.NET Framework

Essentials

Second Edition

Thuan Thai & Hoang Q. Lam

O'REILLY®

Платформа .NET

Основы

Второе издание

Туан Тай, Хонг К.Лэм

Санкт-Петербург
2003

Туан Тай, Хонг К. Лэм

Платформа .NET. Основы, 2-е издание

Перевод Л. Фрейдина

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*А. Королев*
Редактор	*В. Овчинников*
Корректура	*С. Беляева*
Верстка	*Н. Гриценко*

Тай Т., Лэм Х. К.
Платформа .NET. Основы. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 336 с., ил.
ISBN 5-93286-054-5

Перед вами краткое введение в платформу Microsoft .NET, призванное помочь разработчикам перейти от традиционного программирования для Windows к созданию приложений в среде .NET. В книге подробно описаны общеязыковая среда выполнения Common Language Runtime (CLR) и набор базовых классов, радикально упрощающих разработку крупномасштабных приложений. Рассмотрен механизм языковой интеграции и приведено описание цикла компонентной и корпоративной разработки с использованием .NET Framework. Кроме того, обсуждаются основы ключевых технологий .NET: работа с данными (ADO.NET) и XML, веб-службы (Web Services), веб-формы (Web Forms, ASP.NET) и Windows Forms.

Книга предназначена в основном для разработчиков, имеющих опыт в программировании COM, создании объектно-ориентированных, компонентных и корпоративных Windows-приложений с помощью языков Visual Basic, Visual C++, Java™ и C/C++, но может быть полезна всем желающим изучать Microsoft .NET Framework.

ISBN 5-93286-054-5
ISBN 0-596-00302-1 (англ)

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 14.10.2002. Формат 70x100 1/16 . Печать офсетная.
Объем 21 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие 7

1. Обзор .NET 13

Microsoft .NET 14
Платформа .NET 15
Цели разработки .NET Framework 16
.NET Framework 22

2. Common Language Runtime 25

Среда CLR 25
Исполняемые файлы CLR 26
Метаданные 31
Сборки и манифесты 39
Промежуточный язык IL 46
CTS и CLS 49
Выполнение в среде CLR 54
Заключение 60

3. Программирование в среде .NET 61

Общая модель программирования 61
Языковая интеграция 78
Заключение 83

4. Работа с .NET-компонентами 84

Варианты развертывания 84
Распределенные компоненты 95
Службы COM+ в .NET 99
Организация пула объектов 106
Очереди сообщений 113
Заключение 116

5. Данные и XML 117

Архитектура ADO.NET 118
Достоинства ADO.NET 119
Компоненты сущностей 122
Управляемые поставщики 136
Наборы данных и XML 148
Заключение 160

6. Веб-службы 161

Веб-службы на практике 161
Структура Web Services 164
Поставщик веб-службы 175
Клиенты веб-служб 180
Веб-службы и безопасность 199
Заключение 201

7. Веб-формы 202

ASP 202
ASP.NET 204
Пространство имен System.Web.UI 205
Синтаксис веб-форм 214
Разработка приложения ASP.NET 222
ASP.NET и веб-службы 238
Привязка данных и применение шаблонов 241
Управление состоянием и масштабируемость 247
Заключение 253

8. Windows Forms 254

Введение в Windows Forms 254
Пространство имен System.Windows.Forms 256
Разработка Windows Forms 262
Windows Forms и веб-службы 287
Заключение 288

A. Языки .NET 289

B. Основные сокращения 291

C. Общие типы данных 297

D. Основные утилиты 304

Алфавитный указатель 314

Предисловие

Данная книга представляет собой краткое введение в Microsoft .NET Framework и предназначена для того, чтобы помочь программистам осуществить переход от традиционного программирования для Windows к миру программирования в среде .NET. Microsoft .NET Framework включает Common Language Runtime (CLR, общеязыковая среда выполнения) и набор базовых классов, радикально упрощающих разработку крупномасштабных приложений и сервисов. Книга детально описывает CLR, так что вы сможете активно применять ее новые возможности. В ней показано, как в действительности работает языковая интеграция, и приведено описание цикла компонентной и корпоративной разработки с использованием .NET Framework. Кроме того, имеется введение в четыре ключевых технологии .NET: работу с данными (ADO.NET) и XML, веб-сервисы (Web Services), веб-формы (Web Forms, ASP.NET) и Windows Forms.

Для подготовки рукописи, создания всех примеров и рисунков в этой книге мы использовали последнюю версию Microsoft Visual Studio .NET и .NET Framework SDK. Хотя мы старались сделать все возможное, чтобы техническое содержимое книги было актуальным, возможно, что кое-что с момента написания книги изменилось. Чтобы все время быть в курсе изменений, регулярно посещайте сайты *http://msdb.microsoft.com/net*, *http://www.gotdotnet.com* и страницу этой книги на сайте O'Reilly, *http://www.oreilly.com/catalog/dotnet_frmess2/*.

Наша аудитория

Хотя данная книга предназначена для всех желающих узнать о Microsoft .NET Framework, она нацелена на разработчиков с опытом создания Windows-приложений с помощью Visual Studio 6 и языков Visual Basic и Visual C++. Разработчики на Java™ и C/C++ также хорошо подготовлены для представленного здесь материала. Чтобы получить от книги максимум, вам следует иметь опыт в разработке объектно-ориентированных, компонентных, корпоративных и веб-приложений. Опыт в программировании COM также будет весьма полезным.

Об этой книге

Для тех, кто не знаком с каждой из описанных технологий, мы построили эту книгу, основываясь на кратком курсе, который Туан (Thuan) читал в нескольких компаниях с августа 2000 года, таким образом, что каждая глава базируется на знаниях из предыдущей главы. Чтобы дать вам общий обзор, приведем краткие аннотации глав и приложений этой книги.

Глава 1 «Обзор .NET» содержит краткое описание платформы Microsoft .NET. Она рассказывает о целях разработки .NET Framework и знакомит вас с компонентами .NET Framework.

В главе 2 «Common Language Runtime» мы приподнимем «капот» и заглянем внутрь CLR. Здесь рассматривается богатая библиотека времени выполнения CLR, а также другие средства.

В главе 3 «Программирование в .NET» содержится введение в .NET-программирование. Вы изучите простую программу, использующую концепцию объектно-ориентированной и компонентной разработки на четырех различных языках: Managed C++, VB.NET, C# и IL. Вы также сможете испытать преимущества языковой интеграции.

Глава 4 «Работа с .NET-компонентами» демонстрирует простоту компонентной и корпоративной разработки в .NET. Кроме рассмотрения средств развертывания компонентов, вы также исследуете полноценные программы, использующие преимущества транзакций, пула объектов, безопасности на основе ролей и очереди сообщений – и все в одной главе.

В главе 5 «Данные и XML» описана архитектура ADO.NET и ее преимущества. Кроме возможности отсоединения, что способствует улучшению масштабируемости, набор данных ADO.NET также тесно интегрирован с XML, что улучшает возможности взаимодействия между платформами. Глава знакомит вас с объектами .NET для доступа к данным, а также с пространством имен XML.

В главе 6 «Веб-службы» описано следующее поколение программных компонентов, к которым можно осуществлять доступ через Интернет. В этой главе мы обсудим протоколы, поддерживаемые веб-службами, а также опишем, как публиковать и находить их. Вы узнаете, каким образом XML, используемый в связке с HTTP, ломает закрытую природу текущей модели компонентно-ориентированной разработки программного обеспечения и обеспечивает лучшие возможности межсетевого взаимодействия.

В главе 7 «Веб-формы» дано введение в среду ASP.NET, которая теперь поддерживает объектно-ориентированное и управляемое событиями программирование, в отличие от традиционной разработки ASP.

В этой главе центральное место занимают веб-формы и серверные элементы управления. Кроме того, вы узнаете, как создавать собственные серверные элементы управления, выполнять привязку данных к различным элементам управления .NET, и изучите средства управления состоянием в ASP.NET.

Глава 8 «Windows Forms» переносит традиционное программирование на основе форм на шаг в будущее с помощью классов пространства имен System.Windows.Forms. Аналогично Win32-приложениям, Windows Forms лучше всего использовать для так называемых мощных или «толстых» клиентов, однако с новой, упрощенной процедурой инсталляции в .NET и приходом веб-сервисов Windows Forms стали пригодными для более широкого круга приложений.

Приложение A «Языки .NET» содержит список ссылок на веб-сайты с информацией о языках, использующих CLR, включая некоторые находящиеся в зачаточном состоянии проекты с открытым исходным кодом.

Приложение B «Основные сокращения» содержит список часто используемых сокращений, применяемых в литературе о .NET и на презентациях.

Приложение C «Общие типы данных» содержит несколько списков наиболее часто используемых в .NET типов данных. В этом приложении также проиллюстрировано применение некоторых классов коллекций.

В приложении D «Основные утилиты» приведен обзор наиболее важных утилит, предоставляемых .NET SDK для облегчения разработки в среде .NET.

Теперь, когда вы знаете, о чем эта книга, мы должны объяснить, чего в ней нет. В этой книге мы не обращаемся к маркетинговым аспектам .NET или других компонентов платформ .NET, таких как .NET Enterprise Servers, .NET Building Block Services или операционных систем .NET. Кроме того, не рассмотрен недавно объявленный сервис HailStorm, а также деятельность Microsoft по обеспечению доступности .NET Framework на многочисленных типах устройств.

Требования к читателю

В этой книге предполагается, что вы разработчик Windows- и веб-приложений, свободно разбирающийся в объектно-ориентированном и компонентном программировании. Также предполагается, что вы имеете базовые знания по XML. Хотя знание COM не является обязательным требованием, если вы имеете опыт в программировании COM, то в еще большей степени оцените эту книгу и .NET Framework.

Соглашения, используемые в этой книге

В этой книге приняты следующие соглашения о шрифтовом оформлении.

Курсив используется для:

- имен каталогов, файлов и программ;
- интернет-адресов, таких как имена доменов и URL;
- новых терминов там, где они определяются.

`Моноширинный шрифт` применяется для:

- командных строк и параметров, которые требуют дословного написания;
- прямых цитат и конкретных имен методов из примеров кода, а также конкретных значений атрибутов и параметров в коде;
- тегов XML-элементов.

`Моноширинный жирный шрифт` применяется для:

- вводимых пользователем данных в коде, которые должны печататься дословно;
- мест в коде, на которые мы бы хотели обратить внимание читателя.

`Моноширинный курсив` применяется для элементов кода, которые следует заменить на конкретные значения.

В примерах синтаксиса кода мы иногда будем использовать обозначение `[`*`value`*`]+` для одного или более экземпляров значения value и `[`*`value`*`]*` для обозначения нуля или более экземпляров значения value.

Как с нами связаться

Мы, насколько могли, аккуратно протестировали и сверили информацию из этой книги, однако вы можете обнаружить произошедшие изменения (или даже допущенные нами ошибки!). Просьба сообщать нам о любых найденных ошибках, а также о предложениях по будущим изданиям, написав по адресу:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (FAX)

Вы также можете присылать нам и электронные сообщения. Чтобы включить свой адрес в список рассылки или запросить каталог, отправьте письмо по адресу

info@oreilly.com

Чтобы задать технические вопросы или прислать комментарии к книге, отправьте письмо по адресу

bookquestions@oreilly.com

У нас есть веб-сайт этой книги, где мы поместим примеры, найденные ошибки и любые планы будущих изданий. Эта страница доступна по адресу

http://www.oreilly.com/catalog/dotnetfrmess2/

Дополнительную информацию по этой и другим книгам можно найти на веб-сайте издательства O'Reilly

http://www.oreilly.com

Дополнительная информация о среде .NET в целом приведена на .NET-центре издательства O'Reilly *http://dotnet.oreilly.com* и на *.NET DevCenter по адресу http://www.oreillynet.com/dotnet/*.

Благодарности

Сотрудники издательства O'Reilly не устают поражать нас предоставляемой ими поддержкой. Я хотел бы поблагодарить Джона Осборна (John Osborn), продлившего договор для написания этой книги, за постоянную поддержку проекта. Мы также хотим поблагодарить Нэнси Котари (Nancy Kotary) за выполненную трудную работу, позволившую обеспечить выход книги в установленные жесткие сроки. Нэнси провела большую работу по рецензированию наших материалов и координации проекта. Без Джона и Нэнси эта книга никогда не была бы написана. Спасибо сотрудникам O'Reilly, занимающимся производством и дизайном, за воплощение этой книги в реальность: Эмме Колби (Emma Colby), Татьяне Диаз (Tatiana Diaz), Дэвиду Футато (David Futato), Колину Горману (Coleen Gorman), Роберту Романо (Robert Romano), Майку Сьерра (Mike Sierra), Элли Волькхаузен (Ellie Volckhausen) и Джо Визда (Joe Wizda).

Спасибо Брайану Джепсону (Brian Jepson), внесшему значительный вклад в эту книгу с самого начала проекта. Брайан проделал, безуслов-

но, выдающуюся работу по чтению, тестированию и обеспечению соответствия технического содержимого каждой главы последнему релизу. Мы также хотим поблагодарить Деннис Анжелин (Dennis Angeline) и Брэда Меррила (Brad Merrill) из Microsoft за ответы на технические вопросы по CLR и языкам программирования.

Хоанг (Hoang) хочет поблагодарить своих родителей и семью за их поддержку и понимание того, что он выпал из их жизни на несколько месяцев. Мама и папа, ваши постоянные усилия, чтобы дети оказались там, где они сейчас, никогда не могут быть оплачены. Хоанг также благодарит свою жену, ВанДу (VanDu), источник своего вдохновения. Не надо недооценивать ее вклад в эту книгу. И, наконец, последнее, но не менее важное, персональная *благодарность* Туану (Thuan), который все время подталкивал меня к долгожданному завершению книги.

5

Данные и XML

Почти все, что мы делаем в программной индустрии, так или иначе имеет отношение к данным. На каком-то этапе все разработчики программного обеспечения должны работать с данными, возможно, с применением базы данных, текстового файла, электронной таблицы или другого метода хранения данных. Существует много различных технологий для работы с данными и управления ими, и постоянно появляются все новые методы, позволяющие улучшить эти технологии. Данные методы начинаются с функциональных API и заканчиваются объектно-ориентированными средами и фирменными библиотеками.

Несколько лет назад для простого настольного приложения на Visual Basic было обычным обращение к базе данных в формате Microsoft Access, хранящейся на локальном жестком диске, но сейчас такая ситуация уже не типична. Современные приложения используют преимущества технологий распределенных компонентов для достижения масштабируемости и межплатформенного взаимодействия, тем самым расширяя сферу применимости приложения до уровня предприятия. Если ActiveX Data Objects (ADO) хорошо служили VB-приложению несколько лет назад, скоро они не смогут соответствовать возрастающим требованиям к масштабируемости, производительности и возможности взаимодействия между несколькими платформами.

Тут на помощь приходит ADO.NET, предоставляющая огромные преимущества, которые позволяют создавать еще лучшие корпоративные приложения. В этой главе рассказывается о преимуществах ADO.NET, ее архитектуре, основных классах ADO.NET, о том, как эти классы работают, и об интеграции ADO.NET и XML.

Архитектура ADO.NET

Объектная модель Microsoft ADO.NET включает в себя две отдельных группы классов: *компоненты сущностей* (*content components*) и *компоненты управляемых поставщиков* (*managed-provider components*). Компоненты сущностей включают класс DataSet и другие вспомогательные классы, такие как DataTable, DataRow, DataColumn и DataRelation. В этих классах находится фактическое содержимое пересылаемых данных. Компоненты управляемых поставщиков помогают в извлечении и обновлении данных. Разработчики могут использовать объекты Connection, Command и DataReader для непосредственных манипуляций с данными. Чаще всего разработчики отводят классу DataAdapter роль канала передачи данных между хранилищем данных и компонентами сущностей. Данные могут быть либо реальными строками из базы данных, либо иметь другую форму, такую как XML-файл или таблица Excel.

На рис. 5.1 показана высокоуровневая архитектура ADO.NET. Разработчики, знакомые с ADO, без труда поймут назначение объектов Connection и Command. Сейчас мы предлагаем вашему вниманию краткий обзор, а детали будут рассмотрены позже.

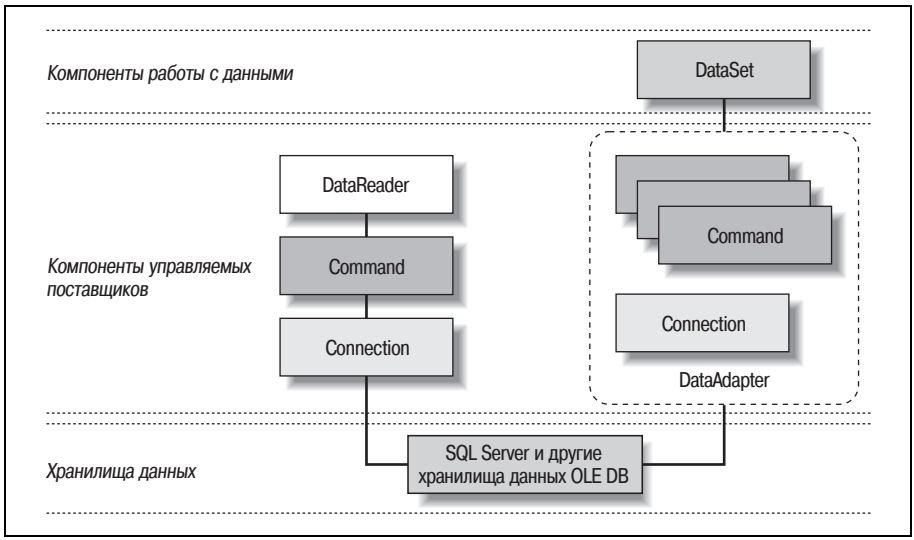

Рис. 5.1. *Высокоуровневая архитектура ADO.NET*

DataReader – это новый объект, предоставляющий быстрый, однонаправленный, только для чтения, доступ к данным. Эта структура схожа с ADO Recordset с серверным, однонаправленным, доступным только для чтения курсором.

Класс DataSet аналогичен облегченному кэшу отдельной базы данных из хранилища данных. Он позволяет читать и писать данные и схему в

XML и, как вы увидите позднее, тесно интегрирован с классом XmlDataDocument.

Класс DataAdapter представляет собой высокоуровневую абстракцию классов Connection и Command. Он позволяет загружать содержимое хранилища данных в объект DataSet и согласовывать изменения в объекте DataSet с хранилищем данных.

Достоинства ADO.NET

ADO.NET обладает рядом достоинств, которые можно разделить на следующие категории:

Межплатформенное взаимодействие

Возможность коммуникации между разнородными средами.

Масштабируемость

Возможность обслуживания большого количества клиентов без сокращения производительности системы.

Продуктивность

Возможность быстрой разработки надежных приложений для доступа к данным с помощью богатой и расширяемой компонентной объектной модели ADO.NET.

Производительность

Улучшение производительности по сравнению с предыдущими версиями ADO, обусловленное использованием модели отсоединенных данных.

Межплатформенное взаимодействие

Любые коммуникации связаны с обменом данными, будь это коммуникация между распределенными компонентами по методике «запрос-отклик» или технология, базирующаяся на сообщениях. Современные распределенные системы предполагают, что компоненты, участвующие в коммуникации, используют один и тот же протокол и формат данных. Это предположение является слишком ограничивающим для клиентской базы в масштабах предприятия или нескольких компаний. Уровни доступа к данным не должны накладывать таких ограничений.

В современной архитектуре распределенных сетевых приложений Microsoft Windows (Distributed interNet Applications, DNA) компоненты приложений передают и получают данные в виде отсоединенных наборов записей ADO. Компоненты, предоставляющие данные, как и компоненты, получающие данные, обязаны использовать технологию COM (Component Object Model). *Полезная нагрузка* (*payload*), т. е. фак-

тически передаваемое содержимое, упаковывается в формат данных под названием Network Data Representation (NDR). Из этих NDR-пакетов формируются потоки между компонентами.

Современным системам Windows DNA присущи две проблемы. Первая – необходимость наличия библиотеки COM на обеих сторонах коммуникационного канала. Вторая – сложность в установке и в организации этого взаимодействия через брандмауэры. Если ваши компоненты промежуточного уровня базируются на COM/DCOM и используются во внутренней сети, все будет хорошо. Другими словами: если все ваши компоненты основаны на технологии Microsoft, все нормально. Однако электронная коммерция (e-commerce) требует от корпоративных приложений умения взаимодействовать не только с системами, разработанными Microsoft. Необходимо улучшение библиотеки ADO, имеющее целью преодоление ограничений COM/DCOM и устранение затруднений, возникающих при совместном использовании данных многоплатформенными компонентами.

ADO.NET снимает ограничения на обмен данными, используя XML в качестве формата полезной нагрузки. Поскольку этот формат текстовый и легко обрабатывается, он представляет собой хороший выбор в качестве общего, платформо-независимого и переносимого формата данных. Более того, XML есть ни что иное, как структурированный текст, и поэтому его применение в качестве формата данных поверх сетевого протокола HTTP минимизирует проблемы, связанные с брандмауэрами. При использовании ADO и его XML-формата клиентам не надо знать COM для воссоздания упакованных данных. Необходим лишь XML-анализатор, доступный во многих вариантах на многих различных платформах. Производители и потребители данных должны только придерживаться XML-схемы для обмена данными между собой.

Масштабируемость

В модели «клиент-сервер» клиент обычно устанавливает соединение с сервером и удерживает его, пока не будут выполнены все запросы. Такое решение нормально работает в малых и средних приложениях, но оно не масштабируется на крупное предприятие. Как только количество клиентов достигает определенного предела, сервер становится узким местом, поскольку соединения с базами данных съедают ресурсы сети и процессора. ADO.NET отходит от модели «клиент-сервер», внедряя использование отсоединенных наборов данных. Когда клиент запрашивает какие-либо данные, они извлекаются, передаются клиенту, и (как только это становится возможным) соединение разрывается. Соединение между клиентом и источником данных является кратковременным, поэтому данный прием позволяет большему количеству клиентов запрашивать информацию от сервера, решая тем самым проблему ограниченного количества соединений.

Может показаться, что установка и разрыв соединений – это неудачная идея, т. к. затраты на установление соединений обычно высоки. Но это имеет значение лишь в отсутствие пула соединений. ADO.NET автоматически сохраняет соединения с источником данных в пуле, поэтому, когда приложение думает, что оно разрывает соединение, фактически оно возвращает соединение в пул ресурсов. Это позволяет повторно использовать соединения, избегая затрат на восстановление соединения с нуля.

Такая работа с данными не новость для ADO-программистов. Отсоединенные наборы записей появились в более ранних версиях ADO. Однако в ADO разработчик сам отвечал за реализацию этой возможности, тогда как в ADO.NET данные являются отсоединенными по своей природе.

Библиотека ADO.NET улучшена по сравнению с предшественником, вырастая из модели «клиент-сервер» в модель распределенных компонентов DCOM. Отсоединенные наборы данных как парадигма обмена данными делают ADO.NET значительно лучше масштабируемой по сравнению со своими предшественниками.

Продуктивность

Богатые возможности базовых классов ADO.NET позволяют разработчикам увеличить производительность труда. У тех, кто работает с текущей версией ADO, не должно быть трудностей с быстрым переходом к новой объектной модели, поскольку ADO.NET является естественным развитием ADO. Базовая функциональность осталась прежней. У нас все еще есть объект Connection, представляющий собой канал связи, через который выполняются команды.[1] В ADO.NET функциональность разбивается и распределяется между всеми объектами модели значительно лучше, чем в предыдущих версиях ADO. Например, объект Connection теперь отвечает только за присоединение и отсоединение от источника данных. В ADO.NET мы больше не можем выполнить запрос непосредственно через объект Connection. Не исключено, что некоторым разработчикам будет не хватать данной возможности, но это шаг в правильном направлении, т. к. он ведет к единству компонентной разработки.

ADO.NET также увеличивает производительность труда разработчиков за счет расширяемости. Базовые классы ADO.NET являются управляемым кодом, поэтому разработчики могут наследовать и расширять эти классы для своих специальных нужд. Те, кто предпочитает не выполнять эту низкоуровневую работу, могут использовать среду работы с данными в Visual Studio.NET для генерации этих классов.

---

[1] Вместе со знакомыми объектами Connection и Command ADO.NET предоставляет несколько новых объектов, таких как DataSet и DataAdapter. Все эти объекты обсуждались в разделе «Архитектура ADO.NET» ранее в этой главе.

Visual Studio.NET представляет собой прекрасное средство быстрой разработки приложений (RAD), базирующееся на ADO.NET. Для генерации типизированных наборов данных ADO.NET может применяться Component Designer. Типизированные наборы данных – это расширенные типы, моделирующие ваши данные. Код, полученный при помощи Component Designer, читается значительно легче, чем тот, который получался на выходе предыдущих генераторов кода от Microsoft. Кроме того, эти сгенерированные классы безопасны в отношении типов, сокращая таким образом возможность ошибок и позволяя компиляторам и CLR проверять правильность использования типов.

Короче говоря, ADO.NET улучшает продуктивность разработчиков за счет богатых и расширяемых базовых классов. Эти возможности дополняются богатым набором средств Visual Studio.NET для ADO.NET, обеспечивающих быструю разработку приложений.

Производительность

В ADO.NET в основном применяются отсоединенные наборы данных, поэтому система получает преимущества за счет улучшения производительности и масштабируемости. Сервер баз данных больше не является узким местом при росте количества запросов на соединение. Управляемые поставщики в ADO.NET также обеспечивают неявную организацию пула соединений, что сокращает время, требуемое для открытия соединения.

Ранее для обеспечения совместимости типов данных с COM маршалинг наборов записей требовал преобразования типов. Поскольку отсоединенный набор данных использует XML-формат, больше нет необходимости в таком преобразовании во время передачи данных, в отличие от работы с данными в формате NDR (Network Data Representation).

Компоненты сущностей

Компоненты сущностей служат для инкапсуляции данных. В предыдущих версиях ADO таким компонентом являлся объект Recordset. Данные, содержащиеся в компоненте Recordset, представлены в форме таблицы, состоящей из столбцов и строк. В ADO.NET данные, инкапсулированные в компоненте DataSet, имеют форму реляционной базы данных, состоящей из таблиц и отношений между ними. Это и есть основное усовершенствование в технологии доступа к данным. В этом разделе мы приведем общий обзор базовых классов, представляющих собой компоненты сущностей, в том числе DataSet, DataTable, DataColumn, DataRow, DataView и DataRelation.[1]

[1] Полный список всех классов можно найти в Microsoft .NET SDK.

DataSet

Если вы знакомы с ADO, то знаете, что обычно данные между компонентами передаются в виде *наборов записей* (*recordsets*). Набор записей содержит данные в табличной форме. Независимо от того, включает ли набор записей информацию из одной или нескольких таблиц базы данных, данные всегда возвращаются в виде строк и столбцов, словно они взяты из одной таблицы. ADO.NET позволяет компонентам приложения совместно использовать нечто большее, чем просто набор записей. Вот одна из наиболее важных черт ADO.NET: вместо набора записей передается набор данных (DataSet).

Объект DataSet можно рассматривать как размещенное в памяти представление базы данных. Он может содержать несколько объектов DataTable и DataRelation. В предыдущих версиях ADO наиболее близкую к этому функциональность можно было получить, обмениваясь данными в виде цепочки объектов Recordset. Клиентское приложение, получив эту цепочку, могло добраться до любого набора записей с помощью метода NextRecordset(); однако не было способа описать отношения между наборами записей в цепочке. В ADO.NET разработчики могут перемещаться и манипулировать коллекцией таблиц и отношений между ними.

Как мы упомянули ранее, ADO.NET применяет отсоединенные наборы данных, т. к. библиотека нацелена на распределенную архитектуру. Отсоединенный набор данных должен предоставлять способ отслеживать внесенные изменения. Объект DataSet предоставляет несколько методов, позволяющих позднее легко согласовать все выполненные с набором данных манипуляции с реальной базой данных (или другим источником данных). Эти методы включают: HasChanges(), HasErrors, GetChanges(), AcceptChanges() и RejectChanges(). Они могут применяться для проверки изменений, произошедших в объекте DataSet, получить изменения в форме измененного DataSet, проверить изменения на ошибки и затем принять или отменить изменения. Для того чтобы передать изменения в хранилище данных (наиболее обычная ситуация), просто попросите DataSet выполнить обновление.

DataSet полезен для корпоративных веб-приложений, по своей природе являющихся отсоединенными. Об изменениях данных на сервере мы не узнаем, пока не обновим отредактированные записи или не выполним любые другие действия, требующие согласования с базой данных.

Как показано на рис. 5.2, DataSet содержит две важных коллекции. Первая – это Tables (имеющая тип DataTableCollection), содержащая коллекцию таблиц, относящихся к набору данных. Вторая коллекция содержит все отношения между таблицами и соответственно имеет название Relations (и тип DataRelationCollection).

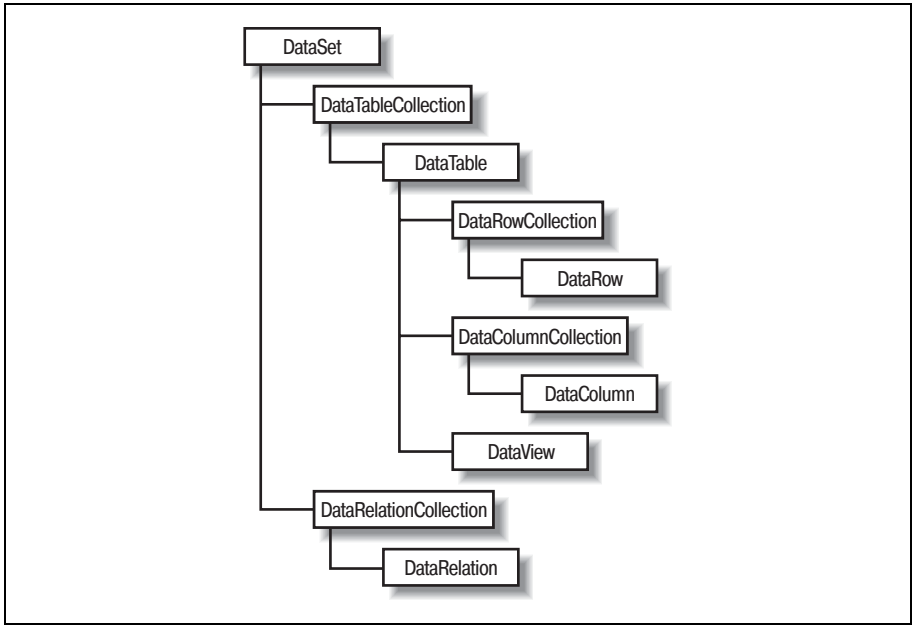

***Рис. 5.2.** Важнейшие объекты данных в ADO.NET, включенные в DataSet*

Создание набора данных: пример на C#

Все таблицы и отношения внутри набора данных доступны через свойства Tables и Relations соответственно. Обычно таблицы берутся из какого-либо источника данных, такого как SQL Server, или из другой базы данных; однако сначала мы хотим показать здесь основные элементы набора данных. Следующий блок кода на C# демонстрирует, как динамически создать набор данных, состоящий из двух таблиц, Orders и OrderDetails, и отношение между этими двумя таблицами.

```
using System;
using System.Data;

// Для краткости объявления класса и метода опущены...

// Создаем объект DataSet
DataSet m_ds = new DataSet("DynamicDS");

// Добавляем новую таблицу "Order" к коллекции таблиц объекта m_ds
m_ds.Tables.Add ("Order");

// Добавляем новые столбцы к таблице "Order"
m_ds.Tables["Order"].Columns.Add("OrderID",
                                 Type.GetType("System.Int32"));
m_ds.Tables["Order"].Columns.Add("CustomerFirstName",
                                 Type.GetType("System.String"));
```

```
m_ds.Tables["Order"].Columns.Add("CustomerLastName",
                                 Type.GetType("System.String"));
m_ds.Tables["Order"].Columns.Add("Date",
                                 Type.GetType("System.DateTime"));

// Регистрируем столбец "OrderID" как первичный ключ таблицы "Order"
DataColumn[] keys = new DataColumn[1];
keys[0] = m_ds.Tables["Order"].Columns["OrderID"];
m_ds.Tables["Order"].PrimaryKey = keys;

// Добавляем новую таблицу "OrderDetail" в коллекцию таблиц объекта m_ds
m_ds.Tables.Add ("OrderDetail");

// Добавляем новые столбцы к таблице "OrderDetail"
m_ds.Tables["OrderDetail"].Columns.Add("fk_OrderID",
                                Type.GetType("System.Int32"));
m_ds.Tables["OrderDetail"].Columns.Add("ProductCode",
                                Type.GetType("System.String"));
m_ds.Tables["OrderDetail"].Columns.Add("Quantity",
                                Type.GetType("System.Int32"));
m_ds.Tables["OrderDetail"].Columns.Add("Price",
                                Type.GetType("System.Currency"));

// Получаем объекты DataColumn из двух объектов DataTable в DataSet.
DataColumn parentCol = m_ds.Tables["Order"].Columns["OrderID"];
DataColumn childCol = m_ds.Tables["OrderDetail"].Columns["fk_OrderID"];

// Создаем и добавляем в DataSet отношение между таблицами.
m_ds.Relations.Add(new DataRelation("Order_OrderDetail",
                                    parentCol,
                                    childCol));
m_ds.Relations["Order_OrderDetail"].Nested = true;
```

Давайте отметим некоторые важные моменты в этом блоке кода. Создав с помощью оператора new экземпляр объекта DataSet, мы добавляем несколько таблиц с помощью метода Add объекта Tables. Аналогичные действия выполняются при добавлении столбцов в каждую коллекцию Columns объектов Tables. К любым из добавленных таблиц или столбцов можно позднее обратиться по имени. Чтобы назначить первичный ключ для таблицы Order, мы создаем массив DataColumn, содержащий одно или более полей, являющихся простым или составным ключом. В нашем случае имеется только одно ключевое поле, `OrderID`. Свойству таблицы PrimaryKey мы присваиваем массив ключевых столбцов. Для создания отношения между двумя таблицами мы сначала создаем объект DataRelation с именем `Order_OrderDetail`, содержащий два связанных столбца двух таблиц, а затем добавляем его в коллекцию Relations объекта DataSet. Последнее предложение указывает на наше желание представить отношение между таблицами Order и OrderDetail в виде вложенной структуры, что упрощает работу с этими сущностями в XML.

Следующий блок кода C# показывает, как вставить данные в каждую из двух таблиц:

```
DataRow newRow;
newRow = m_ds.Tables["Order"].NewRow();
newRow["OrderID"] = 101;
newRow["CustomerFirstName"] = "John";
newRow["CustomerLastName"] = "Doe";
newRow["Date"] = new DateTime(2001, 5, 1);
m_ds.Tables["Order"].Rows.Add(newRow);
newRow = m_ds.Tables["Order"].NewRow();
newRow["OrderID"] = 102;
newRow["CustomerFirstName"] = "Jane";
newRow["CustomerLastName"] = "Doe";
newRow["Date"] = new DateTime(2001, 4, 29);
m_ds.Tables["Order"].Rows.Add(newRow);

newRow = m_ds.Tables["OrderDetail"].NewRow();
newRow["fk_OrderID"] = 101;
newRow["ProductCode"] = "Item-100";
newRow["Quantity"] = 7;
newRow["Price"] = "59.95";
m_ds.Tables["OrderDetail"].Rows.Add(newRow);

newRow = m_ds.Tables["OrderDetail"].NewRow();
newRow["fk_OrderID"] = 101;
newRow["ProductCode"] = "Item-200";
newRow["Quantity"] = 1;
newRow["Price"] = "9.25";
m_ds.Tables["OrderDetail"].Rows.Add(newRow);

newRow = m_ds.Tables["OrderDetail"].NewRow();
newRow["fk_OrderID"] = 102;
newRow["ProductCode"] = "Item-200";
newRow["Quantity"] = 3;
newRow["Price"] = "9.25";
m_ds.Tables["OrderDetail"].Rows.Add(newRow);
```

Tables и Relations – это важные свойства объекта DataSet. Они не только описывают структуру базы данных в памяти, но и хранят содержимое DataSet в объектах DataTables внутри коллекции.

XML и наборы таблиц

Итак, у нас есть объект DataSet, заполненный таблицами и отношениями, давайте посмотрим, как он поможет в организации взаимодействия между компонентами. Ответом будет XML. DataSet имеет несколько методов, тесно интегрирующих DataSet с XML, делая его, таким образом, подходящим для универсального межплатформенного взаимодействия. Эти методы включают WriteXml(), WriteXmlSchema(), ReadXml() и ReadXmlSchema().

Метод WriteXmlSchema() выводит только схему таблиц, включая все таблицы и отношения между ними. WriteXml() может выводить как схему, так и табличные данные в виде строки в формате XML. И WriteXmlSchema() и WriteXml() принимают в качестве аргумента объект Stream, TextWriter, XmlWriter или строку, обозначающую имя файла. WriteXml() в качестве второго аргумента принимает объект XmlWriteMode, что позволяет при необходимости вместе с данными записать и схему.

По умолчанию WriteXml() пишет только данные. Чтобы также записать и схему, необходимо передать в качестве второго аргумента XmlWriteMode.WriteSchema. Можно также преобразовать в XML только данные, если явно использовать свойство XmlWriteMode.IgnoreSchema. Кроме того, можно задать режим XmlWriteMode.DiffGram. В этом режиме из DataSet будут выведены как исходные, так и измененные данные. Мы вернемся к этой теме, когда будем говорить о методе GetChanges().

Объект DataSet также предоставляет методы по восстановлению себя из XML-документа. Для чтения XML-документов с данными предназначен метод ReadXmlData(), для чтения XML-документов со схемами – ReadXmlSchema().

Следующий код создает XML-документ из ранее созданного набора данных:

```
// Выводим ранее показанный DataSet на консоль (а также в XML-файл)
m_ds.WriteXml(Console.Out, XmlWriteMode.WriteSchema);
m_ds.WriteXml("DS_Orders.xml", XmlWriteMode.WriteSchema);

// Создаем новый объект DataSet
DataSet ds2 = new DataSet("RestoredDS");
ds2.ReadXml("DS_Orders.xml");
```

Теперь посмотрим на получившийся XML-файл и на то, как в нем представлены наши данные:

```
<?xml version="1.0" standalone="yes"?>
<DynamicDS>
  <xs:schema id="DynamicDS"
             xmlns=""
             xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="DynamicDS" msdata:IsDataSet="true">

      <xs:complexType>
        <xs:choice maxOccurs="unbounded">

          <xs:element name="Order">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="OrderID"
                            type="xs:int" />
```

```
                <xs:element name="CustomerFirstName"
                            type="xs:string" minOccurs="0" />
                <xs:element name="CustomerLastName"
                            type="xs:string" minOccurs="0" />
                <xs:element name="Date"
                            type="xs:dateTime" minOccurs="0" />

                <xs:element name="OrderDetail"
                            minOccurs="0" maxOccurs="unbounded">

                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="fk_OrderID"
                                  type="xs:int" minOccurs="0" />
                      <xs:element name="ProductCode"
                                  type="xs:string" minOccurs="0" />
                      <xs:element name="Quantity"
                                  type="xs:int" minOccurs="0" />
                      <xs:element name="Price"
                                  msdata:DataType="System.Currency,
                                  mscorlib, Version=1.0.3300.0,
                                  Culture=neutral,
                                  PublicKeyToken=b77a5c561934e089"
                                  type="xs:string" minOccurs="0" />
                    </xs:sequence>
                  </xs:complexType>

                </xs:element>

              </xs:sequence>
            </xs:complexType>
          </xs:element>

        </xs:choice>
      </xs:complexType>

      <xs:unique name="Constraint1"
                 msdata:PrimaryKey="true">
        <xs:selector xpath=".//Order" />
        <xs:field xpath="OrderID" />
      </xs:unique>

      <xs:keyref name="Order_OrderDetail"
                 refer="Constraint1"
                 msdata:IsNested="true">
        <xs:selector xpath=".//OrderDetail" />
        <xs:field xpath="fk_OrderID" />
      </xs:keyref>

    </xs:element>
  </xs:schema>

  <... Раздел данных ...>

</DynamicDS>
```

Имя корневого элемента – DynamicDS, потому что таким было имя ранее созданного нами набора данных. Тег xsd:schema содержит определения всех таблиц и отношений набора данных DynamicDS. Поскольку мы указали, что отношение должно быть вложенным, схема показывает xsd:element для таблицы OrderDetail вложенным в xsd:element таблицы Order. Все столбцы также представлены в виде элементов xsd:element.

После определений таблиц в документе находятся определения ключей различных типов. Элемент xsd:unique используется вместе с msdata:PrimaryKey для ключей, как показано в элементе xsd:unique с именем Constraint1. Атрибут msdata:PrimaryKey обозначает первичный ключ, имеющий побочный эффект обеспечения уникальности (каждое значение OrderID в таблице должно быть уникальным).

Элемент xsd:keyref используется для внешних ключей, как показано в ключе Order_OrderDetail, ссылающемся на ключ Constraint1. Так объединяются строки таблиц OrderDetail и Order, для которых OrderDetail.fk_OrderID = Order.OrderID.

Теперь давайте посмотрим на часть XML-файла, содержащую данные:

```
<Order>
  <OrderID>101</OrderID>
  <CustomerFirstName>John</CustomerFirstName>
  <CustomerLastName>Doe</CustomerLastName>
  <Date>2001-05-01T00:00:00.0000000-04:00</Date>
  <OrderDetail>
    <fk_OrderID>101</fk_OrderID>
    <ProductCode>Item-100</ProductCode>
    <Quantity>7</Quantity>
    <Price>59.95</Price>
  </OrderDetail>
  <OrderDetail>
    <fk_OrderID>101</fk_OrderID>
    <ProductCode>Item-200</ProductCode>
    <Quantity>1</Quantity>
    <Price>9.25</Price>
  </OrderDetail>
</Order>
<Order>
  <OrderID>102</OrderID>
  <CustomerFirstName>Jane</CustomerFirstName>
  <CustomerLastName>Doe</CustomerLastName>
  <Date>2001-04-29T00:00:00.0000000-04:00</Date>
  <OrderDetail>
    <fk_OrderID>102</fk_OrderID>
    <ProductCode>Item-200</ProductCode>
    <Quantity>3</Quantity>
    <Price>9.25</Price>
  </OrderDetail>
</Order>
```

Эта часть XML-документа практически не требует пояснений. Для каждой строки данных таблицы Order мы получаем одну запись типа Order. То же самое для таблицы OrderDetail. Строка OrderDetail, связанная с определенной записью Order, вкладывается внутрь элемента Order.

Так как набор данных, по существу, отсоединен от своего источника, изменения в наборе данных должны отслеживаться им самим. Это делается с помощью методов HasChanges(), GetChanges() и Merge(). Приложение может внести изменения в набор данных и затем попросить объект DataAdapter синхронизировать изменения с источником данных посредством метода Update.

Следующий блок кода демонстрирует, как отслеживать и управлять изменениями в объекте DataSet:

```
m_ds.AcceptChanges();
/* Вносим изменения в набор данных */
m_ds.Tables["OrderDetail"].Rows[0]["Quantity"] = 12;

if(m_ds.HasChanges()){

  /* Получаем копию набора данных, содержащих внесенные  изменения. */
  DataSet changeDS = m_ds.GetChanges();

  /* Выводим измененные строки. */
  changeDS.WriteXml("ChangedDS.xml" , XmlWriteMode.DiffGram);

  /* Фиксируем все изменения. */
  m_ds.AcceptChanges();

}
```

Так как мы создаем этот набор данных динамически, первый вызов метода AcceptChange() сообщает объекту DataSet о сохранении сделанных к этому моменту изменений. Зная, что DataSet должен снова начать отслеживать изменения, мы затем изменяем значение ячейки Quantity в одной из строк таблицы OrderDetail. Затем мы запрашиваем у набора данных все выполненные изменения и переносим их в другой набор данных с именем changeDS. Этот набор данных приведен в следующей распечатке XML-кода, полученной в режиме DiffGram. Заметьте, что поскольку таблица OrderDetail является дочерней по отношению к таблице Order, то изменения также включают строку из родительской таблицы.

```
<?xml version="1.0" standalone="yes"?>
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
                 xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  <DynamicDS>
    <Order diffgr:id="Order1" msdata:rowOrder="0">
      <OrderID>101</OrderID>
      <CustomerFirstName>John</CustomerFirstName>
      <CustomerLastName>Doe</CustomerLastName>
      <Date>2001-05-01T00:00:00.0000000-04:00</Date>
```

```
      <OrderDetail diffgr:id="OrderDetail1"
                   msdata:rowOrder="0" diffgr:hasChanges="modified">
        <fk_OrderID>101</fk_OrderID>
        <ProductCode>Item-100</ProductCode>
        <Quantity>12</Quantity>
        <Price>59.95</Price>
      </OrderDetail>
    </Order>
  </DynamicDS>

  <diffgr:before>
    <OrderDetail diffgr:id="OrderDetail1" msdata:rowOrder="0">
      <fk_OrderID>101</fk_OrderID>
      <ProductCode>Item-100</ProductCode>
      <Quantity>7</Quantity>
      <Price>59.95</Price>
    </OrderDetail>
  </diffgr:before>

</diffgr:diffgram>
```

Мы бы хотели особо отметить, что объект DataSet – это наиболее важная конструкция в ADO.NET. Он не привязан к исходному представлению, такому как SQL Server или Microsoft Access, и поэтому характеризуется исключительной переносимостью. Его формат данных описывается в его собственной схеме, а данные – в виде чистого XML. Набор данных самодостаточен независимо от того, как он был создан – чтением данных из SQL Server, из Microsoft Access, из внешнего XML-файла или даже был динамически сгенерирован, как мы видели в приведенном ранее примере. Это переносимая сущность, основанная на XML, без сомнения должна стать новым стандартом обмена данными.

О DataSet мы сказали достаточно. Давайте углубимся и перейдем от DataSet к DataTable.

DataTable

Объект DataTable представляет собой таблицу данных и потому содержит коллекцию объектов DataColumn в свойстве Columns и коллекцию объектов DataRow в свойстве Rows. Свойство Columns определяет структуру таблицы, а свойство Rows предоставляет доступ к реальным данным строк. Поля таблицы представлены в виде объектов DataColumn, а записи таблицы – в виде объектов DataRow. Вот простой пример, выводящий имена всех столбцов в виде строки заголовков, а затем все строки данных:

```
/* Обходим содержимое DataTable и выводим заголовки всех
 * столбцов, а также все строки данных.
 */
DataTable myTable = m_ds.Tables["OrderDetail"];
```

```
/* Выводим названия всех столбцов. */
foreach(DataColumn c in myTable.Columns) {
  Console.Write(c.ColumnName + "\t");
}
Console.WriteLine(""); // новая строка

/* Обрабатываем каждую строку. */
foreach(DataRow r in myTable.Rows) {

  /* Выводим каждый столбец. */
  foreach(DataColumn c in myTable.Columns) {
    Console.Write(r[c] + "\t");
  }
  Console.WriteLine(""); // новая строка

}
```

Вот результат работы этого кода:

```
fk_OrderID    ProductCode     Quantity   Price
101           Item-100        12         59.95
101           Item-200        1          9.25
102           Item-200        3          9.25
```

Обычно таблица имеет одно или несколько полей, выступающих в качестве первичного ключа. Эта функциональность предоставляется свойством PrimaryKey. Так как первичный ключ может содержать более одного поля, это свойство является массивом объектов DataColumn. Мы вернемся к этому фрагменту кода, чтобы рассмотреть его в другом контексте. Заметьте, что в данном примере первичный ключ состоит только из одного поля, поэтому массив имеет размер 1.

```
// Регистрируем столбец "OrderID" как первичный ключ для таблицы "Order"
DataColumn[] keys = new DataColumn[1];
keys[0] = m_ds.Tables["Order"].Columns["OrderID"];
m_ds.Tables["Order"].PrimaryKey = keys;
```

Отношения и ограничения

Отношения (*relations*) определяют, каким образом таблицы базы данных соотносятся друг с другом. Объект DataSet целиком сохраняет коллекцию отношений между таблицами в свойстве Relations; однако каждая таблица, которая участвует в отношении, также должна знать о нем. За это отвечают два свойства объекта DataTable – ChildRelations и ParentRelations. В ChildRelations перечислены все отношения, в которых эта таблица участвует как основная таблица. С другой стороны, в ParentRelations перечислены отношения, в которых таблица играет роль подчиненной. Мы предоставим дополнительную информацию на тему отношений, когда в одном из следующих разделов главы рассмотрим объект DataRelation.

Обсуждая тему таблиц и отношений, важно понять, как устанавливаются ограничения. Существует два типа ограничений, которые можно устанавливать и накладывать на данные – UniqueConstraint и Fore-

ignKeyConstraint. UniqueConstraint требует, чтобы значение поля таблицы было уникальным. ForeignKeyConstraint определяет правила отношений между таблицами. Для ForeignKeyConstraint мы можем установить правила UpdateRule и DeleteRule, определяющие, каким образом приложение должно себя вести при обновлении или удалении строки данных в родительской таблице.

В табл. 5.1 показаны значения ограничений и соответствующие им действия для правил ForeignKeyConstraint.

Таблица 5.1. Типы ограничений и их назначение

Значение	Действия
None	Ничего.
Cascade	Зависимые строки (определяемые по внешнему ключу) удаляются/обновляются при удалении/обновлении родительской строки.
SetDefault	При удалении родительской строки внешние ключи в зависимых строках устанавливаются в значение по умолчанию.
SetNull	При удалении родительской строки внешние ключи в зависимых строках устанавливаются в null.

Ограничения активизируются, только когда свойство EnforceConstraint объекта DataSet установлено в `true`.

Следующий блок кода показывает, каким образом мы изменили ограничение внешнего ключа для таблиц Order и OrderDetail, чтобы разрешить каскадное удаление:

```
m_ds.Relations["Order_OrderDetail"].ChildKeyConstraint.DeleteRule =
Rule.Cascade;

m_ds.WriteXml("DS_BeforeCascadeDelete.xml");
m_ds.Tables["Order"].Rows[0].Delete();
m_ds.WriteXml("DS_AfterCascadeDelete.xml");
```

В результате работы этого кода в наборе данных остается лишь один заказ (заказ 102), содержащий одну позицию.

DataView

Объект DataView аналогичен *представлению* (*view*) в обычном программировании баз данных. Мы можем создать различные специализированные представления для объекта DataTable, каждое из которых имеет различные порядки сортировки и критерии фильтрации. Эти представления можно использовать для обхода, поиска и редактирования отдельных записей. Эта концепция ADO.NET ближе всего к старому набору записей (Recordset) из ADO. В ADO.NET DataView служит другой полезной задаче – привязке данных к формам Windows и вебформам. Мы покажем пример использования DataView, когда будем обсуждать привязку данных в главах 7 и 8.

DataRelation

Объект DataSet, являющийся лишь коллекцией объектов DataTable, не столь полезен. Коллекция объектов, возвращаемая серверным компонентом, дает мало преимуществ по сравнению с цепочкой наборов записей в предыдущих версиях ADO. Чтобы ваше клиентское приложение могло наиболее полно использовать возвращаемые таблицы, также должны быть возвращены отношения между ними. Здесь на помощь приходит объект DataRelation.

С помощью объекта DataRelation можно определить отношения между объектами DataTable. Клиентские компоненты могут либо просматривать отдельную таблицу, либо перемещаться по иерархии таблиц с помощью этих отношений. Например, можно найти определенную строку в родительской таблице и затем обойти все зависимые строки в дочерней таблице.

DataRelation содержит имя родительской таблицы, имя дочерней таблицы, столбец родительской таблицы (первичный ключ) и столбец дочерней таблицы (внешний ключ).

В DataSet может входить произвольное число объектов DataTable и DataRelation, поэтому ADO.NET позволяет создавать значительно более гибкую среду, в которой потребители информации могут использовать данные в том виде, в котором им хочется.

Одним из примеров может быть задача отображения всей информации об определенной родительской таблице и всех зависимых строк дочерней таблицы. В родительской таблице имеется десять строк. Каждая из строк родительской таблицы имеет десять зависимых строк в дочерней таблице. Рассмотрим два подхода, позволяющих предоставить эти данные потребителю. В первом подходе мы просто используем объединение (join) в строке запроса:

```
Select
  Order.CustomerFirstName, Order.CustomerLastName, Order.OrderDate,
  OrderDetail.ProductCode, OrderDetail.Quantity, OrderDetail.Price
from
  Order, OrderDetail
    where Order.OrderID = OrderDetail.fk_OrderID
```

Результирующий набор содержит 100 строк, где в каждой группе из десяти строк дублируется информация о родительской строке.

Второй подход состоит в том, что сначала из родительской таблицы извлекается список из десяти строк:

```
Select
  Order.OrderID,
  Order.CustomerFirstName, Order.CustomerLastName, Order.OrderDate
from
  Order
```

Затем для каждой из десяти строк родительской таблицы извлекаются зависимые строки из дочерней таблицы:

```
Select
  OrderDetail.ProductCode, OrderDetail.Quantity, OrderDetail.Price
from
  OrderDetail where fk_OrderID = thisOrderID
```

Второй подход из-за отсутствия избыточных данных расходует меньше ресурсов, однако вам придется совершить 11 обходов (один для родительской таблицы и десять для каждого родителя в дочерней таблице).

Лучше всего получить родительскую таблицу, дочернюю таблицу и отношения между ними с помощью одного обхода, вообще без избыточных данных. Это одно из самых больших преимуществ объекта DataSet. Следующий блок кода демонстрирует мощь, обусловленную наличием в DataSet и таблиц и отношений:

```
/*
 * Показываем один заказ по заданному номеру.
 */
public static void DisplaySingleOrder(DataSet m_ds, int iOrderID) {

  Decimal runningTotal = 0;
  Decimal lineTotal = 0;
  Decimal dPrice = 0;
  int iQty = 0;

  DataTable oTable = m_ds.Tables["Order"];

  // Ищем заказ в таблице Order.
  DataRow oRow = oTable.Rows.Find(iOrderID);

  /* Переходим к таблице OrderDetail
   * с помощью отношения Order_Details.
   */
  DataRow[] arrRows = oRow.GetChildRows("Order_OrderDetail");

  /* Выводим информацию по заказу. */
  Console.WriteLine ("Заказ: {0}", iOrderID);
  Console.WriteLine ("Имя: {0} {1}",
                     oRow["CustomerFirstName"].ToString(),
                     oRow["CustomerLastName"].ToString());
  Console.WriteLine ("Дата: {0}", oRow["Date"].ToString());
  Console.WriteLine("---------------------------");


  /*
   * Выводим и рассчитываем итог по строке для каждой позиции
   */
  for(int i = 0; i < arrRows.Length; i++)  {

    foreach(DataColumn myColumn in m_ds.Tables["OrderDetail"].Columns)
    {
      Console.Write(arrRows[i][myColumn] + " ");
    }
```

```
        iQty = System.Int32.Parse(arrRows[i]["Quantity"].ToString());
        dPrice = System.Decimal.Parse(arrRows[i]["Price"].ToString());

        lineTotal = iQty * dPrice;
        Console.WriteLine("{0}", lineTotal);

        /* Сохраняем промежуточный итог. */
        runningTotal += lineTotal;
      }

      /* Выводим общий итог по заказу. */
      Console.WriteLine("Total: {0}", runningTotal);
    }
```

Функция DisplaySingleOrder ищет единственную строку в таблице Order по заданному номеру заказа (OrderID). Когда эта строка найдена, мы запрашиваем у нее массив зависимых строк из таблицы OrderDetail в соответствии с отношением `Order_OrderDetail`. Получив массив объектов DataRow, мы переходим к отображению всех полей строки. Мы также подсчитываем значение `lineTotal` на основе заказанного количества и цены каждой позиции заказа, а также сохраняем значение в `runningTotal` для расчета итога по всему заказу. Далее показан результат работы функции DisplaySingleOrder.

```
Заказ: 101
Имя: John Doe
Дата: 5/1/2001 12:00:00 AM
---------------------------
101 Item-100 12 59.95 719.4
101 Item-200 1 9.25 9.25
Total: 728.65
```

Управляемые поставщики

Управляемый поставщик (*managed provider*) – это термин, обозначающий группу .NET-компонентов, реализующих фиксированный набор функциональности, определенный в архитектуре ADO.NET. Таким образом обеспечивается общий интерфейс для доступа к данным. Чтобы создать собственный управляемый поставщик, мы должны предоставить собственную реализацию объектов System.Data.Common.DbDataAdapter и реализовать такие интерфейсы, как IDbCommand, IDbConnection и IDataReader. Здесь мы не создаем собственный управляемый поставщик; однако в этом разделе мы все-таки углубимся в каждый из этих классов и интерфейсов.

В большинстве случаев разработчикам нет необходимости знать, как реализованы управляемые поставщики, хотя это может увеличить их продуктивность в отношении применения ADO.NET. Для разработки своих корпоративных приложений достаточно только знать, как использовать имеющиеся в наличии управляемые поставщики. В теку-

щей версии ADO.NET компания Microsoft предоставляет два управляемых поставщика: OLE DB и SQL. Управляемый поставщик OLE DB включает объекты OleDbConnection, OleDbCommand, OleDbParameter и OleDbDataReader. К управляемому поставщику SQL Server прилагается аналогичный набор объектов, имена которых начинаются с SqlClient вместо OleDb, что иллюстрирует рис. 5.3. Реализация этого базового набора функций для управляемых поставщиков содержится в пространстве имен System.Data. Реализация управляемого поставщика OLE DB находится в пространстве имен System.Data.OleDb, управляемого поставщика SQL – в System.Data.SqlClient.

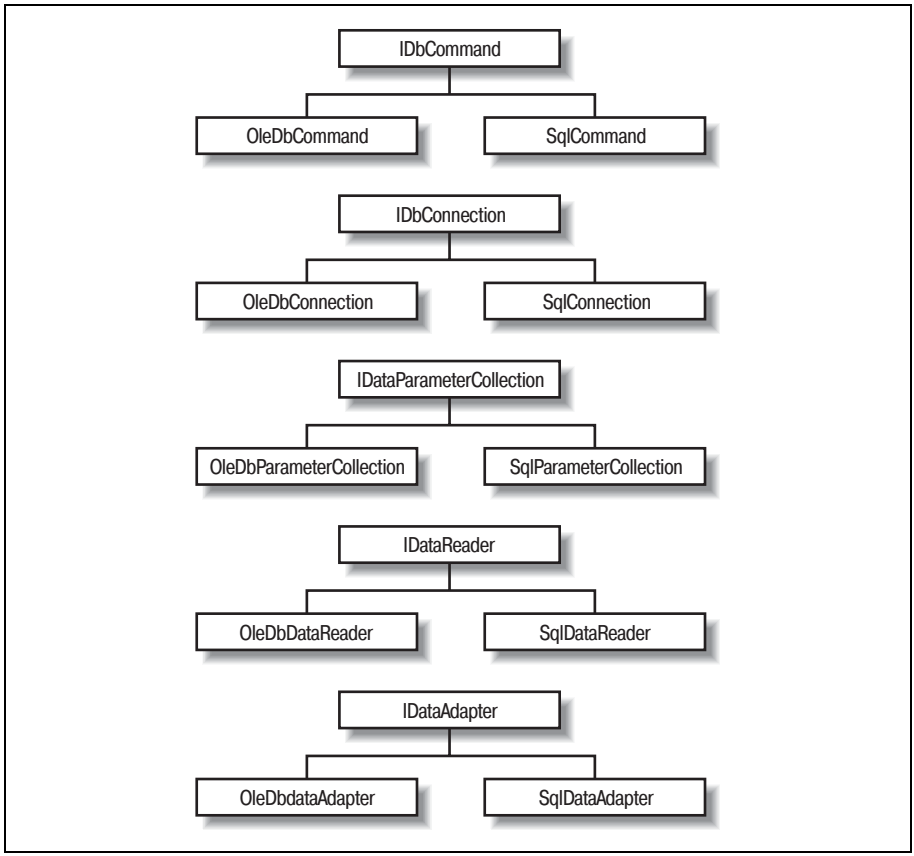

Рис. 5.3. *Иерархия классов среды управляемых поставщиков*

Оба управляемых поставщика реализуют набор интерфейсов для обращения к соответствующему хранилищу данных. Поставщик OLE DB полагается на OLE DB как на уровень доступа к широкому разнообразию источников данных, включая Microsoft SQL Server. По соображениям производительности поставщик SQL использует специальный протокол для связи с SQL Server напрямую. Независимо от способа по-

лучения данных результирующий набор остается неизменным. Это четкое разделение между управляемыми поставщиками и набором данных на основе XML помогает ADO.NET добиться переносимости данных.

На рис. 5.3 показаны базовые классы и две реализации управляемых поставщиков: OLE DB и SQL. Поскольку все управляемые поставщики, в том числе OLE DB и SQL, придерживаются фиксированного, общего набора интерфейсов (IDbCommand, IDbConnection, IDataParameterCollection, IDataReader и IDataAdapter), нетрудно адаптировать приложение к смене поставщика, если в этом возникает необходимость.

Объект Connection

И OleDbConnection и SqlConnection реализуют интерфейс System.Data.IDbConnection и, следовательно, наследуют такие свойства, как строка соединения и состояние соединения. Они реализуют базовый набор методов, определенных в IDbConnection, в том числе Open и Close.

В отличие от объекта Connection, в ADO поддержка транзакций в ADO.NET была перемещена в один из объектов Transaction (такой как OleDbTransaction и SqlTransaction). Причина состоит в том, что предположение о совпадении области действия транзакции с областью действия соединения не всегда справедливо. Например, могут существовать транзакции, охватывающие несколько соединений. Чтобы создать новую транзакцию, выполните метод BeginTransaction() объекта OleDbConnection или SqlConnection. Метод возвращает реализацию интерфейса IDbTransaction, поддерживающую такие функции транзакций, как Commit и Rollback. SqlTransaction также поддерживает сохранение контрольных точек, поэтому мы можем осуществить откат к определенной контрольной точке, а не отменять всю транзакцию. В табл. 5.2 перечислены поставщики, поддерживаемые в управляемом поставщике OLE DB.

Таблица 5.2. Управляемые поставщики ADO

Драйвер	Поставщик
SQLOLEDB	Microsoft OLE DB Provider для SQL Server
MSDAORA	Microsoft OLE DB Provider для Oracle
Microsoft.Jet.OLEDB.4.0	OLE DB Provider для Jet

И снова, если посмотрть на список методов, поддерживаемых OleDbConnection и SqlConnection, можно обнаружить, что их функциональность и фнкциональность старого объекта Connection в ADO во многом одинаковы. Однако ни OleDbConnection, ни SqlConnection больше не позволяют напрямую исполнять команды SQL или специфические для поставщика текстовые команды. Другими словами, метод Execute() больше в объекте соединения не поддерживается. Это более удачное

распределение функциональности между классами. Все выполнение осуществляется через объект Command, который обсуждается в следующем разделе вместе с описанием способа установки соединения.

Объекты Command и DataReader

К счастью для ADO-разработчиков, объекты SqlCommand и OleDbCommand в ADO.NET действуют так же, как объект Command в ADO; однако объекты Command в ADO.NET – единственный способ, с помощью которого мы можем выполнять запросы на вставку, обновление и удаление данных в ADO.NET. Это облегчает изучение объектной модели. Разработчики не столкнутся с таким разнообразием способов выполнения одного и того же действия, как в случае ADO, где можно выполнить запрос как через объект Connection, так и через Command или даже через Recordset.

Выполнение команды

Все команды связываются с объектом Connection через свойство Connection объекта SqlCommand или OleDbCommand. Объект Connection следует рассматривать как канал связи между компонентом, читающим данные, и сервером базы данных. Для выполнения команды должно быть открыто активное соединение. Объект Command также принимает параметры для исполнения хранимой процедуры на сервере. В левом верхнем углу рис. 5.5 показана связь между объектами Command, Connection и Parameters.

Существует два типа команд. Первый тип – это команда-запрос, возвращающая реализацию интерфейса IDataReader. Она реализуется с помощью метода ExecuteReader(). Команды второго типа обычно осуществляют обновление, вставку или удаление строк таблицы базы данных. Этот тип реализуется методом ExecuteNonQuery().

Одно из основных различий между объектом Command в ADO.NET и в ADO заключается в возвращаемых им данных. В ADO результатом выполнения команды-запроса является набор записей, содержащий данные в табличной форме. Однако в ADO.NET наборы записей больше не поддерживаются. Результатом выполнения команды-запроса теперь является объект DataReader (см. следующий раздел). Этот объект может быть либо OleDbDataReader для OLE DB, либо SqlDataReader для SQL Reader, либо любым классом, реализующим интерфейс DataReader для специальных случаев чтения данных. Получив подходящий объект DataReader, можно выполнить операцию Read и получить данные.

Применение объектов Command, Connection и DataReader – это низкоуровневый, прямой способ работы с управляемым поставщиком. Как вы узнаете немного позже, DataAdapter инкапсулирует все эти низкоуровневые операции как более прямой способ извлечения информации из источника данных в отсоединенный набор данных.

Объект DataReader

DataReader – это совсем новая, но вполне понятная для ADO-разработчиков концепция. DataReader аналогичен объекту-потоку в объектно-ориентированном программировании (ООП). Если вам требуется доступ к записям в однонаправленном, последовательном порядке, используйте DataReader, потому что он очень эффективен. Поскольку это серверный курсор, то соединение с сервером открыто в течение всего процесса чтения данных. Из-за постоянно открытого соединения мы советуем применять эту возможность осмотрительно и не заставлять DataReader задерживаться дольше, чем это требуется. В противном случае это может повлиять на масштабируемость приложения.

Следующий код демонстрирует основные приемы использования OleDbConnection, OleDbCommand и OleDbDataReader. Хотя здесь применяется управляемый поставщик OLE DB, строка соединения очень похожа на ту, которую мы раньше использовали в ADO.[1]

```
using System;
using System.Data;
using System.Data.OleDb;

public class pubsdemo {

  public static void Main() {

    /* Строка соединения OLE DB */
    String sConn =
      "provider=sqloledb;server=(local);database=pubs; Integrated
Security=SSPI";

    /* SQL-оператор */
    String sSQL = "select au_fname, au_lname, phone from authors";

    /* Создаем и открываем новое соединение */
    OleDbConnection oConn = new OleDbConnection(sConn);
    oConn.Open();

    /* Создаем новую команду и выполняем SQL-оператор */
    OleDbCommand oCmd = new OleDbCommand(sSQL, oConn);
    OleDbDataReader oReader = oCmd.ExecuteReader();

    /* Ищем индексы интересующих нас столбцов. */
    int idxFirstName = oReader.GetOrdinal("au_fname");
    int idxLastName = oReader.GetOrdinal("au_lname");
    int idxPhone = oReader.GetOrdinal("phone");

   /* Извлекаем и показываем каждый столбец в соответствии с его  индексом. */
    while(oReader.Read()) {
      Console.WriteLine("{0} {1} {2}",
                        oReader.GetValue(idxFirstName),
```

[1] Кроме того, вы можете создать объект Command для текущего соединения следующим образом: `oCmd = oConn.CreateCommand();`.

```
                        oReader.GetValue(idxLastName),
                        oReader.GetValue(idxPhone));
    }

  }
}
```

В этом коде открывается соединение с локальным SQL-сервером (используем интегрированную безопасность[1]) и выполняется запрос имени, фамилии и номера телефона из таблицы authors базы данных pubs. Если в системе не установлена база данных pubs, загрузите файл *instpubs.sql* в Query Analyzer и запустите его (этот файл находится в каталоге *MSSQL\Install* на вашем компьютере). Тем, кто установил примеры VS.NET Quickstart, необходимо изменить параметры сервера в строке соединения на `server=(local)\\NetSDK`, так как при установке примеров устанавливается и экземпляр NetSDK SQL Server, включающий также пресловутую базу данных Pubs. В следующем примере для получения той же информации применяется SqlClient:

```
using System;
using System.Data;
using System.Data.SqlClient;

public class pubsdemo {

  public static void Main() {

    /* Строка соединения для SQL Server */
    String sConn = "server=(local);database=pubs;Integrated Security=SSPI";

    /* SQL-оператор */
    String sSQL = "select au_fname, au_lname, phone from authors";

    /* Создаем и открываем новое соединение */
    SqlConnection oConn = new SqlConnection(sConn);
    oConn.Open();

    /* Создаем новую команду и выполняем SQL-оператор */
    SqlCommand oCmd = new SqlCommand(sSQL, oConn);
    SqlDataReader oReader = oCmd.ExecuteReader();

    /* Ищем индексы интересующих нас столбцов */
    int idxFirstName = oReader.GetOrdinal("au_fname");
    int idxLastName = oReader.GetOrdinal("au_lname");
```

[1] Пожалуйста, имейте в виду, что организация пула соединений с базами данных основывается на уникальности строк соединения. Если вы используете интегрированную модель безопасности SQL Server, и ваш код доступа к данным работает в контекстах безопасности каждого из подключенных пользователей, эффективность пула соединений уменьшится. Эта проблема решается созданием небольшого набора учетных записей Windows; мы не обсуждаем вопросы безопасности в этой книге особенно подробно по причине ее небольшого размера.

```
        int idxPhone = oReader.GetOrdinal("phone");

        /* Извлекаем и показываем каждый столбец в соответствии с его индексом. */
        while(oReader.Read()) {
          Console.WriteLine("{0} {1} {2}",
                            oReader.GetValue(idxFirstName),
                            oReader.GetValue(idxLastName),
                            oReader.GetValue(idxPhone));
        }

      }
    }
```

Объект DataAdapter

Вместе с введением DataReader в ADO.NET также появился объект DataAdapter, выполняющий функцию моста между источником данных и отсоединенным набором данных. Он содержит соединение и несколько команд для извлечения информации из хранилища данных в одну таблицу набора данных и обновления информации в хранилище данных в соответствии с изменениями, кэшированными в наборе данных. Несмотря на то что каждому объекту DataAdapter соответствует только одна таблица в наборе данных, можно иметь несколько адаптеров для заполнения объекта DataSet с несколькими DataTable. Иерархия классов DataAdapter показана на рис. 5.4. И OleDbDataAdapter и SqlDataAdapter – производные от DbDataAdapter, который, в свою очередь, является производным от абстрактного класса DataAdapter. Этот абстрактный класс реализует интерфейс IDataAdapter, определяющий поддержку методов Fill и Update. Интерфейс IDataAdapter определяется в пространстве имен System.Data, как и сам объект DataSet.

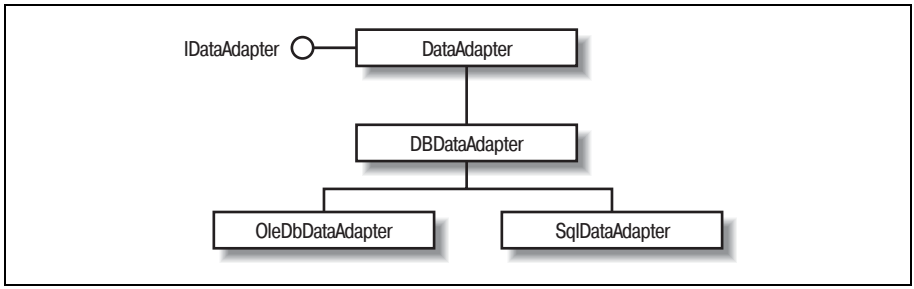

Рис. 5.4. *Иерархия класса DataSetCommand*

OleDbDataAdapter и SqlDataAdapter могут заполнять DataSet строками и обновлять источник данных при внесении изменений в набор данных. Например, OleDbAdapter может применяться для переноса данных от поставщика OLE DB в объект DataSet с помощью метода OleDbDataAdapter.Fill(). Затем можно изменить содержащиеся в объекте DataSet данные и зафиксировать внесенные изменения в базе дан-

ных с помощью метода OleDbDataAdapter.Update(). SqlDataAdapter поддерживает те же самые методы. Эти адаптеры действуют как посредники, передавая данные между сервером базы данных и отсоединенным набором данных.

DataAdapter извлекает данные при помощи SQL-команды SELECT (доступной через свойство SelectCommand). Эта команда применяется в реализации метода Fill интерфейса IDataAdapter. Обновление данных DataAdapter выполняет посредством SQL-команд UPDATE, INSERT и DELETE (доступных через свойства UpdateCommand, InsertCommand и DeleteCommand).

Наряду с методами Fill и Update класса DbDataAdapter, объекты OleDbDataAdapter и SqlDataAdapter также наследуют свойство TableMappings, коллекцию объектов TableMapping, позволяющих ставить в соответствие реальным именам столбцов базы данных пользовательские имена столбцов. Это еще больше изолирует объект DataSet от источника, из которого берутся данные. Даже именам таблиц и столбцов могут быть поставлены в соответствие более понятные имена, что облегчает использование набора данных. Разработчик приложения может больше заниматься тем, что он умеет делать лучше всего, а именно реализацией бизнес-логики, а не расшифровкой загадочных имен столбцов базы данных. На рис. 5.5 показана взаимосвязь между компонентами управляемых поставщиков.

Из четырех команд в объекте IDbDataAdapter только команда SELECT является обязательной. Остальные команды необязательны, т. к. могут генерироваться системой автоматически. Однако автоматическая генерация этих команд работает только тогда, когда соблюдаются определенные условия. Например, если объект DataAdapter заполняет набор данных из некоторого представления базы данных, включающего более одной таблицы, необходимо явно определить все четыре команды. Другой пример: когда адаптер не возвращает ключевых полей таблицы, система не сможет сгенерировать команды для вставки, обновления или удаления. Типичная схема использования DataAdapter включает следующие этапы:

- создание объекта DataAdapter (OleDbDataAdapter или SqlDataAdapter);
- задание строки запроса для внутреннего объекта SelectCommand;
- задание строки соединения для объекта Connection, связанного с объектом SelectCommand;
- задание строки запросов и соединений для InsertCommand, UpdateCommand или DeleteCommand (рекомендуется);
- вызов метода Fill() для заполнения заданного набора данных результатами выполнения запроса;
- внесение изменений и вызов метода Update() с измененным DataSet (не обязательно).

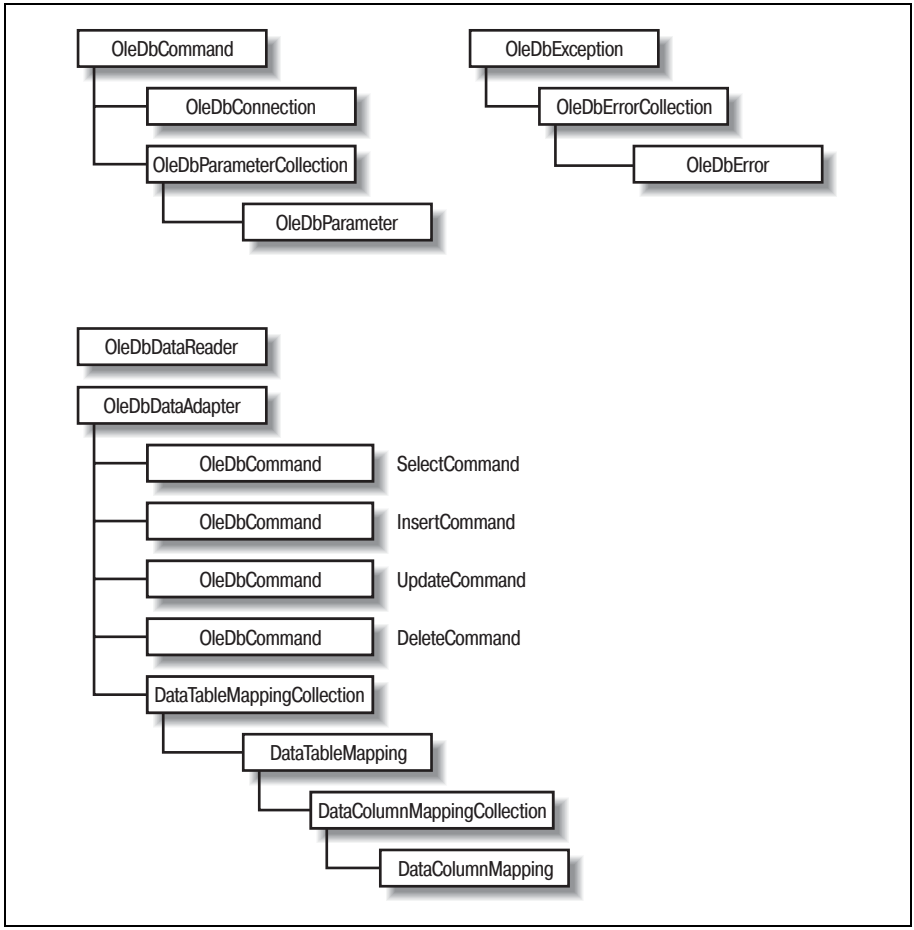

Рис. 5.5. *OleDbDataAdapter и вспомогательные классы*

Следующий фрагмент кода демонстрирует эти этапы:

```
static DataSet GenerateDS() {

  /* Создаем объект DataSet. */
  DataSet ds = new DataSet("DBDataSet");
  String sConn =
      "provider=SQLOLEDB;server=(local);database=pubs; Integrated
Security=SSPI ";

  /* Создаем адаптеры DataSet. */
  OleDbDataAdapter dsAdapter1 =
      new OleDbDataAdapter("select * from authors", sConn);

  OleDbDataAdapter dsAdapter2 =
      new OleDbDataAdapter("select * from titles", sConn);
```

```
    OleDbDataAdapter dsAdapter3 =
        new OleDbDataAdapter("select * from titleauthor", sConn);

    /* Заполняем набор данных из трех таблиц. */
    dsAdapter1.Fill(ds, "authors");
    dsAdapter2.Fill(ds, "titles");
    dsAdapter3.Fill(ds, "titleauthor");

    // Добавляем два отношения между тремя таблицами. */
    ds.Relations.Add("authors2titleauthor",
                     ds.Tables["authors"].Columns["au_id"],
                     ds.Tables["titleauthor"].Columns["au_id"]);

    ds.Relations.Add("titles2titleauthor",
                     ds.Tables["titles"].Columns["title_id"],
                     ds.Tables["titleauthor"].Columns["title_id"]);

    // Возвращаем DataSet
    return ds;

}
```

Здесь продемонстрировано создание набора данных из трех таблиц базы данных *pubs*. Набор данных также содержит два отношения, связывающие три таблицы между собой. Давайте посмотрим на набор данных в формате XML, генерируемый следующими строками кода:

```
DataSet ds = GenerateDS();
ds.WriteXml("DBDataSet.xml", XmlWriteMode.WriteSchema);
```

Содержимое файла *DBDataSet.xml* (с некоторыми пропусками для краткости) показано ниже:

```
<?xml version="1.0" standalone="yes"?>
<DBDataSet>
  <xsd:schema id="DBDataSet" targetNamespace="" xmlns=""
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="DBDataSet" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">

          <xsd:element name="authors">
            <xsd:complexType>
              <xsd:sequence>
                <!-- для краткости определения столбцов упрощены -->
                <xsd:element name="au_id" type="xsd:string" />
                <xsd:element name="au_lname" type="xsd:string" />
                <xsd:element name="au_fname" type="xsd:string" />
                <xsd:element name="phone" type="xsd:string" />
                <xsd:element name="address" type="xsd:string" />
                <xsd:element name="city" type="xsd:string" />
                <xsd:element name="state" type="xsd:string" />
                <xsd:element name="zip" type="xsd:string" />
```

```
              <xsd:element name="contract" type="xsd:boolean" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>

        <!-- для краткости titles и titleauthor опущены -->

      </xsd:choice>
    </xsd:complexType>

    <xsd:unique name="Constraint1">
      <xsd:selector xpath=".//authors" />
      <xsd:field xpath="au_id" />
    </xsd:unique>

    <xsd:unique name="titles_Constraint1"
                msdata:ConstraintName="Constraint1">
      <xsd:selector xpath=".//titles" />
      <xsd:field xpath="title_id" />
    </xsd:unique>

    <xsd:keyref name="titles2titleauthor"
                refer="titles_Constraint1">
      <xsd:selector xpath=".//titleauthor" />
      <xsd:field xpath="title_id" />
    </xsd:keyref>

    <xsd:keyref name="authors2titleauthor"
                refer="Constraint1">
      <xsd:selector xpath=".//titleauthor" />
      <xsd:field xpath="au_id" />
    </xsd:keyref>

  </xsd:element>
</xsd:schema>

<!-- Некоторые строки для краткости удалены -->

<authors>
  <au_id>899-46-2035</au_id>
  <au_lname>Ringer</au_lname>
  <au_fname>Anne</au_fname>
  <phone>801 826-0752</phone>
  <address>67 Seventh Av.</address>
  <city>Salt Lake City</city>
  <state>UT</state>
  <zip>84152</zip>
  <contract>true</contract>
</authors>

<titles>
  <title_id>PS2091</title_id>
  <title>Is Anger the Enemy?</title>
  <type>psychology  </type>
  <pub_id>0736</pub_id>
```

```
    <price>10.95</price>
    <advance>2275</advance>
    <royalty>12</royalty>
    <ytd_sales>2045</ytd_sales>
    <notes>Carefully researched study of the effects of strong
    emotions on the body. Metabolic charts included.</notes>
    <pubdate>1991-06-15T00:00:00.0000</pubdate>
  </titles>
    <title_id>MC3021</title_id>
    <title>The Gourmet Microwave</title>
    <type>mod_cook</type>
    <pub_id>0877</pub_id>
    <price>2.99</price>
    <advance>15000</advance>
    <royalty>24</royalty>
    <ytd_sales>22246</ytd_sales>
    <notes>Traditional French gourmet recipes adapted for modern
    microwave cooking.</notes>
    <pubdate>1991-06-18T00:00:00.0000</pubdate>
  </titles>

  <titleauthor>
    <au_id>899-46-2035</au_id>
    <title_id>MC3021</title_id>
    <au_ord>2</au_ord>
    <royaltyper>25</royaltyper>
  </titleauthor>
  <titleauthor>
    <au_id>899-46-2035</au_id>
    <title_id>PS2091</title_id>
    <au_ord>2</au_ord>
    <royaltyper>50</royaltyper>
  </titleauthor>

</DBDataSet>
```

Таблицы представлены в виде пар тегов `<xsd:element name="имя таблицы" >...</xsd:element>`, содержащих определения столбцов. Кроме одного элемента xsd:element для каждой таблицы имеется по одному элементу xsd:unique для каждого ключа и по одному xsd:keyref на каждое отношение. Элемент xsd:unique определяет ключ в родительской таблице отношения. Тег xsd:keyref используется для дочерних таблиц отношения. Элемент xsd:keyref играет роль внешнего ключа и ссылается на ключ в родительской таблице.

Для экономии места мы сократили часть XML, содержащую данные, до одного автора, `Anne Ringer`, и двух написанных ею книг.

Может иметься несколько разных объектов DataAdapter для заполнения одного объекта DataSet. Каждый из этих адаптеров может обращаться к совершенно разным источникам данных или серверам. Другими словами, мы можем сконструировать объект DataSet, заполнен-

ный данными, получаемыми от нескольких серверов. В предыдущем примере у нас было три объекта DataAdapter; однако все они обращались к одному серверу.

Наборы данных и XML

XML быстро набирает популярность. Корпоративные приложения используют XML как основной формат для обмена данными.

ADO.NET отходит от набора записей на базе COM и в качестве своего формата транспортировки данных применяет XML. Поскольку XML является платформо-независимым, ADO.NET расширяет охватываемую область, включая в нее все, что может кодировать и декодировать XML. Это большое преимущество по сравнению с ADO, т. к. набор записей на базе COM зависит от платформы.

XML-анализаторы

Даже с учетом того, что XML представляет собой текстовый формат, пригодный для чтения пользователем, все равно нужен какой-нибудь способ программного чтения, просмотра и изменения XML. Этим и занимаются XML-анализаторы. Есть два типа XML-анализаторов: на основе деревьев и на основе потоков. В зависимости от предъявляемых требований оба типа анализаторов могут дополнять друг друга и хорошо работать на вас.

XML-анализаторы на основе деревьев читают XML-файл (или поток) целиком для формирования дерева XML-узлов. Считайте, что XML-узлы – это ваши XML-теги:

```
<car>
  <vin>VIO0000383148374</vin>
  <make>Acura</make>
  <model>Integra</model>
  <year>1995</year>
</car>
```

После преобразования в дерево эта информация будет иметь один корневой узел: `car`; под ним будут находиться четыре узла: `vin`, `make`, `model` и `year`. Как вы и могли подозревать, если XML-поток по своей природе очень велик, применение XML-анализатора на основе дерева будет не самой лучшей идеей. Дерево будет слишком большим и займет очень много памяти.

Потоковый XML-анализатор читает XML-поток по мере его получения. SAX (Simple API for XML, простой API для XML) – это спецификация для такого типа анализаторов. Анализатор в процессе чтения данных генерирует события, уведомляя приложение о теге или тексте, которые были им только что прочитаны. Он не пытается создать полное дерево всех XML-узлов, как это делает анализатор на основе дере-

ва. Следовательно, потребляемый им объем памяти минимален. Этот вид XML-анализаторов идеален для обработки больших XML-файлов с целью поиска малой части информации. В среде .NET появился еще один потоковый XML-анализатор: XmlReader. Если SAX при чтении данных проталкивает событие в приложение, XmlReader позволяет приложению получать данные из потока.

Microsoft реализует в своих анализаторах оба типа разбора XML. Поскольку XML настолько мощное средство, компания Microsoft, как и другие лидеры индустрии, внедряет XML почти во все свои разработки. Применение включает следующие области (но не ограничено ими):

- XML+HTTP в SOAP
- XML+SQL в SQL 2000
- XML в BizTalk
- XML+DataSet в ADO.NET
- XML в Web Services и Web Services Discovery (DISCO) (см. главу 6)

В этой главе мы обсудим XML+Dataset в ADO.NET, а XML в Web Services будет рассмотрен в следующей главе. Так как XML используется везде в архитектуре .NET, мы также приведем общий обзор XML-классов.

XML-классы

Чтобы разобраться в XML-анализаторе Microsoft на базе дерева, поддерживающем объектную модель документа (Document Object Model, DOM), нам необходимо знать лишь небольшое количество объектов:

- XmlNode и производные от него классы
- XmlNodeList, коллекция XmlNode
- XmlNamedNodeMap, коллекция XmlAttribute

Рассмотрим простой пример, чтобы увидеть, как XML-узлы преобразуются в эти объекты XML DOM.

XmlNode и его производные

XmlNode – это базовый класс, представляющий один узел XML-документа. В объектной модели почти все является производным от XmlNode, в том числе XmlAttribute, XmlDocument, XmlElement и XmlText, а также и другие типы XML-узлов.

Следующий фрагмент XML демонстрирует соответствие XML-тегов типам узлов дерева DOM:

```
<books>
<book category="How To">
<title>How to drive in DC metropolitan</title>
<author>Jack Daniel</author>
```

```
<price>19.95</price>
</book>
<book category="Fiction">
<title>Bring down the fence</title>
<author>Jack Smith</author>
<price>9.95</price>
</book>
</books>
```

После анализа этого XML-потока получится дерево, изображенное на рис. 5.6. Оно содержит один корневой узел, являющийся прямым производным от XmlNode. Этот корневой узел имеет тип XmlDocument. Под корневым узлом `books` располагаются два дочерних узла, оба производных от XmlNode. В этот раз они имеют тип XmlElement. Под каждым узлом элемента `book` имеется четыре дочерних элемента. Первый дочерний элемент – `category`. Этот узел имеет тип XmlAttribute, производный от XmlNode. Следующие три дочерних элемента имеют тип XmlElement: `title`, `author` и `price`. Каждый из этих элементов имеет один дочерний элемент типа XmlText.

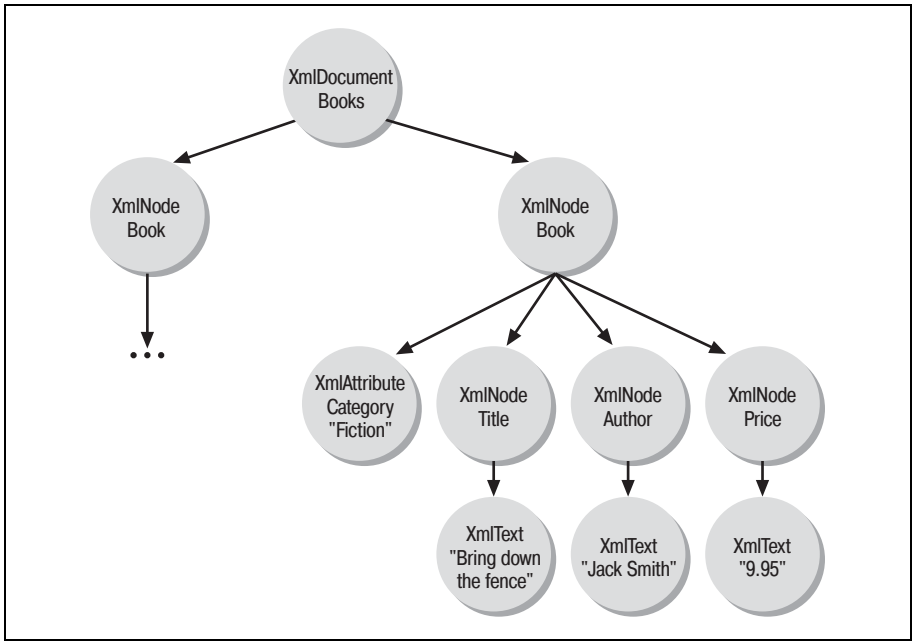

Рис. 5.6. *Древовидное представление XML-документа*

Как базовый класс XmlNode поддерживает несколько методов, помогающих в конструировании дерева XML-документа. Эти методы включают: AppendChild(), PrependChild(), InsertBefore(), InsertAfter() и Clone().

XmlNode также поддерживает группу свойств, помогающих в навигации по дереву XML-документа. Эти свойства включают: FirstChild, NextSibling, PrevousSibling, LastChild, ChildNodes и ParentNode. Свойство ChildNodes может применяться для перемещения вниз от корневого элемента. Для обхода в обратном направлении от любого из узлов предназначено свойство ParentNode.

XmlNodeList

Если XmlNode представляет один XML-элемент, то XmlNodeList представляет коллекцию из нуля или более объектов XmlNode. Свойство ChildNodes объекта XmlNode имеет тип XmlNodeList. Посмотрев на корневой узел `books`, мы увидим, что его свойство ChildNodes будет коллекцией из двух XmlNode. XmlNodeList поддерживает перечисление, и мы можем перебрать всю коллекцию и получить каждый из объектов XmlNode. Мы также можем обращаться к элементам коллекции по индексам, начиная с нуля.

Каждый объект XmlElement `book` будет иметь коллекцию ChildNodes, включающую объекты XmlElement `title`, `author` и `price`.

XmlNamedNodeMap

Аналогично XmlNodeList, XmlNamedNodeMap – это объект-коллекция. Точнее говоря, это коллекция объектов XmlAttribute, допускающих как перечисление, так и индексирование атрибутов по имени. Каждый объект XmlNode имеет свойство с именем Attributes. В случае элементов `book` эти коллекции содержат всего один атрибут – `category`.

XmlDocument

В дополнение ко всем методам и свойствам, поддерживаемым объектом XmlNode, этот производный от XmlNode класс добавляет или ограничивает методы и свойства. Здесь мы рассмотрим XmlDocument лишь как пример производного от XmlNode класса.

XmlDocument расширяет XmlNode и добавляет несколько вспомогательных функций. Эти функции используются для создания других типов XmlNode, таких как XmlAttribute, XmlComment, XmlElement и XmlText. Кроме обеспечения возможности создания XML-узлов других типов, XmlDocument также предоставляет механизм для загрузки и сохранения XML-содержимого.

Следующий код демонстрирует, как с помощью DOM программно сгенерировать XmlDocument:

```
using System;
using System.Xml;

public class XmlDemo {

  public static void Main() {
```

```
        // Код, демонстрирующий программную генерацию XmlDocument
        XmlDocument xmlDom = new XmlDocument();
        xmlDom.AppendChild(xmlDom.CreateElement("", "books", ""));
        XmlElement xmlRoot = xmlDom.DocumentElement;
        XmlElement xmlBook;
        XmlElement xmlTitle, xmlAuthor, xmlPrice;
        XmlText xmlText;

        xmlBook= xmlDom.CreateElement("", "book", "");
        xmlBook.SetAttribute("category", "", "How To");

        xmlTitle = xmlDom.CreateElement("", "title", "");
        xmlText = xmlDom.CreateTextNode("How to drive in DC metropolitan");
        xmlTitle.AppendChild(xmlText);
        xmlBook.AppendChild(xmlTitle);

        xmlAuthor = xmlDom.CreateElement("", "author", "");
        xmlText = xmlDom.CreateTextNode("Jack Daniel");
        xmlAuthor.AppendChild(xmlText);
        xmlBook.AppendChild(xmlAuthor);

        xmlPrice = xmlDom.CreateElement("", "price", "");
        xmlText = xmlDom.CreateTextNode("19.95");
        xmlPrice.AppendChild(xmlText);
        xmlBook.AppendChild(xmlPrice);

        xmlRoot.AppendChild(xmlBook);


        xmlBook= xmlDom.CreateElement("", "book", "");
        xmlBook.SetAttribute("category", "", "Fiction");

        xmlTitle = xmlDom.CreateElement("", "title", "");
        xmlText = xmlDom.CreateTextNode("Bring down the fence");
        xmlTitle.AppendChild(xmlText);
        xmlBook.AppendChild(xmlTitle);

        xmlAuthor = xmlDom.CreateElement("", "author", "");
        xmlText = xmlDom.CreateTextNode("Jack Smith");
        xmlAuthor.AppendChild(xmlText);
        xmlBook.AppendChild(xmlAuthor);

        xmlPrice = xmlDom.CreateElement("", "price", "");
        xmlText = xmlDom.CreateTextNode("9.95");
        xmlPrice.AppendChild(xmlText);
        xmlBook.AppendChild(xmlPrice);

        xmlRoot.AppendChild(xmlBook);

        Console.WriteLine(xmlDom.InnerXml);

    }

}
```

XmlDocument также поддерживает методы LoadXml и Load, формирующие полное XML-дерево из входного параметра. LoadXml принимает

в качестве аргумента строку в формате XML, а Load – имя файла, объект TextReader или XmlReader. Следующий пример продолжается с того места, на котором закончился предыдущий. Дерево XML сохраняется в файле с именем *books.xml*. Затем этот файл загружается обратно в другое XML-дерево. Это новое дерево выводится в тот же XML-поток, что и предыдущее.

```
...
xmlDom.Save("books.xml");
XmlDocument xmlDom2 = new XmlDocument();
xmlDom2.Load("books.xml");
Console.WriteLine(xmlDom2.InnerXml);
```

XmlReader

Объект XmlReader – это быстрый, некэширующий, однонаправленный способ доступа к потоковым XML-данным. Имеется два производных класса от XmlReader: XmlTextReader и XmlNodeReader. Оба этих объекта читают XML по одному тегу за раз. Единственное различие – это используемый ими источник данных. Как следует из названий, XmlTextReader читает поток чистого XML-текста, а XmlNodeReader читает поток узлов из XmlDocument. Поток может начинаться либо в начале XML-файла (если читается весь XML-документ), либо только с определенного узла (если документ читается частично).

Рассмотрим следующий фрагмент XML для обработки заказа. Если этот файл большой, неразумно читать его в XmlDocument и выполнять его обработку в объекте. Вместо этого следует читать только те узлы и атрибуты, которые нам нужны, и игнорировать остальные. Мы можем использовать для этого производные от XmlReader классы.

```
<Orders>
<Order id="ABC001" ...>
<Item code="101" qty="3" price="299.00" ...>Монитор 17"</Item>
<Item code="102" qty="1" price="15.99" ...>Клавиатура</Item>
<Item code="103" qty="2" price="395.95" ...>Системный блок</Item>
</Order>
<Order id="ABC002" ...>
<Item code="101b" qty="1" price="499.00" ...>Монитор 21"</Item>
<Item code="102" qty="1" price="15.99" ...>Клавиатура</Item>
</Order>
<...>
</Orders>
```

Следующий блок кода обходит и обрабатывает каждый заказ из большого входного файла `Order.xml`:

```
using System;
using System.IO;
using System.Xml;
```

```
class TestXMLReader
{

static void Main(string[] args)
{
    TestXMLReader tstObj = new TestXMLReader();
    StreamReader myStream = new StreamReader("Orders.xml");
    XmlTextReader xmlTxtRdr = new XmlTextReader(myStream);
    while(xmlTxtRdr.Read())
    {
        if(xmlTxtRdr.NodeType == XmlNodeType.Element
           && xmlTxtRdr.Name == "Order")
        {
            tstObj.ProcessOrder(xmlTxtRdr);
        }
    }
}

public void ProcessOrder(XmlTextReader reader)
{
    Console.WriteLine("Начинаем обработку заказа: " +
                      reader.GetAttribute("id"));
    while(!(reader.NodeType == XmlNodeType.EndElement
          && reader.Name == "Order")
          && reader.Read())
    {
        // Обрабатываем содержимое заказа
        if(reader.NodeType == XmlNodeType.Element
           && reader.Name == "Item")
        {
            Console.WriteLine("Код:" + reader.GetAttribute("code") +
                              ". Кол-во: " + reader.GetAttribute("qty"));
        }
    }
}

}
```

Посмотрим внимательнее, что происходит. После создания объекта XmlTextReader с потоком данных из строки нам останется лишь выполнять в цикле операцию Read() до тех пор, пока читать будет нечего. В процессе чтения мы начинаем обработку заказа, только если встречаем узел, имеющий тип XmlElement и имя `Order`. Внутри функции ProcessOrder мы читаем и обрабатываем все позиции в заказе, пока не встретим завершающий тег `Order`. Тогда мы возвращаемся из функции и переходим к поиску следующего тега `Order` для обработки следующего заказа.

XmlNodeReader аналогичен XmlTextReader, т. к. оба дают возможность последовательной обработки XML. Однако XmlNodeReader читает узлы XML из полного дерева XML или его фрагмента. Это значит, что XmlNodeReader не поможет вам в чтении больших XML-файлов.

XmlWriter

Объект XmlWriter – это быстрый, некэширующий способ записи потоковых XML-данных. Он также поддерживает пространства имен. Единственным производным от XmlWriter классом является XmlTextWriter.

XmlWriter поддерживает пространства имен путем предоставления нескольких перегруженных функций, которые используют пространство имен для связи с элементом. Если пространство имен уже определено и для него существует префикс, XmlWriter автоматически записывает имя элемента с этим префиксом. Почти все методы записи элементов перегруженные – для поддержки пространств имен.

Следующий код показывает, как использовать объект XmlTextWrite для создания действительного XML-файла:

```
XmlTextWriter writer =
  new XmlTextWriter("test.xml", new System.Text.ASCIIEncoding());
writer.Formatting = Formatting.Indented;
writer.Indentation = 4;
writer.WriteStartDocument();
writer.WriteComment("Comment");
writer.WriteStartElement("ElementName", "myns");
writer.WriteStartAttribute("prefix", "attrName", "myns");
writer.WriteEndAttribute();
writer.WriteElementString("ElementName", "myns", "value");
writer.WriteEndElement();
writer.WriteEndDocument();
writer.Flush();
writer.Close();
```

В результате в файле *text.xml* будет создан следующий XML-документ:

```
<?xml version="1.0" encoding="us-ascii"?>
<!--Comment-->
<ElementName prefix:attrName="" xmlns:prefix="myns" xmlns="myns">
    <prefix:ElementName>value</prefix:ElementName>
</ElementName>
```

XslTransform

XslTransform преобразует XML из одного формата в другой. Этот класс обычно применяется в программах преобразования данных или для преобразования XML в HTML для отображения XML-данных в броузере. Следующий код демонстрирует, каким образом происходит такое преобразование:

```
using System;
using System.Xml;           // XmlTextWriter
using System.Xml.Xsl;       // XslTransform
using System.Xml.XPath;     // XPathDocument
using System.IO;            // StreamReader
```

```
public class XSLDemo {
  public static void Main() {
    XslTransform xslt = new XslTransform();
    xslt.Load("XSLTemplate.xsl");
    XPathDocument xDoc = new XPathDocument("Books.xml");
    XmlTextWriter writer = new XmlTextWriter("Books.html", null);
    xslt.Transform(xDoc, null, writer);
    writer.Close();
    StreamReader stream = new StreamReader("Books.html");
    Console.Write(stream.ReadToEnd());
  }
}
```

По существу, в этом коде выполняется преобразование XML из файла *Books.xml*, который вы видели раньше, в HTML для отображения в броузере. Хотя в приведенном фрагменте вы и можете заменить XPathDocument на XmlDocument, предпочтительным в этом случае является класс XPathDocument, т. к. он оптимизирован для XSLT-преобразований.[1] На рис. 5.7 и 5.8 показаны исходный XML и HTML, полученный при просмотре в броузере.

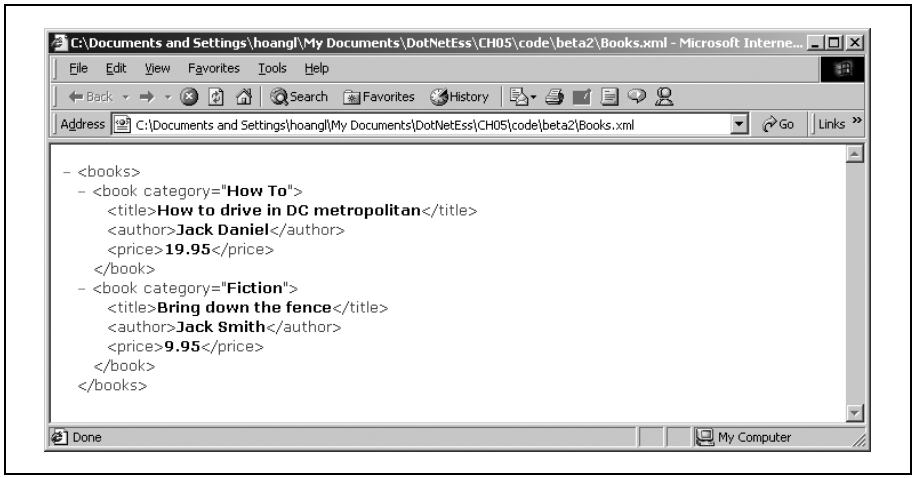

Рис. 5.7. *Books.xml, отображаемый в Internet Explorer*

Для преобразования XML был использован следующий XSL-шаблон:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "/" >
```

[1] XPathDocument загружает данные быстрее, чем XmlDocument, поскольку он не выполняет идентификацию узлов и проверку правил. Побочным эффектом этого является то, что содержимое доступно только для чтения.

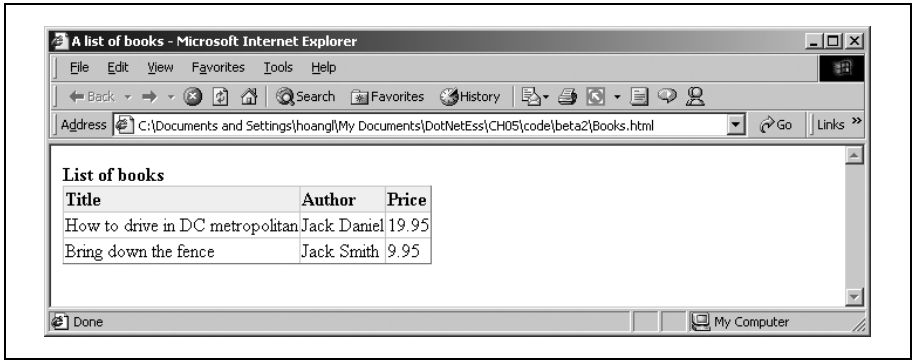

***Рис. 5.8.** Books.html, отображаемый в Internet Explorer*

```
<html>
<head><title>A list of books</title></head>
<style>
.hdr { background-color=#ffeedd; font-weight=bold; }
</style>
<body>
<B>List of books</B>
<table style="border-collapse:collapse" border="1">
<tr>
  <td class="hdr">Title</td>
  <td class="hdr">Author</td>
  <td class="hdr">Price</td>
</tr>
<xsl:for-each select="//books/book">
<tr>
  <td><xsl:value-of select="title"/></td>
  <td><xsl:value-of select="author"/></td>
  <td><xsl:value-of select="price"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>

</xsl:template>
</xsl:stylesheet>
```

XmlDataDocument

Одной из важнейших черт ADO.NET является тесная интеграция DataSet с XML. Объект DataSet может быть легко преобразован в XML и обратно, облегчая обмен данными с любыми другими компонентами в корпоративной системе. Схема набора данных может быть загружена и сохранена в виде определения XML-схемы (XML Schema Definition, XSD), как было описано выше.

XmlDataDocument может быть связан с объектом DataSet. Следующий фрагмент кода иллюстрирует, как установить эту связь:

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Xml;

class TestXMLDataDocument
{

static void Main(string[] args)
{
    TestXMLDataDocument tstObj = new TestXMLDataDocument();

    // создаем XmlDataDocument с DataSet
    XmlDataDocument doc = tstObj.GenerateXmlDataDocument();

    XmlNodeReader myXMLReader = new XmlNodeReader(doc);
    while (myXMLReader.Read())
    {
        if(myXMLReader.NodeType == XmlNodeType.Element
            && myXMLReader.Name == "Orders")
        {
            tstObj.ProcessOrder(myXMLReader);
        }
    }
}

public void ProcessOrder(XmlNodeReader reader)
{
    Console.Write("Начинаем обработку заказа: ");
    while(!(reader.NodeType == XmlNodeType.EndElement
        && reader.Name == "Orders")
        && reader.Read())
    {
        if(reader.NodeType == XmlNodeType.Element
            && reader.Name == "OrderID")
        {
            reader.Read();
            Console.WriteLine(reader.Value);
        }
        if(reader.NodeType == XmlNodeType.Element
            && reader.Name == "OrderDetails")
        {
            ProcessLine(reader);
        }
    }
}

public void ProcessLine(XmlNodeReader reader)
{
    while(!(reader.NodeType == XmlNodeType.EndElement
        && reader.Name == "OrderDetails")
```

```
        && reader.Read())
    {
        if(reader.NodeType == XmlNodeType.Element && reader.Name == "
ProductID")
        {
            reader.Read();
            Console.Write(".  Код: " + reader.Value);
        }
        if(reader.NodeType == XmlNodeType.Element && reader.Name == "
Quantity")
        {
            reader.Read();
            Console.WriteLine(".  Количество: " + reader.Value);
        }
    }
}
public XmlDataDocument GenerateXmlDataDocument()
{
    /* Создаем объект DataSet. */
    DataSet ds = new DataSet("DBDataSet");
    String sConn =
        "provider=SQLOLEDB;server=(local);database=NorthWind;Integrated
Security=SSPI";

    /* Создаем адаптеры для DataSet. */
    OleDbDataAdapter dsAdapter1 =
        new OleDbDataAdapter("select * from Orders", sConn);

    OleDbDataAdapter dsAdapter2 =
        new OleDbDataAdapter("select * from [Order Details]", sConn);

    /* Заполняем набор данных из трех таблиц. */
    dsAdapter1.Fill(ds, "Orders");
    dsAdapter2.Fill(ds, "OrderDetails");

    DataColumn[] keys = new DataColumn[1];
    keys[0] = ds.Tables["Orders"].Columns["OrderID"];
    ds.Tables["Orders"].PrimaryKey = keys;


    /* Добавляем два отношения между тремя таблицами. */
    ds.Relations.Add("Orders_OrderDetails",
        ds.Tables["Orders"].Columns["OrderID"],
        ds.Tables["OrderDetails"].Columns["OrderID"]);

    ds.Relations["Orders_OrderDetails"].Nested = true;
    //ds.WriteXml("NorthWindOrders.xml");

    return new XmlDataDocument(ds);

}

}
```

В предыдущем разделе в описании объекта DataSet было показано, что при наличии набора данных можно сохранить данные из него в XML-строке или в файле. В этот раз мы продемонстрировали, как преобразовать объект DataSet в XmlDataDocument, с которым можно выполнять манипуляции в памяти.

Заключение

В данной главе описано ядро ADO.NET. Ориентируясь на отсоединенные наборы данных, ADO.NET позволяет не только построить высокопроизводительные, масштабируемые решения для электронной коммерции, но и позволяет приложению охватывать другие платформы за счет использования XML. Эта глава представляет собой общий обзор классов, составляющих ADO.NET, и служит для того, чтобы познакомить вас с библиотекой System.Xml. В следующей главе мы углубимся в построение программного обеспечения в виде служб. Мы воспользуемся ADO.NET как механизмом доступа и обмена данными в наших программных службах.