIsFirstInsertTrigger, ExecIsFirstUpdateTrigger и ExecIsFirstDeleteTrigger, но создается впечатление, что указанные недокументированные свойства действуют так же, как и документированные.

После проверки в каждом триггере того, что вызов на выполнение произошел в правильной последовательности, триггер вызывает хранимую процедуру для осуществления соответствующего обновления в базе данных сервера публикаций. Триггер передает этой процедуре значения, относящиеся к каждому столбцу в таблице, и структуру отображения, указывающую, какие из этих столбцов были фактически модифицированы. (Такая структура отображения создается с помощью функции COLUMNS_UPDATED.) Если изменения, внесенные той операцией DML, которая активизировала триггер, затронули несколько строк, то триггер открывает курсор на соответствующей таблице, подвергшейся воздействию операций удаления и/или вставки, и вызывает хранимую процедуру по одному разу для каждой модифицированной строки.

В вызываемой хранимой процедуре используются первичный ключ таблицы и глобально уникальный идентификатор GUID, хранящийся в столбце уникального идентификатора, для обеспечения того, чтобы операция обновления была применена к правильно выбранной строке. Кроме того, в хранимой процедуре используется переданная ей структура отображения для определения того, какой столбец (столбцы) подлежит изменению. Для упрощения организации работы предусмотрено, что процедура снова присваивает столбцу прежнее значение, если столбец не подвергался изменению в первоначальной операции DML, выполненной в базе данных подписчика. К сожалению, применение такого подхода приводит к тому, что другие триггеры, заданные на таблице, в которых применяются синтаксические конструкции COLUMNS_UPDATED и IF UPDATE, получают ложную информацию о том, будто изменились значения в каждом столбце таблицы, независимо от того, происходит ли это в действительности.

Информация о модификации данных передается на сервер публикаций с помощью обычных средств связанного сервера SQL Server; при этом в случае необходимости происходит обращение к службе Microsoft Distributed Transaction Coordinator. Если операции внесения изменений в базе данных сервера публикаций завершаются успешно, то триггер разрешает зафиксировать изменения в базе данных подписчика; в противном случае осуществляется откат этих изменений.

В приложениях подписчика необходимо учитывать вероятность того, что операции обновления базы данных сервера публикаций могут окончиться неудачей. Такая ситуация может возникать по многим причинам, включая возможный конфликт с обновлениями, которые вносят другие подписчики или сам сервер публикаций. Но обычно для выхода из этой ситуации достаточно просто подождать несколько секунд и попытаться снова выполнить обновление.

После успешного применения обновления к базе данных сервера публикаций другие подписчики получат данные этого обновления ко времени следующего обновления снимка. Следует отметить, что вариант, рассматриваемый в данном

разделе, требует, чтобы в транзакции участвовали только базы данных подписчика, вносящего обновление, и сервера публикаций. Поэтому описанный подход требует меньше ресурсов по сравнению с типичным подходом, основанным на использовании распределенной транзакции, согласно которому все получатели данных должны участвовать в одной и той же распределенной транзакции.

Обновление в порядке очереди

Публикация, для которой разрешено обновление в порядке очереди, применяется во многом аналогично той публикации, для которой разрешено немедленное обновление – в базе данных подписчик активизируются триггеры, которые обеспечивают распространение изменений на базу данных сервера публикаций, а затем эти изменения распространяются по другим подписчикам. Но, безусловно, имеются и различия, определяемые тем, что изменения хранятся в очереди до тех пор, пока не появится возможность отправить данные об изменениях в базу данных сервера публикаций. По умолчанию в качестве очереди используется таблица SQL Server, имеющая выразительное имя `MSreplication_queue`, но очередь может быть также реализована с помощью средств MSMQ (Microsoft Message Queuing – средства ведения очередей сообщений Microsoft). Чтобы перевести публикацию, первоначально настроенную на использование таблицы `MSreplication_queue`, на применение средств MSMQ, можно внести изменения на вкладке Updatable диалогового окна Publication Properties программы Enterprise Manager.

После того как подписчик вносит изменения в таблицу транзакционной репликации, которая предназначена для проведения обновлений в порядке очереди, активизируется триггер и вводит запись в очередь, сообщая тем самым об обновлении. Если для ведения очереди используется таблица `MSreplication_queue`, то постановка записи в очередь сводится к тому, что в ответ на изменение каждой строки в реплицируемой таблице вводится одна строка в таблицу `MSreplication_queue`. Если же очередь реализована с применением средств MSMQ, то данные об обновлениях сохраняются в очереди на распределительном сервере. А если распределительный сервер не доступен, то средства MSMQ ставят данные об обновлениях в очередь на компьютере подписчика до тех пор, пока не появится возможность установить связь с распределительным сервером.

Чтение данных из очереди и применение хранящихся в очереди данных об изменениях в базе данных сервера публикаций обеспечивает программа Queue Reader Agent. Если используется таблица `MSreplication_queue`, то указанная программа-агент считывает хранящиеся в очереди данные об изменениях непосредственно из таблицы. Если применяются средства MSMQ, то агент считывает изменения из очереди, которая ведется на компьютере распределительного сервера.

В случае обнаружения конфликтов разрешение конфликтов осуществляется в соответствии с правилами разрешения конфликтов, установленными при первоначальном создании публикации. В связи с этим могут создаваться компенсационные команды для отката транзакции в базе данных подписчика, но такие команды передаются только в базу данных подписчика, который был инициатором внесения изменений, но не передаются всем другим подписчикам на данную публикацию.

Немедленное обновление с обновлением в порядке очереди в качестве резервного варианта

Режим обновления, имеющий столь длинное название (немедленное обновление с обновлением в порядке очереди в качестве резервного варианта), вопреки своему названию, фактически осуществляется отнюдь не по такому принципу, что обновление в порядке очереди начинает применяться автоматически, если попытка немедленного обновления оканчивается неудачей (например, из-за того, что не удается установить соединение между базой данных подписчика и базой данных сервера публикаций). Дело в том, что пользователь должен разрешить переход на резервный вариант (вариант с использованием очередей) вручную. После этого пользователь сможет переключиться на режим немедленного обновления только после того, будет установлено соединение между базой данных подписчика и базой данных сервера публикаций, и программа Queue Reader Agent применит все хранящиеся в очереди данные об обновлении.

В рассматриваемом варианте обновления не применяется автоматический переход в режим обновления в порядке очереди, поскольку чаще всего нарушения связи между сервером подписчика и сервером публикаций могут быть легко устранены. Поэтому обычно лучше сразу устранить возникшие проблемы связи, чем автоматически переходить в режим записи данных об обновлениях в очередь, организованную в виде таблицы или хранилища MSMQ.

Проверка достоверности реплицированных данных

Как и в случае репликации снимка и репликации путем слияния, программа SQL Server позволяет проверять достоверность данных, распространяемых в виде публикации на основе транзакционной репликации. Например, можно проверить, имеют ли публикуемые таблицы в базе данных подписчика и в базе данных сервера публикаций одинаковое количество строк. Другой вариант может предусматривать сверку контрольных сумм или двоичных контрольных сумм данных в таблицах. Если используется проверка контрольной суммы, то сервер сравнивает 32-битовый код CRC (Cyclic Redundancy Check — циклический избыточный код) для каждой статьи таблицы в базе данных сервера публикаций и в базе данных подписчика. Поскольку код CRC вычисляется отдельно для каждого столбца, то порядок столбцов в каждой статье из базы данных сервера публикаций и базы данных подписчика может изменяться, не влияя на результаты сравнения. При проведении сравнения не учитываются данные из столбцов типов `text` и `image`.

Для проверки достоверности реплицируемых данных необходимо выполнить описанные ниже действия.

1. Разверните узел Publishers под узлом Replication Monitor в окне Enterprise Manager на распределительном сервере и выберите сервер публикаций, в базе данных которого находится публикация, подлежащая проверке.

2. В списке публикаций, развернувшемся в правой части окна, найдите публикацию, которую требуется проверить, и щелкните на ней правой кнопкой мыши. Выберите команду Validate Subscriptions.
3. В диалоговом окне Validate Subscriptions выберите подписки, которые требуется проверить. (Перечень подписок может оказаться пустым, если применяются только подписки, доставка которых инициируется анонимными подписчиками; в таком случае выберите команду Validate all subscriptions.)
4. Щелкните на кнопке Validate Options, чтобы определить тип выполняемой проверки. В частности, можно выбрать вариант с "быстрым" подсчетом количества строк (на основе кэшированной информации) или выполнить выборку фактического количества строк для каждой таблицы с помощью запроса T-SQL. Другие варианты состоят в применении способа сравнения контрольных сумм или двоичных контрольных сумм (в последнем случае и база данных сервера публикаций, и база данных подписчика должны эксплуатироваться под управлением SQL Server 2000 или более поздней версии сервера.
5. Закончив определение опций проверки, щелкните на кнопке OK, затем для проведения проверки достоверности данных щелкните на кнопке OK в диалоговом окне Validate Subscriptions.

Осуществление описанных выше действий приводит к тому, что в базе данных сервера публикаций вызывается хранимая процедура `sp_article_validation`. Затем процедура `sp_article_validation` вызывает процедуру `sp_replpostcmd` для передачи вызова процедуры `sp_table_validation` в базу данных распределительного сервера. Этот вызов включает значения количества строк и/или контрольных сумм из таблицы сервера публикаций, чтобы можно было сравнить эти значения со значениями в базах данных подписчиков. После этого программа Distribution Agent перехватывает вызов процедуры `sp_table_validation` и выполняет вызов на сервере подписчика. Если проверка завершается неудачей, то процедура `sp_table_validation` активизирует ошибки с помощью команды `RAISERROR` языка T-SQL, что приводит к регистрации ошибок в таблицах `MSdistribution_history` и `MSrepl_errors` базы данных распределительного сервера.

Для того чтобы ознакомиться с этой последовательностью действий на практике, рассмотрим простое упражнение, в котором показано, как принудительно вызвать ошибку при проверке достоверности данных.

Упражнение

Упражнение 22.1. Проверка достоверности транзакционной публикации

1. Создайте транзакционную публикацию для таблицы `Customers` базы данных `Northwind`. Разрешите использовать для доступа к этой публикации анонимные подписки, а для остальных параметров оставьте неизменными значения, предусмотренные по умолчанию.

2. Создайте анонимную подписку для новой публикации. Укажите в качестве целевой базы данных для этой подписки базу данных `pubs`.
3. Проверьте, осуществляется ли репликация таблицы `Customers` базы данных `Northwind` в базе данных `pubs`. После того как начнется успешная репликация, перейдите к п. 4.
4. Обновите значение в одном из столбцов таблицы `Customers` в базе данных `pubs` так, чтобы это значение отличалось от значения в той версии таблицы `Customers`, которая находится в базе данных сервера публикаций.
5. Подключитесь к распределительному серверу из программы Enterprise Manager. Разверните подузел Publishers узла Replication Monitor в дереве Enterprise Manager, найдите обозначение применяемого сервера публикаций и щелкните на этом обозначении.
6. Щелкните правой кнопкой мыши на обозначении публикации в списке, развернувшимся справа, и выберите команду Validate Subscriptions.
7. Оставьте отмеченным переключатель Validate all subscriptions и щелкните на кнопке Validation Options.
8. Отметьте опцию Compare checksums to verify row data (Сравнить контрольные суммы для проверки данных строки) и опцию This subscriber is a server running SQL Server 2000, use a binary checksum (Сервер подписчика имеет версию SQL Server 2000, использовать двоичные контрольные суммы), затем щелкните на кнопке OK.
9. Щелкните на кнопке OK, чтобы начать процесс проверки данных.
10. После следующего этапа прогона программы Distribution Agent дважды щелкните на обозначении этой программы в списке Agents\Distribution Agents, развернутом под узлом Replication Monitor на распределительном сервере, чтобы вывести на экран хронологию работы этой программы-агента. Щелкните на кнопке Session Details для ознакомления с подробными сведениями о сеансах, входящих в перечень хронологии агента.
11. Вы должны обнаружить запись, указывающую, что проверка данных для рассматриваемой статьи таблицы окончилась неудачей. Чтобы ознакомиться с дополнительной информацией об обнаруженной ошибке, можно щелкнуть на кнопке Error Details. Щелкните на кнопке Close, чтобы выйти из диалогового окна Session Details, затем еще раз щелкните на кнопке Close, чтобы выйти из диалогового окна Distribution Agent History.
12. Обнаруженные ошибки можно также просматривать непосредственно с помощью таблиц базы данных распределительного сервера. Для этого необходимо выполнить следующие запросы из программы Query Analyzer:

```
SELECT * FROM distribution..MSdistribution_history
SELECT * FROM distribution..MSrepl_errors
```

Здесь *distribution* — имя используемой базы данных распределительного сервера. Вы должны обнаружить в обеих таблицах записи, указывающие на возможные проблемы проверки данных.

Кроме того, можно выполнить настройку конфигурации предупреждающих сообщений о репликации для получения сообщений, связанных с обнаружением ошибок при проверке данных, а также обеспечить автоматическую повторную инициализацию подписки (подписок), в которых обнаружены ошибки. Для того

чтобы предусмотреть использование указанных возможностей, выполните описанные ниже действия.

1. Щелкните на узле Replication Alerts под узлом Replication Monitor на распределительном сервере.
2. Щелкните правой кнопкой мыши на записи Subscriber has failed validation (Обнаружение ошибок в базе данных подписчика) в развернувшемся справа списке и выберите команду Properties.
3. На вкладке General в диалоговом окне Properties щелкните на флажке Enabled, чтобы разрешить передачу предупреждающего сообщения. На вкладке Response можно задать конфигурацию конкретных операторов.
4. Если необходимо обеспечить автоматическую повторную инициализацию подписки, при проверке которой были обнаружены ошибки, щелкните на флажке Execute job вкладки Response и выберите задание Reinitialize subscriptions having data validation failures (Повторно инициализировать подписки с обнаруженными ошибками проверки данных) в разворачивающемся списке справа от этого флажка.
5. Щелкните на кнопке OK, чтобы сохранить информацию о конфигурации предупреждающего сообщения.

Следует отметить, что применение баз данных подписчиков с немедленным обновлением может привести к неудачному завершению проверки данных в промежутке времени между внесением изменения в базу данных подписчика и окончанием распространения этого изменения на базу данных сервера публикаций. Вполне естественно, что в указанном промежутке времени две копии опубликованной статьи не совпадают, поэтому проверка достоверности данных завершается неудачей. Единственным надежным способом предотвращения такой ситуации является отказ от внесения изменений в базу данных подписчика на то время, когда проходит процесс проверки данных.

Заслуживает также внимания то, что проверки по контрольной сумме не поддерживаются для тех публикаций, в которых используются средства DTS для преобразования данных, поскольку в результате преобразования значения данных в базе данных сервера публикаций и в базе данных подписчика становятся различными, поэтому значения контрольных сумм вряд ли будут в них совпадать.

Кроме того, проверка количества строк не поддерживается для статей, настройка конфигурации которых выполнена в виде горизонтальных секций DTS, поскольку критерии выборки для секции хранятся в пакете DTS, тогда как при использовании обычных критериев выборки для репликации части таблиц хранятся в базе данных сервера публикаций в виде представлений.

К тому же, по очевидным причинам не могут проводиться проверки достоверности данных подписок, распространяемых путем публикации по базам данных подписчиков, отличным от баз данных SQL Server, поскольку в базах данных таких подписчиков вряд ли можно найти хранимые процедуры SQL Server, предназначенные для осуществления проверок.

Пропуск ошибок

Чтобы разрешить пропуск ошибок, возникающих в процессе транзакционной репликации, можно задать параметр командной строки `-SkipErrors`, который предусмотрен для программ Distribution Agent и Log Reader Agent. В обычных условиях при эксплуатации этих программ-агентов в непрерывном режиме обнаружение ошибки приводит к аварийному завершению процесса распределения публикаций. Указанный параметр можно использовать для задания списка номеров ошибок (с разграничителями в виде двоеточия), которые следует игнорировать в процессе репликации. При обнаружении программой-агентом одной из ошибок, указанных в списке, производится просто регистрация ошибки в таблице `MSrepl_errors`, и работа продолжается. Для ознакомления с примером того, как применять на практике эту функциональную возможность, ознакомьтесь с профилем "Continue on data consistency errors" (Продолжить работу после обнаружения несогласованности данных) для программы Distribution Agent.

Удаление устаревших данных

Одним из стандартных заданий SQL Server Agent, которые вводятся в действие во время инсталляции средств репликации, является задача Distribution clean up. После того как все подписчики получат переданные им реплицированные транзакции, вызывается хранимая процедура `sp_MSdistribution_cleanup` для удаления доставленных транзакций из базы данных распределительного сервера. Продолжительность времени, в течение которого доставленные транзакции остаются в базе данных распределительного сервера, известна под названием *продолжительности хранения* и по умолчанию для транзакционной репликации составляет 72 часа. Чтобы изменить значение продолжительности хранения для распределительного сервера, необходимо выполнить описанные ниже действия.

1. Щелкните правой кнопкой мыши на узле Replication в окне Enterprise Manager на распределительном сервере и выберите в меню команду Configure Publishing, Subscribers, and Distribution.
2. На вкладке Distributor выберите обозначение базы данных распределительного сервера и щелкните на кнопке Properties.
3. В разделе Transaction retention открывшегося диалогового окна можно задать минимальную и максимальную продолжительности хранения в часах или сутках.

Если заданное значение продолжительности хранения согласовано с периодичностью резервного копирования целевых баз данных подписчика, то можно всегда гарантировать наличие в базе данных распределительного сервера всех данных, которые необходимы для восстановления целевой базы данных подписчика. Благодаря этому можно довольно просто обеспечить восстановление потерянной в результате аварии базы данных подписчика, поскольку подписчик может просто

повторно загрузить последнюю резервную копию, затем синхронизировать восстановленную базу данных с базой данных сервера публикаций и фактически немедленно восстановить актуальное состояние данных.

Резюме

Преимуществами транзакционной репликации является превосходное сочетание расширенных функциональных возможностей и несложного администрирования. Для транзакционной репликации предусмотрены более гибкие и всесторонние опции репликации по сравнению с репликацией снимка, но в то же время настройка и сопровождение осуществляются гораздо проще, чем при использовании репликации путем слияния.

С точки зрения архитектуры программного обеспечения можно отметить, что транзакционная репликация реализуется с помощью трех программ-агентов (Snapshot Agent, Log Reader Agent и Distribution Agent), а также с помощью кода, включенного в программное обеспечение самой СУБД SQL Server. Программа Snapshot Agent создает начальный снимок для транзакционных публикаций по такому же принципу, как и для публикации снимка. Затем программа Log Reader Agent считывает данные об изменениях, внесенных в базу данных сервера публикаций, и записывает информацию об изменениях, относящуюся к опубликованным статьям, в таблицы `MSrepl_commands` и `MSrepl_transactions` базы данных распределительного сервера. После этого программа Distribution Agent перехватывает данные об изменениях и применяет их к базам данных подписчиков.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Репликация путем слияния является единственной из всех трех разновидностей репликации, которая позволяет подписчикам вносить изменения в реплицируемые данные.
2. Какая расширенная хранимая процедура, реализованная внутри программы SQL Server, вызывается программой Log Reader Agent в целях выборки из журнала транзакций данных об изменениях, относящихся к опубликованной базе данных?
3. Чему равна заданная по умолчанию продолжительность хранения для транзакционной репликации?
4. Подтвердите или опровергните следующее утверждение. Безусловно, программа Snapshot Agent устанавливает разделяемые блокировки на статьях таблиц до начала создания снимка для транзакционной публикации, но конфигурация публикации может быть определена так, чтобы было разрешено вносить изменения в указанные объекты и во время создания снимка.
5. Какая таблица в базе данных распределительного сервера используется для хранения команд репликации, которые должны быть выполнены программой Distribution Agent в базах данных подписчиков?

6. Какая таблица в базе данных распределительного сервера используется для хранения информации об ошибках, относящихся к репликации?
7. Какая таблица в базе данных распределительного сервера хранит хронологическую информацию для программы Distribution Agent?
8. Возможно ли выполнить настройку конфигурации средств обработки предупреждающих сообщений системы репликации так, чтобы автоматически осуществлялась повторная инициализация подписок, для которых проверка окончилась неудачей?
9. Какая хранимая процедура вызывается в базе данных подписчика для проверки статьи таблицы?
10. Дайте определение термина *продолжительность хранения*.
11. Какую хранимую процедуру необходимо вызвать, чтобы задать значение опции синхронизации работы средств репликации со средствами резервного копирования для базы данных?
12. Предусмотрена ли возможность эксплуатировать одновременно несколько экземпляров программы Log Reader Agent применительно к одной и той же опубликованной базе данных?
13. Какая внутренняя расширенная процедура может быть вызвана в программе Log Reader Agent для указания на то, что обработка набора записей из журнала транзакций с данными об изменениях завершена?
14. Одно из трех соглашений о вызовах, которые могут использоваться для вызова процедур обновления, используемых в транзакционной репликации, основано на синтаксической конструкции `CALL`. Назовите две другие синтаксические конструкции и объясните, в чем состоят их отличительные особенности.
15. Для чего предназначен параметр `@maxtrans` процедуры `sp_replcmds`?
16. Подтвердите или опровергните следующее утверждение. Способы репликации, основанные на использовании средств доставки журнала и средств транзакционной репликации, являются взаимоисключающими: оба эти способа предусматривают чтение журнала транзакций опубликованной базы данных, поэтому не могут эксплуатироваться одновременно.
17. Подтвердите или опровергните следующее утверждение. Для того чтобы в случае нарушения связи база данных подписчика транзакционной репликации, действующая в режиме немедленного обновления, могла перейти в режим обновления в порядке очереди, необходимо разрешить выполнение такого перехода вручную.
18. Может ли подписчик, работающий в версии SQL Server 7.0, организовать подписку на параллельно обрабатываемый снимок с доставкой, инициируемой подписчиком?
19. Предположим, что применительно к статье опубликованной таблицы выполняется оператор `UPDATE` языка T-SQL, который изменяет 100 строк статьи. Какое минимальное количество новых записей должно появиться в таблице `MSrepl_commands` на распределительном сервере?
20. В чем состоит самый простой способ корректировки значения продолжительности хранения для распределительного сервера?

ГЛАВА 23

Репликация путем слияния

Безусловно, и транзакционная репликация, и репликация снимка реализуют принцип двунаправленного обновления данных, хотя и не предоставляют такие возможности и такое разнообразие вариантов, как репликация путем слияния. Несмотря на то, что транзакционная репликация и репликация снимка предоставляют возможность распространять обновления, введенные подписчиком, а также осуществлять обновления в порядке очереди, оба эти вида репликации не были с самого начала предназначены для выполнения указанных задач, поскольку фактически представляют собой просто механизмы передачи данных, поддерживающие наряду с этим некоторый удобный набор функциональных средств обновления данных. К тому же, репликация снимка и транзакционная репликация не предназначены для использования в таких вариантах среды, где подписчики часто работают в автономном режиме или обновляют данные столь же часто или даже более часто, чем это происходит на сервере публикаций. Кроме того, указанные виды репликации не позволяют публиковать данные, передавая их непосредственно подписчикам, использующим версию SQL Server CE. Несмотря на сказанное выше, репликация путем слияния является гораздо более сложной в части настройки конфигурации и сопровождения, не гарантирует транзакционной непротиворечивости, а также может не позволить достичь такой же производительности, как и транзакционная репликация, в простых сценариях двунаправленного обновления. Кроме того, репликация путем слияния — это относительно более новая разновидность репликации (по сравнению с двумя указанными выше), которая впервые появилась в версии SQL Server 7.0.

Как было отмечено в главе 21, в настоящей главе автор не стремится описать каждый отдельный нюанс репликации путем слияния или показать, как осуществляется администрирование при использовании этого вида репликации. Данная глава посвящена рассмотрению того, как действует репликация путем слияния. Подробное описание многих нюансов и деталей того, как выполняется настройка конфигурации и администрирование репликации путем слияния, а также сведения, касающиеся других видов репликации, можно найти в оперативной документации Books Online.

Краткий обзор

В программе SQL Server репликация путем слияния реализуется с помощью программ Snapshot Agent и Merge Agent. Как и в случае репликации снимка и транзакционной репликации, программа Snapshot Agent обеспечивает создание

начального снимка данных, который в дальнейшем должен распространяться сервером публикаций. После создания начального снимка программа Snapshot Agent создает в базе данных сервера публикаций синхронизационные задания, а также формирует относящиеся к репликации системные таблицы, хранимые процедуры и триггеры. Необходимые для подписчика системные объекты создаются при первом применении снимка к базе данных подписчика.

Следует отметить, что таблица подписки службы репликации путем слияния позволяет подписываться одновременно только на одну публикацию. Если пользователь предпримет попытку подписаться с помощью одной таблицы подписчика на статьи из двух различных публикаций, то после применения начального снимка работа программы Merge Agent окончится неудачей.

При репликации путем слияния программа Distribution Agent не применяется. Дело в том, что в случае репликации путем слияния те функции, которые при репликации снимка и транзакционной репликации обычно выполняет программа Distribution Agent, выполняются в службе репликации путем слияния программой Merge Agent. Кроме того, в репликации путем слияния база данных распределительного сервера используется не в такой степени. Основным назначением этой базы данных становится поддержка ведения журналов и накопления хронологических данных агента; к тому же, в базе данных распределительного сервера больше не хранятся данные, предназначенные для перенаправления подписчикам и от подписчиков. Свои собственные таблицы с промежуточным накоплением ведутся в каждой базе данных сервера публикаций и базе данных подписчика. Безусловно, распределительный сервер в случае репликации путем слияния выполняет столь ограниченные функции, что нередко распределительный сервер эксплуатируется на том же компьютере, что и сервер публикаций.

Программа Merge Agent обеспечивает получение начального снимка и применение этого снимка к базам данных подписчиков. После применения снимка программа Merge Agent приступает к сбору инкрементных обновлений, внесенных после создания снимка, применяет эти изменения к базам данных подписчиков, а также выгружает изменения, внесенные в базах данных подписчиков, на сервер публикаций. Такой процесс называется *синхронизацией*. Если доставка данных подписки инициируется распределительным сервером, то программа Merge Agent эксплуатируется на том же компьютере, на котором работает распределительный сервер, а если доставка подписок инициируется подписчиком, то указанная программа эксплуатируется на компьютере подписчика. Как и в случае использования программы Distribution Agent для репликации снимка и транзакционной репликации, можно обеспечить эксплуатацию программы Merge Agent на другом компьютере с применением средств дистанционной активизации агента.

Так же, как и другие агенты репликации, программа Merge Agent представляет собой терминальное приложение, в котором используется интерфейс ODBC. Обычно для вызова на выполнение программы Merge Agent применяется программа SQL Server Agent, но запуск программы Merge Agent можно также выполнить из командной строки. Исполняемый файл этой программы имеет имя `replmerge.exe`. Проверив количество потоков в данной программе с помощью утилиты Perfmon или подключившись с помощью отладчика и сформировав стеки потоков, можно убедиться в том, что это — многопотоковое приложение.

Программа Merge Agent применяет изменения к базе данных сервера публикаций или подписчика, вызывая хранимые процедуры, сформированные программой Snapshot Agent. Для вставки, обновления и удаления строк применяются отдельные хранимые процедуры. Как правило, эти процедуры имеют имена `sp_upd_GUID`, `sp_ins_GUID` и `sp_del_GUID`, где *GUID*— уникальный идентификатор статьи (взятый из столбца `artid` таблицы `sysmergearticles`), который относится к соответствующей статье. Эти процедуры вызываются с помощью средств RPC.

Программа Merge Agent приобретает способность обнаруживать изменившиеся данные в базе данных сервера публикаций и базах данных подписчиков благодаря тому, что специальные триггеры регистрируют сведения об изменениях данных опубликованных таблиц в системных таблицах репликации путем слияния в базе данных сервера публикаций и базах данных подписчиков по мере того, как происходят эти изменения. После каждой вставки или обновления строки соответствующее изменение записывается в таблицу `MSmerge_contents`, а после удаления каждой строки триггер записывает сведения об этом в таблицу `MSmerge_tombstone`. Эти таблицы, вместе взятые, выполняют те же функции при репликации путем слияния, что и таблица `MSrepl_commands` при репликации снимка и транзакционной репликации. Столбец `rowguid` каждой из этих таблиц можно использовать для создания обратного соединения, направленного к первоначальной опубликованной таблице. Проверяя таблицы `MSmerge_contents` и `MSmerge_tombstone`, наряду с таблицами `MSmerge_genhistory` и `MSmerge_info`, программа Merge Agent получает возможность определить, какие строки должны быть отправлены другому участнику репликации в составе текущей операции синхронизации.

Если таблица публикуется в качестве статьи в составе публикации, распространяемой путем слияния, то таблица должна иметь столбец уникального идентификатора, который обозначен свойством `ROWGUIDCOL` и имеет заданный на нем уникальный индекс. Если такой столбец не существует, то в таблицу автоматически вводится столбец с именем `rowguidcol`, и на этом столбце создается уникальный индекс. В столбце хранится значение глобально уникального идентификатора GUID. Система гарантирует, что значения идентификаторов GUID будут уникальными даже применительно ко всем компьютерам в мире, работающим в сети. Это позволяет задавать для конкретной строки идентификатор, уникальный среди многочисленных баз данных серверов публикаций и баз данных подписчиков на разных компьютерах.

Следует отметить, что в случае репликации путем слияния сервер публикаций и сервер подписчика становятся в значительной степени более равноправными партнерами. В отличие от репликации снимка и транзакционной репликации, при которых сервер публикаций явно доминирует, репликация путем слияния предназначена специально для двунаправленной репликации, поэтому основная часть наиболее важных системных таблиц, существующих в базе данных сервера публикаций, существует также в базе данных подписчика. Например, сервер подписчика, участвующий в репликации путем слияния, регистрирует изменения в опубликованных статьях с помощью своих собственных таблиц `MSmerge_contents` и `MSmerge_tombstone`, точно так же, как сервер публикаций. На сервере подписчика имеется таблица `MSmerge_genhistory`, которая обладает такой же структурой, как аналогичная таблица на сервере публикаций.

Сервер подписчика передает и получает сведения о поколениях отслеживаемых строк с помощью своей собственной таблицы MSmerge_replinfo, точно так же, как и сервер публикаций. Единственным исключением, нарушающим полное равноправие, является набор таблиц конфликтов. Для каждой опубликованной статьи предусмотрена собственная таблица конфликтов. После разрешения любого конфликта подробные сведения о том, как был разрешен конфликт, записываются в таблицу конфликтов. Поскольку по умолчанию разрешение конфликтов всегда происходит так, что решающую роль в этом играет сервер публикаций, таблицы конфликтов поддерживаются только на сервере публикаций. Не считая этого, все системные таблицы и процессы передачи измененных данных в версии репликации путем слияния являются практически одинаковыми и для сервера подписчика, и для сервера публикаций.

Разрешение конфликтов

При репликации путем слияния применяется развитая и расширяемая система разрешения конфликтов. Прежде всего, эта система позволяет определить, в чем фактически заключается конфликт, — считается ли конфликтом изменение, внесенное в одну и ту же строку статьи двумя разными участниками репликации путем слияния, или изменение, внесенное в один и тот же столбец одной и той же строки статьи двумя разными участниками. Если применяется обнаружение конфликтов с учетом столбцов, то два участника репликации путем слияния получают возможность изменять значения в разных столбцах одной и той же строки, не становясь участниками конфликта. Если же применяется обнаружение конфликтов с учетом строк, то изменения, внесенные двумя разными участниками в одной и той же строке, вызывают конфликт, который должен быть разрешен.

При репликации путем слияния конфликты обнаруживаются с помощью проверки значений rowguidcol, номеров поколений, значений столбцов lineage, а в случае распознавания конфликтов с учетом столбцов — значений в столбце colvl таблицы MSmerge_contents. Дополнительные сведения о назначении столбцов rowguidcol и generation будут приведены ниже, а теперь рассмотрим, для чего применяются столбцы lineage и colvl. Столбец lineage существует и в таблице MSmerge_contents, и в таблице MSmerge_tombstone и имеет тип varbinary(249). Значение lineage показывает хронологию изменений, внесенных в строку, и состоит из пар "имя/номер версии" с обозначением сервера публикаций и сервера подписчика, которые участвовали в предыдущих изменениях, внесенных в эту строку. Столбец colvl отслеживает аналогичную информацию, касающуюся отдельных столбцов, и имеет тип varbinary(2048). Этот столбец используется в режиме обнаружения конфликтов с учетом столбцов.

В системе предусмотрено применяемое по умолчанию средство разрешения конфликтов (сервер публикаций побеждает во всех конфликтах с сервером подписчика, а подписчики с высоким приоритетом побеждают в конфликтах с подписчиками, имеющими низкий приоритет), а также несколько других средств разрешения конфликтов, которые можно использовать в зависимости от кон-

кретных деловых потребностей. Например, пользователь может задать условие, что побеждает сервер публикаций или сервер подписчика, который внес изменение первым или последним. Можно также указать, что побеждает минимальное или максимальное из двух конфликтующих значений. Кроме того, пользователь может создавать собственные средства разрешения конфликтов либо в виде COM-объектов, либо в виде хранимых процедур. Для ознакомления со списком установленных в настоящее время средств разрешения конфликтов можно воспользоваться хранимой процедурой `sp_enumcustomresolvers`.

Если базы данных сервера публикаций и серверы подписчика находятся в синхронизированном состоянии, то изменения, внесенные сервером подписчика, выгружаются на сервер публикаций в первую очередь, а затем на сервер подписчика загружаются изменения, внесенные сервером публикаций. Такая организация работы позволяет осуществлять распознавание конфликтов на ранних стадиях, поскольку разрешение конфликтов всегда происходит таким образом, что решающую роль играет сервер публикаций, а применяемое по умолчанию средство разрешения конфликтов действует на основании условия, что сервер публикаций побеждает во всех конфликтах.

В процессе синхронизации в первую очередь обрабатывается информация об операциях удаления, а затем — информация об операциях вставки и обновления. Это означает, что сразу после начала очередного этапа синхронизации на сервер публикаций выгружаются изменения, зарегистрированные в таблице `MSmerge_tombstone` сервера подписчика. А затем после выгрузки информации об операциях удаления выгружается информация об операциях вставки и обновления. Как уже было сказано выше, данные об операциях вставки и обновления хранятся в таблицах `MSmerge_contents` сервера публикаций и сервера подписчика.

После проведения каждого сеанса синхронизации базы данных сервера публикаций с базой данных одного из серверов подписчиков сервер публикаций фактически становится владельцем изменений, внесенных подписчиком в опубликованные данные. Если на сервере подписчика внесены изменения, не конфликтующие с изменениями, внесенными на сервере публикаций (или если было применено средство разрешения конфликтов, которое позволило подписчику победить), то соответствующие изменения применяются к базе данных сервера публикаций. Затем во время сеансов синхронизации, проводимых сервером публикаций с другими подписчиками, эти изменения применяются к базам данных подписчиков при условии, что конфликты не обнаруживаются. Если же возникают конфликты, происходит разрешение конфликтов, после чего соответствующие изменения вносятся в базу данных сервера публикаций или в базу данных подписчика.

Иногда возникают ситуации, в которых на этапе синхронизации в базу данных сервера публикаций поступают изменения, внесенные в базу данных низкоприоритетного подписчика, а в дальнейшем эти изменения полностью отменяются после синхронизации базы данных сервера публикаций с базой данных высокоприоритетного подписчика. В результате этого возникают различия между информацией в базе данных низкоприоритетного подписчика, с одной стороны, и информацией в базе данных сервера публикаций и в базе данных высокоприоритетного подписчика, с другой стороны. Такие различия сохраняются до проведения очередного

сеанса работы программы Merge Agent, после чего данные во всех базах данных снова становятся синхронизированными. Это означает, что при использовании средств репликации путем слияния вполне возможна такая ситуация, в которой в разных базах данных на время остаются различные версии одной и той же строки, даже после однократного проведения сеанса синхронизации со всеми подписчиками. Безусловно, в конечном итоге все участники процесса репликации будут иметь в своем распоряжении одинаковые данные, но для достижения такого состояния может потребоваться проведение нескольких сеансов синхронизации.

Поколения

Каждой строке, отслеживаемой в таблице `MSmerge_contents` или `MSmerge_tombstone`, присваивается номер поколения с использованием столбца `generation`. Столбец `generation` имеет простой целочисленный тип данных и применяется как своего рода логические часы, позволяющие программе Merge Agent определить, когда было внесено изменение в строку и как это изменение соотносится во времени с изменениями в той же строке, внесенными другими участниками процесса репликации. При этом вместо значения даты и времени используется целочисленное значение, поскольку это позволяет избегать какой-либо зависимости от нарушений синхронизации часов между узлами, а также проще решать такие проблемы, возникающие при организации взаимодействия между узлами, как различия часовых поясов.

В таблице `MSmerge_contents` сопровождается только по одной строке в расчете на каждую строку, вставленную или обновленную в реплицируемой таблице. После каждого обновления строки обновляется номер поколения этой строки в таблице `MSmerge_contents`, и вместо него записывается текущий номер поколения, который задан в таблице `MSmerge_genhistory`. После каждой синхронизации статьи, хранящейся в базах данных сервера публикации и подписчика, программа Merge Agent обновляет значение в таблице `MSmerge_replinfo` для указания на то, что было отправлено и получено последнее поколение строки (соответственно с помощью вызовов процедур `sp_MSsetlastsentgen` и `sp_MSsetlastrecgen`). Затем вызывается процедура `sp_MSupdategenhistory` для обновления таблицы `MSmerge_genhistory` и записи очередного номера поколения для каждой статьи, применительно к которой была выполнена выборка изменений.

Номер поколения позволяет организовывать данные об изменениях, применяемых к различным статьям, в отдельные пакеты или группы. На первый взгляд может показаться, что в связи с этим номера поколений являются уникальными только для каждой статьи в публикации и служат в качестве своего рода вторичного ключа для идентификатора статьи, но дело обстоит иначе. Номера поколений не используются повторно в различных статьях и имеют глобальную область определения (в пределах таблицы `MSmerge_genhistory`). Если системе требуется новый номер поколения для статьи, которая еще не имеет номера поколения в таблице `MSmerge_genhistory` (например, если в таблицу статьи впервые вставляется строка), то система просто определяет максимальный номер поколе-

ния в таблице MSmerge_genhistory и добавляет к нему 1. Поэтому две статьи из одной и той же публикации не обращаются к одному и тому же номеру поколения и не используют одинаковый номер поколения. Каждая статья имеет свой собственный текущий номер поколения, который сопровождается в таблице MSmerge_genhistory. Это позволяет выборочно применять изменения к другим участникам синхронизации, выполняемой в рамках репликации путем слияния, а также дает возможность отдельно отслеживать изменения в различных статьях.

Текущий номер поколения для опубликованной базы данных увеличивается от одного прогона программы Merge Agent к другому. До выполнения очередного прогона программы Merge Agent в изменениях, внесенных в конкретную статью, используется текущий номер поколения для этой статьи и данный номер регистрируется вместе с данными об изменениях в таблицах MSmerge_comments и MSmerge_tombstone.

Чтобы понять, как применяются номера поколений в процессе репликации путем слияния, рассмотрим конкретный пример. Предположим, что таблицы TableA и TableB опубликованы в составе двух публикаций, реплицируемых путем слияния, и что эти таблицы являются единственными двумя опубликованными таблицами из всех таблиц какой-то конкретной базы данных. Обновляется ряд строк в таблице TableA, и данные об этих строках записываются в таблицу MSmerge_contents с указанием текущего (максимального) номера поколения для TableA, обнаруженного в таблице MSmerge_genhistory и равного 2. Затем программа Merge Agent выполняет очередной прогон, осуществляет выборку обновленных строк из таблицы MSmerge_contents, а также обновляет значение максимального номера поколения для TableA в таблице MSmerge_genhistory, присваивая ему 4, поскольку в таблице MSmerge_genhistory уже есть другие статьи, имеющие максимальный номер поколения 3. После этого происходит обновление строк в TableB. Программа Merge Agent регистрирует модифицированные строки в таблице MSmerge_contents с использованием текущего номера поколения для TableB, который равен 3. Во время следующего прогона программа Merge Agent считывает данные о новых изменениях и обновляет содержимое таблицы MSmerge_genhistory таким образом, чтобы в этой таблице для TableB применялся номер поколения 5, поскольку максимальный номер поколения 4 уже зарезервирован для другой таблицы. В этот момент текущий номер поколения для TableA равен 4 и текущий номер поколения для TableB равен 5. Если после этого произойдет обновление еще одной строки в TableA, то изменение будет зарегистрировано в таблице MSmerge_contents с номером поколения 4. А при очередном прогоне программы Merge Agent и выборке изменений, внесенных в TableA, текущий номер поколения для TableA будет установлен равным 6, поскольку в настоящее время максимальным зарегистрированным номером поколения является 5. Итак, изменения в TableA связаны с номерами поколений 2, 4 и 6, а изменения в TableB — с номерами поколений 3 и 5. После каждого назначения текущего номера поколения для какой-то конкретной статьи в программе Merge Agent задается также отметка верхнего уровня для следующего номера поколения, который должен быть выработан, независимо от статьи.

В некоторых обстоятельствах номер поколения для определенной строки может быть установлен равным 0, независимо от того, какой номер поколения зарегистрирован в таблице MSmerge_genhistory для статьи, содержащей эту строку. В качестве примера можно указать такую ситуацию, в которой происходит обновление строки в базе данных сервера публикаций и удаление той же строки в базе данных подписчика. Если применяется предусмотренная по умолчанию стратегия разрешения конфликтов (в которой сервер публикаций всегда побеждает), то будет обнаружено, что обновление в базе данных сервера публикаций конфликтует с удалением в базе данных подписчика, поэтому операция удаления в базе данных подписчика будет принудительно отменена. В таком случае произойдет повторная вставка строки в базу данных подписчика, а в таблицу MSmerge_contents, относящуюся к базе данных подписчика, будет вставлена запись, относящаяся к рассматриваемой строке, и имеющая номер поколения 0. Строка в таблице MSmerge_tombstone, относящейся к базе данных подписчика, удаляется, поэтому складывается такое впечатление, что в базе данных подписчика не происходило никакого удаления. Информация о конфликте регистрируется в таблице MSmerge_delete_conflicts, относящейся к базе данных сервера публикаций, а в качестве значения столбца reason_text регистрируется сообщение, имеющее примерно такой смысл: "Одна и та же строка была обновлена в базе данных TUK\PHRIP.Northwind и удалена в базе данных TUK\PHRIP.testrepl. Средством разрешения конфликтов в качестве победителя была выбрана база данных, в которой произошло обновление".

Применение критериев выборки

Как и при использовании репликации других типов, в процессе репликации путем слияния имеется возможность применять критерии выборки для создания вертикальных и горизонтальных секций. Критерии выборки горизонтальных секций (или критерии выборки строк) реализуются с помощью конструкции WHERE языка SQL и ограничивают состав строк, включаемых в публикацию, с учетом критериев, заданных пользователем. Критерии выборки вертикальных секций (или критерии выборки столбцов) ограничивают состав столбцов, включаемых в публикацию.

В случае использования критериев выборки столбцов создается представление, которое включает только требуемые столбцы. Это представление получает имя в форме *publication_article*_VIEW (здесь *publication* — публикация, а *article* — статья) и находится в базе данных сервера публикаций. В столбце sync_objid таблицы sysmergearticles хранится не базовый объект представления, а идентификатор объекта этого представления, поэтому запросы, выполняемые программой Merge Agent для обнаружения модифицированных строк, применяются к представлению.

Что же касается базы данных подписчика, то столбец sync_objid таблицы sysmergearticles содержит идентификатор базового объекта. Применение критериев выборки обеспечивается с помощью определяемых пользователем хранимых процедур (формируемых программой Snapshot Agent), которые исключают столбцы, не заданные в составе критериев выборки.

При создании критериев выборки строк происходит аналогичный процесс. В базе данных сервера публикаций создается представление, имеющее имя в форме *publication_article*_VIEW. С помощью этого представления осуществляется выборка строк из соответствующей статьи таблицы, отвечающих выражению критериев выборки (критерии выборки определяются с помощью конструкции WHERE). Само выражение критериев выборки создается при их определении. Ниже показано, как выглядят типичные критерии выборки строк.

```
create view [Northwind_Huck_Photech_VIEW] as
select  [Huck_Photech].*  from [dbo].[ Huck_Photech] [Huck_Photech]
where ( (id>1) )
and ({ fn ISPALUSER('256ABC83-6C4D-49AF-8456-766443672303') } = 1)
```

В данном случае критерии выборки основаны на использовании значения столбца id таблицы, поэтому создается конструкция WHERE, которая ограничивает состав возвращаемых строк теми строками, которые соответствуют заданному множеству значений id (используемое при этом выражение выделено полужирным шрифтом). Кроме того, в этом представлении вызывается функция ISPALUSER (с использованием синтаксической конструкции ODBC с управляющим кодом) для обеспечения того, чтобы идентификатор пользователя, получающего доступ к представлению, находился в списке доступа к публикации, относящемуся к данной публикации. (Глобально уникальный идентификатор GUID, передаваемый в функцию ISPALUSER, соответствует значению pubid для публикации, хранящемуся в таблице sysmergearticles.)

Как и в случае представления с критерием выборки по вертикали (по столбцам), идентификатор объекта для представления с критерием выборки по горизонтали (по строкам) хранится в столбце sync_objid таблицы sysmergearticles. А в базе данных подписчика хранится идентификатор объекта самой статьи таблицы.

Если в базе данных подписчика происходит вставка строки, которая не соответствует критерию выборки по горизонтали, то программа Merge Agent удаляет эту строку во время следующего сеанса синхронизации базы данных подписчика с базой данных сервера публикаций. Программа записывает удаленную строку в таблицу MSmerge_tombstone, соответствующую базе данных подписчика, и задает "System delete" (Удалено системой) в качестве значения MSmerge_tombstone.reason.

Оптимизация процесса синхронизации

Программа SQL Server поддерживает специальные средства оптимизации для публикаций с критериями выборки по горизонтали, которые позволяют следить в базе данных сервера публикаций за тем, какие строки поступают и выходят из определяемой с помощью критериев выборки секции опубликованной таблицы. Если определена такая публикация с критериями выборки по горизонтали, в которой часто изменяются значения в столбце (или столбцах), применяемом в качестве критериев выборки, то может потребоваться разрешить использование указанных средств оптимизации (доступ к этим средствам предоставляется с помощью опции Optimize Synchronization программы-мастера Create Publication Wizard), поскольку оптимизация позволяет существенно уменьшить объем дан-

ных, передаваемых подписчикам. Рассмотрим ситуацию, в которой к таблице применяются критерии выборки по горизонтали, основанные на использовании столбца ZipCode (Почтовый код). Если в какой-то конкретной строке таблицы изменяется значение почтового кода, то изменяется и состав секции таблицы в соответствии с этим изменением. По умолчанию в программе Merge Agent не предусмотрен способ, позволяющий определить, к какой секции когда-то относилась строка со вновь введенными изменениями, поэтому в программе невозможно определить, каким подписчикам следует направить информацию об изменении в строке. Поэтому программа Merge Agent вынуждена передавать информацию об изменениях всем подписчикам, а это приводит к бесполезному расходованию пропускной способности и сетевых ресурсов. Разрешив использование опции Optimize Synchronization, вы предоставляете программе Merge Agent возможность определять предыдущие значения в столбцах, входящих в состав критериев выборки. Это позволяет определить, какому подписчику (подписчикам) требуется отправить информацию об обновлении строки.

Следует учитывать, что применение указанной опции требует дополнительного объема памяти в базе данных сервера публикаций. Точный объем дополнительного пространства зависит от размеров столбцов, применяемых в критериях выборки. Если столбцы в критериях выборки имеют общий объем 50 байтов, а количество строк равно 100 тыс., то для использования опции Optimize Synchronization требуется 5 Мбайт дополнительного пространства. Но обычно такие дополнительные затраты вполне оправданы, если учесть, какую колоссальную экономию пропускной способности обеспечивает эта опция.

Динамически определяемые критерии выборки

Динамически определяемые критерии выборки представляют собой критерии выборки по горизонтали специального типа, в которых в составе критериев выборки используются такие недетерминированные функции T-SQL, как HOST_NAME или SUSER_SNAME. Динамически определяемые критерии выборки могут содержать прямую ссылку на недетерминированную функцию или ссылку на определяемую пользователем функцию, которая либо содержит ссылку на недетерминированную функцию, либо принимает такую функцию в качестве параметра. Основной замысел, лежащий в основе динамически определяемых критериев выборки, состоит в том, что эти критерии выборки должны возвращать различные наборы строк в зависимости от того, каким подписчиком они используются.

При создании публикации, включающей динамически определяемые критерии выборки, пользователь вправе предоставить системе возможность проверять, изменяется ли значение недетерминированной функции (функций), применяемой для регламентации состава строк, передаваемых подписчику, от одного этапа синхронизации к другому. Это позволяет обеспечить единообразное секционирование данных при каждой синхронизации. Например, если для задания динамически определяемых критериев выборки используется функция HOST_NAME (Имя хоста), а имя компьютера подписчика изменяется в какой-то момент после очередной доставки строк подписчику программой Merge Agent, то

подписку приходится инициализировать повторно, поскольку строки, предоставленные подписчику на предыдущем этапе, выходят за пределы секции, формируемой с помощью динамически определяемых критериев выборки. Дело в том, что в связи с самим существованием таких критериев выборки обновление тех же строк с учетом изменений, внесенных в базу данных сервера публикаций, с применением программы Merge Agent становится невозможным.

Критерии выборки на основе соединения

Критерии выборки на основе соединения позволяют распространить критерии выборки строк с одной опубликованной таблицы на другую. После определения критериев выборки по горизонтали для одной таблицы публикации эти критерии можно распространить на другую таблицу публикации путем задания критериев выборки на основе соединения (по существу, состоящих из конструкций JOIN языка T-SQL), которые связывают первую таблицу со второй. В результате этого выборка строк из второй таблицы будет осуществляться с учетом заданного условия соединения, и подписчикам будут передаваться только такие строки, которые соответствуют условию соединения.

Таким образом, появляется возможность связать сразу несколько таблиц, обеспечивая выборку из всех таблиц с учетом критериев выборки по горизонтали для первой таблицы в сочетании с условиями соединения, заданными для остальных таблиц. Во время синхронизации обеспечивается принудительное применение соотношений, установленных между таблицами, что позволяет задавать сложные связи между опубликованными статьями.

Динамически сопровождаемые снимки

По умолчанию при задании динамически определяемых критериев выборки программа Merge Agent применяет начальный снимок к базе данных подписчика, передавая каждый раз по одной строке, чтобы сохранялась возможность проверить, соответствует ли каждая строка критериям выборки (поскольку динамически определяемые критерии выборки не формируются, пока не начнется синхронизация). Вполне естественно, что осуществление такого режима применения начального снимка при наличии значительных объемов данных может потребовать много времени. Использование начального снимка можно существенно ускорить путем создания динамически сопровождаемого снимка. Динамически сопровождаемый снимок позволяет подготовить набор файлов данных в формате BCP, выборка данных в которых выполнена заранее с учетом заданных значений SUSER_SNAME и HOST_NAME. Иными словами, если система обслуживает подписчика, который подключается к базе данных сервера публикаций под каким-то конкретным идентификатором пользователя в целях синхронизации публикации с динамически определяемыми критериями выборки с помощью программы Merge Agent, то для данного конкретного подписчика можно заранее создать снимок и сохранить файлы снимка в собственном каталоге подписчика. В таком случае после подключения программы Merge Agent подписчика к базе дан-

ных сервера публикаций в целях синхронизации эта программа может вставлять данные в базу данных в режиме массового копирования с обычной скоростью, а не выполнять вставку каждый раз по одной строке.

Чтобы создать и применить динамически сопровождаемый снимок, выполните описанные ниже действия.

1. Создайте публикацию с динамически определяемыми критериями выборки и сформируйте для этой публикации применяемый по умолчанию снимок.
2. Создайте задание динамически сопровождаемого снимка с помощью программы-мастера Create Dynamic Snapshot Job (доступ к этой программе можно получить, щелкнув правой кнопкой мыши на обозначении публикации под узлом `Databases\YourDatabase\Publications` в программе Enterprise Manager).
3. Создайте подписку с доставкой, инициируемой подписчиком, и укажите местонахождение снимка, который был передан на обработку программе-мастеру Create Dynamic Snapshot Job. Отметьте флажок, указывающий, что это — динамически сопровождаемый снимок.

Следует учитывать, что заданиями динамически сопровождаемых снимков нельзя управлять с помощью программы Replication Monitor. Безусловно, всеми заданиями на репликацию можно управлять с помощью программы SQL Server Agent, но большинством из таких заданий можно также управлять с помощью программы Replication Monitor. Задания динамически сопровождаемых снимков представляют собой исключения. Вместо этого для управления заданиями динамически сопровождаемых снимков необходимо использовать команду `Management\SQL Server Agent\Jobs`.

Управление диапазонами идентификаторов

Одной из проблем, с которыми часто приходится сталкиваться в обычных сценариях репликации, является эффективное управление диапазонами идентификаторов в базе данных сервера публикаций и базе данных подписчика. Дело в том, что если вставка новых строк может осуществляться многими участниками репликации, то приходится предусматривать способ предотвращения коллизий между применяемыми значениями идентификаторов.

Существует несколько возможных решений этой проблемы. Одно из решений, широко используемых автором в прошлом, состояло в том, что по участникам репликации распределялись четные и нечетные или отрицательные и положительные значения идентификаторов. Например, предположим, что используется простая реализация средств репликации, в которой имеется сервер публикаций и один подписчик. В таком случае можно назначить для сервера публикаций значение идентификатора, равное 1, и увеличивать его на 1 после вставки каждой строки, а подписчику присвоить начальное значение -1 (напомним, что

в программе SQL Server тип данных int, лежащий в основе типа данных идентификатора, имеет знак) и складывать с этим значением значение -1 после вставки каждой новой строки. Теперь допустим, что в применяемой реализации средств репликации используется четыре компьютера, по одному для сервера публикаций и трех подписчиков. В таком случае можно задать для первого компьютера начальное значение 1 и складывать с этим значением значение 2 после вставки каждой новой строки (что приводит к получению положительных нечетных значений идентификатора); второму компьютеру присвоить начальное значение 2 и складывать с ним 2 (получая положительные четные значения); третьему компьютеру присвоить значение -1 и складывать с ним -2 после вставки каждой новой строки (что приводит к получению отрицательных нечетных значений); а четвертому компьютеру присвоить значение -2 и складывать его с -2 (получая отрицательные четные значения). Такой подход можно распространить на еще большее количество компьютеров, дополнительно подразделяя каждый диапазон идентификаторов на меньшие части. Основная идея такого подхода заключается в том, что можно найти множество оригинальных способов предотвращения коллизий между значениями идентификаторов, применяемых в базах данных серверов публикаций и в базах данных подписчиков, неуклонно применяя ряд способов определения начального значения столбца идентификатора и значения приращений.

Еще один способ управления значениями идентификаторов в сценариях репликации состоит в использовании опции NOT FOR REPLICATION при определении столбца идентификатора identity. При этом либо задаются совместимые диапазоны идентификаторов для базы данных сервера публикаций и баз данных подписчиков, либо значения идентификаторов определяются программным путем. При вставке значения идентификатора непосредственно с помощью оператора INSERT (если разрешена опция SET IDENTITY INSERT) начальное значение идентификатора переустанавливается. Задание опции NOT FOR REPLICATION предоставляет агентам репликации возможность управлять переопределением начального значения идентификатора, поэтому при вставке агентом репликации значения в столбец идентификатора, который был создан с опцией NOT FOR REPLICATION, начальное значение идентификатора не переустанавливается. Благодаря этому средства репликации получают возможность выполнять все действия, необходимые для доставки данных из одного места в другое в пределах топологии (т.е. вставлять, обновлять и удалять данные), не нарушая принципов обычного использования столбцов идентификаторов и в базе данных сервера публикаций, и в базах данных подписчиков.

При использовании способа, в основе которого лежит опция NOT FOR REPLICATION, необходимо также создать ограничение CHECK на статьях публикации, чтобы значения идентификаторов в базах данных каждого из участников сценария репликации не перекрывались. Поскольку при этом пользователь берет на себя ответственность за управление этими значениями самостоятельно, он должен гарантировать, что присваивание и управление значениями идентификаторов происходит логически обоснованно и единообразно.

Еще один вариант предотвращения проблем, связанных с коллизиями значений идентификаторов в сценариях репликации, состоит в том, чтобы вообще не использовать столбцы идентификаторов (или исключать эти столбцы из публикаций). Если

столбец идентификатора действительно не требует передачи в базу данных подписчика, то появляется возможность исключить этот столбец из публикации с помощью критериев выборки по вертикали. Если же такой столбец входит в состав первичного ключа таблицы, то этот столбец нельзя изъять с помощью критериев выборки, поэтому может потребоваться определить другой столбец (или столбцы) в качестве первичного ключа, если это возможно и приемлемо в конкретном сценарии репликации. Например, вполне приемлемым кандидатом на использование в качестве первичного ключа является столбец `rowguidcol`, введенный в результате развертывания средств репликации путем слияния (хотя для каждого из значений этого столбца требуется 16 байтов памяти, а этот объем далек от идеального). Еще один вариант состоит в том, что иногда можно воспользоваться некоторыми другими столбцами или комбинацией столбцов. Общий вывод состоит в том, что, исключая необходимость публикации значений идентификаторов, вы сразу же исключаете возможность коллизий значений идентификаторов между базой данных сервера публикаций и базами данных подписчиков и вместе с тем оставляете за собой право воспользоваться преимуществами автоматической выработки значений идентификаторов в базе данных сервера публикаций с помощью программы SQL Server, если будет решено просто исключать столбцы идентификаторов с помощью критериев выборки из публикации, а не удалять их из таблицы.

Наилучший способ управления значениями идентификаторов в сценариях репликации состоит в том, чтобы предоставить возможность программе SQL Server обеспечивать такое управление от имени пользователя. Программа SQL Server обладает способностью управлять диапазонами значений идентификаторов в базе данных сервера публикаций и базах данных подписчиков применительно к публикациям, применяемым в рамках репликации путем слияния (в этом сценарии возможность двунаправленной модификации предусмотрена по умолчанию), а также применительно к публикациям, сопровождаемым в рамках транзакционной репликации и репликации снимка (в этих сценариях возможность двунаправленной модификации предоставляется, если разрешено использование баз данных подписчиков с обновлением в порядке очереди). После добавления статьи к публикации можно щелкнуть на кнопке со знаком троеточия на странице Specify Articles программы-мастера Create Publication Wizard, чтобы вывести на экран диалоговое окно Table Article Properties. Затем можно щелкнуть на вкладке Identity Range, чтобы ввести в действие средства автоматического управления диапазонами значений идентификаторов программы SQL Server.

В диалоговом окне Table Article Properties можно задать три важных значения. Первым из них является размер диапазона идентификаторов в базе данных сервера публикаций, а вторым – размер диапазона идентификаторов в базах данных подписчиков. Оба эти значения по умолчанию равны 100, но при желании могут быть заданы бо́льшие или меньшие значения. Третьим значением является пороговое значение, после достижения которого необходимо переходить к присваиванию нового диапазона. Программа SQL Server автоматически назначает совместимые диапазоны значений идентификаторов для базы данных сервера публикаций и для каждой базы данных подписчика с помощью соответствующего агента репликации. При такой организации репликации ни один из участников

репликации не начинает работу с диапазона, пересекающегося с другим диапазоном. В дальнейшем по мере вставки строк в таблицы на каждом узле и синхронизации изменений, вносимых в базу данных сервера публикаций и базы данных подписчиков, проверяются назначенные диапазоны идентификаторов для определения количества оставшихся идентификаторов. После достижения порогового значения, указанного в диалоговом окне Table Article Properties, будет создан новый диапазон для соответствующей базы данных, а в качестве начального значения идентификатора будет задано начало нового диапазона.

При этом используется пороговое, а не конечное значение диапазона, поскольку новый диапазон задается только на этапе синхронизации, а если интервалы времени между этапами слишком велики, то значения диапазона, назначенного для таблицы, могут быть исчерпаны до момента проведения очередного этапа синхронизации. Поэтому при использовании порогового значения предпринимается попытка оставить в конце диапазона достаточно большой запас значений идентификаторов, чтобы можно было создать новый диапазон еще до того, как будет исчерпан предыдущий, при условии, что синхронизация проводится достаточно часто. Если же исчерпываются значения в управляемом диапазоне значений идентификаторов, то появляется примерно такое сообщение:

```
Server: Msg 548, Level 16, State 2, Line 1
The identity range managed by replication is full and must be
updated by a replication agent. The INSERT conflict occurred in
database 'Northwind', table 'Huck_Photech', column 'id'.
Sp_adjustpublisheridentityrange can be called to get a new
identity range.
The statement has been terminated.
```

Как указано в этом сообщении, можно вызвать процедуру sp_adjustpublisheridentityrange (или просто дождаться проведения очередной синхронизации), чтобы устранить проблему нехватки значений идентификаторов. Следует отметить, что процедура sp_adjustpublisheridentityrange позволяет исправить ситуацию независимо от того, возникла ли указанная проблема в базе данных сервера публикаций или в одной из баз данных подписчиков.

Для того чтобы предотвратить возникновение указанной ошибки, необходимо иметь возможность корректировать размеры диапазонов, пороговые значения и периодичность синхронизации. С подробными сведениями об управляемом диапазоне идентификаторов можно ознакомиться, просмотрев таблицу MSrepl_identity_range.

Резюме

Настройка конфигурации и управление репликацией путем слияния гораздо сложнее по сравнению с транзакционной репликацией или репликацией снимка, но репликация путем слияния предоставляет несколько возможностей, не предусмотренных в двух других видах репликации. Тем не менее репликация путем слияния не всегда является наилучшим вариантом в любом сценарии, в котором требуется двунаправленная репликация данных (для этого часто вполне подходит

транзакционная репликация), хотя и предоставляет больше возможностей организации работы по сравнению с репликацией снимка или транзакционной репликацией, особенно в отношении пользователей, часто работающих в автономном режиме, и подписчиков SQL Server CE.

Система репликации путем слияния поддерживает целый ряд механизмов обнаружения и устранения конфликтов. Прежде всего, эта система позволяет определить, какая ситуация рассматривается как конфликтная, а затем указать, кто из участников репликации побеждает в случае обнаружения конфликта.

Вопросы для самопроверки

1. Подтвердите или опровергните следующее утверждение. Для управления заданиями по созданию динамических снимков можно использовать диалоговое окно Replication Monitor программы Enterprise Manager.
2. По умолчанию предусмотрено, какая из сторон должна всегда побеждать в конфликте между сервером публикации и высокоприоритетным подписчиком. Укажите, какой именно участник репликации определен по умолчанию как победитель.
3. Подтвердите или опровергните следующее утверждение. В типичном сценарии репликации путем слияния должно быть достаточным проведение не больше одного сеанса синхронизации для обеспечения синхронизации данных в базе данных сервера публикаций и базах данных подписчиков, независимо от количества подписчиков.
4. Почему применение динамически сопровождаемого снимка обеспечивает повышение производительности для публикаций с динамически определяемыми критериями выборки?
5. Опишите данные, хранящиеся в столбце `lineage`.
6. Какая таблица используется в процессе репликации путем слияния для ведения очереди удаленных строк?
7. Какая таблица используется в процессе репликации путем слияния для отслеживания последних значений номеров поколений переданных и полученных строк?
8. Подтвердите или опровергните следующее утверждение. Репликация путем слияния — это единственный тип репликации, применяемой для распространения данных по подписчикам, которые используют программы, отличные от SQL Server, например, программу Microsoft Access.
9. Подтвердите или опровергните следующее утверждение. Репликация путем слияния — это единственный тип репликации, применяемой для распространения данных по подписчикам, использующим программу SQL Server CE.
10. Какой способ внесения изменений в базу данных подписчика используется в программе Merge Agent — вызов хранимой процедуры или выполнение команды UPDATE языка T-SQL?

11. Подтвердите или опровергните следующее утверждение. Программа Merge Agent – однопотоковое приложение.
12. Назовите имя столбца уникального идентификатора, который автоматически добавляется при инициализации средств репликации путем слияния к статье таблицы (при условии, что этот столбец еще не задан)?
13. Опишите назначение столбца `colv1` таблицы `MSmerge_contents`.
14. Подтвердите или опровергните следующее утверждение. Программа Distribution Agent обеспечивает передачу данных об обновлении от подписчика, применяющего средства публикации путем слияния, к серверу публикаций.
15. Подтвердите или опровергните следующее утверждение. В ходе репликации путем слияния активизируется триггер, который модифицирует значение в столбце GUID строки после внесения каждого изменения в строку.
16. В сочетании с какими критериями выборки обычно используется функция `SUSER_SNAME` языка T-SQL?
17. Опишите назначение хранимой процедуры `sp_enumcustomresolvers`.
18. Как изменяется начальное значение идентификатора, если для столбца идентификации не задана опция `NOT FOR REPLICATION`, и в каком-то соединении происходит вставка значения в столбец (если разрешена опция `SET IDENTITY_INSERT`)?
19. В чем состоит наилучший способ предотвращения коллизий между значениями идентификаторов, используемыми сервером публикаций и подписчиками или используемыми несколькими подписчиками?
20. В каком выпуске SQL Server впервые были введены в действие средства репликация путем слияния?
21. Опишите назначение критериев выборки, применяемых в операциях соединения.
22. Подтвердите или опровергните следующее утверждение. В ходе синхронизации прежде всего обрабатываются данные о выполненных операциях удаления, а затем данные об операциях вставки и обновления.
23. Подтвердите или опровергните следующее утверждение. В ходе синхронизации прежде всего происходит загрузка в базы данных подписчиков изменений, внесенных в базу данных сервера публикаций, а затем изменения последовательно выгружаются из баз данных подписчиков в базу данных сервера публикаций.
24. Какие функции выполняет база данных распределительного сервера в ходе репликации путем слияния?
25. Возможно ли обеспечить разрешение конфликтов с использованием хранимой процедуры?

ЧАСТЬ IV

Недокументированные средства программы SQL Server

ГЛАВА 24

Поиск недокументированных средств

Автор посвятил по одной главе в каждой из двух своих последних книг по программе SQL Server описанию недокументированных средств этого программного продукта. В каждой книге был представлен список недокументированных хранимых процедур, расширенных процедур, функций, флажков трассировки, синтаксических конструкций команд и средств программного продукта. Но в настоящей книге принят подход, согласно которому считается не менее важным описание того, как работает тот или иной компонент, чем демонстрация его практического использования, поэтому автор решил отказаться от простого перечисления недокументированных средств. Но взамен будет показано, как находить подобные недокументированные средства самостоятельно.

Автор надеется, что, сделав явными такие скрытые средства, особенно те, которые относятся к языку Transact-SQL (основному средству, с помощью которого пользователи получают доступ к серверу), он побудит разработчиков либо описать недокументированные средства, либо сделать их полностью недоступными для пользователя. Автор на своем опыте убедился в том, что программа SQL Server характеризуется непревзойденным количеством скрытых возможностей. Создается впечатление, что в этой программе гораздо больше недокументированных средств по сравнению с аналогичными программными продуктами. Возможно, это связано с тем, что SQL Server в целом обладает значительно большим запасом средств, чем подобные программные продукты, а возможно, причина состоит в том, что разработчики SQL Server слишком во многом полагаются на недокументированные функциональные возможности.

Многие из недокументированных средств являются достаточно надежными для использования в таких важных частях рассматриваемого программного продукта, как система репликации или программа-мастер Index Tuning Wizard, чтобы можно было полностью на них опереться, поэтому утверждения, что эти средства не следует документировать, поскольку они не подходят для конечного пользователя, кажутся недостаточно убедительными. Разумеется, некоторые недокументированные средства применяются для реализации каких-то побочных функций рассматриваемого программного продукта и вполне могут оказаться не совсем

пригодными для более общего пользования, поэтому можно понять, почему они не документированы. Мало того, некоторые флажки трассировки при их неправильном использовании могут быть даже опасными, поэтому, например, автор всегда проверяет, какие из этих флажков можно раскрывать для читателя, а какие — нет. Вообще говоря, автор приводит в своих книгах только такие недокументированные средства, которые считаются безопасными для использования или настолько полезными, что просто их нельзя не упомянуть.

Кроме того, автор считает, что некоторые недокументированные средства не описаны в документации просто потому, что их упустили из виду. Программа SQL Server в целом и язык Transact-SQL в частности предоставляют исключительно широкие возможности, поэтому нет ничего удивительного в том, что неизбежно часть из этих возможностей остается не отраженной в документации данного программного продукта. Возьмем, например, свойства `OBJECTPROPERTY` с именами `TriggerInsertOrder`, `TriggerDeleteOrder` и `TriggerUpdateOrder`. Автор не может понять, почему они не описаны в документации. По этой причине он не может определить, для чего они предназначены. Эти свойства используются в триггерах репликации для проверки того, является ли данный триггер первым из триггеров, выполненных применительно к операции DML конкретного типа. Для получения той же информации можно применить документированные имена свойств `ExecIsFirstInsertTrigger`, `ExecIsFirstUpdateTrigger` и `ExecIsFirstDeleteTrigger`, поэтому создается впечатление, что в действительности недокументированные имена свойств не нужны. Но даже если так и есть на самом деле, почему бы не описать эти имена свойств в документации, чтобы пользователи могли понять, что важно только, какой триггер выполняется первым или последним применительно к конкретной операции DML, а в остальном порядок выполнения триггеров не отражен в документации, поэтому нельзя учитывать этот порядок, организуя эксплуатацию приложения в какой-либо форме?

Но независимо от того, по какой причине осталось недокументированным то или иное средство, мы можем лучше узнать о программном продукте, ознакомившись с ним и поняв, какие сведения можно получить с его помощью. Автор считает, что подход, в котором предусматривается изучение недокументированных средств, особенно оправдан в такой книге, которая предназначена для описания работы программного продукта. Зная недокументированные средства программного продукта и понимая их назначение, читатель получает лучшее представление о проекте самого продукта. В частности, создается лучшее представление о том, каковы ограничения недокументированных средств и для чего они были предназначены по замыслу проектировщиков. Все указанные возможности являются весьма благоприятными, независимо от того, действительно ли какие-либо недокументированные средства будут применяться в дальнейшей работе.

Кроме того, подводя итог сказанному выше, автор хочет повторить оговорку, приведенную в его предыдущих книгах, — используйте недокументированные средства на свой страх и риск. Оставляя какие-либо средства программного продукта недокументированными, поставщик резервирует за собой право вносить изменения в любое время. Появление внеочередного выпуска, программного исправления для системы защиты, служебного пакета или новой версии

программного продукта может привести к изменению или исключению недокументированных функциональных средств, на основе которых непредусмотрительный пользователь разработал свой код. Кроме того, продумывая возможность использования недокументированного средства, следует помнить, что оно могло остаться не проверенным настолько тщательно, как остальная часть программного продукта (автору почти не приходилось сталкиваться с тем, чтобы в испытательных подразделениях разрабатывались наборы тестов для недокументированных средств), и могут не работать достаточно надежно во всех ситуациях. К тому же, зачастую использование недокументированного средства становится не самым лучшим способом выполнения намеченного задания, а часть таких средств может даже оказаться полностью избыточной (например, свойство `TriggerInsertOrder`). Радость, вызванная тем, что удалось открыть недокументированную расширенную процедуру и непосредственно применить ее в коде, эксплуатируемом на производстве, быстро сменяется разочарованием после того, как обнаруживается, что происходят аварийные отказы сервера из-за нарушений доступа. А с точки зрения поддержки программного продукта "недокументированный" означает "неподдерживаемый", поэтому не следует ожидать, что поставщик будет и в дальнейшем предоставлять функциональные возможности, которые намеренно исключены из документированного набора средств программного продукта. Общий вывод состоит в следующем — недокументированные средства могут использоваться лишь в случае крайней необходимости.

Теперь, после того как автор изложил свой подход к использованию недокументированных средств, перейдем к обсуждению вопроса о том, как найти недокументированные средства SQL Server. Задача поиска таких средств оказывается неожиданно легкой, поэтому на первых порах даже возникает удивление, какое огромное количество скрытых функциональных возможностей лежит прямо перед вами.

ПРИМЕЧАНИЕ. Автор может на полном основании утверждать, что компания Microsoft намеревается в значительной части ограничить доступ к недокументированным средствам в следующем выпуске SQL Server. Возможно, этот шаг принесет больше пользы, чем вреда, но на данное время указанные средства все еще существуют и остаются легко доступными, поэтому рассмотрим способы выявления скрытых возможностей, позволяющие лучше понять, как работает этот программный продукт.

Богатства, скрытые в таблице `syscomments`

Обильный источник недокументированных средств, команд, функций и синтаксических конструкций можно найти в системной таблице `syscomments`. Именно в таблице `syscomments` хранится исходный код каждого процедурного объекта (каждой хранимой процедуры, триггера, представления, определяемой пользователем функции, значения по умолчанию и объекта правила) базы данных. Собственная копия такой таблицы имеется в каждой базе данных. Автор

предпочитает пользоваться таблицами syscomments из баз данных master, msdb и баз данных распределительного сервера системы репликации. Кроме того, большой интерес представляют недокументированные средства, скрывающиеся в хранимых процедурах, представлениях и триггерах из баз данных серверов публикаций и серверов подписчиков, которые сформированы в ходе развертывания системы репликации.

Недокументированные команды DBCC

Автор обнаружил широкий перечень недокументированных команд, изучая таблицу master..syscomments. Например, чтобы найти все команды DBCC (на сервере, поддержка имен в котором организована с учетом регистра), вызываемые процедурными объектами в базе данных master, можно выполнить следующую команду T-SQL:

```
SELECT OBJECT_NAME(id),
SUBSTRING(text,PATINDEX('%dbcc%',text),50) as text
FROM master..syscomments
WHERE PATINDEX('%dbcc%',text)<>0
```

Недокументированные флажки трассировки

Чтобы найти только те вызовы, которые относятся к команде DBCC TRACEON (и тем самым найти ссылки на недокументированные флажки трассировки), можно выполнить следующий запрос:

```
SELECT OBJECT_NAME(id),
SUBSTRING(text,PATINDEX('%traceon%',text),50) as text
FROM master..syscomments
WHERE PATINDEX('%traceon%',text)<>0
```

Другие недокументированные объекты таблицы syscomments

Удобная возможность состоит в том, что некоторые процедурные объекты в базах данных master, msdb и базах данных распределительного сервера содержат информацию о том, используются ли в них недокументированные средства, поскольку включают где-то в своем исходном коде слово "undocumented" или слова "DO NOT DOCUMENT". Автор пришел к выводу, что это — весьма полезная особенность. Подобные процедурные объекты можно найти, выполнив примерно такой запрос:

```
SELECT OBJECT_NAME(id),
SUBSTRING(text,PATINDEX('%document%',text),50) as text
FROM master..syscomments
WHERE PATINDEX('%document%',text)<>0
```

Недокументированные объекты таблицы sysobjects

Системная таблица sysobjects имеется в каждой базе данных и содержит по одной строке для каждого объекта из базы данных. Имея некоторое представление о том, каково имя или тип некоторого недокументированного объекта, можно проверить его наличие в дереве объектов программы Enterprise Manager и в окне Object Browser программы Query Analyzer. Кроме того, для получения такой информации можно непосредственно выполнить запрос к таблице sysobjects, поскольку в конечном итоге именно к этой таблице обращаются программы Enterprise Manager и Query Analyzer. Для упрощения в приведенном ниже коде этого раздела непосредственно выполняются запросы к таблице sysobjects.

Недокументированные расширенные процедуры

Расширенные процедуры должны находиться в базе данных master и обычно имеют имя с префиксом xp_. Ниже приведен запрос к таблице sysobjects, который возвращает имена расширенных процедур, зарегистрированных в базе данных master.

```
SELECT LEFT(name, 30)
FROM master..sysobjects
WHERE TYPE='X'
```

Заслуживает внимания то, что некоторые из этих процедур имеют префиксы sp_, а не традиционный префикс xp_. Кроме того, некоторые из расширенных процедур реализованы внутри программы сервера, а не во внешних библиотеках DLL. Расширенные процедуры, реализованные внутри программы SQL Server, называются *специальными процедурами*. Для получения списка специальных процедур, зарегистрированных в базе данных master, можно выполнить примерно такой запрос:

```
SELECT OBJECT_NAME(c.id)
FROM master..syscomments c JOIN master..sysobjects o ON
    (c.id=o.id)
WHERE o.type='X'
AND c.text NOT LIKE '%.dll%' OR c.text IS NULL
```

Имя библиотеки DLL, содержащей обычную расширенную процедуру, приведено в столбце text таблицы syscomments, относящемся к расширенной процедуре. А специальные процедуры не находятся в какой-либо библиотеке DLL, поэтому относящиеся к ним записи в таблице syscomments содержат что-то иное, кроме имени библиотеки DLL. Приведенный выше запрос выводит на внешнее устройство такие записи, относящиеся к расширенным процедурам, из таблицы syscomments, которые соответствуют указанному условию.

Недостаточно знать о том, что существует какая-либо расширенная процедура, желательно также знать, для чего она предназначена. Многие расширенные процедуры вызываются обычными процедурными объектами, такими как хранимые процедуры и определяемые пользователем функции. Ниже приведен запрос T-SQL,

позволяющий получить список всех расширенных процедур с именами, начинающимися с префикса xp_, которые вызываются объектами из базы данных master.

```
SELECT OBJECT_NAME(id),
SUBSTRING(text,PATINDEX('%xp_%',text),50) as text
FROM master..syscomments
WHERE text LIKE '%xp\_%' ESCAPE '\'
```

В данном запросе автор использовал управляющий символ, заданный с помощью конструкции ESCAPE, чтобы исключить ложные совпадения, связанные с тем, что символ "_" представляет собой подстановочный символ.

Недокументированные функции

Недокументированные функциональные возможности часто скрываются также в недокументированных определяемых пользователем функциях. Некоторые из них представляют собой объекты непривилегированного режима, а другие являются системными функциями. Имена системных функций начинаются с префикса fn_, а сами системные функции принадлежат псевдопользователю system_function_schema. Чтобы узнать, какие из этих функций существуют в конкретной базе данных, можно выполнить простой запрос применительно к соответствующей таблице sysobjects. Ситуация, в которой имена, начинающиеся с префикса fn_, относятся к другим объектам, кроме функций, встречаются редко, поэтому обычно достаточно выполнить поиск исключительно только по префиксу имени. Ниже приведен запрос, который возвращает из базы данных master..sysobjects список всех объектов, имена которых начинаются с префикса fn_. Поскольку возможно создать недокументированную функцию, не принадлежащую псевдопользователю system_function_schema, в этом запросе не предусмотрено ограничение поиска с учетом конкретного владельца.

```
SELECT LEFT(name, 30)
FROM master..sysobjects
WHERE LEFT(name, 3) = 'fn_'
```

Как и при использовании расширенных процедур, недостаточно знать только о существовании функции, не зная о ее назначении. Ниже приведен запрос, который показывает все процедурные объекты в базе данных master, которые ссылаются на объекты с именами, начинающимися с префикса fn_.

```
SELECT OBJECT_NAME(id), SUBSTRING(text,PATINDEX('%fn_%',text),50)
    as text
FROM master..syscomments
WHERE text LIKE '%fn\_%' ESCAPE '\'
```

Безусловно, этот запрос (и все другие, приведенные в данной главе, которые действуют аналогичным образом) может возвращать неверные результаты поиска из-за случайных совпадений с подстроками в тексте таблицы syscomments. Очевидно, что читатель должен проверять результаты, полученные при выполнении таких запросов, и устранять все возвращаемые ими строки со случайными совпадениями.

Поддержка сценариев для недокументированных и системных объектов

В программе Enterprise Manager отменяется пункт меню Generate SQL Script (Сформировать сценарий SQL) для объектов, обозначенных как системные объекты (таковыми являются объекты, для которых установлен системный бит с помощью вызова процедуры `sp_MS_marksystemobject`). Программа Enterprise Manager все еще позволяет дважды щелкнуть кнопкой мыши на обозначении системного объекта в окне Enterprise Manager, чтобы вывести на экран исходный код объекта, но не дает возможности редактировать исходный код этого объекта. Такая организация работы позволяет исключить возможность непреднамеренной модификации важных внутренних объектов. Диалоговое окно Properties, относящееся к хранимым процедурам и аналогичным объектам, остается очень неудобным в использовании, поэтому автор старается не подвергать себя ненужным затруднениям в попытке просматривать или редактировать исходный код процедурных объектов с помощью этого диалогового окна. Кроме того, диалоговое окно Properties является модальным (и поэтому не имеет кнопки развертывания); в нем также отсутствует регулирующий размеры захват в нижнем правом углу, а такой захват всегда должна иметь оконная форма с изменяемыми размерами, заслуживающая названия таковой. Поэтому, чтобы не приходилось преодолевать сложности, связанные с использованием программы Enterprise Manager, автор обычно предпочитает вывести код сценарной поддержки хранимых процедур в файл, чтобы затем просматривать или редактировать этот код с помощью другого инструментального средства. В программе Enterprise Manager предпринимаются попытки помешать пользователю нанести хоть малейший вред себе самому в ходе работы с системными процедурами. Но как можно быстро просмотреть или отредактировать текст системной хранимой процедуры, если нельзя легко сформировать сценарий и не хочется прибегать к использованию диалогового окна Stored Procedure Properties программы Enterprise Manager, имеющего столь ограниченные возможности? К счастью, программа Query Analyzer не обнаруживает такой же заботы о безопасности работы пользователя в системе, как программа Enterprise Manager. Окно Object Browser программы Query Analyzer позволяет сформировать сценарий для любого незашифрованного объекта, имеющегося на сервере, включая системные объекты. Поэтому, если возникает необходимость просматривать или редактировать код процедурных объектов (особенно системных объектов) с использованием применимого для этого редактора, автор обращается к программе Query Analyzer.

Еще один способ вывести на внешнее устройство исходный код для системного процедурного объекта состоит в использовании процедуры `sp_helptext`. Эту процедуру можно применять для ознакомления с исходным кодом любого незашифрованного процедурного объекта в базе данных.

Россыпь сокровищ программы Profiler

Превосходный способ ознакомления с тем, как используются недокументированные средства в программе SQL Server, состоит в регистрации хода выполнения хранимых процедур и пакетов команд на сервере с помощью программы Profiler, особенно применительно к инструментальным средствам, включенным в поставку SQL Server. К числу таких инструментальных средств, к которым часто обращается автор, относятся Enterprise Manager и Index Tuning Wizard. Некоторые скрытые функциональные возможности иногда обнаруживаются также во время наблюдения за работой агентов репликации.

Изучение трассировок Profiler, перехваченных в ходе эксплуатации подобных инструментальных средств, позволяет выявить интересные дополнительные сведения о том, как работает сервер. Например, наблюдая за трассировкой Profiler во время эксплуатации программы-мастера Index Tuning Wizard, автор обнаружил, что эта программа создает так называемые "гипотетические" индексы, т.е. основанные только на статистических данных индексы, используемые для определения того, позволяет ли другая стратегия индексации повысить производительность данного конкретного запроса или снизить рабочую нагрузку. (Для определения того, является ли индекс гипотетическим, можно применить свойство `IsHypothetical` с именем `INDEXPROPERTY`.) Программа-мастер Index Tuning Wizard создает такие индексы, вызывая команду `CREATE INDEX` с использованием недокументированной опции из конструкции `WITH`, т.е. опции `statistics_only`. Эта информация позволяет кое-что узнать о том, как работает сервер. Например, если бы автор еще не знал об этом, то, обнаружив указанную ситуацию, мог бы понять, что оптимизатор не обращается к данным при принятии решения о том, следует ли использовать индекс, иными словами, оптимизируя план запроса, оптимизатор проверяет только статистические данные, относящиеся к индексу. Кроме того, становится понятным, благодаря чему указанная программа-мастер приобретает способность оценить такое большое количество разных типов индексов за столь короткий (вернее, относительно короткий) промежуток времени: для создания проверочных индексов не используются какие-либо операции доступа к самим данным. Поэтому, открыв указанное недокументированное средство, можно кое-что узнать не только о программе-мастере Index Tuning Wizard, но и о самой программе SQL Server.

Программа Profiler является также удобным средством определения того, как используются недокументированные расширенные процедуры и функции. В частности, недокументированные расширенные процедуры и функции широко применяются средствами SQL-DMO и программой Enterprise Manager; перехват трассировки Profiler в процессе прогона программы Enterprise Manager (в различные периоды ее эксплуатации) позволяет выявлять скрытые возможности разного рода.

Но автор, изучая трассировки программы Profiler, обнаружил одну любопытную особенность — в исходном коде этого инструментального средства, очевидно, заложено требование, чтобы оставались скрытыми строки, содержащие текст "`-- sp_password`". Учитывая то, что этот текст отображается применительно к событиям `SP:StmtStarting/StmtCompleted`, относящимся к системной процедуре

sp_password, по-видимому, такое требование реализовано в коде в целях предотвращения регистрации случаев смены пароля в трассировках Profiler. Тем не менее само указанное требование реализовано столь примитивно, что взломщик получает возможность скрывать следы применяемых им трассировок, добавляя подстроку "-- sp_password" к любой строке в пакете команд T-SQL и в хранимой процедуре, следы которой он не хочет показать в трассировке.

Изучение инсталляционных сценариев

Еще одним хорошим источником недокументированной информации является набор инсталляционных сценариев, который входит в поставку программы SQL Server. Эти сценарии можно найти в подкаталоге Install корневого каталога инсталляции SQL Server. Применительно к этим сценариям можно провести такие же операции поиска, которые применялись к таблице syscomments для обнаружения недокументированных флажков трассировки, команд DBCC, хранимых процедур, определяемых пользователем функций и т.д. Например, автор узнал, как создавать собственные представления INFORMATION_SCHEMA (и описал этот процесс в своей последней книге), изучая сценарий ansiview.sql. Кроме того, автор определил, как создавать собственные системные функции, изучая сценарий instdist.sql. Остается еще очень много недокументированной информации, которую можно получить, изучая сценарии, поставляемые с этим программным продуктом, особенно в той части, которая касается создания объектов. Известно, что сервер каким-то образом создает многие эти объекты с помощью языка Transact-SQL. Если мы хотим использовать такие же функциональные возможности или лучше понять, как работает сервер, то вполне можем начать с инсталляционных сценариев.

Импорт библиотек DLL

Наконец, для проверки наличия недокументированных средств можно использовать таблицу импорта исполняемого файла SQL Server. Каждый исполняемый файл Windows имеет таблицу импорта. В этой таблице перечислены функции, на которые имеется явно заданная ссылка в исполняемом файле, применительно к каждой библиотеке DLL, которая должна быть загружена в приложении. Изучая эту таблицу, можно узнать, какие недокументированные функции должны быть загружены в исполняемом файле. Определив имя функции, можно воспользоваться отладчиком, чтобы установить точку останова и узнать, при каких условиях вызывается эта функция.

Наглядным примером применения описанного подхода является библиотека OPENDS60.DLL, которая входит в поставку программы SQL Server. Эта библиотека DLL реализует API-интерфейс Open Data Services (Открытые службы данных), который используется внутри сервера и служит для создания расширенных процедур. Рассматривая таблицу импорта для программы sqlservr.exe (исполняемого файла SQL Server), можно обнаружить, что сервер импортирует

большое количество функций из библиотеки OPENDS60.DLL, причем многие из этих функций являются недокументированными. Подключившись с помощью отладчика к программе sqlservr.exe и установив точки останова для этих функций можно определить, при каких обстоятельствах они используются. Благодаря этому появляется возможность точно узнать, для чего предназначены эти функции, а также понять, как и почему они вызываются сервером.

Резюме

Недокументированные средства следует использовать исключительно в тех обстоятельствах, когда нет другого способа добиться поставленной цели. В данном случае наилучшая стратегия состоит в том, чтобы вообще не применять эти средства, если на то нет явного указания от компании Microsoft. Применяйте исследования недокументированных средств программы SQL Server как способ более глубокого познания самого программного продукта, а не как способ поиска интересных процедур, которые можно было бы вводить без проверки в производственные системы.

Вопрос для самопроверки

1. Укажите три причины, по которым не следует использовать недокументированные средства SQL Server.

ГЛАВА 25

Программа DTSDIAG

В завершение настоящей книги будет представлено диагностическое приложение, которое может помочь в работе многим специалистам. Данное приложение основано на технологии DTS программы SQL Server и использует объектную модель DTS. Это – программа DTSDIAG, демонстрирующая все возможности приложения, созданного путем объединения технологий, на которых основана программа SQL Server. Если читатель еще не ознакомился с главой 20, посвященной технологии DTS, то рекомендуем сделать это перед переходом к изучению данной главы.

Программа DTSDIAG предназначена для сбора диагностических данных о работе SQL Server. С помощью этой программы можно одновременно обрабатывать данные счетчиков Perfmon/Sysmon; отчеты SQLDIAG; журналы событий приложения, системы и средств защиты; трассировки Profiler; а также вывод сценария обнаружения блокировок (формат вывода этого сценария определен в статьях базы знаний Microsoft Knowledge Base с номерами 251004, "*INF: How to Monitor SQL Server 7.0 Blocking*", и 271509, "*INF: How to Monitor SQL Server 2000 Blocking*").

Краткое описание

Программа DTSDIAG состоит из автономного приложения Visual Basic, четырех пакетов DTS и нескольких разнообразных инструментов с интерфейсом командной строки и сценариев, выполняемых в целях сбора требуемых диагностических данных. Приложение VB позволяет указывать версию программы SQL Server, к которой должно быть выполнено подключение, а также задавать используемую аутентификационную информацию. Запуск процесса сбора информации осуществляется путем щелчка на кнопке Start в окне приложения, а для останова приложения следует щелкнуть на кнопке Stop.

Автору часто приходилось заниматься поиском причин нарушений в работе систем, основанных на использовании программы SQL Server, и в таких случаях он ощущал потребность в применении подобного инструментального средства. При этом во многих ситуациях не удавалось привлечь кого-либо в качестве помощников при сборе диагностической информации Perfmon, Profiler и тому подобных сведений (которые обычно требуются при проведении отладки), поскольку для выполнения подобной задачи нужно иметь достаточно высокую квалификацию. Кроме того, во многих случаях те лица, которых автор пытался привлечь к этой работе, просто не могли собрать диагностические сведения из всех возможных источников. Безусловно, иногда им удавалось собрать нужные диагностические данные, но получение этих данных происходило либо не в тот период времени, в который

требовалось, либо не в одни и те же интервалы времени. Программа DTSDIAG устраняет необходимость в посторонней помощи, поскольку позволяет ввести в конфигурацию данные о том, какие диагностические сведения требуются, а затем направить все необходимое для работы этого инструментального средства на целевой компьютер. Автор указывает типы данных, которые должны быть собраны в файле INI, затем копирует исполняемый файл DTSDIAG и вспомогательные файлы на целевой компьютер, после чего вызывает программу на выполнение. На том узле, где должен быть выполнен сбор данных, остается только указать имя сервера/экземпляра (и версию), к которому должно быть выполнено подключение, и ввести информацию аутентификации, которая имеется в распоряжении специалистов по техническому обслуживанию. В результате этого процесс сбора диагностических данных становится настолько удобным, насколько это возможно, и вместе с тем позволяет задавать всю необходимую информацию о конфигурации.

Структура приложения

Итак, после ознакомления с возможностями программы DTSDIAG рассмотрим, какой исходный код имеется в приложении, относящемся к этой программе. Как уже было сказано, для запуска/останова процесса сбора диагностической информации предназначена кнопка Start/Stop приложения DTSDIAG. Ниже приведен код VB, закрепленный за этой кнопкой (листинг 25.1).

Листинг 25.1. Код VB обработчика событий кнопки Start/Stop

```
Private Sub btStartStop_Click()
If Not bRunning Then
  bRunning = True
  btStartStop.Caption = "Stop"
  ExecutePackage "dtsdiag_template.dts", "dtsdiag.dts",
      App.Path + "\dtsdiag.log"

Else
  btStartStop.Enabled = False
  ExecutePackage "dtsdiag_shutdown_template.dts",
      "dtsdiag_shutdown.dts", App.Path + "\dtsdiag_shutdown.log"
  ExecutePackage "dtsdiag_cleanup_template.dts",
      "dtsdiag_cleanup.dts", App.Path + "\dtsdiag_cleanup.log"
  bRunning = False
btStartStop.Caption = "Start"
  btStartStop.Enabled = True
End If
End Sub
```

Для запуска и останова процесса сбора информации используется одна и та же кнопка, просто изменяется надпись на кнопке в зависимости от состояния процесса сбора информации. Во время запуска процесса сбора информации вызывается процедура `ExecutePackage`, которая вызывает на выполнение пакет `dtsdiag_template.dts`. Процедура `ExecutePackage` сохраняет пакет `dtsdiag_temp-`

late.dts под именем dtsdiag.dts (причины выполнения этого действия будут вскоре описаны) и вызывает пакет dtsdiag.dts на выполнение.

Во время останова процесса сбора информации вызываются два пакета: dtsdiag_shutdown_template.dts и dtsdiag_cleanup_template.dts. Как и пакет dtsdiag_template.dts, указанные пакеты сохраняются как новые пакеты без суффикса _template, затем вновь полученные пакеты вызываются на выполнение.

Некоторые источники диагностических данных, такие как отчеты SQLDIAG и журналы системных событий, могут использоваться для сбора диагностической информации, когда приложение DTSDIAG запущено или остановлено, или в том и другом случае. Указания, касающиеся того, должен ли выполняться сбор подобных диагностических данных и когда этот сбор данных должен осуществляться, находятся в файле DTSDIAG.INI. Для сбора диагностических данных, которые указаны в конфигурации как предназначенные для сбора во время останова приложения, применяется пакет dtsdiag_shutdown_template.dts. А пакет dtsdiag_cleanup_template.dts предназначен для удаления хранимых процедур и других объектов, оставшихся после завершения процесса сбора, в то время как происходит останов приложения DTSDIAG. Кроме того, пакет dtsdiag_cleanup_template.dts проверяет наличие в системе исполняемого файла KILL.EXE (утилиты из комплекта ресурсов Windows NT 4/2000 Resource Kit, которая способна останавливать другие процессы) и предпринимает попытки уничтожить экземпляры osql (утилиты, которая используется в приложении DTSDIAG для сбора основной части диагностических данных).

Файл конфигурации DTSDIAG.INI приложения DTSDIAG имеет очень простой формат, как показано в листинге 25.2.

Листинг 25.2. Пример файла конфигурации DTSDIAG.INI

```
[DTSDIAG]
SQLDiag=1
SQLDiagStartup=0
SQLDiagShutdown=1
EventLogs=1
EventLogsStartup=0
EventLogsShutdown=1
Profiler=1
ProfilerEvents=76,75,92,94,93,95,16,22,21,33,67,55,79,80,61,69,25,
59,60,27,58,14,15,81,17,10,11,35,36,37,19,50,12,13
Perfmon=1
BlockingScript=1
BlockerLatch=0
BlockerFast=1
MaxTraceFileSize=100
MaxPerfmonLogSize=256
PerfmonPollingInterval=5
ProfilerPollingInterval=5
BlockingPollingInterval=120
Counter0=\MSSQL$%s:Buffer Manager\Buffer cache hit ratio
Counter1=\MSSQL$%s:Buffer Manager\Buffer cache hit ratio base
Counter2=\MSSQL$%s:Buffer Manager\Page lookups/sec
...
```

Формат файла, приведенного в листинге 25.2, в основном не требует пояснений. Для каждого типа диагностической информации предусмотрен отдельный логический переключатель. Например, если ключевому слову `Profiler` присвоено значение 1, то предпринимается попытка сбора данных трассировки Profiler, в противном случае такая попытка не предпринимается.

Некоторые из параметров в файле `DTSDIAG.INI` служат в качестве опций для процесса сбора. Например, параметр `ProfilerEvents` содержит разделенный запятыми список событий, информация о которых должна быть включена в трассировку Profiler (эталонный список номеров событий приведен в описании процедуры `sp_trace_setevent` в оперативной документации Books Online). Записи `CounterN` содержат списки счетчиков программы Perfmon/Sysmon, применяемых для сбора информации. Значения параметров `BlockerLatch` и `BlockerFast` представляют собой переключатели, которые указывают на необходимость выполнения сценария обнаружения блокировок (еще раз отметим, что эти параметры описаны в статьях 251004 и 271509 базы знаний Knowledge Base).

Основной процедурой в программе `DTSDIAG.EXE` является `ExecutePackage`. Ниже приведен код этой процедуры (листинг 25.3), а затем дано описание того, что выполняет эта процедура и как осуществляются соответствующие действия.

Листинг 25.3. Код процедуры `ExecutePackage`

```
Private Sub ExecutePackage(SrcName As String, TargName As String,
    LogName As String)
  Dim oPkg                As DTS.Package
  Dim oTask               As DTS.Task
  Dim oCreateProcessTask  As DTS.CreateProcessTask
  Set oPkg = New DTS.Package
  oPkg.LoadFromStorageFile SrcName, ""
  oPkg.LogFileName = LogName

  For Each oTask In oPkg.Tasks
    If 0 <> InStr(1, oTask.Name, "CreateProcess",
        vbTextCompare) Then
      Set oCreateProcessTask = oTask.CustomTask
      oCreateProcessTask.ProcessCommandLine =
          TranslateVars(oCreateProcessTask.ProcessCommandLine)
    End If
  Next

  Dim oFs
  Set oFs = CreateObject("Scripting.FileSystemObject")

  If oFs.FileExists(TargName) Then
    Kill TargName  ' Удалить файл заранее, чтобы его размеры не
                   ' увеличивались до бесконечности
  End If

  Set oFs = Nothing

  oPkg.SaveToStorageFile TargName
  oPkg.Execute
  oPkg.UnInitialize
  Set oPkg = Nothing
End Sub
```

Выполнение процедуры начинается с создания экземпляра объекта DTS Package. Безусловно, для этого объекта предусмотрен более новый интерфейс Package2 (который впервые появился в версии SQL Server 2000), но в данной процедуре для разработки кода применяется интерфейс Package, который позволяет эксплуатировать программу DTSDIAG в версии SQL Server 7.0.

После создания объекта Package из структурированного файла хранения загружается указанный пакет. Все пакеты, применяемые в программе DTSDIAG, хранятся в формате структурированного файла хранения COM.

Затем осуществляется обработка в цикле задач, определенных в пакете, и распознавание каждой задачи Execute Process путем поиска подстроки CreateProcess в имени задачи. Доступ к каждой задаче Execute Process выполняется путем присваивания ранее объявленной переменной DTS.CreateProcessTask значения свойства CustomTask универсального объекта задачи из коллекции Package.Tasks.

В качестве дополнительного пояснения отметим, что обработка в цикле задач Execute Process в каждом пакете выполняется в целях подстановки значений вместо некоторых параметров в свойстве ProcessCommandLine перед вызовом пакета на выполнение. Дело в том, что для сбора диагностических данных с помощью программы DTSDIAG требуется выполнение сложных сценариев и выборка выходных данных, представленных в этих сценариях в виде переменных, поэтому в большинстве прогонов сценариев T-SQL в программе DTSDIAG невозможно использовать типичную задачу Execute SQL. Вместо этого приходится выходить в командный интерпретатор и вызывать программу OSQL.EXE. Безусловно, необходимо, чтобы перед вызовом osql была возможность провести настройку конфигурации, например, необходимо иметь возможность указать сервер и экземпляр, к которому должно быть выполнено подключение, задать опции для некоторых из диагностических хранимых процедур, вызываемых на выполнение, и т.д. Выполнение указанных функций можно было бы существенно упростить с помощью одного из определяемых пользователем объектов задач, описанных выше в данной книге, но для этого потребовалось бы инсталлировать определяемую пользователем задачу на целевом компьютере перед вызовом на выполнение пакетов, содержащих эту задачу. Но автор стремился избежать необходимости регистрировать в системе COM-объекты для получения возможности собирать диагностическую информацию, поэтому в приложении DTSDIAG не применяются какие-либо определяемые пользователем задачи. Вместо этого в указанном приложении используются обычные задачи Execute Process и задаются параметры в свойстве ProcessCommandLine в форме, аналогичной определяемым пользователем задачам ExecuteSQLScript и ExecuteScript, которые были описаны выше в настоящей книге. В рассматриваемом коде VB происходит обработка в цикле указанных задач и замена параметров командной строки соответствующими значениями перед вызовом пакета на выполнение.

Заслуживает также внимания вызов функции TranslateVars. Функция TranslateVars обеспечивает замену параметров в каждом свойстве ProcessCommandLine соответствующими значениями. В действительности эта функция намного сложнее по сравнению с задачей ExecutePackage; краткое описание работы этой функции приведено ниже.

После надлежащей замены параметров значениями в свойстве ProcessCommandLine, относящемся к каждой задаче Execute Process, преобразованный пакет записывается под именем целевого пакета и вызывается на выполнение. После завершения выполнения пакета объект пакета уничтожается, и происходит возврат управления.

Как уже было сказано, процедура TranslateVars обеспечивает замену параметров в свойстве ProcessCommandLine каждой задачи Execute Process соответствующими значениями. Это означает, что, например, процедура заменяет параметр %server_instance% именами сервера и экземпляра, к которым должно быть выполнено подключение. Аналогично эта процедура заменяет параметр %auth_string% соответствующей строкой аутентификации, которая должна быть передана в командной строке osql.

Некоторые значения, необходимые для замены параметров, поступают из файла конфигурации DTSDIAG.INI. Поэтому рассматриваемый код содержит оператор импорта Declare Function библиотеки DLL, относящийся к функции GetPrivateProfileString API-интерфейса, которая представляет собой функцию подсистемы Win32, предназначенную для выборки значений из файла INI. В листинге 25.4 показан исходный код процедуры TranslateVars и импортируемой функции GetPrivateProfileString.

Листинг 25.4. Исходный код процедуры TranslateVars и импортируемой функции GetPrivateProfileString

```
Private Declare Function GetPrivateProfileString Lib "KERNEL32" _
      Alias "GetPrivateProfileStringA" (ByVal AppName As String, _
      ByVal KeyName As String, ByVal keydefault As String, _
      ByVal ReturnString As String, ByVal NumBytes As Long, _
      ByVal FileName As String) As Long
Private Function TranslateVars(CmdLine As String) As String

  Dim strServer As String
  Dim strInstance As String
  Dim strProfilerParms As String
  Dim strBlockerParms As String
  Dim iBlockerPollingIntervalSeconds As Integer
  Dim iBlockerPollingIntervalMinutes As Integer
  Dim strWork As String

  ' Значения параметров файла INI, заданные по умолчанию
  strProfilerParms = ""
  strBlockerParms = ""
  iBlockerPollingIntervalSeconds = 0
  iBlockerPollingIntervalMinutes = 0

  Const BUFFSIZE = 1024

  strWork = Space(BUFFSIZE)

  ' Получить параметры программы Profiler

  ' События
```

```
Res = GetPrivateProfileString("DTSDIAG", "ProfilerEvents", "", _
       strWork, BUFFSIZE, App.Path + "\dtsdiag.ini")

If (0 <> Res) Then
  strProfilerParms = ", @Events=" + Chr(39) + Mid(strWork, 1, _
      Res) + Chr(39)
End If

' MaxTraceFileSize
strWork = Space(BUFFSIZE)
Res = GetPrivateProfileString("DTSDIAG", "MaxTraceFileSize", "", _
       strWork, BUFFSIZE, App.Path + "\dtsdiag.ini")

If (0 <> Res) Then
  strProfilerParms = strProfilerParms + ", @MaxFileSize=" + _
      Mid(strWork, 1, Res)
End If

' Получить параметры программы Blocker
' BlockerLatch
  strWork = Space(BUFFSIZE)
  Res = GetPrivateProfileString("DTSDIAG", "BlockerLatch", "", _
       strWork, BUFFSIZE, App.Path + "\dtsdiag.ini")

  If (0 <> Res) Then
  strBlockerParms = "@latch=" + Mid(strWork, 1, Res)
  End If

  ' BlockerFast
  strWork = Space(BUFFSIZE)
  Res = GetPrivateProfileString("DTSDIAG", "BlockerFast", "", _
       strWork, BUFFSIZE, App.Path + "\dtsdiag.ini")

  If (0 <> Res) Then
  strBlockerParms = strBlockerParms + ", @fast=" + _
      Mid(strWork, 1, Res)
  End If

  ' BlockingPollingInterval
  strWork = Space(BUFFSIZE)
  Res = GetPrivateProfileString("DTSDIAG", _
       "BlockingPollingInterval", "", _
       strWork, BUFFSIZE, App.Path + "\dtsdiag.ini")

  If (0 <> Res) Then
  iBlockerPollingIntervalSeconds = Val(Mid(strWork, 1, Res))

  ' Поскольку производится вставка значения времени, максимальная
  ' величина равна 59
  If iBlockerPollingIntervalSeconds > 59 Then
     iBlockerPollingIntervalMinutes = _
         iBlockerPollingIntervalSeconds / 60
    iBlockerPollingIntervalSeconds = _
         iBlockerPollingIntervalSeconds Mod 60
  End If
  End If
```

```
    ' Извлечь данные об именах сервера и экземпляра из элемента
    ' управления ServerInstance типа TextBox
    Dim iPos As Integer

    iPos = InStr(1, tbServerInstance.Text, "\")

    If 0 <> iPos Then
    strServer = Mid(tbServerInstance.Text, 1, iPos - 1)
    strInstance = Mid(tbServerInstance.Text, iPos + 1)
Else
     strServer = tbServerInstance.Text
    strInstance = ""
  End If

  ' Выполнить подстановку значений параметров
  CmdLine = Replace(CmdLine, "%auth_string%", strAuth)
  CmdLine = Replace(CmdLine, "%ver%", strVer)
  CmdLine = Replace(CmdLine, "%server_instance%", _
      tbServerInstance.Text)
  If taVersion.SelectedItem.Index = 1 Then
     CmdLine = Replace(CmdLine, "%trace_output%", App.Path & _
        "\output\" & "sp_trace.trc")
  Else      ' Исключить расширение имени файла для компонента имени 80
     CmdLine = Replace(CmdLine, "%trace_output%", App.Path & _
        "\output\" & "sp_trace")
  End If
  CmdLine = Replace(CmdLine, "%server%", strServer)
  CmdLine = Replace(CmdLine, "%instance%", strInstance)
  CmdLine = Replace(CmdLine, "%profilerparms%", strProfilerParms)
  CmdLine = Replace(CmdLine, "%blockerparms%", strBlockerParms)
  CmdLine = Replace(CmdLine, "%bis%", _
      Str(iBlockerPollingIntervalSeconds))
  CmdLine = Replace(CmdLine, "%bim%", _
      Str(iBlockerPollingIntervalMinutes))

  TranslateVars = CmdLine
End Function
```

После выборки всех необходимых значений параметров конфигурации из файла DTSDIAG.INI в функции TranslateVars используется функция Replace языка VB для замены каждого параметра соответствующим значением. Работа функции TranslateVars завершается тем, что происходит возврат в качестве результата той командной строки, в которой выполнена замена параметров.

Может возникнуть вопрос, почему в соответствующих пакетах DTS не используется просто задача Dynamic Properties, поскольку указанные значения из файла INI должны быть в конечном итоге введены внутрь пакетов. Ведь нельзя отрицать, что задача Dynamic Properties позволяет осуществлять выборку значений непосредственно из файла INI, не требуя применения какого-либо кода Automation. Вместо разработки внешнего приложения, которое динамически модифицирует пакеты с использованием технологии COM Automation, было бы, наверное, проще использовать задачу Dynamic Properties внутри каждого пакета, если потребуется чтение значений параметров конфигурации из файла INI? Ответ состоит в том, что задача

Dynamic Properties действительно применяется, когда это возможно. Но многие значения параметров конфигурации, которые требуются для работы, должны быть вставлены вместо параметров непосредственно в значения свойств, поэтому указанные значения нельзя просто получить с помощью задачи Dynamic Properties. Дело в том, что использование задачи Dynamic Properties для присваивания свойству значения параметра конфигурации из файла INI осуществимо, только если присваивается значение для всего свойства. А для присваивания значения только для части свойства требуется сценарий или внешний код Automation.

Теперь, после краткого обзора исходного кода VB для приложения DTSDIAG рассмотрим, какие пакеты DTS применяются в этом приложении. Откройте пакет `dtsdiag_template.dts` (находящийся в подкаталоге `CH25\dtsdiag` компакт-диска, прилагаемого к данной книге) в программе DTS Designer, чтобы следить за обсуждением некоторых наиболее важных особенностей этого пакета.

Работа пакета начинается с создания подкаталога `OUTPUT` в каталоге запуска. Если каталог `OUTPUT` уже существует, то этот каталог удаляется и снова создается. Указанный каталог должен содержать все файлы с данными, собранными приложением DTSDIAG. Выходные файлы задач, выполняемых в целях настройки процесса сбора диагностической информации (например, для создания хранимых процедур), имеют имена с префиксом ##. Это позволяет легко отличить указанные файлы от диагностических файлов, которые нас действительно интересуют. Обычно после завершения процесса сбора следует удалять файлы с префиксом ##, поскольку они требуются, только если возникают какие-либо проблемы при использовании приложения DTSDIAG.

Следует отметить, что для загрузки значений параметров конфигурации из файла `DTSDIAG.INI` используется задача Dynamic Properties. Как уже было сказано выше, в рассматриваемом приложении с помощью задачи Dynamic Properties осуществляется загрузка максимально возможного количества значений параметров конфигурации. С диагностической информацией каждого типа связана глобальная переменная, которая управляет тем, должен ли осуществляться сбор этой информации. Например, глобальная переменная `sqldiag` указывает, должна ли вызываться на выполнение программа `SQLDIAG.EXE`. Задача Dynamic Properties задает значение глобальной переменной `sqldiag`, считывая файл `DTSDIAG.INI` и осуществляя выборку значения ключа `SQLDiag`.

В данном пакете процессы Blocker, Profiler и SQLDIAG начинают свою работу с вызова утилиты `osql` для создания хранимых процедур, которые будут вызываться этими процессами для сбора требуемых данных. Процесс Blocker создает две хранимые процедуры. Одной из них является процедура `sp_code_runner`, способная вызывать на выполнение другие процедуры или код T-SQL либо по расписанию, либо после определения истинного значения какого-то логического условия, а второй является либо процедура `sp_blocker_pss70`, либо процедура `sp_blocker_pss80` (в зависимости от версии SQL Server, к которой выполняется подключение). Обе последние процедуры представляют собой хранимые процедуры обнаружения блокировок, которые описаны в статьях Knowledge Base, указанных выше. Процедуры `sp_blockerXXXX` являются собственностью компании Microsoft, поэтому автор не включил их в состав компакт-диска, прилагаемого к данной книге. Читатель может обратиться к упомянутым выше статьям Knowledge

Base по адресу `http://www.microsoft.com` и загрузить указанные процедуры самостоятельно, чтобы использовать приложение DTSDIAG в целях эксплуатации этих процедур. Кроме того, читатель может подготовить собственную процедуру (процедуры) обнаружения блокировок, поскольку в приложении DTSDIAG нет ничего, что требовало бы применения исключительно хранимых процедур Microsoft.

Следует отметить, что исполняемый файл `SQLDIAG.EXE` не вызывается непосредственно из пакета DTS. Программа `SQLDIAG.EXE` должна эксплуатироваться в своем базовом приложении SQL Server, поскольку для вызова этой программы непосредственно из самого пакета потребовалось бы, чтобы этот пакет работал на сервере, а это как раз и нежелательно. Вместо этого вызывается хранимая процедура, которая выходит в тот командный интерпретатор, под управлением которого эксплуатируется программа `SQLDIAG.EXE` на сервере; для этого используется внешняя процедура `xp_cmdshell`. Это позволяет собирать данные из отчетов SQLDIAG с помощью программы, которая физически не выполняется на компьютере SQL Server. Обратите внимание на то, что для применения такого метода требуется осуществить дополнительные действия в кластере SQL Server 2000 (см. статью 233332 из базы знаний Knowledge Base).

Задача Perfmon вызывает на выполнение определяемую пользователем утилиту, написанную автором на языке C++ (также включенную в состав компакт-диска, прилагаемого к данной книге), которая собирает данные о значениях заданного набора счетчиков Perfmon и записывает эти значения в журнал Perfmon, имеющий формат BLG. Указанная утилита носит имя PMC и аналогична утилите LogMan, которая поставляется в составе Windows XP и более поздних версий (см. статью 303133 базы знаний Knowledge Base), но работает также в Windows 9x и более поздних версиях. Следует отметить, что в связи с внедрением Windows XP компания Microsoft внесла изменения в файлы заголовков, поэтому для эксплуатации утилиты PMC в Windows XP или более поздних версиях необходимо получить версию библиотеки `PDH.DLL` (это — библиотека Performance Data Helper, которая реализует машину, лежащую в основе программ Perfmon/Sysmon), поставляемую в составе Windows 2000. Для удобства автор включил файл `PDH.DLL` в каталог `dtsdiag` компакт-диска, прилагаемого к данной книге. Если читатель примет решение эксплуатировать в Windows XP или в более поздних версиях утилиту PMC (вместо эксплуатации утилиты LogMan), автор рекомендует использовать версию `PDH.DLL`, включенную в состав приложения DTSDIAG. Для этого не требуется заменять версию `PDH.DLL`, поставляемую в составе используемой операционной системы, той версией, которая включена в прилагаемый компакт-диск. Достаточно оставить файл `PDH.DLL` в каталоге запуска DTSDIAG, и утилита PMC найдет его после запуска.

Утилита PMC считывает имя файла INI, переданного в качестве параметра (в данном случае `DTSDIAG.INI`), находит в файле INI значения параметров `CounterN` и добавляет каждое из этих значений в журнал Perfmon в формате BLG. Если утилита находит в имени счетчика строку "`%s`", то заменяет ее именем указанного экземпляра SQL Server (при желании это значение может быть передано в командной строке), прежде чем ввести эти данные в журнал Perfmon. Если имя экземпляра не задано, но утилита PMC обнаруживает подстроку "`%s`" в имени счетчика, то принимает предположение, что задан используемый по умолчанию экземпляр

SQL Server, и заменяет всю подстроку "MSSQL$%s" значением "SQLServer", чтобы ввести счетчик, относящийся к заданному по умолчанию экземпляру.

Сбор данных из журнала событий осуществляется с использованием утилиты elogdmp.exe, которая включена в комплект ресурсов Windows 2000 Resource Kit. И эта утилита является собственностью компании Microsoft, поэтому автор не включил ее в компакт-диск, прилагаемый к данной книге. В действительности для получения дампа журнала событий может использоваться любая утилита (например, применима также утилита dumpel.exe из комплекта ресурсов Windows NT 4 Resource Kit), достаточно просто выполнить соответствующим образом настройку конфигурации задач Execute Process, относящихся к журналу событий.

Следует отметить, что сбор информации из журналов событий осуществляется с помощью задачи Execute Package, которая запускает отдельный пакет, выполняющий сбор информации из всех трех журналов параллельно. Это связано с тем, что задача сбора информации из журналов событий является одной из тех задач, которые позволяют накапливать информацию во время запуска, во время останова или в то и другое время. Поэтому, чтобы обеспечить сбор информации из журналов событий с помощью пакета dtsdiag_template.dts, а также dtsdiag_shutdown_template.dts, задачи сбора информации из журналов событий выведены из их собственного пакета, который выполняется по мере необходимости во время запуска или останова.

Еще одной особенностью приложения DTSDIAG, заслуживающей внимания, является тот способ, с помощью которого используются ассоциации потока данных сценария ActiveX для того, чтобы можно было разрешать/запрещать использование некоторых путей исполнения кода внутри пакетов. Выше в данной книге указанный здесь способ описан более подробно. В программе DTSDIAG этот способ, например, служит для запрещения применения пути задачи Profiler, если параметру Profiler в файле DTSDIAG.INI присвоено значение 0 (ложное значение). В листинге 25.5 представлен сценарий ActiveX, который связан с задачей Execute Process для процедуры Create Profiler Proc.

Листинг 25.5. Сценарий ActiveX

```
Function Main()
  If DTSGlobalVariables("profiler") Then
    Main = DTSStepScriptResult_ExecuteTask
  Else
    Main = DTSStepScriptResult_DontExecuteTask
  End If
End Function
```

Как показывает листинг 25.5, в задаче Dynamic Properties присваивается значение глобальной переменной profiler в начале обработки пакета. Если значение этой переменной не равно нулю, то используется путь задачи Profiler; в противном случае этот путь пропускается.

Что касается задач, которые могут быть выполнены при запуске, останове или в том и другом случае, то необходимо проверить вторую глобальную переменную в целях определения того, следует ли выполнять эти задачи. В листинге 25.6 показан сценарий ActiveX, связанный со строкой вызова задачи SQLDIAG.

Листинг 25.6. Сценарий ActiveX, относящийся к задаче SQLDIAG

```
Function Main()
  If (DTSGlobalVariables("sqldiag")) And _
  (DTSGlobalVariables("sqldiagstartup")) Then
    Main = DTSStepScriptResult_ExecuteTask
  Else
    Main = DTSStepScriptResult_DontExecuteTask
  End If
End Function
```

Итак, как показано в листинге 25.6, проверяется не только глобальная переменная `sqldiag`, но и глобальная переменная `sqldiagstartup` (или `sqldiagshutdown`) для определения того, следует ли собирать данные из отчетов SQLDIAG при выполнении данного конкретного шага. В пакете `dtsdiag_template.dts` проверяется глобальная переменная `sqldiagstartup`, а в пакете `dtsdiag_shutdown_template.dts` проверяется глобальная переменная `sqldiagshutdown`.

Резюме

На этом завершается краткое описание приложения DTSDIAG. Читатель может вызвать это вспомогательное приложение на выполнение, чтобы провести с ним дальнейшие эксперименты, а также загрузить различные пакеты приложения в программу DTS Designer, чтобы узнать, какую структуру имеют эти пакеты. Кроме того, можно ознакомиться с кодом VB приложения, чтобы узнать, как осуществляется управление пакетами DTS с помощью средств Automation. Исходный код и вспомогательные файлы для приложения DTSDIAG находятся в подкаталоге `CH25\dtsdiag` компакт-диска, прилагаемого к данной книге.

Естественным этапом развития идей, реализованных в приложении DTSDIAG, была бы загрузка собранных данных в программу SQL Server для анализа. Оставляем решение указанной задачи в качестве упражнения для читателя, но приведем несколько подсказок для тех, кто на это решится. Журналы событий и отчеты SQLDIAG представляют собой простые текстовые файлы, которые можно легко импортировать в таблицы SQL Server после небольшой обработки. Вывод сценария обнаружения блокировок также может быть обработан в виде текста и импортирован в набор таблиц SQL Server, хотя такая операция немного сложнее из-за большего разнообразия форматов вывода. Любую трассировку Profiler можно прочитать в виде набора строк с помощью функции `fn_trace_gettable` языка T-SQL, поэтому импорт трассировки в таблицу осуществляется предельно просто. Журнал Perfmon в формате BLG может быть преобразован в формат CSV с помощью инструментального средства Relog, входящего в поставку Windows XP и более поздних версий (см. статью 303133 базы знаний Knowledge Base), а данные в формате CSV могут быть затем импортированы в программу SQL Server с помощью средств DTS. После ввода всех указанных данных в базу данных SQL Server возможности применения разнообразных сложных средств анализа к диагностическим данным становятся буквально безграничными. Труднее всего вначале собрать эти данные.

Предметный указатель

4

4GT
  4GB Tuning, *174*

A

ACID
  Atomic, Consistent, Isolated, Durable, *554*
ACL
  Access Control List, *165*
APC
  Asynchronous Procedure Call, *121*; *257*
API-интерфейс, *49*
  AWE, *176*
  BCP, *641*
  LoadLibrary, *63*
  LoadLibraryEx, *62*
  OS/2 Presentation Manager, *57*
  Win32, *57*
  Winsock, *360*; *380*
  для XML простой, *710*
  планирования, *105*
  сетевой, *362*
AWE
  Address Windowing Extensions, *51*; *161*; *172*

B

B
  balanced, *520*
BCP
  Bulk Copy Program, *641*
BNS
  Bug Notification Service, *839*
Bosak J., *424*
BPool
  Buffer Pool, *484*
BSD
  Berkeley Software Distribution, *365*
B-дерево, *520*

C

Campbell D., *24*
CIFS
  Common Internet File System, *365*
CLI
  Call-Level Interface, *360*
COM
  Component Object Model, *50*
COM-объект, *416*; *621*; *868*
  SQLXMLBulkLoad, *790*
COM-посредник, *415*; *621*
COM-сервер
  внепроцессный, *415*; *622*
  внутрипроцессный, *416*; *622*
Connolly D., *424*
CR3
  Control Register 3, *177*
CRC
  Cyclic Redundancy Check, *967*

D

DCE
  Distributed Computing Environment, *400*
DCOM
  Distributed COM, *400*; *414*; *624*
DDE
  Dynamic Data Exchange, *405*
DDQ
  Data Driven Query, *830*
DHTML
  Dynamic HTML, *423*
DLL
  Dynamic-Link Library, *57*

DMO
Distributed Management Objects, *629*
DOM
Document Object Model, *422*; *444*; *710*
DPV
Distributed Partitioned View, *696*
DRI
Declarative Referential Integrity, *658*
DSS
Decision Support System, *561*
DSSSL
Document Style Semantics and Specification Language, *423*
DTC
Distributed Transaction Coordinator, *559*
DTD
Document Type Definition, *422*
DTS
Data Transformation Services, *23*; *53*; *865*

E

EPROCESS
Executive PROCESS, *73*
ETHREAD
Executive THREAD, *74*

F

FTS
Full-Text Search, *52*; *674*

G

GAC
Global Assembly Cache, *637*
GDI
Graphics Device Interface, *61*
Geiger K., *26*
GUI
Graphical User Interface, *59*; *157*
GUID
Globally Unique IDentifier, *411*; *788*

H

HAL
Hardware Abstraction Layer, *108*

I

IAM
Index Allocation Map, *520*
IDE
Integrated Development Environment, *184*
IDL
Interface Definition Language, *905*
IID
Interface ID, *411*
ISAPI
Internet Server API, *745*
ISO
International Organization for Standardization, *362*

L

LI
List Item, *439*
LPE
Language Processing and Execution, *463*; *508*
LPV
Local Partitioned View, *696*

M

MemToLeave
Memory To Leave, *484*
MFC
Microsoft Foundation Classes, *91*
MMU
Memory Management Unit, *160*
MSDTC
Microsoft Distributed Transaction Coordinator, *881*
MSMQ
Microsoft Message Queuing, *948*
MTA
MultiThreaded Apartment, *415*; *621*

N

NULL-указатель, *168*

O

ODS
  Open Data Services, *461*; *1000*
ODSOLE
  Open Data Services Object Linking and Embedding, *52*; *620*
OL
  Ordered List, *439*
OLE
  Object Linking and Embedding, *405*
OSI
  Open Systems Interconnection, *362*

P

PAE
  Physical Address Extension, *179*
PDB
  Program DataBase, *67*
PDE
  Page Directory Entry, *177*
PDR
  Page Directory Register, *177*
PE
  Portable Executable, *175*
PEB
  Process Environment Block, *73*; *83*; *169*
PFN
  Page Frame Number, *178*
ProgID
  Programmatic IDentifier, *626*
PSS
  Process Status Structure, *497*
PTE
  Page Table Entry, *177*

Q

QoS
  Quality of Service, *380*
QP
  Query Processor, *463*

R

RID
  Row IDentifier, *522*
RPC
  Remote Procedure Call, *124*; *360*; *366*
RTL
  RunTime Library, *75*

S

SARG
  Search ARGument in a query, *540*
SAX
  Simple API for XML, *445*; *710*
SE
  Storage Engine, *463*
SEH
  Structured Exception Handling, *84*; *93*; *190*
SGML
  Standard Generalized Markup Language, *423*
SMB
  Server Message Block, *360*
sp_
  stored procedure, *640*
SSPI
  Security Support Provider Interface, *831*
STA
  Single-Threaded Apartment, *415*; *621*
Stoppani A., *25*

T

TB
  Translation Buffer, *179*
TDI
  Transport Driver Interface, *365*
TEB
  Thread Environment Block, *87*; *170*
TIL
  Transaction Isolation Level, *555*
TLB
  Translation Look-aside Buffer, *179*; *191*

TLS
Thread Local Storage, *84*
T-SQL
Transact-SQL, *39*

U

UMS
User Mode Scheduler, *22*; *49*; *50*; *461*; *466*
UNC
Universal Naming Convention, *366*
URI
Uniform Resource Identifier, *432*

V

VAD
Virtual Address Descriptor, *71*; *196*
VC++
Visual Studio C++, *180*

W

W3C
World Wide Web Consortium, *424*
Ward B., *28*
Web-узел
Microsoft, *66*
автора, *45*
WSDL
Web Services Description Language, *804*

X

XDR
XML-Data Reduced, *767*
XML
eXtensible Markup Language, *50*
XPath
XML Path, *438*
XSD
XML Schema Definition, *771*
XSI
XML Schema Instance, *436*
XSLT
Extensible Stylesheet Language Transformations, *436*

A

Агент, *936*
Агрегирование, *413*
Адрес
INADDR_ANY, *385*
NULL-указателя, *168*
виртуальный, *177*
петли обратной связи, *388*
электронной почты автора, *45*
Адрес загрузки
применяемый по умолчанию, *63*
Активизация
агента дистанционная, *975*
Анализ
синтаксический, *508*
Анализатор синтаксический
MSXML, *710*
XML, *710*
без проверки допустимости, *430*
с проверкой допустимости, *430*
Аннотация
sql:datatype, *797*
sql:inverse, *774*
sql:key-fields, *778*
sql:limit-field, *777*
sql:limit-value, *777*
sql:mapped, *775*
sql:relationship, *774*
Апартаментный, *415*
Аплет
Services, *684*
Атрибут
at-identity, *787*
dt:type, *797*
guid, *788*
id, *785*
match, *438*
maxOccurs, *435*
MinOccurs, *435*
nullvalue, *783*

PAGE_EXECUTE_READ, *195*
returnid, *787*
select, *438*
updg:mapping-schema, *782*
Атрибут защиты
PAGE_GUARD, *195*
PAGE_READONLY, *194*
страницы, *194*
страницы PAGE_GUARD, *193*
страницы PAGE_WRITECOPY, *195*
Атрибут защиты PAGE_EXECUTE, *194*

Б

База
приоритета, *86*
База данных
dBase, *25*
ISAM, *581*
master, *995*
msdb, *868*; *995*
Northwind, *872*
pubs, *127*; *129*; *872*
Sybase, *26*
tempdb, *584*; *586*
допускающая только чтение, *561*
однопользовательская, *560*
приложения, *836*
распределительного сервера, *975*
секционированный, *561*
экземпляра, *836*
Библиотека
Advapi32.DLL, *61*; *366*
DB-Library, *26*
DLL, *62*; *407*; *416*; *622*
GDI32.DLL, *57*; *61*
Iostream, *461*
Kernel32.DLL, *57*; *61*; *63*; *97*; *135*; *366*
mscoree.DLL, *637*
MSDN Library, *58*
MSXMLn.DLL, *717*
NDIS, *365*
Net-Library, *459*
NTDLL.DLL, *62*
ODS, *461*
ODSOLE70.DLL, *640*
OLE32.DLL, *622*
OPENDS60.DLL, *1000*
Rpcrt4.dll, *402*
SQLFTQRY.DLL, *676*
Super Socket Net-Library, *389*
UMS, *461*
User32.DLL, *57*; *61*; *63*
VBODSOLELib, *659*
Ws2_32.dll, *380*
базовых классов Microsoft, *91*
динамического связывания, *57*
классов, *62*
мультипротокольная Net-Library, *366*; *402*
объектная DTSPackage Object Library, *867*
статическая, *62*
Библиотека
DLL
подсистемы, *61*
RTL
многопотоковая, *91*
однопотоковая, *91*
этапа прогона, *62*; *75*
RPC, *402*
Бит
системной принадлежности, *676*
системный, *998*
Блок
EPROCESS, *177*
KPROCESS, *177*
PEB, *167*
TEB, *102*; *167*
tiddata, *91*
исполнительного потока, *74*
исполнительного процесса, *73*
потока ETHREAD, *87*
процесса EPROCESS, *87*
среды потока, *87*; *170*
среды процесса, *73*; *83*; *169*
Блокировка
RID, *532*
взаимоисключающая, *71*; *135*
страницы, *201*

Буфер
быстрого преобразования адреса, *179*; *191*
для сборки-разборки, *291*
преобразования, *179*
сетевого приема, *497*
сетевой передачи, *497*

В

Ввод-вывод
асинхронный, *256*; *289*
конвейерный, *82*
небуферизованный, *257*; *268*; *269*; *286*; *289*
со сборкой-разборкой, *290*
совмещенный, *257*
файловый асинхронный, *461*
файловый синхронный, *251*
Взаимоблокировка, *133*; *143*
потоков, *141*
Включение, *413*
Возобновление
работы потока, *120*
Время
реальное, *113*
Вывод
на терминал, *336*
Вызов
команды DBCC CALLFULLTEXT, *677*
расширенной процедуры, *63*
управляемого кода, *635*
Вызов процедур
асинхронный, *121*; *257*
дистанционный, *124*; *360*; *366*
Выполнение
расширенных процедур, *478*
Выражение
индексируемое, *541*
регулярное, *632*
свертываемое, *543*
Вытеснение, *105*; *112*
Вычисление
объема физической памяти, *494*

Г

Генератор, *825*; *827*
запросов графический, *867*
Гистограмма
статистическая, *516*

Д

Дамп
стеков потока, *102*
Демаршалинг, *401*; *413*; *457*
Депланирование, *59*
Дерево
реляционных операторов, *508*
Дескриптор, *424*
<LI>, *439*
<OL>, *439*
<TH>, *438*
абзаца <P/>, *439*
виртуального адреса, *71*; *196*
закрывающий, *428*
объекта отображения, *341*
окна, *88*
открывающий, *428*
потока, *85*
процесса, *85*
пустой, *428*
сокета, *389*
файла, *232*; *254*; *389*
экземпляра, *71*
Деструктор
~CBufSearch, *336*
Дефрагментация
индекса, *527*
Диапазон
идентификаторов, *986*
Директива, *725*
#import, *632*
cdata, *729*
element, *726*
hide, *728*
id, *730*
idref, *730*
idrefs, *730*
xml, *726*

Диск
жесткий, *60*; *289*
Диспетчер, *106*
виртуальной памяти, *68*; *160*
динамической области памяти, *216*
окон, *61*
памяти, *497*
системного кэша, *341*
BPool, *462*
Connection, *497*
General, *498*
MemToLeave, *462*
Optimizer, *498*
OS, *499*
Query Plan, *497*
Reserved, *499*
Utility, *498*
для работы в аварийных условиях, *499*
Диспетчер транзакций
Component Services, *414*
Transaction Server, *414*
Дистрибутив
программного обеспечения Беркли, *365*
Документ
DOM, *445*; *710*
DTD, *424*
XML, *428*; *716*
XML Schema, *433*
допустимый, *429*
формально правильный, *429*
Документация
комплекта Platform SDK, *95*
комплекта SDK, *250*
оперативная Books Online, *23*; *34*; *560*; *936*
Доставка
журнала, *955*
многоадресатная, *835*
Доступ
неразрывный, *133*; *134*
Драйвер
Win32k.sys, *61*
минипорта NDIS, *365*
редиректора файловой системы Windows, *367*
сетевого адаптера, *362*
устройства, *112*
устройства привилегированного режима, *61*

Е

Единица
измерения квантов времени, *108*

Ж

Журнал
опережающей записи, *556*
программы Perfmon, *66*
транзакций, *495*; *556*

З

Завершение
процесса, *75*
нормальное, *76*
транзакции, *561*
Завершение работы
потока, *90*
процесса, *87*
Зависимость
между объектами, *658*
Заголовок
исполняемого файла, *175*; *197*
страницы, *495*
Заготовка
запроса, *875*
Загрузка
библиотеки SQL Server Net-Libraries, *63*
массовая, *793*
Загрузчик
операционной системы, *74*
Задание
SQL Server Agent, *937*
приоритета потока, *111*
синхронизационное, *975*
Задача, *869*
Bulk Insert, *872*
Data Driven Query, *874*
DDQ, *830*

Distribution clean up, *971*
DTS, *867*
Read File, *877*
Закладка, *521*; *522*; *523*
Закрепление, *193*
виртуальной памяти, *198*
Закрытие
курсора автоматическое, *608*
Запись
каталога страниц, *177*
таблицы страниц, *177*
уточняющая через точку, *629*
Запрос
URL, *750*
XPath, *748*
ввода-вывода UMS, *475*
динамический, *590*; *591*
клиентский, *462*
контролируемый по таймеру, *476*
на выполнение работы, *471*
поисковый, *877*
с шаблоном, *759*
управляемый данными, *830*
Защита
памяти процесса, *165*
ресурсов, *144*
Знак
тильды, *687*
Значение
Apartment, *416*
Both, *416*
Free, *416*
INFINITE, *468*
INVALID_HANDLE_VALUE, *232*
STATUS_PENDING, *475*
WAIT_ABANDONED, *149*
WAIT_TIMEOUT, *145*

И

Идентификатор
GUID, *788*
IID, *411*
ProgID, *626*
глобально уникальный, *411*; *788*
интерфейса, *411*
класса, *626*
клиента, *72*; *86*
потока, *86*; *148*
программный, *626*
программный ProgID, *414*
процесса, *72*
ресурса универсальный, *432*
строки, *522*
Иерархия
вложения, *778*
Избирательность, *508*; *533*; *539*
Извещение, *827*
DLL_THREAD_DETACH, *90*
неотформатированное, *827*
резюмированное, *835*; *857*
Изолированность, *555*
Импорт
статический, *62*
функций, *900*
функций статический, *63*
Имя
виртуальное SOAP, *802*
системное CONOUT$, *392*
точки сохранения, *573*
Имя хоста
localhost, *388*
Индекс
многостолбцовый, *535*
одностолбцовый, *535*
покрывающий, *522*
составной, *535*
уникальный, *510*
Инициализатор
COM, *622*
Инициализация
COM-объекта, *96*
критической секции, *139*
стека потока, *89*
транзакции, *561*
Интервал
тактовый, *105*
Интерпретатор
командный Cmd.exe, *82*
Интерфейс, *410*; *411*
ADO, *716*

CustomTask, *867*; *895*
CustomTaskUI, *895*
IContentFormatter, *834*
IDeliveryProtocol, *835*
IDispatch, *414*; *621*; *626*
IEventProvider, *826*
IHttpProtocolProvider, *835*
IMalloc, *486*; *500*
IScheduledEventProvider, *826*
IUnknown, *411*
IVBSAXContentHandler, *450*
IVBSAXDeclHandler, *450*
IVBSAXDTDHandler, *450*
IVBSAXErrorHandler, *450*
IVBSAXLexicalHandler, *450*
ODBC, *26*; *937*; *975*
OLE DB, *463*; *865*
SAX, *446*
SQL-DMO, *641*; *647*
SQLXMLBulkLoad, *790*
Windows Sockets, *364*
Winsock, *391*
графический, *66*
графических устройств, *61*
командной строки, *66*
петлевой, *388*
пользовательский графический, *59*; *157*
прикладного программирования, *49*
провайдера поддержки защиты, *831*
уровня вызовов, *360*; *457*
Информация
отладочная, *67*
регистрационная, *647*
семантическая, *426*
Инфраструктура
.NET, *410*
.NET Framework, *635*
Использование
критических секций совместное, *141*
спин-блокировки и критической секции, *140*
Источник данных
ODBC, *868*
OLE DB, *868*
Исчерпание
ресурсов потока, *106*

К

Канал
входящий, *378*
доставки, *823*
именованный, *360*; *366*
Кардинальность, *507*; *533*
Каталог
Binn, *459*
виртуальный, *802*
инсталляционный SQL Server, *459*
снимка, *940*; *942*
страниц процесса, *177*
Квант
времени, *105*; *106*; *108*; *130*
Класс
CBufSearch, *329*
CIoBuf, *320*
ContentHandler, *450*
COpenXMLRange, *736*
CRangeSearch, *350*
ErrorHandler, *450*
SqlXmlAdapter, *799*
SqlXmlCommand, *799*
SqlXmlParameter, *799*
управляемого кода, *798*
управляемый, *799*
управляемый SQLXML, *801*
Кластер, *256*; *289*
Клиент, *364*
TDI, *365*
Winsock, *361*
сокетов, *386*
Ключ
Apartment, *622*
Assembly, *637*
Both, *622*
Free, *622*
HKLM\Software\Microsoft\Notification Services\Instances, *821*
VersionIndependentProgID, *632*
дублирующийся, *791*
индекса, *521*

первичный, *699*
реестра, *416*; *622*
Кнопка, *88*
Код
CRC, *967*
SEH, *94*
ассемблера, *182*
завершения потока STILL_ACTIVE, *91*
исходный триггера, *593*
непривилегированного режима, *61*
поддержки массивов, *663*
потокобезопасный, *134*
циклический избыточный, *967*
четырехпозиционный шестнадцатеричный UCS-2, *781*
Код ошибки
E_NOINTERFACE, *412*
STATUS_GUARD_PAGE_VIOLATION, *193*
Win32, *95*
Кодировка
ANSI, *92*
Unicode, *92*
Количество
процессоров в системе, *276*
Коллекция
COM-объектов SQL-DMO, *417*
Databases, *629*
инструментальных средств, *867*
Кольцо, *58*
Команда
!handle, *243*
!heap, *225*
!peb, *169*
!teb, *170*
BEGIN DISTRIBUTED TRANSACTION, *559*
BEGIN TRAN, *557*; *558*; *568*
BULK INSERT, *417*; *560*; *641*; *789*; *872*
CLOSE, *606*
COMMIT, *569*
COMMIT TRAN, *557*; *558*; *568*
CREATE INDEX, *999*
CREATE STATISTICS, *537*
DBCC CALLFULLTEXT, *676*
DBCC FREEPROCCACHE, *517*
DBCC INDEXDEFRAG, *527*
DBCC LOG, *576*
DBCC OPENTRAN, *575*
DBCC RESOURCE, *956*
DBCC SQLPERFumsstats, *479*; *480*
DBCC TRACEON, *995*
DEALLOCATE, *615*
DECLARE CURSOR, *597*
lm, *717*
NET HELPMSG nnnn, *95*
NET START, *849*
NET START mssearch, *684*
OPEN, *601*
POST, *748*; *779*
READTEXT, *808*
ROLLBACK, *569*; *610*
ROLLBACK TRAN, *558*; *568*; *572*
s, *226*
SAVE TRAN, *572*
SELECT, *602*
SET CURSOR_CLOSE_ON_COMMIT, *609*
SET IDENTITY_INSERT, *590*; *792*
SET IMPLICIT_TRANSACTIONS, *558*; *561*
SET TRANSACTION ISOLATION LEVEL, *555*; *563*
SET XACT_ABORT, *562*
TYPE, *647*
UPDATE STATISTICS, *537*
UPDATETEXT, *962*
Validate, *433*
WAITFOR DELAY, *124*; *129*
вложенная BEGIN TRAN, *568*
недопустимая в транзакции, *575*
отладчика, *66*
Компакт-диск
Windows Support Tools, *65*
Компания
Compaq, *58*
IBM, *443*
Microsoft, *424*
Quarterdeck, *172*
Silicon Graphics, *58*

Sun Microsystems, *424*
Компилятор
Visual C++, *197*
Комплект
MSXML SDK, *452*
Platform SDK, *65*
Компонент
SQLXMLBulkLoad, *789*
UMS, *105*; *156*; *461*
Windows Service, *821*
XML Bulk Load, *789*
внепроцессный, *867*
внутрипроцессный, *867*
подсистемы создания среды, *61*
пользовательского интерфейса, *88*
программы SQL Server, *462*
сетевой Windows, *364*
Компоновка
виртуального адреса, *177*; *178*
Компьютер
многопроцессорный, *180*
однопроцессорный, *179*
Коннолли Д., *424*
Консорциум
World Wide Web, *424*
Константа
CRITSEC, *152*; *153*
INADDR_ANY, *385*
INFINITE, *121*; *144*
IO_STREAMS_PER_PROCESSOR, *320*
MUTEX, *152*; *153*
THREAD_PRIORITY_ABOVE_NORMAL, *111*
THREAD_PRIORITY_NORMAL, *111*
THREADSAFE, *152*; *153*
Конструкция
For Each, *711*
FOR UPDATE, *598*
FOR XML, *718*
FOR XML EXPLICIT, *724*
FOR XML NESTED, *733*
FORMSOF, *689*
GROUP BY, *550*; *593*
INTO, *604*
LIKE, *635*
ORDER BY, *551*
PATINDEX, *635*
SEH, *84*
WHERE CURRENT OF, *598*; *612*
Конструкция синтаксическая
CALL, *960*
MCALL, *960*
XCALL, *960*
файла CMD, *837*
Контейнер
MTA, *415*; *622*
STA, *415*; *622*
апартаментный, *415*; *621*
главный STA, *417*; *623*
Контекст
базы данных, *646*
защиты, *86*
защиты процесса, *72*
потока, *87*
Контроллер
Automation, *621*
Контроль
загрузки процессора, *77*
над созданием потоков, *77*
Конфликт, *977*
Координатор
распределенных транзакций, *559*
распределенных транзакций Microsoft, *881*
Копирование
при записи, *193*; *195*
Коэффициент
заполнения индекса, *524*
Критерий
выборки динамически определяемый, *983*
Курсор, *580*
FORWARD_ONLY, *584*
асинхронный, *607*
глобальный, *600*
динамический, *586*
допускающий только чтение, *597*
ключевого набора, *588*
локальный, *600*
обновляемый, *597*
прокручиваемый, *586*; *602*

с перемещением только вперед, *584*
статический, *587*; *597*
типа INSENSITIVE, *601*
типа KEYSET, *601*
типа STATIC, *601*
Кэш
процессора вторичный, *116*
процессора локальный, *495*
сборок глобальный, *637*
статей, *956*

Л

Лексема
доступа, *72*; *86*

М

Макрокоманда
HasOverlappedIoCompleted, *475*
Мандат
защиты клиента, *367*
клиента именованного канала, *367*
Маркер
параметра, *518*
Маршалинг, *401*; *412*; *457*
Маска
простаивающих процессоров, *118*
родственности процессора, *116*
файла, *269*
Массив
BUF, *494*
двумерный с зубчатыми границами, *666*
структур BUF, *492*
Масштабирование
приложения, *88*
Машина
базы данных, *26*
сценариев ActiveX, *632*
хранения, *463*
Метаязык, *422*
Метка-заполнитель, *519*; *783*; *788*
at-identity, *787*
Метод
AddRef, *412*
BNSSubscribe, *857*
CreateDataSource, *856*
Execute, *790*
ExportData, *645*
GetIDsOfNames, *414*
Idispatch::Invoke, *627*
Invoke, *414*
loadXML, *711*
MoveToContent, *801*
QueryInterface, *412*; *626*
RecalcEnd, *358*
Release, *412*
Save, *716*
ScriptTransfer, *647*
Search, *325*
selectNodes, *711*
UpdateGrid, *856*
Механизм
доступа к ресурсу, *149*
обработки сигналов многопотоковый, *149*
передачи сигналов, *337*
создания потока, *92*
Микропоток, *84*; *95*
рабочий, *470*
UMS, *470*
Многозадачность
кооперативная, *107*
Модель
MTA, *415*; *621*
PAE, *179*
STA, *415*; *621*
апартаментная, *415*
многопотоковая, *415*
однопотоковая, *415*
апартаментно-потоковая, *415*; *621*
документа объектная, *422*; *444*; *710*
компонентных объектов, *50*
многопотокового апартаментного контейнера, *621*
многопотоковой обработки, *621*
объектная DTS, *641*; *865*
однопотокового апартаментного контейнера, *621*
потоковой организации работы, *415*

преобразования адресов памяти, *179*
публикации, *953*
свободно-потоковая, *415*; *621*
эталонная OSI, *363*
эталонная взаимодействия открытых систем, *362*
Модель восстановления, *560*
Bulk-Logged, *560*
Full, *560*
Simple, *560*
Модуль
MMU, *177*
оперативной памяти, *159*
управления памятью, *160*
Монитор
производительности, *66*
Мьютекс, *71*; *135*; *137*; *147*; *148*
несигнальный, *148*
сигнальный, *148*

H

Набор
временных регистров, *86*
входной, *865*
выходной, *865*
данных, *580*
программируемых COM-объектов, *867*
результирующий, *580*; *647*
синхронизационный, *942*
Назначение
библиотек DLL подсистемы, *61*
процесса, *71*
Накат, *556*
Накопление
промежуточных результатов, *551*
Настройка
4 Гбайт, *174*
виртуального каталога, *746*
памяти приложения, *161*; *174*
Неразрывность
выполнения запросов, *472*
Номер
фрейма страницы, *178*
поколения, *979*
текущий, *980*

O

Обеспечение
программное сетевое, *362*
Область
BPool, *484*; *487*
MemToLeave, *484*; *487*
PEB, *169*
TEB, *169*
TLS, *86*
памяти динамическая, *192*; *216*; *522*
закрытая, *216*
пользовательская, *216*; *219*
применяемая по умолчанию, *216*; *217*
применения курсоров, *590*
Обмен
данными динамический, *405*
страничный, *173*
Обновление
в порядке очереди, *948*; *966*
инкрементное, *953*
немедленное, *946*
позиционированное, *612*
Обозначение
#REQUIRED, *431*
Обработка
исключительных ситуаций структурированная, *190*
снимка параллельная, *961*; *963*
Обработчик
системных прерываний, *239*
Обработчик событий, *71*; *711*; *870*
IVBSAXContentHandler, *450*
ODS, *625*
Образ данных
конечный, *779*
начальный, *779*
Объединение
серверов, *696*; *707*
Объект
Application, *867*
Automation, *646*
BulkCopy, *645*
CBufSearch, *276*
Command, *719*
Connection, *868*

CRangeSearch, *350*
DataSet, *856*
DOMDocument, *446*; *711*
DTSDestination, *877*
DTSSource, *877*
Event, *826*
EventCollector, *826*
EventLoader, *826*
NSApplication, *855*
NSInstance, *855*
Recordset, *716*
RegExp, *632*
SAXXMLReader, *450*
SQLServer, *645*
SQLXMLBulkLoad, *795*; *798*
SqlXmlCommand, *801*
SqlXmlParameter, *801*
SubscriptionEnumeration, *856*
Transfer, *647*; *658*
Word.Application, *632*
XmlReader, *801*
XmlTextWriter, *801*
включаемый, *413*
включающий, *413*
документа, *711*
мьютекса привилегированного режима, *153*
отображения файла, *231*; *232*; *340*
пользовательский, *97*
распределения памяти, *486*
репликации ActiveX, *417*
секции памяти, *230*; *231*; *340*
семафора, *147*
синхронизационный, *135*
системный, *998*
управления распределенными сетями, *629*
файла, *248*; *250*
Объект привилегированного режима, *71*; *73*; *148*; *250*
потока, *89*
синхронизационный, *134*
Объект события, *145*; *468*
именованный, *231*
Объект-посредник, *416*; *622*
Объем
адресуемой памяти, *159*
пространства адресов, *60*
страницы, *71*
Объявление
типа документа, *432*
Ограничения
CHECK, *697*; *986*
PRIMARY KEY, *590*
UNIQUE KEY, *590*
внешнего ключа, *658*
первичного ключа, *658*
предшествования, *868*
ссылочной целостности декларативные, *658*
уникального ключа, *658*
Окно, *172*
Disassembly, *101*; *124*
скрытое, *416*; *622*
терминальное, *61*
диалоговое
General Protection Fault, *76*
Performance Options, *112*
Publication Properties, *939*
Оператор
#define, *95*
AND, *542*
CASE, *593*
CLOSE, *606*
COMMIT TRAN, *561*
CONVERT, *519*
DEALLOCATE, *606*
DELETE, *612*
Dim, *414*; *624*
FETCH, *602*; *604*
FETCH RELATIVE 0, *605*
new, *196*; *217*
PRINT, *576*
ROLLBACK TRAN, *561*
SELECT, *576*; *828*
SELECT DISTINCT, *592*
switch, *397*
typedef, *424*
UNION ALL, *858*
UPDATE, *598*; *612*

WAITFOR, *472*
определения типа, *424*
Операция
DML, *523*
SEH, *84*; *93*
асинхронного чтения, *312*
ввода-вывода
асинхронная, *267*; *457*
синхронная, *251*
дефрагментации, *524*
копирования данных, *869*
неразрывная, *471*
ориентированная на обработку строк, *591*; *593*
освобождения мьютекса, *148*
отображения файла, *231*
переключения контекста, *96*; *466*
распределения буфера, *253*
распределения памяти, *167*
регистрируемая в журнале в минимальном объеме, *560*
резервирования памяти, *168*
структурированной обработки исключительных ситуаций, *84*; *93*
Определение
DTD, *430*
связанных серверов, *705*
типа документа, *422*
точки сохранения, *572*
Оптимизатор
запросов, *698*
Оптимизация
запроса, *509*
кода транзакций, *577*
плана тривиальная, *509*
производительности курсоров, *616*
процесса синхронизации, *982*
Опция
/3GB, *166*; *169*; *174*; *176*; *189*
/USERVA, *174*
Allow POST, *748*
Allow XPath, *748*
CLOSE_CURSOR_ON_COMMIT, *610*
ELEMENTS, *721*
Expose runtime errors as HTTP error, *749*
NEXT, *605*
Run on the client, *749*
WITH RECOMPILE, *516*
Организация
Open Software Foundation, *400*
The Open Group, *400*
по стандартизации международная, *362*
процесса внутренняя, *73*
Организация работы
многопотоковая, *89*
ОС
операционная система, *57*
Освобождение, *200*
Откат, *556*
из вложенной транзакции, *571*
Отладка, *35*
библиотеки DLL, *912*
определяемой пользователем задачи DTS, *912*
приложений, *66*
транзакций, *575*
Отладчик, *35*
cdb.exe, *66*
VC++, *342*
Visual C++, *277*
WinDbg, *35*; *65*; *68*; *101*; *110*; *124*; *169*; *225*; *458*
общего назначения, *66*
привилегированного режима, *135*
символический, *35*
Отмена
закрепления, *200*
Отображение
встроенное, *781*
применяемое по умолчанию, *781*
файла на память, *64*
файлов образов, *233*
Оценка
избирательности, *540*
Очистка
кэша процедуры, *517*
Ошибка
отсутствия страницы, *176*; *190*
последняя, *94*

П

Пакет, *362*; *867*
  Debugging Tools for Windows, *66*
  DTS, *417*; *866*
  LotusXSL, *443*
  SQLXML, *417*; *708*
  SuperRecorder, *239*
  WinMac32, *240*
  WinMacro, *239*
  команд, *556*
  обновлений, *780*
  сервисный, *290*
Пакет завершения, *308*; *457*
  ввода-вывода, *149*; *150*
Память
  AWE, *173*; *484*
  TLS, *625*
  виртуальная, *59*; *60*; *64*; *160*; *192*
  дополнительная, *172*
  закрепленная, *193*
  зарезервированная, *192*; *484*
  компьютера физическая оперативная, *60*
  оперативная, *60*
  первичная, *60*
  потока локальная, *84*
  расширенная, *172*
  реальная, *64*; *160*; *163*; *193*; *231*
  с копированием при записи, *64*
  совместно используемая, *192*; *229*; *230*; *340*
  физическая, *60*
Параметр
  /3GB, *459*
  /3GB файла BOOT.INI, *70*; *160*
  /HEAP, *217*
  /LARGEADDRESSAWARE, *175*
  /PAE, *180*
  /STACK, *197*
  /SWAPRUN:CD, *233*
  /SWAPRUN:NET, *233*
  /USERVA, *174*; *459*
  ANSI_DEFAULTS, *558*
  bAlertable, *266*
  bWait, *266*
  contenttype, *761*
  DueDate, *146*
  dwClsContext, *626*
  FILE_END, *253*
  FILE_FLAG_NO_BUFFERING, *268*; *269*; *291*
  FILE_FLAG_OVERLAPPED, *258*; *267*; *269*; *291*; *461*
  FileHandle, *149*
  flags, *740*
  HEAP_GENERATE_EXCEPTIONS, *218*; *220*
  HEAP_NO_SERIALIZE, *218*; *220*
  HEAP_REALLOC_IN_PLACE_ONLY, *219*
  HEAP_ZERO_MEMORY, *218*; *253*
  hostname:port, *388*
  Local Path, *747*
  max server memory, *485*; *504*
  max worker threads, *486*; *502*
  MEM_COMMIT, *198*
  MEM_RESET, *199*
  QUOTED_IDENTIFIER, *692*
  SARG, *540*
  sp_configure, *78*
  XACT_ABORT, *562*
  выходной, *613*
  именованный, *629*
  поиска в запросе, *540*
  процедуры xp_exception, *99*
  со значением NULL, *784*
  файла BOOT.INI, *83*
  функции SetErrorMode, *76*
Параметризация
  автоматическая, *518*; *519*
  пакетов DTS, *889*
Перебазирование, *63*
Перегрузка
  оператора delete, *221*
  оператора new, *221*
Передача
  многоадресатного сообщения, *380*
  файла из нулевой копии, *391*
Переключение
  контекста, *59*; *84*; *105*; *114*
  контекста потоков, *466*

Перекомпиляция
автоматическая, *516*
Переменная
@@ERROR, *562*
автоматически обновляемая @@CURSOR_ROWS, *601*
автоматически сопровождаемая @@TRANCOUNT, *568*; *576*
курсора, *613*
среды TEMP, *795*
типа Variant, *414*; *624*
управляющая @@FETCH_STATUS, *602*
Перенаправление, *82*
Пересечение
индексов, *523*
Перехват
параметров, *515*
Переход
в режим вытеснения, *477*
в состояние ожидания, *120*
рабочего микропотока в режим вытеснения, *478*
Петля обратной связи, *641*
План
запроса, *700*
тривиальный, *509*
Планирование, *59*; *463*
очередей, *114*
потоков, *107*
Планировщик, *469*
UMS, *469*; *482*
Windows, *105*; *482*
вытесняющий, *130*; *467*
кооперативный, *467*
непривилегированного режима, *22*; *49*; *50*; *308*; *461*; *466*
потоков Windows, *70*
скрытый, *479*; *480*
ядра, *106*
Плотность, *508*; *533*
индекса, *508*
просмотра, *524*
Подготовка
к полнотекстовому поиску, *680*
столбца для полнотекстового поиска, *681*
Подкаталог
bin, *459*
Подкачка, *164*
Подписка, *820*; *836*
Подписчик, *836*; *937*
Подсистема
OS/2, *61*
POSIX, *61*
Win32, *61*
создания среды, *61*
Подсчет
ссылок, *412*
Поиск
форм слова, *689*
Поле
Characteristics, *175*
текстовое, *88*
Пользователь
логический, *471*
физический, *471*
Порт
завершения ввода-вывода, *149*; *306*; *337*; *457*
Порядок байтов
сетевой, *385*
хост-компьютера, *385*
Поток, *59*; *70*; *72*; *84*
вызывающий, *85*
главный, *59*; *70*; *87*; *89*
обнуления страниц, *112*
отложенной записи, *495*
первичный, *87*
приложения первичный, *70*
приостановленный, *120*
рабочий, *87*; *89*; *470*
рабочий sp_OACreate, *623*
реального времени, *113*
текущий, *102*
фоновый, *87*; *89*
Потокобезопасность, *150*
Потребитель
внешний, *503*
внутренний, *503*
памяти, *502*
Появление
синего экрана, *178*

Правило
  события подписки, *828*
  согласования, *827*; *828*
  согласования динамически определяемое, *858*
  хроники событий, *828*
Правило подписки
  проверяемое по графику, *828*
Предикат, *507*; *685*
  BETWEEN, *704*
  CONTAINS, *680*; *685*
  FREETEXT, *686*; *689*
  LIKE, *686*
Представление
  INFORMATION_SCHEMA, *1000*
  горизонтально секционированное, *696*
  индексированное, *530*
  секционированное, *696*
    локальное, *696*
    распределенное, *696*; *705*
Предызвещение, *827*
Прекращение
  работы сервера аварийное, *94*
Преобразование
  ActiveX, *876*
  WriteFile, *877*
  XSLT, *438*
  адреса, *160*; *176*; *178*
  имен, *361*; *362*
  курсора неявное, *597*
Прерывание
  от таймера, *109*
Префикс
  dt:, *768*
  NS, *822*
  sp_, *640*
  sp_fulltext_, *676*
  sql:, *771*
  WSA, *381*
  xsl:, *438*
Привязка, *381*; *385*
Приложение
  BNSSubscribe, *851*
  DB-Library, *239*
  DMO, *645*
  DTS Designer, *865*
  DTSPkgGuru, *918*
  findstring, *349*; *358*
  fndstr, *342*
  fstring, *311*; *329*; *392*
  fstring_io_comp, *311*
  fstring_io_comp_out, *331*
  fstring_pipe, *368*; *392*
  fstring_pipe_socket, *392*; *399*
  Notification Services, *821*
  RPC, *400*
  socket_client, *391*
  socket_server, *386*; *391*
  socket_server_rf, *391*
  SQL Server, *581*
  SuperRecorder, *238*
  Win32, *35*
  Windows 3.x, *238*
  Winsock, *381*; *389*
  Winsock без установления логического соединения, *361*
  Winsock с установлением логического соединения, *360*
  диагностическое, *1002*
  для работы в большом пространстве адресов, *161*
  для работы с сокетами, *382*
  испытательное, *915*
  клиентское, *386*
  многопотоковое, *77*; *86*
  серверное, *386*
  терминальное, *180*
  управления подпиской, *821*; *836*; *850*
Пример
  приложения, *269*; *836*
Принадлежность
  к потоку, *148*
Принцип
  использования значения тайм-аута, *514*
  проектирования Strategy, *347*
  целевой стоимости, *514*
Приоритет
  Real-Time, *111*
  базовый, *112*
  потока, *106*; *110*; *111*

программы SQL Server, *123*
Приоритет процесса, *106*; *110*
High, *122*
Real-Time, *122*
Приостановка
работы потока, *120*
Проблема
порочного круга, *617*
синхронизации доступа, *221*
Пробуксовка, *160*; *172*
Проверка
допустимости, *430*
состояния процесса, *74*
Программа
CMD.EXE, *899*
Distribution Agent, *937*; *941*; *953*
DTC, *559*
DTS Designer, *867*
DTS Import/Export Wizard, *866*; *867*
DTS Query Designer, *867*
DTSDIAG, *1002*
Enterprise Manager, *78*; *122*; *417*; *937*
Explorer, *195*; *277*
ImageCfg.exe, *117*
Index Tuning Wizard, *552*
ISQL, *895*
Log Reader Agent, *953*
Merge Agent, *941*; *974*; *975*
Notepad, *81*; *101*
OSQL, *212*
osql.exe, *129*
Perfmon, *66*; *72*; *85*; *107*; *109*; *115*; *162*; *176*; *249*; *576*
Profiler, *999*
Pview, *65*; *80*; *107*
Pviewer, *65*; *107*
QEMM-386, *172*
Query Analyzer, *98*; *123*; *129*; *188*; *511*
Queue Reader Agent, *948*
Recorder, *238*
regasm.exe, *637*
regsvr32, *662*
Replication Monitor, *937*
Snapshot Agent, *937*; *953*; *974*; *976*
Spy++, *115*
SQL Server Agent, *124*
SQL Server Service Manager, *100*
sqlservr.exe, *26*; *38*
Sysmon, *66*
Task Manager, *77*; *111*; *112*; *162*; *176*
TList, *65*
Visual Studio, *62*; *632*
Visual Studio C++, *180*
WinDbg, *66*
WinObj, *238*
XML Spy, *451*
массового копирования, *641*
преобразования XSLT, *439*
Программа-мастер
Create Dynamic Snapshot Job, *985*
Create Publication, *893*; *938*
Create Publication Wizard, *982*
Full-Text Indexing Wizard, *679*
Transform Published Data, *893*
Программирование
аспектно-ориентированное, *43*
экстремальное, *43*
Продолжительность
хранения, *971*
Проект
AppDefinition, *836*
VC++ Makefile, *837*
Прокрутка, *580*
Пространство
адресное процесса, *70*
адресов
виртуальной памяти, *59*; *70*; *87*
непривилегированного режима, *160*
процесса, *159*; *166*
виртуальных адресов, *71*; *163*
закрытых адресов, *60*
закрытых адресов процесса, *169*
имен, *380*; *436*
имен datatypes, *768*
рабочее DTS Designer, *866*
Протокол
CIFS, *365*
HTTP, *745*
Named Pipes, *366*

NetBIOS, *360*; *366*
SMB, *360*
SMTP, *835*
SOAP, *801*
Windows Sockets, *366*
доступа к файлам Internet общий, *365*
сетевой, *362*
файловый, *835*
Процедура
APC, *146*
DumpRegionMemoryStatus, *204*
ex_raise, *735*
execute_rpc, *128*
HeapAlloc, *218*
HeapFree, *218*
sp_adjustpublisheridentityrange, *988*
sp_bindsession, *641*
sp_checkspelling, *630*
sp_configure, *606*
sp_dboption, *601*; *606*
sp_DisplayOAErrorInfo, *646*
sp_executesql, *516*
sp_exporttable, *641*
sp_fulltext_catalog, *685*
sp_fulltext_column, *685*
sp_fulltext_table, *685*
sp_generate_script, *647*
sp_getbindtoken, *641*
sp_helptext, *998*
sp_hexadecimal, *646*
sp_indexoption, *532*
sp_MS_marksystemobject, *676*; *998*
sp_OACreate, *625*
sp_OADestroy, *628*
sp_OAGetErrorInfo, *628*; *646*
sp_OAGetProperty, *625*; *627*
sp_OAMethod, *625*; *627*
sp_OASetProperty, *627*
sp_OAStop, *628*
sp_recompile, *516*
sp_repldone, *958*
sp_showstatdate, *539*
sp_trace_setevent, *1005*
sp_vbscript_reg_ex, *632*
sp_xml_concat, *808*
sp_xml_preparedocument, *96*; *417*; *712*; *734*; *808*
sp_xml_removedocument, *734*
SQLXMLOLEDB, *417*
xp_exception, *98*
обновления, *960*
обратного вызова, *71*
оконная, *416*
оптимизации запроса, *509*
расширенная, *833*; *996*
sp_replcmds, *955*
xp_cmdshell, *647*; *684*
xp_exec, *830*; *831*
xp_execresultset, *831*
xp_NSNotify, *830*
специальная, *640*; *672*; *996*
хранимая, *640*; *757*
NSInsertNotificationn, *829*; *831*
sp_add_dtspackage, *868*
sp_addpublication, *938*
sp_addpublication_snapshot, *938*
sp_article_validation, *968*
sp_browsereplcmds, *943*; *954*
sp_enumcustomresolvers, *978*
sp_MSdistribution_cleanup, *971*
sp_replicationdboption, *955*
sp_run_xml_proc, *811*
waiter, *127*
xp_printstatements, *943*
Процедура-заглушка, *400*
Процесс, *59*; *70*
Csrss подсистемы Win32, *75*
Csrss.exe, *61*
SQL Server, *82*
Win32, *71*
вызывающий, *85*
контрольной точки, *496*
подсистемы создания среды, *61*
потокобезопасный, *59*
родительский, *86*
Процессор
Alpha, *58*; *177*
DOM, *712*
Intel x86, *58*; *71*; *177*; *179*; *189*; *194*; *256*; *289*

MIPS, *58*
RISC, *76*
SAX, *711*; *712*
запросов, *23*; *463*; *552*
идеальный, *106*; *118*
команд, *899*
Псевдодескриптор, *85*
Псевдоним
sql, *771*
Псевдопользователь
system_function_schema, *639*; *997*
Публикация
данных, *745*
Пул
буферов, *198*; *484*
рабочих микропотоков, *469*
Путь
к исходному коду, *68*
к файлам отладочной информации, *67*
локальный, *747*

Р

Разблокирование
страницы, *201*
Раздел
директив, *725*
непривилегированного режима, *60*
привилегированного режима, *60*
Размер
сектора, 256; 289
страницы, 160; 192
Разработка
приложения Notification Services, *839*
Расходование
кванта времени потока, *109*
кванта времени частичное, *109*
пространства адресов бесполезное, *165*
Расширение
BPool, *485*
файла подкачки, *64*
физического адреса, *179*
Расширения
API-интерфейса сервера Internet, *745*
адресные для работы с окнами, *51*; *161*; *172*
Расширитель 32-битовый
Phar Lap, *172*
Plink, *172*
Реализация
интерфейса, *410*; *411*
Регистр
аппаратный, *59*
временный, *59*; *115*
каталога страниц, *177*
программный, *59*
процессора, *115*
управления 3, *177*
Регистрация
библиотеки DLL, *662*
экземпляра, *848*
Редактор
Sequin SQL Editor, *239*
связей, *175*; *197*
текстовый исходного кода Sequin SQL, *632*
Редиректор, *361*
Реестр
системный, *409*
Режим
EXPLICIT, *722*
PAE, *179*; *180*
READ COMMITTED, *565*
READ UNCOMMITTED, *563*
REPEATABLE READ, *566*
SERIALIZABLE, *567*
автоматических транзакций, *561*
байтовый, *367*
микропотоков, *84*; *470*; *479*
непривилегированный, *58*
полусинхронный, *955*
потоков, *470*
потоковый ADO, *732*
привилегированный, *58*
синхронный, *955*
строковый, *367*
Режим работы
процессора Intel x86, *58*
Резервирование
виртуальной памяти, *196*
Реорганизация
индекса, *527*

Репликация
данных двунаправленная, *953*
путем слияния, *974*
слиянием, *937*
снимка, *937*; *941*; *974*
транзакционная, *937*; *953*; *972*; *974*
Рефакторинг, *43*
Родственность
"жесткая", *117*
идеальная, *117*
последнего по времени процессора, *118*
процессора, *106*; *116*; *469*
Роль
bulkadmin, *792*
db_ddladmin, *792*
NSVacuum, *845*
sysadmin, *792*
базы данных db_owner, *792*

C

Самоблокировка, *641*
Сбор
мусора автоматический, *412*
Сборка, *637*
Microsoft.Data.SqlXml, *798*
Microsoft.SqlServer.NotificationServices, *851*
Свертывание
выражений, *543*
Свойства
ACID, *554*
Свойство
BulkLoad, *797*
CheckConstraints, *791*
CommandText, *801*
ConnectionCommand, *795*; *796*
ErrorLogFile, *796*
ForceTableLock, *794*
IgnoreDuplicateKeys, *791*; *792*
KeepIdentity, *792*
KeepNulls, *793*
SchemaGen, *796*
SGDropTables, *798*
SGUseID, *797*
Step priority, *880*
TempFilePath, *794*
Transaction, *794*; *796*
XMLFragment, *790*
потока данных, *879*
проекта Create Symbolic Debug Info, *67*
Связывание
позднее, *413*; *624*
раннее, *413*; *624*
Связь
без установления логических соединений, *380*
надежная, *380*
ненадежная, *380*
потоковая, *380*
с установлением логических соединений, *380*
Сектор, *256*; *289*
Секционирование
списка свободных страниц, *497*
Секция, *497*; *698*
CDATA, *729*
вертикальная, *981*
горизонтальная, *981*
граничная, *170*
критическая, *138*; *139*; *148*; *275*
непривилегированного режима, *166*
привилегированного режима, *169*
присваивания NULL-указателя, *161*
пространства закрытых адресов процесса, *169*
Семафор, *71*; *137*; *147*
Семейство
Windows NT, *60*
Windows NT Server, *459*
операционных систем Windows NT, *195*; *675*
продуктов Windows NT, *108*
протоколов AF_INET, *385*
Сервер
IIS, *410*
SMTP, *834*; *839*
Winsock, *361*
внепроцессный, *626*
внутрипроцессный, *626*

именованного канала, *367*
публикаций, *936*; *975*
публикаций репликационный, *892*
распределительный, *825*; *833*; *936*; *975*
сокетов, *381*; *382*
текста, *675*
файлов отладочной информации, *68*
файлов отладочной информации компании Microsoft, *67*
Серверы
связанные, *706*
Сериализация
доступа, *416*
Сжатие файлов
LDF, *461*
MDF, *461*
базы данных, *462*
Символ
амперсанда, *429*
подстановочный, *688*
пустого дескриптора, *428*
Синтаксис
DTD, *433*
XML, *433*
Синхронизация, *133*; *975*
доступа к терминалу, *275*
доступа к файлу, *251*
непривилегированного режима, *135*
потоков, *134*; *138*; *144*; *151*
Система
LAN Manager, *366*
кэширования, *341*
поддержки принятия решений, *561*
разрешения конфликтов, *977*
синхронизации, *149*
файловая, *247*; *868*
операционная
DOS, 172
OpenVMS, 195
OS/2, 26; 366
UNIX, 195; 365
Windows, 114
Windows 9x, 195
Windows 3.x, 467
файловая
FAT16, *247*
FAT32, *247*
NTFS, *247*
Ситуация
исчерпания ресурсов, *112*
отсутствия страницы, *160*
Ситуация исключительная
INVALID_HANDLE, *74*
STATUS_ACCESS_VIOLATION, *218*
STATUS_GUARD_PAGE, *193*
STATUS_NO_MEMORY, *218*
Win32, *100*
необработанная, *94*
языковая, *93*; *100*
Слово ключевое
BINARY BASE64, *719*
catch, *93*
DISTINCT, *550*
ELEMENTS, *719*
FROM, *605*
GLOBAL, *600*
LOCAL, *600*
throw, *93*
UNION, *550*
XMLDATA, *719*
Служба
DTS, *865*
Microsoft Search, *675*
MSDTC, *881*
Notification Services, *23*; *820*
Web, *801*
администрирования, *676*
индексации Microsoft Indexing Service, *678*
каталогов, *380*
полнотекстовых запросов, *676*
преобразования данных, *23*; *865*
рассылки извещений, *23*; *820*
рассылки извещений о программных ошибках, *839*
сетевая, *362*
Служба данных
открытая, *461*
Смещение, *253*
Снимок, *936*
данных начальный, *975*
динамически сопровождаемый, *984*

начальный, *941*; *984*
параллельно обрабатываемый, *962*
транзакционный, *961*
Событие, *145*; *414*; *711*; *870*
сигнальное, *145*
сигнальное с автоматическим отключением, *145*
языковое, *736*
языковое INSERT BULK, *789*
Согласование, *362*
Соглашение
QoS, *380*
о качестве и классе предоставляемых услуг передачи данных, *380*
об именовании универсальное, *366*
Соединение, *508*; *544*; *868*
внутреннее, *508*
дейтаграммное, *361*; *380*
клиентское, *457*
потоковое, *360*
Создание
области совместно используемой памяти, *232*
объекта процесса, *73*
объекта семафора, *147*
потока, *77*; *89*
публикации снимка, *938*
пустой таблицы, *797*
события, *146*
сокета, *385*
экземпляра COM-объекта, *410*
Сокет, *360*; *389*
Windows, *366*
Winsock, *361*
Сообщение, *827*
ModLoad, *717*
WM_TIMER, *147*
предупреждающее, *257*
Состояние
защиты страницы, *193*
несигнальное, *133*; *143*
ожидания, *110*; *121*
потока, *106*; *109*
приостановленное, *320*
сигнальное, *133*; *143*
Спецификация
RPC, *400*
X/Open XA, *559*
XDR, *767*
XML Schema, *433*
XML-Data, *767*
xml-stylesheet, *764*
Спин-блокировка, *134*; *136*; *137*
привилегированного режима, *154*
Список
динамических областей памяти, *225*
загруженных модулей и процессов, *80*
запросов ввода-вывода, *474*
контролируемый по таймеру, *476*
контроля доступа, *165*
ожидающих микропотоков, *474*
планировщика UMS, *470*
работоспособных микропотоков, *472*
рабочих микропотоков UMS, *472*
свободных страниц, *504*
Сравнение
процессов и потоков, *88*
Среда
разработки интегрированная, *184*
распределенных вычислений, *400*
Средства
ADO, *417*
Automation, *620*
AWE, *176*; *189*
COM Automation, *865*
DCOM, *624*
DOM, *710*
DTS, *417*
LPE, *508*
MSMQ, *948*
MSXML, *446*
ODSOLE, *620*; *635*
OLE DB, *417*
SAX, *711*
SQLMAIL, *96*
SQLXML, *96*
асинхронного ввода-вывода, *248*
ведения очередей сообщений Microsoft, *948*
восстановления базы данных, *556*
инструментальные Platform SDK, *58*

недокументированные, *992*
обмена сообщениями Windows, *416*
опережающей записи в журнал, *556*
отложенной записи, *493*; *557*
поддержки массивов, *671*
полнотекстового поиска, *674*; *693*
распределенной обработки транзакций, *559*
связывания и внедрения объектов открытых служб данных, *620*
сетевого транспорта, *364*
синхронного ввода-вывода, *248*; *251*
управления транзакциями, *554*
языковой обработки и выполнения, *463*; *508*
Средства автоматизации
VBScript инструментальные, *418*
Средства доступа
OLE DB, *417*
SQLXMLOLEDB, *417*; *732*
Средства сжатия
NTFS, *260*; *291*
Средства синхронизации, *133*
непривилегированного режима, *156*
привилегированного режима, *156*
Средство
многофазной перекачки данных, *866*; *870*
проектирования с графическим интерфейсом, *867*
разрешения конфликтов, *977*
чтения журнала, *956*
Средство доступа
OLE DB, *865*; *866*
SQLOLEDB, *868*
к данным о событиях, *825*
базирующееся, *826*
небазирующееся, *826*
полнотекстовое, *676*
с управляемым кодом, *826*
Средство инструментальное
Depends, *62*; *402*
DumpBin, *402*
IIS Virtual Directory Management, *802*
regasm.exe, *637*
SAXON, *452*
Ссылка
на NULL-указатель, *168*; *180*
на NULL-указатель скрытая, *181*
на определение DTD, *432*
Стандарт
ANSI, *558*
XML Schema, *767*
распределенных вычислений, *400*
сетевого программирования RPC, *400*
Статья, *936*
Стек
для выполнения кода, *86*
сетевых протоколов, *361*; *362*
Степень
детализации распределения, *160*; *164*; *168*; *192*
непротиворечивости, *555*
распараллеливания, *308*; *555*
Стоимость
узла AND, *513*
узла OR, *513*
целевая, *514*
Столбец
identity, *786*
msrepl_tran_version, *946*; *964*
идентификации identity, *590*
секционирования, *697*; *699*
Страница, *71*
IAM, *520*
закрепленная, *192*; *200*
закрытая, *200*
зарезервированная, *192*
защищенная, *193*; *195*
кодовая, *264*
незафиксированная, *495*; *557*
обнуляемая по требованию, *199*
свободная, *192*
совместно используемых данных непривилегированного режима, *170*
узлов, *521*
Стратегия
соединения, *544*
Строки
дублирующиеся, *507*

Структура
BUF, *492*; *494*
CONTEXT, *87*; *114*
DISPPARAMS, *627*
MEMORYSTATUS, *712*
OUTPUT_OVERLAPPED, *335*
OVERLAPPED, *254*; *258*; *461*
PROCESS_INFORMATION, *902*
PSS, *497*
sockaddr_in, *388*
soServerAddress, *385*
SRV_PROC, *497*; *957*
STARTUPINFO, *902*
TLS, *84*
W32THREAD, *87*
данных разреженная, *198*
критической секции, *139*
отображения, *114*
отображения битовая, *520*
с итоговыми данными о готовых к выполнению потоках, *114*
статуса процесса, *497*
Суффикс
Notify, *829*
NSMain, *822*
Сущность
символьная, *429*
Схема
IAM, *520*
XML, *424*; *433*
аннотированная, *748*; *770*
распределения индексов, *520*
Схема отображения, *767*; *789*
XDR, *767*
XSD, *771*
Сценарий
ActiveX, *875*
инсталляционный, *1000*
Счетчик, *66*
dwTotalBytesRead, *266*
использования, *73*
приостановок, *120*
программы Perfmon, *66*; *72*; *85*; *162*; *249*

T

Таблица
MSmerge_contents, *976*
MSmerge_genhistory, *976*
MSmerge_info, *976*
MSmerge_replinfo, *977*
MSmerge_tombstone, *976*
MSrepl_commands, *943*; *954*
MSrepl_transactions, *944*
MSreplication_queue, *948*
MSsnapshot_history, *941*; *944*
sysarticles, *956*
syscacheobjects, *519*
syscolumns, *956*
syscomments, *595*; *994*
sysdepends, *658*
sysdtspackages, *868*
sysindexes, *520*
sysmergearticles, *976*; *981*
sysobjects, *595*; *996*
дескрипторов процесса, *87*
импорта, *62*; *1000*
каталога страниц, *177*
конфликтов, *977*
краевая, *741*
опорная, *591*
открытых дескрипторов системных ресурсов, *71*
перекрестная, *591*
подписки, *975*
связывания, *874*
системная syscacheobjects, *498*
стилей, *761*
XML, *833*
XSLT, *437*
каскадная, *425*; *436*
страниц процесса, *169*
указателей каталога страниц, *179*
универсальная, *722*
хронологическая, *827*
экспорта библиотеки DLL, *62*
Таймер
непривилегированного режима, *146*
ожидания, *146*

ожидания привилегированного режима, *146*
Термин
SARG, *540*
Терминал
системы, *392*
Технология
ActiveX, *406*
COM, *406*; *409*; *463*
COM Interop, *635*; *827*
DCOM, *400*; *949*
DDE, *405*
DHTML, *423*
DOM, *444*
Intellisense, *632*
OLE, *405*
XML Schema, *433*
Тип
кластеризованный, *520*
курсора, *584*
некластеризованный, *520*
родственности процессора, *117*
Точка
контрольная, *496*
останова, *180*
сохранения, *572*
Точка входа
language_exec, *714*
main, *92*
WinMain, *92*
Wmain, *92*
wWinMain, *92*
Транзакции
сериализуемые, *555*
Транзакция, *554*; *556*; *780*; *794*
автоматическая, *557*; *561*
вложенная, *568*; *569*
изолированная, *555*
непротиворечивая, *555*
неразрывная, *554*
неявная, *558*; *561*
определяемая пользователем, *558*
распределенная, *559*; *946*
с автоматической фиксацией, *557*
устойчивая, *556*
Трассировка
Profiler, *999*
Триггер, *557*
INSTEAD OF, *745*; *832*

У

Удаление, *612*
Узел
AND, *513*
OR, *513*
корневой, *521*
листовой, *521*
Указатель
команд, *74*; *87*; *89*
на структуру OVERLAPPED, *309*
стека, *87*
файла, *253*
Унификатор, *522*; *954*
Уничтожение
курсора, *606*
открытого курсора, *606*
Упорядочение
доступа, *220*
доступа к динамической области памяти, *217*
Управление
памятью, *500*
распараллеливанием
оптимистическое, *577*
транзакцией явное, *578*
транзакциями автоматическое, *561*
Уровень
абстракции аппаратных средств, *108*
канальный, *363*
листовой, *521*
представительный, *363*
прикладной, *363*
приоритета, *110*
приоритета потока, *111*
сеансовый, *363*
сетевой, *363*
транспортный, *363*
физический, *363*
эталонной модели, *363*
Уровень изоляции транзакции, *555*; *563*

READ UNCOMMITTED, *563*
REPEATABLE READ, *566*
SERIALIZABLE, *567*
Условие
CREATE_SUSPENDED, *120*
родственности процессора, *115*
Установка
региональная, *834*
Устройство
внутреннее потока, *86*
вторичной памяти, *60*
доставки, *834*
подписчиков, *836*
Утечка
памяти, *502*
ресурсов, *76*
Утилита
BCP, *940*
Configure IIS Support, *746*
CopySample, *836*
dtsrunui, *890*
dumpbin, *62*; *175*
ImageCfg, *175*; *197*
NSControl, *822*; *839*
osql, *895*
STRESS.CMD, *832*
tail, *251*
TList, *103*; *241*; *380*; *402*

Ф

Фаза, *870*
завершения пакетного задания, *871*
завершения перекачки данных, *871*
преобразования строк, *870*
Файл, *71*
.INI, *409*
.LIB библиотеки DLL, *63*
AppConfig.XML, *823*
ApplicationDefinitionFileSchema.XSD, *823*
BCP, *940*
BOOT.INI, *60*
ConfigurationFileSchema.XSD, *823*
InstConfig.XML, *823*
LMHOSTS, *366*
NSService.exe, *824*
Ntkrnlpa.exe, *179*
Ntkrnlpamp, *180*
NTOSKRNL.EXE, *57*
PDB, *67*
UMS.DLL, *466*
winerror.h, *95*
выходной терминала CONOUT$, *379*
дампа стека, *101*
журнала исключительных ситуаций, *101*
исполняемый, *415*; *622*
определения приложения, *823*; *841*
отладочной информации, *67*; *68*
отображаемый на память, *64*; *230*; *339*; *340*
подкачки операционной системы, *60*
подкачки системный, *160*; *171*
сжатый, *461*
снимков, *940*
текстовый, *868*
хранения COM структурированный, *868*
Файл исполняемый
explorer.exe, *195*
snapshot.exe, *938*
sqlservr.exe, *459*
переносимый, *175*
программы SQL Server, *459*
ядра NTOSKRNL.EXE, *62*
Файл конфигурации, *408*; *822*
экземпляра, *823*
Фиксация
автоматическая, *561*
транзакции, *557*
транзакции двухфазная, *946*; *964*
Фильтр, *679*
.DOC, *679*
.HTM, *679*
.PPT, *679*
.TXT, *679*
.XLS, *679*
SORT, *285*
Флажок
FILE_FLAG_OVERLAPPED, *253*

IMAGE_FILE_LARGE_ADDRESS_
AWARE, *161*; *175*
OPEN_EXISTING, *253*
Форма
прокручиваемая, *591*; *596*
Формат
AUTO, *718*
BCP, *940*
EXPLICIT, *718*
PDB, *67*
PostScript, *436*
RAW, *718*
RTF, *436*
TeX, *436*
кодов ошибок Win32, *95*
краевой таблицы, *741*
файла отладочной информации, *67*
файловой системы, *247*
Фрагмент
XML, *790*
Фрагментация
виртуальной памяти, *343*
индекса, *524*
логического просмотра, *524*
просмотра экстента, *524*
Фрейм, *363*
Функция
_beginthreadex, *91*; *276*; *350*
_endthreadex, *91*
accept, *382*; *386*; *392*
AcceptEx, *382*; *392*
AllocateUserPhysicalPages, *173*; *245*; *488*
BaseProcessStart, *74*; *89*
BaseThreadStart, *87*; *89*
bind, *385*
CancelWaitableTimer, *146*
CheckForIoPacketAndSetState, *330*
CloseHandle, *74*; *254*; *255*
CLSIDFromProgID, *626*
CLSIDFromString, *626*
CoCreateInstance, *626*
CoInitialize, *415*; *621*
CoInitializeEx, *415*; *621*; *622*
COLUMNPROPERTY, *681*
COLUMNS_UPDATED, *947*; *965*
connect, *382*; *388*
ConnectNamedPipe, *367*; *378*
Create, *822*
CreateEvent, *146*
CreateFile, *232*; *250*; *269*; *341*; *367*
CreateFileMapping, *232*; *341*
CreateIOCompletionPort, *149*; *308*
CreateMappedFile, *242*
CreateMemoryResourceNotification, *494*
CreateNamedPipe, *367*; *378*
CreateObject, *410*; *624*
CreateProcess, *73*; *83*; *110*; *166*
CreateThread, *89*; *91*; *197*
CreateWaitableTimer, *146*
CreateWindow, *242*
CURSOR_STATUS, *613*
DeleteCriticalSection, *139*
DispatchMessage, *416*; *623*
DuplicateHandle, *146*; *250*
EnterCriticalSection, *139*; *140*
EXEC, *516*
execute_rpc, *128*
ExitProcess, *75*
ExitThread, *90*; *91*
FindFirstFile, *275*
FindNextFile, *275*
FlushViewOfFile, *199*
FormatMessage, *95*
fstring_pipe, *379*
FULLTEXTCATALOGPROPERTY, *681*
GetDiskFreeSpace, *256*; *276*
GetExitCodeProcess, *74*
gethostbyname, *388*
GetIDsOfNames, *627*
GetLastError, *95*; *102*
GetLock, *155*
GetMessage, *416*; *623*
GetOverlappedResult, *258*; *260*; *266*
GetProcAddress, *63*
GetProcessHeap, *218*
GetQueuedCompletionStatus, *149*;
150; *309*
GetSystemInfo, *165*; *184*
GlobalAlloc, *217*
GlobalMemoryStatus, *494*; *712*

GlobalMemoryStatusEx, *494*; *713*
HasOverlappedIoCompleted, *258*; *475*
HeapCreate, *218*
HeapDestroy, *218*; *220*
HOST_NAME, *983*
htonl, *385*
htons, *385*
ImpersonateNamedPipeClient, *366*; *367*
InitializeCriticalSection, *139*
InitializeCriticalSectionAndSpinCount, *140*
InterlockedCompareExchange, *339*
InterlockedExchange, *136*; *157*
InterlockedIncrement, *153*
ISABOUT, *691*
ISPALUSER, *982*
language_exec, *125*
LeaveCriticalSection, *139*
listen, *381*; *385*
LoadLibrary, *409*
LocalAlloc, *217*
LockFile, *251*
main, *71*; *84*; *275*
malloc, *167*; *181*; *196*; *217*
MapUserPhysicalPages, *173*; *245*
MapUserPhysicalPagesScatter, *173*
MapViewOfFile, *232*; *242*; *341*; *348*
memcpy, *388*
MessageBeep, *94*
MsgBox, *711*
MsgWaitForMultipleObjects, *147*
NEWID, *788*
NSInsertNotificationn, *829*
NtWaitForSingleObject, *126*
OBJECTPROPERTY, *646*; *681*; *685*
OleInitialize, *415*; *417*; *621*; *622*
OpenEvent, *146*
OpenFile, *250*
OpenOutputFile, *397*
OPENQUERY, *811*
OPENXML, *734*; *735*
PeekMessage, *416*; *623*
PostQueuedCompletionStatus, *149*; *150*; *308*
printf, *254*
QueryMemoryResourceNotification, *494*
QueueUserAPC, *257*; *325*
ReadFile, *254*; *255*; *258*; *312*; *367*
ReadFileEx, *257*; *258*
ReadFileScatter, *290*; *462*
ReadProcessMemory, *87*; *104*; *166*
recv, *382*; *386*
Register, *822*
ReleaseMutex, *148*
ReleaseSemaphore, *148*
ResetEvent, *145*
ResumeThread, *120*
SearchFile, *276*
SearchFiles, *275*
send, *382*
SendKeys, *239*
SetCriticalSectionSpinCount, *140*
SetErrorMode, *76*; *83*
SetEvent, *145*
SetFilePointer, *253*
SetLastError, *95*
SetPriorityClass, *110*
SetThreadAffinityMask, *117*
SetThreadIdealProcessor, *118*
SetThreadPriority, *111*; *880*
SetWaitableTimer, *146*
SetWindowsHookEx, *241*; *242*
SetWorkingSetSize, *201*
Sleep, *121*; *832*
SleepEx, *121*; *257*; *325*
socket, *381*
SpinToFindBuf, *325*
srv_alloc, *503*
StartSearch, *276*
strcpy, *168*; *180*
strncpy, *181*
strstr, *343*
SUSER_SNAME, *983*
SuspendThread, *120*
SwitchToThread, *121*; *137*
TerminateProcess, *75*
TerminateThread, *90*
TlsAlloc, *84*
TlsGetValue, *84*
TlsSetValue, *84*

TransmitFile, *391*
TryEnterCriticalSection, *140*
Update, *824*
VirtualAlloc, *173*; *192*; *198*; *199*; *268*; *286*; *487*
VirtualAllocEx, *192*
VirtualDecommit, *215*
VirtualFree, *192*; *200*
VirtualLock, *200*
VirtualProtect, *173*; *200*
VirtualQuery, *202*; *208*
VirtualQueryEx, *202*; *348*
VirtualUnlock, *201*
WaitForMultipleObjects, *144*; *150*; *157*
WaitForMultipleObjectsEx, *257*
WaitForSingleObject, *75*; *102*; *109*; *126*; *144*; *231*; *468*
WaitForSingleObjectEx, *257*; *267*
WaitNamedPipe, *378*
WideCharToMultiByte, *265*
WriteCompleted, *265*
WriteFile, *255*; *258*; *265*; *367*
WriteFileEx, *257*; *258*; *265*
WriteFileGather, *462*
WriteFileScatter, *290*
WriteProcessMemory, *87*; *166*; *171*
WSAAccept, *382*; *392*
WSASocket, *381*; *385*
ZeroMemory, *265*
ввода-вывода, *459*
доступа к метаданным, *681*
инициализации COM, *415*; *621*
инициализации WSAStartup, *381*
интерфейса OLE DB, *463*
комплексной блокировки, *134*; *138*; *322*
недокументированная, *1000*
обработки массивов, *663*
обратного вызова, *239*
ожидания, *109*; *133*; *144*; *257*
определяемая пользователем, *828*
открытия файла, *250*
потока, *92*
рассылки извещений, *828*
сетевая, *459*
системная, *639*
точки входа, *71*; *84*
экспортируемая, *62*
Функция набора строк, *690*
CONTAINSTABLE, *690*
FREETEXTTABLE, *680*; *693*
Функция-заглушка, *457*
Функция-посредник, *457*

X

Хост-контейнер
MTA, *416*
Хранилище данных
нереляционное, *868*

Ц

Цикл
простоя, *477*

Ч

Частота
тактового интервала, *108*
Число
"магическое", *516*

Ш

Шаблон, *437*
запроса, *748*
обновления, *779*
определения различий, *799*
параметризованный, *760*
проекта декораторный, *43*
проекта фасадный, *43*
проектирования, *347*
серверный, *758*
Шаг
быстрого составления плана, *512*; *513*
обработки транзакции, *512*
параллельной оптимизации, *512*; *513*
полной оптимизации, *512*; *513*

Э

Экземпляр
Notification Services, *821*
схемы XML, *436*
Экран
с синим фоном, *178*
синий, *177*
Экспорт
процедуры DLL, *62*
таблицы, *645*
функции, *62*
Экстент, *520*
Элемент
<?MSSQLError>, *786*
after, *780*
before, *780*
header, *784*
Record, *796*
sql:field, *772*
sql:relation, *772*
sync, *780*; *788*
xsl:attribute, *443*
xsl:choose, *443*
xsl:for-each, *443*
xsl:if, *443*
xsl:sort, *443*
корневой, *428*
Элемент управления
ActiveX, *406*
OCX, *406*
OLE, *406*
Этап
On Insert Failure, *871*
On Insert Success, *871*
On Transform Failure, *870*
полной оптимизации, *511*
преобразования, *514*
тривиальной оптимизации плана, *509*
упрощения, *511*

Я

Ядро
операционной системы, *57*; *62*
Язык
C, *26*
C#, *444*
C++, *196*
Delphi, *62*
DML, *873*
DSSSL, *423*
ECMAScript, *443*
HTML, *421*
IDL, *905*
Java, *444*
JavaScript, *443*
JScript, *671*
SGML, *423*
SQL-92, *558*
VB, *67*
VBScript, *621*; *671*
Visual Basic, *26*; *444*; *621*; *659*; *711*; *865*
Visual C++, *62*; *67*; *194*; *414*
WSDL, *804*
XML, *50*; *420*; *421*; *708*
XPath, *438*; *711*; *748*
XSD, *771*
XSLT, *436*
определения интерфейса, *905*
поддержки сценариев, *671*
разметки, *421*
Ярус, *364*

“Я могу со всей уверенностью сказать, что любой, кто регулярно использует систему управления базами данных SQL Server, сможет узнать для себя много нового, прочитав данную книгу (это относится даже к специалистам компании Microsoft из штаб-квартиры компании в городе Редмонт, повседневно использующим СУБД SQL Server в своей работе)”.

Дэвид Кэмпбелл, руководитель производственного подразделения группы реляционных серверных СУБД, корпорация Microsoft

Очередная книга пользующегося огромным авторитетом автора многих бестселлеров Кена Хендерсона, которая может стать незаменимым источником информации о СУБД Microsoft SQL Server. В книге принят комплексный подход к изложению сведений об архитектуре, внутренней организации и настройке SQL Server, дополняющих существующие справочные руководства и официальную документацию, что позволило достичь непревзойденной глубины и широты охвата данной темы.

Это практическое руководство, в котором за подробным обсуждением неизменно следуют конкретные рекомендации и примеры, начинается с нескольких глав по фундаментальным технологиям Windows, лежащим в основе работы SQL Server, включая процессы и потоки, управление памятью, ввод-вывод и организацию сетей. После этого излагаются сведения о том, из каких компонентов состоит СУБД SQL Server и как функционируют эти компоненты.

Рассматривается весь спектр современных возможностей программного продукта SQL Server, а не только те функциональные средства, которые реализованы с помощью основного исполняемого файла или доступны пользователям уже в течение многих лет. С каждым новым выпуском СУБД SQL Server становится все более зрелой, а круг задач, решаемых с ее помощью, существенно расширяется. Поэтому в книге подробно описаны передовые технологии, которые еще не нашли достаточного отражения в литературе, включая Notification Services, Full Text Search, SQLXML, репликацию, DTS и множество других.

Для анализа функционирования SQL Server используется WinDbg, символический отладчик компании Microsoft, предоставляемый для бесплатно загрузки. Вооружившись вновь приобретенными с ег помощью навыками отладки и разработки кода, читатели смогут продолжить самостоятельное овладение возможностями программного продукта SQL Server.

Прилагаемый к книге компакт-диск содержит большое количество вспомогательных материалов, в том числе полный исходный код больше чем 900 примеров из книги, а также три бесценных инструментальных средства: приложение DTSDIAG, библиотеку VBODSOLE и программу DTS Package Guru. Приложение DTSDIAG позволяе разработчикам и администраторам одновременно собирать такую информацию, как трассировки Profile зафиксированные в журналах сведения о производительности; выходные данные сценариев обнаружения блокировок; содержимое записей системных журналов регистрации событий; а также сообщения SQLDIAG от контролируемой программы SQL Server. В библиотеке VBODSOLE содержится больше двадцати новых функций для сценариев Transact-SQL (T-SQL), созданных на основе технологии COM, в том числе такие расширения T-SQL, как процедуры манипулирования массивами, финансовые функции, функции для работы со строками и системные функции. Программа DTS Package Guru — это редактор пакетов для службы Da Transformation Services программного продукта SQL Server на основе .NET, которая позволяет редактировать любой пакет, доступный для модификации, а также обеспечивает автоматизацию массовых правок пакетов.

Кен Хендерсон — разработчик программного обеспечения СУБД с большим стажем, который участвовал в реализации проектов для ВВС и ВМФ США, компании H&R Block, компании Travelers Insurance, компании J.P. Morgan, ЦРУ и многих других организаций. Он регулярно публикует в журналах свои статьи и является автором сем ранее выпущенных книг, включая такие бестселлеры, как *The Guru's Guide to Transact-SQL* (Addison-Wesley, 2000) и *The Guru's Guide to SQL Server Stored Procedures, XML, and HTML* (Addison-Wesley, 2002).

ISBN 5-8459-0912-0

ВИЛЬЯМС

Addison-Wesley
Pearson Education

www.williamspublishing.com
www.awprofessional.com

9 785845 909121 05187