


[350元等于什么？答案就是等于Microsoft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！ 视频包括浪曦 本杰 中美IT 等290 GB视频教程 需要的请联系QQ：2569343569](#)

[答案就是等于Microsoft .NET技术！](#)

[等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！！！](#)

[视频包括浪曦 本杰 中美IT 等290 GB视频教程 课程目录请百度一下 2569343569 新浪博客](#)

[包你学会.NET 需要的请联系QQ：2569343569](#) 



Beginning ASP.NET 4 in C# and VB

ASP.NET 4入门经典

——涵盖C#和VB.NET(第6版)

(美) Imar Spaanjaars 著
刘伟琴 张格仙 译



清华大学出版社

使用最新版ASP.NET构建内容丰富的Web站点

ASP.NET是.NET Framework的一部分，用于构建内容丰富的动态Web站点。其最新的版本ASP.NET 4对上一版进行了许多改进，包括增强了Web 窗体，并添加了对jQuery的支持。本书循序渐进，逐步讲解了如何使用ASP.NET 4构建内容丰富的Web站点，并提供了大量使用C#和VB的示例。通过实际动手练习，您将学到关于构建Web站点的第一手信息，同时能够深刻理解在浏览器中查看ASP.NET 4页面时，后台到底发生了什么。

本书主要内容

- ◆ 演示了如何构建ASP.NET 4 Web页面
- ◆ 解释了如何使用预置服务器控件添加功能
- ◆ 讨论了如何使用各种开发工具构建ASP.NET Web站点
- ◆ 分享了创建外观风格一致的Web站点技术
- ◆ 讲解了如何在自己的Web站点内实现jQuery和AJAX功能
- ◆ 示范了如何使用数据库和Microsoft ADO.NET Entity Framework
- ◆ 探讨了如何保护和个性化站点
- ◆ 研究了异常处理以及如何调试和跟踪页面

Imar Spaanjaars是一名Microsoft ASP.NET MVP，在荷兰经营着一家名为De Vier Koeden的公司，专门使用ASP.NET 4等Microsoft的技术编写因特网和内部网应用程序。他曾撰著或与他人合著了多本书籍，包括*ASP.NET 2.0 Instant Results*和《ASP.NET 3.5入门经典——涵盖C#和VB.NET(第5版)》，并且是Wrox社区论坛p2p.wrox.com上的主要贡献者之一。

Wrox Beginning guides are crafted to make learning programming languages and technologies easier than you think, providing a structured, tutorial format that will guide you through all the techniques involved.

源代码下载及技术支持

<http://www.wrox.com>

<http://www.tupwk.com.cn/download>

Wrox
An Imprint of
WILEY

上架建议：Web开发/ASP.NET
读者信箱：wkservice@vip.163.com
投稿邮箱：bookservice@263.net



Wrox.com

Programmer Forums

Join our Programmer to Programmer forums to ask and answer programming questions about this book, join discussions on the hottest topics in the industry, and connect with fellow programmers from around the world.

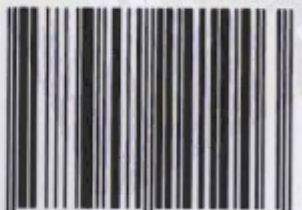
Code Downloads

Take advantage of free code samples from this book, as well as code samples from hundreds of other books, all ready to use.

Read More

Find articles, ebooks, sample chapters and tables of contents for hundreds of books, and more reference resources on programming topics that matter to you.

ISBN 978-7-302-24100-3



9 787302 241003 >

定价：88.00元

ASP.NET 4 入门经典

——涵盖 C# 和 VB.NET

(第 6 版)

清华大学出版社

北 京

Imar Spaanjaars
Beginning ASP.NET 4 in C# and VB
EISBN: 978-0-470-50221-1
Copyright © 2009 by Wiley Publishing, Inc.
All Rights Reserved. This translation published under license.

本书中文简体字版由 Wiley Publishing, Inc. 授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字：01-2010-2256

本书封面贴有 Wiley 公司防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据
ASP.NET 4 入门经典——涵盖 C# 和 VB.NET(第 6 版)/(美) 史潘加斯(Spaanjaars,I.) 著；刘伟琴，张格仙 译.
—北京：清华大学出版社，2010.12
书名原文：Beginning ASP.NET 4 in C# and VB
ISBN 978-7-302-24100-3
I. A… II. ①史… ②刘… ③张… III. 主页制作—程序设计 IV. TP393.092
中国版本图书馆 CIP 数据核字(2010)第 207129 号

责任编辑：王 军 吴 乐
装帧设计：孔祥丰
责任校对：胡雁翎
责任印制：何 芊
出版发行：清华大学出版社
地 址：北京清华大学学研大厦 A 座
http://www.tup.com.cn 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：三河市金元印装有限公司
经 销：全国新华书店
开 本：185×260 印 张：44 字 数：1181 千字
版 次：2010 年 12 月第 1 版 印 次：2010 年 12 月第 1 次印刷
印 数：1~4000
定 价：88.00 元

产品编号：035931-01

作者简介

Imar Spaanjaars 毕业于荷兰 Leisure Management School，主修休闲管理专业，但不久就转入互联网领域。

在超过 12 年的职业生涯中，Imar 曾就职于互联网领域的多家网络公司。最近，他创办了自己的公司 De Vier Koeden(www.devierkoeden.nl)，为客户提供使用 ASP.NET 4 等 Microsoft 开发的技术开发互联网和内部网应用程序方面的咨询和开发等服务。

Imar 编写了多本关于 ASP.NET 和 Macromedia Dreamweaver 的书籍，包括《ASP.NET 3.5 入门经典——涵盖 C# 和 VB.NET(第 5 版)》(由清华大学出版社引进并出版)。他是 Wrox 社区论坛(p2p.wrox.com)的主要贡献者之一，在那里他与其他程序员分享自己的知识。

在 2008 年和 2009 年，鉴于 Imar 对 ASP.NET 社区做出的突出贡献，Microsoft 公司授予他“最有价值专家”称号。

Imar 和他的女朋友 Fleur 居住在荷兰乌德勒支。可以通过他的个人网站 <http://imar.spaanjaars.com> 或者电子邮件 imar@spaanjaars.com 联系他。

序

这是一本有关 ASP.NET 技术的优秀书籍，它的作者是 Microsoft 的“最有价值专家”Imar Spaanjaars。要成为 Microsoft 的“最有价值专家”并不简单，不但需要具有相关领域的专业技能，还必须是一名优秀的老师，并且乐于分享自己的知识。借助博客、文章和书籍等载体，Imar 投入了大量时间来帮助 Web 开发社区。

Imar 在 ASP.NET 社区活跃了很久，并且已经编写了多本关于 Web 开发的书，包括本书的上一版：《ASP.NET 3.5 入门经典——涵盖 C# 和 VB.NET(第 5 版)》(由清华大学出版社引进并出版)。不同于其他书籍，本书没有假定读者已经具有 Web 开发的知识。但与此同时，对于想要升级到 ASP.NET 4 的读者来说，本书也可以提供极大的帮助。书中使用的是 Visual Studio 的免费版本 Visual Web Developer Express 2010，首先用一章的内容介绍如何完成准备工作，然后逐渐引入各个高级概念。

在帮助创建 Visual Studio 2010 for Web Developers 的过程中，我经常与社区领导者交流，以便收集用户反馈，从而能够为每位 Web 开发人员改进这个产品。从 ASP.NET 4 和 VWD 2010 Express 的第一个 Beta 版本开始，Imar 就在使用它们，并给我们提供了大量反馈信息，告诉我们哪些 bug 必须修复，哪些功能应该突出。从这本书里就可以看到他对这种技术的热爱，我相信本书的读者也一定可以感受到这一点。

ASP.NET 4 和 Visual Studio 2010 改进了许多功能，包括构建符合标准的 Web 站点、JScript IntelliSense、jQuery 集成、Ajax、CSS 改进、HTML 和标记代码段、Web 部署和数据集成。我相信，ASP.NET 4 和 Visual Studio 2010 是构建出色 Web 站点的优秀工具，而这本书则能够帮助用户在 Web 开发中游刃有余。

——Vishal R. Joshi
Microsoft Web 平台&工具高级项目经理

前言

为了构建有效且有吸引力的数据库驱动的 Web 站点，需要两个条件：一个是运行 Web 页面的稳固而快速的框架，另一个是创建和编写 Web 页面的丰富而广泛的环境。通过 ASP.NET 4 和 Visual Web Developer 2010 可以满足这两个条件。它们结合在一起形成了一个构建动态的、交互式的 Web 站点的平台。

ASP.NET 4 建立在它广受欢迎的前身 ASP.NET 2.0 和 ASP.NET 3.5 基础之上。除了维持对使用老版本构建的 Web 站点的向后兼容性外，ASP.NET 4 和 Visual Web Developer 2010 还增加了大量新的、激动人心的功能，并对框架和开发工具进行了一些较小的但是很有用的改进。

自 Visual Studio 2003 以来，对于每一个 Visual Studio 新版本(其中包括 Visual Web Developer)的发布，我都惊讶于 Microsoft 在产品中添加的新功能的数量和对产品所做的改进。Visual Studio 2010 也不例外，它的一个主要特性就是对 ADO.NET Entity Framework 4 的完整集成使得几乎不用编写代码就可以访问数据库。Visual Studio 的另一个改变是在 User Interface 中对 Windows Presentation Foundation(WPF)的使用，这除了能带来更好的用户体验外，还为 Visual Studio 自身添加了一个新行为。

尽管对 Visual Studio 自身来说这并不是一个新功能，但是在 Visual Web Developer 中引入 jQuery 是一个很不错的决定，这使得开发人员可以在更少的时间里编写更好的 Web 站点。jQuery 是一个引人注目的客户端的、跨浏览器的 JavaScript 框架(详细内容见第 11 章)。

熟悉 ASP.NET 早期版本的读者会很高兴地发现，新版本中的许多小功能将大大简化开发工作。我们将在本书的适当位置介绍并讨论这些新功能。有关 ASP.NET 新功能的完整列表，请查看 ASP.NET 官方网站(<http://www.asp.net/learn/whitepapers/aspnet4/>)上的白皮书。

关于 Visual Web Developer 2010 最好的消息可能是它的价格：现在仍然可以免费使用。尽管 Visual Studio 2010 的商业版中包括 Visual Web Developer，但是也可以下载并安装免费的 Express Edition。因此，Visual Web Developer 2010 和 ASP.NET 4 可能是如今最引人注目的 Web 开发技术。

0.1 本书读者对象

本书适用于想了解如何在 Microsoft 平台上构建丰富的交互式 Web 站点的任何人。利用从本书学到的知识，可以为构建各种类型的 Web 站点(从简单的只是业余爱好的 Web 站点，到为商业目的构建的站点)打下基础。

Web 编程新手也能使用本书，因为本书并没有事先假定读者具有 Web 开发背景(虽然了解基本的 HTML 和 Web 概念确实很有用)。本书从头开始介绍 Web 开发，说明了如何获得与

安装 Visual Web Developer。后面的章节建立在前面章节的基础上，循序渐进地介绍新技术。

您是否更喜欢 Visual Basic 而不是 C#；或者反过来，更喜欢 C#而不喜欢 Visual Basic？或者认为这两种语言都不错？或者还没有决定要学哪种语言，想两种都学？无论是哪种情况，您都会喜欢本书，因为本书的所有代码示例都是用这两种语言表示的。

即使已经熟悉了 ASP.NET 以前的版本，还是可以从本书中获益。虽然 ASP.NET 4 沿用了以前版本中的很多概念，但是在本书中还是可以发现大量新内容，包括 ADO.NET Entity Framework、jQuery、ASP.NET AJAX 的引入，以及对 ASP.NET 4 框架的许多修改等。

0.2 本书主要内容

本书将介绍如何构建一个名为 Planet Wrox 的功能丰富的、数据驱动的交互式 Web 站点。虽然这句话很长，但是你会发现使用 Visual Web Developer 2010 来开发这样的 Web 站点并不像看起来那样困难。本书将介绍构建 Web 站点的整个过程，从第 1 章的安装 Visual Web Developer 2010，一直到第 19 章将 Web 站点部署到真正的服务器上。本书分为 19 章，每一章重点介绍一个特定主题。

- 第 1 章“ASP.NET 4 入门”。本章介绍如何获得并安装 Visual Web Developer 2010。该章将说明如何下载与安装 Visual Web Developer 2010 的免费版本，即 Express Edition，还介绍了 HTML——每个 Web 页面所涉及的语言。最后概述了 Visual Web Developer 提供的自定义选项。
- 第 2 章“构建 ASP.NET Web 站点”。本章介绍如何构建一个新的 Web 站点，以及如何向它增加新元素，如页面。除了如何构建结构良好的站点外，还会介绍如何用 Visual Web Developer 中的大量工具来创建 HTML 与 ASP.NET 页面。
- 第 3 章“设计 Web 页面”。Visual Web Developer 附带了大量的工具用于创建设计良好的、有吸引力的 Web 页面。本章将说明如何充分利用这些工具。此外，还会介绍 CSS 这种用来定义 Web 页面格式的语言。
- 第 4 章“使用 ASP.NET 服务器控件”。ASP.NET 服务器控件是 ASP.NET 中最重要的概念之一，它们允许使用少量代码构建复杂而功能丰富的 Web 站点。本章介绍了大量可用的服务器控件，解释了它们的用途，并说明了它们的用法。
- 第 5 章“ASP.NET Web 页面编程”。虽然内置 CSS 工具与 ASP.NET 服务器控件非常有助于创建 Web 页面，但是使用编程语言能够增强页面。本章花了大量篇幅介绍 Web 页面编程。值得一提的是，本章(以及本书余下章节)的所有示例都使用 Visual Basic 和 C#两种语言表述，因此可以选择一种最喜欢的语言。
- 第 6 章“创建外观一致的 Web 站点”。一致性比较容易使 Web 站点具有吸引力且给人比较专业的印象。ASP.NET 通过使用母版页来帮助创建外观一致的页面。母版页可以用来定义页面的全局外观。外观(skin)和主题有助于集中控件和 Web 站点中其他可视化元素的外观。本章还会介绍如何创建基页来帮助集中站点中所有页面都需要的编程代码。

- 第7章“导航”。为了帮助访问者在站点中找到浏览路径，ASP.NET 配置了一些导航控件。这些控件用来构建站点的导航结构。可以将它们连接到站点的集中站点地图(定义 Web 站点中的页面)。本章还会介绍如何通过编写程序将用户从一个页面发送到另一个页面。
- 第8章“用户控件”。用户控件是可用在多个 Web 页面中的可重用页面片段。因此它们对于一些重复内容(如菜单、横幅等)很有用。本章将介绍如何创建与使用用户控件，并用一些程序化的智能来增强它们。
- 第9章“验证用户输入有效性”。站点中的大部分交互性是通过用户的输入定义的。本章介绍如何使用 ASP.NET 服务器控件接收、验证和处理用户输入。此外，还将介绍如何从 ASP.NET Web 站点中发送电子邮件，以及如何从正文中读信息。
- 第10章“ASP.NET AJAX”。Microsoft ASP.NET AJAX 允许创建漂亮、无闪烁的 Web 页面，消除了传统桌面应用程序与 Web 应用程序之间的差距。本章将介绍如何用内置的 Ajax 功能增强 Web 页面的表现，从而获得与 Web 站点更平滑的交互。
- 第11章“jQuery”。jQuery 是一个流行的、开源的且跨浏览器的 JavaScript 库，专用于简化与客户端浏览器的 Web 页面交互。本章介绍了 jQuery 的基础知识以及如何给 Web 页面添加丰富的可视化效果和动画。
- 第12章“初识数据库”。了解如何使用数据库对于构建 Web 站点是至关重要的，因为大多数现代 Web 站点都要求使用数据库。本章将介绍 SQL(访问和更改数据库中数据的查询语言)的基础知识。此外，还将介绍 Visual Web Developer 中帮助创建和管理 SQL Server 数据库的数据库工具。
- 第13章“显示和更新数据”。本章建立在从第12章学到的知识的基础上，说明了如何使用 ASP.NET 数据绑定控件与数据源控件创建一个丰富的界面，使用户能与这些控件的目标数据库中的数据交互。
- 第14章“LINQ 和 ADO.NET Entity Framework”。LINQ 是 Microsoft 的解决方案，用来访问对象、数据库、XML 等。ADO.NET Entity Framework(EF)是 Microsoft 数据库访问的新技术。本章将介绍 LINQ 的概念，如何使用内置在 Visual Studio 中的可视化 EF 设计器，以及如何编写 LINQ to EF 查询来让数据进出 SQL Server 数据库。
- 第15章“处理数据——高级主题”。前面的章节大多集中于处理数据的技术基础，而本章从前端角度来看同样的主题。本章将介绍如何使用控件样式来改变数据的可视化外观。本章还将介绍与数据绑定控件的交互，以及如何通过保持经常访问的数据的本地副本来加速 Web 站点。
- 第16章“ASP.NET 4 Web 站点中的安全性”。虽然安全性在本书中提出得相当晚，但是安全性是最首要且重要的主题。本章将介绍如何使用与安全性相关的内置 ASP.NET 功能。介绍若干促进安全性的应用程序服务。还将介绍如何让用户在 Web 站点上注册一个账户，如何区分匿名与登录用户，以及如何管理系统中的用户。
- 第17章“个性化 Web 站点”。本章建立在第16章介绍的安全性功能基础之上，介绍了如何用针对个人用户的内容创建个性化 Web 页面。本章将介绍如何配置与使用 ASP.NET Profile 为已知和匿名访问者存储个性化数据。

- 第 18 章“异常处理、调试和跟踪”。为了理解、改进和修复为 ASP.NET Web 页面编写的代码，需要有优秀的调试工具。Visual Web Developer 提供了出色的调试支持，可以在运行时诊断应用程序的状态，帮助在用户发现问题之前解决问题。
- 第 19 章“部署 Web 站点”。到本书末尾，应当有一个准备公布于众的 Web 站点。但是具体如何做呢？要发布 Web 站点，需要知道和了解哪些事情？本章给出了这些问题的答案，并说明为了运行最终的 Web 站点，如何配置不同的生产系统。

0.3 本书组织结构

本书通过运行示例和详细说明逐步地介绍概念。使用 Wrox 惯有的“试一试”练习与“工作原理”部分，可带领您一步步完成任务，并在任务进行过程中详细说明重要的地方。每个“试一试”后面都有一个详细的“工作原理”部分，用来解释在本练习中执行的步骤。

在每章的末尾都有一些练习，帮助测试从本章学到的知识。各个问题的答案在本书最后的附录 A 中可以找到。如果不知道问题的所有答案也不要担心，后面的章节不会假定您已经完成了前面章节中练习部分的任务。

由于这是一本面向初学者的书，所以对一些主题的介绍不是很详细。本书的每一章中都会提到一些专门介绍本章所讨论主题的其他书籍。在适当的时候还包含了对这些书的引用，所以当想要深入研究某一个特定主题时，可以很容易决定下一步该怎么办。

0.4 使用本书所需条件

本书假定您有一个符合下列要求的系统：

- 能够运行 Visual Web Developer。要了解具体系统要求，请参见该软件附带的 readme 文件。
- 运行 Windows Vista 或 Windows 7(两者都要求至少是 Home Premium 版)，或者是 Windows Server 2008 版本之一。

虽然使用 Windows 的其他版本，如 Windows XP(只要 Visual Web Developer 支持)也能完成大部分练习，但是本书第 19 章的练习要求使用 Microsoft 的 Web 服务器：IIS 7 或更高版本，而它们仅适用于上述要求列表中的 Windows 版本。

第 1 章介绍了如何获得并安装 Visual Web Developer 2010，然后安装 Microsoft .NET Framework 4 和 SQL Server 2008 Express Edition；所需要的只是一个优秀的操作系统以及阅读本书的动力！

0.5 源代码

读者在学习本书中的示例时，既可以手动输入所有的代码，也可以使用本书附带的源代码文件。本书使用的所有源代码都可以从本书合作站点 <http://www.wrox.com/> 或 www.tupwk.com.cn/downpage 上下载。只要登录到站点 <http://www.wrox.com/>，使用 Search 工具或使用书

名列表就可以找到本书。接着单击本书细目页面上的 **Download Code** 链接，就可以获得所有源代码。



提示：由于许多图书的书名都很类似，所以按 **ISBN** 进行搜索是最简单的，本书英文版的 **ISBN** 是 **978-0-470-50221-1**。

在下载了代码后，只需用自己喜欢的解压缩软件对它进行解压缩即可。另外，也可以进入 <http://www.wrox.com/dynamic/books/download.aspx> 上的 **Wrox** 代码下载主页，查看本书和其他 **Wrox** 图书的所有代码。

可以将本书的所有源代码作为一个文件下载(可根据语言来选择版本,C#或 Visual Basic),然后用喜欢的解压缩工具对其进行解压缩即可。提取源代码时，请确保维持作为代码下载一部分的原始文件夹结构。不同的解压缩工具对这个功能有着不同的名称，不过尽可能寻找一个像 **User Folder Names** 或 **Maintain Directory Structure** 这样的功能。从下载的代码中提取了文件之后，最后应有一个名为 **Source** 的文件夹以及一个名为 **Resources** 的文件夹。然后在 **C** 盘的根位置创建一个新文件夹，命名为 **BegASPNET**，并将 **Source** 和 **Resources** 文件夹移到这个新文件夹中，最后得到这样的文件夹：**C:\BegASPNET\Source** 和 **C:\BegASPNET\Resources**。**Source** 文件夹中包含本书 19 章中每一章的源代码文件，以及 **Planet Wrox Web** 站点的最终版本。**Resources** 文件夹包含本书的一些练习中所需要的文件。如果一切正常，最后应看到图 0-1 所示的结构。

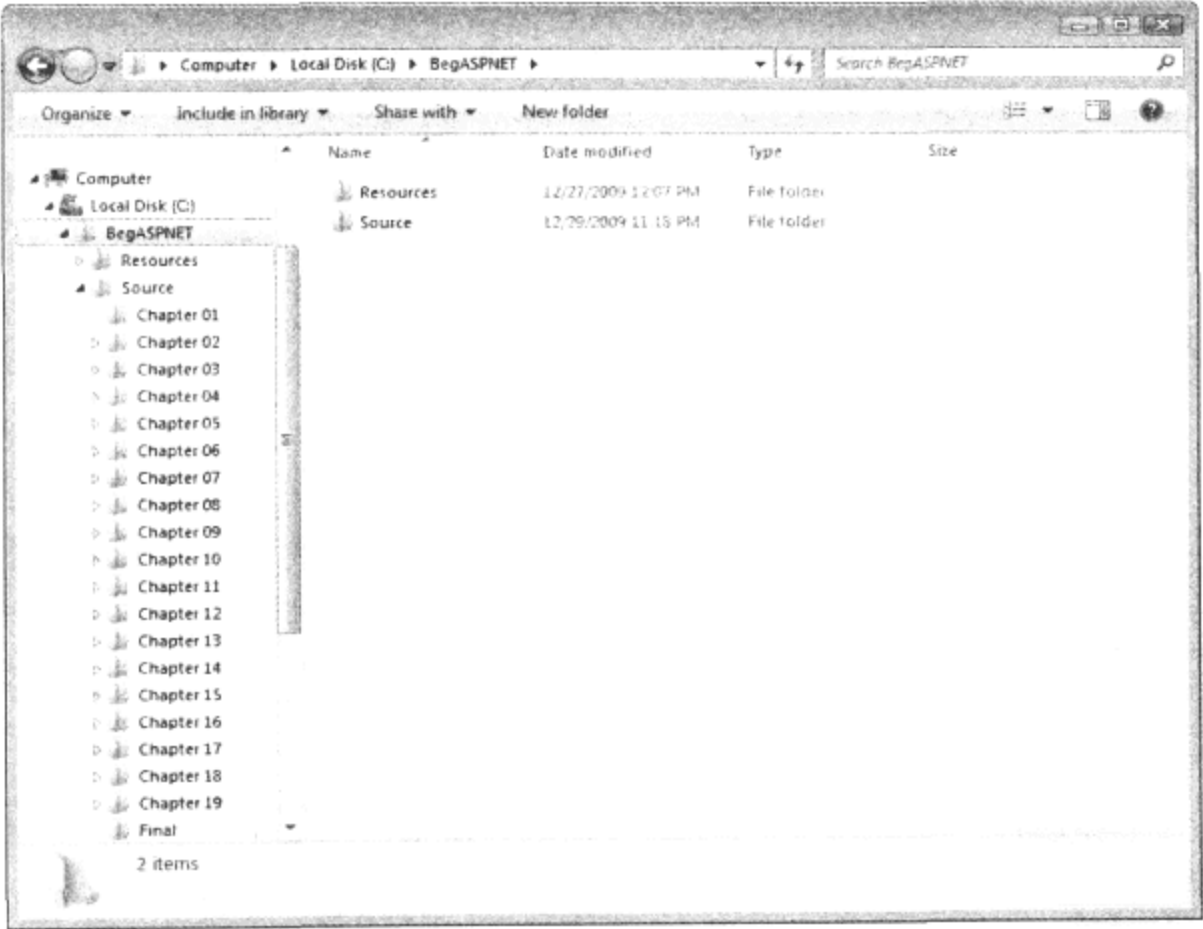


图 0-1

在以后的章节中将在 C:\BegASPNET 文件夹中创建名为 Site 和 Release 的文件夹，从而文件夹结构将为图 0-2 所示的样子。

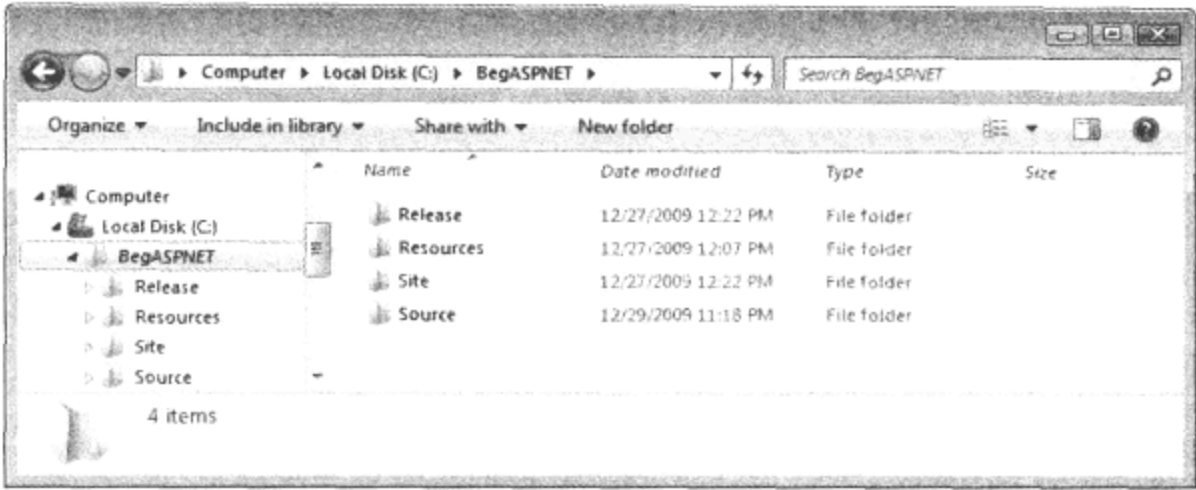


图 0-2

Site 文件夹包含本书将要构建的站点，而 Release 文件夹将包含本书末尾的站点的最终版本。每当做本书的一些练习受阻时，都可以打开 Source 文件夹查看一切最后应是什么样。

如果要为特定章节运行站点来看看它是如何工作的，一定要在 Visual Web Developer 中打开那一章的文件夹作为一个 Web 站点。因此应直接打开诸如 C:\BegASPNET\Source\Chapter12 这样的文件夹，而不是打开它的父文件夹 C:\BegASPNET\Source。

如果想要使用两种编程语言完成操作，则创建第二个文件夹 C:\BegASPNETVB 来存放 Visual Basic 版本的文件。这样一来，这两个站点就可以共存而不产生冲突。如果专门为 C# 语言创建一个文件夹，请不要包含#符号。因为对一个 Web 站点来说，路径名中的#是一个无效字符。

坚持采用这个结构可以确保顺利执行本书的“试一试”练习。错误地混合或嵌套这些文件夹会使练习的完成变得困难，还可能导致发生预料之外的情况和错误。每当遇到本书中没有解释的问题或错误时，请确保站点结构仍然与这里提出的结构紧密相关。

0.6 勘误表

尽管我们已经尽了最大的努力来保证文章或代码中不出现错误，但是错误总是难免的，如果你在本书中找到了错误，例如拼写错误或代码错误，请告诉我们，我们将非常感激。通过勘误表，可以让其他读者避免走入误区，当然，这还有助于提供更高质量的信息。

要在网站上找到本书英文版的勘误表，可以登录 <http://www.wrox.com>，通过 Search 工具或书名列表查找本书，然后在本书的细目页面上，单击 Book Errata 链接。在这个页面上可以查看到 Wrox 编辑已提交和粘贴的所有勘误项。完整的图书列表还包括每本书的勘误表，网址是 www.wrox.com/misc-pages/booklist.shtml。

如果你在勘误表上没有找到错误，那么可以到 www.wrox.com/contact/techsupport.shtml 上，完成上面的表格，并把找到的错误发送给我们。我们将会核查这些信息，如果无误的话，会把它放置到本书的勘误表中，并在本书的后续版本中更正这些问题。

0.7 p2p.wrox.com

要与作者和同行讨论，请加入 p2p.wrox.com 上的 P2P 论坛。这个论坛是一个基于 Web 的系统，便于你张贴与 Wrox 图书相关的消息和相关技术，与其他读者和技术用户交流心得。该论坛提供了订阅功能，当论坛上有新的消息时，它可以给你传送感兴趣的论题。Wrox 作者、编辑和其他业界专家和读者都会到这个论坛上来探讨问题。

在 <http://p2p.wrox.com> 上，有许多不同的论坛，它们不仅有助于阅读本书，还有助于开发自己的应用程序。要加入论坛，可以遵循下面的步骤：

- (1) 进入 p2p.wrox.com，单击 Register 链接。
- (2) 阅读使用协议，并单击 Agree 按钮。
- (3) 填写加入该论坛所需要的信息和自己希望提供的其他信息，并单击 Submit 按钮。
- (4) 你会收到一封电子邮件，其中的信息描述了如何验证账户和完成加入过程。



提示：不加入 P2P 也可以阅读论坛上的消息，但要张贴自己的消息，就必须加入该论坛。

加入论坛后，就可以张贴新消息，回复其他用户张贴的消息。可以随时在 Web 上阅读消息。如果要想让该网站给自己发送特定论坛中的消息，可以单击论坛列表中该论坛名旁边的 **Subscribe to this Forum** 图标。

关于使用 Wrox P2P 的更多信息，可阅读 P2P FAQ，了解论坛软件的工作情况以及 P2P 和 Wrox 图书的许多常见问题。要阅读 FAQ，可以在任意 P2P 页面上单击 FAQ 链接。

目 录

第 1 章	ASP.NET 4 入门	1
1.1	Microsoft Visual Web Developer	2
1.1.1	获取 Visual Web Developer	2
1.1.2	安装 Visual Web Developer Express Edition	3
1.2	创建第一个 ASP.NET 4 Web 站点	4
1.3	ASP.NET 4 简介	8
1.3.1	HTML	9
1.3.2	初识 ASP.NET 标记	13
1.4	IDE	13
1.4.1	主开发区	13
1.4.2	信息窗口	18
1.5	定制 IDE	19
1.5.1	重新排列窗口	19
1.5.2	修改 Toolbox	20
1.5.3	定制文档窗口	22
1.5.4	定制工具栏	22
1.5.5	定制键盘快捷键	23
1.5.6	重置修改	23
1.6	示例应用程序	24
1.7	关于 Visual Web Developer 的实用提示	26
1.8	本章小结	26
1.9	练习	26
第 2 章	构建 ASP.NET Web 站点	29
2.1	使用 VWD 2010 构建 Web 站点	29
2.1.1	不同的项目类型	30
2.1.2	选择正确的 Web 站点模板	31

2.1.3	构建与打开新的 Web 站点	32
2.2	操作 Web 站点中的文件	35
2.2.1	ASP.NET 4 Web 站点的文件类型	35
2.2.2	添加现有文件	39
2.2.3	组织站点	40
2.2.4	特殊文件类型	41
2.3	使用 Web 窗体	42
2.3.1	关于 Web 窗体的不同视图	42
2.3.2	在 Code Behind 和带内联代码的页面之间选择	43
2.3.3	向页面添加标记	48
2.3.4	连接页面	53
2.4	使用 Web 窗体的实用提示	55
2.5	本章小结	55
2.6	练习	55
第 3 章	设计 Web 页面	57
3.1	需要 CSS 的原因	57
3.1.1	HTML 格式化的问题	58
3.1.2	CSS 如何解决格式化问题	58
3.2	CSS 简介	59
3.2.1	CSS 语言	62
3.2.2	样式表	62
3.2.3	向页面中添加 CSS	74
3.3	在 VWD 中使用 CSS	76
3.3.1	在外部样式表中创建新样式	76
3.3.2	创建内嵌和内联样式表	81
3.3.3	应用样式	85
3.3.4	管理样式	87

3.4	关于使用 CSS 的实用提示	89	5.4.1	方法：函数与子例程	151
3.5	本章小结	90	5.4.2	App_Code 文件夹	153
3.6	练习	90	5.4.3	使用名称空间组织代码	157
第 4 章	使用 ASP.NET 服务器控件	93	5.4.4	写注释	160
4.1	服务器控件简介	93	5.5	面向对象编程基础知识	162
4.2	ASP.NET 服务器控件详解	97	5.5.1	重要的面向对象术语	162
4.2.1	在页面中定义控件	97	5.5.2	事件	172
4.2.2	所有控件的共同属性	98	5.6	关于编程的实用提示	173
4.3	控件的类型	100	5.7	本章小结	174
4.3.1	标准控件	100	5.8	练习	174
4.3.2	HTML 控件	112	第 6 章	创建外观一致的 Web 站点	177
4.3.3	数据控件	112	6.1	用母版页创建一致的页面	
4.3.4	有效性验证控件	112		布局	177
4.3.5	导航控件	113	6.1.1	创建母版页	179
4.3.6	登录控件	113	6.1.2	创建内容页	181
4.3.7	Ajax 扩展	113	6.2	使用集中的基页	186
4.3.8	WebParts	113	6.2.1	ASP.NET 页面生命周期	187
4.3.9	动态数据	113	6.2.2	实现基页	188
4.4	ASP.NET 状态引擎	114	6.2.3	创建可重用的页面模板	192
4.4.1	状态的定义及其重要性	114	6.3	主题	195
4.4.2	状态引擎的工作原理	114	6.3.1	不同类型的主题	196
4.4.3	并非所有控件都依赖于 View State	118	6.3.2	在 Theme 和 StyleSheetTheme 之间作选择	196
4.4.4	关于 View State 和性能的 一个注意点	118	6.3.3	应用主题	196
4.5	使用控件的实用提示	119	6.3.4	扩展主题	200
4.6	本章小结	120	6.3.5	动态切换主题	202
4.7	练习	120	6.4	外观	208
第 5 章	ASP.NET Web 页面编程	123	6.4.1	创建一个 skin 文件	209
5.1	编程简介	124	6.4.2	已命名外观	210
5.2	数据类型与变量	124	6.4.3	对特定控件禁用主题	211
5.2.1	转换数据类型	127	6.5	创建一致页面的实用提示	211
5.2.2	使用数组和集合	129	6.6	本章小结	212
5.3	语句	134	6.7	练习	212
5.3.1	运算符	134	第 7 章	导航	215
5.3.2	做决策	142	7.1	在站点中移动	215
5.3.3	循环	148	7.1.1	理解绝对 URL 与 相对 URL	216
5.4	组织代码	151	7.1.2	默认文档	220

7.2 使用导航控件.....	220	9.2.2 从文本文件中读取数据.....	287
7.2.1 导航控件的体系结构	221	9.3 关于验证数据有效性的	
7.2.2 分析 Web.sitemap 文件.....	221	实用提示	291
7.2.3 使用 Menu 控件.....	223	9.4 本章小结	292
7.2.4 使用 TrceView 控件.....	230	9.5 练习	292
7.2.5 使用 SiteMapPath 控件.....	234		
7.3 以编程的方式重定向	236	第 10 章 ASP.NET AJAX.....	295
7.3.1 通过编程将客户重定向		10.1 AJAX 简介.....	296
到不同页面	236	10.2 在项目中 使用 ASP.NET	
7.3.2 服务器端重定向	239	AJAX.....	297
7.4 关于导航的实用提示	240	10.2.1 创建无闪烁页面	297
7.5 本章小结	241	10.2.2 给用户提供反馈	302
7.6 练习	241	10.2.3 使用 Timer 控件	306
		10.3 在 Ajax Web 站点中使用 Web	
第 8 章 用户控件	243	服务和页面方法	307
8.1 用户控件简介	243	10.3.1 Web 服务的定义	307
8.1.1 创建用户控件	244	10.3.2 创建 Web 服务	308
8.1.2 向内容页或母版页中		10.3.3 在 Ajax Web 站点中	
添加用户控件	247	使用 Web 服务	311
8.1.3 用户控件的站点范围		10.3.4 页面方法简介	317
注册	250	10.3.5 客户端 ASP.NET AJAX	
8.1.4 关于用户控件的警告	251	Library	320
8.2 向用户控件添加逻辑	253	10.3.6 这仅仅是开始	322
8.2.1 为属性创建自己的数据		10.4 有关 Ajax 的实用提示	322
类型	253	10.5 本章小结	323
8.2.2 实现 View State 属性	258	10.6 练习	323
8.2.3 关于 View State 要考虑的			
事项	263	第 11 章 jQuery	325
8.3 关于用户控件的实用提示	263	11.1 jQuery 简介	326
8.4 本章小结	263	11.1.1 选择引用 jQuery	
8.5 练习	264	的位置	326
		11.1.2 包含 jQuery 库的	
第 9 章 验证用户输入有效性.....	265	不同方式	327
9.1 收集用户数据	266	11.2 jQuery 语法	331
9.1.1 验证 Web 窗体中用户输入		11.2.1 jQuery Core.....	331
的有效性	267	11.2.2 使用 jQuery 进行选择	332
9.1.2 理解请求有效性验证	281	11.3 使用 jQuery 修改 DOM.....	338
9.2 在服务器上处理数据	282	11.3.1 CSS 方法	338
9.2.1 从 Web 站点中发送		11.3.2 处理事件	339
电子邮件	282	11.3.3 jQuery 的各种功能	341

11.3.4 使用 jQuery 时常犯的错误.....	341	13.2.3 在 web.config 文件中存储连接字符串.....	393
11.4 使用 jQuery 的效果.....	346	13.2.4 筛选数据.....	395
11.5 扩展 jQuery.....	351	13.3 自定义数据控件的外观.....	400
11.6 关于 jQuery 的实用提示.....	354	13.4 更新和插入数据.....	405
11.7 本章小结.....	354	13.5 显示和更新数据的实用提示.....	417
11.8 练习.....	354	13.6 本章小结.....	418
第 12 章 初识数据库.....	357	13.7 练习.....	418
12.1 数据库的概念.....	358	第 14 章 LINQ 和 ADO.NET Entity Framework.....	421
12.2 不同类型的关系数据库.....	359	14.1 LINQ 简介.....	421
12.3 运用 SQL 处理数据库数据.....	359	14.1.1 LINQ to Objects.....	422
12.4 使用 SQL 检索和操纵数据.....	361	14.1.2 LINQ to XML.....	423
12.4.1 读取数据.....	361	14.1.3 LINQ to ADO.NET.....	423
12.4.2 创建数据.....	369	14.2 ADO.NET Entity Framework 简介.....	423
12.4.3 更新数据.....	369	14.3 将数据模型映射到对象模型.....	424
12.4.4 删除数据.....	369	14.4 查询语法.....	430
12.5 创建自己的表.....	372	14.4.1 标准查询操作符.....	430
12.5.1 SQL Server 中的数据类型.....	372	14.4.2 用匿名类型定形数据.....	433
12.5.2 了解主键和标识列.....	373	14.5 结合使用服务器控件和 LINQ 查询.....	438
12.5.3 创建表之间的关系.....	376	14.5.1 在 Entity Framework 中使用数据控件.....	438
12.6 有关数据库的实用提示.....	379	14.5.2 有关性能的一些注意点.....	464
12.7 本章小结.....	380	14.6 有关 LINQ 和 ADO.NET Entity Framework 的实用提示.....	464
12.8 练习.....	380	14.7 本章小结.....	465
第 13 章 显示和更新数据.....	383	14.8 练习.....	465
13.1 数据控件.....	383	第 15 章 处理数据——高级主题.....	467
13.1.1 数据绑定控件.....	383	15.1 使用样式格式化控件.....	467
13.1.2 数据源控件.....	385	15.1.1 关于样式.....	468
13.1.3 其他数据控件.....	386	15.1.2 合并样式、主题和外观.....	472
13.2 联合使用数据源和数据绑定控件.....	386	15.2 处理事件.....	475
13.2.1 使用 GridView 显示和编辑数据.....	386		
13.2.2 使用 DetailsView 插入数据.....	391		

15.2.1 回顾 ASP.NET 页面和 控件生命周期	476	17.1.1 配置 Profile	546
15.2.2 ASP.NET 页面生命周期和 数据控件中的事件	480	17.1.2 使用 Profile	552
15.2.3 处理数据源控件中发生 的错误	485	17.2 Profile 的其他使用方法	568
15.3 手动编写数据访问代码	489	17.2.1 匿名标识	569
15.4 缓存	499	17.2.2 清除旧的匿名配置 文件	569
15.4.1 缓存数据的常见问题	500	17.2.3 查看其他用户的配置 文件	570
15.4.2 在 ASP.NET Web 应用 程序中缓存数据的不同 方法	500	17.3 关于个性化的实用提示	573
15.5 有关数据的实用提示	509	17.4 本章小结	574
15.6 本章小结	509	17.5 练习	574
15.7 练习	509	第 18 章 异常处理、调试和跟踪	577
第 16 章 ASP.NET 4 Web 站点中的 安全性	511	18.1 异常处理	577
16.1 关于安全性	511	18.1.1 不同类型的错误	578
16.1.1 身份：您是谁	512	18.1.2 捕获和处理异常	579
16.1.2 身份验证：如何证明 您是谁	512	18.1.3 全局错误处理和自定义 错误页面	586
16.1.3 授权：允许您做什么	512	18.2 调试基础知识	592
16.1.4 ASP.NET 应用程序 服务	512	18.3 调试的工具支持	595
16.2 登录控件	514	18.3.1 在调试代码中移动	596
16.2.1 登录控件	518	18.3.2 调试窗口	596
16.2.2 配置 Web 应用程序	528	18.4 调试客户端脚本	601
16.3 Role Manager	531	18.5 跟踪 ASP.NET Web 页面	604
16.3.1 配置 Role Manager	532	18.5.1 使用标准的跟踪功能	605
16.3.2 使用 WSAT 管理用户	532	18.5.2 添加自己的信息到 Trace 中	608
16.3.3 配置 Web 应用程序 使用角色	535	18.5.3 跟踪和性能	610
16.3.4 以编程方式检查角色	539	18.5.4 安全警告	610
16.4 有关安全性的实用提示	542	18.6 有关调试的实用提示	610
16.5 本章小结	543	18.7 本章小结	611
16.6 练习	543	18.8 练习	611
第 17 章 个性化 Web 站点	545	第 19 章 部署 Web 站点	613
17.1 Profile	546	19.1 准备部署 Web 站点	613
		19.1.1 避免硬编码的设置	614
		19.1.2 web.config 文件	614
		19.1.3 表达式语法	614
		19.1.4 WebConfiguration- Manager 类	615

19.2	复制 Web 站点.....	620	19.4	将数据移动到远程 服务器中.....	636
19.2.1	创建 Web 站点的简单 副本	621	19.4.1	使用 Database Publishing Wizard.....	636
19.2.2	发布 Web 站点	623	19.4.2	重建数据库	638
19.3	在 IIS 下运行站点	624	19.5	部署清单	639
19.3.1	安装和配置 Web 服务器	625	19.6	补充资源	640
19.3.2	了解 IIS 中的安全性	630	19.7	本章小结	641
19.3.3	Planet Wrox 的 NTFS 设置	631	19.8	练习	641
19.3.4	检修 Web 服务器错误	634	附录 A	练习答案	643
			附录 B	配置 SQL Server 2008	667

350元等于什么？答案就是等于Microsoft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！ 视频包括浪曦 本杰 中美IT 等290 GB视频教程 需要的请联系QQ：2569343569

答案就是等于Microsoft .NET技术！

等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！！！

视频包括浪曦 本杰 中美IT 等290 GB视频教程 课程目录请百度一下 2569343569 新浪博客

包你学会.NET 需要的请联系QQ：2569343569



本章要点

- 如何获取和安装 Visual Web Developer 2010 Express 和 Visual Studio 2010
- 如何使用 Visual Web Developer 创建第一个 Web 站点
- 服务器如何处理 ASP.NET 页面并将其传送给浏览器
- 如何使用和定制开发环境

自从.NET Framework 1.0 在 2002 年初首次发布以来，Microsoft 花了大量精力和时间来开发 ASP.NET，它是.NET Framework 的一部分，可以用来构建富 Web 应用程序。首次发布意味着从过去的 Microsoft 技术向构建 ASP (Active Server Page，活动服务器页面，现在人们常称之为传统 ASP)Web 站点的飞跃。与传统 ASP 相比，ASP.NET 1.0 及相关的 Visual Studio .NET 2002 的引入给开发人员带来了如下好处：

- 页面显示与代码清楚地分开。使用传统 ASP 时，编程逻辑常常散布在整个页面的 HTML 中，使得后面对页面的修改比较困难。
- 开发模型更接近于桌面应用程序的编程方式。这样很多 Visual Basic 桌面程序员可以轻松地转换到 Web 应用程序。
- 它有一个功能丰富的开发工具(称为 Visual Studio .NET)，开发人员可以通过它可视化地创建和编写 Web 应用程序代码。
- 有几种面向对象的编程语言可供选择，其中 Visual Basic .NET 和 C#(读作 C-Sharp)是目前最流行的两种语言。
- 它可以访问整个.NET Framework，这意味着 Web 开发人员首次拥有了一种统一且容易的方式，来使用数据库、文件、e-mail、网络工具等许多高级功能。

尽管 ASP.NET 远优于旧模型，但使用它也意味着构建应用程序时的复杂性以及所需知识量的增加，所以它对于许多新的程序员来说，更难上手。

在 2002 年首次发布以后，Microsoft 在 2003 年发布了.NET Framework 的另一个版本(称为.NET 1.1)和 Visual Studio .NET IDE。尽管在架构和开发工具方面都有了不少新的改进，但很多人还是习

惯把这些看作是初始版本的一个服务包。

2005 年 11 月，Microsoft 发布了 Visual Studio 2005 和 ASP.NET 2.0。让全球许多开发人员感到惊喜的是，Microsoft 又大大改进和扩展了产品，增加了许多功能和工具来帮助降低 ASP.NET 1.0 所带来的复杂性。新的向导和智能控件减少了构建应用程序所需的代码，降低了新开发人员的学习难度，并且提高了开发效率。

尽管 Visual Studio 2005 和 ASP.NET 2.0 的功能已经很丰富了，但 Microsoft 仍旧努力向 2007 年 11 月发布的 Visual Studio 2008 和 ASP.NET 3.5 中添加了一系列很酷的新功能。主要的新功能包括 LINQ(将在第 14 章中介绍)以及 AJAX 框架整合(将在第 10 章中介绍)。2008 年 8 月，Microsoft 发布了用于 Visual Studio 和 .NET Framework 的 Service Pack 1，其中引入了一些重要的新功能，如 ADO.NET Entity Framework(第 14 章将会讨论)和动态数据。

目前的版本是 Visual Studio 2010(通常读作 twenty-ten)和 ASP.NET 4，它是在已成功发行的 Visual Studio 2008 和 ASP.NET 3.5 基础之上构建的，保留了其中很多令人喜爱的功能，并增加了一些其他领域的新功能和工具。

在本书接下来的 19 章中，将会介绍如何使用 Visual Web Developer 2010(它是 Microsoft 为 ASP.NET Web 应用程序设计的开发工具)构建功能完全的 ASP.NET Web 站点。本书将引导您了解构建功能完全的、数据库驱动的 Web 站点的过程。从本章的基本 Web 站点开始，直到第 19 章将它部署到生产环境中。

本书的示例站点和所有示例都是用 Visual Web Developer 2010(VWD)构建的，因此需要把它安装到开发机器上。下一节将介绍如何获得和安装 VWD。安装并运行 VWD 后，就知道如何创建第一个 Web 站点，接着我们介绍 VWD 的许多功能。

1.1 Microsoft Visual Web Developer

尽管从理论上讲，只用 Notepad 或其他文本编辑器就可以编写 ASP.NET Web 应用程序，但您可能还是希望安装 Microsoft Visual Web Developer 2010。VWD 是专门为构建 ASP.NET Web 站点而开发的，因此，其中包含了大量有助于快速创建复杂 ASP.NET Web 应用程序的工具。

Visual Web Developer 有两个版本：一个是独立而免费的版本，称为 Microsoft Visual Web Developer 2010 Express；还有一个版本是作为较大的开发套件 Visual Studio 2010 的一部分，它有不同的版本可用，且各个版本的价格各不相同。虽然 VWD 的 Express Edition 是免费的，但是它包含了创建复杂且功能丰富的 Web 应用程序所需的所有功能和工具。本书中的所有示例都可以用免费的 Express Edition 构建出来，因此不需要为了学习本书而花大本钱去购买 Visual Studio 2010 的商业版本。

VWD 很容易获得。从 Microsoft 站点下载即可，具体方法参见下一小节。

1.1.1 获取 Visual Web Developer

可以从 Microsoft 站点 www.microsoft.com/express/ 上下载 VWD 的免费版本。在 Express 的主页上，依次单击 Download 链接，直到打开提供了下载 Express 产品的下载页面，其中包括 Visual Web Developer 2010 Express Edition。在这个页面上可以以 Web 安装方式下载 Visual Web Developer 2010

Express Edition，这里只下载安装程序，文件的其余部分在安装过程中下载。一定要在这个页面上选择 Visual Web Developer 2010，而不要选择其他免费的 Express 产品或 Visual Web Developer 的以前版本。也可以从这个页面上方便地以 ISO 映像方式下载所有的 Express 产品，以便刻录到 DVD 上。

不要被以 Web 安装方式下载的大约是 3.5MB 的文件大小所迷惑。以这种方式下载的文件只是从 Internet 上下载必需文件的安装程序。全部的下载量依赖于当前的系统，大约在 180MB 到 270MB 之间。

如果想试试同样包含 VWD 的 Visual Studio 2010 的完整版本，则可以从 Microsoft 的站点 <http://msdn.microsoft.com/vstudio> 上下载有免费试用期的版本。如果需要刻录到 DVD 上，则可以选择下载一个 ISO 映像程序。

最后，可以从 www.microsoft.com/web 和 www.asp.net/vwd/ 站点上下载 Microsoft Web Platform Installer(WPI)应用程序，其中就包含了 VWD。除了 VWD 以外，这个工具还便于访问其他许多与 Web 开发相关的工具和程序。通过使用 WPI 这个优秀的工具，可以同时获得大量与 Web 开发相关的程序和工具。我经常使用这个工具来快速建立一个开发环境。

1.1.2 安装 Visual Web Developer Express Edition

Visual Web Developer 的安装很简单，只是过程有点长。根据所选的安装方法、计算机配置和 Internet 连接速度，安装 VWD 可能需要 20 分钟到一个小时，甚至更长时间。

试一试

安装 Visual Web Developer 2010 Express Edition

本“试一试”练习用来指导如何在计算机上安装 VWD Express Edition。它假定用户选择的是从 Web 上下载，尽管从 DVD 安装 Express 版本的过程几乎也是这样。安装 Visual Studio 2010 的完整版本所需执行的步骤与之相似，只是看到的屏幕略有不同。

不管安装 VWD 的哪个版本，都要安装 SQL Server 2008 Express Edition 的 Service Pack 1 版——本书的很多示例都会用到这个组件。如果安装的是 Visual Studio 2010 的完整版，那么在安装过程中会看到要安装的功能列表中包括安装 SQL Server 的选项。如果安装 VWD Express Edition，Installer Options 对话框中就会出现选择 SQL Server 的选项。Web Platform Installer 包括相似的选项，允许安装 SQL Server 2008 Express 的 SP1 版或者稍后定位到 Web Platform | Database 下。

(1) 在安装 Web 版本时，运行从 Microsoft Web 站点上下载的文件。或者从 Visual Web Developer DVD 上启动安装过程。

(2) 启动了安装程序后，单击 Next 按钮，阅读并接受许可条款，并再次单击 Next 按钮。

(3) 在 Installer Options 页面上，确保选择了 Microsoft SQL Server 2008 Express Edition。虽然这两个选项使要下载的文件大小增加了不少，但是它们对于构建数据驱动的 ASP.NET Web 应用程序来说非常有用。如果没有看到 SQL Server 选项，就表示已经安装过了。如果无法确定是否已安装了 SQL Server 2008，则可以参考附录 B。再次单击 Next 按钮。可能会看到其他的也可安装的可选组件，例如 Microsoft Silverlight，虽然它在本书中并不使用。

(4) 在 Destination Folder 页面上，如果主磁盘的空间足够，可以让 Install In Folder 字段保留默认值。否则，单击 Browse 按钮选择另一个位置。

(5) 单击 Install 按钮。如果使用的是基于 Web 的安装程序，该安装应用程序就会首先从 Internet 上将文件下载到计算机上。在安装过程中，将出现一个显示 VWD 的下载和安装进度的屏幕(类似于

图 1-1 所示)。

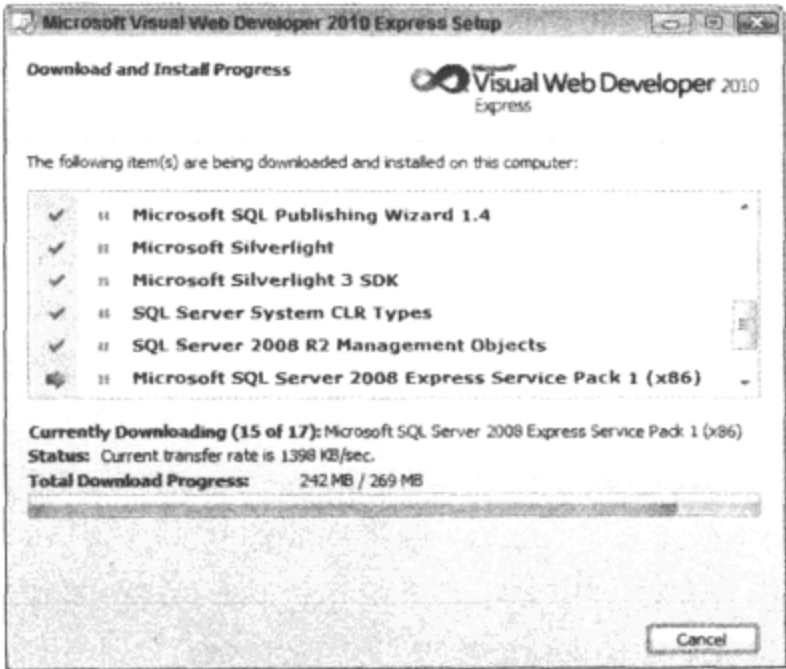


图 1-1

(6) 完成了应用程序的安装后，可能会出现一个对话框，要求重启计算机。重启后，VWD 就可以使用了。

工作原理

上面的“试一试”练习介绍了 VWD 2010 Express Edition 的安装过程。在 Installation Options 对话框中，选择了 Microsoft SQL Server 2008 Express，它是 Microsoft 数据库引擎的免费版本。从第 12 章开始，本书将多次讨论和使用 SQL Server 2008。附录 B 说明了如何使用免费 SQL Server Management Studio Express Edition 对 SQL Server 2008 的各个版本进行安全设置。

现在已经安装了 VWD，接下来应启动它并开始使用。下一节将介绍如何使用 VWD 创建第一个站点。您将了解到如何创建站点、向 Web 页面中添加内容，以及如何在浏览器中浏览该页面。

1.2 创建第一个 ASP.NET 4 Web 站点

您可能已经迫不及待地想要开始创建第一个 ASP.NET Web 站点，所以，我们现在就不多作 VWD 的理论概述了。下面的“试一试”练习将直接进入实战运用，它会指出如何构建第一个 Web 项目。然后，在“工作原理”部分及其后的小节中，会作一些解释，使您对在浏览器中浏览 ASP.NET 页面时后台的工作有一个很好的了解。

试一试

创建第一个 ASP.NET Web 站点

- (1) 如果还没有启动 VWD 2010，则从 Windows 的“开始”菜单中启动 VWD 2010。如果是第一次启动 VWD，则在开始使用 VWD 工作之前，会有一些延迟，因为它要先做一些必要的配置。以后再启动时就会快得多。
- (2) 如果使用的是 Visual Studio 的商业版本，则在首次启动 Visual Studio 时还会出现一个对话框，

要求在不同的设置集合中作出选择。在这个对话框中所作的选择将会影响窗口、工具箱、菜单和快捷键的布局。选择 **Web Development Settings**，因为这些设置是专门为 ASP.NET 开发人员设计的。通常，可以通过重置设置来选择不同的配置文件，本章后面将会解释。

(3) 完全配置好 VWD 后，就会出现主屏幕，如图 1-2 所示。

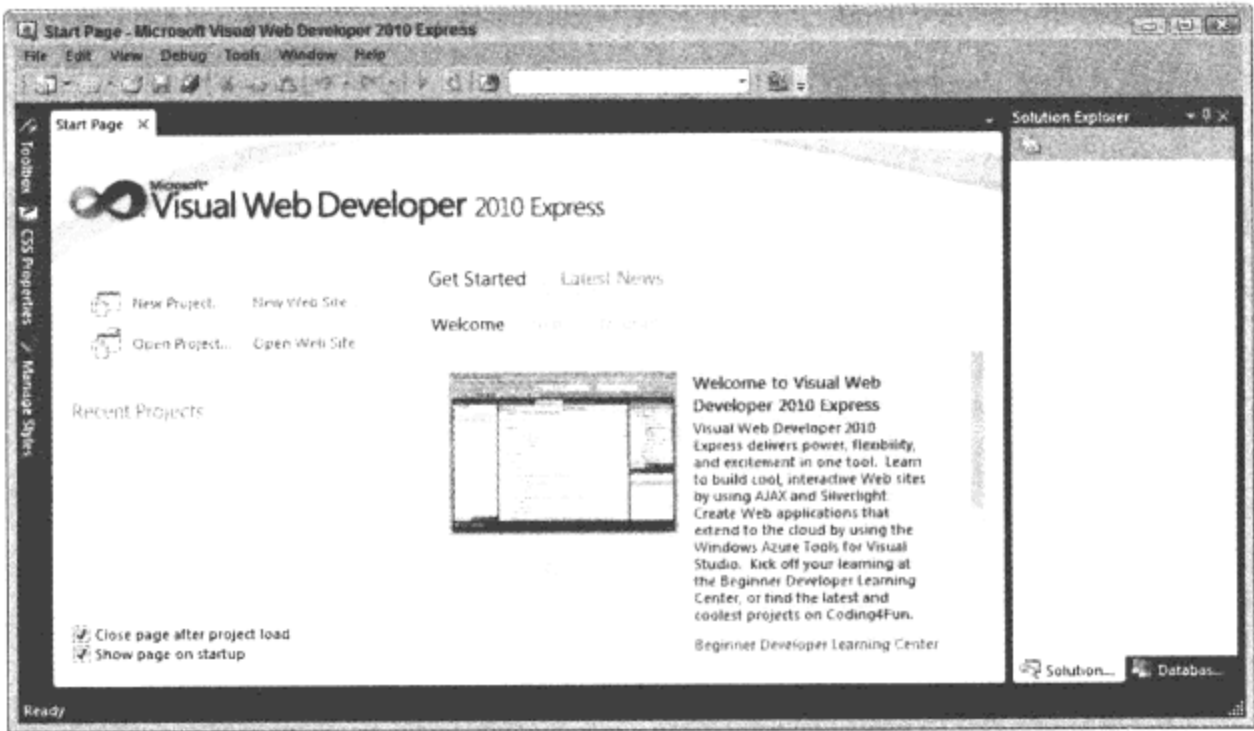


图 1-2

下一节将对所有的窗口、工具栏、面板和菜单进行描述。因此这里将重点放在创建新 Web 站点上。单击左上角的 **File** 菜单，并选择 **New Web Site**。如果使用的是 Visual Studio 的商业版本，那么根据首次启动 Visual Studio 时选择的设置，可能会先打开子菜单 **New**。(注意，不要选择 **New Project** 菜单项，因为它是用来创建另一种类型的 .NET 应用程序的)。这时会出现 **New Web Site** 对话框，如图 1-3 所示。

(4) 在左边的 **Installed Templates** 部分，选择一种供站点使用的编程语言。本书显示的所有示例都使用了 **Visual Basic** 和 **Visual C#**，因此可以根据自己的喜好选择一种语言。

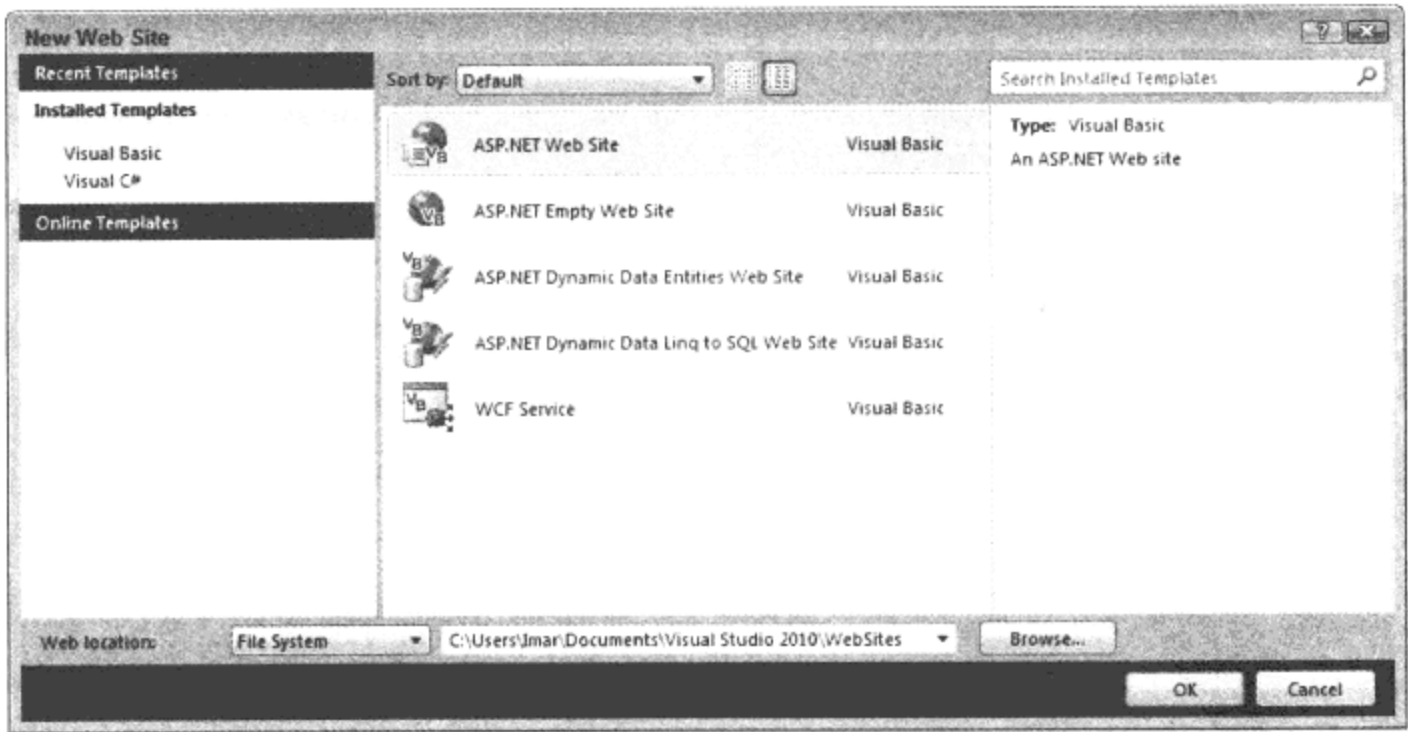


图 1-3

(5) 在中间的模板列表框中，确认选择了 ASP.NET Web Site。还要确认在左下方的 Web Location 下拉列表中选中 File System。如果愿意，可以改变 Web 站点在磁盘上的存储位置，单击 Browse 按钮，在计算机的硬盘驱动器上选择一个新位置即可。就目前而言，用默认位置即可——Documents 文件夹下的一个文件夹，因此可以让这个位置保持原样。

(6) 单击 OK 按钮。这时 VWD 就创建了一个新的 Web 站点，其中包括了许多如图 1-4 所示的文件和文件夹，它们可以用来开始创建 Web 站点。还会打开文件 Default.aspx，这样就可以看到该页面的代码。



图 1-4

(7) 删除<asp:Content>代码块中的代码(起始标记是<h2>，结束标记是</p>），用下面突出显示的文本和代码进行替换：

```
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
    <h2>Hello World</h2>
    <p>Welcome to Beginning ASP.NET 4 on <%= DateTime.Now.ToString() %></p>
</asp:Content>
```

在本书中，您将会看到很多这种格式的代码。当书中要求输入这种粗体格式的代码时，只需输入突出显示的代码即可。其余的代码应该已经存在于文件中了。

不用关心欢迎消息中使用尖括号(<>)的代码和分数符号；到本书后面自然就会知道它是如何工作的。虽然到目前为止，您可能还不熟悉这样的代码，但是不难猜出它是用来做什么的：输出今天的日期和时间。

(8) 按 Ctrl+F5 组合键在默认的 Web 浏览器中打开页面，如图 1-5 所示。

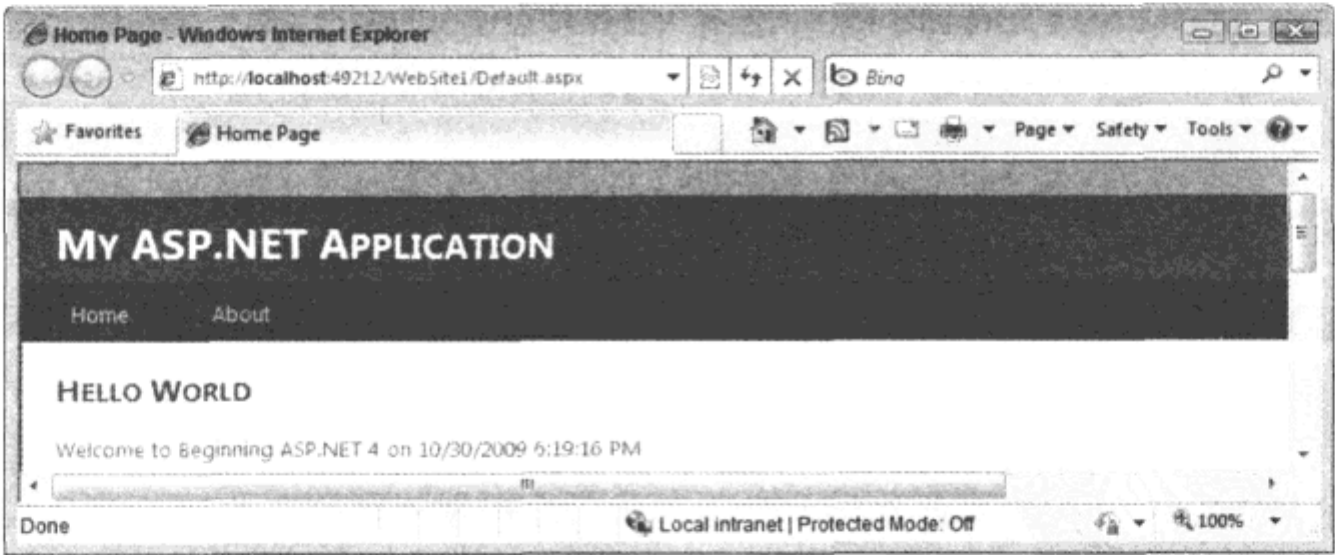


图 1-5

如果出现一个对话框，要求输入用户名和密码，则关闭浏览器并返回到 VWD。在 Solution Explorer 中右击站点(它是图 1-4 中的第一个选项)并选择 Property Pages。在 Start Options 部分清空 NTLM 身份验证项的复选框。然后单击 OK 按钮并再次按下 Ctrl+F5 组合键，在浏览器中查看页面。如果在 Internet Explorer 中看到了关于 Intranet 设置的信息警告条，则单击该警告条并选择 Enable

Intranet Settings。如果想先了解一下关于这些设置的更多含义，请从弹出菜单中选择 What are Intranet Settings 项。

如果在页面中没有看到时间和日期，或者收到了错误消息，则回顾一下欢迎消息中的代码。它以前尖括号(<)开头，后面跟着一个百分号和一个等号，以一个百分号和另一个后尖括号(>)结束。尽管如此，还是要确保输入和此处完全相同的代码，包括大小写也要一致。如果使用的语言是 C#，则这一点特别重要，因为该语言是区分大小写的。

(9) 注意，Windows 的任务栏中会出现一个带屏幕提示的小图标，如图 1-6 所示。



图 1-6

如果没有看到该图标，则可以右击 Windows 任务栏中其他图标旁边的箭头并选择 Customize Notification Icons。然后把 WebDev.WebServer40.exe 选项设置为 Show Icon and Notifications。这个图标属于 ASP.NET Development Server。该 Web 服务器由 VWD 自动启动，以响应对页面的请求。本书后面将会介绍 Web 服务器如何处理页面。

这就是用 VWD 创建第一个 ASP.NET 4 Web 站点的过程。

工作原理

虽然刚刚在上面的“试一试”练习中创建的这个 Web 站点相当简单，但是让 Default.aspx 页面显示在浏览器中的过程却没有那么简单。ASP.NET 页面(根据它的扩展名，也称为 ASPX 页面)本身并不能做太多的事。在浏览器能够显示它之前，需要一个 Web 服务器对它进行处理。这就是 VWD 自动启动内置的 ASP.NET Development Server 来处理页面请求的原因。接下来，它会启动默认的 Web 浏览器并定向到“试一试”练习中的 Web 服务器的地址：<http://localhost:49212/WebSite1/Default.aspx>，不过每次启动 Web 服务器时这个地址中的实际数字可能都不同，因为该数字是 VWD 随机选择的。

重要的是要意识到在 VWD 中修改的 ASPX 文件并不是在浏览器中最终显示的文件。

当在 VWD 中创建一个页面时，就向它添加了一个标记(markup)。ASPX 页面中的标记由以下内容组成：纯文本、HTML、ASP.NET 服务器控件的代码(在本章和第 4 章中将对它进行更多的介绍)、用 Visual Basic .NET 或 C#编写的代码等。

当在浏览器中请求一个 ASPX 页面时，Web 服务器就会处理这个页面，执行它在文件中找到的所有代码，并有效地将 ASP.NET 标记转换为纯 HTML，然后发送给显示这个页面的浏览器。在上面的“试一试”练习中，最终的 HTML 会引起浏览器显示当前日期和时间。HTML(HyperText Markup Language)是浏览器用来显示 Web 页面的语言。本章后面将会介绍 HTML，以及如何使用 HTML。

要查看最终的 HTML 与原始的 ASPX 页面有什么区别，请在浏览器中打开该页面的源代码。在大多数浏览器中，可以通过右击页面并选择 View Source 或 View Page Source 命令来打开源代码窗口。这样还会打开一个默认的文本编辑器，用于显示该页面的 HTML 代码。

如果在完成上面的“试一试”练习的操作后已经关闭了浏览器，则也可以在 VWD 中按 Ctrl+F5 组合键再次打开该页面并选择 View Source 命令。

文本编辑器中的大部分 HTML 与原始的 ASPX 页面相似。然而，如果看一下显示欢迎消息和当

前日期与时间的那一行代码，就会注意到它们有很大的区别。现在看到的不是尖括号和百分号之间的代码，而是实际的日期和时间。

```
<div class="main">
  <h2>Hello World</h2>
  <p>Welcome to Beginning ASP.NET 4 on 10/30/2009 6:19:16 PM</p>
</div>
```

当 Web 服务器处理页面时，它就从服务器中查询当前日期和时间，并把它们插入到要发送给浏览器的 HTML 中。根据 Windows 安装中的语言设置，会看到为适应 Windows 区域设置而进行格式化的不同日期和时间。

下一节将更详细地介绍 ASP.NET 工作原理。

1.3 ASP.NET 4 简介

当在 Web 浏览器中输入 `www.wrox.com` 这样的 Web 地址并按下 Enter 键时，浏览器就会向那个地址的服务器发送一个请求。这个过程是通过 HTTP(HyperText Transfer Protocol，超文本传输协议)完成的。HTTP 是 Web 浏览器与 Web 服务器之间进行通信的协议。当发送地址时，就是向服务器发送了一个请求。当服务器处于活动状态且请求有效时，服务器就会接受并处理请求，然后将响应发送回客户端浏览器。请求与响应之间的关系如图 1-7 所示。

由于使用了内置的 Development Web Server，因此服务器和客户端是同一台机器。但是，在现实情况下，会将 Web 站点放到外部 Web 服务器上，该服务器可以被许多不同的客户端访问。

对于简单的静态文件，比如 HTML 文件或图像，Web 服务器只是简单地从它的本地硬盘驱动器中读取文件并发送给浏览器。然而，对于动态文件，比如 ASPX 页面，这样做显然是不够的。假如 Web 服务器将 ASPX 文件直接作为文本文件发送给浏览器，那么在浏览器中就看不到当前的日期与时间，而只能看到实际的代码(`<%= DateTime.Now.ToString() %>`)。因此，Web 服务器不是直接发送这个文件，而是将这个请求传递给能够处理该页面的另一个软件模块。这是通过所谓的 Application Mapping 或 Handler Mapping 概念来完成的，其中文件的扩展名(在本例中是.aspx)表明了能够处理它的应用程序。就.aspx 页面而言，请求最终由 ASP.NET 运行库(它是 Microsoft .NET Framework 的一部分，专门为处理 Web 请求而设计)处理。

在页面的处理过程中，有 3 个重要方面会影响页面最终出现在浏览器中的方式：

- **静态文本** 任何静态文本(如 HTML、CSS 或者可以放在页面中的 JavaScript 代码)都是直接发送给浏览器的。本章和接下来的几章将介绍有关 HTML、CSS 和 JavaScript(一种在客户端使用的编程语言)的更多知识，其中第 3 章将详细讨论 CSS。

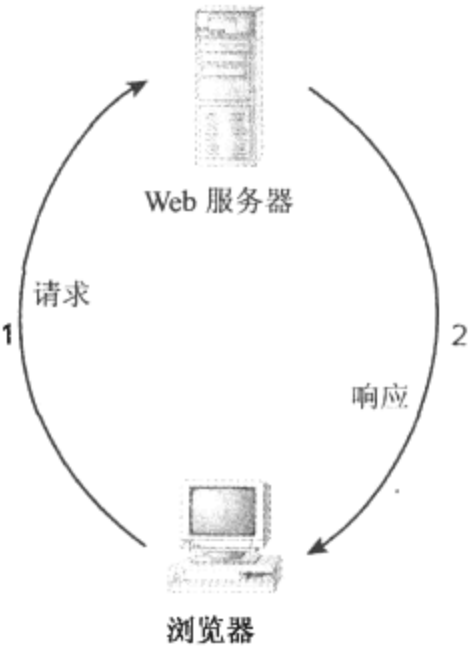


图 1-7

- **ASP.NET 服务器控件** 这些控件位于 ASPX 页面中，在处理它们时，它们会显示插在页面中的 HTML。在本章讨论了 HTML 之后，将介绍有关服务器控件的更多知识，第 4 章集中讨论了 ASP.NET 服务器控件。
- **编写代码** 正如在“试一试”练习中显示的那样，可以把代码直接嵌在页面中，如 Visual Basic .NET 或 C# 代码。此外，可以将代码放在单独的代码文件(称为 Code Behind 文件)中。该代码可以由运行库自动执行，或者基于用户的动作执行。无论采用哪种方式，代码的执行都会大大影响页面的显示方式，包括访问数据库、执行计算、隐藏或显示特定控件等。在第 5 章中将会介绍有关 Code Behind 文件的更多内容，还会详细地讨论 ASP.NET Web 页面程序的编写。

处理了页面并收集了页面的所有 HTML 后，就将它发送回浏览器。然后浏览器会读取该 HTML 并进行分析，最终显示出该页面。

HTML 对于显示 Web 页面很关键，下一节将概述 HTML。

1.3.1 HTML

HTML 是实际创建 Web 页面的语言，如今现有的每个 Web 浏览器都能理解这种语言。自从 20 世纪 90 年代初以来，它就成为了 World Wide Web 的驱动力量、Internet 处理 Web 页面的部分。HTML 文档是含有标记、文本和影响文本的附加数据的简单文本文件。

1. HTML 元素和标记

HTML 用尖括号间的文本指示内容在浏览器中如何显示。这种带有尖括号的文本称为标记(tag)：含有文本或其它内容的一对标记称为元素。再看一下上面的“试一试”练习中，通过打开浏览器中的页面的源代码窗口所看到的 HTML 如下所示：

```
<h2>Hello World</h2>
<p>Welcome to Beginning ASP.NET 4 on 10/30/2009 6:19:16 PM</p>
```

该示例的第一行含有一个带起始标记<h2>和结束标记</h2>的<h2>元素。此元素用来表示二级标题(如果在浏览器最终的源代码中向上滚动，就会看到<h1>元素)。注意，元素的结束标记和起始标记相似，只是前面多了个斜杠(/)：</h2>。这些起始标记和结束标记之间的所有文本都被看作是标题部分，因此被呈现为标题。在大多数浏览器中，这意味着标题会呈现为较大的字体。与<h2>标记类似，还有创建一直到 6 级标题的标记，比如<h1>、<h3>等。

在标题元素下面，可以看到一个<p>元素，它用来表示段落。<p>标记对中的所有文本都被看作是段落部分。默认情况下，浏览器呈现段落时会在下方留出一些空白，不过也可以忽略这个行为。

HTML 中有许多可用的标记；不可能在这里一一介绍。表 1-1 列出了一些最重要的标记，并说明了它们的用法。要了解关于 HTML 元素的完整列表，参见维护 HTML 标准的组织的站点：www.w3.org/TR/html401/index/elements.html。

表 1-1

标 记	说 明	示 例
<html>	用来表示整个页面的开始和结束	<html> ...All other content goes here </html>
<head>	用来表示包含页面数据的页面的特殊部分，包括其标题以及对外部资源的引用	<head> ... Content goes here </head>
<title>	用来定义页面的标题。这个标题将会显示在浏览器的标题栏中	<title> Welcome to Planet Wrox 4 </title>
<body>	用来表示页面体的开始和结束	<body> Page body goes here </body>
<a>	用来将一个 Web 页面链接到另一个页面	 Visit the Wrox site
	用来向页面中嵌入图像	
 <i> <u>	用来将文本格式化为粗体、斜体或下划线字体	This is bold text while <i>this text is in italic</i>
<form> <input> <textarea> <select>	用于描述允许用户向服务器提交信息的输入格式	<input type="text" value="Some Text" />
<table> <tr> <td>	这些标记用来创建含有表格的布局。<table>标记定义了整个表，而<tr>和<td>标记分别用来定义行和单元格	<table> <tr> <td>This is a Cell in Column 1</td> <td>This is a Cell in Column 2</td> </tr> </table>
 	这 3 个标记用来创建带有编号或项目符号的列表。和标记定义了列表的外观(可能是无序的，带一个简单的项目符号；也可能是有序的，带有编号)，而标记用来表示列表中的项	 First item with a bullet Second item with a bullet First item with a number Second item with a number

(续表)		
标 记	说 明	示 例
	这个标记用来包装和影响文档的其他部分。它作为内联文本出现，所以不会向页面添加换行符	<p>This is some normal text while this text appears in red</p>
<div>	与标记一样，<div>标记用来作为其他元素的容器。然而，<div>标记是块级元素，默认情况下它会使<div>元素后面出现显式的换行符	<div> This is some text on 1 line </div> <div> This text is put directly under the previous text on a new line. </div>

2. HTML 属性

表 1-1 中除了有 HTML 元素外，还显示了 HTML 特性。这些特性包含了一些改变特定元素行为方式的额外信息。例如，使用标记显示一个图像，src 特性定义图像的源代码。类似地，标记含有一个将文本改为红色的 style 特性。style 特性的值(color: red;)是层叠样式表(Cascading Style Sheet, CSS)的一部分，这个概念将在第 3 章详细介绍。对于 HTML 元素，在 W3C 的 Web 站点上有一个很长的可用特性列表：www.w3.org/TR/html401/index/attributes.html。

不需要记住所有这些元素和特性。在大多数情况下，VWD 会自动地生成它们。而在其他情况下，当需要手工输入它们时，VWD 会提供 IntelliSense 工具，帮助找到正确的标记或特性。第 2 章将介绍 IntelliSense 工具。

3. HTML 和 XHTML 的区别

除了 HTML 外，还有术语 XHTML。虽然两者的名称看起来非常相似，但是它们之间有一些有趣的区别，需要加以注意。XHTML 是用 XML(eXtensible Markup Language)语言重新表示的 HTML。XML 是一种通用的、用来描述数据的、基于文本与标记的语言，它也作为其他许多语言(包括 XHTML)的基础语言。

因此，XHTML 实际上很大程度上是用 XML 规则重写的 HTML。这些规则相当简单，而且大多数时候 VWD 都会帮助您作出正确决定，或者提供错误列表和关于如何修复的建议。

总是闭合标记

在 XHTML 中，如果用<p>开始了一个段落，就必须在页面后面的某个地方用</p>闭合该段落。对于没有结束标记的标记也是如此，比如或
(用来输入一个换行符)。在 XHTML 中，这些标记被写为自结束标记，其中结束标记中的斜杠直接嵌在标记自身中，比如在或
中。

标记和特性名称总是使用小写

XML 是区分大小写的，XHTML 通过强制所有标记采用小写来应用该规则。虽然标记和特性必须都是小写，但是实际值不必是这样。因此，前面显示 logo 图像的示例是完全有效的 XHTML，尽管图像名称中使用了大写的 L。

总是用引号括起特性值

每当在标记中写特性时，都要用引号将它的值括起来。例如，在写标记和它的 src 属性时，应该这样写：

```

```

而不是如下这样：

```
<img src=Logo.gif />
```

注意，也可以用单引号括起特性值，如下面这个示例：

```
<img src='Logo.gif' />
```

有时还有必要嵌套单引号和双引号。当有些特殊 ASP.NET 语法要求使用双引号时，应当用单引号括起特性的值：

```
<asp:Label ID="TitleLabel" runat="server" Text='<%# Eval("Title") %>' />
```

在本书的其他章节中将不时地出现这种语法。

为了保持一致，本书在最终出现在客户端的所有 HTML 中可能用到引号的地方都使用双引号。

正确地嵌套标记

在写嵌套标记时，确保首先闭合最后打开的内部标记，然后闭合外部标记。考虑这个示例，即同时用粗体和斜体格式化一段文本：

```
<b><i>This is some formatted text</i></b>
```

注意<i>标记是在标记之前闭合的。交换结束标记的顺序会导致无效的 XHTML：

```
<b><i>This is some formatted text</b></i>
```

总是向页面中添加一个 DOCTYPE 声明

DOCTYPE 给出了期望的关于 HTML 类型的浏览器信息。默认情况下，VWD 向页面中添加的 DOCTYPE 是 XHTML 1.0 Transitional。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

DOCTYPE 大大地影响了浏览器(如 Internet Explorer)呈现页面的方式。VWD 中的 XHTML 1.0 Transitional 的默认 DOCTYPE 很好地保证了有效标记和页面在各主流浏览器中显示的一致性。



提示： 如果想了解关于 XHTML 的更多信息，请参考清华大学出版社引进并出版的《Web 编程入门经典——HTML、XHTML 和 CSS(第 2 版)》一书。

除了 HTML 之外，ASP.NET Web 页面也可能包含其他标记。大多数页面上都有一个或多个 ASP.NET 服务器控件，给出了一些附加的功能。下一节将简要介绍这些 ASP.NET 服务器控件，第 4 章再深入介绍它们。

1.3.2 初识 ASP.NET 标记

在某种程度上，ASP.NET 服务器控件的标记与 HTML 的标记很相似。它也像 HTML 一样用尖括号和结束标记表示标记、元素和特性。然而，它们之间还是有一些不同之处的。

对于初学者来说，大部分 ASP.NET 标记都是以 asp:前缀开头。例如，ASP.NET 中的按钮类似于下面这样表示：

```
<asp:Button ID="Button1" runat="server" Text="Click Me" />
```

注意此标记是用斜线字符(/)自闭合的，所以不必另外输入结束标记。

还应注意到的另一件事是标记和特性名不必都是小写。因为服务器上有一个 ASP.NET 服务器控件，所以不必在客户端的浏览器中使用 XHTML 规则。然而，当要求服务器控件将它的 HTML 发送到配置为输出 XHTML 的页面时，它就会应用 XHTML 规则。因此，当在浏览器中呈现为 XHTML 时，这个按钮的代码类似于如下这样：

```
<input type="submit" name="Button1" value="Click Me" id="Button1" />
```

注意整个标记及它的属性符合 XHTML 标准。将服务器控件转换为它的 HTML 表示的过程与前面看到的代码显示当前日期的过程相似。服务器控件由 ASP.NET 处理程序在服务器中进行处理。结果产生的 HTML 被发送到浏览器并在浏览器中显示。

我们已经介绍了 ASP.NET 页面和它生成的 HTML 的基础知识，接下来介绍 VWD。学会使用这个应用程序及其众多的工具和窗口，是构建有趣、漂亮且实用的 Web 站点的重要一步。

1.4 IDE

VWD 是目前为止构建 ASP.NET Web 页面使用最为广泛、功能最为丰富的集成开发环境(IDE)。缩略词 IDE 是指构建复杂 Web 应用程序所需的所有独立工具都集成在一个环境中。VWD 不需要在文本编辑器中编写代码、在命令行中编译代码、在单独的应用程序中编写 HTML 和 CSS 以及在另一个应用程序中管理数据库；它允许在同一个环境中执行所有这些任务及更多其他的任务。由于不必一直进行工具转换，因此这样一来效率得到了提高，并且因为许多内置工具的工作方式都是一样的，因此使得人们能够更容易地学习 VWD 的新功能。

1.4.1 主开发区

为了熟悉 VWD 的界面中含有的众多工具，可以参看图 1-8。它显示了用 VWD 创建第一个 Web 站点后所看到的屏幕，不过现在它突出显示了部分最重要的屏幕元素。如果已经熟悉了 Visual Web Developer 之前的版本，就可以跳过该部分并在本章后面的“试一试”练习中学习有关知识。

如果安装的是 Visual Studio 以前的版本，屏幕可能会与图 1-8 有所不同，因为 Visual Studio 2010 能够导入老版本的设置。

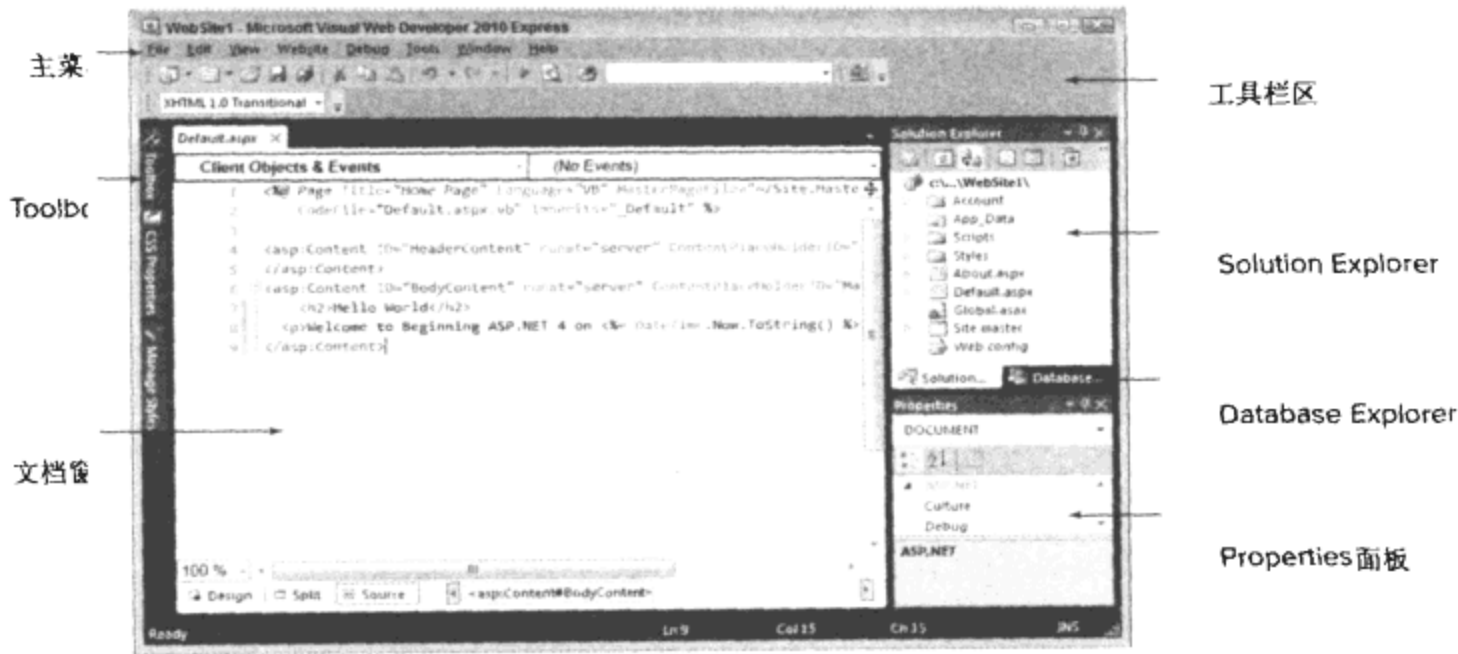


图 1-8

1. 选择开发配置文件

Visual Web Developer Express 面向 ASP.NET 开发新手以及经验丰富的 Web 开发人员，因此它提供了 3 种不同的开发配置文件供开发人员选择：基本设置(Basic Settings)、代码优化(Code Optimized)以及专家设置(Expert Settings)。在基本设置模式下，许多不常使用的菜单项会被隐藏或放在自己的子菜单中。代码优化模式对于纯编码会话来说很有用处，而 VWD 的这些设计特色往往是您不感兴趣的。该模式隐藏了诸如 Toolbox 和 Properties 面板(如图 1-8 所示)这样的项。而专家设置模式允许访问 VWD 的全部功能。可以使用 Tools | Settings 菜单来转换不同的设置。在本书中，假设使用专家设置模式。本书开头所介绍的全部功能可能不会都用到，但是到最后肯定会用到大多数的功能。由于菜单项可以根据选择的模式改变其位置，因此需要使用专家设置模式，以便更简单地使用特定的菜单项或功能。

2. 主菜单

在这个应用程序的上方，Windows 标题栏的下面，可以看到主菜单。此菜单栏中包含的菜单项您一定很熟悉，因为在很多其他的 Windows 应用程序中都可以找到这些菜单项，如 File、Edit、Help 菜单，以及一些 VWD 中特有的菜单，如 Website 和 Debug 菜单。该菜单可以根据执行的具体任务动态地改变，因此在使用应用程序的过程中您会发现某些菜单项有时出现、有时消失。可以通过使用 Help | Manage Help Settings 菜单进行在线配置和离线帮助。离线帮助需要首先安装该应用程序，而在线帮助则需要连接到 Internet。

3. 工具栏区

在菜单的下方，可以看到工具栏区，在该区域显示了不同的工具栏，从而可以快速地访问 VWD 中的大部分常见功能。在图 1-8 中，只启用了两个工具栏，但是在面向特定任务的场景中，VWD 会随之打开很多可以使用的其他工具栏。有些工具栏会在执行需要特定工具栏出现的任务时自动地出现，但是也可以根据自己的喜好启用或禁用工具栏。要启用或禁用工具栏，只需右击现有工具栏

或菜单栏，从出现的菜单中选择工具栏即可。

4. Toolbox

在主屏幕的左边，可以看到折叠在 VWD 边缘的 Toolbox 选项卡。如果把鼠标指针悬停在该选项卡上，Toolbox 就会展开，这样就能看到它包含的内容。如果单击 Toolbox(或者有小钉图标的其他面板)右上角的小钉图标，它就会锁定 IDE，使它保持打开状态。

与菜单栏和工具栏一样，Toolbox 会自动更新，以显示与正在执行的任务相关的内容。在编辑标准 ASPX 页面时，Toolbox 会显示可用于页面的许多控件。可以简单地从 Toolbox 中拖动一个控件，然后把它放到希望在页面中出现的位置上。这些控件将在第 4 章详细讨论。

Toolbox 包含多个类别，其中的工具可以根据意愿展开和折叠，以便找到正确的工具。也可以重新排列列表中工具的顺序，从 Toolbox 中添加和删除工具，甚至可以向其中添加自己的工具。关于定制 IDE 将在本章后面讨论。

如果在屏幕上看不到 Toolbox，则可以按 Ctrl+Alt+X 组合键打开它，或者从 View 菜单中选择 Toolbox 命令来打开(假设已经选择了 Tools | Settings 菜单中的专家设置选项)。

Toolbox 选项卡下面还有两个选项卡：CSS Properties 和 Manage Styles。这两个选项卡都会在第 3 章详细讨论。

5. Solution Explorer

在屏幕右边，可以看到 Solution Explorer。Solution Explorer 是一个重要窗口，因为它提供了组成 Web 站点的文件概览。Solution Explorer 没有把所有文件都放在一个大文件夹中，而是将文件存储在单独的文件夹中，创建了一个有逻辑且有组织的站点结构。可以用 Solution Explorer 向站点中添加新文件，使用拖放功能或者剪切粘贴功能来移动现有文件，从项目中重命名文件以及删除文件等。Solution Explorer 的大部分功能都隐藏在它的右击菜单中，该菜单根据在浏览器窗口中右击的项目而改变。

在 Solution Explorer 的上方有一个小工具栏，可以用来快速访问与 Web 站点相关的一些功能：包括打开选中项目的 Properties 面板，刷新 Solution Explorer 窗口，嵌套相关文件的选项，以及用来复制和配置 Web 站点的两个按钮。大多数这些功能在本书后面都会讨论到。

可以通过从主菜单中选择 View | Solution Explorer 或者按 Ctrl+Alt+L 组合键来访问 Solution Explorer。

6. Database Explorer

这个窗口隐藏在图 1-8 所示的 Solution Explorer 后面，通过它可以操作数据库。如果使用的是 Visual Studio 的商业版本，如 Visual Studio 2010 Professional，那么这个窗口就称为 Server Explorer，它可能位于屏幕左侧。

Database Explorer 将在关于数据库的那几章(从第 12 章开始)详细讨论。

7. Properties 面板

用 Properties 面板可以查看和编辑 Visual Studio 中的许多项目的属性，包括 Solution Explorer 中的文件、Web 页面上的控件、页面本身的属性及其他更多内容。这个窗口会不断地更新以反映选中

的项目。按 F4 键可以快速打开 Properties 面板。这个快捷键还可以用来强制 Properties 面板显示选中项目的详细信息。

8. 文档窗口

应用程序中间的文档窗口是主要区域。大部分动作都在这里发生。可以用文档窗口来操作很多不同的文档格式，包括 ASPX 和 HTML 文件、CSS 和 JavaScript 文件、VB 和 C#的代码文件、XML 和文本文件，甚至图像文件。此外，用这个窗口还可以管理数据库、创建站点的副本，并在内置的微型浏览器中浏览页面，等。

在图 1-8 所示的文档窗口下方可以看到 3 个按钮，分别是 Design、Split 和 Source。在操作含有标记的文件(如 ASPX 和 HTML 页面)时，这些按钮会自动出现。它允许打开页面的 Design View(让您对页面在浏览器中的样子有个概念)、它的 Markup View(HTML 和其他标记)，或者同时打开这两者。其工作原理将在第 2 章详细解释。现在重要的是要知道，可以通过单击相应的按钮在 Markup 视图、Split 视图和 Design 视图之间切换。Markup 视图也常称为 Source 视图或 Code 视图窗口。然而，为了避免与用于编辑 Code Behind 文件的代码编辑器混淆，本书只采用 Markup 视图这个术语。

默认情况下文档窗口是一个带选项卡的窗口，这意味着它能驻留多个文档，各个文档通过选项卡用窗口上方显示的文件名进行区分。各选项卡的右击菜单中包含使用该文件的一些有用的快捷键，包括保存与关闭文件，以及在 Windows Explorer 中打开该文件的父文件夹。

要在文档之间进行切换，可以按 Ctrl+Tab 组合键，或者单击文档窗口右上角的下拉箭头，该文档窗口邻近 Solution Explorer，如图 1-8 所示。单击下拉箭头会显示出一个打开文档的列表，因此可以轻而易举地从中选择要打开的文档。

切换文档的另一种方式是按下 Ctrl+Tab 组合键，然后按住 Ctrl 键。在弹出的窗口中，可以选择要在右手边的那一栏中使用的一个文档。然后可以在具有打开的文档的列表中向上或向下移动光标。这样选择正确的文件就变得相当容易。

在同一个对话框中，可以看到一个包含所有活动工具窗口的列表。单击这个列表中的一个窗口，可以将它显示在屏幕上；如有必要，还能将它移到其他窗口的前面。

9. Start Page

每次启动 VWD 时，Start Page 都要加载到文档窗口中。有了 Start Page，就可以快速地创建新的 Web 站点或者打开现有站点和其他项目。Start Page 还提供了一些有关 Web 开发的信息的链接。

为了对如何使用所有这些窗口有一个感性的认识，下面的“试一试”练习将会介绍如何构建一个包含一些 ASP.NET 服务器控件的简单 Web 页面。

试一试

创建第一个 ASP.NET Web 页面

这个“试一试”练习将创建一个只有一个页面的新 Web 站点，该页面中包含若干个 ASP.NET 服务器控件。该练习将介绍如何使用文档窗口和 Solution Explorer 等窗口，以及如何使用 Toolbox 和 Properties 面板向页面中添加 ASP.NET 服务器控件并修改它们的外观。

- (1) 确保启动了 Visual Web Developer 2010。
- (2) 如果使用的是 Express 版本，则选择 Tools | Settings 并打开专家设置中的开发人员配置，以

便访问 VWD 的全部功能集。

- (3) 在 File 菜单中选择 New Web Site。如果使用的是 Visual Studio 的商业版本，则可能需要选择 File | New | Web Site。这样会打开 New Web Site 对话框。
- (4) 在该对话框中，确保选中的是 ASP.NET Empty Web Site，而不是先前练习中使用的 ASP.NET Web Site。一定要从 Web Location 的下拉列表中选择 File System。然后单击 OK 创建新站点。
- (5) 接着，右击 Solution Explorer 中新的 Web 站点。确保单击了最上面描述类似这样内容的元素：C:\.\WebSite2\。图 1-4 中突出显示了该元素。从显示的上下文菜单中选择 Add New Item 命令。
- (6) 在出现的新窗口中，单击 Web 窗体并输入 ControlsDemo 作为名称。当单击 Add 按钮时，它会自动添加扩展名 ASPX。可以允许对话框中的其他设置保留其默认值。该页面应该能在 Markup 视图中打开，显示默认的 HTML，如<html>、<head>、<title>和<body>元素，这些元素是在创建新页面时由 Visual Web Developer 自动添加的。
- (7) 单击文档窗口下方的 Design 按钮将页面切换到 Design 视图。
- (8) 如果 Toolbox 还没有打开，则按 Ctrl+Alt+X 组合键打开它，或者将鼠标悬停在 Toolbox 选项卡上来显示它，然后单击锁定图标使 Toolbox 一直可见。从 Toolbox 中将一个 TextBox 和一个 Button 拖放到页面 Design 视图中的阴影区域内。最后应当看到一个类似于图 1-9 所示的页面。

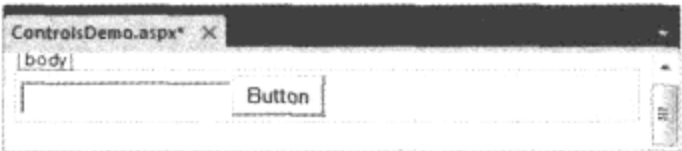


图 1-9

- (9) 右击 Design 视图中的 Button 按钮并选择 Properties 命令。在 Properties 面板中，定位到 Appearance 下面的 Text 属性(如图 1-10 所示)并将它从 Button 改为 Submit Information。一旦按下了 Tab 键或者 Properties 面板之外的某处单击了鼠标，页面的 Design 视图就会更新，并在按钮上显示新文本。

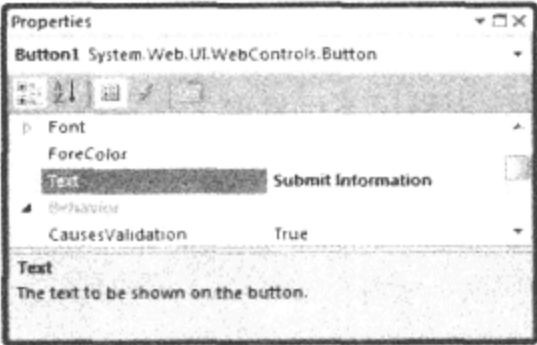


图 1-10

- (10) 按 Ctrl+F5 组合键在默认浏览器中打开该页面。注意，没有必要显式地保存对页面的修改(不过用快捷键 Ctrl+S 经常保存会有好处的)。按 Ctrl+F5 组合键运行页面后，VWD 就会自动保存对打开文档的所有修改。

注意：如果不喜欢这种行为，可以进行修改。在主菜单的 Options 对话框中修改它，这个对话框可以从 Tools 菜单中访问。一定要确保选中了 Show All Settings，然后打开 Projects 和 Solutions 节点，并选择 Build 和 Run。在 Before Building 列表中，当在浏览器中打开一个页面时可以修改 VWD 的行为。

(11) 在文本框中输入一些文本，然后单击这个按钮。注意，当重新加载页面后，文本仍然会显示在文本框中。如果没有显示，则很可能是因为没有为这个按钮编写任何代码。

工作原理

当从 Toolbox 中拖动 Button 和 TextBox 到页面的 Design 视图中时，VWD 会自动在 Markup 视图中添加相应的代码。类似地，当在 Properties 面板中修改按钮的 Text 属性时，VWD 会自动更新 Markup 视图中控件的标记。如果不使用 Properties 面板，也可以在 Markup 视图中 Text 属性的引号之间直接输入文本。

修改了 Text 属性之后，页面在 Markup 视图中应包含如下代码：

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Submit Information" />
```

当按 Ctrl+F5 组合键查看浏览器中的页面时，Web 服务器会收到请求，页面则由 ASP.NET 运行库处理，为页面产生的最终的 HTML 将被发送到浏览器。

当输入一些文本并单击按钮时，就会重复同样的过程：Web 服务器接收请求，处理页面，将结果发送回浏览器。当单击该按钮时，就引发了一个回发(postback)，此时页面中的所有信息(如在文本框中输入的文本)都会被发送回服务器。ASP.NET 通过再次呈现页面来对回发作出响应。然而，这次它用发送到页面的值预先填充了控件，如 TextBox。

使用浏览器的 View Source 命令查看页面最终的 HTML 代码(如果已经关闭了，则按 Ctrl+F5 组合键从 VWD 中重新运行页面)。看到的代码应类似于下面这样：

```
<input name="TextBox1" type="text" value="Hello World" id="TextBox1" />
<input type="submit" name="Button1" value="Submit Information" id="Button1" />
```

从前面的示例中可以看出，最终的 HTML 与原始 ASPX 标记有相当大的区别。

回发是 ASP.NET 中的一个重要概念，在第 4 章和第 9 章中将更详细地介绍它们。

在 VWD 中驻留的窗口和工具面板远远不止目前为止所提到的这些。下一节将简要介绍一些构建 ASP.NET Web 页面时最常用到的窗口。如果使用的是专家设置模式，则这里提到的所有窗口都可以从 VWD 的主 View 菜单中访问。

1.4.2 信息窗口

除了在启动 VWD 时见到的默认窗口外，VWD 中还有很多可用窗口。在本书的其余部分，将运用到这些窗口，现在要先介绍几个重要的窗口。您可以访问接下来讨论的主 View 菜单中的所有窗口。

1. Error List

Error List 提供了一个列表，列出了当前因为某种原因在站点中被中断的内容，包括 ASPX 或 HTML 文件中的错误标记，以及 VB 或 C#文件中的编程错误。这个窗口甚至可以显示 XML 和 CSS 文件中的错误。这个错误列表显示了 3 类消息——Errors、Warnings 和 Messages，它们分别表示不同的问题严重程度。图 1-11 显示了 CSS 和 XHTML 有问题的页面的错误列表。

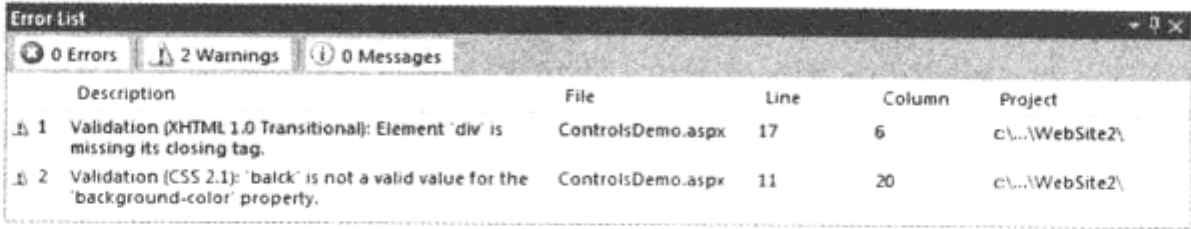


图 1-11

2. Output 窗口

当使用 Build 菜单构建站点时，Output 窗口会提示是否构建成功。如果构建失败，如存在编程错误，那么 Output 窗口就会指出构建失败的原因。在 Visual Studio 的商业版本中，Output 窗口还用来输出其他信息，包括外部插件程序的状态。构建或编译 Web 站点将在本书的第 19 章中讨论，该章节主要介绍 Web 站点的部署。

3. Find Results 窗口

当开始管理站点的内容时，VWD 的 Find 和 Replace 功能是非常有用的工具。在工作中经常需要替换当前文档甚至整个站点中的某些文本。Find in Files(Ctrl+Shift+F)和 Replace in Files (Ctrl+Shift+H)都会在 Find Results 窗口中输出它们的结果，如图 1-12 所示：

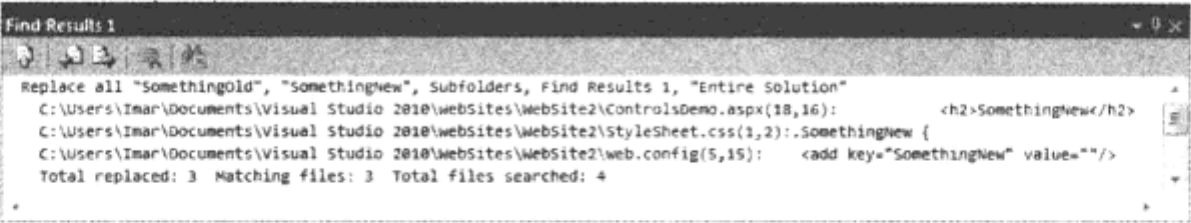


图 1-12

因为同时打开多个信息窗口可能会占用宝贵的屏幕空间，所以最好停靠(dock)它们。这样，一次就只看到一个窗口，且仍然能够快速访问其他窗口。下一节将解释如何定制 IDE，包括如何停靠窗口。

1.5 定制 IDE

虽然 VWD 的标准配置和它的工具窗口相当有用，但是也可以根据自己的偏好定制 IDE。有时可能需要把窗口重新安排到容易找到的位置，有时可能希望打开其他经常用到的窗口。VWD 是可以完全自定义的，而且 IDE 的任何细节都可以进行调整。下一节将介绍如何完成大多数常见的自定义任务。

1.5.1 重新排列窗口

可以在主 IDE 中通过拖放重新排列窗口。只需简单地按住窗口的标题栏或者它下面的选项卡，并沿着新位置的方向拖动即可。开始拖动窗口时，会看到 VWD 提供的关于窗口将停在何处的可视化线索(如图 1-13 所示)。

如果按住指示器边上的 4 个方块指示器之一拖动窗口，VWD 就会显示一个预览，表明窗口将如何停靠在一个现有窗口旁边。当放下它时，窗口就会出现在它的新位置上。如果将窗口放在大指

示器中间的方块上，窗口就会停靠在那个窗口的位置，并与之共享相同的屏幕空间。每个窗口都有自己的选项卡，这一点从图 1-13 下方的窗口上可以看出来。

除了在 IDE 中将窗口与其他窗口停靠在一起外，还可以使之成为浮动窗口。要把停靠窗口改为浮动窗口，可以将窗口从它的当前位置拖走，并放在 IDE 中的任何位置，不需要单击屏幕中的某个可视化线索。也可以从主菜单中选择 Window | Float 命令使其浮动。

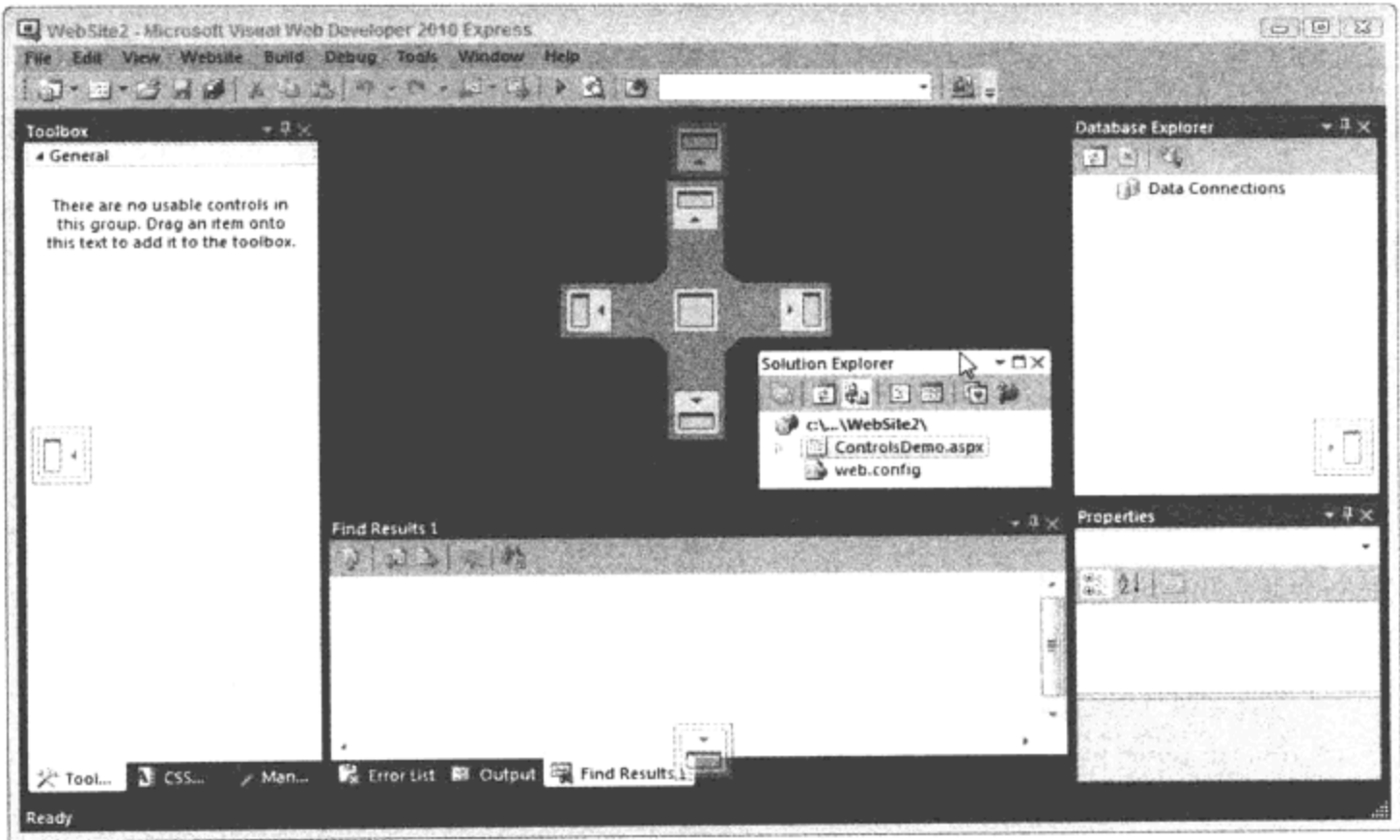


图 1-13

要将浮动面板重新恢复到它原来的停靠位置，从主菜单中选择 Window | Dock 命令即可。请确保不是选择 Dock 作为工具窗口(如工具箱、Solution Explorer 或以主文档窗口结束)的标记文档。由于两个窗口将共享相同的空间，因此这使得与打开的文件一起使用这些工具窗口变得很困难。

1.5.2 修改 Toolbox

Toolbox 也是可以修改的。默认情况下，Toolbox 按照字母顺序对控件进行分类，但是也可以使用拖放工具对它们重新进行排列。为此，需要打开 Toolbox(按 Ctrl+Alt+X 组合键)，把一个控件(如 Standard 类别下的 TextBox)拖到一个不同的位置上。也可以通过右击它们并从上下文菜单中选择 Delete 命令把它们从 Toolbox 中删除。不要担心控件会永久失去，因为可以从同样的菜单中选择 Reset Toolbox 命令来重置 Toolbox。

还可以向 Toolbox 中添加自己的控件。这个功能最常用于代码段。只要简单地突出显示文档窗口中的一些文本或代码，并把它拖到 Toolbox 中。然后可以右击该项目并选择 Rename Item 命令给它起一个更有意义的名称，以方便识别。

为了避免 Toolbox 与自己的代码段混淆，可以为它们创建一个单独的类别。可以通过从 Toolbox 的右击菜单中选择 Add Tab 命令来完成。输入一个名称然后按下 Enter 键，Toolbox 选项卡就可以使用了。

在下面的“试一试”练习中，可以修改 VWD IDE，将它定制为自己喜欢的样子。

试一试

定制 IDE

这个练习是为了实践打开和重新排列 Visual Web Developer IDE 中的窗口。不用害怕使 IDE 陷入混乱。本章稍后将介绍如何将 IDE 重置为第一次打开时的样子。

(1) 如果关闭了在上一个“试一试”练习中建立的 Web 站点，就再打开它，或者使用 File 菜单新建一个站点。

(2) 从 View 菜单中选择 Error List 命令打开 Error List 窗口。如果没有看到 Error List 选项，则首先选择 Tools | Settings | Expert Settings。注意默认情况下，它停靠在文档窗口的下方。

(3) 还是从 View 菜单中，选择 Task List 命令。默认情况下，它会停靠在与 Error List 相同的空间中，两个窗口的选项卡彼此相邻。

(4) 单击 Task List 的选项卡，在按住鼠标按钮的同时，沿着文档窗口的方向将 Task List 拖离它的位置。释放后，它就会显示为 IDE 中的浮动窗口。要恢复该窗口，右击它的标题栏并选择 Dock 即可。注意这个选项卡会返回到同样的选项卡组中，但是可能会出现在不同的位置上。要改变选项卡在选项卡组中出现的位置，把选项卡拖到其他选项卡上，并在所需的位置处释放。

(5) 如果愿意，可以对 IDE 中默认可见的其他窗口或从 View 菜单下找到的窗口重复上面的步骤。花一些时间来熟悉所有不同的窗口，并练习如何把它们排列在屏幕上。由于在本书的其余部分将会频繁地用到这些窗口，所以最好先熟悉它们的位置。

(6) 接下来，双击打开 Solution Explorer 中的 ControlsDemo.aspx 页面(或者如果创建了一个新的 Web 站点，则首先添加一个新的 ASPX 页面)。当该页面打开时，Toolbox 会自动可视。如果不可视，则按 Ctrl+Alt+X 组合键打开它。

(7) 右击 Toolbox 并选择 Add Tab 命令。输入 HTML Fragments 作为它的新名称并按下 Enter 键。这样就向 Toolbox 中添加了一个新类别，它的行为与所有其他类别一样。

(8) 当文档窗口在 Markup 视图中显示了 ASPX 页面后，则在起始<div>标记后直接输入<h1>。注意 VWD 会自动地插入结束标记</h1>。最后在 Markup 视图中看到的代码如下所示：

```
<form id="form1" runat="server">
  <div>
    <h1></h1>
  </div>
```

(9) 突出显示起始和结束<h1>标记，然后将 Markup 视图窗口中的选项拖到在步骤(7)中创建的新 Toolbox 选项卡上。该选项会显示这样的 Text: <h1></h1>。

(10) 右击刚刚创建的 Toolbox 项目，选择 Rename Item 命令，并输入 Heading 1 作为其名称。

(11) 再次把光标放到文档窗口中，按下 Ctrl+D 后直接按 Ctrl+K，以便格式化文档窗口中的文档。另一种方法是从主菜单中选择 Edit | Format Document 命令来格式化文档。这是根据在 Text Editor 选项对话框中设定的规则对文档进行格式化的。还可以对一些其他的文档类型进行格式化，包括 C#、VB.NET 代码以及 XML 文件。

从现在起，每当在 Markup 视图中的文档需要一个标题时，只需把光标放在文档窗口中希望标题出现的地方，然后双击 Toolbox 中适当的标题即可。

工作原理

该“试一试”练习中的大部分步骤都是容易理解的。首先打开几个构建 Web 站点时经常需要的

窗口。然后使用 IDE 的拖放功能将窗口的布局重新排列为个人偏爱的样子。

然后将几个 HTML 片段添加到 Toolbox 的自定义选项卡中。当把任何标记拖到 Toolbox 中时，VWD 都会为它创建一个含有选定标记的 Toolbox 项目。每当在页面中需要该标记的一个副本时，只要双击该项目或把它从 Toolbox 中拖到 Markup 视图窗口中即可。这样对于频繁用到的 HTML 片段可以节省很多时间。该技术一般用于较大的代码块，而对于类似于<h1>这样的标记，VWD 则有一个更好的工具，即 Code Snippets，稍后会在本书中介绍该工具。

最后，使用 VWD 的文档格式化选项改变文档中代码的布局。这样有助于保持代码的有序性且使其更易于阅读。通过使用 Tools | Options 命令打开的可访问的选项对话框，可以完全改变代码的格式化方式。然后扩展 Text Editor | HTML | Formatting 命令的路径并单击 Tag Specific 选项。

除了窗口布局和 Toolbox 外，VWD 允许在 IDE 中进行更多的定制。下一节将讲解如何定制其他 3 个重要的 IDE 功能：文档窗口、工具栏和键盘快捷键。

1.5.3 定制文档窗口

可视化的 Web Developer 大大增加了文档窗口中文本显示方式的灵活性。可以修改一些属性，如字体大小、字体颜色，甚至是文本的背景色。可以通过选择 Tools | Options 命令来访问 Font 和 Colors 设置，确保选中了对话框底部的 Show All Settings 选项，然后选择 Environment | Fonts and Colors 命令。

在文档窗口中，笔者喜欢定制 tab 的大小，它控制缩进代码时插入的空格数目。要修改 tab 的大小，选择 Tools | Options 命令，然后在 Text Editor 下方选择 All Languages | Tabs 选项。如果看不到这个选项，那么要先选择下方的 Show All Settings 选项。通常将 Tab Size 和 Indent Size 都设置为 2，让 Tab 面板中的其他设置保持不动。笔者还喜欢定制在 HTML 元素前后换行符的数目。选择 Text Editor | HTML | Formatting 命令，然后单击 Tag Specific 选项，就可以使用 Options 窗口完全控制中断的行的数目。

除了将 Tab Size 设置为 2 以及在一些 HTML 元素周围设置的换行符的数目之外，本书所有的屏幕截图显示的都是 Visual Web Developer 的默认配置。

1.5.4 定制工具栏

工具栏可以通过 3 种方式定制：显示或隐藏内置工具栏、在现有工具栏上添加或删除按钮、使用经常用到的按钮创建自己的工具栏。

1. 启用与禁用工具栏

可以通过右击任何现有工具栏或者菜单栏禁用和启用工具栏，然后从列表选择一个合适的项目。一旦显示了工具栏，就能用它的工具栏左边的拖动网格把它拖到工具栏区的新位置。

2. 编辑现有工具栏

如果感觉现有工具栏缺少了一个重要的按钮，或者工具栏中包含一个几乎用不到的按钮，则可以定制工具栏上的按钮。要做到这一点，右击任何工具栏或菜单栏，并选择 Customize 命令，然后

切换到 **Commands** 选项卡并从工具栏的下拉菜单中选择希望修改的工具栏。通过右边的 **Command** 按钮，可以添加新的命令并删除已有的命令，或者改变命令的顺序。

如果希望将工具栏移动到窗口的左侧、右侧或底端，则可以切换到定制窗口的工具栏选项卡上，选择工具栏并单击 **Modify Selection**。

3. 创建自己的工具栏

如果想将经常用到的功能组合在一起，那么最好创建自己的工具栏。要创建一个新工具栏，用上一节介绍的方法打开定制窗口。单击 **New** 按钮并输入工具栏的名称。然后切换到 **Commands** 选项卡，用与修改现有工具栏一样的方式修改自己的工具栏。

1.5.5 定制键盘快捷键

很多开发人员喜欢的另一个设置是修改键盘快捷键。使用键盘快捷键是节省时间的好方法，因为它们允许用一个简单的键盘命令来执行任务，而不需要拿起鼠标选择菜单的正确项目。要修改键盘快捷键，选择 **Tools | Options**，展开 **Environment**，然后单击 **Keyboard** 按钮。定位到命令列表中要修改快捷键的命令。由于这个列表中含有许多项目，因此可以通过输入命令的几个字母来过滤列表。例如，在包含字段的 **Show** 命令中输入 **print** 就会给出所有与打印相关的命令。

下一步，在 **Press** 快捷键字段中，输入一个新的快捷键并单击 **Assign**。VWD 允许为一个命令输入两个快捷键。例如，可以将命令 **Close All Documents** 绑定到命令 **Ctrl+K**、**Ctrl+O** 上。要执行这个命令，需要快速而连续地同时按下组合键。虽然两个快捷键看起来不太必要，但是它大大增加了可用快捷键的数目。

1.5.6 重置修改

如果觉得尝试了大量的自定义选项而弄乱了 VWD，也不要担心。有多种办法可以把 VWD 恢复到它以前的状态。

1. 重置 Window 布局

这个功能可以从 **Window** 菜单中访问，它用来将所有窗口重置为第一次启动 VWD 时的位置。如果误放了太多窗口且最后得到的 IDE 比较混乱，那么这个命令就比较有用。

2. 重置 Toolbox

如果误删了 **Toolbox** 中的项目，或者删除了整个选项卡，可以通过右击 **Toolbox** 并选择 **Reset Toolbox** 命令将 **Toolbox** 重置为它的原始状态。在使用这个命令前需要慎重考虑，因为它也会删除您所有的自定义代码段。

3. 重置所有设置

如果您跟着前面的“试一试”练习做了，而且又进行了一些自定义设置，那么 IDE 现在很可能

是这两种状态之一：要么看起来就是所希望的样子，要么看起来完全混乱了。对于后面那种情况，您需要先学习一下如何轻松地消除混乱状态。

要将所有的 VWD 设置完全恢复到它们刚安装后的状态，请根据所使用的 Visual Web Developer 版本，选择 Tools | Settings | Import and Export Settings 命令，或者选择 Tools | Import and Export Settings 命令。接下来，选择 Reset All Settings 选项并单击 Next 按钮。如果愿意，可以为现有设置创建一个备份；否则，就选择 No，Just Reset Settings 选项。此时，会打开另一个界面，允许在众多的设置集合中进行选择。要选择专家设置或 Web 开发，因为这些选项提供了对本书中涉及的所有功能的访问。最后，单击 Finish 按钮。这个动作会导致所有设置重置为它们的默认值，包括窗口布局、工具箱及工具箱定制、快捷键以及可能在 VWD Options 对话框中修改了的所有内容。因此，只有当确实想要 VWD 的一个全新配置时，才使用这个命令。

了解了关于 ASP.NET 页面和 VWD 的一些基本知识后，就可以进行一些实际操作了。在第 2 章中，将更详细地介绍如何创建 ASP.NET Web 站点和 Web 页面。您将学会如何以一种合乎逻辑的、结构化的方式组织您的站点，如何将许多不同类型的文件添加到站点中，如何使用它们，以及如何链接到站点中的页面。

但是，在学习第 2 章之前，还需了解一个更重要的话题：伴随本书的示例应用程序。

1.6 示例应用程序

构建 Web 站点是本书的宗旨，因此很有必要先了解一下本书的完整而实用的示例站点，我们将用它来展示 ASP.NET 的很多功能。

本书将构建的这个站点叫做 Planet Wrox，这是一个为对音乐感兴趣的人创建的在线社区。这个站点向它的访问者提供了下列功能：

- 评论管理员在站点上发布的 CD 和音乐会。
- Gig Pics 部分，它是一个在线相册，用户可以共享在音乐会上拍的照片。
- 在站点提供的不同图形化主题之间切换，这样就可以在不更改站点内容的情况下改变站点的外观。
- 音乐首选项，它影响用户在站点上看到的信息。
- 访问注册用户的奖励内容。

从管理员的角度(即作为站点拥有者的您)，站点允许做下列事情：

- 添加和维护评论。
- 管理系统中不同的音乐类型。
- 管理站点访问者创建的相册。

图 1-14 显示了 Planet Wrox 的主页。

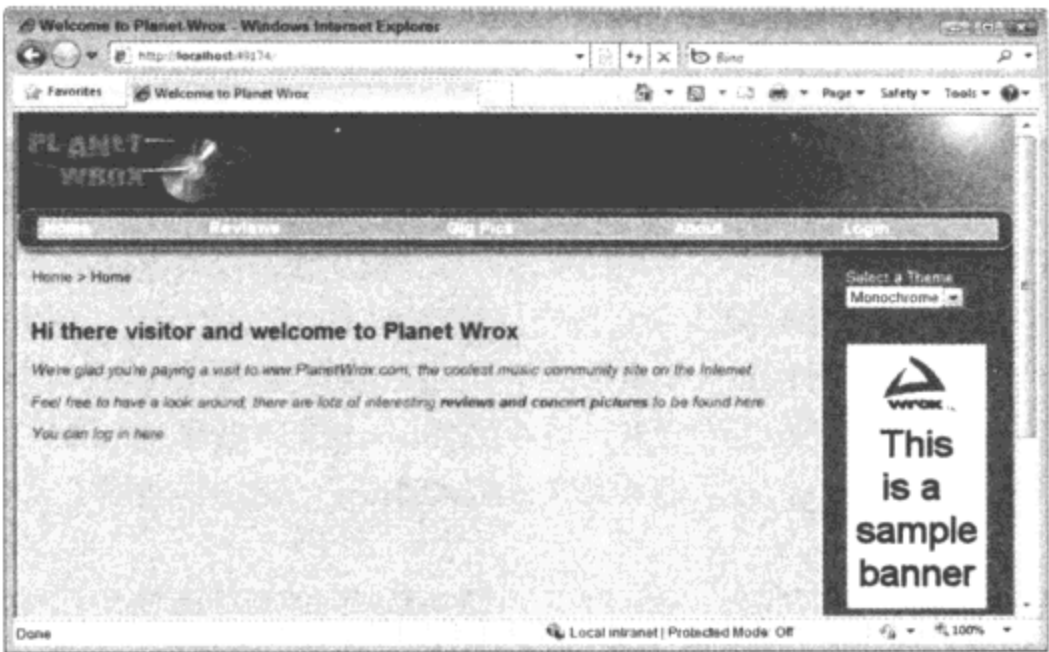


图 1-14

图 1-15 显示了 Planet Wrox 的另一个页面，只是应用的主题不同。该页面允许用户输入他们的个人信息并根据他们喜爱的音乐类型指定首选项。



图 1-15

在 www.PlanetWrox.com 上可以找到一个站点在线运行示例。在该站点上可以从用户的角度使用站点。

也可以从 Wrox 网站 www.wrox.com 上下载该示例应用程序以及本书所有其他示例所需的源代码。

学习完本书，就能在其他 Web 站点中构建该示例站点的所有功能(甚至更多)。这听起来似乎相当复杂，但不用太担心。笔者会逐步提供指导，从应用程序开始一直介绍到最后一个功能。只要保

持做这件事的兴趣，相信您一定能做好它的。

1.7 关于 Visual Web Developer 的实用提示

本书大部分章节的末尾都有一小节有用的提示。这些提示要么没有出现在正文中的任何地方，要么鼓励你进一步探索或进行某些测试。有时它们可能看起来似乎并不相干，或者乍一看难以理解，但是当顺着本书的指导做下去并回过头来看前面章节的提示时，就会发现它们实际上是有意义的。如果在第一次看到某些内容时不能完全理解，也不要担心。进行一些思考，然后过几天再回顾这一主题。让这个想法稍微深入一点，很可能其意义自然就明白了。这不仅适用于实用提示部分，而且适用于整本书。

- 在开始继续学习第 2 章之前，先多了解一下 VWD。向站点中添加几个页面，从 Toolbox 中拖放一些控件到页面上，并在浏览器中浏览它们。这样，当开始学习第 2 章时就会对其中的工具和许多可用的控件有一个更好的了解。
- 熟悉调整 Visual Web Developer IDE 的许多选项。在构建 Web 站点时，大部分时间会花在 IDE 上，因此尽可能将其调整为喜欢的样子是很有意义的。不要怕弄乱了；因为无论多乱，总是可以恢复为以前的设置的。
- 花一些时间浏览在 VWD 的 Options 对话框中找到的设置(可以通过 Tools | Options 菜单命令访问)。许多设置是容易理解的，而且确实有助于进一步将 IDE 调整为喜欢的样子。

1.8 本章小结

本章介绍了很多重要的基础知识，使我们对 ASP.NET 4 和 VWD 2010 有了初步了解。首先从总体上简单介绍了 Microsoft .NET Framework 的历史，并特别介绍了 ASP.NET。

然后说明了如何获得和安装 Visual Web Developer 2010 Express Edition。VWD 是创建 ASP.NET 4 Web 页面时使用最为广泛和最为通用的工具。为了更有效地使用这个工具，本章介绍了如何使用和定制 IDE 的主要功能。在接下来的几章中，将进一步介绍使用 VWD 中的很多工具的方法。

重要的是要了解 VWD 如何让页面进入 Web 浏览器。了解处理请求的 Web 服务器，了解如何处理页面以及最终发送到浏览器中的 HTML 对于了解 ASP.NET 是至关重要的。本章简要地介绍了 Web 页面的请求方式和提供给浏览器的方式。

第 2 章将更加详细地介绍如何创建 Web 站点。

1.9 练习

1. 解释 VWD 中的页面标记与浏览器中的最终 HTML 页面之间的区别。
2. 解释 HTML 与 XHTML 之间的区别。两者有什么关系？
3. 假设您有若干 HTML 片段想在站点中大量使用。使这些片段在 VWD 中可用的最佳方式是什么？

- 4. 重置部分或全部 IDE 自定义设置有哪 3 种方法？
 - 5. 如果要修改页面上某个控件的属性，如按钮上的文本，可以使用哪两种方法？
- 练习的答案见附录 A。

本章要点回顾

特性	标记中用于定义或改变其行为的额外信息
元素	包含文本或其他内容的标记对
HTML	超文本标记语言：浏览器使用该语言显示 Web 页面
HTTP	超文本传输协议：用于 Web 浏览器和 Web 服务器之间的通信
IDE	集成开发环境：一个由应用程序和工具所组成的集合，用于开发应用程序
JavaScript	一种用于与客户端浏览器上显示的 Web 页面进行交互的语言
Tag	用尖括号括住的文本，用来创建 HTML 元素
Visual Studio 2010	创建.NET 应用程序的开发环境
Visual Web Developer	Visual Studio 的一部分(也可以独立地作为免费的 Express 版本使用)，用于创建 ASP.NET Web 应用程序
XHTML	使用 XML 规则重写的 HTML

第 2 章

构建 ASP.NET Web 站点

本章要点

- 构建 ASP.NET Web 站点的两种不同的项目类型
- 用于站点开发的不同项目模板
- ASP.NET 中众多可用的众多文件类型及它们的用途
- 构建易于现在和将来管理的结构化 Web 站点的方法
- 如何使用设计工具创建格式化的 Web 页面

要构建漂亮、实用的成功 Web 站点，就必须了解几种重要的技术与语言，包括(X)HTML、ASP.NET、CSS(Cascading Style Sheets, 层叠样式表)、服务器端编程语言(如 C#或 VB)以及客户端语言(如 JavaScript)。本章和接下来的几章将奠定这几种技术的坚实基础，因此在学完全书后您会非常熟悉这些最重要的概念。

除了这些技术外，您还必须了解在第 1 章中介绍的 Visual Web Developer IDE。需要知道如何创建站点，添加页面和管理 Visual Web Developer(VWD)提供的所有工具栏和窗口。此外，还需要知道如何在 VWD 中用 HTML 和服务控件构建和设计 Web 页面。

本章将详细介绍如何构建与管理 Web 站点。还将介绍如何创建 ASP.NET Web 页面、向它们添加标记，从而使创建的页面能够向用户展示信息并对他们的行为作出响应。

虽然在第 1 章中您已经构建了第一个 ASP.NET Web 站点，但本章将开始更深入地介绍如何新建一个 Web 站点。因为当您开始构建一个新站点时需要作许多选择，所以了解所有不同的选项并为您的场景选择正确的选项是很重要的。

2.1 使用 VWD 2010 构建 Web 站点

第 1 章快速概览了如何使用 VWD 构建 Web 站点。您只需简单地从 File 菜单中选择 New Web Site 命令，选择一种语言，选择标准的 ASP.NET Web Site 模板，并单击 OK 按钮即可。然而，New Web Site 对话框中的内容远不止第 1 章中提到的那些。您可能已经注意到，可以从若干模板中选择一个

来构建不同类型的站点。但是在介绍用来构建新 Web 站点的各种模板前，我们将大致介绍一下 VWD 中可用的各种不同的项目类型。

2.1.1 不同的项目类型

在 Visual Web Developer 2008 Express Edition 发布之初，只有一个项目类型可用：Web Site Project。为了使用第二种项目类型(Web Application Project)，需要用到 Visual Studio 的一个商业版本。2008 年 8 月，Microsoft 发布了 Visual Web Developer 2008 的 Service Pack 1 版本，使得免费 Express 版本的用户也可以使用 Web Application Project 模板。这一功能在 VWD 2010 中得到了保留，因此现在有两种项目类型可供选择。下面将讨论这两种项目类型。

1. Web Site Project

Web Site Project 表示在 VWD 中创建的 Web 站点项目。创建新的 Web Site Project 的方法是：从 Visual Web Developer 的主菜单中选择 File | New Web Site 命令，或者选择 File | New | Web Site 命令。

Web Site Project 是在 Visual Studio 2005 中引入的，它增加了创建和使用 Web 站点的灵活性。相比于 Visual Studio .NET 的早期版本，Web Site Project 站点只是一个 Windows 文件夹，以及其中的一组文件和子文件夹。在该 Web 站点中没有跟踪所有单个文件的集合文件(称为项目文件，扩展名是.vbproj 或.csproj)。只需将 VWD 指向一个文件夹，它就会一直把这个文件夹作为 Web 站点打开。这样就可以非常容易地创建站点的副本、移动它们或者与别人共享，因为它不依赖于您本地系统中的文件。由于缺少中心项目文件，因此 Web Site Projects 通常简称为 Web Sites，在本书的其余部分就使用这个术语。

在进行开发的过程中，Microsoft 不仅收到了大量的正面反馈，也收到了不少负面响应。开发人员抱怨 Web Site Projects 对于他们的开发环境来说太受限制。由于没有容器文件来跟踪站点中的所有内容，因此很难从站点中排除文件或文件夹，也不容易使用源代码控制系统——一个集中的系统，开发人员可以用来协同开发一个项目，并自动跟踪项目中的变化。同样，Web Site Projects 也影响了编译和部署 Web 站点的方式，使得习惯于以前的模型的开发人员难以对新项目类型应用他们的知识和技能。

作为对这种批评的响应，Microsoft 在 2006 年 5 月发布了 Web Application Project 模板作为 Visual Studio 2005 标准版本及更高版本的插件。Web Application Project 现在是 Visual Web Developer 所有版本(包括免费版本和商业版本)中一个不可或缺的部分。

2. Web Application Project

因为跟踪 Web 站点中所有内容的单个项目文件将整个 Web 站点作为一个项目进行管理，所以 Web Application Project 使得团队开发人员，以及那些需要对站点内容和编译及部署过程有更多控制的开发人员更容易使用 VWD 构建 Web 站点。

在 VWD 2010 中，可以通过 File | New Project 对话框来创建一个新的 Web Application Project。在该对话框中，单击首选编程语言(可以是 Visual Basic 或 Visual C#)，然后单击 Web 类别，在其中您会发现若干个 ASP.NET Web 应用程序模板。其中一个可用的项目模板是 ASP.NET MVC 2 Web Application，它是根据模型-视图-控制器(Model View Controller, MVC)模式来创建应用程序的，该模式是 Web 应用程序开发的另一个流行方式。本书中不使用或讨论 MVC，不过如果要了解关于它

的更多内容，可以参考 Wrox 出版社出版的 *Beginning ASP.NET MVC 1.0*(ISBN:978-0-470-43399-7) 一书。

本书中使用的是 Web Site Project 模板，这是因为该模板对于 ASP.NET 初学者来说很容易使用。但是接下来您就会发现在构建站点时，使用 Web Application Project 模板与使用 Web Site Project 模板有很多共同之处。如果希望跟随本书，则需要使用 Web Site Project 模板。如果没有指定特定的项目类型，那么本书在涉及一般的 Web 站点时将互换使用 Web 站点和 Web 应用程序这两个术语。

了解了不同的项目类型后，接下来就要考虑不同的 Web 站点模板及它们的选项。

2.1.2 选择正确的 Web 站点模板

VWD 中的 New Web Site 对话框中包含不同的 Web 站点模板，各个模板的用途各不相同。

图 2-1 显示了 VWD 中的 New Web Site 对话框。根据您的 VWD 版本，可以选择 File | New Web Site 或者 File | New | Web Site 命令打开这个对话框。如果出现的对话框与图 2-1 不同，则要确保选择的是 File | New Web Site 而不要误选了 File | New Project。

在对话框的左侧部分，可以从 Visual Basic 和 Visual C#中选择一种作为站点的编程语言。中间的部分显示了默认安装的 ASP.NET Web 站点模板。其中的每一个模板都会在下一节中讨论。当您创建自己的模板时(第 6 章将介绍如何创建自己的模板)，或者安装其他方提供的模板时，它们也会显示在这个区域中。

在 Planet Wrox Web 站点中，ASP.NET Empty Web Site 模板是贯穿全书用到的。其他的模板只在下面几节中作简要描述，以便您了解如何使用它们。系统安装的模板的精确列表是根据 Visual Studio 的版本和安装的组件而建立的。即便其中有其他的模板也不要担心，只要有 ASP.NET Web Site 和 ASP.NET Empty Web Site 模板即可。

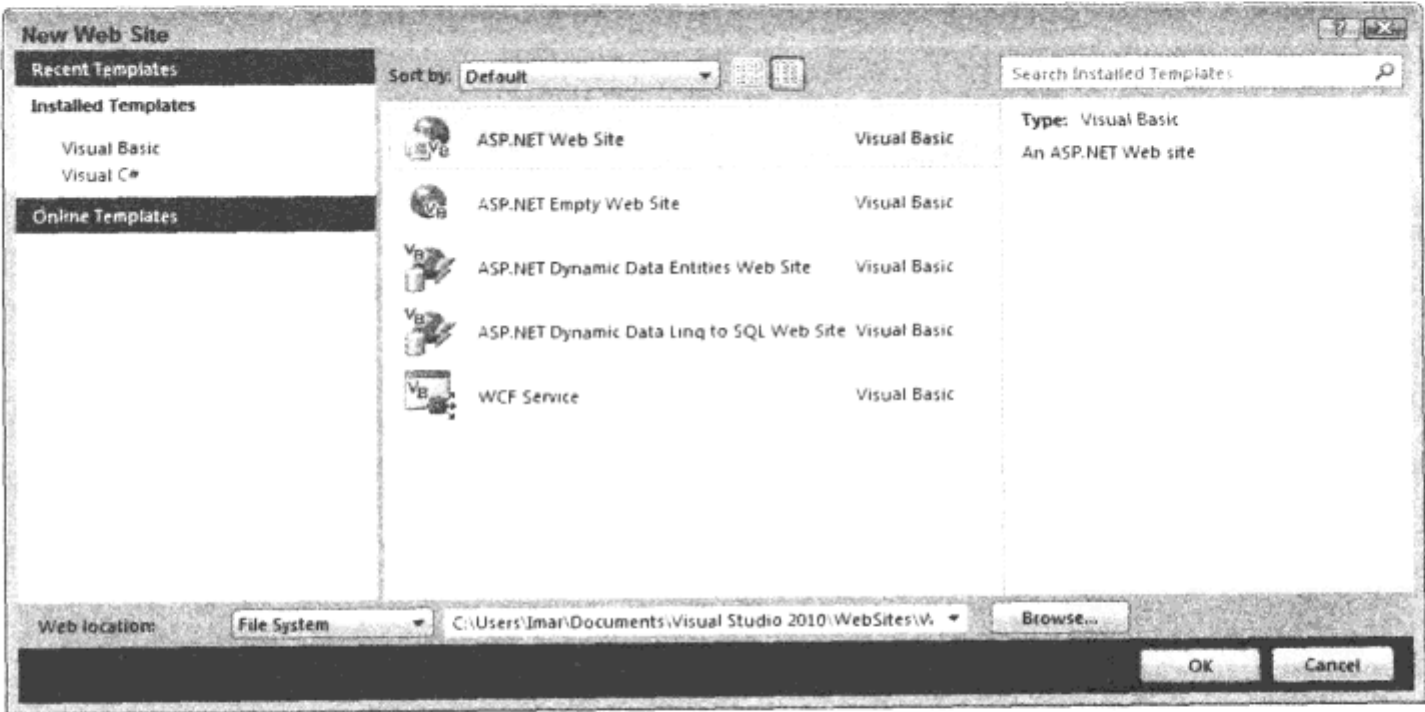


图 2-1

1. ASP.NET Web 站点

这个模板允许配置一个基本的 ASP.NET Web 站点。它包含许多文件和文件夹用于开始站点的发展。各种不同的文件类型将在本章后面讨论。特殊的 App_Data 文件夹和 Account 文件夹中页面的功

能将在本书后面讨论。

一旦你决定开始开发一个真实的 ASP.NET Web 站点时，该模板会是一个很好的起点。

2. ASP.NET Empty Web Site

ASP.NET Empty Web Site 模板只包含一个配置文件(web.config)。如果包含用于构建新的 Web 站点或希望从头开始构建站点时使用的许多已有文件，则 ASP.NET Empty Web Site 模板会很有用处。使用该模板作为本书中构建的示例 Web 站点的基础，并在学习本书的过程中添加文件和文件夹。

3. WCF Service

该模板可用于构建包含一个或多个 WCF Service 的 Web 站点。WCF Service 模板与 Web Service 模板有点相似，它用来创建可通过网络调用的方法。第 10 章将会介绍如何在浏览器中创建和使用 Web 服务。然而，WCF Services 或 Windows Communication Foundation Services 比这个简单的 Web 服务要复杂得多，而且提供了更大的灵活性。WCF Services 不在本书的讨论范围内，如果要了解关于它们的更多内容，可以参见清华大学出版社引进并出版的《WCF 高级编程》一书。

4. Dynamic Data Web 站点

Dynamic Data 的两个模板用于构建灵活且强大的 Web 站点来管理数据库中的数据，而不需要手动输入许多代码。本书不讨论这些模板，但是将更多地介绍第 14 章中使用的 Microsoft ADO.NET Entity Framework 这一模板。

虽然看起来似乎您必须对正确的 Web 站点模板作出一个清晰的选择，但是其实并没有关系。由于 VWD 中的 ASP.NET Web 站点实质上只是对文件夹的一个引用，因此很容易将一个模板的类型添加到另一个模板上。例如，第 10 章将介绍的向标准的 ASP.NET Web 站点或 ASP.NET Empty Web 站点中添加 Web 服务文件，这是完全可以接受的(而且非常常见)。

2.1.3 构建与打开新的 Web 站点

有几种不同的方式可以构建新的 Web 站点和打开现有 Web 站点。选择哪种方式在很大程度上取决于访问 Web 站点的方式(从本地或远程计算机)，以及您是想使用 VWD 配备的内置 Web 服务器，还是使用 Windows 自带的 Web 服务器。

本书的所有示例都假定从本地硬盘驱动器中打开站点，并且使用内置 Web 服务器，因为使用它开发站点非常方便。然而，第 19 章将介绍如何使用和配置 Internet Information Services，或者简称为 IIS。几乎 Windows 的所有版本都配备了这个高级 Web 服务器。IIS 主要用于驻留 Web 站点的产品，因为它能在高通信量的情况下服务 Web 页面。

1. 构建新的 Web 站点

下一个“试一试”练习将带领您构建 Planet Wrox Web 站点，这个站点是本书将要使用的项目。除了特别说明外，本书余下部分的所有练习都假定您在 VWD 中打开了此 Web 站点。该练习要求将 Web 站点存储在 C:\BegASPNET\Site 文件夹中。注意一下这个文件夹的名称，因为它是在全书中都要用到的。如果您决定使用其他文件夹，则一定要确保每次见到本书中的这个名称时都是使用您自己的位置。还要确保没有使用诸如井号(#)这样的特殊字符，也不要再在文件夹的名称中插入空格，以

免在构建站点时出现问题。

试一试

构建一个新的 ASP.NET 4 Web 站点

(1) 首先用 Windows 资源管理器或“我的电脑”在 C 盘的根文件中创建一个名为 BegASPNET 的文件夹。在这个文件夹内再创建一个文件夹，命名为 Site。最终应得到一个全名为 C:\BegASPNET\Site 的文件夹。如果您根据本书的“前言”中的指示做了，并且解压了本书的源代码包，那么您就已经有了 BegASPNET 文件夹。这个文件夹中又包含了 Source 和 Resources 文件夹，但还是需要创建 Site 文件夹。如果希望同时使用 VB.NET 和 C#语言，则可以创建两个文件夹——BegASPNETVB 和 BegASPNETCS 并使用 VWD 的两个实例。

(2) 启动 Visual Web Developer 2010, 并根据您的 VWD 版本选择 File | New Web Site 或 File | New | Web Site 命令。



常见错误：不要使用 File | New Project 错误地创建一个新的 Web Application in Project，因为该项目模板不适合本书中的练习。

(3) 在左侧的 Installed Templates 区域中选择 Visual Basic 或 Visual C#。本书中所有的示例都是用这两种编程语言显示的，因此您可以选择一种最喜欢的语言。

(4) 在中间区域选择 ASP.NET Empty Web Site。

(5) 在 Web location 的下拉列表中，确保选中了 File System。列表中还有其他两个选项，分别是 HTTP 和 FTP。HTTP 用来使用所谓的 Microsoft FrontPage Server Extensions 打开一个运行 IIS 的远程站点，而 FTP 则用来在 FTP 服务器中打开一个站点。

(6) 单击 location 文本框旁边的 Browse 按钮，浏览在本练习第一步中创建的文件夹，并单击 Open 按钮。

最终的屏幕看起来应像图 2-2 所示的那样，只是您可能把它设成了 Visual C#而不是 Visual Basic。

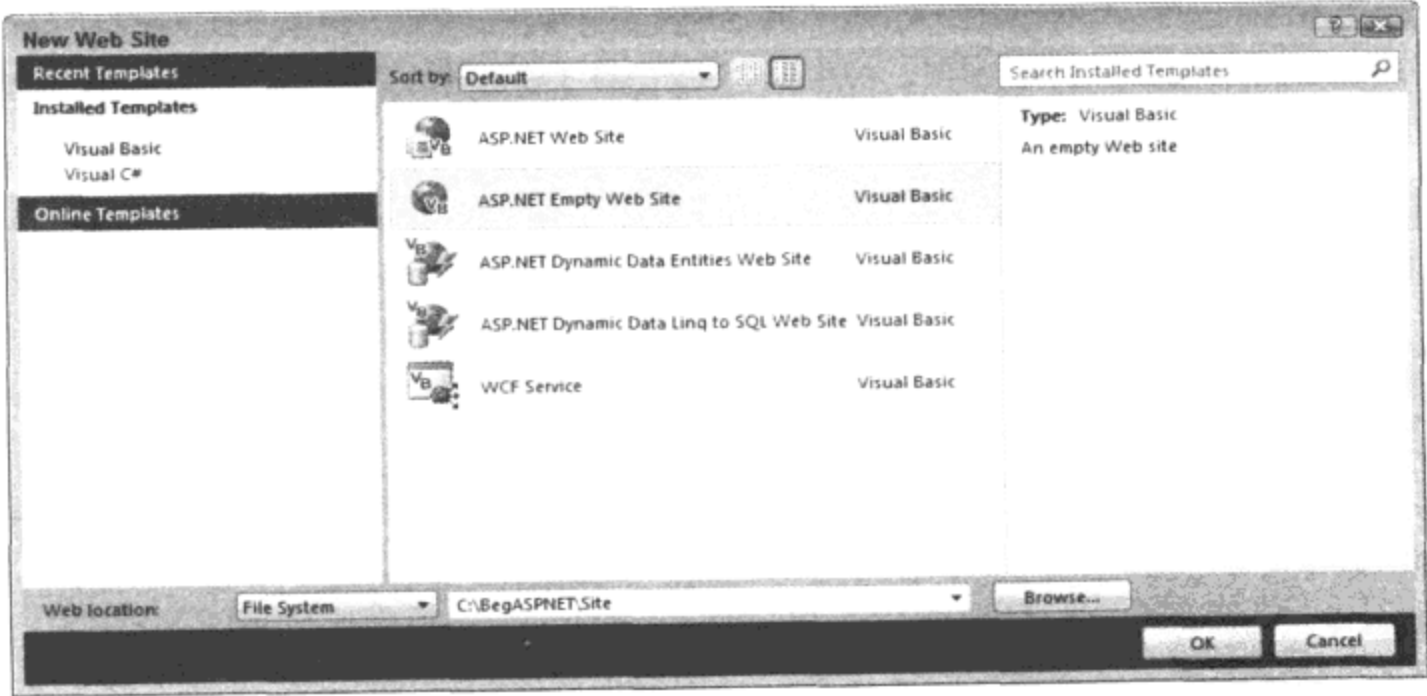


图 2-2

(7) 单击 OK 按钮后，VWD 会构建一个新的站点。

工作原理

单击 OK 按钮后，VWD 会构建一个新的空 Web 站点。这个新的 Web 站点中只包含一个配置文件(名为 web.config)。在 Solution Explorer 中， Web 站点现在看起来如图 2-3 所示。



图 2-3

在 VWD 看来，基于 Empty Web Site 模板的 Web 站点只是一个简单的 Windows 文件夹，它是在磁盘上包含同样文件的实际文件夹。从图 2-4 中可以看出，构建该站点不需要其他文件，该图是一个 Windows 资源管理器，显示了文件夹 C:\BegASPNET\Site 中的文件。

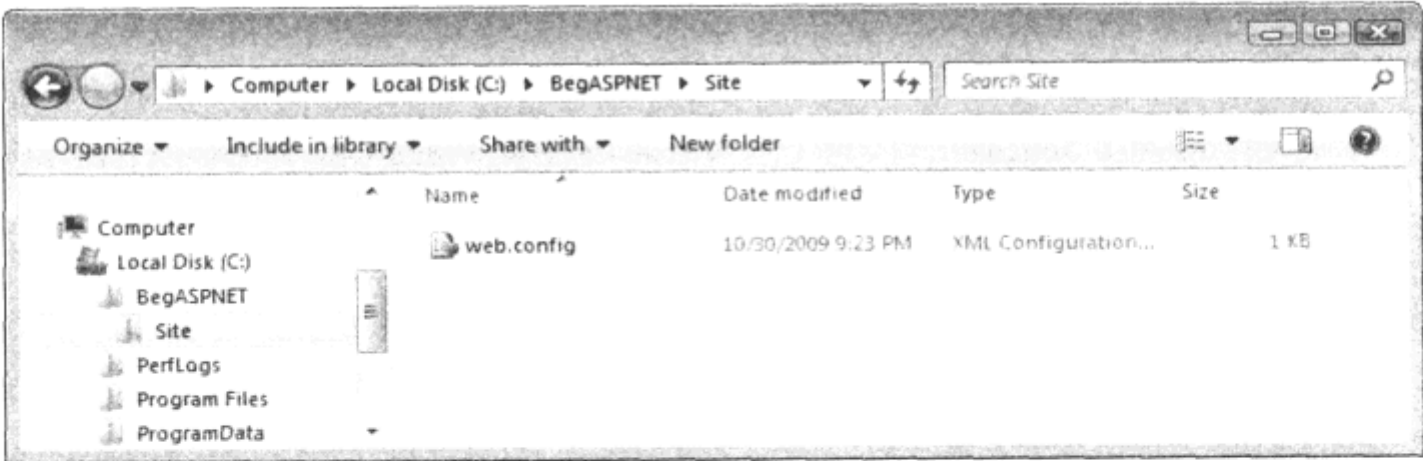


图 2-4

如果没有看见 Web 文件的.config 扩展名，不用担心。在稍后的练习中将会介绍如何查看文件的扩展名。

当您按部就班地照本书学下去时，自然会向这个站点中添加新的文件和文件夹。这些附加的文件和文件夹会显示在 Solution Explorer 中，并且也会出现在 C:\BegASPNET\Site 这一 Windows 文件夹中。

打开基于 Web Site Project 模板的 Web 站点与构建新站点的过程非常相似。下一节将介绍如何在 VWD 中打开现有站点。

2. 打开现有 Web 站点

与构建新站点一样，在 VWD 中打开现有站点时，根据 Web 站点的源位置，会有几种选择。可以选择从本地文件系统中打开站点，也可以从本地 IIS Web 服务器中打开站点，还可以用 FTP 从远程服务器中打开站点，或者用 Microsoft FrontPage Server Extensions 从远程站点中打开。图 2-5 显示了 VWD 中的 Open Web Site 对话框。

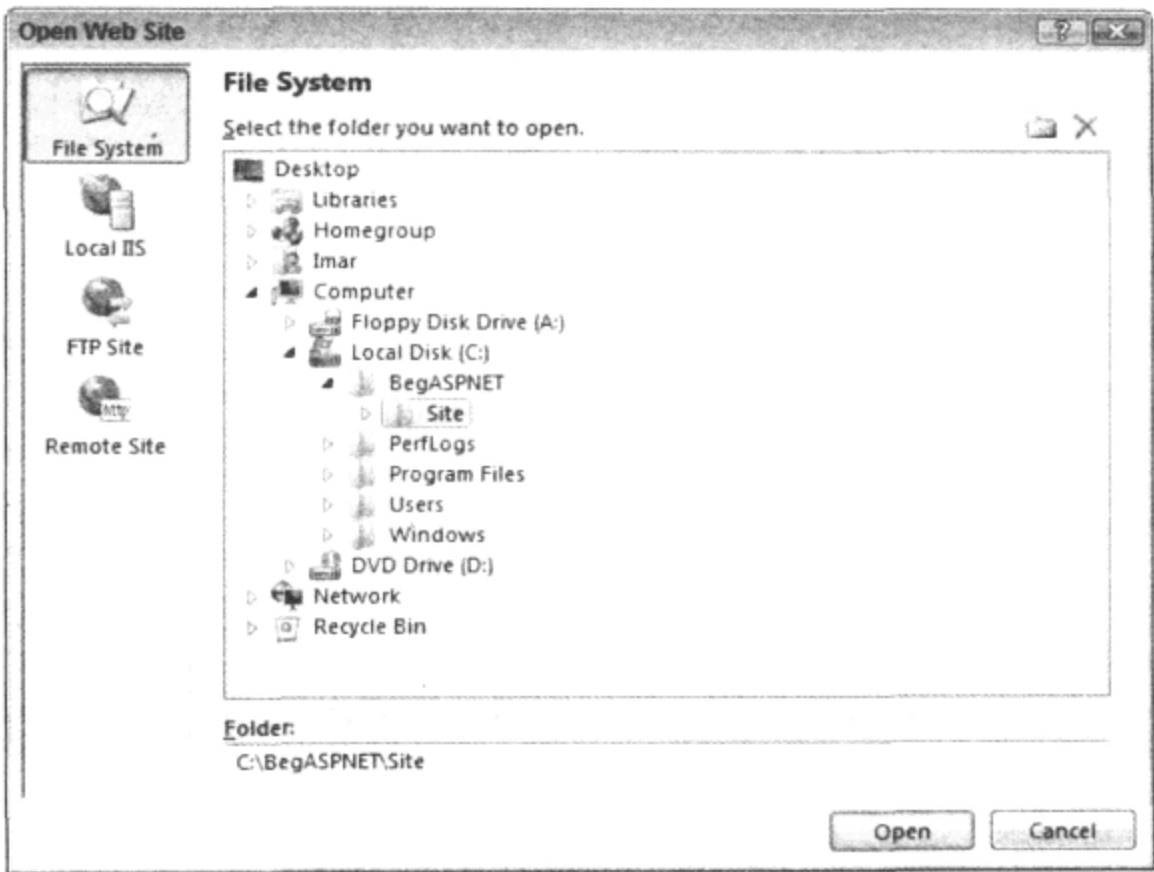


图 2-5

本书的所有示例都假定您总是使用窗口左栏的第一个按钮(File System 按钮)从本地文件系统中打开 Planet Wrox 这一 Web 站点。然后在右边的窗格中定位到 Web 站点(本例中是 C:\BegASPNET\Site)并单击 Open 按钮。

在上一个“试一试”练习中创建的站点是个空站点。要让站点更有用，就要向其中添加文件。下一节将着重讨论要添加到站点中的多种文件类型以及将文件添加到站点中的方式。

2.2 操作 Web 站点中的文件

虽然 ASP.NET 4 Web 站点至少由一个 Web 窗体(扩展名为.aspx 的文件)组成,但是它常常是由更多文件组成的。VWD 中有许多不同的文件类型可用，各个类型提供了不同的功能。下一节将介绍 VWD 中用到的最重要的文件类型。此外，还会介绍向站点中添加这些文件的几种不同方法。

2.2.1 ASP.NET 4 Web 站点的文件类型

为了让您对各种不同类型的文件在 ASP.NET 中的使用有个印象，图 2-6 显示了允许向站点中添加新文件的对话框(可以右击 Solution Explorer 中的 Web 站点并选择 Add New Item 命令，或者从主菜单中选择 Website | Add New Item 命令打开)。

这些能够添加到站点中的文件可以分组到不同的类别中。下面将讨论其中最重要的文件(贯穿全书的示例都用到的文件)。

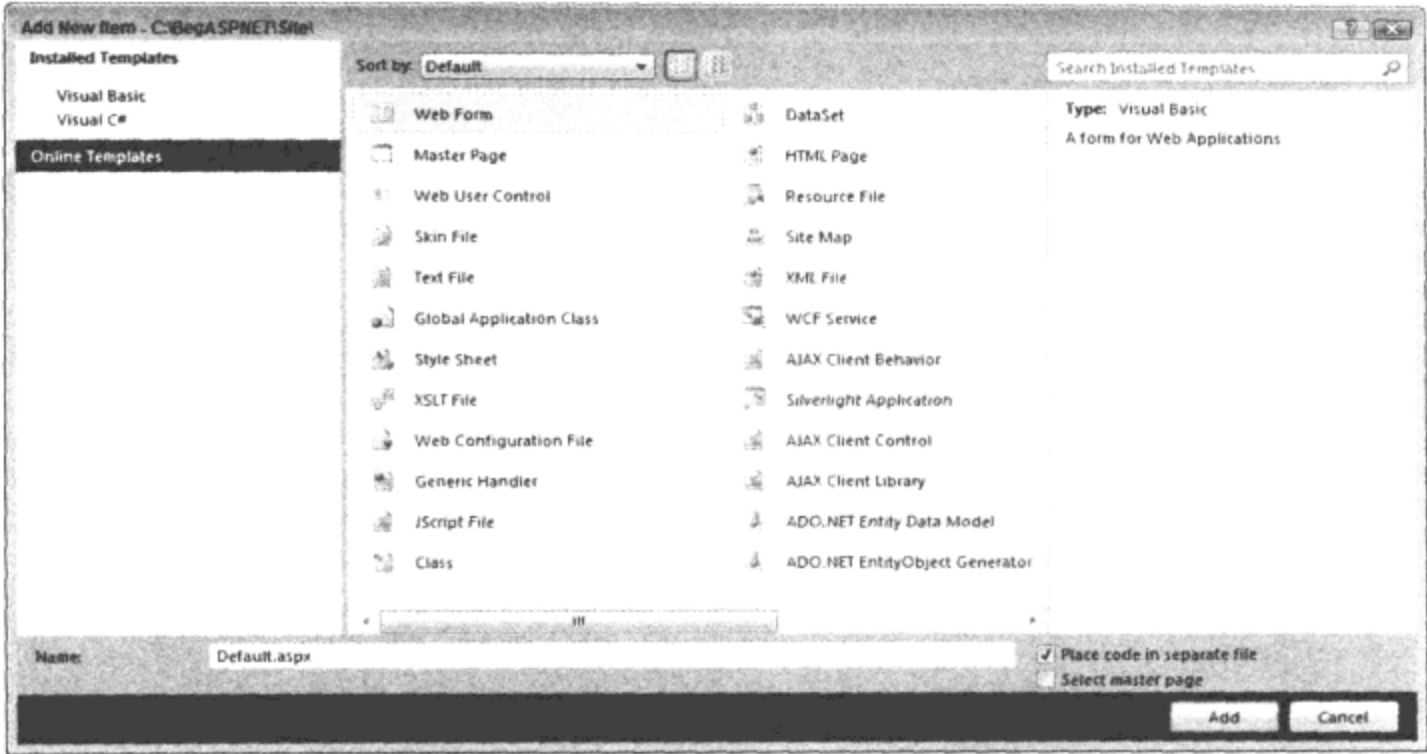


图 2-6

1. Web 文件

Web 文件是 Web 应用程序中特有的文件，可以由浏览器直接请求，也可以用来构建在浏览器中请求的 Web 页面的一部分。表 2-1 中列出了各种 Web 文件和它们的扩展名，并说明了各种文件的用法。

表 2-1

文件类型	扩展名	说明
Web Form	.aspx	这些文件是所有 ASP.NET Web 站点都要用到的文件。Web 窗体表示用户在他们的浏览器中浏览的页面
Master Page	.master	这些文件允许定义 Web 站点的全局结构和外观。在第 6 章中可以看到它们的用法
Web User Control	.ascx	包含可重用在站点的多个页面中的页面片段。第 8 章将专门介绍用户控件
HTML Page	.htm/ .html	可用来显示 Web 站点中的静态 HTML
Style Sheet	.css	包含允许定制 Web 站点的样式和格式的 CSS 代码。第 3 章将介绍关于 CSS 的更多信息
Web Configuration File	.config	包含用在整个站点中的全局配置信息。在本书后面，从第 4 章开始，您将会了解如何使用 web.config 文件
Site Map	.sitemap	包含一个层次结构，表示站点中 XML 格式的文件。Site Map 用于导航并在第 7 章讨论
JScript File	.js	包含可以在客户端浏览器中执行的 JavaScript(Microsoft 称之为 JScript)
Skin File	.skin	包含 Web 站点中的控件的设计信息。Skin 将在第 6 章讨论

下面的“试一试”练习展示了如何向这个全书都会用到的站点中添加新的母版页。

试一试 向站点中添加文件

- (1) 如果站点还没有打开，则选择 File | Open Web Site 命令打开先前构建的 Planet Wrox Web 站点。一定要从 File System 中打开站点，定位到包含站点的文件夹(C:\BegASPNET\Site)，并单击 Open 按钮。
- (2) 在 Solution Explorer 中，右击站点并选择 New Folder 命令，如图 2-7 所示。

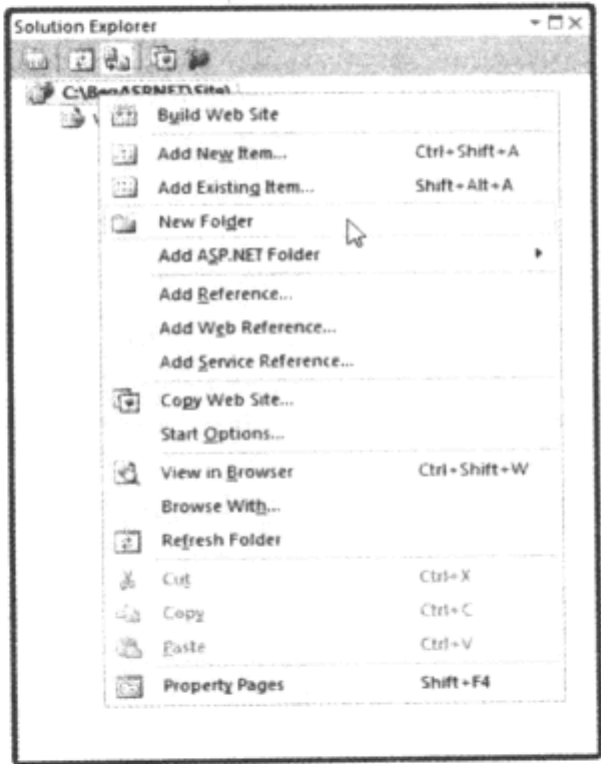


图 2-7



常见错误：一定要单击实际站点而不是 web.config 文件，否则就得不到正确的菜单项。

- (3) 输入 MasterPages 作为文件夹的名称并按下 Enter 键。然后右击这个新文件夹并选择 Add New Item 命令。另一种方法是从 Visual Web Developer 的主菜单中选择 File | New File 命令或 Website | Add New Item 命令。
- (4) 在显示的对话框中，选择 Master Page 并将其命名为 Frontend。当添加文件时，VWD 会自动地添加.master 扩展名。确保在 Installed Templates 中已经选择了要用于站点的语言，并选中了右下角的 Place Code in Separate File 复选框。最后，单击 Add 按钮，就将母版页添加到了站点中，并且已经在文档窗口中自动打开。

工作原理

这个简单的练习显示了如何通过两种方式将新项添加到 Web 站点中。虽然在这一阶段站点还不是很完善，但是您添加的文件已经形成了本书其余内容的基础。
下一节将简单地介绍其余的文件类型。

2. Code Files

将代码文件添加到站点中的方式与添加 Web 文件的方式相同。表 2-2 描述了各种类型的代码文件。

表 2-2

文件类型	扩展名	说明
Web Service	.asmx	可以被其他系统调用，包括浏览器，可以包含能在您的服务器上执行的代码。Web Service 将在第 10 章介绍
Class	.cs / .vb	可以包含构建 Web 站点的代码。注意 Code Behind 文件(稍后讨论)也有这样的扩展名，因为它们实质上是类文件。C#使用带有.cs 扩展名的文件，而 Visual Basic 使用的是带有.vb 扩展名的文件
Global Application Class	.asax	可以包含为了响应站点中的有趣事情而触发的代码，例如应用程序的开头或者当在站点中某处发生错误时。在第 18 章您将看到如何使用这个文件

除了 Code Files 类别外，还有一组值得研究的文件：Data Files。

3. Data Files

Data Files 用来存储可以用在站点和其他应用程序中的数据。这组文件由 XML 文件、数据库文件以及与使用数据相关的文件组成，如表 2-3 所示。

表 2-3

文件类型	扩展名	说明
XML File	.xml	用来存储 XML 格式的数据。除了纯 XML 文件外，ASP.NET 还支持几种基于 XML 的文件，其中两种您在以前已经见过：web.config 和 Site Map
SQL Server Database	.mdf	扩展名为.mdf 的文件是 Microsoft SQL Server 使用的数据库。在第 12 章及以后章节中将会讨论数据库
ADO.NET Entity Data Model	.dbml	用于声明性地访问数据库，不需要写代码。从技术上来讲，这并不是一个数据文件，因为它不包含实际数据。然而，由于它们与数据库绑定得如此紧密，因此把它们归组在这个标题下是有意义的。在第 14 章您将了解关于 ADO.NET Entity Framework 的更多内容

正如您在前面的“试一试”练习中所看到的，添加这些类型之一的任何文件真的很容易。只要轻松地将现有文件添加到站点中即可。

2.2.2 添加现有文件

在 Web 站点中创建的文件不一定都要是全新的。在有些情况下可以重用其他项目中的一些文件。例如，在多个站点上重用一個 logo 或 CSS 文件。可以通过在 Solution Explorer 中右击 Web 站点并选择 Add Existing Item 命令轻松地添加现有文件。在出现的对话框中，可以浏览文件，也可以通过按住 Ctrl 键来选择多个文件。最后，当单击 Add 按钮时，文件就添加到 Web 站点中了。

然而，向站点中添加文件还有一个更容易的方式，在需要向站点中添加多个现有文件和文件夹时用这种办法会节省大量时间：拖放。下面的“试一试”练习就说明了具体做法。

试一试

向站点中添加现有文件

(1) 在 Windows 环境下，最小化所有打开的应用程序，右击桌面并选择 New | Text Document 命令。如果没有看到这个选项，则只需用 Notepad 简单地创建一个新的文本文档并保存在桌面上。

(2) 将该文本文件重命名为 Styles.css。一定要用.css 替换.txt 扩展名。如果没有看到初始的.txt 扩展名，而且文件的图标也没有从文本文件转换成 CSS 文件(默认情况下它与文本文件的图标相同，只是上面有一个齿轮符号，不过您可能安装了可以修改 CSS 文件图标的软件)，那么就表示 Windows 被配置为隐藏已知文件类型的扩展名了。如果是这种情况，则打开 Windows 资源管理器并在 Windows XP 中选择 Tools | Folder Options 命令，或者在 Windows Vista 和 Windows 7 中单击 Organize 按钮，然后选择 Folder and Search Options 选项。在这两种情况下，切换到 View 选项卡，并取消选中 Hide Extensions for Known File Types 选项。现在您可能需要将文件名从 Styles.css.txt 改为 Styles.css。

当将文件名从.txt 改为.css 时，Windows 可能会给出一个警告，指出如果继续进行下去文件可能会变得不可用。您可以对这个问题放心地回答 Yes 以继续转换。

(3) 重新排列 VWD，以便也能看到桌面上带 CSS 文件的部分。您可以使用 VWD Windows 标题栏上 Close 按钮旁边的 Restore Down 按钮，来使 VWD 退出全屏模式。

(4) 单击桌面上的 CSS 文件，按住鼠标按钮将该文件拖到 Solution Explorer 中。一定要把文件拖到 Solution Explorer 中，而不要拖到 VWD 中的其他部分，否则文件就添加不进去。例如，当把它拖到文档窗口时，VWD 只是简单地为您打开文件，而不是把它添加到站点中。

(5) 当在 Solution Explorer 中的 Web 站点节点或现有文件上释放鼠标时(如图 2-8 所示)，CSS 文件就会添加到站点中。



注意：如果使用 Windows Vista 或 Windows 7 并以管理员的身份运行 VWD，那么将可能无法工作，因为 Windows 不允许 Windows Explorer 和 VWD 相互通信。在这种情况下，可以使用之前讨论的 Add Existing Item 菜单来添加现有文件或使用复制粘贴命令。

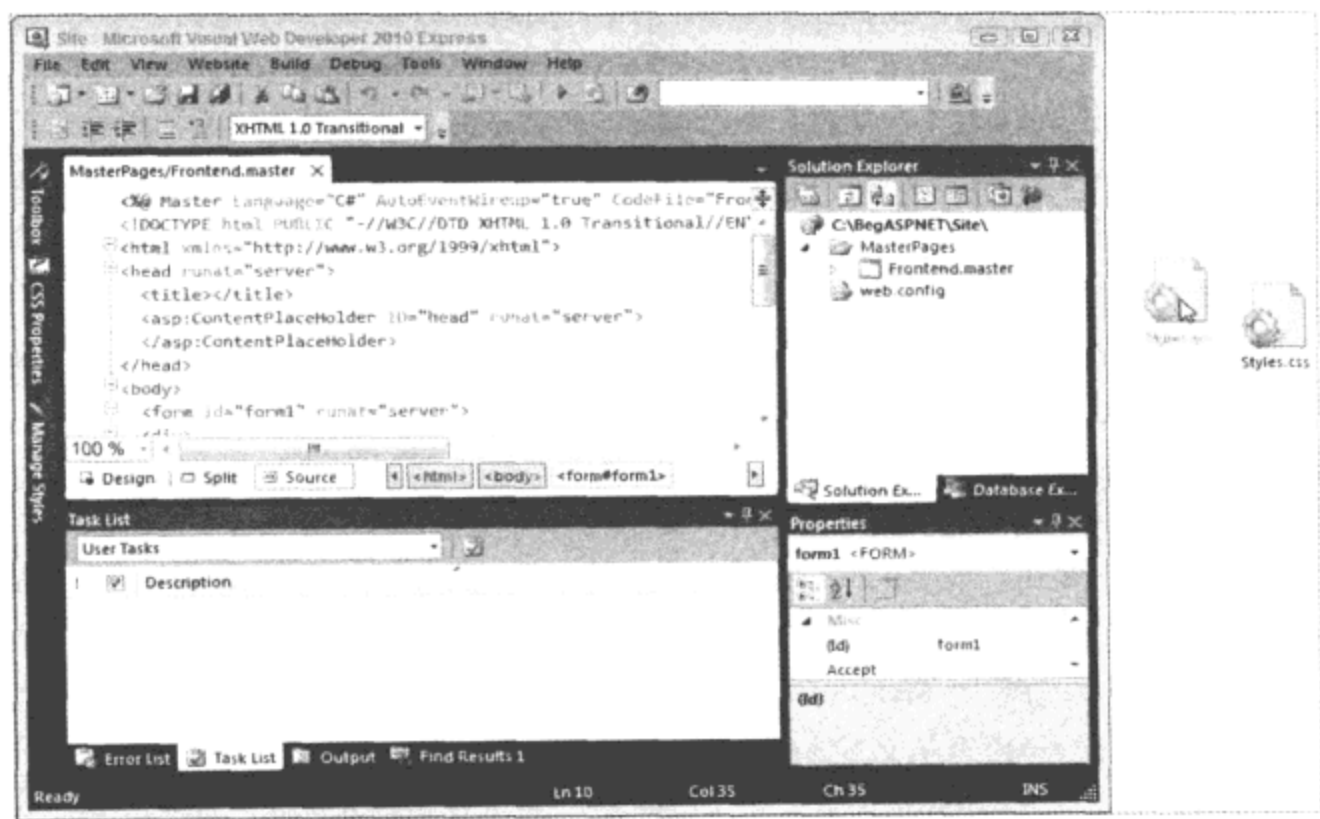


图 2-8

工作原理

在这个“试一试”练习中要指出的重要事情是，当将文件添加到站点中时，VWD 创建了该文件的一个副本。因此，在对 VWD 中的副本作修改时不会影响桌面上的原始 Styles.css 文件。这样，就容易将现有 Web 站点之外的文件拖放到新站点中，而不会影响原始文件。使用 VWD 中的 Add Existing Item 对话框添加文件时也是如此。

如果在 VWD 之外向 Web 站点的文件夹中添加文件，它们可能不会立即显示出来。可以通过单击 Solution Explorer 的工具栏上的 Refresh 按钮得到文件列表的一个刷新副本。

2.2.3 组织站点

由于组成站点的文件有很多，因此最好按功能将它们分组到单独的文件夹中。例如，所有的 Style Sheet 文件都能归到 Styles 文件夹中，.js 文件可以归到 Scripts 文件夹中，用户控件可以归到 Controls 文件夹中，母版页可以存储在 MasterPages 文件夹中。这是个人习惯问题，但是结构化和组织良好的站点更容易管理与理解。下面的“试一试”练习解释了如何将文件移动到新的文件夹中以便管理该站点。

试一试

组织 Web 站点

- (1) 右击 Solution Explorer 中的 Planet Wrox 站点，并选择 New Folder 命令。
- (2) 输入 Styles 作为新文件夹名并按下 Enter 键。
- (3) 创建另一个文件夹 Controls。本书的其余部分会用到这两个文件夹。
- (4) 把之前添加的 Styles.css 文件拖放到 Styles 文件夹中。

(5) 如果一切顺利，那么现在的 Solution Explorer 看起来应该如图 2-9 所示。

如果您的 Solution Explorer 看上去不同于图 2-9 所示的样子，请再根据本“试一试”练习做一次，直到您的站点看起来与图 2-9 完全相同为止，包括里面的文件夹结构和文件都要相同。本书以后的“试一试”练习都假定您在 Web 站点中有了正确的文件夹和文件。

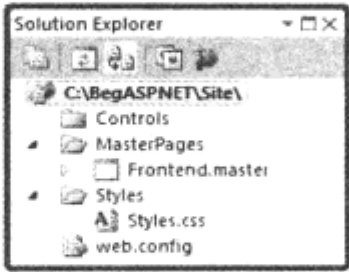


图 2-9

工作原理

结构和组织对于站点的管理很重要。虽然您可能想将所有文件都添加到项目的根文件夹中，但是最好不要这么做。如果是非常小的站点可能看不出什么差别，但是一旦站点开始变大，您就会发现如果没有良好的结构，文件就变得非常难以管理。将相关的文件放在单独的文件夹中是构建有组织的站点的第一步。而将相同类型的文件放在一个文件夹中只是优化站点的一种方法。在后面的章节中，您将发现也常用独立的文件夹来分组功能类似的文件。例如，所有只能由站点的管理员访问的文件都放在一个名为 Management 的文件夹中。

VWD 的拖放功能使它可以轻松地重组站点。只要简单地选择一个或多个文件，并把它们拖到新位置上即可。如果在扩展站点时继续用这些重组方法，您会发现第二天或者从现在起的 6 个月内，在需要时就可以没有任何问题地找到正确的文件。

2.2.4 特殊文件类型

上一节列出的有些文件要求您把它们放在一个特殊文件夹中，这种特殊文件夹是相对于上一节提到的可选组织化文件夹结构而言的。当您试图将一个文件添加到它的特殊文件夹之外时，IDE 就会发出警告，而且会提示创建该文件夹并将文件放在那里。例如，当试图向站点中添加一个类文件时(扩展名为.vb 或.cs)，就会看到如图 2-10 所示的警告。

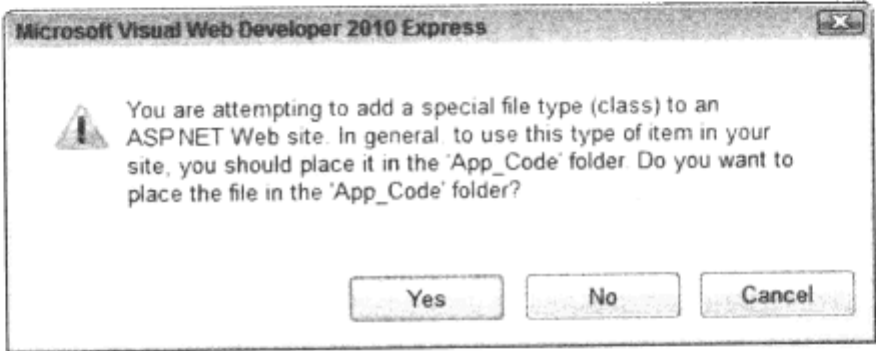


图 2-10

当看到这个对话框时，总是单击 Yes 按钮。否则文件就不能正确地起作用。其他文件类型也有类似的对话框，包括 skin 和数据库文件。

您已经对组成 Web 站点的各种文件类型有了很好的了解，现在我们就来更具体地看一下其中的一种类型：.aspx 文件，也称为 Web 窗体。

2.3 使用 Web 窗体

由.aspx 文件表示的 Web 窗体是任何一种 ASP.NET 4 Web 应用程序的核心。它们是用户访问站点时在其浏览器中看到的实际页面。

正如在第 1 章所看到的，Web 窗体可能由这些内容混合而成：HTML、ASP.NET 服务器控件、客户端 JavaScript、CSS 和编程逻辑。为了更容易地看到这些代码最终出现在浏览器中的状态，VWD 提供了关于页面的几种不同的视图。

2.3.1 关于 Web 窗体的不同视图

VWD 允许从几个不同的角度查看 Web 窗体。当一个带有标记的文件(如 Web 窗体或母版页)在文档窗口中打开时，会看到在窗口的左下方有 3 个按钮。使用这 3 个按钮可以在不同的视图间切换，如图 2-11 所示。

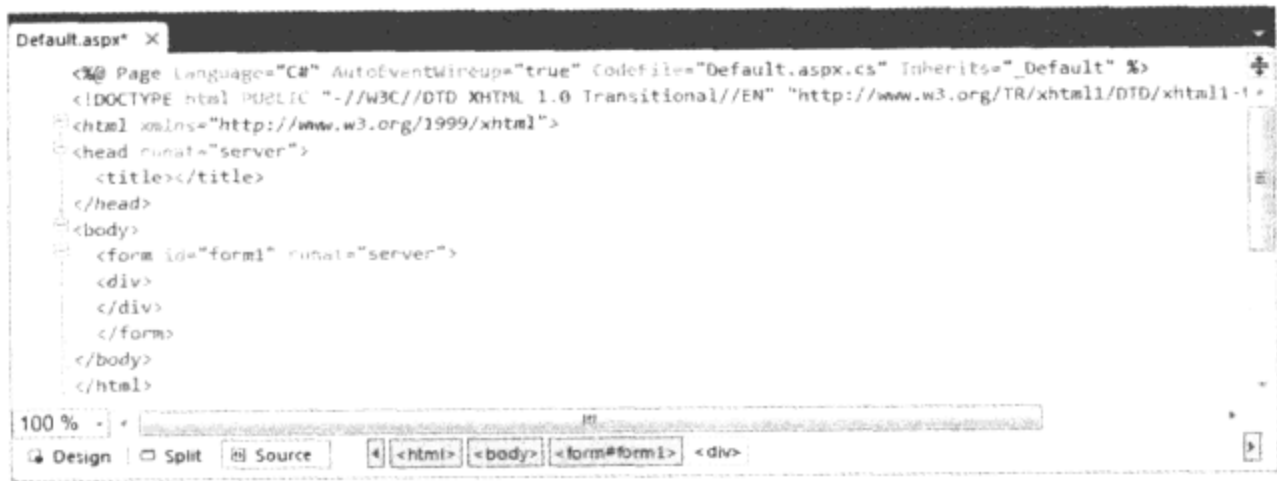


图 2-11

Source 视图是打开页面时的默认视图。Source 视图显示了原始 HTML 和页面的其他标记，而且如果要调整页面的内容或想修改某处时，它就非常有用。我们在第 1 章中已解释过，使用术语 Markup 视图来指代 ASPX 和 HTML 页面的标记，而不使用 Source 视图这一术语。

Design 按钮允许将文档窗口切换到 Design 视图，这样就能知道页面最终是什么样子。在 Design 视图中，可以用 View 主菜单下的 Visual Aids 和 Formatting Marks 子菜单来控制可视化标志，比如换行符、边界和空格。这两个子菜单都提供了一个 Show 菜单项，允许立即打开或关闭所有 Visual Aids。如果想看看页面最终出现在浏览器中的样子，那么把这两者都关闭是一个好主意。然而，只应当使用 Design 视图来了解页面最终的样子。虽然 VWD 有一个出色的呈现引擎，可以把页面相当漂亮地展现在 Design 视图中，但还是应当经常在不同的浏览器中检查页面，因为在 VWD 中看到的是页面处理前的标记。页面上的服务器控件可能会给出修改浏览器中页面外观的 HTML。因此，建议尽可能频繁地在浏览器中浏览页面，这样就能检查出页面的外观是不是您想要的样子。同时建议您在所能使用到的众多不同的浏览器中检测您的站点，因为它们显示 Web 页面的方式可能略有不同。Planet Wrox Web 站点已经用 Microsoft Internet Explorer、Firefox、Google Chrome、Safari 和 Opera 检测过了。在本书的不同地方，将会看到这些浏览器的屏幕截图。

Split 按钮可以用来同时查看 Design 视图和 Markup 视图，如图 2-12 所示。

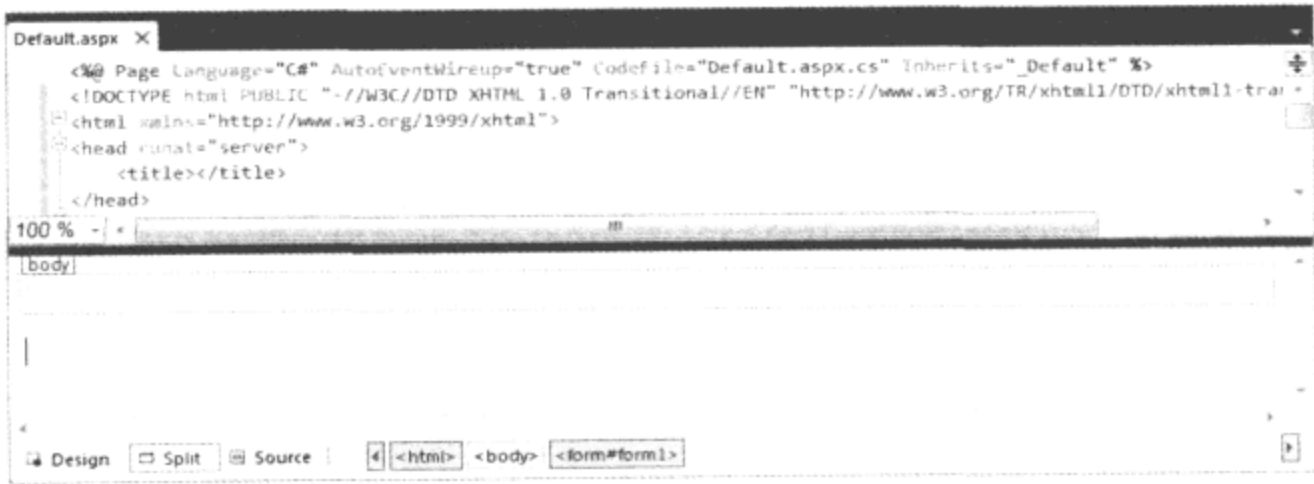


图 2-12

如果要查看向页面的 Design 视图中添加控件时 VWD 生成的代码，那么 Split 视图就是相当不错的选择。另一种方法也非常有用，即在 Markup 视图中修改页面的标记时，可以查看它最终在 Design 视图中的样子。有时 Design 视图会变得与 Markup 视图不同步。如果发生了这种情况，Design 视图上方会出现一条消息。只需单击这条消息或者保存整个页面就可以更新 Design 窗口。

如果希望在 Markup 视图以外的视图中打开页面，则选择 Tools | Options 命令。然后展开 HTML Designer，并且在 General 类别中设置您喜欢的视图。

除了 HTML 及您在 Markup 视图窗口中看到的其他标记外，Web 窗体也可能包含使用 C#或 Visual Basic .NET 编写的代码。代码所放的位置取决于所创建的 Web 窗体的类型。下一节将更详细地解释这两个选择。

2.3.2 在 Code Behind 和带内联代码的页面之间选择

Web 窗体有两种形式：一种是带 Code Behind 文件的.aspx 文件(根据带附加.vb 或.cs 扩展名的 Web 窗体命名的文件)，另一种是嵌套了代码的.aspx 文件，常称为带内联代码(inline code)的 Web 窗体。虽然在第 5 章之前您不会看到太多的代码，但是了解这些 Web 窗体类型之间的区别还是很重要的。一开始，带内联代码的 Web 窗体看起来要稍微容易理解一些。由于构建 Web 站点所需的代码是相同的 Web 窗体部分，因此可以清楚地看到代码与文件是如何关联的。然而，随着页面变得越来越大，您也向页面中添加了更多的功能，那时把代码放在单独的文件中通常会更加方便。那样的话，就把代码完全从标记中分离了出来，您就可以集中精力解决手头的任务。

在下面的“试一试”练习中将添加两个文件，它们演示了 Code Behind 和内联代码的区别。

试一试 向站点中添加带代码的 Web 窗体

将在本练习中添加的文件不是最终的应用程序所需要的文件。为了避免把项目弄混乱，应把它们放在一个单独的 Demos 文件夹中。

(1) 在 Solution Explorer 中，右击 Web 站点并选择 New Folder 命令。将文件夹命名为 Demos 并按下 Enter 键。

(2) 右击 Demos 文件夹并选择 Add New Item 命令。在出现的对话框的左方选择喜欢的编程语言，单击 Web 窗体模板并命名文件为 CodeBehind.aspx。确保选中了 Place Code in Separate File 复选框。最后，单击 Add 按钮。这个页面应当已在 Markup 视图中打开了，因此可以看到页面的 HTML。

(3) 在文档窗口下方，单击 **Design** 按钮将页面从 **Markup** 视图切换到 **Design** 视图。可以看到这个页面的背景是白色的，上方有一个小的虚线矩形。这个虚线矩形是在 **Markup** 视图中看到的<div>标记。

(4) 从 **Toolbox** 中，将 **Standard** 类别中的一个 **Label** 控件拖到页面的虚线框区域中。记住，如果 **Toolbox** 还没有打开，可以用快捷键 **Ctrl+Alt+X** 打开。在 **Design** 视图中，屏幕现在看起来应如图 2-13 所示。



图 2-13

(5) 双击<div>标记虚线下方某处的空白区域。VWD 就会从 **Design** 视图切换到文件的 **Code Behind** 模式，并添加在向浏览器中加载页面时激活的代码。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _  
    Handles Me.Load  
  
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)  
{  
  
}
```

虽然这种奇怪的代码这时看起来有点吓人，但是不要太担心。正如您所看到的，在大多数情况下，VWD 会自动添加它。在以后的章节中，将具体看到这样的代码是如何工作的，不过目前重要的是了解将要放在这些代码行之间的代码，在 **Visual Basic** 中是以 **Protected Sub** 开头，以 **End Sub** 结束，在 **C#**中是以花括号括起来的，当在浏览器中请求页面时就会运行它们。如果使用的是 **Visual Basic**，在该代码段中就看不到下划线。在这里添加下划线是为了将代码划分为两行。在本练习后面的“工作原理”部分您将会知道这样做的原因。

从现在开始看到的所有代码示例都包括 **Visual Basic(VB.NET)**和 **C#**版本，因此您总是能选择一种适合您的编程语言的代码。

(6) 把光标放在 VWD 创建的代码的起始行上，并将突出显示的指示今天的日期与时间的代码添加到 **label** 中，这个 **label** 最终会出现在浏览器中。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _  
    Handles Me.Load  
    Label1.Text = "Hello World; the time is now " & DateTime.Now.ToString()
```



```
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = "Hello World; the time is now " + DateTime.Now.ToString();
}
```

注意，一旦输入了表示 Label1 的 L，就会看到一个选项列表。这是 VWD 的 IntelliSense 功能，这是一个不错的工具，有助于快速地编写代码。不需要输入整个单词 Label1，只需简单地输入字母 L 或者字母 La，然后就可以从列表中选择正确的项，如图 2-14 所示。



图 2-14

要完成选中的单词，可以按下 Enter 或者 Tab 键，甚至可以按下句点键。在后一种情况下，会立即看到另一个列表，它允许通过输入前几个字母来挑选单词 Text，然后按下 Tab 或 Enter 键完成这个单词。这是一个效率很高的工具，因为只需稍微击几次键就能写出完整的代码。在许多其他的文件类型中也能使用 IntelliSense 功能，包括 ASPX、HTML、CSS、JavaScript 和 XML。在许多情况下，如果您开始输入，就会自动弹出这个带有选项的列表。如果没有弹出，可以按下 Ctrl+空格键来激活它。如果该列表覆盖了代码窗口中的部分代码，那么一直按下 Ctrl 键就可以使窗口变得透明。

(7) 右击 Solution Explorer 中的页面，并选择 View in Browser 命令。如果看到一个对话框询问是否要保存修改，单击 Yes 该页面就会出现在浏览器窗口中，如图 2-15 所示。

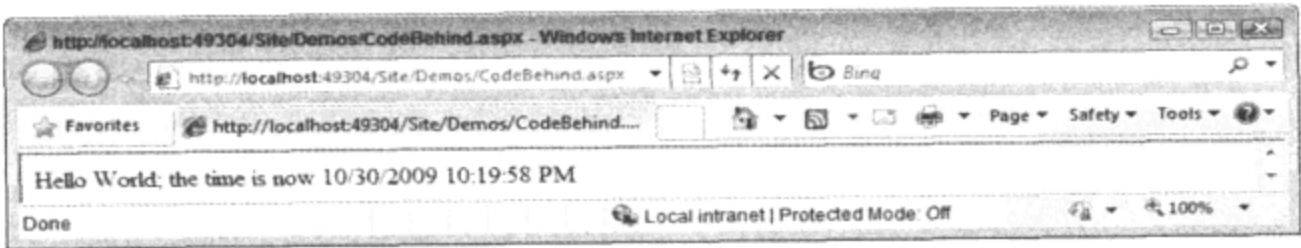


图 2-15

如果没有看到带有日期和时间的消息出现，或者在浏览器中看到了页面错误的消息，那么要确保已经将修改保存到了所有打开的页面中。要立即保存所有页面，按下 Ctrl+Shift+S 组合键或者单击工具栏上的 Save All 按钮(带有多个软盘符号的那个按钮)。此外，一定要确保输入了正确语言的代码。当创建这个新页面时，选择一种能应用到整个页面的编程语言。不能在一个页面中混合使

用两种语言。因此如果一开始使用的是 Visual C# 页面，则一定要输入“试一试”练习中的 C# 代码段。

(8) 用内联代码配置页面的方法非常类似。从向 Demos 文件夹中添加一个新的 Web 窗体文件开始。将其命名为 `Inline.aspx`，并确保取消选中 `Place Code in Separate File` 选项。

(9) 与第(3)、(4)、(5)步一样，将页面切换到 Design 视图中，拖动一个 label 控件到 `<div>` 标记内，然后双击现在包含该 label 的 `<div>` 之外页面的某处。VWD 现在不是打开一个 Code Behind 文件，而是将页面切换到 Markup 视图中，并直接在页面中添加 `Page_Load` 代码。

(10) 在 VWD 插入的代码块中的空行上，输入在本练习第(6)步中突出显示的代码行。确保使用正确的编程语言。最后在 `.aspx` 文件上应看到下列代码。

VB.NET

```
<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello World; the time is now " & DateTime.Now.ToString()
    End Sub
</script>
```

C#

```
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = "Hello World; the time is now " + DateTime.Now.ToString();
    }
</script>
```

(11) 右击 Solution Explorer 中的页面并选择 View in Browser 命令。也可以按下 `Ctrl+F5` 组合键在浏览器中打开页面。您应看到与第(7)步相同的页面。

工作原理

在运行时，带内联代码的页面的行为与使用 Code Behind 文件的页面相同。在这两种情况下，ASP.NET 运行库看到 `Page_Load` 代码并执行它在其中发现的任何代码。在这个“试一试”练习中，这意味着将 `Label 1` 的 `Text` 设置为欢迎消息和今天的日期与时间。

注意在本例中，C# 代码看起来与 VB.NET 代码非常相似。设置 Label 的文本的代码在两种语言中几乎相同。一个区别在于 VB.NET 用 `&` 符号把两块文本连在一起，而 C# 用的是加号 `(+)` 字符。还可以在 VB.NET 中使用加号将字符串连接起来，但是会遇到第 5 章中将要介绍的一些警告。另一个区别是在 C# 中所有的代码行都必须以分号 `(;)` 结尾，表示代码单元的结束，而 Visual Basic 用的是换行符。如果希望将使用 Visual Basic 语言编写的一个长的代码行分成多行，则需要使用下划线字符。在之前的版本中，VB.NET 在很多不同的地方都需要使用下划线。但是，在配置了 Visual Web Developer 2010 的 Visual Basic10 版本中，该语言的设计者大大减少了需要使用下划线的次数。

如果希望恰好在 `Handles` 关键字前将代码分为多行，则需要使用下划线，请参看练习第(6)步中的代码段：

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
```



```
Handles Me.Load
Label1.Text = "Hello World; the time is now " & DateTime.Now.ToString()
End Sub
```

注意在页面中，Visual Basic 示例第一行的末尾没有下划线()。这里添加它是因为本书的页面的宽度不够在一行中显示整个代码语句。在本书其余部分的其他 Visual Basic 示例中您将会看到更多这样的下划线。如果您决定手动输入下划线来使代码更可读，则不要忘记在实际下划线之前多输入一个空格，否则代码会无法运行。

在 C#中就不需要这个字符，因为该语言本身允许按下 Enter 键来中断太长的行。这是因为 C# 使用了分号专门作为行结束符，而不是在源代码中使用换行符来表示结束。

可以通过右击 View in Browser 选项或者按下 Ctrl+F5 组合键在浏览器中打开页面。使用 View in Browser 选项，总是可以打开右击的页面。使用 Ctrl+F5 快捷键，可以打开在文档窗口中当前活动文档的页面，这是当前在 Solution Explorer 中选中的页面，或者打开设置为 Web 站点的起始页(start page) 的文件。此外，所有打开的文件都会自动保存，并且在浏览器中打开请求页面之前会检查站点的错误。

可以通过在 Solution Explorer 中右击并选择 Set As Start Page 命令将一个页面指定为起始页。如果想在后面的阶段控制这个行为，则右击 Solution Explorer 中的 Web 站点，并选择 Property Pages 命令。在 Start Options 类别中，可以指明希望打开当前活动页面，也可以指定一个特定页面，如图 2-16 所示。

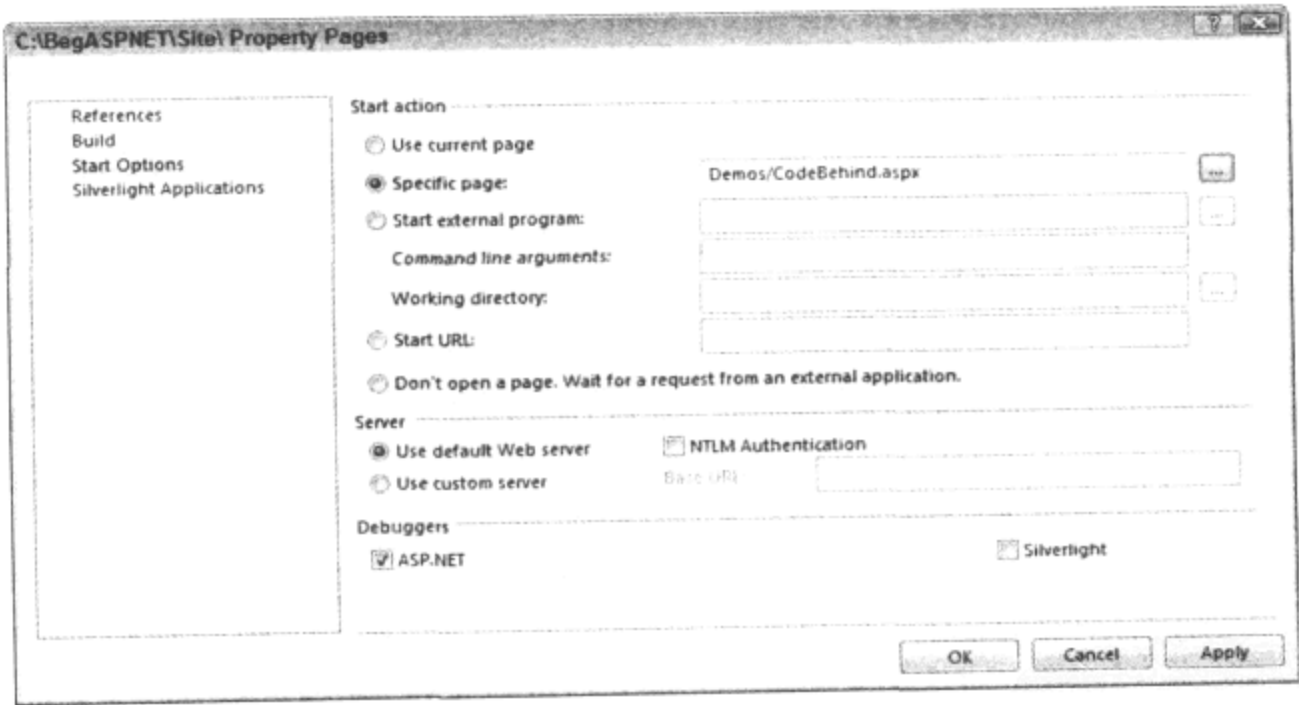


图 2-16

在上面的练习中，介绍了如何添加一个包含简单的 Label 控件的页面。此外，您还看到了如何写一些更新有今天的日期与时间的 label 的代码。暂时可以忽略该代码；这里提到它只是为了演示 Code Behind 代码和内联代码之间的区别。在第 5 章将会介绍用 Visual Basic 和 C#编程的更多知识。

为了制作一个引人注目的页面，需要的内容显然远远不止一个显示今天日期和时间的简单的 Label 控件。下一节将介绍如何向页面中添加内容和 HTML，以及如何定制它的样式和格式。

2.3.3 向页面添加标记

有几种方式可以向页面中添加 HTML 和其他标记。首先，可以简单地在 Markup 视图窗口中输入。然而，这并不总是最好的选择，因为它会强制手动输入大量代码。要更容易地向页面中插入新 HTML 并向其应用格式，Design 视图窗口提供了几个有用的工具。这些工具包括 Formatting 工具栏和菜单项 Format 和 Table。要使这些工具起作用，需要在 Design 视图中包含它们的文档。如果是在 Split 视图模式下工作，则一定要确保焦点在 Design 视图部分，否则就会发现大多数工具都不可用。

1. 插入和格式化文本

在 Design 视图和 Markup 视图中都可以输入文本。只需将光标放在所需的位置并开始输入即可。当切换到 Design 视图时，Formatting 工具栏就变得可用，其中的选项如图 2-17 所示。

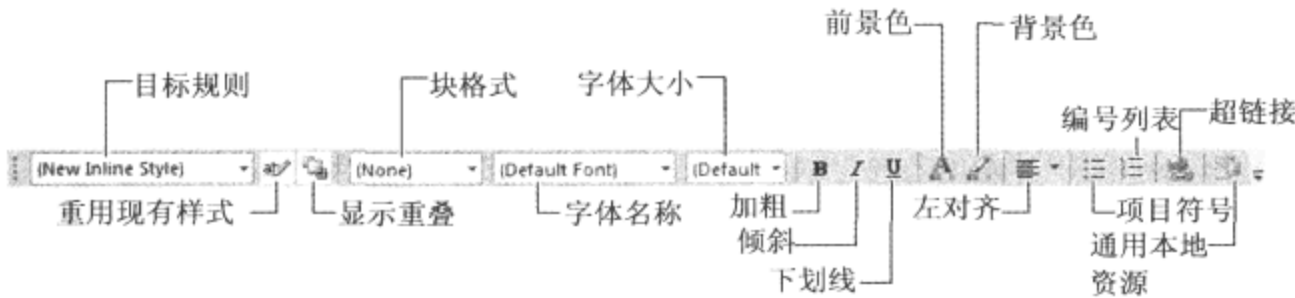


图 2-17

使用块格式下拉列表可以插入 HTML 标记，如<p>表示段落，<h1>~<h6>表示标题，、、标记表示列表。可以直接从这个下拉列表中选择一项插入页面中，或者也可以先选择一些文本，然后从列表中选择适当的块元素，并把选中的文本包括在标记内。

字体名称下拉列表允许修改字体，字体大小下拉列表可以用来修改字体的大小。

这个工具栏上的许多按钮的功能与其他编辑环境中的相应按钮功能完全相同。例如，B 按钮用粗体字格式化文本。类似地，I 和 U 按钮分别使字体倾斜和添加下划线。

在下面的“试一试”练习中，将看到如何使用这些工具来创建 Planet Wrox Web 站点的主页。

试一试

添加带格式的文本

在本“试一试”练习中，将创建一个名为 Default.aspx 的 Web 窗体并向其添加一些基础内容。

(1) 通过 Add New Item 命令在站点对话框的根位置添加一个新的 Web 窗体并将其命名为 Default.aspx。确保选中的是 Place Code in Separate File 选项并单击 Add 按钮。使用文档窗口下方的 Design 按钮切换到 Design 视图。

(2) 在虚线矩形内单击，直到看到表示<div>标记当前是活动状态的“浮起”。与此同时，代码窗口下方的标记导航器应突出显示最后一个块，它上面有文本<div>，如图 2-18 所示。

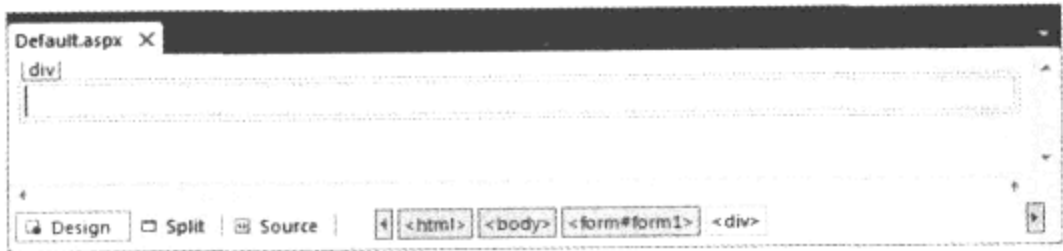


图 2-18

(3) 输入 “Hi there visitor and welcome to Planet Wrox”，并使用鼠标突出显示这段文字。从块格式下拉列表(参见图 2-17)中选择 Heading 1 <h1>。注意这时有一个带有文本 h1 的小浮块出现在这段文本的正上方，表示 VWD 自动为您创建了一个标题。图 2-19 显示了有<h1>标记的 Design 视图。



图 2-19

(4) 把光标放在单词 Wrox 后的标题末尾并按下 Enter 键。这时会插入一个新段(由一个小浮起表示，上面有字母 p)，因此就可以直接开始输入了。

(5) 输入图 2-20 中显示的文本(或是自己构思的文本)，对 Planet Wrox 站点的访问者发出欢迎消息。注意一输入逗号，文本 “www.PlanetWrox.com” 就变成了蓝色，以表明 VWD 已经将其看作是一个 Web 地址，并且将其变成了一个链接。可以按 Enter 键来开始一个新的段落。选择文本 “paying a visit”，单击 Formatting 工具栏上的 Foreground Color 按钮，并在出现的对话框中选择一个不同的颜色。然后选择一些别的文本，如 “reviews and concert pictures”，并单击 Bold 按钮。此时，Design 视图的显示应如图 2-20 所示。



图 2-20

此主页的代码现在看起来应如下面所示(代码已经被略微重格式化以适合本书的空间):

```
<div>
  <h1>Hi there visitor and welcome to Planet Wrox</h1>
  <p>
    We're glad you're paying a visit to
    <a href="http://www.PlanetWrox.com">www.PlanetWrox.com</a>,
    the coolest music community site on the Internet.
  </p>
  <p>
    Feel free to have a look around; there are lots of interesting <strong>reviews
    and concert pictures</strong> to be found here.
  </p>
</div>
```

```
</p>
</div>
```

(6) 按下 Ctrl+F5 组合键在浏览器中打开该页面，或者在 Solution Explorer 中右击该页面然后选择 View in Browser 命令。

工作原理

当使用各种 Formatting 工具栏按钮时，如 Foreground Color，VWD 会为您插入适当的 HTML 和 CSS 代码。例如，当单击 B 按钮时，VWD 会在选中的文本两头插入一对标记。当单击 I 按钮时，它就会添加一对标记使文本倾斜。在这个练习中，当改变文本颜色时，VWD 还会插入一个 class 特性(前面的代码示例中出现过)，指向一个名为 style1 的类。这种样式的代码将添加到文件的上方，代码应如下所示：

```
<style type="text/css">
  .style1
  {
    color: #FF0000;
  }
</style>
```

如果选择的是不同的颜色，那么代码可能和上面略有不同。在这里所看到的代码将在第 3 章解释。目前，只需记住这段代码将文本颜色设置为红色即可。

注意 VWD 用它的与 HTML 兼容的变体“we’re”替换欢迎消息中的“we’re”里面的单引号(‘)。使用这种代码可能向页面中插入浏览器显示起来有些麻烦的字符，或者在 HTML 中有特殊含义的字符，如&，它就写作&。当在 Design 视图中输入文本时，VWD 会自动插入相关字符的等价代码；然而，如果直接在 Markup 视图中输入，就必须亲自替换。

如果您的代码和这里显示的不一样，不要担心。VWD 中的许多设置都会影响生成的代码。

到目前为止，这些练习都是关于在页面中添加和样式化文本的。然而，VWD 也允许插入其他 HTML 标记，比如表和项目符号。下一节将说明它们的工作原理。

2. 添加表与其他标记

如果需要显示结构化或重复的数据，如购物车中的产品清单、相册中的照片或者表单中的输入控件，则 HTML 表相当好用。关于是否要用表来布置页面，网上也有许多争论。例如，如果页面中包含一个带 logo 的标题，一个主内容区，下方还有一个页脚，就可以使用一个 3 行的表来完成它。总体而言，为了这个目的而使用表不是太好的主意，因为它们会向页面中添加大量无关的标记，而且常常难以维护。此外，使用 CSS 往往可以达到同样的效果，这将在第 3 章中讲解。尽管表可能带来这些缺点，但是在显示表格或其他结构化信息时，表仍然是 HTML 工具箱中的重要工具。

试一试

使用 Format 和 Table 菜单

本练习将指出如何用 Table 菜单向页面中添加表，以及如何添加行和列。此外，还将说明如何

添加其他结构化元素，比如项目符号列表。

- (1) 在 Demos 文件夹中，创建一个新的 Web 窗体，名为 TableDemo.aspx。通过选中 Place Code in Separate File 选项确保它使用 Code Behind 文件。
- (2) 将页面切换到 Design 视图，在页面中的标准<div>标记的虚线框内单击，并选择 Table | Insert Table 命令。这时会出现 Insert Table 对话框，如图 2-21 所示。

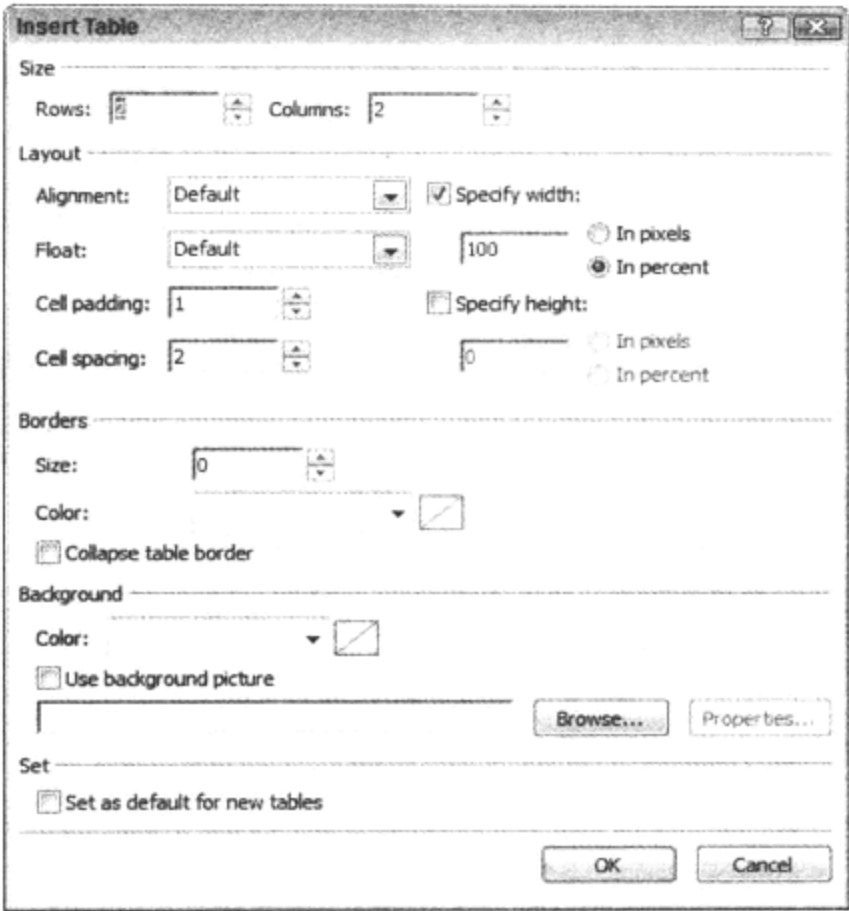


图 2-21

- (3) 将 Rows 设置为 3，Columns 为 2。保持其他设定项为默认值，并单击 OK 按钮。这个表就会插入到页面中。
- (4) 如果只看到一个表单元格，而不是有 3 行 2 列的完整表，就需要启用表的 Visual Aid。要做到这一点，从主菜单中选择 View | Visual Aids | Visible Borders 命令来打开边界。此时 Design 视图应如图 2-22 所示。

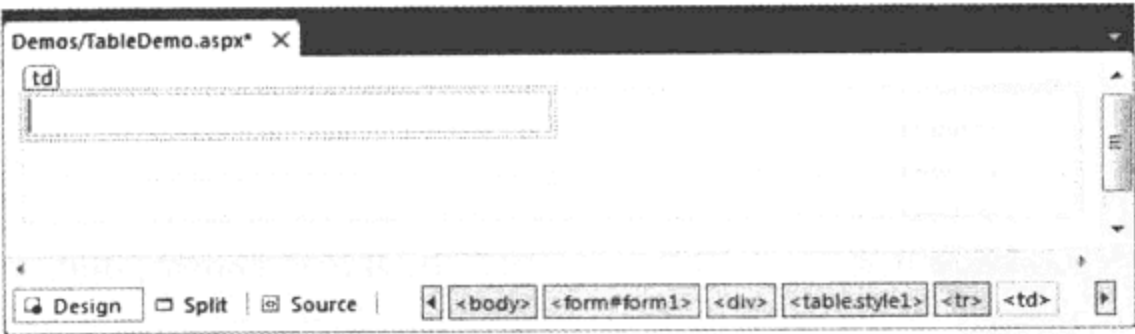


图 2-22

- (5) 把表中第一个单元格的右边界拖到左边。这时就可以看到一个可视化指示器显示了单元格的宽度。把它往左边拖直到宽度为 200 像素，如图 2-23 所示。



图 2-23

- (6) 要将更多行或列添加到表中，可以右击现有单元格。从出现的弹出菜单中选择 **Insert** 命令在不同位置添加额外的行或列。类似地，可以用 **Delete**、**Select** 和 **Modify** 选项删除行或列、合并单元格或进行选择。对于本练习，不需要添加额外的行或列，但如果已经添加了也没有关系。
- (7) 把光标放在第一行的第一个单元格中，输入 **Bulleted List**。
- (8) 把光标放在第一行的第二个单元格中，并从 **Format** 菜单中选择 **Bullets and Numbering** 命令。
- (9) 切换到 **Plain Bullets** 选项卡，单击带圆形实心项目符号的图片(如图 2-24 所示)，并单击 **OK** 按钮。

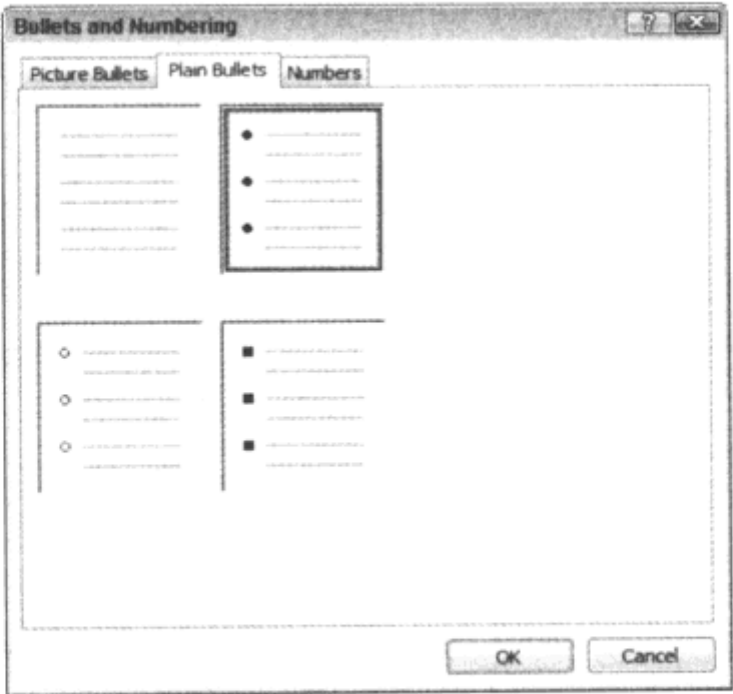


图 2-24

- (10) 输入一些文本，例如喜欢的音乐类型(**Punk**、**Rock** 及 **Techno** 等)，然后按下 **Enter** 键。VWD 会自动为您插入一个新的项目符号，从而可以继续向列表中添加新的项。再添加两种类型，这样就有 3 个项目符号了。
- (11) 重复第(7)~(10)步，但这次创建一个编号列表。首先，在第二行的第一个单元格中输入一个 **Numbered List**，然后将光标放到同一行的第二个单元格中，并选择 **Format | Bullets and Numbering** 命令。切换到 **Numbers** 选项卡(如图 2-24 所示，位于 **Plain Bullets** 后边)并单击第一行中的第二个图，它显示了一个标准的编号列表，然后单击 **OK** 按钮。为列表输入一些项，每输入一个项就按一次 **Enter** 键。
- (12) 按下 **Ctrl+F5** 组合键在浏览器中打开页面。这时应看到一个如图 2-25 所示的屏幕。

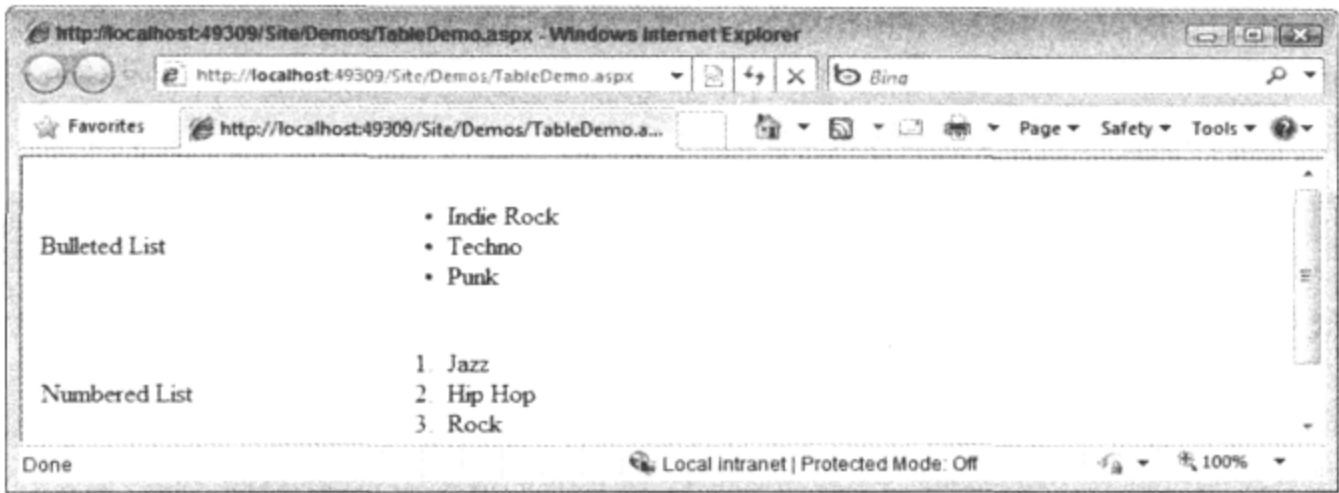


图 2-25

工作原理

当通过可用的菜单可视化地插入了一些页面元素，比如表或列表时，VWD 就会在 Markup 视图中插入必需的标记。当插入一个表时，VWD 会添加一个<table>标记和几个<tr>及<td>标记，来分别定义行和单元格。它还向表应用了一个 style 属性来控制表的宽度。如果把列的宽度设为 200 像素时它还会为<td>标记创建另一种样式。类似地，当插入一个列表时，VWD 会插入一个标记表示编号或有序列表，插入一个标记表示项目符号列表或无序列表。在这些标记内，用标记定义列表中的每个项。

除了到目前为止所看到的 HTML 标记外，还需要了解另一个重要标记：<a>标记，它用来在页面之间创建链接。

2.3.4 连接页面

Web 站点的一个重要功能是链接站点中的页面。链接允许访问者从一个页面进入同一站点中的另一个页面，或者进入到 Internet 上完全不同站点中的页面。在页面之间创建链接的方式有几种，包括：

- HTML <a>元素，已在本章解释。
- 使用<asp:HyperLink>控件，将在第 7 章讨论。
- 通过代码编程，这将在本书后面讨论。

下面的练习显示了如何轻松地从一个页面链接到另一个页面。

试一试

链接页面

在本“试一试”练习中，将通过添加链接到另一个页面的文本来修改早先创建的页面 TableDemo.aspx。一旦在浏览器中运行该页面并单击那个链接，新页面就会替换老页面。

- (1) 从 Demos 文件夹中打开页面 TableDemo.aspx。
- (2) 如有必要，切换到 Design 视图。
- (3) 在第三行的第一个单元格中，输入文本 Link。
- (4) 在同一行的第二个单元格中，输入文本 Go to the Homepage of Planet Wrox 并用鼠标突出显示它。
- (5) 在 Formatting 工具栏上，单击 Convert to HyperLink 按钮。它是工具栏上的最后一个按钮，

上面有一个绿色的地球仪。如果没有看到这个按钮，那就是因为它被其他工具栏遮住了。可以将这个 Formatting 工具栏拖到一个新位置，或者选择 Format | Convert to Hyperlink 命令打开同样的对话框。

(6) 在出现的对话框中，单击 Browse 按钮并定位到站点根文件中的 Default.aspx 页面，然后单击 OK 按钮。再次单击 OK 按钮关闭 Hyperlink 对话框。此时页面的 Design 视图应如图 2-26 所示。

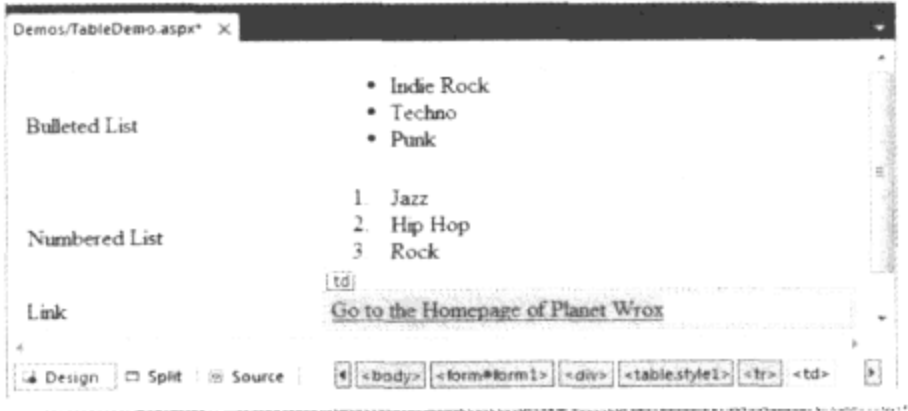


图 2-26

(7) 切换到 Markup 视图并注意链接的 HTML 是如何插入的：

```
<a href="../../../Default.aspx">Go to the Homepage of Planet Wrox</a>
```

注意 href 属性指向要链接到的页面。

(8) 如果要从 Markup 视图中修改要链接的页面，单击 href 属性的起始和结束引号之间的某处，并按下 Ctrl+空格键。这时会弹出一个对话框，允许您选择另一个页面。此外，也可以单击 Pick URL 选项，并浏览站点中某处的新页面。

(9) 右击 Solution Explorer 中的页面 TableDemo.aspx，并选择 View in Browser 命令。当页面完成加载后，单击 Go to the Homepage of Planet Wrox 链接。请求将被发送到 Web 服务器，作为响应，现在得到了 Web 站点的主页。

工作原理

页面之间的链接很可能是一个 Web 页面中最重要的元素，因为它们允许在自己的站点中的一个页面和另一个页面之间创建连接，不论那个页面是自己的站点中的一个页面，还是在 Internet 上某处一个完全不同的服务器上的页面。对于应出现在页面上某处的简单链接来说，最容易创建的是带有一个 href 属性集的 HTML <a>标记。当用户单击这样一个链接时，浏览器从服务器上请求新页面并显示它。href 值中的两个点(.)指的是父目录。完整的 href 属性表示“返回文件夹的层次结构中的上一层，然后选择文件 Default.aspx”。在第 7 章中将会看到关于链接的更多内容及其工作原理。

注意并不限制您只能链接到自己的站点的页面。如果要链接到外部页面，只需用下面示例中显示的完整页面地址替换 href 属性即可。

```
<a href="http://www.wrox.com">Go to the Wrox Homepage</a>
```

重要的是要包括 http://前缀。否则浏览器就会在 Web 站点上寻找名为 www.wrox.com 的文件或文件夹。

在第 3 章中将用到本章关于页面创建与格式化的知识来处理用 CSS 设计 Web 页面的事务。

除了可视化工具(如格式化工具栏和表菜单)外，Visual Web Developer 还有另一种在页面中快速插入代码的好方法：代码段。代码段允许只使用很少的击键次数，就可以向页面中插入大的代码块。

在第 3 章中将会看到运行的代码段。

2.4 使用 Web 窗体的实用提示

下面是使用 Web 窗体的一些提示：

- 总是试着使用 Code Behind 模式窗体而不是内联代码设计 Web 窗体。虽然一开始您可能不会注意到两者之间的明显区别，但是随着站点和页面的扩大，您将会发现使用代码与标记分离的页面会轻松得多。
- 花一些时间熟悉 Format 和 Table 菜单中不同的菜单项。它们中的大部分都会生成可以插入到页面中的 HTML 标记。看一下为您生成的 HTML 标记与属性，并试着在代码中直接修改它们，然后再通过菜单和工具栏做一遍。通过这种方式，可以对各种可用的标记及它们的行为有一个良好的感觉。
- 试验连接站点页面的链接。注意根据要链接页面的位置，VWD 会创建不同的链接。第 7 章会更详细地介绍链接和定位站点中的页面的各种方式。

2.5 本章小结

本章介绍了一些帮助构建可维护和结构化的 ASP.NET Web 站点的重要主题。了解不同项目类型和模板之间的区别后，可以只用所需的文件构建一个 Web 项目。

这也适用于可以添加到站点中的不同文件类型。由于各个文件类型都有特定的用途，因此重要的是要知道那个用途是什么，以及可以如何使用该文件。

在构建 ASP.NET Web 页面时会执行的一个常见操作是向页面中添加标记。正如您在本章和第 1 章所看到的，标记有几种形式，包括纯 HTML 和 ASP.NET 服务器控件。知道如何用 VWD 提供的大量菜单选项和工具栏向页面中添加标记，对于构建漂亮的 Web 页面是至关重要的。

现在您已经对于创建和修改 Web 窗体有了扎实的理解，下一步将了解如何把那些只有几个控件的黑白页面变成有吸引力的 Web 页面。第 3 章将介绍如何使用 VWD 中的许多 CSS 工具来创建所需的效果。

2.6 练习

1. 指出 Web Files 类别中 3 个可以添加到站点中的重要文件。描述各文件的用途。
2. 如何使 Web 页面中的一块文本既加粗又倾斜？最终的 HTML 是怎样的？
3. 指出在 VWD 中向 ASP.NET Web 站点添加现有文件的 3 种不同方式。
4. VWD 为 ASPX 页面提供了哪些不同的视图？VWD 中有没有提供别的视图？

练习的答案见附录 A。

本章要点回顾

Code Behind	一种在独立的代码文件中存储服务器端代码的页面模式
Design 视图	提供页面的图形表示
File Types	ASP.NET 支持多种不同的文件类型，包括 WebForms(.aspx)，母版页(.master)，CSS 文件(.css)，JavaScript(.js)以及 SQL Server 数据库(.mdf)
内联代码	一种页面模型，将服务器端代码与标记存储在相同的文件中
Markup 视图	允许查看页面标记
项目模板	通过针对特定的应用场合建立站点，启动 Web 开发
项目类型	VWD 提供了两种项目类型：Web Application Project 和 Web Site Project
Split 视图	允许同时查看 Markup 视图和 Design 视图
Web 窗体	在客户端上显示 Web 站点的用户界面

本章要点

- CSS 的定义及需要 CSS 的原因
- CSS 语言简介及如何编写 CSS
- 向 ASP.NET 页面和外部文件添加 CSS 代码的各种不同方法
- VWD 提供的大量快速编写 CSS 的工具

在前两章中创建的页面看起来相当单调且无趣。那是因为它们缺少样式化信息，并默认应用了浏览器的标准布局。要把页面打扮得漂漂亮亮的，需要使用某种方法来改变它们在浏览器中的表现 (presentation)。最常用的方法是使用 CSS(Cascading Style Sheet, 层叠样式表)语言。CSS 是在 Web(包括 ASP.NET Web 页面)上格式化和设计信息的实际语言。使用 CSS 可以快速地改变 Web 页面的外观，创建您自己设计的或是公司统一要求的完美外观。

Visual Web Developer 的早期版本——VWD 2008 对使用 CSS 提供了大力支持。新的 VWD 2010 在它的基础上从很多方面提高了对 CSS 的支持，包括把页面呈现得与它们最终在浏览器中显示的效果非常接近这一功能。这些 CSS 工具有助于可视化地创建 CSS 代码，使得页面的样式化更加容易，而不需要知道或记得 CSS 的所有细枝末节。

要了解 CSS 在 ASP.NET Web 站点中的关联性和必要性，需要先了解 HTML 的缺点。下一节将指出纯 HTML 表现存在的问题，以及 CSS 如何克服这些问题。

3.1 需要 CSS 的原因

从 Internet 出现伊始，Web 页面主要由文本和图像组成。文本是使用纯 HTML 格式化的，用 这样的标记使文本加粗，并用标记影响字体、大小和颜色。Web 开发人员很快就发现他们需要更强大的功能来格式化页面，因此诞生了 CSS 以弥补 HTML 在样式方面的缺陷。

3.1.1 HTML 格式化的问题

使用 HTML 进行格式化的问题之一是它提供的样式化页面的选项很有限。可以用、及这样的标记来改变文本的外观，用 bgcolor 这样的属性来改变 HTML 元素的背景颜色。还有几个属性可用来改变链接出现在页面中的方式。

显然，这个功能集不足以创建符合用户期望与需求的生动 Web 页面。

HTML 影响 Web 页面构建的另一个问题是样式化信息应用到页面的方式。在设计时，HTML 会强制要求在 HTML 文档中嵌入格式化信息，使得以后难以对设计进行重用或修改。如下面这个示例：

```
<p><font face="Arial" color="red" size="+1">
  This is red text in an Arial type face and slightly larger than the default text.
</font></p>
```

这段代码的问题在于实际数据(<p>元素中的文本)与表现(在本例中是用标记格式化的文本)混淆在一起。理想情况下，这两者应当分开，以便各自能方便地修改而不会互相影响。

假设在站点的各个页面中用<p>和标记来标识第一段。很显然，这样的代码很难维护。在决定把字体的颜色从红色改为深蓝色时，会发生什么状况呢？或者，如果公司统一要求使用 Verdana 字体而不是 Arial 字体时，会出现什么情况呢？结论是在做必需的修改时要访问站点的每个页面。

除了维护性问题外，HTML 格式化的另一个问题是：在用户的浏览器中不能轻松地在运行时修改格式。至于上面的代码段中的 HTML，没有什么方法可以让访问者修改字体大小或颜色这样的属性，但对视觉有障碍的人来说常常有这样的要求。如果想给访问者提供另一种采用较大的字体和不同颜色的版本，就需要为原始页面创建一个副本，再进行必要的修改。

HTML 格式化的最后一个问题是，页面中的附加标记大大增加了页面的大小。这样，由于需要从 Web 站点中的各个页面上下载信息，下载和显示就会变慢。而且，当需要滚动大型的 HTML 文件来查找需要的内容时，页面也会变得难以维护。

简言之，使用 HTML 格式化存在以下这些问题：

- 它的有限功能集远远满足不了页面的格式化需求。
- 数据与表现混合在相同的文件中。
- HTML 无法在浏览器中于运行时轻松地切换格式。
- 必需的格式化标记与属性使页面更大，因此加载和显示更慢。

幸运的是，CSS 能够解决所有这些问题。

3.1.2 CSS 如何解决格式化问题

CSS 几乎能在所有可能的方面格式化 Web 页面。它提供了一套丰富的选项，可以修改 Web 页面的各个细微方面，包括字体(大小、颜色、字体等)、颜色和背景色、围绕 HTML 元素的边框、元素在页面中的位置以及其他很多方面。当今所有的主流浏览器基本都能接受 CSS，它就是 Web 页面可视化表现的语言，而且在 Web 开发人员中非常流行。

CSS 允许在外部文件中定义所有的格式化信息，从而解决了数据与表现混在一起的问题。然后，ASPX 或 HTML 页面就能引用这些文件，并且浏览器会应用正确的样式。有了这样的分离，HTML 文档中就仅含有要显示的内容，而 CSS 文件会仅定义显示的方式。因而，可以修改或替换这两个文

档之一，而让另一个文档保持不变。此外，CSS 可以直接放在 HTML 或 ASPX 页面中，这样就可以添加真正必要的 CSS 小代码块。直接把 CSS 放在 HTML 或 ASPX 页面上时要谨慎，因为那样就不再能从一个集中的地方控制样式信息了。

由于所有的 CSS 代码可以放在单独的文件中，因此很容易让用户在两个样式之间选择，例如，采用较大字体的样式。可以创建外部样式表的一个副本来进行必要的修改，然后向用户提供这个可选的样式表。在第 6 章讨论 ASP.NET 主题时将介绍它的工作原理。

独立样式表文件的另一个好处是它降低了站点的带宽需求。由于样式表不会随着每个请求而改变，因此在第一次下载时浏览器会保存样式表的一个本地副本。从那时起，它就会采用这个缓存副本，而不是一再地从服务器上请求它。有时当浏览器还没有下载修改过的最新的 CSS 文件时，这种缓存就能派上用场。如果发现浏览器没有显示对 CSS 文件所做的修改，请在浏览器中(不是在 VWD 中)使用 Ctrl+F5 或 Ctrl+R 键从服务器中获得一个刷新后的副本。

上面介绍了 CSS 之所以重要的原因，下面我们介绍它的呈现方式以及如何使用它。

3.2 CSS 简介

就语法而言，CSS 是一种容易学习的语言。它的“语法”仅由几个概念组成，使得它相当容易入门。CSS 的难点在于所有主流浏览器呈现页面的方式。尽管实际上每种现代桌面浏览器都能够理解 CSS，但当根据 CSS 标准显示页面时，它们都有各自的“怪癖”。这个标准是由提出 HTML 标准的同一个组织 W3C(World Wide Web Consortium)提出的，它出现过 3 个版本：1.0、2.1 和 3.0。在这 3 个版本中，2.1 版本是现在人们最普遍接受的，它包含了版本 1.0 中的所有内容，但是在其基础上又增加了大量功能。这个版本也是 VWD 默认使用和生成的版本。版本 3.0 目前正在开发中，预计不久的将来主流浏览器就会大力支持它。

在学习 CSS 的实际语法之前，最好先看一个示例。在下面的“试一试”练习中，将会写一个简单的 ASPX 页面，该页面包含了一些 CSS 以格式化页面内容。这个练习有助于理解 CSS 语言，在练习后面的小节中将会全面讨论这个语言。

试一试

写第一个 CSS

在本“试一试”练习中将写一些 CSS 来修改一个标题和两个段落的外观。首先手动为页面编写代码；本章的后半部分会介绍如何在 VWD 中使用可用的 CSS 工具。

(1) 在 Planet Wrox 项目中的 Demos 文件夹中新建一个 Web 窗体，命名为 CssDemo.aspx。对于本练习来说，选择内联代码或 Code Behind 都没有关系。

(2) 确保该页面在 Markup 视图中，然后在源代码中定位</title>结束标记。把光标放在这一行的最后，按下 Enter 键，在 title 和 head 标记中间空出一行。在这一新行上输入 style 然后按下 Tab 键。VWD 将会自动完成<style>元素的添加。再按两次 Enter 键，在标记之间留出一些空间。最终可以得到如下的粗体代码：

```
<title></title>
<style type="text/css">
```

```
</style>
</head>
```



注意：代码完成功能使用代码段把一部分代码(如<style>元素)和一个标识符(在本例中是 style)关联起来。这个代码段对于快速插入一段代码非常有用，只需输入一个简写的标识符即可。许多代码段都是可用的，在学习本书时，会在适当的地方指出它们。

除了使用样式代码段，还可以手动输入整个代码。注意，一旦输入了起始尖括号(<)，就会弹出一个列表供您选择<style>标记。type 属性也是如此，只要输入字母 ty，列表中就会预选了 type 属性。只需按下 Tab 或 Enter 键完成单词即可。此外，属性值 text/css 也有这样的帮助功能，只需在列表中选择它，并按下 Tab 或 Enter 键，它的值就会自动插入，而且带有完好的双引号。

(3) 接下来，在起始和结束<style>标记之间，输入下面突出显示的 CSS 代码：

```
<style type="text/css">
  h1
  {
    font-size: 20px;
    color: Green;
  }

  p
  {
    color: Blue;
    font-style: italic;
  }

  .RightAligned
  {
    text-align: right;
  }
</style>
```

输入这段代码时要非常小心，因为 CSS 的语法相当严格。列表中的第一项是样式化一级标题的 h1 标记，因此它的大小为 20 像素，并显示为绿色字体。注意 font-size 和 20px 之间用冒号隔开，且该行以分号结束。

列表中的第二项仅含有字母 p，它定义了页面中所有<p>元素的外观。

最后一项有一个前缀句点(.)，后面跟着文本 RightAligned。这个项用来右对齐页面中的文本。

(4) 把页面稍微向下滚动一点，直至看到起始<div>标记。在这个标记后面输入下面的突出显示代码：

```
<div>
  <h1>Welcome to this CSS Demo page</h1>
  <p>CSS makes it super easy to style your pages.</p>
  <p class="RightAligned">
    With very little code, you can quickly change the looks of a page.
  </p>
</div>
```


如果不想直接输入这段代码，也可以在 Design 视图中，使用 Formatting 工具栏来创建像<h1>和<p>这样的元素。在这里，需要切换到 Markup 视图来输入 class="RightAligned"，但是在本章以后的练习中，就可以看到如何让 IDE 自动写出这个代码。

(5) 如果切换到 Design 视图(或者 Split 视图)中，将看到设计器会用在页面的<style>元素中定义的格式显示文本。图 3-1 显示了 Split 视图中的页面，这里可以同时看到代码与设计效果。

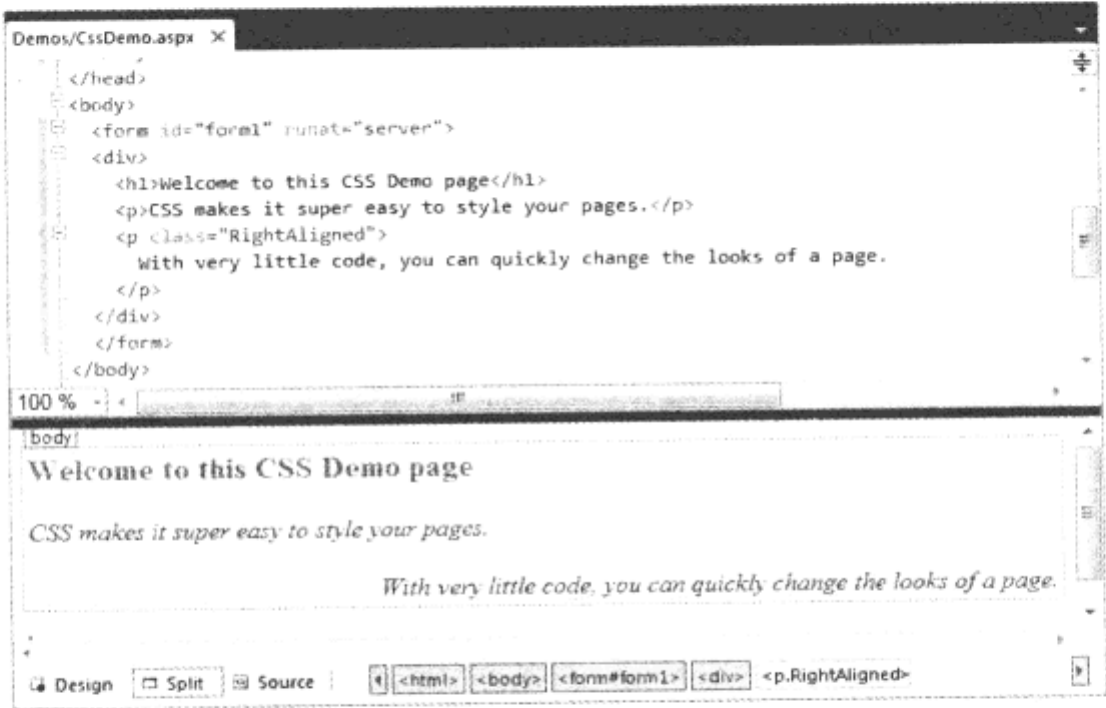


图 3-1

尽管本书是黑白版的，不太容易看出不同的字体颜色，但是从图 3-1 中可以清楚地看出<h1>的字体更大。图 3-1 还表明，所有段落(纯段落和带 class="RightAligned"的段落)现在都是用斜体字显示的。最后，可以看到最后的段落会与窗口右对齐，这是因为这个标记上的 class 属性被设为了 RightAligned。

如果没有看到最后一段贴合在文档窗口的右边，请确保在<style>标记中和在 class 属性中输入的 RightAligned 完全一样。因为 CSS 是区分大小写的，所以 RightAligned 和 rightaligned 之间是有很大的区别的。

(6) 若要在浏览器中显示该页面，则按下 Ctrl+F5 组合键。在浏览器中看到的页面与在 VWD 的 Design 视图中看到的预览完全相同。

工作原理

虽然在本练习中输入的代码相当简单，但是在后台浏览器(以及 VWD 2010 的 Design 视图)中却要做不少工作才能实现它。首先，要向页面的<head>部分添加一些样式。

```
<style type="text/css">
  h1
  {
    font-size: 20px;
    color: Green;
  }

  ...
</style>
```

<style>标记用来包装一个嵌在页面中的样式表，它的 type 属性设置为 text/css。在<style>标记之间从 h1 开始到闭合花括号的代码块称为规则集，或者仅称为规则。这段代码段中的规则定义了页面中所有<h1>元素的外观。代码块上方的 h1 称为选择器(selector)，用来表示应当向什么元素应用该格式化信息。在这种情况下，选择器直接映射到 HTML 元素上。还有很多其他选择器可以使用(在下一节将会介绍)。图 3-2 显示了元素相互之间的关系。

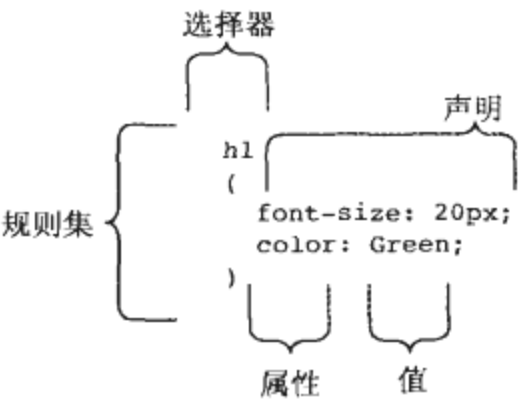


图 3-2

在花括号之间可以看到要应用到标题的样式信息。花括号中的每一行称为声明(declaration)。声明依次由一个属性、一个冒号以及一个值组成。声明末尾的分号(;)把它与下一个声明分隔开，除了规则集中的最后一个声明以外，其他所有的声明都需要使用分号隔开。然而，为了一致性，最好把分号添加到所有声明上，在本书的其余部分就是这么做的。

当浏览器加载这个页面时，它也是以在<style>标记之间定义的样式显示。然后，每当它遇到一个匹配该选择器的 HTML 元素时，就会向那个元素应用 CSS 规则。因此，对于<h1>和<p>元素，就会应用它们各自的规则。这样就导致标题的字体颜色变成绿色，字体更大；而段落会变成蓝色，字体为斜体。

但是最后一段为什么变成了蓝色并且右对齐呢？在 CSS 中，可以有来自不同源代码的规则。最后一个<p>标记从样式定义的 p 选择器中获得它的样式信息。因此，p 规则规定了段落使用蓝色且倾斜的字体。然而，它还定义了一个类。这个类为 RightAligned，使文本与窗口右边对齐。在末尾，最后一个<p>元素同时从两个选择器中得到它的规则。可以定义自己的类并为其指定名称(如 RightAligned 类的使用)，从而以更加灵活的方式设计页面和元素。

下一节将进一步地研究 CSS 的语法，给出关于选择器、属性和值的详细介绍。

3.2.1 CSS 语言

正如上一个“试一试”练习中所表明的，层叠样式表实际上是规则的集合。规则则是选择器和一个或多个声明的组合，而声明又可以分解为属性和值。您很可能对上面介绍的这些术语感到迷惑，因此在下一节中将再次介绍其中的大部分术语，并都提供详细的解释和代码示例，以显示它们的用途和工作方式。

3.2.2 样式表

样式表中包含应当应用到页面元素的所有相关样式信息。样式表最简单的形式如下所示：

```
h1
{
```



```
color: Green;
}
```

从前面的“试一试”练习中可以看出，样式表也可以包含多个规则。同时，每个规则可以包含多个声明，允许将它们组合在一个选择器下，比如：

```
h1
{
    font-size: 20px;
    color: Green;
}
```

这段代码在功能上等同于：

```
h1
{
    font-size: 20px;
}
h1
{
    color: Green;
}
```

在精简形式中，两个声明组合在同一个选择器下，因此阅读、理解和维护要容易得多。我们建议尽可能采用这种语法。

要能够样式化页面上的元素，浏览器必须知道 3 件事情：

- 必须样式化页面上的什么元素？
- 必须样式化元素的什么部分？
- 希望选中元素的那部分看起来是什么样子？

这些问题的答案由选择器、属性和值给出。

1. 选择器

顾名思义，选择器用来在页面内选择或指向一个或多个特定元素。有若干个不同的选择器可用，通过它们可以对要样式化的元素进行很细化的控制。选择器回答了第一个问题：必须样式化页面的什么元素？下一节将介绍 4 种最重要的选择器类型。

Universal 选择器

Universal 选择器由星号(*)表示，它适用于页面中的所有元素。Universal 选择器可以用来进行一些全局设置，比如字体。下面的规则集将页面中的所有元素的字体改为 Arial：

```
*
{
    font-family: Arial;
}
```

Type 选择器

Type 选择器允许指向一个特定类型的 HTML 元素。有了 Type 选择器，那种类型的所有 HTML 元素都会被相应地样式化。

```
h1
{
    color: Green;
}
```

这个 Type 选择器现在应用到代码中的所有<h1>标记上，并把它们设为绿色。Type 选择器是不区分大小写的，因此既可以用 h1 也可以用 H1 来表示相同的标题。

ID 选择器

ID 选择器总是以井号(#)为前缀，允许查阅页面中的单个元素。在 HTML 或 ASPX 页面内，可以使用 id 属性给每个元素赋予一个独一无二的 ID。使用 ID 选择器，就能改变那个元素的行为，如：

```
#IntroText
{
    font-style: italic;
}
```

由于可以在站点的多个页面上重用此 ID(只须在一个页面内独一无二)，因此可以用这个规则来快速地修改在每个页面上使用一次(但是在站点中使用多于一次)的元素的外观。如下面的 HTML 代码：

```
<p id="IntroText">I am italic because I have the right ID.</p>
<p id="BodyText">I am NOT italic because I have a different ID.</p>
```

在这个示例中，#IntroText 选择器将修改第一段的字体——它有匹配的 id 属性，但是让其他段落保持不变。ID 选择器是区分大小写的，因此，要保证 id 属性和选择器总是使用相同的大小写。

Class 选择器

Class 选择器可以用来通过 class 属性样式化多个 HTML 元素。当需要赋予若干个不相关的 HTML 元素相同的格式化类型时，使用这个选择器就非常方便。下面的规则将所有 HTML 元素的 class 属性设置为 Highlight，从而将它们的文本改为红色粗体字：

```
.Highlight
{
    font-weight: bold;
    color: Red;
}
```

下面的代码段使用 Highlight 类创建元素的内容，并让链接<a>显示为粗体字：

```
This is normal text but <span class="Highlight">this is Red and Bold.</span>
This is also normal text but
    <a href="CssDemo.aspx" class="Highlight">this link is Red and Bold as well.</a>
```

注意这个选择器在它的名称中用了句点，但是当涉及 class 属性中的选择器时，不要使用这个句点。class 属性是非常有用的，因为它允许对多种不同用途重用一块 CSS，而不必考虑使用该类的 HTML 元素是什么。

CSS 支持很多选择器类型，以便对目标元素进行更多的控制，不过最常用的就是刚才介绍的那

4 种类型。

组合和合并选择器

CSS 还允许组合多个选择器，用逗号将它们分开。如果要向不同的元素应用相同的样式，这是很方便的。下面的规则将页面中的所有标题都改为红色：

```
h1, h2, h3, h4, h5, h6
{
    color: Red;
}
```

而且，用 CSS 也可以合并选择器，这样就能有层次地指向一个页面中的特定元素。可以通过用一个空格分开选择器来做到这一点。下面的示例把落在 id 为 MainContent 的元素内的所有<p>元素作为目标，这个规则仅影响落在 MainContent 元素内的段落，而保持所有其他段落不变。

```
#MainContent p
{
    font-size: 18px;
}
```

注意，合并与组合有很大的区别。组合只是一个避免反复输入相同声明的快捷键，而合并允许将文档中的特定元素作为目标。

使用合并，不会限制只能使用 ID 和 Type 选择器；也可以通过其他选择器使用，正如下面的示例所演示的：

```
#MainContent p.Attention
{
    font-weight: bold;
}
```

这个规则会修改一个 id 为 Maincontent 的元素内类为 Attention 的所有段落，而不影响其他段落。下面的 HTML 代码段使用此规则显示效果：

```
<div id="MainContent">
  <p class="Attention">My class is Attention, so my text is bold.</p>
  <p>My text is not bold, as it lacks the Attention class.</p>
</div>
<p class="Attention">I am NOT bold because I don't fall within MainContent.</p>
```

在页面中应用某种样式时需要回答的第二个问题是必须样式化元素的哪一部分。它通过属性来回答。

2. 属性

属性是元素的一部分，可通过样式表修改。CSS 规范定义了一个长属性列表(VWD 的 IntelliSense 列表显示了 100 多项)，但在大多数 Web 站点中不会用到所有项。表 3-1 列出了部分最常见的 CSS 属性，并说明了它们的应用场合。

表 3-1

| 属 性 | 说 明 | 示 例 |
|---|---|--|
| background-color
background-image | 指定元素的背景色或图像 | background-color:White;
background-image:
url(Image.jpg); |
| border | 指定元素的边框 | border: 3px solid black; |
| color | 修改字体颜色 | color: Green; |
| display | 修改元素的显示方式，允许隐藏或显示它们 | display: none;
这种设置使元素被隐藏，不占用任何屏幕空间 |
| float | 允许用左浮动或右浮动将元素浮动在页面上。
其他的内容则被放在相对的位置上 | float: left;
该设定使跟着一个浮动的其他内容被放在元素的右上角。在本章后面将介绍它的工作原理 |
| font-family
font-size
font-style
font-weight | 修改页面上使用的字体外观 | font-family: Arial;
font-size: 18px;
font-style: italic;
font-weight: bold; |
| height
width | 设置页面中元素的高度或宽度 | height: 100px;
width: 200px; |
| margin
padding | 设置元素内部(内边距)或外部(页边距)的可用空间 | padding: 0;
margin: 20px; |
| visibility | 控制页面中的元素是否可见。不可见的元素仍然会占用屏幕空间，只是看不到它们而已 | visibility: hidden;
这会使元素不可见。然而，它仍然会占用页面的原始空间，就好像元素仍然在那里，只不过是完全透明的 |

幸运的是，VWD 会通过它的许多 CSS 工具帮助找到恰当的属性，因此不必全部记住它们。



注意：CSS 中可用的属性要比这里描述的多得多。要查看更多关于 CSS 的详细信息，请查看 VWD 附带的 IntelliSense 列表或者访问 www.w3schools.com/css/css_reference.asp 链接。

要使一个属性有用，需要给它赋值，这就回答了第三个问题：您希望选中元素的部分看起来是什么样的？

3. 值

与属性一样，值也有很多风格。可用的值取决于具体的属性。例如，color 属性采用表示颜色的

值。它可能是颜色名称(如 White)，也可能是代表红、绿、蓝(RGB)成分的十六进制数(如#FF0000)，还可以用 CSS rgb 表示法来设置。下面的示例在功能上是完全等价的：

```
h1
{
  color: Red;
}

h1
{
  color: #FF0000;
}

h1
{
  color: rgb(100%, 0%, 0%);
}
```

第一个声明使用了颜色名称 Red，而另外两个示例使用了一个 RGB 值来指定红色。使用颜色名称能增强 CSS 代码的可读性，但是由于可以指定名称的颜色相当少，所以常常需要用十六进制表示法来得到想要的精确颜色。在 VWD 中输入一个颜色属性时，会显示已知颜色的一个列表，如图 3-3 所示。如果想要输入不同的颜色，只需忽略该列表(或者按下 Escape 键关闭它)，然后输入自己的值。在本章后面的“试一试”练习中，将看到如何使用颜色选取器来选择非标准颜色。



图 3-3

还有很多其他值，包括大小单位(px、cm 等)、字体、图像(采用 url(SomeImage.jpg)的形式)，或者所谓的枚举，如边框样式，允许将边框样式设置为 solid、dashed、double 等。

4. 使用缩略版本

CSS 的很多属性允许写一个缩略版本以及一个较扩展的版本。例如，border 属性。在最短形式中，可以将 border 属性设置为这样：

```
border: 1px solid Black;
```

这个 border 属性应用到 HTML 元素的 4 条边上。边框大小将为 1px，样式将为 solid(一些其他选项包括 dashed、dotted 和 double)，边框颜色将被设置为 Black。

这是一种容易的方式，可以快速地将 HTML 的 4 条边框设置为相同的值。然而，如果希望对各条边框和它们的属性有更多的控制，可以使用扩展版本，如下所示：

```
border-top-width: 1px;
border-top-style: solid;
border-top-color: Black;
border-right-width: 1px;
border-right-style: solid;
border-right-color: Black;
border-bottom-width: 1px;
border-bottom-style: solid;
border-bottom-color: Black;
border-left-width: 1px;
border-left-style: solid;
border-left-color: Black;
```

这个长版本将使各边应用完全相同的样式：4 条边都是 1 像素宽的黑色实线边框。在大多数情况下，人们可能更喜欢缩略版本而不是扩展版本，因为它更容易阅读和维护。然而，如果需要对边框有绝对的控制，例如，如果希望 HTML 元素的左边和上边是 2 像素的虚线边框，右边和下边是绿色实线边框，那么最好单独设置所有 4 个方向的各个 border 属性。

支持缩略版本的其他 CSS 属性包括 font、background、list-style、margin 和 padding。如果不确定某个属性是否支持缩略版本，就使用 IntelliSense 弹出列表。当在 CSS 文件或<style>块中输入一个属性时，按下 Ctrl+空格键，就会弹出这个列表。

尽管有时候看起来需要通过测试和检错反复地编写 CSS，而这只是为了得到正确的结果，事实上在 CSS 的后台有一个十分精确的模型，它决定了页面上元素的布局方式。这个模型就是 CSS Box Model(CSS 方框模型)。

5. CSS 方框模型

CSS 方框模型描述了将 3 个重要的 CSS 属性应用于 HTML 元素的方式，这 3 个属性是 padding、border 和 margin。图 3-4 是这个方框模型的图形表示：

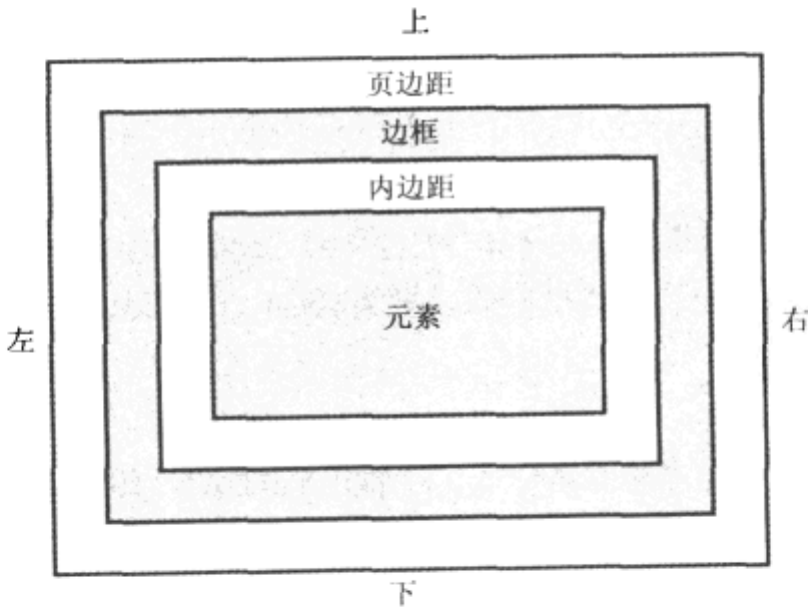


图 3-4

图 3-4 的中间部分是一个 HTML 元素，如一个具有特定高度和宽度的<p>或<div>元素。与它最

近的一层是内边距(padding)，即边框内部包围 HTML 元素的白色部分。padding 的外层是边框(border)。最后，该图的最外层是页边距(margin)，它定义了某个元素(包括它的内边距和边框)与其周围元素之间的空间。元素外层的 3 个属性(padding、border 以及 margin)所占的空间总计等于元素在页面中所占的空间。要想知道这是如何工作的，请看下面的 CSS 和 HTML 代码：

```
.MyDiv
{
    width: 200px;
    padding: 10px;
    border: 2px solid black;
}
...
<div class="MyDiv">Element</div>
```

运行上面的代码会在浏览器中显示出一个具有 2 像素黑色边框的矩形框，其中有一个<div>元素，如图 3-5 所示：

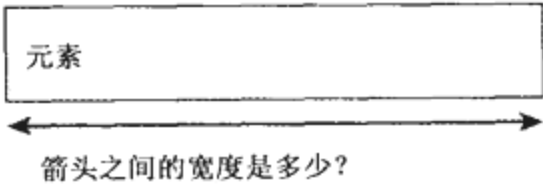


图 3-5

现在，先试着回答这个问题：<div>元素下面的箭头之间的宽度是多少？

如果你猜的是 224 像素，就猜对了！这个箭头的宽度是 3 个值的和，即实际元素的宽度(200 像素)、两侧包围元素的内边距的宽度(两个 10 像素)和两侧边框的宽度(两个 2 像素)；这 3 个值加起来就是 224 像素。因此，如果希望整个方框的宽度是 200 像素，只需将 MyDiv 选择器的 width 属性设为 176 像素即可。

这个例子只演示了宽度属性的效果，但是这些规则同样适用于元素的高度属性。在布局页面时要记住这个方框模型。如果得到的元素宽于或高于期望的值，就要核对 CSS 样式表中 width、height、padding、border 以及 margin 属性的值。

在下面的“试一试”练习中，将修改在第 2 章中构建的站点的主页。在此练习中将为站点添加基本布局，然后用样式表样式化页面。在第 6 章把它升级为母版页时，将再次用到这个页面。

试一试 样式化 Planet Wrox 主页

在这个练习中将修改两个文件：首先，向 Default.aspx 页面中添加基本布局元素，来为标题、菜单、主内容区域、滚动条及页脚创建空间。然后编辑 Styles 文件夹中的 Styles.css 文件，修改这些元素的大小和位置。最后，把该样式表附加到页面上，这样，在设计器或浏览器中浏览页面时就会应用这些样式信息。

- (1) 打开 Web 站点根文件夹中的文件 Default.aspx，如有必要，切换到 Markup 视图。
- (2) 修改<form>元素中的代码，结果如下所示：

```
<form id="form1" runat="server">
  <div id="PageWrapper">
    <div id="Header">Header Goes Here</div>
    <div id="MenuWrapper">Menu Goes Here</div>
    <div id="MainContent">
      <h1>Hi there visitor and welcome to Planet Wrox</h1>
      ...
    </div>
    <div id="Sidebar">Sidebar Goes Here</div>
    <div id="Footer">Footer Goes Here</div>
  </div>
</form>
```

一定要保证在第 2 章中添加的欢迎消息在 MainContent <div>元素的起始和结束标记中出现。

(3) 从 Styles 文件夹中打开文件 Styles.css。如果之前在这个文件中添加了一些代码，则要先把这些代码删除掉。

(4) 在页面的上方，输入下面的代码，使用 ID 选择器选择 Header <div>:

```
#Header
{
}
```

(5) 把鼠标放在花括号之间，然后从主菜单中选择 Styles | Build Style 命令。此外，也可以通过右击 ID 选择器，或者单击 Styles 工具栏上的 Build Style 按钮来选择同样的选项。这时将出现如图 3-6 所示的 Modify Style 对话框。

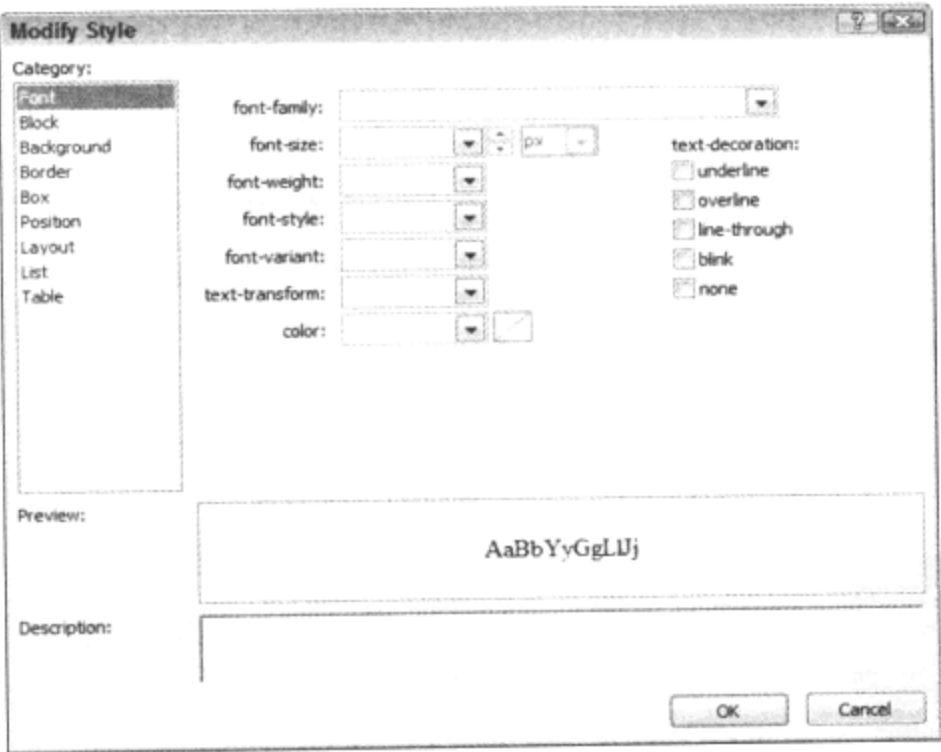


图 3-6

(6) 在左边的 Category 列表中，单击 Background 选项，然后打开背景色的下拉列表。从出现的颜色选取器中，单击 Silver 颜色，如图 3-7 所示。

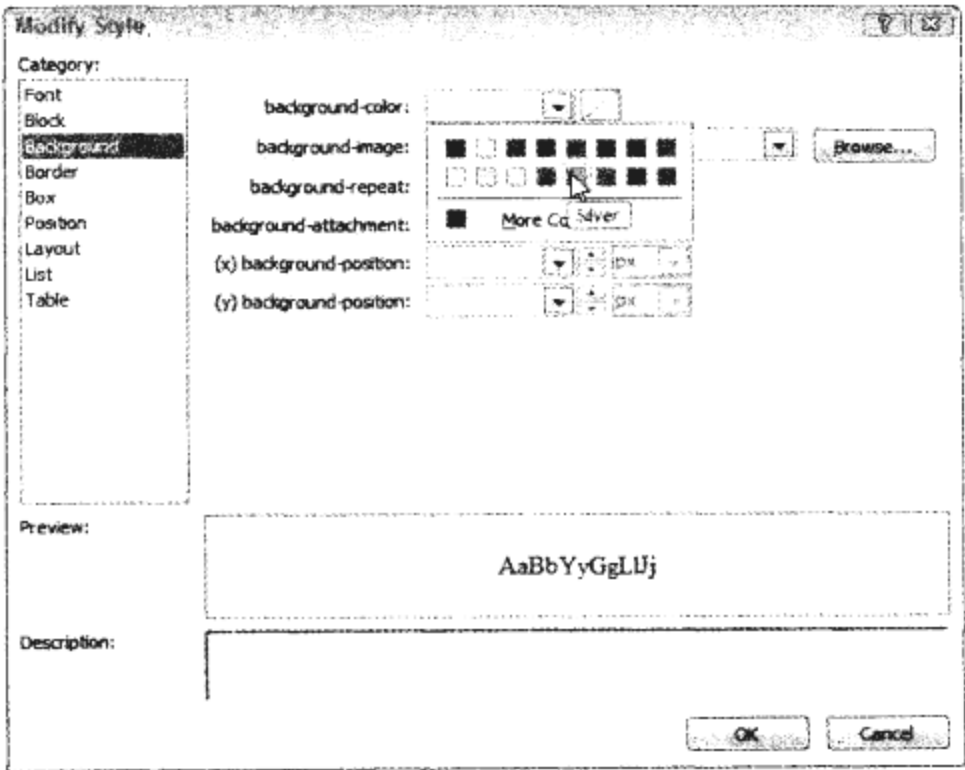


图 3-7

此外，还可以直接在背景色文本框中输入颜色的十六进制代码，如 Silver(#C0C0C0)。

(7) 通过单击左边的列表切换到 Position 类别。出现的面板用来设置与位置相关的信息，包括高度与宽度。在宽度下面，输入 844，一定要选中右边下拉列表中的 px。高度输入 86。单击 OK 按钮关闭对话框，并向代码中插入声明，它现在应该如下所示：

```
#Header
{
    background-color: #C0C0C0;
    width: 844px;
    height: 86px;
}
```

(8) 重复第(4)~(7)步，创建下面的规则：

```
*
{
    font-family: Arial;
}

h1
{
    font-size: 20px;
}

#PageWrapper
{
    width: 844px;
}

#MenuWrapper
{
```

```
        width: 844px;
    }

    #MainContent
    {
        width: 664px;
        float: left;
    }

    #Sidebar
    {
        background-color: Gray;
        width: 180px;
        float: left;
    }

    #Footer
    {
        background-color: #C0C0C0;
        width: 844px;
        clear: both;
    }
```

在 Modify Style 对话框的 Layout 类别中将出现 float 和 clear 属性。

(9) 完成规则的创建后，保存并关闭文件 Styles.css，因为现在对它的操作已经完成。

(10) 再次打开文件 Default.aspx，并切换到 Design 视图。从 Solution Explorer 中，把 Styles 文件夹中的文件 Styles.css 拖到页面上，就会发现 Design 视图发生了改变以反映在样式表中编写的代码。当把样式表放到页面上时，VWD 会在 Markup 视图中页面的<head>部分插入向文档附加样式表的代码。

```
<head runat="server">
    <title></title>
    <style type="text/css">
        .style1
        {
            color: #FF0000;
        }
    </style>
    <link href="Styles/Styles.css" rel="stylesheet" type="text/css" />
</head>
```

也可以在 Markup 视图中页面的<head>部分直接从 Solution Explorer 中拖动一个现有样式表。这时，VWD 会添加相同的<link>元素。

(11) 最后，保存对所有打开的文档的修改(按下 Ctrl+Shift+S 组合键)，然后在浏览器中请求 Default.aspx 页面。这时屏幕应类似于图 3-8 所示，它显示了 Mozilla Firefox 中的页面。

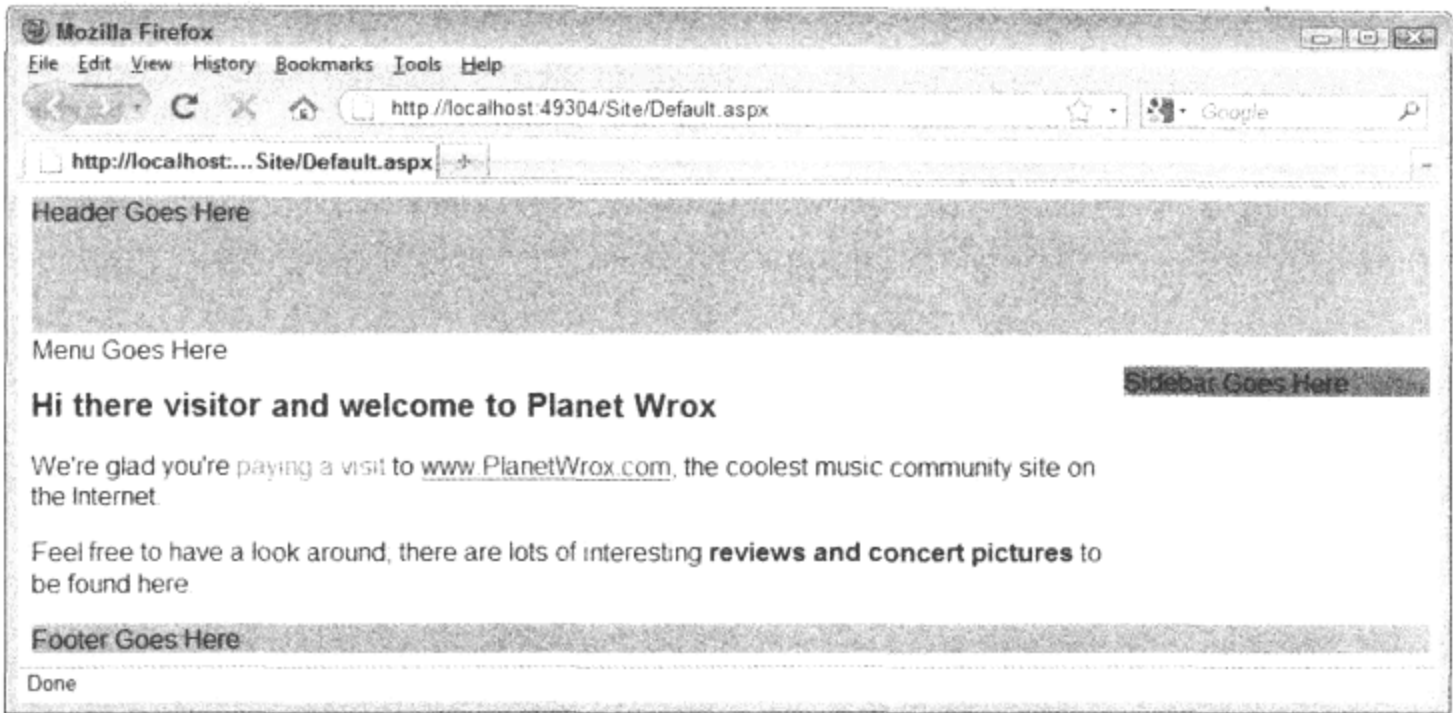


图 3-8

工作原理

Style Builder 使开发人员很容易选择 CSS 属性和修改它们的值。不需要记住关于 CSS 的所有细枝末节，相反，还可以可视化地创建 CSS 代码。虽然这个工具可以做大部分工作，但是如果能够阅读并理解 CSS 代码，仍然是很有用的。有时，当需要调整代码时，直接在 Document Window 中调整通常比打开 Style Builder 调整要快。

注意，Header、PageWrapper、MenuWrapper 和 Footer 的精确宽度是 844 像素。这样，站点就可以很好地适合于 1024×768 像素的屏幕宽度(这种屏幕大小如今很普遍)，因而不用在窗口的边界中进行缩放。有时，用较大屏幕的系统则会增加页面右边的空白。

还要注意，MainContent 区域和 Sidebar 彼此相邻。这是用 CSS 的 float 属性实现的：

```
#MainContent
{
    width: 664px;
    float: left;
}

#Sidebar
{
    background-color: Gray;
    width: 180px;
    float: left;
}
```

这段代码可以让 MainContent “浮动”在 Sidebar 的左侧，有效地将 Sidebar 放在它的左边。还需要让 Sidebar “浮动”。如果不进行这样的设置，它将被放置在页面的左边，而这里正好没有为它应用 CSS 样式。

这两个元素的宽度合起来共 844 像素，恰好等于它们的父元素 PageWrapper 的宽度。
要结束浮动并让 Footer 元素直接放在 MainContent 和 Sidebar 元素下面，可使用 clear 属性清除

可能有效的任何浮动(左或右):

```
#Footer
{
    background-color: #C0C0C0;
    width: 844px;
    clear: both;
}
```

灰色背景只是临时添加到代码中的, 因此很容易看出<div>最终位于何处。在将来的练习中, 会再次修改这个 CSS 文件来适应 Planet Wrox Web 站点的模式。

要告诉浏览器应用什么样式, 需在页面的标题中链接样式表:

```
<link href="Styles/Styles.css" rel="stylesheet" type="text/css" />
```

这行代码会通知浏览器到文件夹 Styles 中查找一个名为 Styles.css 的文件, 并将该文件中的所有规则应用到当前文档中。一旦浏览器下载了这个 CSS 文件, 它就会将在那里找到的所有样式应用到 HTML 元素上, 最终产生如图 3-8 所示的布局。

在这个练习中, 可以看出如何用<link>标记向一个页面链接样式表。然而, 在 Web 页面中包括样式表的方法有好多种。

3.2.3 向页面中添加 CSS

向 Web 页面中添加 CSS 样式表的首选方法是通过<link>标记, 就像前面的练习中所指明的, 它指向一个外部 CSS 文件。我们来看下面这个<link>, 通过它可以了解向页面中嵌入样式表时有哪些选择:

```
<link href="StyleSheet.css" rel="Stylesheet" type="text/css" media="screen" />
```

href 属性指向站点内的一个文件, 就像第 2 章介绍的在两个页面之间创建链接时那样。rel 和 type 属性告诉浏览器链接的文件实际上是一个层叠样式表。media 属性很有趣: 它允许以不同的设备为目标, 包括显示器、打印机、手持设备, 甚至包括为视觉有障碍的访问者准备的盲文点字设备和助听器。media 的默认属性是 screen, 因此如果目标是标准桌面浏览器, 就完全可以省略这个属性。

在本章开头曾经简要介绍过包括样式表的另一种方法: 使用嵌入的<style>元素。<style>元素应放在 ASPX 或 HTML 页面上方的<head>标记之间。在<style>标记内, 可以编写与以前看到的完全相同的 CSS 代码。例如, 要单独修改当前页面中<h1>元素的外观, 可以向页面的<head>标记内添加下列代码:

```
<head runat="server">
    <title></title>
    <style type="text/css">
        h1
        {
            color: Blue;
        }
    </style>
</head>
```

向 HTML 元素应用 CSS 的最后一种方法是使用带有第 2 章指出的样式属性的内联样式。由于

style 属性已经应用到了特定的 HTML 元素，因此不需要选择器，而可以直接把声明写在这个属性中：

```
<span style="color: White; background-color: Black;">
  This is white text on a black background.
</span>
```

在外部、内嵌和内联样式表之间进行选择

既然有这么多选择可以向站点中添加样式表，那么使用哪种方法最好？一般而言，外部样式表优于内嵌样式表，内嵌样式表优于内联样式表。外部样式表允许通过单个文件改变整个站点的外观。只要对外部样式表文件做一次修改，使用这个样式表的所有页面就会自动进行这样的修改。

然而，在某些情况下使用内嵌样式表和内联样式表也是完全可以接受的。如果要修改单个页面的外观而不想影响到站点中的其他页面，那么内嵌样式表就是最佳选择。内联样式表也是如此，如果只想修改单个页面中单个元素的行为且相当确定其他 HTML 元素不需要同样的声明时，就使用内联样式。

需要考虑的一件重要的事情是不同类型的样式表互相覆盖的方式。如果有多个带不同属性值的同等选择器，那么会优先采用最后定义的那个选择器。例如，有一个在名为 Styles.css 的外部样式表中定义的规则，它将所有<h1>标记的颜色都设置为绿色：

```
h1
{
  color: Green;
}
```

现在假设要将这个样式表应用到一个页面中，该页面中对同样的 h1 还有一个嵌入规则，但设置了不同的颜色：

```
<link href="Styles/Styles.css" rel="stylesheet" type="text/css" />
<style type="text/css">
  h1
  {
    color: Blue;
  }
</style>
```

运行这段代码，页面中的实际<h1>标记的颜色就会是蓝色。这是因为将颜色定义为蓝色的内嵌样式表在页面中定义得较晚，因此它覆盖掉了外部文件中定义的设置。如果采用这样的样式：

```
<style type="text/css">
  h1
  {
    color: Blue;
  }
</style>
<link href="Styles/Styles.css" rel="stylesheet" type="text/css" />
```

则标题将会是绿色的，因为此时外部样式表中的设置否决了内嵌样式表中的值。

内联样式表也是采用这样的机制。由于它们是直接在 HTML 元素上定义的，因此与内嵌和外部样式表相比，会优先采用它们的设置。

也要注意 CSS 通常否决 HTML 元素上的属性。例如，如果有一个 CSS 规则设定了某个图像的宽度和高度，站点就会忽略 img 元素上的 height 和 width 属性，在这个例子中最终图像会显示为 100 像素的方形：

```
img
{
    height: 100px;
    width: 100px;
}
...

```



注意：关于 CSS 的知识远不止这里提到的这些。要了解关于 CSS 的更多信息，请参见清华大学出版社引进并出版的《ASP.NET Web 界面设计三剑客:CSS、Themes 和 Master Pages》(ISBN: 978-7-302-19992-2) 或者《CSS 入门经典(第 2 版)》(ISBN: 978-7-302-17954-2)。

一般而言，建议在<head>部分的上方应用外部文件，后面跟上内嵌样式表。用这种方式，外部文件可以定义元素的全局外观，然后可以用内嵌样式来否决外部设置。

用 VWD 可以轻松地将内嵌样式表移到外部 CSS 文件中。下面一节将讨论 VWD 中其余的 CSS 工具，在其中会介绍具体如何操作。

3.3 在 VWD 中使用 CSS

VWD 中有几个使用 CSS 的便利工具，如下所示：

- **Style Sheet toolbar** 用来快速访问并创建新规则与样式。
- **CSS Properties 面板** 用来修改属性值。
- **Manage Styles 窗口** 用来组织站点的样式，将它们从内嵌样式表改为外部样式表，反之亦然；对它们重新排序；将现有样式表链接到一个文档；创建新的内联、内嵌或外部样式表。
- **Apply Styles 窗口** 用来从站点中选择所有可用样式，并将它们快速地应用到页面中的不同元素上。
- **Style Builder** 用来可视化地创建声明。
- **Add Style Rule 窗口** 帮助构建较复杂的选择器。

下面几节会详细解释这 6 个工具。

3.3.1 在外部样式表中创建新样式

在前面的“试一试”练习中，向 CSS 文件中手动添加了选择器，然后用 Style Builder 写了规则。然而，也可以使用 VWD 工具来写选择器。在下面的“试一试”练习中，将介绍如何使用 Add Style Rule 窗口在外部文件中创建一个新规则，然后使用 Style Builder 来修改这个规则。

试一试 在现有样式表中创建新样式

在本练习中，将创建一个影响 MainContent 区域中所有链接的新样式。通过使用合并选择器，可以仅将这个内容区域中的链接作为目标，而保持其他链接不变。

- (1) 首先从 Styles 文件夹中打开文件 Styles.css。
 - (2) 向下滚动文件，把光标放在末尾刚好位于#Footer 规则下面的位置。
 - (3) 确保 Style Sheet 工具栏可见，然后单击第一个按钮 Add Style Rule，或者从主菜单中选择 Styles | Add Style Rule 命令。有了这个对话框，就能可视化地创建一个合并选择器。在该对话框的左边，可以在 Element、Class 和 ID 选择器三者中进行选择。
- 选择最后一个选项 Element ID，然后在它的文本框中输入 MainContent。对话框如图 3-9 所示。

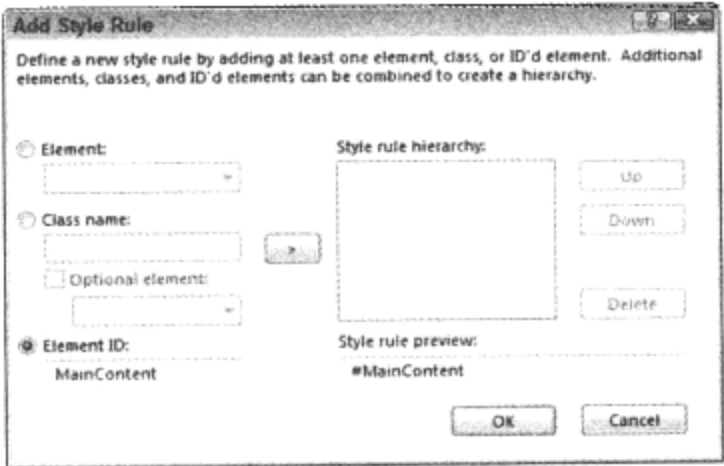


图 3-9

如果在 Style rule preview 区域中 MainContent 的前面没有看到#字符，请确保选择的是 Element ID 而非 Element。单击屏幕中间带右箭头的按钮，把选择器添加到 Style rule hierarchy 列表中。

- (4) 接着，选择对话框上方的 Element 单选按钮，然后从它的下拉列表中选择 a(为了链接)并再次单击箭头按钮。屏幕现在应在 Style rule preview 区域显示选择器的一个预览，如图 3-10 所示。

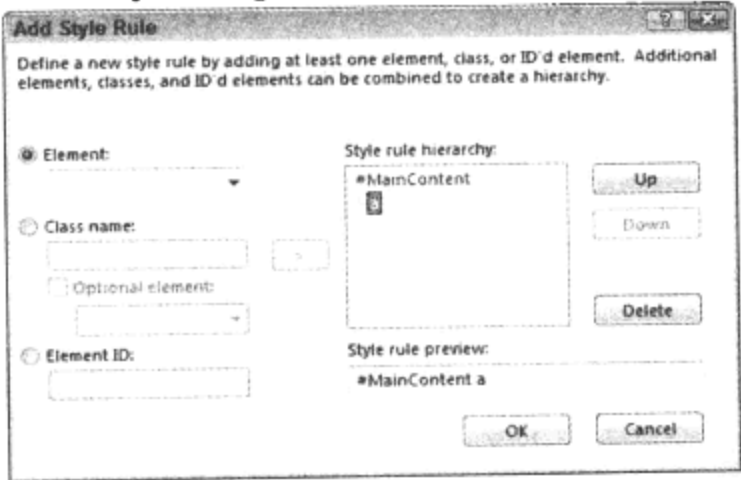


图 3-10

- (5) 单击 OK 按钮向样式表文件中添加选择器。最后应当看到一个如下的空规则：

```
#MainContent a
{
}
```

- (6) 在刚刚插入的规则的花括号之间右击，并选择 Build Style 命令。
- (7) 在 Font 类别中，单击下拉列表框的箭头按钮将颜色属性改为#008000，然后单击最上面一行

的绿色方块。

(8) 在对话框右边的 text-decoration 部分，选中 underline 选项前的复选框。Modify Style 对话框现在应当如图 3-11 所示。

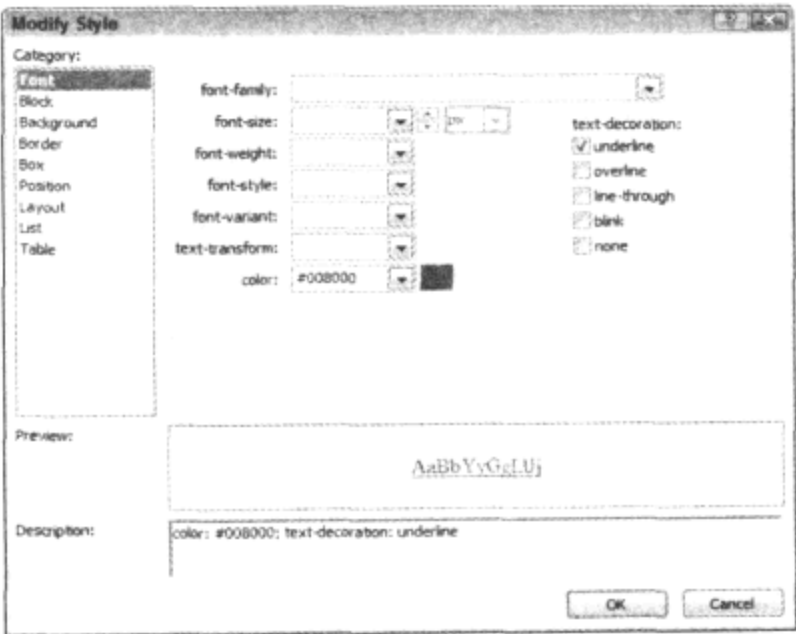


图 3-11

(9) 单击 OK 按钮关闭对话框。回到 CSS 文件中，选择刚刚创建整个规则集(包括#MainContent a 和两个花括号)，将其复制到剪贴板中，然后再往原始规则集下方粘贴两次。

(10) 将刚刚从#MainContent a 中粘贴来的第一个选择器重命名为#MainContent a:visited。这个样式用于用户已经访问过的链接。

(11) 右击刚刚创建的新规则，并选择 Build Style 命令。在 Font 类别中，在颜色文本框中输入 #FF0000 将颜色从绿色改为红色，然后单击 OK 按钮。

(12) 将文件中的第三个选择器从#MainContent a 改为#MainContent a:hover。当用户将鼠标悬停在它们上面时会应用这个样式。

(13) 再次右击刚刚创建的新选择器，并选择 Build Style 命令。在 Font 类别中，通过在颜色文本框中输入#FFA500，将颜色从绿色改为橙色。单击 OK 按钮关闭 Modify Style 对话框。

最后应在 CSS 文件中已存在的样式下面看到以下 3 个规则：

```
#MainContent a
{
    color: #008000;
    text-decoration: underline;
}

#MainContent a:visited
{
    color: #FF0000;
    text-decoration: underline;
}

#MainContent a:hover
{
    color: #FFA500;
    text-decoration: underline;
}
```

(14) 保存并关闭 Styles.css 文件。

工作原理

在这个“试一试”练习中，首先用 Add Style Rule 对话框创建了一个新规则。直接在代码编辑器中输入规则很快捷。然而，在创建复杂的合并规则时，Add Style Rule 对话框可以帮助理解和创建规则的层次结构。

Modify Style 对话框是创建新 CSS 声明的极佳工具。不需要记住各种 CSS 属性和它们的值，只要简单地在一个组织好的对话框中指定并一起单击即可。各种不同的 CSS 属性都根据逻辑被分组到不同类别下，以便更容易找到和修改它们。

a 选择器上的: hover 和: visited 部分您可能不太熟悉。这些选择器被称为伪类选择器。a: visited 选择器仅用于已经在浏览器中访问过的链接，a: hover 选择器仅用于当用户将鼠标悬停在链接上时的 <a> 标记。在下一个“试一试”练习中将看到这两个选择器在浏览器中的效果。

创建了样式表后，需要做的下一件事情就是将样式表附加到文档上。有几种方法可以做到这一点：可以手动输入代码，也可以将文件拖放到 Markup 视图或 Design 视图中。下面的“试一试”练习演示了第 3 种选择：使用 Manage Styles 窗口。

试一试

将新样式表附加到文档上

在本练习中，将在 Default.aspx 页面中去掉和重新贴上名为 Styles.css 的样式表，这样就会看到向 ASPX 页面附加样式表文件的另一种方法。然后再向这个页面添加一些文本与链接，就可以看到在前面创建的规则集的行为。

(1) 将页面 Default.aspx 切换到 Markup 视图中，并从在本章前面添加的<head>部分中删除<link /> 元素，然后切换到 Design 视图并确保 Manage Styles 窗口是打开的。如果没有打开，则在文档窗口中的某处单击一下以激活 Design 视图，然后从主菜单中选择 View | Manage Styles 命令，这时就会出现如图 3-12 所示的窗口。如果在 View 菜单中没有看到此项，就请回顾第 1 章中的内容，先选择 Tools | Settings | Expert Settings 命令后再执行上述菜单命令。

Manage Styles 窗口提供了一个总体视图，可以看到应用到当前文档的所有外部和内嵌样式表。注意 VWD 是如何看待已经包含内嵌样式的当前文档的：它把它看作在第 2 章创建的 style1。

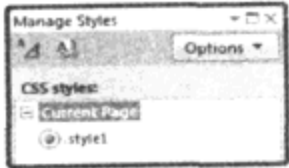


图 3-12

(2) 单击 Manage Styles 窗口中的 Attach Style Sheet 按钮(图 3-12 中的第二个按钮)，定位到根站点中的 Styles 文件夹，选择 Styles.css 文件。单击 OK 按钮，样式表就会再次插入到页面中。

(3) 当附加样式表时，Manage Styles 窗口(如图 3-13 所示)就会被更新。现在显示的是最新附加的样式表。

(4) 在 Design 视图中页面 Default.aspx 仍然打开的情况下，在段落中选择文本“look around”，如果在上一个“试一试”练习中输入的是别的内容，就选择那个文本。在这一阶段，最重要的是把一些文本转化为链接。

(5) 在 Formatting 工具栏上，单击 Convert to Hyperlink 按钮(其上有一个球状的链接标志)，在出现的对话框中单击 Browse 按钮，并在根站点中选择文件 Default.aspx。这样，链接就会指向在其中定义它的那个页面，对于本练习来说这样很好。单击 OK 按钮两次，关闭对话框。

(6) 保存对所有打开的文档所做的修改(从主菜单中选择 File | Save All 命令或按下 Ctrl+Shift+S 组合键)，然后按下 Ctrl+F5 组合键，在浏览器中请求 Default.aspx 页面。这时应看到出现一个页面，其上有一个带有下划线的“look around”链接，如图 3-14 所示。



图 3-13



图 3-14

(7) 把鼠标悬停在“look around”链接上；可以注意到它变成了橙色。

(8) 单击“look around”链接，就会重新加载页面。该链接现在变成红色的了。如果在第一次访问时链接已经是红色的，不要担心。这可能是因为在浏览器中打开过这个页面，导致浏览器将它标记为已访问过的链接。浏览器会记录访问过的页面，然后把正确的样式应用到新的和已访问过的链接。如果要在 Internet Explorer 中查看指定的行为，可以通过选择 Tools | Internet Options 命令来打开 Internet Options 对话框。然后在 General 选项卡上，单击 Delete 按钮。在弹出的对话框中，根据屏幕上的提示清除整个浏览历史，因此如果这时在浏览器中按下 Ctrl+F5 组合键重新加载页面，链接应变为绿色的。其他浏览器也有类似的清除浏览器浏览历史的选项。例如，在 Firefox 中可以用 Tools | Clear Recent History 菜单选项清除浏览历史。如果没有这个菜单项，则选择 Tools | Options 命令，然后切换到 Privacy 选项卡，在那里也可以清除浏览历史。

此外，还可以在不同的浏览器中打开页面。要选择另一个浏览器，在 Solution Explorer 中右击 Default.aspx，并从上下文菜单中选择 Browse With 命令。如果另一个浏览器已经列在那里，则从列表中选择它，然后单击 Browse 按钮。也可以通过单击 Set as Default 按钮将这个浏览器设为默认浏览器。

如果没有列出想要的浏览器，则单击 Add 按钮，然后单击 Program Name 框旁边的省略号来查找喜欢的浏览器。当浏览器显示在列表中时，单击以选择它，然后单击 Browse 按钮在那个浏览器中打开页面。这个页面现在应出现在另一个浏览器中。

工作原理

Manage Styles 窗口提供了当前页面中的活动样式表的快速概览，可能是一个外部或附加样式表，也可能是页面的<head>部分中的内嵌样式表。它是一个非常有用的窗口，可以将新样式附加到当前文档中，将一个位置的样式移到另一个位置，具体做法将在下一节介绍。当在浏览器中打开页面时，就已经下载了更新后的样式表，然后浏览器会将#MainContent a:visited 选择器应用到之前访问过的

页面的所有链接上。将鼠标悬停在一个链接上时，就应用了选择器#MainContent a:hover，使链接变成橙色。

在另一个浏览器中查看页面是个不错的方法。尽管现代浏览器对页面的显示越来越相似，但是仍有一些细微的差别需要注意，并在 HTML 和 CSS 代码中针对这种差别进行处理。在系统中安装几个不同的浏览器(例如 IE、Firefox、Safari、Opera 和 Chrome 等)，把按照这个“试一试”练习中的说明把它们分配到 Browse With 对话框中，并在这些浏览器中尽可能多地测试页面，这将有助于确保页面的显示在主流浏览器中完全一致。

尽管外部样式表可能很有用，但是仍然有一些时候会确实想使用内嵌或内联样式。创建并管理这些样式相当容易，下一节将会介绍。

3.3.2 创建内嵌和内联样式表

在操作 Design 视图中的页面时，常常需要对页面的部分内容作些细微调整，如样式化一块文本，对齐一个图像，或者向一个元素应用边框。在本阶段，需要决定是创建一个内联、内嵌还是外部样式表。正如前面所指出的，如果预测到以后会重用一個样式，则应选择外部或内嵌样式表。但在 VWD 中采用何种样式表并不是太紧要，它允许创建所有 3 个级别的样式表，而且它还允许轻松地将内嵌样式升级为外部样式，或者将内联样式信息复制到不同位置，因此提供了很大的灵活性，以后也可以随时改变主意。

在下一个“试一试”练习中，将介绍如何创建内联和内嵌样式表。以后将介绍如何将这些样式移动到外部样式表中，使其他页面可以重用相同的样式。

试一试 在一个页面中创建内嵌和内联样式

在本“试一试”练习中，将向页面的<h1>元素添加一个样式规则，以删除浏览器在这个标题周围保留的默认页边距。此外，还将用一个类来样式化第一段，给它设计一个与众不同的外观，使之突出于页面上其他段落。

- (1) 回到 VWD，并确保 Design 视图中的页面 Default.aspx 是打开的。
- (2) 在文档窗口中的 h1 元素上单击一次以选中它，然后选择 Format | New Style 命令。这时会出现 New Style 对话框(如图 3-15 所示)，它与以前出现过的 Modify Style 对话框十分相似。
- (3) 在屏幕的上方，打开 Selector 下拉列表并选择内联样式。它是列表中的第一项，这样就可以确保将新样式作为内联样式应用到<h1>元素上。

- (4) 切换到 Box 类别，如图 3-16 所示。

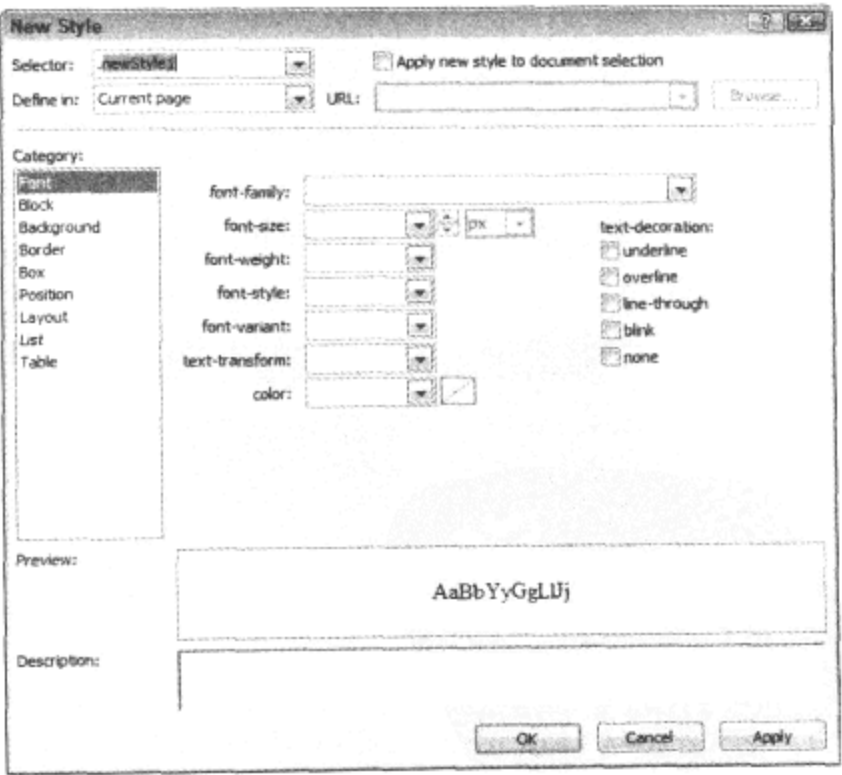


图 3-15

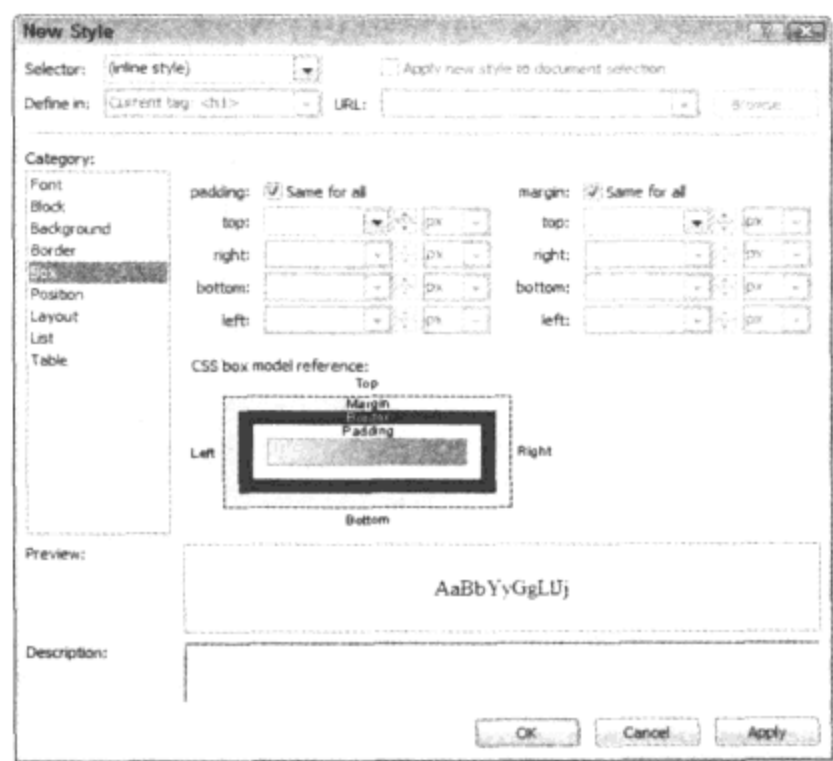


图 3-16

这个对话框中有一个方便的关系图，可作为 CSS 方框模型的补充，它显示了 CSS 属性最终出现的位置，如 Padding、Border 和 Margin。

默认情况下，浏览器会在<h1>元素上方或下方绘制一些白色空白，但空白的实际大小因不同的浏览器而异。为了给各个浏览器以相同的设置，可以把内边距重置为 0，然后在标题下方应用一点页边距，这样就可以在它和紧邻它的元素之间生成一些距离。为此，在 top 文本框中把内边距设置为 0。保持选中 Same for all 选项，VWD 会创建一个缩略声明。然后取消选中 margin 部分的 Same for all 选项，在 top、right 和 left 文本框中输入 0，在 bottom 文本框中输入 10。将所有下拉列表设置为 px 并单击 OK 按钮。最后会在 Markup 视图中得到下面的带内嵌样式的<h1>元素。

```
<h1 style="padding: 0px; margin: 0px 0px 10px 0px">
  Hi there visitor and welcome to Planet Wrox
</h1>
```

(5) 接下来，在 Design 视图中单击选中第一段。这时会出现一个小浮起，表示选中了一个<p>元素，如图 3-17 所示。文档窗口底部的标记选择器应突出显示<p>元素。

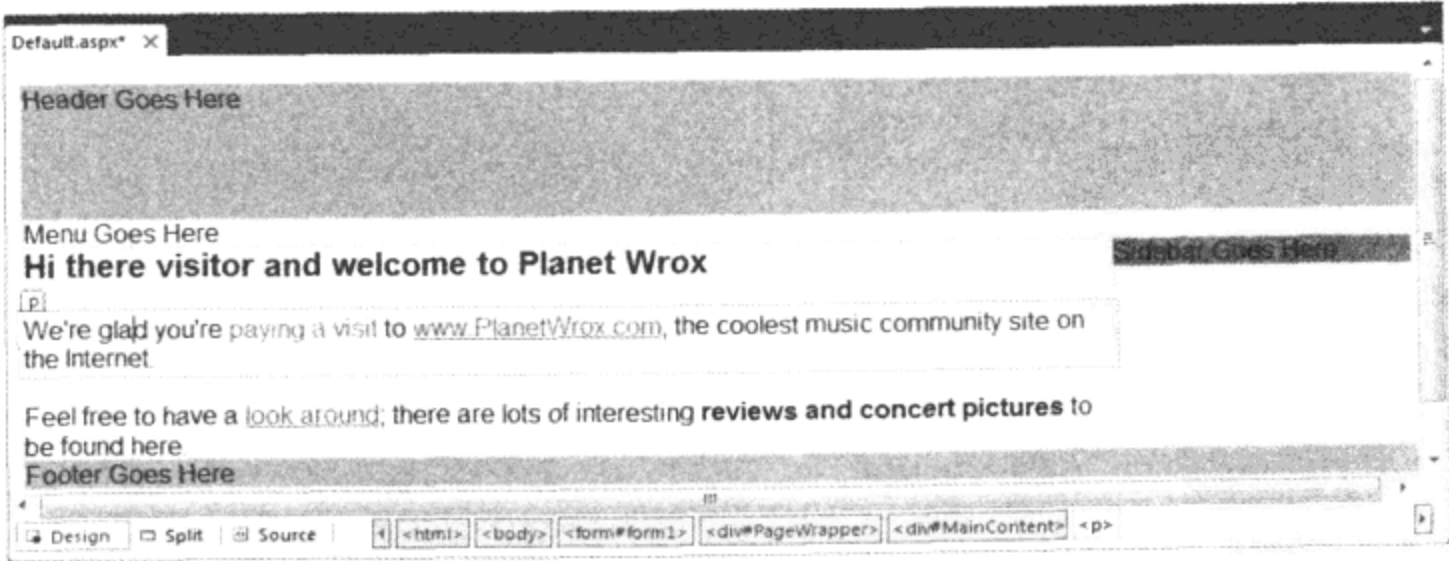


图 3-17

- (6) 在这一段仍然被选中的状态下，从主菜单中选择 **Format | New Style** 命令。这时，不是创建内联样式，而是在 **Selector** 框中输入文本 `.Introduction`，如图 3-18 所示。不要忘记输入选择器名称前面的句点(.)。
- (7) 在屏幕上方，选中 **Apply new style to document selection** 前面的复选框。在这个设定打开的情况下，要创建的新类就会被应用到已选中的 `<p>` 标记上。
- (8) 从 **font-style** 下拉列表中选择 **italic**。**New Style** 对话框现在看起来应当如图 3-18 所示。
- (9) 最后，单击 **OK** 按钮。注意整个段落现在已经显示为斜体。

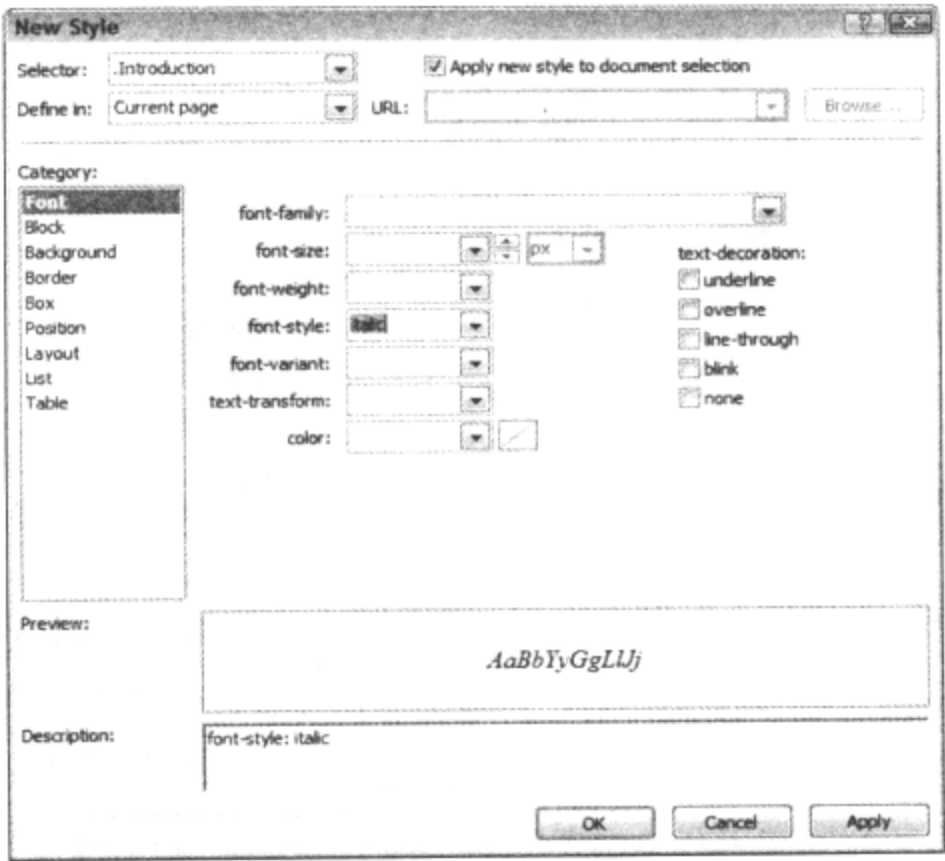


图 3-18

- (10) 在 `<p>` 标记仍然被选中的情况下，通过选择 **View | CSS Properties** 命令打开 **CSS Properties** 面板(如图 3-19 所示)。从这个面板中可以纵览所有的 CSS 属性，并显示页面中哪些属性当前是活动的。

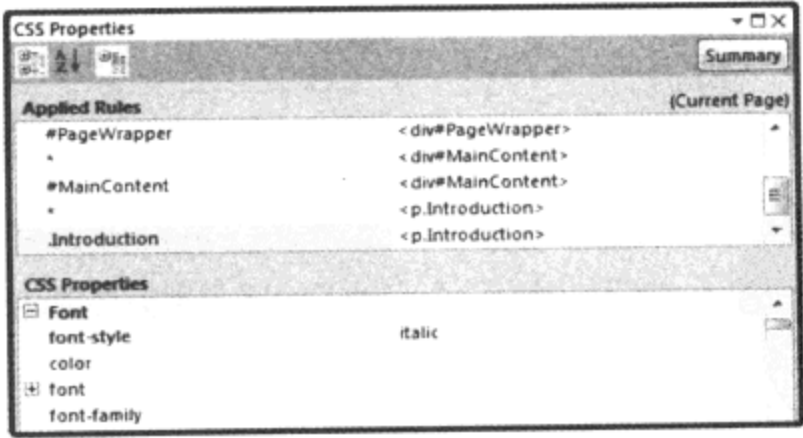


图 3-19

该 **CSS Properties** 面板的上半部分显示了应用的规则列表，下半部分用来显示这些规则的 CSS 属性。在图 3-19 中可以看到应用到 `.Introduction` 选择器的规则。显示为蓝色粗体字的属性有它们的

值集，而其他属性则显示为正常字体。如果没有看到这些样式，请单击 CSS Properties 面板的工具栏上的第三个按钮，这样可以移动这个列表中配备的属性。

(11) 在 CSS Properties 列表的下半部分，定位到 color 属性并把它设置为深蓝色，如#003399。要做到这一点，需打开属性值的下拉列表，并从颜色选取器中选择一种颜色。如果所寻找的颜色不可用，那么请单击 More Colors 按钮以扩展颜色选取器，如图 3-20 所示。

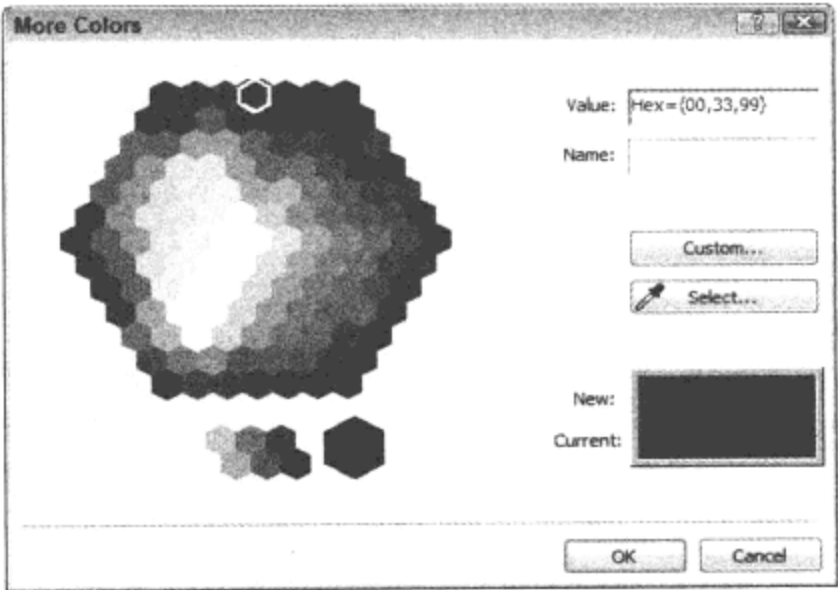


图 3-20

也可以不使用颜色选取器，直接在 Properties 面板中输入一个值。下面是 CSS Properties 面板中所有属性的工作方式：它们允许直接输入值，或者使用下拉列表或在属性的值框后面的省略号按钮来可视化地修改属性值。图 3-21 显示了在一个下拉列表中可用的各种字体样式属性。

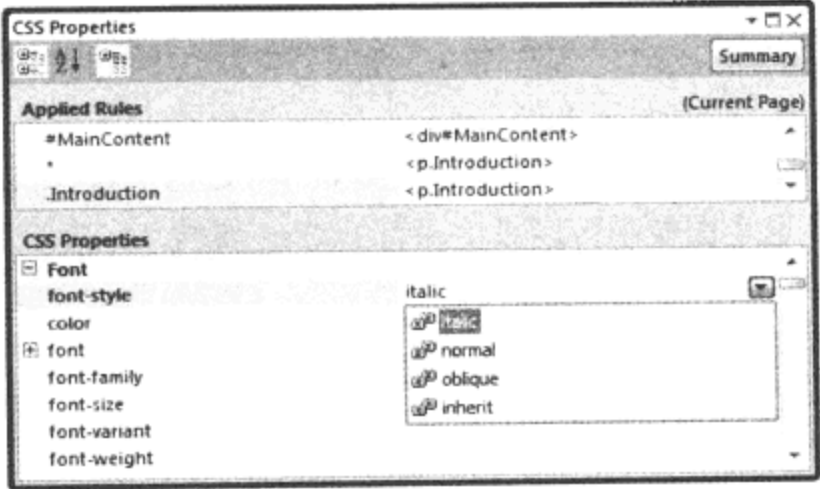


图 3-21

要特别注意窗口上方的 3 个按钮，因为它们容纳了一些有用的功能。前两个按钮允许在按类别的模式和按字母顺序的模式之间切换，使得更容易找到正确的属性。第三个按钮用来在列表上方(正如图 3-21 所示的情况)或者在它们于列表中的默认位置显示应用的属性。

(12) 最后，保存所有修改，并在浏览器中打开 Default.aspx 页面(如图 3-22 所示)。可以发现第一段现在除了文本中的链接为绿色外，其余内容都显示为蓝色斜体字。此外，如果跟着第 2 章的所有指令做了，那么文本“paying a visit”就是红色的，它是由嵌入的 CSS 类设置的。

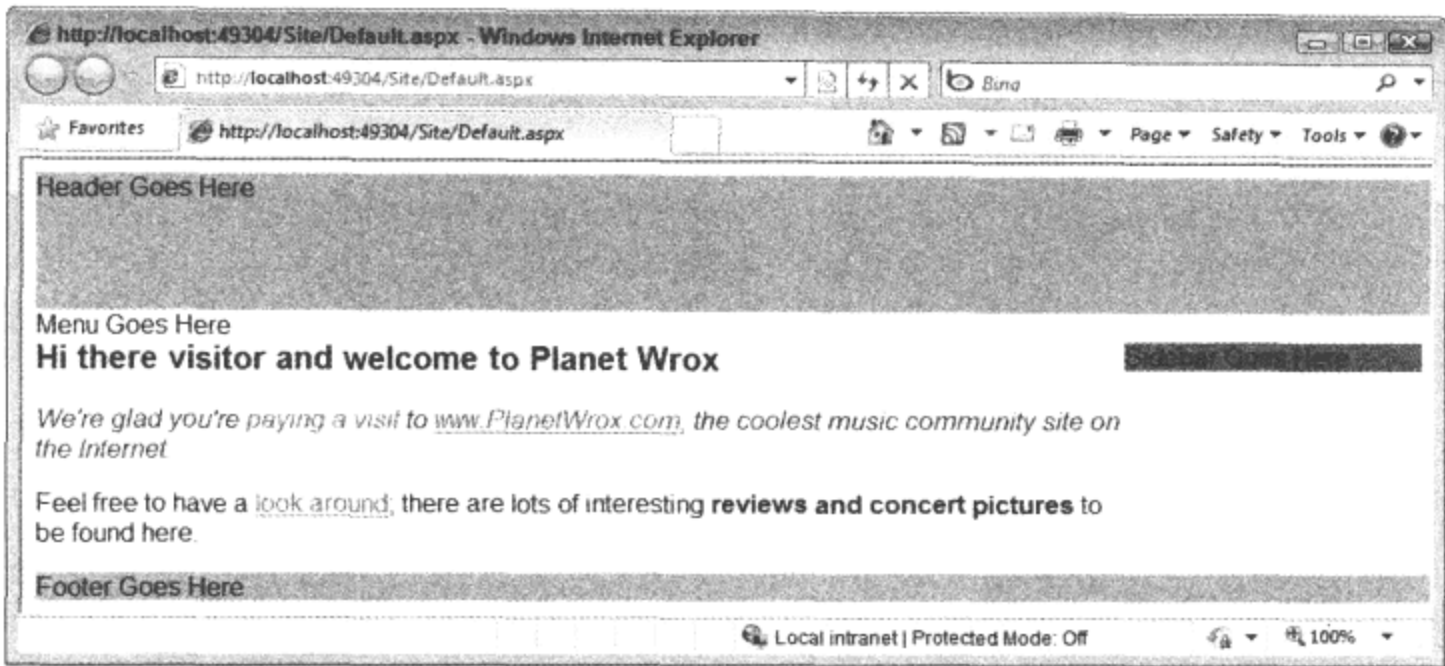


图 3-22

(13) 切换回 VWD 并在 Markup 视图中查看页面。在页面的<head>部分，应当看到下面的内嵌样式表：

```
.Introduction
{
    font-style: italic;
    color: #003399;
}
</style>
<link href="Styles/Styles.css" rel="stylesheet" type="text/css" />
</head>
```

工作原理

VWD 提供的众多工具使得为 Web 站点编写 CSS 很简单。不需要手写任何代码，也不需要记得 CSS 标准所支持的各种属性。相反，只需从 CSS Properties 面板上的不同列表中选择它们。这个面板允许手动输入值， 也提供了方便的工具来从下拉列表中选择颜色、文件和项。

在 Properties 面板中作的所有修改都将应用到相关的样式表中，不管使用的是内联、内嵌还是外部样式表。与此同时，还会更新 Design 视图以反映设置的新 CSS 选项。

当查看<h1>元素时，可以看到 VWD 通过将一个内边距值设置为 0px 创建了一个内联样式，以反映 4 边的情况，并将一个页边距设置为 0px 0px 10px 0px 来逐个控制 4 边。

在创建了一组有用且可以重用的样式后，需要有一种方法将现有样式应用到其他页面或 HTML 元素中。下面将介绍其工作原理。

3.3.3 应用样式

如果使用过 Microsoft Word，则很可能习惯 Styles 对话框，它列出了所有可用的样式，并允许把它们应用到文本的选中部分。用这种方式可以快速地 向文本块应用同样的格式。在 VWD 中的工作方式与此类似。从主菜单中选择 View | Apply Styles 命令打开 Apply Styles 窗口，就可以轻松地 向页面中的元素应用样式规则。

试一试 使用 Apply Styles 窗口

在这个练习中，将重用Introduction 类，并将它也应用到页面的第二段。这样，这两个段落最后的外观就是相同的。

(1) 仍然使 Default.aspx 处于打开状态，并确保在 Design 视图中，然后通过单击选中页面中的第二段。确保文档窗口下方的标记选择器显示<p>标记被选中，而不是可能为<p>元素的一部分的其他标记，如。如果文本中只有一段，那么要先创建一个新段落(在 Design 视图中第一段后面按下 Enter 键)，输入一些文本，然后选中那个段落。

(2) 通过选择 View | Apply Styles 命令打开 Apply Styles 窗口。这个窗口显示了它在当前页面中找到的所有选择器和附加的任何样式表。如果没有看到如图 3-23 所示的所有样式，则单击 Options 按钮并选择 Show All Styles 选项。

为了帮助找到正确的样式，VWD 使用了几个不同的可视化线索。首先，Apply Styles 窗口用红、绿、蓝和黄色点分别表示 ID 选择器、类选择器、元素选择器和内联样式。图 3-23 仅显示红色和绿色点(因为本书是黑白的看不出来)，因为<p>元素没有应用任何内联样式。然而，如果选择了<h1>元素，就会出现一个内联样式和一个元素选择器。在试着这么做时，一定要再次确认选中了<p>元素。此外，页面中当前使用的样式都由附加的圆圈着，图 3-23 中的所有选择器就是那样。

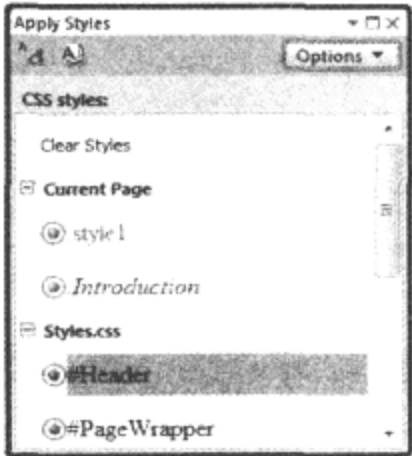


图 3-23

(3) 单击 CSS Styles 列表中的 Introduction 类，VWD 会向<p>标记添加一个 class 属性：

```
<p class="Introduction">
    Feel free to have a <a href="Default.aspx">look around</a>; there are lots of
    interesting <strong>reviews and concert pictures</strong> to be found here.
</p>
```

如果要应用多个类，则在单击列表中其他类之一时，按住 Ctrl 键，这样就会应用一个类列表，元素的 class 属性之间由一个空格隔开。也可以对 Markup 视图选中的样式应用同样的步骤。

(4) 单击 Clear Styles 按钮，就可以快速地从标记中删除现有类和内联样式。考虑第 2 章中的一个 HTML 片段，那时用了 Formatting 工具栏格式化页面中的文本。如果使用的是 Foreground Color 按钮，那么最后代码会类似这样：

```
We're glad you're <span class="style1">paying a visit</span>
```

要删除 class 属性，可选择标记选择器中的标记，或者简单地在 Markup 视图中单击标记，然后在 Apply Styles 窗口中单击 Clear Styles，这个项可以从图 3-23 上方看到。最后的 HTML 会是这样：

```
We're glad you're paying a visit
```

因为文本两端的空没有用，所以 VWD 也删除了它。
从 HTML 元素中删除样式属性的工作方式也是这样。

工作原理

再次说明，VWD 能保持所有的相关窗口同步：Design 视图、Markup 视图和各种 CSS 设计工具。当从 Apply Styles 窗口中应用一个类时，VWD 会将请求的类添加到 Markup 视图中选中的 HTML 元素中，然后它也会更新 Design 视图窗口。类似地，当从 Design 视图中的内嵌样式中删除一个选择器或声明时，Design 视图和 CSS Tools 窗口都会进行更新。

本章要介绍的最后一个 CSS 功能位于 Manage Styles 和 Apply Styles 窗口中。除了可以向文档中附加 CSS 文件外，这些窗口还有助于轻松地管理样式。

3.3.4 管理样式

由于添加新的内联和内嵌样式是如此容易，因此页面可能很快变得混乱不堪。要实现可重用性，应将尽可能多的内联和内嵌样式移到一个外部样式表中。这就是 Apply Styles 和 Manage Styles 窗口的作用所在。

试一试 使用 Manage Styles 和 Apply Styles 窗口管理样式

在本章前面，修改了<h1>元素，并向标题应用了内边距和页边距。然而，Default.aspx 不是在标题中从该样式获益的唯一页面，因此有必要把它移到 Styles.css 文件中。类似地，Introduction 类似乎重用性够好，因此可以把它包括到 Styles.css 文件中，以便其他页面能访问它。本“试一试”练习显示了如何在站点中移动样式。

- (1) 确保 Default.aspx 仍然是打开的，如有必要，切换到 Markup 视图。
- (2) 定位到<h1>标记并再单击一次。VWD 突显了位于文档窗口底部的标记选择器中的标记，以表明它是活动标记，如图 3-24 所示。
- (3) 通过从主菜单中选择 View | Apply Styles 命令打开 Apply Styles 窗口。如果窗口与其他窗口停靠在一起，则只需简单地单击它的选项卡来让它成为活动状态。请确保窗口不是偶然地与主文档窗口停靠在一起，而是浮动的或是放置在文档窗口的一侧。Apply Styles 窗口下方显示了内联样式(如图 3-25 所示)。

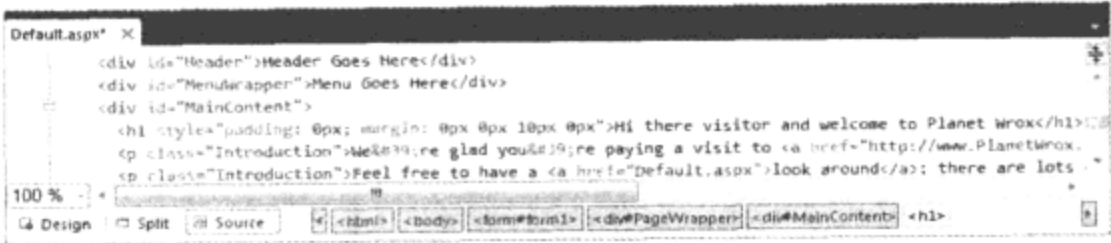


图 3-24

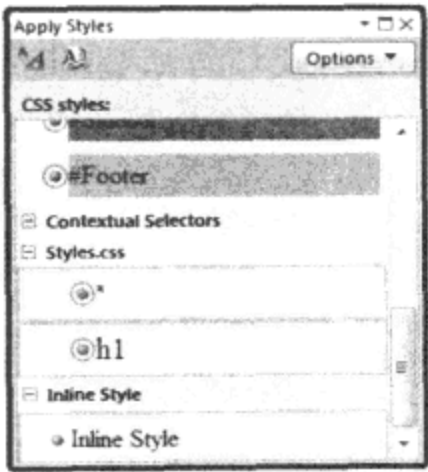


图 3-25

(4) 右击 Inline Style，并选择 New Style Copy 命令。这时会出现 New Style 对话框，允许创建一个基于当前选项的新样式。在这个窗口的上方，从 Selector 下拉列表中选择 h1，并从 Define in 下拉列表中选择 Existing style sheet 选项。从 URL 下拉列表中，选择 Styles/Styles.css 选项。如果这个项不可用，则单击 Browse 按钮来定位并选择它。对话框最后应如图 3-26 所示。

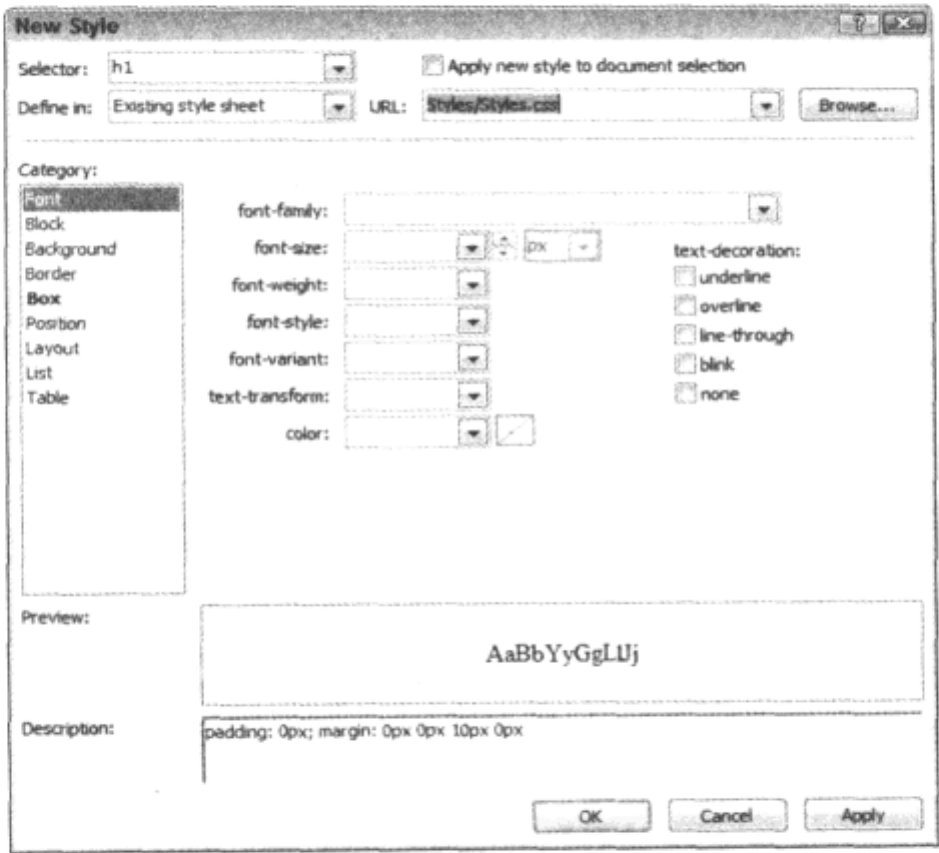


图 3-26

(5) 单击 OK 按钮关闭对话框。VWD 会创建 h1 样式的一个副本，并把它放在文件 Styles.css 中。注意 VWD 在 Styles.css 文件中为 h1 创建了一个新的选择器，而不是向现有规则集中添加内边距和页边距信息。如果愿意，可以将这两个选择器手动合并为一个。

(6) 在 Apply Styles 窗口中，再次右击 Inline Style，这次从上下文菜单中选择 Remove Inline Style 命令。这样会把样式属性从 h1 元素中删除。

(7) 切换到 Manage Styles 窗口。请再次确认窗口是放在文档窗口的一侧，而不是停靠在文档窗口内部。在 Current Page 项下，定位 Introduction 选择器，如图 3-27 所示。

(8) 单击 Introduction 选择器，然后把它拖到 Styles.css 的区域中，例如把它放在 h1 选择器后面。注意，当鼠标悬停在选择器上时，VWD 会在选择器之间画线，表示选择器最后将出现的位置。图 3-27 显示了如何将 Introduction 选择器从当前页面拖到 h1 和 #PageWrapper 选择器之间的 Styles.css 区域中。

(9) 一旦把选择器放到 Manage Styles 窗口的 Styles.css 部分中，则与该选择器关联的样式就会从当前页面中去除，然后插入到 Styles.css 中。由于那个 CSS 文件包括在使用 <link/> 元素的当前页面中，因此在 Design

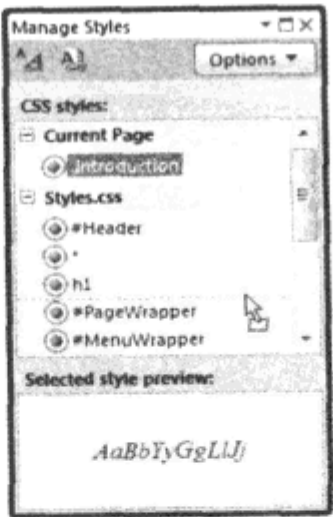


图 3-27

视图中看不到区别。现在，可以把空<style>元素从 Default.aspx 中删除，因为不再需要它了。

(10) 保存所作的任何修改，然后通过按下 Ctrl+F5 组合键在浏览器中打开 Default.aspx 页面。注意，这些段落没有改变，仍然使用的是蓝色斜体字。

工作原理

遗憾的是，VWD 不允许把内联样式移到外部样式表文件中。然而，可以通过创建现有样式的副本，然后删除原始内联样式来获得同样的效果。在文件之间移动内嵌或外部样式表就容易得多。可以简单地把样式从一个文件拖到另一个文件中，并且 VWD 会自动移动代码。这使得组织 CSS 极其容易。不要让所有嵌入的 CSS 都留在页面中，因为那样就可能会碰到它，现在可以简单地把它拖放到一个外部文件中。这使得在其他页面中重用那些样式容易得多，从而降低了页面大小和页面膨胀，并使站点更容易管理。显然，重要的是 CSS 所移到的文件被附加到了正在使用的页面中。

3.4 关于使用 CSS 的实用提示

请根据这些提示来制作大部分 CSS：

- 花一些时间熟悉 CSS 支持的众多属性。最佳方法是在 Demos 文件夹中创建一个崭新的页面，再创建几个像<div>和<p>标记这样的 HTML 元素，然后简单地配备几个不同的属性。通过在 CSS Properties 面板上尝试其中的很多属性，就能对可用的选项心中有数。这样，以后如果要向一些内容应用某种效果，就会容易得多。
- 当创建自定义 CSS 类时，试着起一些能描述规则的行为而不是描述外观的名称。例如，使用一个名为.Introduction 的类样式化页面的第一段，它就是一个不错的描述。通过它可以修改底层值，而不会影响名称的实际意思。但是像.BlueAndItalic 这样的类名以后肯定会带来问题，如果以后决定将蓝色改为黑色怎么办呢？只好要么用不是描述自身行为的非常别扭的类名，要么就要重命名类，然后更新整个站点，将对旧类的引用改为.BlackAndItalic。
- 尽量创建一个较小而且可以重用的规则集。在必要时可以合并，而不是创建庞大完整的规则，只能在单个 UI 元素上使用。例如，不要创建这样的样式：

```
.ImportantHeading
{
    font-size: 20px;
    font-weight: bold;
    color: red;
}
```

最好创建一些容易重用的轻量级规则：

```
h1
{
    font-size: 20px;
}

.Attention
```



```
{
    font-weight: bold;
    color: red;
}
```

在将.Attention类应用到如下标题时:<h1 class="Attention">,将会得到与赋予它 ImportantHeading类时完全相同的行为。然而,使用单独的 Attention 类时,就创建了一个可重用的规则,可以应用到其他需要用户注意的元素上,如<p>或元素。

3.5 本章小结

本章详细介绍了 CSS——样式化 ASPX 和 HTML Web 页面最重要的语言。

CSS 克服了使用 HTML 样式化 Web 页面时存在的局限,因为它可以用来最小化页面膨胀,提供对页面外观更好的控制,而且通常有助于创建加载更快且容易维护的 Web 站点。

很好地了解了 CSS 术语后,将会发现使用 VWD 提供的很多 CSS 工具变得容易了。像 Manage Styles 和 Apply Styles 窗口、Style Builder 和代码编辑器中的智能 IntelliSense 等工具,都使 CSS 的编写和管理轻松愉快。

CSS 不仅可以像本章介绍的这样应用到 HTML 中,而且也可以应用到 ASP.NET 服务器控件上。应用到这些控件的 CSS 最终在浏览器中会显示为纯 HTML,这里采用的原理与本章提到的相同。第 4 章将详细介绍很多可用的 ASP.NET 服务器控件。

3.6 练习

1. 使用外部样式表优于内嵌样式表的地方主要是什么?
2. 写一个 CSS 规则,将站点中所有的一级标题(h1)的外观改为:
 - 标题使用 Arial 字体
 - 标题应是蓝色的
 - 标题必须是 18 像素的字体大小
 - 标题上边框和左边框应为蓝色细边

对于最后一个要求,可以在 CSS 文件中查看 VWD 的 IntelliSense 列表来找到边框属性的另一个缩略版本。

3. 在下面两个规则中,哪个规则比较容易在 Web 站点中跨页面重用? 请解释原因。

```
#MainContent
{
    border: 1px solid blue;
}

.BoxWithBorders
{
    border: 1px solid blue;
}
```

4. VWD 允许以多种不同的方法向页面中附加外部样式表。请指出其中 3 种不同方法。
练习的答案见附录 A。

本章要点回顾

CSS	层叠样式表，在浏览器中布局 Web 页面的语言
CSS 方框模型	用于计算元素尺寸大小的模型，包括高度、宽度、内边距边框和页边距
声明	为要应用该声明的元素决定样式的属性和值的组合
内嵌样式表	在页面的<style />元素中定义的 CSS 代码
外部样式表	在一个单独的文件中定义的 CSS 代码，之后通过<link />元素被包括在页面中
内联样式表	使用 style 属性直接在元素上定义的 CSS
规则集	一个选择器和一个或多个包含在一对花括号中的声明的组合
选择器	一个 CSS 指令，指向页面中的一个或多个元素。现在有多种选择器，包括全局选择器、ID 和类选择器以及元素选择器

第4章

使用 ASP.NET 服务器控件

本章要点

- ASP.NET 服务器控件的概念
- 可自由支配的各种类型的服务器控件
- 大部分服务器控件都拥有的行为
- ASP.NET 运行库如何在页面上处理服务器控件
- 服务器控件如何在不同的回发之间维持其状态

ASP.NET 服务器控件是 ASP.NET 的重要组成部分。在 VWD 中构建的几乎所有页面都包含一个或多个服务器控件。这些控件有各种各样的类型和大小，有 Button 和 Label 这样的简单控件，也有复杂的控件，如第 7 章和第 13 章将会介绍的可以显示数据源(如数据库或是 XML 文件)中数据的控件 TreeView 和 GridView。

ASP.NET 服务器控件的体系结构已经完全集成到了 ASP.NET 中，为我们提供了一个在当今构建 Web 站点的技术中相当独特的功能集。本章将介绍这些服务器控件是什么，它们的工作原理是什么，以及在安装 Visual Web Developer 时哪些控件可以即装即用。

本章首先一般性地介绍几个服务器控件，指出如何通过把它们添加到 Design 或 Markup 视图中以在代码中定义它们。

下一节将全面介绍 VWD Toolbox 中的诸多可用控件。

4.1 服务器控件简介

了解服务器控件如何工作以及它们完全不同于以其他语言(如传统 ASP 或 PHP)定义控件的方式是很重要的。PHP 是创建动态 Web 站点的另一种流行编程语言。

例如，为了改变使用这些语言写的文本框中的文本，要使用纯 HTML，并与服务器端代码混合在一起。其工作原理与第 2 章中在页面上显示了当前日期与时间的那个示例类似。例如，要使用传统 ASP 创建一个包含一条消息及当前时间的文本框，可以使用下面的代码：

```
<input type="text" value="Hello World, the time is <%=Time()%>" />
```

可以看出，这段代码中包含纯 HTML，它与一个服务器端代码块混合在一起。该代码块由<%和%>界定，用等于号(=)输出当前时间。这种编程方式有一个重大缺陷：HTML 和服务器端代码混在一起，使得页面难以编写和管理。虽然这只是一个很小的示例，代码还不是很理解，但是这种编程方法很快就会导致页面非常混乱且复杂。

使用服务器控件就不会这样。在 ASP.NET 中，控件“存活”在服务器上的 ASPX 页面内。当在浏览器中请求页面时，服务器端控件就由 ASP.NET 运行库(负责接收和处理 ASPX 页面请求的引擎)处理。然后控件就会输出客户端 HTML 代码，并将其附加到最终页面输出的后面。最终出现在浏览器中用来构建页面的就是该 HTML 代码。

因此，不需要直接在页面中定义 HTML 控件，只需用下面的语法定义 ASP.NET 服务器控件即可，其中斜体部分根据控件的不同而不同。

```
<asp:TypeOfControl ID="ControlName" Runat="Server" />
```

使用 ASP.NET 4 自带的控件时，通常要在控件名前添加 asp:前缀。例如，要创建一个可以容纳相同的欢迎消息和当前时间的 TextBox，可以使用下面的语法：

```
<asp:TextBox ID="Message" Runat="Server" />
```

注意，此控件有两个特性：ID 和 Runat。ID 特性用来唯一地标识页面中的一个控件，以便对它进行编程。注意要让页面上的每个控件有一个唯一的 ID；否则 ASP.NET 运行库就无法理解指的是哪个控件。如果不小心输入了重复的控件 ID，VWD 会在错误列表中指出这一问题。强制性的 Runat 特性用来指出该控件存活在服务器上。如果没有这个特性，这些控件就不被处理，且最终会直接显示为 HTML 源代码。如果发现浏览器的 HTML 的最后输出中少了某个控件，请确保该控件包含了这个必需特性。注意，对于非服务器端元素(如纯 HTML 元素)，Runat 特性是可选的。如果在非服务器端控件上使用了这个特性，可以通过编写代码访问它们。在本书后面将介绍关于它们的更多知识。

输入 runat 并按下 Tab 键可以轻松地为现有元素添加 Runat 特性。

前面关于 TextBox 的示例使用的是可以自动结束的标记，这个标记会将结束斜杠自动嵌入在起始标记后。对于不需要包含子内容(如文本或其他控件)的控件来说，这是很常见的。然而，使用单独的结束标记的较长版本也是可以接受的。如下所示：

```
<asp:TextBox ID="Message" Runat="Server"></asp:TextBox>
```

可以通过选择 Tools | Options | Text Editor | HTML|Formatting | Tag Specific Options 命令来控制每个标记的默认闭合行为。

可以通过页面中的内联代码或者独立的 Code Behind 文件中的代码对这个文本框编程，详见第 2 章。要设置欢迎消息和时间，可以使用下面的代码：

VB.NET

```
Message.Text = "Hello World, the time is " & DateTime.Now.TimeOfDay.ToString()
```


C#

```
Message.Text = "Hello World, the time is " + DateTime.Now.TimeOfDay.ToString();
```

从上面的代码可以看出，页面的标记部分中，控件的定义现在已经与定义文本框中显示的文本的实际代码分离开来。这样就可以更轻松地定义和为文本框(或者其他任何控件)编写程序，因为它允许一次只集中于一个任务：要么声明控件和它在页面的标记部分中的可视化外观，要么从代码块中通过编程实现它的行为。

下面的“试一试”练习将说明服务器控件如何向客户端发送它们底层的 HTML。

试一试

使用服务器控件

在本练习中，将向页面添加一个 TextBox 控件、一个 Label 控件和一个 Button 控件。在浏览器中请求该页面时，会将这些服务器控件转换为 HTML，然后再发送给客户端。如果在浏览器中查看页面的最终 HTML，就能发现这些 HTML 完全不同于初始的 ASP.NET 标记。

- (1) 在 Visual Web Developer 中打开 Planet Wrox 项目。
- (2) 在 Solution Explorer 的 Demos 文件夹中，创建一个新的 Web 窗体，命名为 ControlsDemo.aspx。选择要使用的编程语言，并确保 Web 窗体使用的是 Code Behind 模式。
- (3) 切换到 Design 视图。从 Toolbox 中，把一个 TextBox、一个 Button 和一个 Label 控件拖放到创建页面时添加的<div>标记的虚线内的设计界面上。

在 TextBox 前面输入文本 Your name，并将光标放在 Design 视图中 Button 和 Label 控件之间，再按下 Enter 键，添加一个换行符。如果光标放不到两个控件之间，可以把它放在 Label 控件后面，然后按下左箭头键两次。第一次按时，会选中 Label 控件；按第二次时，光标就会位于两个控件之间，这时就可以按下 Enter 键。此时 Design 视图应如图 4-1 所示。

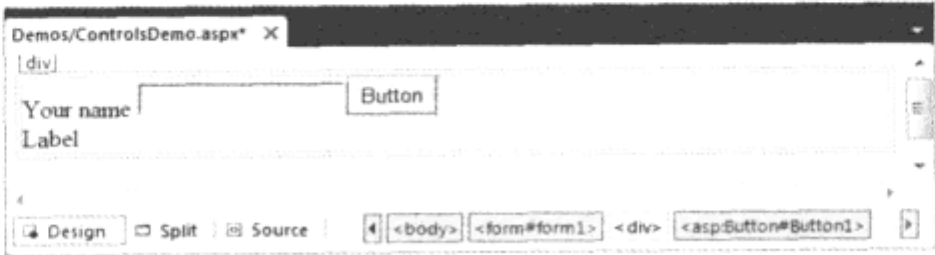


图 4-1

- (4) 右击 Button 按钮并选择 Properties 命令，打开控件的 Properties 面板。选择 Button 后按下 F4 键也能达到同样的效果。这时会出现如图 4-2 所示的窗口，可以在这里修改控件的属性，从而影响控件在运行时的行为。
- (5) 把控件的 Text 属性设置为 Submit Information，然后把它的 ID(它会一直存在于列表下方的括号中)设置为 SubmitButton。
- (6) 用 Properties 面板把 TextBox 的 ID 改为 YourName。
- (7) 用 Properties 面板清除 Label 的 Text 属性。可以在面板中右击该属性的标签选择 Reset 命令，或者手动清除文本。可以让它的 ID 保留为 Label1。

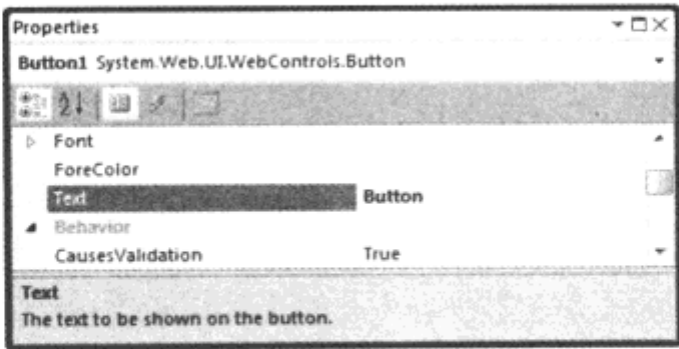


图 4-2

(8) 仍然是在 Design 视图中，双击该按钮使 VWD 向页面的 Code Behind 文件中添加一些代码。在浏览器中单击该按钮时就会激活此 Code Behind 文件。将下面突出显示的代码添加到 VWD 插入的代码块中：

VB.NET

```
Protected Sub SubmitButton_Click(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles SubmitButton.Click
    Label1.Text = "Your name is " & YourName.Text
End Sub
```

C#

```
protected void SubmitButton_Click(object sender, EventArgs e)
{
    Label1.Text = "Your name is " + YourName.Text;
}
```

注意，VB.NET 代码中不需要使用下划线来分开超过两行的代码。而之前的 VB.NET 版本要求使用下划线来分开超过两行的代码。

(9) 保存对页面所做的修改，然后按下 Ctrl+F5 组合键在浏览器中打开它。当它出现在浏览器中时，不要单击该按钮，而是通过在浏览器中右击页面打开页面的源代码，并选择 View Source 或 View Page Source 命令。看到的 HTML 代码应如下所示(在这里稍微改变了 HTML 的格式以适应页面)：

```
<div>
  Your name <input name="YourName" type="text" id="YourName" />
  <input type="submit" name="SubmitButton" value="Submit Information"
    id="SubmitButton" />

  <br />
  <span id="Label1"></span>
</div>
```

(10) 切换回浏览器，在文本框中输入名称，然后单击该按钮。当页面开始重新加载时，通过浏览器的右击菜单再次在浏览器中打开页面的源代码。这时代码看起来应如下所示：

```
<div>
  Your name <input name="YourName" type="text" value="Imar" id="YourName" />
  <input type="submit" name="SubmitButton" value="Submit Information"
    id="SubmitButton" />

  <br />
```

```
<span id="Label1">Your name is Imar</span>
</div>
```

注意其中两行突出显示的代码已经改变了，现在显示的是在文本框中输入的名称。此时，可以忽略页面中的其他 HTML。

工作原理

顾名思义，ASP.NET 服务器控件存活在服务器上的 ASPX 页面中，在该页面中控件可以由 ASP.NET 运行库处理。当在浏览器中请求一个页面时，运行库就会用创建的控件生成 ASPX 文件在内存中的一个表示。一旦运行库打算向浏览器发送 HTML，它就会向页面中的各个控件请求它们的 HTML，然后注入最后的结果中。例如，当首次加载 Label 控件并请求它的 HTML 时，它会返回下面的代码：

```
<span id="Label1"></span>
```

从上面这行代码可以看出，虽然使用<asp:Label>语法定义了 Label 控件，但是它最终出现在浏览器中的只是一个简单的元素。因为该 Label 控件的 Text 属性是空的，所以在两个标记中看不到任何内容。其他控件也是如此：<asp:TextBox>最终出现为<input type="text">，而<asp:Button>最终显示为<input type="submit">。

当单击按钮时，控件会引发一个回发，将页面中控件的信息发送给服务器，然后在那里再次加载页面。此外，还会执行为处理按钮的 Click 事件所写的处理程序。该代码会采用在文本框中输入的名称，然后把它赋予 Label 控件：

```
Label1.Text = "Your name is " + YourName.Text;
```

目前，不用关心编写处理按钮的 Click 事件的代码的语法。在第 5 章将介绍其工作原理，以及为什么需要该代码。

在这一阶段，Label 控件包含在文本框中所输入的文本，因此当要求输出它的 HTML 时，这次会返回：

```
<span id="Label1">Your name is Imar</span>
```

本章后面讨论 ASP.NET 状态引擎时会详细介绍回发。

4.2 ASP.NET 服务器控件详解

由于在构建 ASP.NET Web 页面时大部分时间都会用到服务器控件，因此需要详细了解它们的工作原理以及如何使用它们。下一节将介绍如何向页面中添加控件，以及如何改变它们工作在浏览器中的方式。在后面的几节中，将概述所有服务器控件共同的行为。理解了这些共同行为后，就会发现把它们推广到其他控件以及新控件上都很容易，因此很快就能熟悉它们。

4.2.1 在页面中定义控件

正如在前面的“试一试”练习中所演示的，可以简单地把 Toolbox 中的控件拖放到页面的设计

界面上。这样，一开始就可以轻易地向页面中添加一组控件。然而，由于设计界面工作方式的原因，有时难以把它们精确地添加到想放的位置上。例如，可能很不容易把控件拖到一个 HTML 元素的起始和结束标记之间。幸运的是，在 Markup 视图中可以轻松地拖动 Toolbox 中的控件。此外，也可以直接在 Markup 视图输入控件的标记，让 IntelliSense 和代码段帮助输入不同的标记和属性。同样还可以发现，Properties 面板在 Markup 视图中也是可用的。只需简单地单击相关标记，Properties 面板就会更新以反映选中的标记。这样就可以轻松地修改控件的属性，还仍然可以准确地看到生成了什么标记。

如果在页面中查看一些控件的 Properties 面板，就会注意到它们中的很多都有类似的属性。下一节将介绍这些属性的具体内容和作用。

4.2.2 所有控件的共同属性

VWD Toolbox 中的大多数服务器控件都有一些共同的行为。其中一种行为称为“属性”，用于定义一个控件可以包含和显示的数据。第 5 章将深入介绍这些属性和其他的行为类型。每个服务器控件都有一个 ID，用来在页面中唯一地标识它；一个 Runat 特性，该特性总是设置为 Server，表示应在服务器上处理该控件；以及一个 ClientID 特性，它包含将赋予最终 HTML 中的元素的客户端 ID 特性。在 ASP.NET 3.5 及其之前版本中，ClientID 通常是自动生成的，而在 ASP.NET 4 中引入了新的 ClientIDMode 属性，它提供了更多的在客户端控制元素 ID 的方法。后面的章节将介绍它的工作原理。Runat 特性实际上并不属于服务器控件，但是有必要指出，控件的标记应当作为服务器控件处理，而不是最终在浏览器中显示为纯文本或 HTML。

除了这些属性外，很多服务器控件还有更多共同的属性。表 4-1 列出了最常见的属性，并说明了它们的用途。

表 4-1

属 性	说 明
AccessKey	允许设置一个键，使用这个键，就可以按下关联的字母在客户端访问控件
BackColor ForeColor	允许修改控件背景的颜色(BackColor)和文本的颜色(ForeColor)
BorderColor BorderStyle BorderWidth	· 修改浏览器中控件的边框。与第 3 章中的 CSS 边框属性的相似之处是没有重叠。这 3 个 ASP.NET 属性中的每一个都直接映射到它的 CSS 部分
CssClass	用来定义浏览器中控件的 HTML class 特性。这个类名随后指向在页面或在外部 CSS 文件中定义的 CSS 类
Enabled	确定用户是否可以与浏览器中的控件交互。例如，如果文本框是禁用的(Enabled="False")，就不能修改它的文本
Font	允许定义与字体有关的各种设置，如 Font-Size、Font-Names 和 Font-Bold
Height Width	确定浏览器中控件的高度和宽度
TabIndex	设置客户端 HTML tabindex 特性，确定用户按下 Tab 键时焦点沿着页面中控件移动的顺序

(续表)

属 性	说 明
ToolTip	允许设置浏览器中控件的工具提示。这个工具提示在 HTML 中被呈现为 title 特性，当用户把鼠标悬停在相关的 HTML 元素上时就会显示出来
Visible	确定是否将控件发送给浏览器。应该真的看到它作为一种服务器端可见的设置，因为不可见的控件根本不会发送给浏览器。这意味着它相当不同于第 3 章中在客户端隐藏元素的 CSS display 与 visibility 属性

要看到所有这些特性最终出现在浏览器中的样子，参见下面的 TextBox 服务器控件的标记：

```
<asp:TextBox AccessKey="a" BackColor="Black" ForeColor="White" Font-Size="30px"
  BorderColor="Yellow" BorderStyle="Dashed" BorderWidth="4" CssClass="TextBox"
  Enabled="True" Height="40" Width="200" TabIndex="1" ToolTip="Hover text here"
  Visible="True" ID="TextBox1" runat="server" Text="Hello World">
</asp:TextBox>
```

当在浏览器中请求带有这个控件的页面时，最终的 HTML 如下所示：

```
<input name="TextBox1" type="text" value="Hello World" id="TextBox1" accesskey="a"
  tabindex="1" title="Hover text here" class="TextBox" style="color:White;
  background-color:Black;border-color:Yellow;border-width:4px;
  border-style:Dashed;font-size:30px;height:40px;width:200px;"
/>
```

浏览器中文本框会显示如图 4-3 所示的效果。
注意，大部分服务器端控件属性都被移到了带 style 特性的 CSS 内联样式中。



图 4-3

在构建 Web 站点时，很少用这样的方式定义 TextBox。第 3 章曾经指出，应当尽可能地避免使用内联样式，而是选择外部 CSS 样式表。使用这个服务器端控件可以完成同样的行为：

```
<asp:TextBox ID="TextBox1" AccessKey="a" CssClass="TextBox" TabIndex="1"
  ToolTip="Hover text here" runat="server" Text="Hello World">
</asp:TextBox>
```

下面是 CSS 类：

```
.TextBox
{
    background-color: Black;
    color: White;
    font-size: 30px;
    border-color: Yellow;
    border-style: Dashed;
    border-width: 4px;
    height: 40px;
    width: 200px;
}
```

显然，第二个示例更容易阅读、重用和维护。如果希望另一个文本框与它有完全相同的样子，只需将 `TextBox` 赋予那个控件的 `CssClass` 即可。同时，注意我忽略了 `Enabled` 和 `Visible` 两个属性。因为这两个属性都默认为 `True`，因此不需要显式地在该控件的声明中指出。

虽然我们建议用 CSS 类来代替这些内联样式，但是最好了解服务器端控件属性，以在需要对它们进行更细化的控制时知道怎么使用。如果通过编程方式修改控件的属性(将在以后进行介绍)，它们最终将仍然显示为内联样式，因此可能会覆盖内嵌或外部样式表中的设置。

现在已经介绍了所有服务器控件都有的共同行为，下面具体看一下 ASP.NET 4 配备的大量控件。

4.3 控件的类型

ASP.NET 4 本身附带了大量的服务器控件，能够满足 Web 开发的大部分需要。为了更容易地找到正确的控件，因此将它们放在 Visual Web Developer Toolbox(按 `Ctrl+Alt+X` 组合键即可访问)的各个单独的控件类别中。图 4-4 显示了 Toolbox 中的所有可用类别。

在下面几节中，将看到各类别所包含的控件以及在设计时要求它们完成的任务。

在讨论各种控件时，我们会提到控件的属性(property)。例如，`TextBox` 有一个 `Text` 属性(在其他的属性中)，`ListBox` 有一个 `SelectedItem` 属性。有些属性只能通过编程设置，而不能用 `Properties` 面板来设置。关于以编程方式读取和修改控件属性将在第 5 章详细讨论。

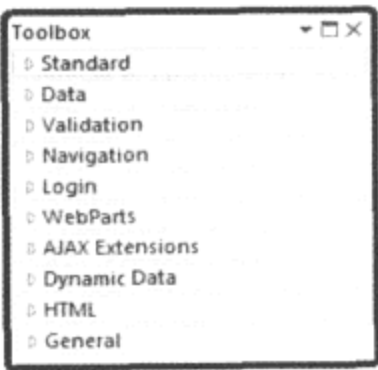


图 4-4

4.3.1 标准控件

`Standard` 类别中包含很多基本控件，几乎所有的 Web 页面都需要它们。已经介绍了其中的一部分，如本章前面介绍的 `TextBox`、`Button` 和 `Label` 控件。图 4-5 显示了 `Standard` 类别中的所有控件。

由于很多控件本身会表明它们的用途，因此下面几节不打算详细描述所有控件，仅仅简要强调几个重要控件。

1. 简单控件

Toolbox 中包含很多简单易懂的控件，包括 `TextBox`、`Button`、`Label`、`HyperLink`、`RadioButton`

和 **Checkbox**。在 **Toolbox** 中，它们的图标提供了很好的线索，可以猜出它们在浏览器中的作用。在本书的其余部分，会多次用到这些控件。

2. 列表控件

标准类别中还有许多在浏览器中表现为列表的控件。这些控件包括 **ListBox**、**DropDownList**、**CheckBoxList**、**RadioButtonList** 和 **BulletedList**。要向列表中添加项，可以在控件的起始和结束标记之间定义 `<asp:ListItem>` 元素，如下面的示例所示：

```
<asp:DropDownList ID="FavoriteLanguage" runat="server">
  <asp:ListItem Value="C#">C#</asp:ListItem>
  <asp:ListItem Value="Visual Basic">Visual Basic</asp:ListItem>
  <asp:ListItem Value="CSS">CSS</asp:ListItem>
</asp:DropDownList>
```

DropDownList 控件允许用户一次只能选择一项。要以编程方式查看列表控件中当前活动和选中的项，可以查看它的 **SelectedValue**、**SelectedItem** 或 **SelectedIndex** 属性。**SelectedValue** 返回一个包含选中项的值的字符串，如上面示例中的 **C#**或 **Visual Basic**。**SelectedIndex** 返回列表中项基于 0 的索引。对于上面的示例，如果用户选择 **C#**，那么 **SelectedIndex** 将会为 0。类似地，当用户选择 **CSS** 时，索引将为 2(列表中的第三项)。**BulletedList** 控件不允许用户作选择，因而也不支持这些属性。

对于允许多重选择的控件(如 **CheckBoxList** 和 **ListBox**)，可以在 **Items** 集合之间循环，并且查看选中了哪些项。在这种情况下，**SelectedItem** 仅返回列表中第一个选中的项；而不是返回所有选中项。下面的“试一试”练习将说明如何访问所有选中项。

为了了解如何向列表控件中添加列表项，以及如何读取选中的值，下面的“试一试”练习会指导如何使用两个要求用户选择喜欢的编程语言的列表控件来创建一个简单的 Web 窗体。



图 4-5

试一试 使用列表控件

在本练习中将向页面中添加两个列表控件。此外，还会添加一个按钮，这样，当单击它时就会在一个 **Label** 控件中将选中的项显示为文本。

(1) 在 **Demos** 文件夹中，创建一个新的 Web 窗体，命名为 **ListControls.aspx**。一定要选中 **Place Code in Separate File** 选项来创建 **Code Behind** 文件。

(2) 切换到 **Design** 视图，从 **Toolbox** 中把一个 **DropDownList** 控件拖到页面中已经出现的 `<div>` 标记的虚线内的设计界面上。

(3) 注意，一旦把 **DropDownList** 控件拖放到页面上，就会出现一个名为 **DropDownList Tasks** 的弹出菜单，如图 4-6 所示。



图 4-6

弹出菜单称为 **Smart Tasks** 面板。当它出现时，就可以执行属于控件的大部分常见任务。在 **DropDownList** 中，有 3 个选项。第一个选项允许把控件与数据源绑定在一起，具体将在第 13 章介绍。第二个选项用来向列表中手动添加项。而最后一个选项用来设置控件的 **AutoPostBack** 属性。选中这个选项后，一旦用户从列表中选择了一个新项，控件就会将它包含的页面提交回服务器。

只有具有很多功能的复杂控件才会出现 **Smart Tasks** 面板。像 **Button** 和 **Label** 这样的简单控件不会出现这个面板。要重新打开 **Smart Tasks** 面板，可以右击设计器中的控件，并选择 **Show Smart Tag** 命令。也可以单击控件右上角的小箭头，如图 4-6 所示。

在 **DropDownList** 的 **Smart Tasks** 面板上，单击 **Edit Items** 链接打开 **ListItem Collection Editor** 对话框，如图 4-7 所示。

这个对话框可以用来向列表控件中添加新项。通过此窗口添加的项将被添加为控件的标记之间的 `<asp:ListItem>` 元素。

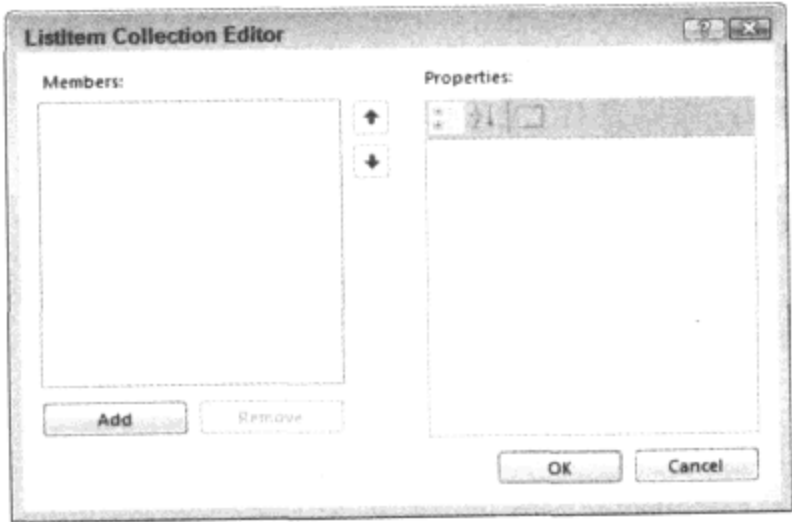


图 4-7

(4) 单击屏幕左边的 **Add** 按钮插入一个新的列表项。然后在右边的 **Properties** 面板中，为 **Text** 属性输入 **C#**，然后按下 **Tab** 键。一旦离开了 **Text** 属性，值就也复制给了 **Value** 属性。如果希望 **Text** 和 **Value** 属性采用相同的值，那么这样就会非常方便。不过，给 **Value** 属性指定一个不同的值会更好(通常就是这样做的)。

(5) 将步骤(4)重复两次，创建 **Visual Basic** 和 **CSS** 的列表项。可以使用对话框中间的上下箭头改变列表中项的次序。最后，单击 **OK** 按钮把项插入到页面中。最后的代码应如下所示：

```
<asp:DropDownList ID="DropDownList1" runat="server">
  <asp:ListItem>C#</asp:ListItem>
  <asp:ListItem>Visual Basic</asp:ListItem>
  <asp:ListItem>CSS</asp:ListItem>
</asp:DropDownList>
```

(6) 切换到 **Markup** 视图，然后从 **Toolbox** 中把 **CheckBoxList** 控件直接拖到位于 **DropDownList** 后面的代码窗口中。

(7) 复制在第(4)步和第(5)步创建的 **DropDownList** 中的 3 个 `<asp:ListItem>` 元素，并把它们粘贴到 **CheckBoxList** 的起始和结束标记之间。最后代码应如下所示：

```
<asp:ListItem>CSS</asp:ListItem>
</asp:DropDownList>
<asp:CheckBoxList ID="CheckBoxList1" runat="server">
  <asp:ListItem>C#</asp:ListItem>
  <asp:ListItem>Visual Basic</asp:ListItem>
  <asp:ListItem>CSS</asp:ListItem>
</asp:CheckBoxList>
```

(8) 切换到 Design 视图,然后在 Design 视图中把一个 Button 控件从 Toolbox 中拖动到 CheckBoxList 控件右边。Button 控件会放在 CheckBoxList 下面。接下来,拖动一个 Label 控件,并把它放在 Button 控件的右边。把光标放在控件之间然后按下 Enter 键两次,在 Button 和 Label 控件之间创建一些空间。双击 Button 以打开页面的 Code Behind 文件。

(9) 在 VWD 添加的代码块中,添加下面突出显示的代码,当用户单击该按钮时会执行这些代码:

VB.NET

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Button1.Click
    Label1.Text = "In the DDL you selected " &
        DropDownList1.SelectedValue & "<br />"

    For Each item As ListItem In CheckBoxList1.Items
        If item.Selected Then
            Label1.Text &= "In the CBL you selected " & item.Value & "<br />"
        End If
    Next
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = "In the DDL you selected " +
        DropDownList1.SelectedValue + "<br />";

    foreach (ListItem item in CheckBoxList1.Items)
    {
        if (item.Selected == true)
        {
            Label1.Text += "In the CBL you selected " + item.Value + "<br />";
        }
    }
}
```

注意,本例的 VB.NET 代码再次用到下划线来分开超过两行的代码。如果要让关键字 Handles 单独成为一行,在 VB.NET 中就要求使用下划线。

(10) 保存对页面所做的修改,然后在浏览器中请求它。从 DropDownList 中选择一项,选中 CheckBoxList 中的一项或多项,然后单击按钮。这时将出现如图 4-8 所示的内容。图 4-8 显示了 Mozilla Firefox 中的页面。

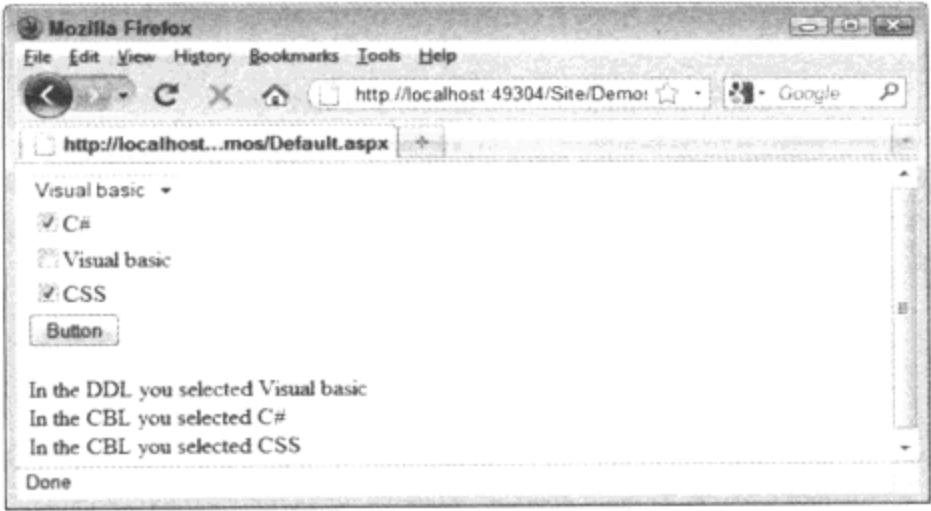


图 4-8

工作原理

各种列表控件都使用了<asp:ListItem>元素。这样，只要把它们从一个控件复制到另一个控件就可以轻松地重用它们。由于 DropDownList 一次仅支持一个选中项，因此相当容易得到选中的值。整个过程只需一行代码(这里显示为 C#代码)：

```
Label1.Text = "In the DDL you selected " + DropDownList1.SelectedValue + "<br />";
```

CheckBoxList 控件允许用户一次性地选择多个项。因此，需要稍微多写一点代码在项集合之间循环，以选中各个项的 Selected 属性(这里也是显示为 C#代码)。

```
foreach (ListItem item in CheckBoxList1.Items)
{
    if (item.Selected == true)
    {
        Label1.Text += "In the CBL you selected " + item.Value + "<br />";
    }
}
```

CheckBoxList 和其他列表控件有一个 Items 集合，包含了在代码中定义的所有项。因此，在这个“试一试”练习给出的代码中，CheckBoxList1 包含 3 项：分别适用于 C#、Visual Basic 和 CSS。每个 ListItem 又包含一个 Selected 属性，用来确定用户有没有选中列表中的项。

使用一个 foreach 循环(在 VB.NET 中是 For Each)，可以在 ListItem 元素的集合之间迭代(iterate)来逐个测试 Selected 属性。如果选中了列表中的项，它的 Selected 属性就是 true，它的 Value 就会附加到 Label 的文本后面。注意上面的代码示例使用+=(在 VB.NET 中是&=)为列表项的 Value 及其文本赋值。+=和&=语法是下面语法的简写：

```
Label1.Text = Label1.Text + "In the CBL you selected " + item.Value + "<br />";
```

这段代码采用了 Label 控件中的当前文本，并向它附加字符串"In the CBL you selected"+ item.Value + "
", 然后重新把整个字符串赋予回该 label 的 Text 属性。使用+=语法往往要稍微容易编写和理解一些，但是较长的版本也很常用。

VB.NET 和 C#都支持 For Each 循环，只是这两种语言在语法上略有不同。第 5 章将介绍关于循环和其他语言结构的更多知识。

同样要注意一下 `ListItems` 的建立方式。在这个“试一试”练习之前的第一个示例中有一些含有值和文本的 `ListItems` 元素：

```
<asp:ListItem Value="C#">C#</asp:ListItem>
<asp:ListItem Value="Visual Basic">Visual Basic</asp:ListItem>
<asp:ListItem Value="CSS">CSS</asp:ListItem>
```

当使用 `ListItems` Collection Editor 亲自向该列表添加项时，没有得到 `Value` 特性：

```
<asp:ListItem>C#</asp:ListItem>
<asp:ListItem>Visual Basic</asp:ListItem>
<asp:ListItem>CSS</asp:ListItem>
```

没有得到 `Value` 特性，是因为没有在 `ListItems` Collection Editor 中为该项提供显式的值。如果省略了 `Value`，就会隐式地将 `ListItems` 的起始和结束标记之间的文本作为值，在很多情况下这是很好的。然而，列表中有不同的 `Value` 和 `Text` 属性也相当常见。例如，如果有一个国家列表，就可以用国家的全名作为 `Text`(像 Netherlands)，并使用官方国家代码(nl)作为下拉列表的 `Value`。在本书的其他章节中会用到这些列表控件。

3. 容器控件

常常需要以某种方式将相关的内容和控件组合到一起操作。可以把控件(及其他标记)放在某个容器控件中，如 `Panel`、`PlaceHolder`、`MultiView`，或 `Wizard`。例如，可以使用 `PlaceHolder` 或 `Panel` 控件同时隐藏或显示几个控件。不用分别隐藏每个控件，只需隐藏包含各个控件和标记的整个容器即可。这两个控件各有优缺点。`PlaceHolder` 控件的好处是它不会向页面发布它自己的 HTML，因此可以用作容器控件，而不会在最终页面中产生任何副作用。然而，它缺少设计时支持，因此在 VWD 中难以在设计时管理 `PlaceHolder` 内的控件。相反，`Panel` 控件允许轻松地访问所有控件以及它所包含的其他内容，但是它自己则呈现为 `<div>` 标记，在很多情况下这不是一个问题，因此，一般最好使用 `Panel` 控件，因为它具有设计时支持。

`MultiView`(可以包含一个或多个 `<asp:View>` 控件)和 `Wizard` 相似的地方是：它们允许将一个长页面划分为多个区域，这样很有好处，例如容易填写长表单。`Wizard` 具有使用 `Previous`、`Next` 和 `Finish` 按钮在页面间移动的内置支持，而 `MultiView` 则必须通过编程进行控制。

面板控件详解

在下面的“试一试”练习中，将使用 `Panel` 控件为其他控件和标记创建一个容器。到目前为止只添加了一些文本，但是在随后的“试一试”练习中，将向面板中添加 ASP.NET 控件。

试一试 使用 Panel 控件

本练习介绍如何把 `Panel` 控件作为一些简单文本的容器。此外，还将使用一个简单的 `CheckBox` 控件来控制 `Panel` 在服务器上的可见性。

- (1) 首先用 Demos 文件夹中名为 Containers.aspx 的 Code Behind 文件创建一个新的 Web 窗体。
- (2) 将页面切换到 Design 视图，然后在设计界面上，从 Toolbox 中把一个 `CheckBox` 和一个 `Panel` 控件拖放到 `<div>` 元素的虚线框中。
- (3) 将 `CheckBox` 控件的 `Text` 属性设置为 Show Panel，给它起一个具有描述意义的名字，并用

Properties 面板将它的 AutoPostBack 属性设置为 True。除了可以在下拉列表中选择 True 来设置该属性外，还可以通过双击 AutoPostBack 属性或它的值在 True 和 False 之间进行选择。

(4) 在 Properties 面板上将 Panel 控件的 Visible 属性设置为 False。这样，当页面首次加载时就会隐藏 Panel 控件。

(5) 在 Panel 控件内，输入一些文本(如，I am visible now)。注意该面板的行为类似于 VWD 设计界面其余部分的行为。可以直接向它添加文本，选中并格式化它，甚至可以从 Toolbox 中向其添加新控件。该面板的代码最后应如下所示：

```
<asp:Panel ID="Panel1" runat="server" Visible="False">
    I am visible now
</asp:Panel>
```

(6) 在 Design 视图中双击 Checkbox 控件，在 VWD 添加的代码中输入下面突出显示的代码：

VB.NET

```
Protected Sub CheckBox1_CheckedChanged(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
    Panel1.Visible = CheckBox1.Checked
End Sub
```

C#

```
protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
    Panel1.Visible = CheckBox1.Checked;
}
```

(7) 保存所有修改，然后按下 Ctrl+F5 组合键在浏览器中请求页面。

(8) 当页面首次加载时，能看到的只有 CheckBox 和它旁边的文本。当单击 CheckBox 控件，在其中放置复选标记时，页面会重新加载，并会显示在第(5)步中输入的文本。



常见错误: 如果什么也没有发生，则回到 VWD 中页面的源代码处，确保 CheckBox 控件的 AutoPostBack 属性设置为 True。

这时如果查看浏览器中的 HTML(右击页面并选择 View Source 或 View Page Source 命令)，则会发现在第(5)步中输入的文本由<div>标记括了起来，其 id 为 Panel1：

```
<div id="Panel1">
    I am visible now
</div>
```

工作原理

在本“试一试”练习的第(4)步，将 Panel 控件的 Visible 属性设置为 False。这意味着当加载页面时，该控件在服务器上不可见，因此它的 HTML 永远不会让它出现在浏览器中。然后，当选中

CheckBox 时，会发生一个回发，将窗体中包含的信息发送到服务器上。在服务器上会运行一些代码，每当复选框的状态从选中改为非选中(或者反过来)时，就会激活这些代码。在该代码块中，会执行下面的代码(这里显示的是 C#代码)：

```
Panel1.Visible = CheckBox1.Checked;
```

这意味着只有选中复选框时 Panel 才是可见的。如果没有选中复选框，Panel 就会自动隐藏起来。

在第 5 章我们将介绍产生这种情况的代码的更多知识，以及为什么需要这样的代码，如带有 CheckBox1_CheckedChanged 的代码行。

正如前面指出的，在 Visual Web Developer 中很容易将文本和其他标记添加到 Panel 控件中。目前只添加了一些纯文本，下一节将介绍如何添加 Wizard 控件以及如何使用它。

Wizard 控件的神奇之处

Wizard 控件是一个出色的工具，可以将大的 Web 窗体分解，并把它们向用户呈现为字节大小的信息块。它不会在一个页面上让用户混淆许多控件及其上的文本，而是可以将页面分解，在独立的向导页面上呈现各个部分。然后，Wizard 控件会通过自动创建 Next、Previous 和 Finish 按钮来处理所有导航问题。在下面的“试一试”练习中，将使用向导来询问用户的名字及其喜爱的编程语言。虽然示例本身相当简单，却可以轻松地把这两个问题放在相同的页面上，且不使用户混淆。这个示例说明了向导如何工作，以及它为什么有作用。因此可以在更大的 Web 窗体上轻松地应用同样的技术。

试一试

用向导创建容易使用的表单

在前面的“试一试”练习创建的面板中，放置一个 Wizard 控件，允许用户填写一个分布在两个页面内的表单。这个向导将采用两步，一步是用户输入详细信息，另一步是在结果页面显示用户提供的数据。

(1) 确保在 Design 视图中 Containers.aspx 仍然是打开的。删除在上一个“试一试”练习中输入的文本“ I am visible now”，然后从 Toolbox 中把一个 Wizard 控件拖放到 Panel 内。把它的右边线再往右拖一些，将控件总宽度增加到 500px。页面现在如图 4-9 所示。



图 4-9

(2) 打开向导的 Smart Tasks 面板(单击右上角的箭头)，选择 Add | Remove WizardSteps 命令。在出现的对话框中单击 Add 按钮插入向导的第三步，如图 4-10 所示。

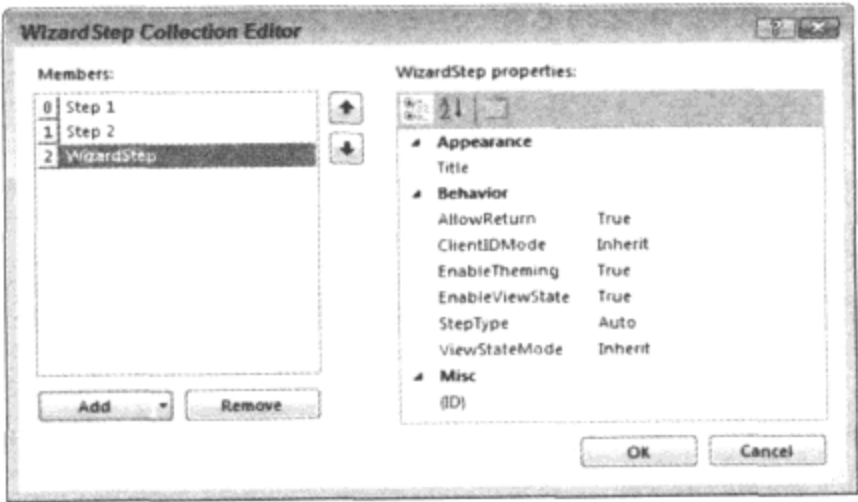


图 4-10

- (3) 单击对话框左边 Members 列表中名为 Step1 的第一个 WizardStep，然后将它的 Title 从 Step 1 改为 About You。将其他两步的 Title 分别设置为 Favorite Language 和 Ready。
- (4) 将第二步的 StepType(现在名为 Favorite Language)改为 Finish，最后一步的改为 Complete。可以让第一步的 StepType 保持设置为 Auto。单击 OK 按钮关闭 WizardStep Collection Editor 对话框。
- (5) 在 Design 视图中，单击左边列表中的 About You，让它成为活动步骤，然后将一个 Label 和一个 TextBox 控件拖到 Wizard 的右边。要将它们拖到 Wizard 右上角的灰色矩形框内，否则控件最后不会出现在 Wizard 内。将 Label 的 Text 属性设置为 Type your name，然后将 TextBox 的 ID 改为 YourName。完成后，Wizard 应如图 4-11 所示。

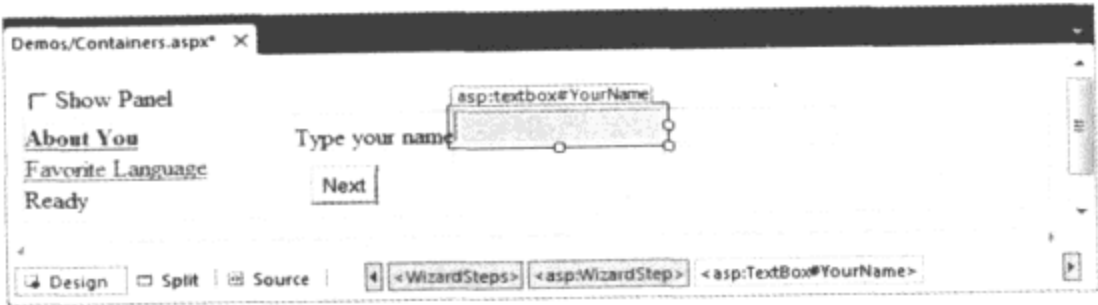


图 4-11

- (6) 单击左边列表中的 Favorite Language 项使它成为活动步骤。如果 Favorite Language 没有变成粗体，则打开 Wizard 的 Properties 面板，将 ActiveStepIndex 设置为 1。然后将一个 DropDownList 控件添加到向导步骤右半部分带灰边的矩形框内。通过将其 ID 设置为 FavoriteLanguage 来重命名该 DropDownList 控件。打开这个 DropDownList 控件的 Smart Tasks 面板，并选择 Edit Items 命令。添加在前面的“试一试”练习中添加的 3 项，分别用于 C#、Visual Basic 和 CSS。如果愿意，可以从页面 ListControls.aspx 中复制这 3 项并把它们粘贴到第(2)步中的<asp:DropDownList>标记之间。第二步的代码最后应是这样：

```
</asp:WizardStep>
<asp:WizardStep runat="server" Title="Favorite Language" StepType="Finish">
  <asp:DropDownList ID="FavoriteLanguage" runat="server">
    <asp:ListItem>C#</asp:ListItem>
    <asp:ListItem>Visual Basic</asp:ListItem>
    <asp:ListItem>CSS</asp:ListItem>
  </asp:DropDownList>
```

```
</asp:WizardStep>
<asp:WizardStep runat="server" StepType="Complete" Title="Ready">
```

(7) 切换到 Markup 视图，然后在名为 Ready 的最后一步 WizardStep 内，从 Toolbox 中拖动一个 标签(label)控件，并通过将它的 ID 设置为 Result 来重命名它。如果试图在 Design 视图中切换到最后一步，可能会发现向导消失了。如果发生这种情况，请切换到 Markup 视图并再次在<Wizard>控件的起始标记中将 ActiveStepIndex 设置为 0。

(8) 在 Design 视图中双击向导，添加下面突出显示的代码，当用户单击向导最后一步上的 Finish 按钮时会执行这段代码。如果在下面的代码段中发现 VWD 没有创建正确的代码，请选择 Wizard，按 F4 键打开控件的 Properties 面板，然后单击上面带有闪电图形的按钮(Properties 面板的工具栏中左起第 4 个按钮)，如图 4-12 所示。

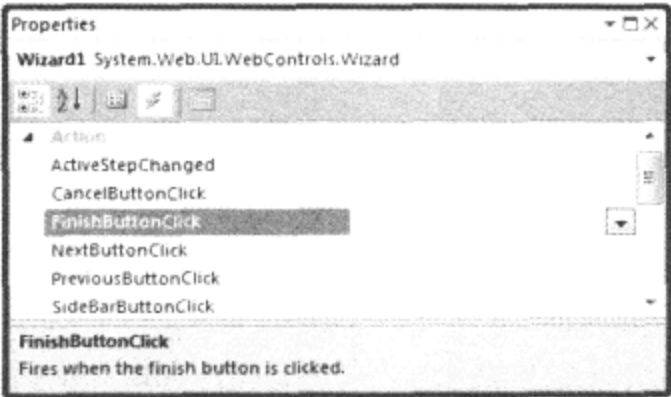


图 4-12

这一部分通常称为 Properties 面板的 Events 选项卡。定位并双击 Action 类别中的 FinishButtonClick。使用这两种方法都会得到 Wizard1_FinishButtonClick 的一些代码，这些代码需要用以下代码来扩展 (第 5 章将会介绍闪电按钮的具体作用)：

VB.NET

```
Protected Sub Wizard1_FinishButtonClick(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.WizardNavigationEventArgs) _
    Handles Wizard1.FinishButtonClick
    Result.Text = "Your name is " & YourName.Text
    Result.Text &= "<br />Your favorite language is " &
        FavoriteLanguage.SelectedValue
End Sub
```

C#

```
protected void Wizard1_FinishButtonClick(object sender,
    WizardNavigationEventArgs e)
{
    Result.Text = "Your name is " + YourName.Text;
    Result.Text += "<br />Your favorite language is " +
        FavoriteLanguage.SelectedValue;
}
```

(9) 切换回 Design 视图并打开 Wizard 的 Properties 面板，确保将它的 ActiveStepIndex 属性设置为 0。设计器会记住设计的最后一步，并在 Markup 视图中将该值存储在 Wizard 的 ActiveStepIndex

中。为了确保向导先启动第一页，应总是在保存修改和运行页面前将 `ActiveStepIndex` 设置为 0(或者在 `Design` 视图中单击 `Wizard` 控件中的第一步)。

(10) 按下 `Ctrl+F5` 组合键在浏览器中打开页面。选中 `CheckBox` 让 `Panel` 可见，然后在第一个向导页面上输入您的名字。单击 `Next` 按钮并选择喜欢的编程语言。注意现在 `Previous` 按钮已经可用了，如果要修改名字，它允许回到向导的第一步。除了可以单击 `Next` 和 `Previous` 按钮，也可以单击浏览器中向导左边的链接完成同样的操作。当单击 `Finish` 按钮时，将看到在向导中输入的信息，如图 4-13 所示。

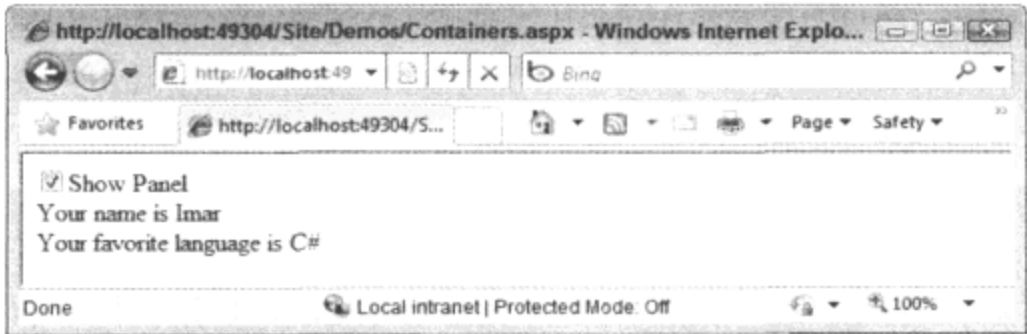


图 4-13

工作原理

`Wizard` 控件能够完成大部分非常复杂的工作。它会处理导航，确定何时显示正确的按钮(如 `Next`、`Previous` 和 `Finish`)，并确保在结果页面中，在向导步骤中添加的控件值仍然可用，这样就可以在结果标签(label)中显示它们。它使用 `View State` 的概念完成这些工作，在本章快结束时将会更详细地介绍这一概念。

您所要做的只是定义这些步骤并设置它们的 `StepType`。第一步的 `StepType` 被设置为 `Auto`。采用这个设置，向导就会计算出显示什么按钮。由于它是 `Wizard` 中的第一步，因此它会省去 `Previous` 按钮，因为它没有上一步。将第二步的 `StepType` 设置为 `Finish`，这样会通知向导绘制一个 `Previous` 按钮和一个 `Finish` 按钮。当单击 `Finish` 按钮时，向导会到达最后一步，该步的 `StepType` 会设置为 `Complete`。在这一步上，导航按钮是隐藏的，只能看到带结果的标签，它是用页面的 `Code Behind` 文件中的代码指定的。在第 5 章，将介绍完成这一功能的 `Code Behind` 中的代码的更多内容。

除了在前几节看到的那些控件外，还有另外几个控件值得研究。这里不会详细讨论每个控件，因为它们在这本书中不会再用。如果想了解这些控件的详细描述，可以参考 `VWD` 所附带的文档，也可以在 `Microsoft Development Network(MSDN)` 站点的 `http://msdn.microsoft.com` 链接上了解更多的信息。要在这个站点上查看这些控件的信息，在搜索引擎上输入控件名+`Control MSDN` 即可。例如：要想查看 `Wizard` 控件的更多信息，就搜索 `Wizard Control MSDN`。一般情况下，`MSDN` 站点就在搜索结果的最上边。

4. 其他标准控件

本节简要讨论 `Toolbox` 的 `Standard` 类别中的其余控件。本书后面章节的示例应用程序中将用到其中的不少控件。

LinkButton 和 ImageButton

对 `LinkButton` 和 `ImageButton` 控件的操作类似于普通的 `Button` 控件。单击它们时，两者都会引

起向服务器的一个回发。LinkButton 把自己显示为一个简单的<a>元素，但只发生回发(使用 JavaScript)，而不请求新页面。ImageButton 也是如此，只是显示一个图像，用户可以单击它以触发回发。

Image 和 ImageMap

这两个控件在浏览器中显示图像时非常相似。ImageMap 允许在图像上定义 hotspots，当单击时，要么引起一个到服务器的回发，要么导航到另一个页面。

Calendar

Calendar 控件提供了一个功能丰富的接口，允许用户选择日期。在本章最后讨论 ASP.NET 状态引擎时将介绍关于它的更多信息。

FileUpload

FileUpload 控件允许用户上传可以存储在服务器上的文件。在本书后面将介绍关于此控件的更多信息。

Literal、Localize 和 Substitute

这 3 个控件看起来有些像 Label 控件，因为它们都可以显示静态文本或 HTML。Literal 最大的优点是它本身不呈现额外的标记；它仅显示赋予 Text 属性的信息，因此对于显示 HTML 或者显示在 Code Behind 中构建的或从数据库检索的 JavaScript 非常有用。

Localize 控件在使用多种语言的 Web 站点中，并且能够从翻译后的资源文件中检索其内容。Substitute 控件用在高级缓存场景中，并且允许仅更新部分没有完全缓存的页面。有关这两个控件更深入的解释不在本书的范围之内，但是，如果想了解关于它们的详细解释，可以参见清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C# 和 VB.NET(第 7 版)》一书(ISBN：978-7-302-23524-8)。

AdRotator

这个控件允许在 Web 站点上显示随机广告。这些广告来自在服务器上创建的 XML 文件。由于它缺少像单击跟踪和记录这样的大多数最简单的情况所必需的高级功能，因此这个控件在当今的 Web 站点中用得不是太多。

HiddenField

HiddenField 控件可用来将数据存储在用各个请求提交的页面中。如果希望页面记住特定数据，而用户在页面中又不会看到，那么该控件就很有用。由于这个字段会显示在页面的 HTML 源代码中，因此终端用户可以访问，永远不要在其中存储任何敏感数据。

XML

XML 控件允许将数据从 XML 格式转换为另一种格式(如 XHTML)，以便显示在页面上。清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C# 和 VB.NET(第 7 版)》一书(ISBN：978-7-302-23254-8)中有该控件的详细描述。

Table

<asp:Table>控件在很多方面等同于它的 HTML<table>控件。然而，由于该控件位于服务器上，

因此可以对它进行编程，动态地创建新的列和行，以及向其中添加动态数据。

这个控件是我们讨论的 Toolbox 中最后一个 Standard 类别的控件。在大多数 Web 页面中，都会用到这些控件中的至少几个控件。本节的剩余部分将讨论 Toolbox 中的其他类别。由于其中的大部分控件都会在本书的其他章节以某种形式用到，因此本章只是简要地描述一下它们的用途，以便对它们有个大致的了解。其他章节中会有一些交叉引用，从中可以发现关于它们的更多信息。

4.3.2 HTML 控件

Toolbox 的 HTML 类别中包含许多 HTML 控件，它们看起来与 Standard 类别中的控件相似。例如，Input (Button)控件看起来就像<asp:Button>。类似地，Select 控件有<asp:DropDownList>和<asp:ListBox>作为它的对应控件。

与 ASP.NET 服务器控件相反，HTML 控件是客户端控件，在浏览器中直接出现在最终的 HTML 中。可以通过向它们添加 Runat="Server"特性将它们提供给服务器端代码。这样就可以对它们编程，从 Web 窗体的 Code Behind 到一些有影响的事情(如它们的可视性)都可以。

HTML 控件的功能比 Standard 类别中的控件的功能少得多。例如，Select 控件缺少使用 ListItem Collection Editor 向列表中添加新项的设计时支持。这样就不得不在 VWD 的 Markup 视图中手动添加项。

由于 Standard 和 HTML 类别中的控件看起来彼此很像，因此下一节将讨论它们的区别，并给出一些提示：何时最好使用这两类控件中的哪一种。

如何在 Standard 和 HTML 控件之间选择

Toolbox 中的 Standard 和 HTML 类别的控件之间似乎有一些重叠。因此应选择哪种类别呢？何时选择这种类别？一般来说，Standard 类别中的真正服务器控件提供了更多的功能，无论是在 VWD 中的设计时支持方面还是在运行时能做的事情方面都是如此。不过这种功能性是有代价的。因为它们增加了复杂度，所以处理服务器控件会多花一点时间。然而，在大多数 Web 站点上，可能不会注意到这一差别。只有当有一个高通信量的 Web 站点，且在页面上有很多控件时，使用 HTML 控件才会提供稍好一些的性能，并且与服务器控件相比，它在服务器上使用更少的内存。

在大多数情况下，人们更愿意使用服务器控件而不是与它们对应的 HTML 控件。因为服务器控件提供了更多的功能，在页面中更灵活，可以给用户带来更丰富的体验。而且有比较好设计时支持，因此值得选择。

如果十分确信不需要服务器控件提供的这些功能，则可以选择 HTML 控件。

接下来将快速地介绍 Toolbox 中的其他类别。

4.3.3 数据控件

数据控件是在 ASP.NET 2.0 中引入的，它提供了非常方便的方式来访问各种数据源，如数据库、XML 文件以及对象。使用数据控件，不需要像在 ASP.NET 的早期版本中那样编写很多代码才能访问数据源，只需把数据控件指向适当的数据源即可，ASP.NET 运行库会负责处理大部分难题。在第 13 章及以后章节中，可以看到关于这些控件的更多内容。

4.3.4 有效性验证控件

有效性验证控件可以用来快速创建具有有效性验证规则的 Web 窗体，防止用户输入无效数据。

例如，可以强制用户在必需的字段中输入值，并检查输入的数据是否符合特定格式，如是否为有效日期，或者是否为 1~10 之间的数字等。它们甚至允许编写自定义代码来创建不会被标准控件覆盖的有效性验证例程。有效性验证控件的出色之处是它们既可以在客户端也可以在服务器上执行，这样就能创建响应性好而且安全的 Web 应用程序。第 9 章将更深入地介绍这些控件。

4.3.5 导航控件

在 Toolbox 中的 Navigation 类别下的控件用来让用户找到他们在站点中浏览的路径。TreeView 控件表示数据的层次结构，并且可以用来显示站点的结构，从而可以轻松地访问站点中的所有页面。Menu 控件的功能与之类似，并且可以选择显示水平还是垂直折叠菜单。

SiteMapPath 控件在 Web 页面中创建一个“痕迹导航”，允许用户在站点的层次结构页面中轻松地找到浏览路径。

在第 7 章专门讨论 Web 站点中的导航问题时将介绍这些控件的运用。

4.3.6 登录控件

与数据和导航控件一样，登录控件也是在 ASP.NET 2.0 中引入的，并且在 ASP.NET 4 中仍有着强大的功能。有了登录控件，只需很少的工作量就可以构建安全的 Web 站点。用户需要注册和登录后才能访问 Web 站点的特定部分(甚至是整个 Web 站点)。此外，它们提供了一些工具，让用户可以修改他们的密码；或者如果他们忘了老密码，则可以请求一个新的密码，并允许根据登录状态和用户的角色显示不同的数据。第 16 章将介绍关于 ASP.NET 的安全功能与登录控件的更多详细信息。

4.3.7 Ajax 扩展

在 2005 年 11 月官方发布 ASP.NET 2.0 一年多后，Microsoft 发布了 ASP.NET 2.0 AJAX Extensions 1.0 作为 ASP.NET 2.0 的插件。有了这些扩展，可以创建无闪烁的 Web 应用程序，且不需要完整回发就能从客户端 JavaScript 中检索服务器上的数据。自从 Ajax 在 2005 年成为一种热门技术以来，Microsoft 一直致力于成为顶级 Ajax 的实现者。Ajax 扩展现在已经完全集成到了 VWD 2008 IDE 中，并在 VWD 2010 中升级为 AJAX 4。第 10 章将重点介绍 Ajax。

4.3.8 WebParts

ASP.NET WebParts 是一组控件，允许 Web 页面的终端用户修改 Web 站点的外观和行为。用户只需通过几个简单的动作，就能修改 Web 站点的整个外观。这些动作包括：重新排列内容、隐藏或显示部分 Web 页面，以及向页面中添加其他内容片段。ASP.NET WebParts 超出了本书的讨论范围，因为只是介绍它就需要一整本书。如果要了解关于 WebParts 的更多内容，可以参见由清华大学出版社引进并出版的《Web Parts 与自定义控件高级编程(ASP.NET 2.0 版)》一书(ISBN: 978-7-302-14182-27)。虽然这本书是针对 ASP.NET 2.0 的，但是书中的很多概念仍然适用于 ASP.NET 4。

4.3.9 动态数据

这种类别的控件用于动态数据 Web 站点。动态数据站点允许在数据库中快速创建用户界面来管理数据。本书不对这些控件作进一步讨论。

4.4 ASP.NET 状态引擎

在第 3 章中，我们创建了一个带有 TextBox 和 Button 控件的页面。在那个“试一试”练习中，已经在浏览器中运行了该页面，输入了一些文本，并单击了按钮。这个按钮会导致发向服务器的一个回发，当重新加载页面时，该文本仍然会出现在文本框中。使用本章介绍的 Wizard 控件基本上也能完成同样的操作，使用这个控件还可以维护文本框和下拉列表中的值。如果熟悉其他 Web 技术，如 ASP 或 PHP，那么就会知道该功能的强大之处。在那些语言中，常常需要编写很多代码才能实现这个功能。那么，在 ASP.NET 中为什么会实现这种功能？它是如何自动发生的呢？

文本框中的文本由 ASP.NET 状态引擎维护，这是一个完全集成在 ASP.NET 运行库中的功能。它启用控件来维护它们跨回发的状态，因此在页面的每个回发之后它们的值和设置仍然是可用的。

4.4.1 状态的定义及其重要性

要理解状态的含义，重要的是要意识到在设计时 HTTP(用来在 Web 浏览器中请求和服务页面的协议)是无状态的。它的意思是 Web 服务器不会跟踪从特定浏览器中发出的请求。就 Web 服务器而言，从浏览器中对服务器发出的页面请求和单击链接到其他页面的请求仍然是原来的意思。Web 服务器不会记忆以前请求的页面。

这样会产生一些有趣的问题。例如，我们看一个简单的登录到 Web 站点的登录页面，如 Web 邮件程序。从图 4-14 中可以看到一个示例登录框。

现在假设用一个正确的用户名和错误的密码登录。那么页面会通知此次登录失败，这时用户会希望已经自动填好了用户名，而不希望 Remember me next time 复选框也保持选中。以这种方式，用户就容易输入正确的密码并再次单击 Log In 按钮。这只是一个简单的示例，但如果控件能够维持它们自身的状态，这在很多情况下都会很有用。

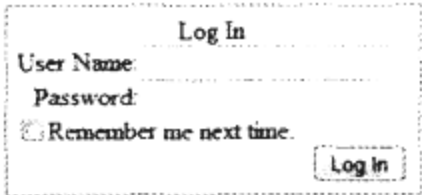


图 4-14

然而，默认情况下 Web 页面或控件不会自己完成这一切。由于每个请求都是独立的请求，因此服务器不会在回发之后再次填充文本框，而是简单地以它首次加载页面时的方式处理页面。在其他 Web 技术中，如传统 ASP 或 PHP，可以通过手动编写代码，在回发后预先填充控件来实现这一功能。幸运的是，ASP.NET 将这个功能集成到了 ASP.NET 功能集中，因此这样做就容易多了。

4.4.2 状态引擎的工作原理

ASP.NET 中的状态引擎可以存储很多控件的状态。它不仅能存储用户输入控件(如 TextBox 和 CheckBox)的状态，而且可以存储其他控件(如 Label，甚至是 Calendar)的状态。这个功能通过一个示例可以很好地演示。下面的“试一试”练习显示了如何使用能维持自身状态的控件创建一个页面。然后接下来的小节会解释 ASP.NET 是如何做到这些的。

试一试 分析 ASP.NET 状态引擎

- 在本练习中将向页面添加 Label、Button 和 Calendar 控件。这些控件用以演示 ASP.NET 的一些内部工作方式，包括回发和 ASP.NET 维护状态的方式。
- (1) 在 Demos 文件夹下面，创建一个新页面，命名为 State.aspx。一定要使用 Code Behind 模式，并且不要忘记从下拉列表中选择喜欢的编程语言。
 - (2) 将页面切换到 Design 视图中，在虚线<div>内单击以把焦点放入其中，然后从主菜单中选择 Table | Insert Table 命令，单击 OK 按钮插入一个两行两列的表。
 - (3) 在第一行的第一个单元格中，从 Toolbox 中拖动一个 Label 控件。在第二行的第一个单元格中，拖放一个 Calendar 控件。
 - (4) 注意，一旦把日历控件放入单元格中，Calendar 控件的 Smart Tasks 面板就会弹出，如图 4-15 所示。



图 4-15

- 在 Calendar 的例子中，这个面板上只有一个选项：Auto Format，它允许修改日历的外观。单击该链接，从预先定义的配色方案中选择一种，如 Simple，并单击 OK 按钮。
- (5) 然后，把 Button 控件拖到表的右列中的两个单元格中。
 - (6) 单击第一行中的 Button 控件，然后按下 F4 键打开 Properties 面板。将 Button 的 ID 设置为 SetDate，并将它的 Text 属性设置为 Set Date。ID 属性将一直位于属性列表末尾，或者，如果让属性列表按字母顺序排列的话，它就一直位于列表开头。
 - (7) 为第 2 个按钮重复上一步，不过将它命名为 PlainPostBack，并将它的 Text 属性设置为 Plain PostBack。此时，页面在 Design 视图中应如图 4-16 所示。



图 4-16

(8) 在 Design 视图中双击 Set Date 按钮，并在 VWD 插入的代码行之间的空行上添加下面突出显示的代码：

VB.NET

```
Protected Sub SetDate_Click(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles SetDate.Click
    Label1.Text = DateTime.Now.ToString()
End Sub
```

C#

```
protected void SetDate_Click(object sender, EventArgs e)
{
    Label1.Text = DateTime.Now.ToString();
}
```

(9) 按下 Ctrl+F5 组合键在浏览器中打开该页面。单击某一天以在日历上选择一个日期。注意，单击了日期后，页面可能会重新加载，这是由一个回发引起的。在本练习后面的“工作原理”部分会介绍更多这方面的知识。

(10) 单击几次 Set Date 按钮。页面会再次发送回服务器，并且每次单击该按钮时 Label 都会更新为今天的日期和时间。等待几秒钟，然后单击 Plain PostBack 按钮。这时又发生了一个回发，然后页面就会重新加载。现在看一下 Label 的文本。它仍然包含上次单击 Set Date 按钮时显示的日期和时间。再多单击 Plain PostBack 按钮几次，注意，Label 没有变化。

(11) 回到 VWD，在 Design 视图中打开 Label 控件的 Properties 面板。定位到 EnableViewState 属性，并通过从下拉列表中选择 False 或双击它的属性名或值，将它设置为 False。

(12) 在浏览器中再次打开页面，重复第(9)步和第(10)步，单击日历和按钮。这次，当单击 Plain PostBack 按钮时，将看到 Label 控件显示它的默认初始文本：Label。

工作原理

为了理解它的工作原理，需要了解几个重要元素。首先，再次在浏览器中打开页面，然后查看它的 HTML 源代码。可以通过在浏览器中右击页面，然后选择 View Source 或 View Page Source 菜单项打开它的 HTML 源代码。在靠近窗口上方的地方，可以看到如下的<form>元素。

```
<form name="form1" method="post" action="State.aspx" id="form1">
...
</form>
```

HTML <form>元素用于让用户从浏览器向服务器提交信息。用户可以用一些控件来输入信息，如文本框、下拉列表、复选框等。表单的提交方式有两种：POST(如前面的<form>元素所示)或 GET。在前一种情况下，所有来自表单的数据都被添加到请求的主体中，然后发送给服务器。在 GET 方法中，所有的数据都被附加到请求的实际地址后面。差异的复杂性目前还不是太重要；重要的是要了解<form>元素的用途：它用来封装一些表单控件，这些控件的值会被一起提交回服务器。

当单击 Button 这样的控件时，会导致向服务器发送一个回发。在这个回发期间，表单中的所有相关信息都会被提交回服务器，在服务器上可以用来重构页面。

默认情况下，所有的 ASP.NET Web 窗体总是使用 POST 方法向服务器发送数据。同样，默认情况下整个 ASP.NET 页面总是恰好就包含一个表单。由于这种情况如此普遍，在 VWD 中创建的一个新页面(或将在第 6 章介绍的母版页)已经包含了<form>元素，因而不必亲自添加它。最后，知道 ASP.NET Web 窗体在默认情况下总是提交回给它自己是很重要的。在其他 Web 环境中，如传统 ASP 和 PHP，通常会将页面的动作特性设置到另一个页面来处理用户提交的数据。然而，使用了 ASP.NET 页面后，即使显式地在代码中设置了动作特性，ASP.NET 运行库也会将它恢复为当前页面的名称。



注意：ASP.NET 支持一种叫做跨页面回发的功能，它允许从一个页面向另一个页面提交数据。要了解更多关于这个概念的信息，可在 MSDN 站点上搜索 Cross Page PostBacks 或参考清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C#和 VB.NET》一书(ISBN: 978-7-302-23524-8)。

接下来要介绍是隐藏的_VIEWSTATE 字段，可以在下面的代码段中突出显示的代码中看到：

```
<form name="form1" method="post" action="State.aspx" id="form1">
...
  <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
    value="/wEPDwULLTE5Njc4MzkzNDdkZI+LOWZMZpVv0hc7i/HFGMd008oc" />
</div>
```

虽然一开始出现的文本中除了一些随机字符外什么也没有，但是它实际上包含了有用的信息。为了保护存储在这个字段中的信息，以及为了缩小它的大小，ASP.NET 转换了上面的字符串中的页面状态。如果仔细查看字段值，就会发现控件 Label1 的一个带当前日期与时间的值。

当加载 ASP.NET 页面时，ASP.NET 运行库会用关于该页面的信息填充这个隐藏字段。例如，当单击 Set Date 按钮引起一个回发时，它会为 Label1 控件的 Text 属性添加值。类似地，它还包含 Calendar 的选中日期。当通过回发提交回该页面时，就会通过请求发送该隐藏字段_VIEWSTATE 中的值。然后，当 ASP.NET 在服务器上创建了新页面时，_VIEWSTATE 字段中的信息就会被读取并应用到页面中的控件上。通过这种方式，像 Label 这样的控件就能维持它的文本，甚至在将页面发送回服务器后也可以。

为了重申，下面扼要总结一下上面的“试一试”练习发生的过程：

- (1) 通过从 VWD 中打开页面来在浏览器中请求它。
- (2) 浏览器通过一个页面请求从服务器中得到请求的页面。
- (3) ASP.NET 运行库从磁盘中读取页面文件，处理它，并将最终产生的 HTML 发送给浏览器。在这一阶段，所有控件都被设置为它们在页面标记中定义的默认值。例如，Label 控件的 Text 属性被设置为 Label。
- (4) 当页面显示在浏览器中时，单击 Set Date 按钮。这会引发一个向服务器的回发。
- (5) 在服务器上，会重新构建页面，与第(3)步描述的首次加载页面的情况相似。在这一阶段，所有控件都包含它们的默认值。因此，Label1 控件的 Text 属性仍然是设置为 Label。在设置默认值后不久，运行库会用它在 View State 中找到的值覆盖控件的这些默认值。然而，由于这是第一个回发，而且 Label 控件的 Text 属性还没有改变，它的值并没有出现在 View State 中，因此 Text 属性只包含默认单词 Label。

(6) 当仍然在处理同一个请求时，ASP.NET 运行库会激活 `SetDate_Click` 中的代码。可以看到，这段代码会将 `Label` 控件的 `Text` 属性改为当前日期与时间。ASP.NET 运行库会看到这个变化，并将这个新值存储在 `View State` 中，因此它对于以后的回发仍然是可用的。

(7) 然后，单击了 `PlainPostBack` 按钮。与其他按钮一样，这会引发一个回发。该页面会被再次构建，并设置所有默认值。这也意味着 `Label1` 控件的 `Text` 属性只包含单词 `Label`。然而，之后不久，在同一个处理周期中，ASP.NET 运行库会处理 `View State`，恢复它在隐藏的 `_VIEWSTATE` 字段中发现的控件值。在本例中，它发现 `Text` 属性中是当前日期和时间，并再次将它指派给 `Label` 控件。由于 `PlainPostBack` 按钮不会再修改 `Label` 的 `Text` 属性，因此 `Text` 属性不会变化：它包含前一个回发中的日期和时间。最后，整个页面会被发送给浏览器，在那里 `label` 会正确地显示它的前一个值。

(8) 最后，将 `EnableViewState` 设置为 `False`，从而关闭了 `Label` 控件的 `View State`。关掉这个设置后，ASP.NET 运行库不会再跟踪 `Label` 控件。因此，当单击 `PlainPostBack` 按钮时，ASP.NET 运行库无法在 `View State` 中找到该标签的任何信息，这样最终会导致这个标签显示它自己的默认文本，即单词 `Label`。

4.4.3 并非所有控件都依赖于 View State

应当了解，并不是所有控件都一直依赖于 `View State`。有很多控件能维持它们自己的某些状态。这些控件包括 `TextBox`、`CheckBox`、`RadioButton` 和 `DropDownList`。它们能维持它们的值，这是因为它们在浏览器中被呈现为标准的 HTML 表单控件。例如，`TextBox` 服务器控件在浏览器中看起来如下所示：

```
<input name="TextBox1" type="text" value="Initial Text" id="TextBox1" />
```

当发送回一个带有这样的 `TextBox` 的页面时，浏览器也会将控件的值发送回服务器。然后 ASP.NET 运行库就能再次用这个值来预先填写文本框，而不需要从 `View State` 中获取值。显然，这也比将值存储在 `View State` 中更有效。如果存储在 `View State` 中，值就会被发送到服务器中两次：一次在文本框中，另一次在 `View State` 中。尤其是当值比较大时，这样会大大增加页面的大小，因此增加了加载的时间。但对于一些功能，如跟踪客户端的变化时，这些控件仍需要将值存储到 `View State` 中，并根据需要来做这些工作。

4.4.4 关于 View State 和性能的一个注意点

由于 `View State` 引擎给页面添加了相当数量的信息，因此，在不需要时最好关闭它。这样就能最小化隐藏的 `_VIEWSTATE` 字段的大小，这意味着页面会变得较小，因而在浏览器中加载时就会较快。关闭 `View State` 很容易，可以在三个地方做到：

- 在 Web 站点级别 可以在根站点的 `web.config` 文件中通过修改 `<system.web>` 下面的 `<pages>` 元素，将 `enableViewState` 特性设置为 `false` 来完成。

```
<pages enableViewState="false">
...
</pages>
```

在站点级别关闭 `View State` 会出现问题，因为之后无法为特定的控件打开这个功能。幸运的是，ASP.NET 4 提供了一个新的属性 `ViewStateMode`，它提供了关于 `View State` 如何使用的

更多控制。

- 在页面级别 在每个页面上方可以发现页面指令，即一系列告诉 ASP.NET 运行库页面应采用什么行为的指示。在页面指令中，可以将 `EnableViewState` 设置为 `False`：

```
<%@ Page Language="VB" AutoEventWireup="False" CodeFile="State.aspx.vb"
    Inherits="Demos_State" EnableViewState="False" %>
```

对于确信根本不需要 `View State` 的页面来说，这种方法是有用的。

- 在控件级别 各个 ASP.NET 服务器控件允许分别设置各个 `EnableViewState`，这样可以选择关闭某些控件，而使其他控件保持打开。

一旦在一个更高级别(`web.config` 或页面级别)上关闭了 `View State`，就不能再在一个低层级别(页面或特定控件级别)打开这个功能。然而，使用新的 `ViewStateMode` 属性仍能完成如下工作：

- 禁止在 `web.config` 文件中关闭 `View State`。
- 在页面级别，将 `EnableViewState` 设置为 `True`，将 `ViewStateMode` 设置为 `Disabled`，如下所示：

```
<%@ Page Language="C#" ...EnableViewState="True" ViewStateMode="Disabled" %>
```

运行上边的代码可以关闭页面中所有控件的 `View State`，除了那些再次明确地将 `ViewStateMode` 属性设置为 `Enabled` 的控件以外。

- 如果想让控件支持 `View State`，可使用如下代码将控件的 `ViewStateMode` 设置为 `Enabled`：

```
<asp:Label ID="Label1" runat="server" Text="Label" ViewStateMode="Enabled" />
```

如果想在演示页面中看到效果，则要修改 `State.aspx` 的页面指令，就像在前面的例子中一样将 `EnableViewState` 设置为 `True`、将 `ViewStateMode` 设置为 `Disabled`。然后在页面中创建第二个 `Label` 控件，并将第一个 `Label` 控件的 `ViewStateMode` 设置为 `Enabled`：

```
<asp:Label ID="Label1" runat="server" Text="Label" ViewStateMode="Enabled" />
<asp:Label ID="Label2" runat="server" Text="Label" />
```

在页面的后台代码中，将当天的日期和时间也赋予第二个标签控件：

VB.NET

```
Label1.Text = DateTime.Now.ToString()
Label2.Text = DateTime.Now.ToString()
```

C#

```
Label1.Text = DateTime.Now.ToString();
Label2.Text = DateTime.Now.ToString();
```

最后，运行上一个“试一试”练习中的第(9)和(10)步。注意，第一个 `Label` 控件会维持它的文本内容，而第二个控件则设置为默认文本 `Label`。

4.5 使用控件的实用提示

下面的列表指出了关于使用控件的一些实用提示：

- 花一些时间在标准类别中试试各种不同的控件。虽然很多控件会在本书中多次用到和讨论到，但是最好知道应如何使用它们以及它们是如何操作的。如果现在用几个示例页面练习使用它们，在以后的章节中再出现这些控件时就会抢先一步理解它们。
- 重点考虑一下如何关闭有些不需要 View State 功能的控件的 View State。在很多情况下，几乎看不出它们的区别，但是特别是遇到在第 13 章和以后的其他章节中讨论的数据驱动控件时，禁用 View State 可以大大缩小 Web 页面的大小，使得加载时间缩短，从而使用户感觉更好。
- 在用多个控件设计一个复杂 Web 窗体来接受用户输入时，写出必需的功能。在开始编码前，思考一下应用程序的(技术)设计，这样更容易创建一致且经过严密思考的用户界面。在以后发现弄错了再对页面作大量修改会比一开始就纠正过来要花多得多的时间。
- 实践 View State 机制更好地了解它的工作原理。创建几个如在上一个“试一试”练习中创建的页面。然后，关闭页面或控件级别的 View State，并查看页面的行为。注意一些控件，如 TextBox，即使关闭了 View State，这些控件也仍然能维持它们的值。

4.6 本章小结

本章详细介绍了大量的 ASP.NET 服务器控件。由于这些控件如此重要，而且在每个 ASP.NET 应用程序中都会用到。因此，了解 Toolbox 中有哪些控件可用、它们各自的用途、它们的工作原理以及它们如何维持自身状态非常关键。

ASP.NET 中的一个最大的发明是状态引擎，它允许控件跨回发维持它们的状态。状态引擎确实可以节省时间，不需要在每个 Web 页面中为了重用这个行为而编写很多冗长且烦琐的代码。然而，如果关闭 View State 意味着更高的效率，就把它关闭。

本章也介绍了一些用 Visual Basic 和 C#编写的简单的服务器端代码。第 5 章将更深入地介绍 ASP.NET 页面中的编程。其中将介绍一种编程语言，了解它含有哪些要素，以及如何自己编写 ASP.NET 页面使用的代码。而且，最好的是，示例都是用 Visual Basic 和 C#完成的，因此可以使用自己喜欢的语言。

4.7 练习

1. 解释使服务器控件维持自身状态的机制。
 2. ASP.NET 运行库如何在回发之间跟踪控件状态？
 3. 指出<asp:DropDownList>和<asp:ListBox>之间的区别。
 4. 在改变浏览器中 CheckBox 的选中状态时，需要使用什么属性来引发一个向服务器的回发？
 5. 很多服务器控件都有一组共同的属性，能影响它们在运行时的外观。说出可以改变元素样式(如颜色、边框和大小)的 3 个属性。
 6. 相比于设置单个控件属性，如 BackColor 与 ForeColor，设置一个与 CSS 相关的属性会更好。这个属性的名称是什么？它带来了什么好处？
- 练习的答案见附录 A。

本章要点回顾

_VIEWSTATE	用于把状态从服务器传给客户端或者从客户端传给服务器的隐藏表单字段
容器控件	作为容器的服务器控件，将其他内容和控件包含在内
Events 选项卡	Properties 面板中用于设置控件的事件(如按钮的单击事件)处理程序的部分
列表控件	将选项以列表形式呈现给用户的服务器控件。此类控件包括 DropDownList、CheckBoxList 等
Post 和 Get 方法	从客户端向服务器提交数据的不同方法。Post 方法用于将数据添加到请求的主体中，而 Get 方法则是将数据附加到请求页面的地址上
回发	将表单数据从客户端浏览器送回服务器的过程
服务器控件	ASP.NET 的主要部分，用于在浏览器中构建 Web 页面的用户界面
Smart Tasks 面板	某些控件的活动面板，有助于完成一些常见任务
View State	让 ASP.NET 控件在客户端维持状态的机制

本章要点

- 如何在编程环境中使用数据类型、变量、对象和集合
- 在代码中制定决策的各种方式
- 创建重用性功能模块的可用选项
- 编写组织良好且文档化的代码的各种方式
- 什么是面向对象，以及如何在应用程序中使用面向对象编程

前面 4 章已经创建了一些 Web 窗体，其中包含的基本都是 ASP.NET 服务器控件和纯 HTML。只有少数几个示例包含用 C#或 Visual Basic(VB.NET)写的实际编程代码，而且其中大多数代码相当简单。然而，并不是所有的页面都一直这样简单。虽然与.NET Framework 过去的 1.x 系列或其他技术 (如传统 ASP 或 PHP 技术) 相比，不少可以自由支配的智能服务器控件能够使需要编写的代码量减少，但是能够阅读、理解和写代码是 Web 开发工具的一项基本能力。

本章将介绍 Web 应用程序的编程基础以及 Web 应用程序编程之外的一些内容。与本书其他章节的大部分示例一样，本章整章都采用了 VB.NET 和 C#语言的示例。对于本章介绍的每个概念或理论部分，都能同时看到 VB.NET 和 C#语言的示例。选用哪种语言完全由您决定。



注意：为了从本章得到最大收获，建议实际尝试一下本章出现的大部分代码。大多数示例都可以用一个简单的 ASPX 页面来测试。拖动一个 Label 和一个 Button 控件到页面上，在 Design 视图中双击 Button。然后在 VWD 添加的代码块的开放代码行上输入示例代码后按下 Ctrl+F5 键。页面完成加载后，单击按钮以执行代码。有些示例会调用不能正常运行的虚构代码。使用它们只是为了说明所讨论的主题。

5.1 编程简介

要开始编程，关键是要了解一套各类语言和应用程序的程序员都使用的术语。本章接下来会介绍大量的术语和概念。大部分术语都附带了代码示例，因此可以看出它们在实际代码中的运用。

注意，本章并不是对编程的完整介绍。这里并没有覆盖编程语言的所有细节，而是主要关注成功构建日常 Web 站点所需要了解的关键概念。熟悉了这些概念后，就较容易通过学习您喜欢的语言的更多功能来加深对编程的了解。



注意：如果对用 VB.NET 或 C#编程比较有兴趣，可以参考 Wrox 出版社出版的 *Beginning Microsoft Visual Basic 2010*(ISBN: 978-0-470-50222-8)和 *Beginning Microsoft Visual C# 2010*(ISBN: 978-0-470-50226-6)。

在本章及接下来的几章中，编写的代码将被添加到 Web 页面的 Code Behind 中，或者添加到位于 App_Code 文件夹中的独立类文件中。当 ASP.NET 运行库处理包含代码的页面请求时，它会先编译在页面、Code Behind 文件或类文件中发现的任何代码。代码编译完后，便会从人类可读的编程语言(如 C#或 VB.NET)转换为 .NET Framework 运行库能够理解和执行的中间语言(IL)。ASP.NET Web 站点编译过程的结果是在系统的临时文件夹中生成一个或多个扩展名为 DLL 的文件。只有在修改页面后第一次请求时才会发生这种编译过程。以后对同一个页面的请求就会重用同一个 DLL 文件。幸运的是，在 ASP.NET Web 站点中，编译是在后台发生的，因此通常不需要关心它。

要了解编程，首先需要学习的概念是数据类型与变量，因为它们是所有编程语言的构建基块。



注意：ASP.NET 使用的 .NET Framework 很大，包含几千种类型，这些类型又包含成千上万个方法、属性等。显然，您不可能记住框架中的所有类型，因此需要很好地利用诸如 IntelliSense 和在线帮助这样的资源。在 MSDN 站点(<http://msdn.microsoft.com/en-us/library/>)内导航有时会是一个令人畏惧的工作。但是，我发现搜索一些信息(如 typeName type .NET MSDN)通常能够带来所需要的东西。因此，如果希望学习有关 String 类的更多知识，可以在自己常用的搜索引擎中输入 string class .NET MSDN。十之八九搜索到的第一个结果便是一个到 MSDN Web 站点上相关页面的链接，在该页面中您可以学习关于类的更多知识，诸如它在何处被定义、设置以及如何使用该类。

5.2 数据类型与变量

初次接触一些编程环境中使用的数据时，可能不会意识到每块数据都是有数据类型的。例如，您会以为计算机会以存储今天的日期和数字 26 的方式来存储短语“Hello World”。然而，为了有效地使用数据，很多编程语言都有不同的数据类型，其中每种数据类型都被约束为特定的信息类型。.NET Framework 本身就包含一个很长的数据类型列表，允许使用数值(如 Integer、Short 和

Double)、文本字符串(Char 和 String)、日期(DateTime)、true/false(Boolean)等数据类型。本节后面将介绍最常用的数据类型。

这里每个数据的主要类型都是一个特殊的数据类型。要使用某种数据，可以将它存储在事先用必需的数据类型声明的变量中。在 VB.NET 中，使用 Dim myVariable As DataType 声明变量；而在 C#中，用 DataType myVariable 声明变量。下面的示例说明了如何声明两个变量：存储一个数的 Integer (在 C#中是 int)，和存储一串文本的 String(在 C#中是 string)。

VB.NET

```
' Declare a variable of type Integer to hold medium sized whole numbers.
Dim distanceInMiles As Integer

' Declare a variable to hold some text like a first name.
Dim firstName As String
```

C#

```
// Declare a variable of type int to hold medium sized whole numbers.
int distanceInMiles;

// Declare a variable to hold some text like a first name.
string firstName;
```

这两个代码示例中还包含注释。在 VB.NET 中，注释前缀是一个单引号(')；而在 C#中，注释前缀是两个斜线(//)。本章后面将介绍关于代码注释的更多内容。

声明一个变量后，就能赋予它一个值。可以直接向变量指定数值及 Boolean 等类型。要向变量指定 String 类型，需要用双引号把它括起来：

VB.NET

```
Dim distanceInMiles As Integer
distanceInMiles = 437

Dim firstName As String
firstName = "Imar"
```

C#

```
int distanceInMiles;
distanceInMiles = 437;

string firstName;
firstName = "Imar";
```

除了分开的声明和赋值外，也可以在同一步中完成声明并直接向新创建的变量赋一个值：

VB.NET

```
Dim distanceInMiles As Integer = 437
```

C#

```
int distanceInMiles = 437;
```

虽然变量名可以是任何喜欢的名称，但还是建议为各个变量起一个能描述其用途的有意义的名称。例如，将存储第一个名称的字符串命名为 `firstName`，而存储某人年龄的变量则简单地命名为 `age`。为了帮助以后在代码中查看变量的类型，当将鼠标悬停在代码编辑器中的变量上时，Visual Studio 中的 VWD 和所有其他产品都会显示一个有用的工具提示，这样就非常容易看出变量的类型。图 5-1 表明，C#示例中的 `distanceInMiles` 变量的类型是 `int`。

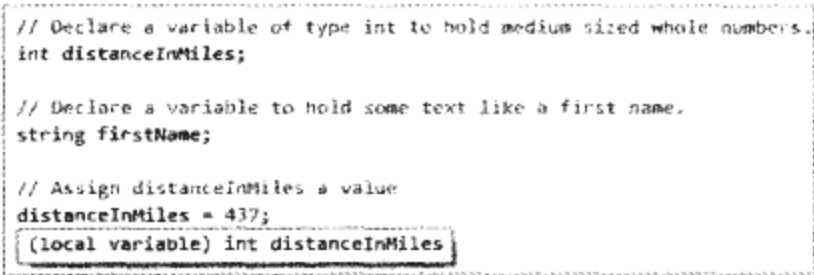


图 5-1

建议不要用几个字母作为变量的前缀来表示类型。例如，存储某人名字的 `String` 可以写为 `firstName`，而不是 `sFirstName`。这种表示法称为匈牙利表示法，基本上已经过时了。IDE 喜欢使用 VWD，它有智能 `IntelliSense` 和其他编程工具，所以确实不需要这种表示法了。不使用匈牙利表示法，代码更容易阅读(`age` 比 `iAge` 更可读)和维护，因为可以修改变量的类型，而不需要在使用它的任何地方重命名。

Microsoft .NET(因此是 ASP.NET 环境)支持很多不同的编程语言，包括 VB.NET 和 C#及其他语言。这些语言之间都能相互通信。例如，可以使用 C#写一些代码，用 Visual C# Express Edition 将它编译为 `.dll` 文件(这是一种包含可以在其他.NET 应用程序中重用的代码的文件)，然后在用 VB.NET 为主要语言的 Web 应用程序中使用它。由于这种互用性，因此有必要建立某种允许所有.NET 编程语言互相理解的系统。这个系统称为 CTS(Common Type System，通用类型系统)。CTS 定义了所有与 CTS 兼容的语言都可使用的数据类型。然后，各种语言可以自由定义一套基本类型(`primitive type`)，它们其实是.NET Framework 中复杂类型描述的简写或别名。因此，即使 CTS 定义了一种称为 `System.Int32` 的类型，像 C#这样的语言仍然可以自由地将这种类型定义为 `int`，以便开发人员使用起来更方便。

表 5-1 列出了.NET Framework 中最常用的 CTS 类型及它们的 C#与 VB.NET 别名。该表还列出了变量的范围和用途。

表 5-1

.NET	C#	VB.NET	说 明
System.Byte	byte	Byte	用来存储 0~255 的较小正整数。当没有显式指定值时，默认值为 0
System.Int16	short	Short	可以存储 - 32768~32767 之间的整数。默认值为 0
System.Int32	int	Integer	能存储从 - 2147483 648~2147483 647 之间的整数。默认值为 0

(续表)

.NET	C#	VB.NET	说 明
System.Int64	long	Long	容纳 - 9223 372 036 854 775 808 ~9223 372 036 854 775 807 之间的大数值。默认值为 0
System.Single	float	Single	能存储 - 3.402 823 5E+38~3.402 823 5E+38 之间带小数位的大数值。默认值为 0.0
System.Double	double	Double	能容纳大分数。当遇到分数时，它没有 Decimal 精确，但是当不需要极其精确时，应选择 Double 而不要选择 Decimal，因为 Double 要快一点。默认值为 0.0
System.Decimal	decimal	Decimal	存储高精度的极大分数。默认值为 0。该数据类型常用来存储货币值
System.Boolean	bool	Boolean	用来容纳简单的布尔值：VB 中的 True，C#中的 true，或者 VB 中的 False，C#中的 false。默认值为 False
System.DateTime	n/a	Date	VB.NET 为存储日期和时间值的数据类型 System.DateTime 定义别名来存储日期和时间值。C#没有为这种类型定义别名。默认值为 1/1/0001:12:00 am
System.Char	char	Char	容纳一个字符串。默认值为 Nothing(在 C#中是 null)
System.String	string	String	能容纳长达 20 亿个字符的文本。默认值为 Nothing(在 C#中是 null)
System.SByte	sbyte	SByte	用来存储 - 128~127 的小整数。默认值为 0
System.UInt16	ushort	UShort	类似于 System.Int16，但是此数据类型只能存储 0~65 535 之间的无符号整数。默认值为 0。其他前缀为 U 的数据类型也都是无符号类型
System.UInt32	uint	UInteger	能存储从 0~4294967295 之间的整数。默认值为 0
System.UInt64	ulong	ULong	能存储从 0~18446744073 709 551 615 之间的整数。默认值为 0
System.Object	object	Object	.NET 中所有数据类型的父类型，包括 CTS 类型和自己定义的类型。各种数据类型也是对象，本章后面将会解释。默认值为 Nothing(在 C#中是 null)

标准的.NET 类型都附带有前缀 System，该前缀后面带有一个句点。这个 System 部分是该数据类型的名称空间。本章后面将会介绍什么是名称空间以及它们的用途。

有时需要将数据从一种类型转换为另一种类型。例如，可能会从某个来源得到 Integer 类型的数据，需要将它作为 Double 类型的数据使用。有几种方式可以做到这一点。

5.2.1 转换数据类型

转换数据类型最常用的方式是转换为 String。Web 应用程序多处需要使用字符串类型。例如，从 TextBox 返回的 Text 是一个 String，DropDownList 的 SelectedValue 也是字符串类型。要将任何

Object 转换为 String，可以直接调用它的 ToString()方法。.NET 中的每个对象都支持这种方法，尽管各对象的具体行为可能不尽相同。就目前而言，重要的是要了解 ToString()是关于对象的一种方法(或者是一种操作)，这些对象包括 String 或 Double，甚至是父对象本身。本章后面讨论面向对象编程时会介绍关于方法和对象的更多知识。

使用 ToString()方法并不难，比如下面这个演示在 Label 控件上输出今天的日期与时间的示例。


VB.NET

```
Label1.Text = System.DateTime.Now.ToString()
```

C#

```
Label1.Text = System.DateTime.Now.ToString();
```

转换数据类型的另一种方式是使用 Convert 类。



注意：在.NET 中，类是一个重要的概念，在本章后面关于类的部分中再对它详细讨论。目前重要的是理解类是.NET 中所使用对象的一个蓝图。可以创建自己的类，也可以使用.NET Framework 中的许多标准类。

Convert 类中包含将多种数据类型转换为另一种类型的功能。下面显示了一个将包含看起来像 Boolean 类型的值的 String 转换为真正的 Boolean 类型的简单示例：

VB.NET

```
Dim myBoolean1 As Boolean = Convert.ToBoolean("True")           ' Results in True
Dim myBoolean2 As Boolean = Convert.ToBoolean("False")          ' Results in False
```

C#

```
bool myBoolean1 = Convert.ToBoolean("True");                     // Results in true
bool myBoolean2 = Convert.ToBoolean("False");                     // Results in false
```

除了 ToBoolean 方法之外，Convert 还提供了许多其他的转换方法，包括.ToInt32(用于数值转换)、ToDateTime(用于日期转换)以及 ToString。

还有一种转换数据类型的方式是强制类型转换(casting)。强制类型转换实际上是强制一种类型变为另一种类型，它不同于将数据类型的底层值转变为新值的转换(converting)。

强制类型转换只适用于兼容的数据类型。例如，不能将 DateTime 强制转换为 Integer。然而，可以强制转换相似的类型，如将 Double 转换为 Integer，或者将 String 转换为 Object。但是倒过来转换则不一定总能成立。前面我们已经说过，.NET Framework 中的每种数据类型都是基于 Object 数据类型的。例如，这意味着，String 类型也是 Object 类型。然而，Object 类型并不也都是 String 类型。要记住下面这种情况：试图将一种类型强制转换成另一种类型时得到一个编译错误。本书后面几章会介绍如何强制转换能相互兼容的数据类型。

要使用 VB.NET 将一种类型强制转换为另一种类型，有几种方法可以实现。第一种方法是，可以使用 CType 和 DirectCast。CType 稍灵活一些，它允许在看起来相似的两个对象之间强制转换。

而 `DirectCast` 只允许在兼容类型之间转换，不过它执行得稍快一些。下面的 VB.NET 示例显示了其工作原理。

```
Dim o1 As Object = 1
Dim i1 As Integer = DirectCast(o1, Integer) ' Works, because o1 is an Integer
Dim i2 As Integer = CType(o1, Integer)      ' Works, because o1 is an Integer

Dim o2 As Double = 1
Dim i3 As Integer = DirectCast(o2, Integer) ' Fails, because o2 is not an
                                             ' Integer
Dim i4 As Integer = CType(o2, Integer)      ' Works, because o2 looks like an
                                             ' Integer
```

这个示例的第一部分声明了一个对象 `o1`，并赋予它一个 `Integer` 值 `1`。虽然 `o1` 对外显示的是 `Object` 类型，但实际上它的底层值仍然是 `Integer` 类型。当调用 `DirectCast` 时，会成功进行强制转换，因为 `o1` 实际上是 `Integer` 型的。

在第二个示例中，`o2` 被声明为 `Double` 型，这种数据类型看起来有点像 `Integer`，不过并不真的是 `Integer` 类型。因此调用 `DirectCast` 就会失败，因为 `Double` 类型不能被强制转换为 `Integer`；而另一方面，使用 `CType` 却能很好地转换，因为变量 `o2` 的底层值看起来像 `Integer`，因此被转换成了 `Integer` 类型。

在 VB.NET 中进行强制转换的第三种方法是使用关键字 `TryCast`，它与另外两种方法有些类似。当不能正确强制转换某个对象时，`TryCast` 就会返回 `Nothing`，而 `DirectCast` 和 `CType` 会使代码崩溃。

C#中有两种强制类型转换的方法。最常见的方法是把数据类型放在要转换的变量名前面的括号中：

```
object o1 = 1;
int i1 = (int)o1; // Works

double o2 = 1;
int i2 = (int)o2; // Works
```

此外，还可以使用关键字 `as`，其工作方式类似于 VB.NET 中的 `TryCast`。如果强制转换不成功，代码也不会崩溃。下面的代码示例说明无法将 `Integer` 转换为 `ArrayList`(本章后面会遇到这种类型)。变量 `myList` 不会崩溃，它只不过会包含空值来表示强制转换操作没有成功。

```
object o1 = 1;
ArrayList myList = o1 as ArrayList; // Doesn't cast, but doesn't crash either.
```

在本书其余章节中还会更深入地讨论转换和强制类型转换。

5.2.2 使用数组和集合

到目前为止所提到的数据类型都是相当简单而独立的对象。例如，将值 `True` 或 `False` 存储在 `Boolean` 类型中，而将 `123` 这样的数字存储为 `Integer` 型。但是，如果需要存储很多整数，该怎么办呢？如果要存储形状复杂(如多边形)的点，可能需要存储很多整数。或者，如果需要将应用程序支持的所有角色都存储在单个变量中，以便将它们显示在 Web 页面上的 `Management` 部分。这时就要用到数组和集合。

1. 定义和使用数组

数组相当于相同类型的事物的大包或者一个列表。在声明时，就在一个数组中定义了事物的数据类型。数组中的各项由序号(所谓的索引)标识，序号从 0 开始，所以数组是基于 0 的数组。在 VB.NET 中声明和访问数组时会使用圆括号，而在 C#中使用方括号。定义了数组并填充了它的元素后，可以通过基于 0 的元素索引(0、1、2 等)访问数组中的元素。

下面的代码段定义了名为 roles 的数组，其中可以同时存储两个角色。

VB.NET

```
Dim roles(1) As String
```

C#

```
string[] roles = new string[2];
```

看出 VB.NET 与 C#示例之间的区别了吗？那并不是打印错误。在 VB.NET 中通过指定上边界来定义数组的大小。上边界是数组中能访问的最后一个元素。由于数组是基于 0 的(也就是说，数组中的第一个数组元素索引为 0)，因此那意味着需要容纳两个数组元素的空间，上边界是 1，含有的数组元素为 0 和 1。

另一方面，在 C#中，不是定义上边界，而是定义数组的大小。因此在 C#中，只是将大小指定为 2 来得到一个有两个元素的数组。

此外，C#要求使用关键字 new 实例化一个新数组。VB.NET 会自动完成这一步，如果像在 C#示例中那样添加 New 关键字，则会抛出一个错误。在本书后面还会看到 new(在 VB.NET 中是 New)关键字的使用。

向数组中输入角色名的语法如下：

VB.NET

```
roles(0) = "Administrators"  
roles(1) = "ContentManagers"
```

C#

```
roles[0] = "Administrators";  
roles[1] = "ContentManagers";
```

与数组的声明一样，VB.NET 中使用圆括号，C#中使用方括号来指出数组中的元素。注意 roles[0] 是指数组中的第一个元素，roles[1]是指第二个元素。

数组在设计时规定了固定大小。因此，对于前面的定义两个元素空间的数组示例，使用下面的代码将抛出一个错误：

VB.NET

```
roles(2) = "Members" ' Throws an error
```

C#

```
roles[2] = "Members"; // Throws an error
```

此代码试图向只有两个元素空间的数组中挤进第三个角色。显然这并不合适，因此会抛出一个错误“Index was outside the bounds of the array”(索引超出数组上边界)。但是，如果在以后的阶段，在运行时需要在数组中创建更多空间时，该怎么办呢？在 VB.NET 中这个问题相当容易解决。可以使用 ReDim 语句：

```
ReDim Preserve roles(2)
roles(2) = "Members" ' Works fine now
```

这行代码将数组重定义为新的大小：上边界为 2，因此创建了 3 个元素的空间。要让当前数组中的项保持不动，必须有 Preserve 关键字。如果没有这个关键字，重定大小后的数组就会是空的组。

在 C#中，需要稍微多做一些工作。在该语言中，需要创建一个具有所需大小的新数组，然后将旧数组中的元素复制到新数组中。接着就可以将旧变量指向新变量，并添加元素：

```
string[] tempRoles = new string[3]; // Create new array with required size
Array.Copy(roles, tempRoles, roles.Length); // Use Copy to copy the elements
// from the old to the new array
roles = tempRoles; // Assign the new array to the
// old variable

roles[2] = "Members"; // Works fine now
```

不需要创建新数组和复制元素，也可以通过一个称为“泛型”(将在本章后面介绍)的概念使用 Resize 来重定数组大小。下面是使用 Resize 的一些代码，效果与前面的代码段相同：

```
Array.Resize<string>(ref roles, 3); // Resize the array so it can
// hold three elements

roles[2] = "Members"; // Works fine now
```

现在不用担心看不懂这种奇怪的泛型代码；不会经常用到它，因为 .NET Framework 提供了固定数组大小的备用方案。

当开始用到数组时，会发现它们在运行时运行得很快，不过功能上欠缺了一些。例如，不太容易向数组中添加新元素或者从数组中删除现有元素。幸运的是，.NET Framework 提供了一组有用的集合，给出了所需的这些功能。

2. 定义与使用集合

集合(collection)类似于数组，它们都允许在一个变量中存储多个对象。集合的工作也类似于一个包：可以直接将若干项拖到包中，它就会容纳它们。集合的不同之处在于它们允许在包中使用数据的方式。集合不是简单地通过索引访问各个项，大多数集合会提供一个 Add 方法向集合中添加项。类似地，它们有 Remove 和 Clear 方法可以从集合中删除一个或所有项。与数组一样，它们允许迭代(iterate)或循环，遍历项以访问集合中的项。

当在 .NET Framework 1.0 中首次引入集合时，ArrayList 和 Hashtable 很快就开始流行，因为它们非常容易使用。ArrayList 允许添加任意对象，然后按添加的顺序存储，而 Hashtable 允许存储由自

定义键引用的对象。这些集合优于数组的地方主要在于它们可以根据需要增长。与前面的示例中需要重定数组大小才能为第三个角色增加空间不同的是，ArrayList 会根据需要动态增长。下面的示例说明了它是如何工作的：

VB.NET

```
Dim roles As New ArrayList()           ' Create a new ArrayList. You don't need
                                         ' to set its size explicitly

roles.Add("Administrators")             ' Add the first role
roles.Add("ContentManagers")           ' Add the second role
roles.Add("Members")                   ' Keep adding roles and the ArrayList
                                         ' grows as necessary
```

C#

```
ArrayList roles = new ArrayList(); // Create a new ArrayList. You don't need
                                     // to set its size explicitly

roles.Add("Administrators");           // Add the first role
roles.Add("ContentManagers");          // Add the second role
roles.Add("Members");                  // Keep adding roles and the ArrayList
                                     // grows as necessary
```

因为这段代码现在调用一个方法(Add)，而不是向数组中的预定义索引赋予一个项，所以在 VB.NET 和 C#中都是用圆括号()。本章后面将讨论这个方法的使用。

虽然集合解决了数组存在的问题，但是它们也带来了自身的问题。ArrayList 最大的缺陷是它不是强类型的。这意味着可以使用 Add 方法向列表中添加任何对象。这表示 ArrayList 能同时容纳不同类型的对象。一开始这可能不是大问题，但是一旦开始使用包含多个对象类型的 ArrayList 时，很快就会明白这为什么会成为一个问题。仍然以角色示例为例。有了数组和 ArrayList 版本，代码只需简单地添加几个包含角色名称的字符串，然后就能用这 3 个字符串构建 Web 窗体中的下拉列表，使用户可以从中选择一个角色。到目前为止这还是不错的。但是如果列表中的一个选项不是字符串的话，怎么办？如果另一名开发人员不小心写了一些向 ArrayList 添加 DropDownList 控件的代码，会出现什么情况呢？由于 ArrayList 能够接受所有对象，所以它不会提示什么。然而，如果希望接受一个 String，却得到一个 DropDownList 控件，代码就会崩溃。

在.NET 2.0 中，Microsoft 引入了一个概念，称为泛型(generic)。.NET 4 中仍然强调了泛型，它有助于克服 ArrayList 这样的弱类型集合带来的问题。

3. 泛型

由于.NET 2.0 中引入了泛型，因此泛型会在.NET Framework 中的很多地方出现。虽然它们通常用于使用集合的情况，但是泛型的使用并不仅限于集合，也可以将它们用于简单类型的对象。

泛型之于代码就相当于 Microsoft Word 模板之于字处理。它们可以用来写代码模板，用于不同类型的不同情况中。有了泛型，就能定义不显式指定类型的泛型代码模板。只有在使用代码的时候才会定义类型。这样做的好处是可以对多种数据类型反复重用同样的模板，而不需要重新输入和维护代码的多个版本。除了在自己的代码定义中使用泛型外，可以发现.NET Framework 中有很多支持

泛型的对象和集合供代码使用。

为了了解如何使用泛型，看看下面的示例。它就是在前面使用 `ArrayList` 时见到的同样代码，不过这次约束了列表的类型，只能接收字符串：

VB.NET

```
Dim roles As New List(Of String)

roles.Add("Administrators")
roles.Add("ContentManagers")
roles.Add("Members")
```

C#

```
List<string> roles = new List<string>();

roles.Add("Administrators");
roles.Add("ContentManagers");
roles.Add("Members");
```

要使角色列表类型安全(`type safe`)不需要修改太多代码。然而，在 VB.NET 中定义了 `List(Of String)`(C#中定义了 `List<string>`)，新列表现在被设置为仅允许通过它的 `Add` 方法添加字符串。它会顺利编译：

```
roles.Add("Administrators");
```

下面的代码编译会失败，因为 `33` 不是 `String` 类型：

```
roles.Add(33);
```

与字符串的泛型列表类似，也可以创建容纳其他类型的列表。例如：

VB.NET

```
Dim intList As New List(Of Integer)           ' Can hold Integers only
Dim boolList As New List(Of Boolean)          ' Can hold Booleans only
Dim buttonList As New List (Of Button)        ' Can hold Button controls only
```

C#

```
List<int> intList = new List<int>();           // Can hold ints only
List<bool> boolList = new List<bool>();        // Can hold bools only
List<Button> buttonList = new List<Button>();  // Can hold Button controls only
```



注意：光是讲述泛型就要用一整本书，这里提到的只是它的一点皮毛。Wrox 出版社出版了一本这样的书： *Professional .NET 2.0 Generics*，ISBN: 978-0-7645-5988-4。尽管它最初是为 ASP.NET 2.0 而写的，但是书中介绍的所有概念和示例现在仍然适用。

尽管上面提到的泛型示例只涉及了泛型的皮毛。然而，当构建 ASP.NET Web 站点时，往往并不需要用到泛型提供的所有高级功能。由于 `List` 集合相当有用，因此这里必须讨论一下。毫无疑问，

您肯定会在自己的代码中以某种方式使用这个集合。

虽然对于向集合中添加元素来说，Add 方法很有用，但是当需要同时向一个集合添加多个元素时，该方法有时会很繁琐。为了使其更简单，.NET 提供了集合初始化器(collection initializers)。使用集合初始化器可以声明一个集合并一次性添加多个元素。这是通过将元素添加到一对花括号中(在 VB.NET 中，要在前面添加关键字 From)实现的，如下所示：

VB.NET

```
Dim myList As New List(Of Integer) From {1, 2, 3, 4, 5}
```

C#

```
List<int> myList = new List<int>() { 1, 2, 3, 4, 5 };
```

运行这段代码后，列表中便会填充 5 个整数。

集合初始化器的使用并不局限于 List 类或整数，它还可以用于其他的集合类型和数据类型。

5.3 语句

为了让程序或 Web 站点做一些有用的事，需要提供一些可以执行的代码语句。语句的动作范围很广，如显示按钮、发送电子邮件、当用户单击按钮时执行这段或那段代码等。然而，仅仅执行这些动作还不够。常常需要在仅当某些条件为真时执行一些代码。例如，如果某电子商务网站的访问者一次购买了超过 100 美元的商品，它们可能会得到 10%的折扣；否则就要全价付款。因此，条件或决策是编程语言中非常重要的语句。另一种重要语句是循环(loop)。循环允许重复某段代码若干次。例如，可以有从 1~10 的循环，在每次循环后执行一些操作。或者，可以在一个购物车的产品之间循环，求和得到总价钱。

最后一个重要部分是运算符。运算符可以对值做一些事，或者，更确切地说，允许对它们进行运算。例如，可以使用运算符加上或减去某些值、连接(合并)它们，或者与其他值相比较。

下面 3 节将深入研究运算符、决策制定和循环。

5.3.1 运算符

最重要的运算符可以根据逻辑划分为 5 种类型。在这 5 种类型中，赋值运算符可能是最容易理解和使用的 一种运算符。

1. 赋值运算符

赋值运算符用于向变量赋值。这个值可以来自多个源：常量值(如数字 6)、另一个变量的值、表达式或函数的结果(后面将会讨论)。赋值运算符的最简形式如下：

VB.NET

```
Dim age As Integer = 38
```

C#

```
int age = 38;
```

当某人刚过完生日后，他的 age 变量是多少呢？此时，需要将他的 age 值加 1。这时就用到了算术运算符。

2. 算术运算符

算术运算符可以对变量和值进行大部分常规计算，如加法、减法和除法。表 5-2 列出了 VB.NET 和 C#语言的算术运算符。

表 5-2

VB.NET	C#	用 法
+	+	将两个值加在一起。这些值可以是 Int32 这样的数值类型，但也可以是 String(在这种情况下加法就是连接)
-	-	从一个值中减去另一个值
*	*	两个值相乘
/	/	两个值相除
\	n/a	两个值相除，但总是返回舍入整数
^	n/a	将一个值作为另一个值的幂
Mod	%	两个整数相除，返回余数

前 5 个运算符应该比较熟悉，它们的用法也相当简单。下面的代码段显示了可以使用这些运算符进行的基本运算：

VB.NET

```
Dim firstNumber As Integer = 100
Dim secondNumber As Single = 23.5
Dim result As Double = 0

result = firstNumber + secondNumber      ' Results in 123.5
result = firstNumber - secondNumber      ' Results in 76.5
result = firstNumber * secondNumber      ' Results in 2350
result = firstNumber / secondNumber      ' Results in 4.25531914893617
result = firstNumber \ secondNumber      ' Results in 4
```

C#

```
int firstNumber = 100;
float secondNumber = 23.5F;
double result = 0;

result = firstNumber + secondNumber;    // Results in 123.5
result = firstNumber - secondNumber;    // Results in 76.5
result = firstNumber * secondNumber;    // Results in 2350
```



```
result = firstNumber / secondNumber;    // Results in 4.25531914893617
```

VB.NET 支持\运算符，它主要是执行除法，然后丢掉余数，将得到的值向下取为最近的整数。C#对此算法没有特定的运算符。然而，当将两个值相除时，结果也总是整数。这意味着 7(存储为 int 型)除以 2(存储为 int 型)结果将为 3。重要的是要意识到会发生取整，否则就会得到非预期的结果。

注意，在 C#示例中需要向值 23.5 后面添加字母 F。这就告知了编译器，实际上想要 float 型而不是 double 型的值。

最后两个运算符需要稍微多作些解释。首先，^运算符(把一个数作为另一个数的幂)仅用于 VB.NET 语言。

VB.NET

```
Dim result As Double

result = 2 ^ 3           ' Results in 8 (2 * 2 * 2)
result = 3 ^ 2           ' Results in 9 (3 * 3)
```

C#不支持此运算符，但是可以使用.NET Framework 支持的 Math.Pow 来完成这样的行为。下面的代码段在功能上等价于上面那段代码：

C#

```
result = Math.Pow(2, 3);    // Results in 8 (2 * 2 * 2)
result = Math.Pow(3, 2);    // Results in 9 (3 * 3)
```

当然，在 VB.NET 中也可以使用 Math.Pow，因此如果用的语言是 VB.NET，就有两种方法可以选择。

最后一个运算符为求模(mod 或 modulus)运算符。它返回两个数相除的余数，如：

VB.NET

```
Dim firstNumber As Integer = 17
Dim secondNumber As Integer = 3
Dim result As Integer = firstNumber Mod secondNumber    ' Results in 2
```

C#

```
int firstNumber = 17;
int secondNumber = 3;
int result = firstNumber % secondNumber;    // Results in 2
```

简单地说，求模运算符是以尽可能多的次数从第一个数中减去第二个数，然后返回余数。在上面的示例中减法过程持续进行 5 次，减去的总和为 15，得到余数为 2，然后将 2 返回并存储在结果中。求模运算符通常用于确定一个数是奇数还是偶数。

使用运算符时，重要的是要记住它们的优先级。通过下面的计算，可以看出优先级的重要性：

```
2+10*4
```

该运算的结果是什么？如果首先将 2 和 10 相加，然后将得到的结果与 4 相乘，则得到的结果是 48。但是正确的结果是 42，即首先将 10 和 4 相乘，得到 40。然后把 2 加到 40 上，得到最终结果

42. 表 5-3 列出了 VB.NET 和 C#中的运算符的优先级。

表 5-3

VB.NET	优 先 级	C#	优 先 级
^	求幂	*, /, %	乘法、除法和求模
*, /	乘法和除法	+, -	加法和减法
\	整除		
Mod	求模		
+, -	加法、减法以及使用+号进行字符串连接		
&	字符串连接		

可以在表达式中使用圆括号来强制改变运算的顺序。首先对括号表达式中的内容求值，就会得到一个不同的顺序。例如：

```
(2+10)*4
```

现在该表达式的结果是 48，因为在乘法运算之前先进行了加法运算。

另一组比较常见的运算符是比较运算符，用来对值进行比较。

3. 比较运算符

与算术运算符一样，VB.NET 与 C#都有各自的一组比较运算符。比较运算符总是比较两个值或两个表达式，然后返回一个 Boolean 值作为结果。表 5-4 列出了最常见的比较运算符。

表 5-4

VB.NET	C#	用 法
=	==	检查两个值是否相等
<>	!=	检查两个值是否不等
<	<	检查第一个值是否小于第二个值
>	>	检查第一个值是否大于第二个值
<=	<=	检查第一个值是否小于等于第二个值
>=	>=	检查第一个值是否大于等于第二个值
Is	is	在 VB.NET 中：比较两个对象； 在 C#中：检查一个变量是否属于某种类型

从表 5-4 中注意到的第一件事是 C#使用两个等于号(==)作为标准比较运算符。这样就使它明显不同于赋值运算符。在 C#中常犯的错误是在比较两个值时只用一个等号。例如下面的示例：

```
if (result = 4)
{
    // Do something here with result
}
```

这段代码的意图是想看看结果是否等于 4。然而，由于使用了赋值运算符而不是正确的比较运算符，因此会得到编译错误，如图 5-2 所示。

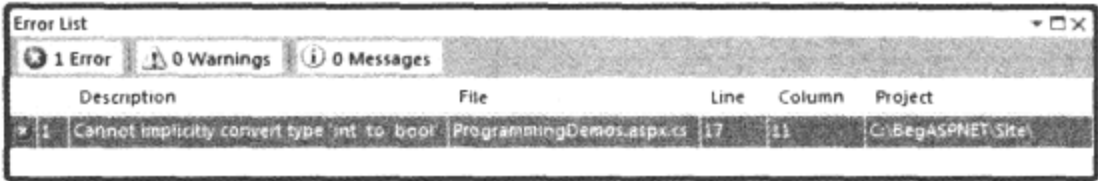


图 5-2

乍一看这条错误消息有些不好理解。然而，如果仔细看一下代码，其意义就会明了。首先对 result 赋值 4，然后将该值用于 if 语句中。但是 if 语句需要一个 Boolean 值来确定是否应执行 if 块内的代码。因为无法将整型值转换成 Boolean 值，所以会得到一个编译错误。修正这个错误并不难，只要采用正确的比较运算符即可：

```
if (result == 4)
{
    // Do something here with result
}
```

与简单的比较运算符相似，可以使用其他运算符来比较两个值：

VB.NET

```
4 > 5           ' 4 is not greater than 5; evaluates to False
4 <> 5          ' 4 is not equal to 5; evaluates to True
5 >= 4          ' 5 is greater than or equal to 4; evaluates to True
```

C#

```
4 > 5           // 4 is not greater than 5; evaluates to false
4 != 5          // 4 is not equal to 5; evaluates to true
5 >= 4          // 5 is greater than or equal to 4; evaluates to true
```

VB.NET 中的 Is 关键字和 C#中的 is 做的事情完全不同。在 VB.NET 中，Is 比较两个对象的实例，在本章的后半部分将会介绍这方面的知识。在 C#中，使用 is 来判断某个变量是否与某个类型兼容。在 VB.NET 中可以使用 TypeOf 运算符来完成这个判断。下面两个示例在功能上是等价的：

VB.NET

```
Dim myTextBox As TextBox = New TextBox()

If TypeOf myTextBox Is TextBox Then
    ' Run some code when myTextBox is a TextBox
End If
```

C#

```
TextBox myTextBox = new TextBox();
```



```
if (myTextBox is TextBox)
{
    // Run some code when myTextBox is a TextBox
}
```

有一种算术运算符允许将两个值彼此相加。即，用加号(+)将两个值加在一起。但是，如果要合并两个值，而不是相加，该怎么办呢？这时就要用到连接运算符。

4. 连接运算符

要连接两个字符串，在 C#中使用+字符，在 VB.NET 中用&字符。此外，可以用+=和&=合并连接与赋值运算符。看下面的示例：

VB.NET

```
Dim firstString As String = "Hello "
Dim secondString As String = "World"
Dim result As String

' The following three blocks are all functionally equivalent
' and result in the value "Hello World"

result = firstString & secondString

result = firstString
result = result & secondString

result = firstString
result &= secondString
```

C#

```
string firstString = "Hello ";
string secondString = "World";
string result;

// The following three blocks are all functionally equivalent
// and result in the value "Hello World"

result = firstString + secondString;

result = firstString;
result = result + secondString;

result = firstString;
result += secondString;
```

除了 VB.NET 中的&和&=连接运算符，还可以用+和+=。然而，根据要连接的表达式的数据类型，有时可能得不到预期的结果。看一下这段代码：

```
Dim firstNumber As String = "4"
Dim secondNumber As Integer = 5
Dim result As String = firstNumber + secondNumber
```

由于 firstNumber 是 String 类型，可能预期最终结果是 45，即 4 与 5 连接。然而默认情况下，

VB.NET 编译器会隐式地将 String "4"转换为数字 4，然后就会进行相加而不是连接，因此 result 为值 9。

为了避免这种歧义的状况，应总是使用&和&=运算符来连接值。此外，可以将下面这行代码添加到代码文件上方，让 VB.NET 不再自动转换这些值。

```
Option Strict On
```

这样就能强制编译器在打算进行隐式转换时生成错误，如上面的示例所示。

最后一组值得一提的运算符是逻辑运算符，将在下一节讨论。

5. 逻辑运算符

逻辑运算符用于合并多个表达式的结果，确定多个条件是真还是假。表 5-5 列出了最常见的逻辑运算符。

表 5-5

VB.NET	C#	用 法
And	&	当两个表达式结果都为 True 值时返回 True
Or		如果至少有一个表达式结果为 True 值就返回 True
Not	!	对表达式的结果取反
AndAlso	&&	允许短路逻辑 And 条件检查
OrElse		允许短路逻辑 Or 条件检查

And、Or 和 Not 运算符(在 C# 中是&、| 和!)的用法相当简单，如下面的代码段所示：

VB.NET

```
Dim num1 As Integer = 3
Dim num2 As Integer = 7

If num1 = 3 And num2 = 7 Then      ' Evaluates to True because both
                                   ' expressions are True

If num1 = 2 And num2 = 7 Then      ' Evaluates to False because num1 is not 2

If num1 = 3 Or num2 = 11 Then      ' Evaluates to True because num1 is 3

If Not num1 = 5 Then                ' Evaluates to True because num1 is not 5
```

C#

```
int num1 = 3;
int num2 = 7;

if (num1 == 3 & num2 == 7)         // Evaluates to true because both
                                   // expressions are true

if (num1 == 2 & num2 == 7)         // Evaluates to false because num1 is not 2
```

```
if (num1 == 3 | num2 == 11)           // Evaluates to true because num1 is 3

if (!(num1 == 5))                     // Evaluates to true because num1 is not 5
```

VB.NET 中的 **AndAlso** 和 **OrElse** 运算符(C#中是**&&**和**|**)的工作方式非常类似于 **And** 和 **Or**(C#中是**&**和**|**)。它们的区别在于当第一个表达式已经能确定整个表达式的值时，根本不会计算第二个表达式的值。当使用一个简单的 **And** 运算符时：

```
If num1 = 2 And num2 = 7 Then
```

两个表达式都会检查。这意味着既要求 **num1** 又要求 **num2** 的值，来判断它们是否分别等于 2 和 7。然而，由于 **num1** 不等于 2，因此就不用求 **num2** 的值了，因为第二个表达式的结果根本不会改变合并表达式的最终结果。这时就可以用 **AndAlso** 运算符(在 C#中为**&&**)短路逻辑条件检查：

VB.NET

```
If num1 = 2 AndAlso num2 = 7 Then
```

C#

```
if (num1 == 2 && num2 == 7)
```

在这段代码中，表达式 **num2 =7**(C#中为 **num2==7**)根本不会计算，因为 **num1** 已经不符合必需的条件了。

对于这些简单的表达式，这种短路运算的好处可能看不太明显，但是如果其中一个表达式实际上是缓慢的、耗时长的运算时，这样做就能大大提高性能。我们来看看下面的虚构代码：

VB.NET

```
If userName = "Administrator" And GetNumberOfRecordsFromDatabase() > 0 Then
```

C#

```
if (userName == "Administrator" & GetNumberOfRecordsFromDatabase() > 0)
```

仅在当前用户为 **Administrator** 时才会执行这个 **If** 块的代码，这种对数据库的虚构调用至少会返回一个记录。现在，设想 **GetNumberOfRecordsFromDatabase()**是一个长运算。如果当前用户不是 **Administrator**，那么执行这个运算就很浪费时间。可以用 **AndAlso**(C#中是**&&**)来解决这个问题：

VB.NET

```
If userName = "Administrator" AndAlso GetNumberOfRecordsFromDatabase() > 0 Then
```

C#

```
if (userName == "Administrator" && GetNumberOfRecordsFromDatabase() > 0)
```

现在，只有当前用户为 **Administrator** 时才会执行 **GetNumberOfRecordsFromDatabase()**。对于所有其他用户，会忽略此代码，从而提高性能。

前面的示例中大多都使用了 **If** 语句来演示逻辑运算符。**If** 语句本身也是一种非常重要的语言结构。下面将讨论代码中 **If** 语句和其他决策方式。

5.3.2 做决策

开发人员经常需要在应用程序中做决策。例如，当某个用户不是管理员时，需要隐藏 Web 窗体上的一个按钮。或者，有时需要将表中的偶数行显示为浅灰色背景，而将奇数行显示为白色背景。所有这些决策都可以用几个不同的逻辑结构来完成：If、If Else、ElseIf 和 switch 或 Select Case 语句。

1. If、If Else 和 ElseIf 结构

If 语句是所有决策语句中最简单的语句。If 语句含有两个相关部分：测试条件，当条件为 True 时执行代码。例如，可以使用如下代码做一个只有管理员可见的按钮：

VB.NET

```
If User.IsInRole("Administrators") Then
    DeleteButton.Visible = True
End If
```

C#

```
if (User.IsInRole("Administrators"))
{
    DeleteButton.Visible = true;
}
```

注意，VB.NET 使用 If 和 End If 关键字，而 C#使用 if 和一对花括号来表示要执行的代码块。同样，在 C#中，要求将要测试的条件用圆括号括起来，而 VB.NET 要求在条件后面使用关键字 Then。

如果条件不是 True，往往要执行一个不同的动作。只要使用逻辑非运算符 Not 或者!，就能直接写另一个语句：

VB.NET

```
If User.IsInRole("Administrators") Then
    DeleteButton.Visible = True
End If
If Not User.IsInRole("Administrators") Then
    DeleteButton.Visible = False
End If
```

C#

```
if (User.IsInRole("Administrators"))
{
    DeleteButton.Visible = true;
}
if (!User.IsInRole("Administrators"))
{
    DeleteButton.Visible = false;
}
```

显然，这样会导致混乱的代码，因为需要为每个表达式重复计算两次：为 True 的情况计算一次，为 False 的情况再计算一次。幸运的是，有一种较容易的解决方案：Else 块(在 C#中是 else)。

VB.NET

```
If User.IsInRole("Administrators") Then
```

```
        DeleteButton.Visible = True
    Else
        DeleteButton.Visible = False
    End If
```

```
C#
if (User.IsInRole("Administrators"))
{
    DeleteButton.Visible = true;
}
else
{
    DeleteButton.Visible = false;
}
```

对于简单的条件，这种 If Else 结构还不错。但是在多于两个选项的情况下仅用 If Else 就不够了。在这些情况下，在 VB.NET 中可以使用 ElseIf，在 C#中使用 else if 梯次结构(ladder)。

假定站点使用了 3 种不同的角色：管理员、内容管理人员和标准成员。管理员能创建和删除内容；内容管理人员只能创建新内容，而对于成员这两个动作都不能做。要显示或隐藏相关按钮，可以使用下面的代码：

```
VB.NET
If User.IsInRole("Administrators") Then
    CreateNewArticleButton.Visible = True
    DeleteArticleButton.Visible = True
ElseIf User.IsInRole("ContentManagers") Then
    CreateNewArticleButton.Visible = True
    DeleteArticleButton.Visible = False
ElseIf User.IsInRole("Members") Then
    CreateNewArticleButton.Visible = False
    DeleteArticleButton.Visible = False
End If
```

```
C#
if (User.IsInRole("Administrators"))
{
    CreateNewArticleButton.Visible = true;
    DeleteArticleButton.Visible = true;
}
else if (User.IsInRole("ContentManagers"))
{
    CreateNewArticleButton.Visible = true;
    DeleteArticleButton.Visible = false;
}
else if (User.IsInRole("Members"))
{
    CreateNewArticleButton.Visible = false;
    DeleteArticleButton.Visible = false;
}
```

尽管 ElseIf 或 else if 梯次结构有助于使代码更可读，但是如果有很多表达式要测试，最终仍然可能写出难读的代码。如果遇到这种情况，可以使用 Select Case(VB.NET)或 switch(C#)语句。

2. Switches/Select Case 结构

假定正在为星期六举行的音乐会构建一个 Web 站点。在这个星期中，访问者能够在线购买星期六的演出票。为了鼓励访问者尽早购票，您决定给早买者一定的折扣：在这个星期中，购票越早，票越便宜。计算折扣率的代码如下所示，其中使用了 Select Case/switch 语句：

VB.NET

```
Dim today As DateTime = DateTime.Now
Dim discountRate As Double = 0

Select Case today.DayOfWeek
    Case DayOfWeek.Monday
        discountRate = 0.4
    Case DayOfWeek.Tuesday
        discountRate = 0.3
    Case DayOfWeek.Wednesday
        discountRate = 0.2
    Case DayOfWeek.Thursday
        discountRate = 0.1
    Case Else
        discountRate = 0
End Select
```

C#

```
DateTime today = DateTime.Now;
double discountRate = 0;

switch (today.DayOfWeek)
{
    case DayOfWeek.Monday:
        discountRate = 0.4;
        break;
    case DayOfWeek.Tuesday:
        discountRate = 0.3;
        break;
    case DayOfWeek.Wednesday:
        discountRate = 0.2;
        break;
    case DayOfWeek.Thursday:
        discountRate = 0.1;
        break;
    default:
        discountRate = 0;
        break;
}
```

对于有折扣的每一天(星期一到星期四)，都有一个 Case 块。VB.NET 与 C#语法之间的区别很小：C#中 case 的字母 c 用小写，而且每个 case 标签后面要求有个冒号。此外，需要用 break 语句退出各个块。在运行时，给条件(today.DayOfWeek)赋值并执行正确的块。重要的是要了解只会执行相关的块，而不会执行其他块。当没有找到有效的块时(在星期五到星期天之间的某一天执行代码)，则会激活 Case Else 或 default 块。不要求写 Case Else 或 default 块。不过还是建议写这样的块，因为这样

可以使代码更清晰易读。上面的示例中可以省略这样的块，因为 `discountRate` 已经在代码块的上方得到了默认值 0。

为了对到目前为止所提到的语句有个感性认识，下面的“试一试”练习在一个小演示应用程序中使用了这些语句。

试一试 创建一个简单的基于 Web 的计算器

在本练习中将创建一个简单的计算器，能进行值的加、减、乘、除运算。它显示了如何使用一些逻辑和赋值运算符演示 If 和 Select Case/switch 结构。

(1) 首先在 Demos 文件夹下创建一个新 Web 窗体 `CalculatorDemo.aspx`。注意不要命名为页面 `Calculator`，否则在本章后面创建这个名称的类时会遇到麻烦。同样，要确保使用 Code Behind 模型并从语言下拉列表中选择正确的编程语言。

(2) 将页面切换到 Design 视图中，单击虚线矩形把焦点放入其中，从主菜单中选择 Table | Insert Table 命令添加一个 3 行 3 列的表。合并第一行的 3 个单元格：选中它们，右击选中区域，从出现的菜单中选择 Modify | Merge Cells 命令。

(3) 向页面中添加下列控件，将它们的 ID 和其他属性设置成如表 5-6 所示的值，并将控件排列为图 5-3 所示的样子。

表 5-6

控 件 类 型	控 件 ID	属 性 设 置
Label	ResultLabel	清除它的 Text 属性。要做到这一点，在 Properties 面板中右击属性名并选择 Reset 命令
TextBox	Value1Box1	
DropDownList	OperatorList	使用 DropDownList 的 Smart Tasks 面板为下面的 4 个算术运算符添加 4 个 ListItem: + - * /
TextBox	ValueBox2	
Button	CalculateButton	将按钮的 Text 属性设置为 Calculate

完成后，页面在 Design 视图中应如图 5-3 所示。

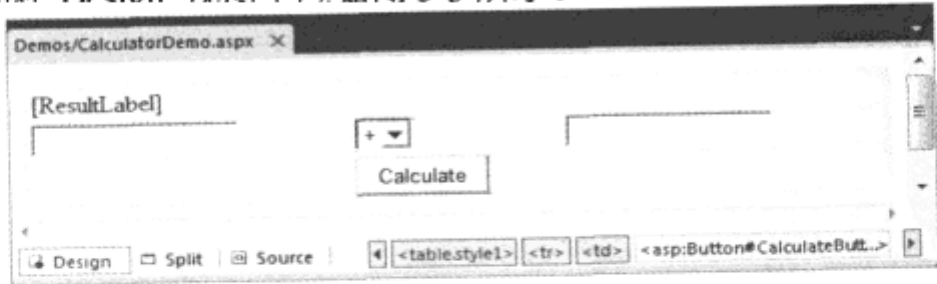


图 5-3

(4) 双击 Calculate 按钮并将下面突出显示的代码添加到 VWD 中的代码占位符中：

VB.NET

```
Protected Sub CalculateButton_Click(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles CalculateButton.Click
    If ValueBox1.Text.Length > 0 AndAlso ValueBox2.Text.Length > 0 Then

        Dim result As Double = 0
        Dim value1 As Double = Convert.ToDouble(ValueBox1.Text)
        Dim value2 As Double = Convert.ToDouble(ValueBox2.Text)

        Select Case OperatorList.SelectedValue
            Case "+"
                result = value1 + value2
            Case "-"
                result = value1 - value2
            Case "*"
                result = value1 * value2
            Case "/"
                result = value1 / value2
        End Select
        ResultLabel.Text = result.ToString()
    Else
        ResultLabel.Text = String.Empty
    End If
End Sub
```

C#

```
protected void CalculateButton_Click(object sender, EventArgs e)
{
    if (ValueBox1.Text.Length > 0 && ValueBox2.Text.Length > 0)
    {
        double result = 0;
        double value1 = Convert.ToDouble(ValueBox1.Text);
        double value2 = Convert.ToDouble(ValueBox2.Text);

        switch (OperatorList.SelectedValue)
        {
            case "+":
                result = value1 + value2;
                break;
            case "-":
                result = value1 - value2;
                break;
            case "*":
                result = value1 * value2;
                break;
            case "/":
                result = value1 / value2;
                break;
        }
        ResultLabel.Text = result.ToString();
    }
    else
```

```
{
    ResultLabel.Text = string.Empty;
}
}
```

(5) 保存所有修改然后在浏览器中按下 Ctrl+F5 组合键打开页面。如果看到一个错误提示而没有看到页面，请检查输入的代码是否与这里显示的完全相同，以及有没有根据表 5-6 命名所有控件。

(6) 在第一个和第二个文本框中各输入一个数，从下拉列表中选择一个运算符，然后单击 Calculate 按钮。这时就会激活 Code Behind 文件中的代码，然后，会根据在下拉列表中选择项，执行恰当的计算并用结果更新标签。

(7) 继续进行练习并尝试一些其他数和运算符；可以发现每次单击 Calculate 按钮时，计算器都会进行正确的运算。

工作原理

当输入两个值然后单击 Calculate 按钮时，就会激活 Code Behind 文件中的下列代码：

```
VB.NET
If ValueBox1.Text.Length > 0 AndAlso ValueBox2.Text.Length > 0 Then

C#
if (ValueBox1.Text.Length > 0 && ValueBox2.Text.Length > 0)
```

这段代码对于确保两个文本框中都含有值是必需的(第 9 章将介绍进行这种有效性验证的更简洁的方法)。该代码使用了一个简单的 If 语句来确保两个字段中都含有值。它还使用了运算符 AndAlso 或 &&，避免在第一个 TextBox 是空值时还检查第二个 TextBox 的 Text 属性。

然后，代码声明一个 Double 变量来存储计算的结果，然后从两个文本框控件中取得值，用 Convert 类的 ToDouble 方法将值转换为 Double 类型，然后设置一个 Select Case(在 C#中是 switch)块来处理从下拉列表选择的运算符类型。

```
VB.NET
Select Case OperatorList.SelectedValue
    Case "+"
        result = value1 + value2

C#
switch (OperatorList.SelectedValue)
{
    case "+":
        result = value1 + value2;
        break;
```

对于下拉列表中的每一项，都有一个 case 语句。当从列表选择了+运算符时，就会激活第一个 case 块中的代码，并将在两个文本框中输入的数值的和赋值给 result。同样地，当选择减号运算符时，就将两个值相减。

最后，将结果转换为 String，然后显示在名为 Result Label 的标签上。

Select Case / switch 语句是所要讨论的在代码中做决策的最后一个语句。还剩最后一组语句没有

介绍：循环，用来在代码中或者集合中的对象上进行循环。

5.3.3 循环

循环在许多应用程序中都非常有用，因为它们允许反复执行代码，而只需编写该代码一次。例如，如果有一个 Web 站点需要通过电子邮件将业务资讯发送给它的 20000 位订阅者，则只需写一次发送业务资讯的代码，然后用一个循环将业务资讯发送给该代码在数据库中找到的每位订阅者。

循环有几种类型，每种类型都有各自的用法和优点。

1. For 循环

For 循环简单地重复它的代码预定义的次数。在代码中定义进行循环时要迭代的确切次数。For 循环格式如下：

VB.NET

```
For counter [As datatype] = start To end [Step stepSize]
    ' Code that must be executed for each iteration
Next [counter]
```

C#

```
for (startCondition; endCondition; step definition)
{
    // Code that must be executed for each iteration
}
```

它看起来有点奇怪，但是如果是具体示例就比较好理解了：

VB.NET

```
For loopCount As Integer = 1 To 10
    Label1.Text &= loopCount.ToString() & "<br />"
Next
```

C#

```
for (int loopCount = 1; loopCount <= 10; loopCount++)
{
    Label1.Text += loopCount.ToString() + "<br />";
}
```

虽然这两种语言的语法相当不同，但是两个代码示例都执行了相同的动作：它们在一个 Label 控件上写 1~10 的数字。即，该循环首先向变量 loopCount 赋予 1。接着，将值转换为 String，并赋予 Label 控件。然后，loopCount 递增 1，循环就会继续。这个过程会继续进行，直到 loopCount 为 10，然后循环结束。本例中使用了硬编码的数字。然而，可以用变量或其他对象中的动态值代替开始和结束条件。例如，如果使用前面介绍的角色数组，就可以这样写出数组中的每个角色：

VB.NET

```
For loopCount As Integer = 0 To roles.Length - 1
    Label1.Text &= roles(loopCount) & "<br />"
Next
```

```
C#
for (int loopCount = 0; loopCount < roles.Length; loopCount ++){
    Label1.Text += roles[loopCount] + "<br />";
}
```

因为这些数组是基于 0 索引的，所以需要在 VB.NET 中用 `roles(0)`、在 C# 中用 `roles[0]` 来表示第一项。这也意味着循环需要从 0 开始。数组的 `Length` 属性返回数组包含的项的总数。所以当数组中有 3 个角色时，`Length` 会返回 3。因此，VB.NET 中代码会从 `Length` 中减去 1，并用得到的值作为循环的结束条件，使循环从 0~2 运行，依次访问这 3 个元素。

不过 C# 示例就没有从 `Length` 中减去 1。相反，它使用了这个表达式：

```
loopCount < roles.Length;
```

因此，只要 `loopCount` 的值小于数组的长度，循环就会继续。同样，这会使得循环访问所有 3 个数组元素，从 0 到 2。

通过在每次迭代中将 `loopCount` 变量的值加 1 来对前面的例子进行循环操作。为了增大循环的步长，可以使用 VB.NET 中的关键字 `Step`，而 C# 允许直接在定义 `For` 循环时指定循环的步长大小：

```
VB.NET
For loopCount As Integer = 0 To 10 Step 2
Next
```

```
C#
for (int loopCount = 0; loopCount <= 10; loopCount = loopCount + 2){
}
```

这个循环给 `loopCount` 的赋值是 0 到 10 之间的偶数。

如果是在数组或数据集合上循环，则还可以使用另一种更易读且易用的循环：`For Each` 或 `foreach` 循环。

2. For Each / foreach 循环

VB.NET 中的 `For Each` 循环和 C# 中的 `foreach` 循环简单地在集合中的所有项上迭代。以角色数组为例，可以执行下面的代码在 `Label` 控件上显示每个角色的名称：

```
VB.NET
For Each role As String In roles
    Label1.Text &= role & "<br />"
Next
```

```
C#
foreach (string role in roles)
{
    Label1.Text += role + "<br />";
}
```

由于 `roles` 变量是字符串的数组，因此也需要用 `String` 设置循环。同样，如果所循环的集合中包

含 Integer 或 Boolean 数据类型，就要分别用 Integer 或 Boolean 设置循环。

除了 For 和 For Each 循环外，还有一种循环：While 循环。

3. While 循环

顾名思义，While 循环是当某个条件为真时进行循环。与其他两个自结束的循环不同，如果不小心，While 循环可能会无限循环下去。下面的示例说明了如何使用 While 循环：

VB.NET

```
Dim success As Boolean = False
While Not success
    success = SendEmailMessage()
End While
```

C#

```
bool success = false;
while (!success)
{
    success = SendEmailMessage();
}
```

这段代码试图通过使用虚构的 SendEmailMessage 方法来发送电子邮件消息，它会一直尝试直到成功——即，只要变量 success 含有值 False(在 C#中是 false)。注意其中使用了 Not 和!来对变量 success 的值取反。SendEmailMessage 方法被假定为当成功时返回 True，失败时返回 False。如果一切按计划进行，那么代码会进行循环并调用 SendEmailMessage。如果它返回 True，则不再满足循环条件，循环就会终止。然而，当 SendEmailMessage 返回 False 时，因为邮件服务器已关闭(例如)，所以循环会继续进行并再次调用 SendEmailMessage。

为了避免 While 循环进入无限循环，最好在尝试一定次数以后添加一个终止循环的条件。例如，下面的代码有助于避免当邮件服务器关闭时出现无限循环：

VB.NET

```
Dim success As Boolean = False
Dim loopCount As Integer = 0
While Not success And loopCount < 3
    success = SendEmailMessage()
    loopCount = loopCount + 1
End While
```

C#

```
bool success = false;
int loopCount = 0;
while (!success && loopCount < 3)
{
    success = SendEmailMessage();
    loopCount = loopCount + 1;
}
```

在这段代码中，变量 loopCount 负责在调用 3 次 SendEmailMessage 后退出循环。如果不使用 loopCount = loopCount + 1，也可以将连接与赋值运算符结合起来使用，如下所示：

VB.NET

```
loopCount += 1
```

C#

```
loopCount += 1;

// Alternatively C# enables you to do this:
loopCount++;
```

所有这些示例的结果都相同：loopCount 值递增 1，然后再将新的总和赋予 loopCount。C#快捷方法 loopCount++是非常常用的将变量递增 1 的方式。类似地，可以使用 loopCount--或 loopCount -=1 将值递减 1。后一种也可以用于 VB.NET。

除了 While 循环外，还有其他几种循环，如 Do While 循环(确保代码至少执行一次)，Do Until 循环是一直执行直到某个条件为真时结束循环，它与 While 循环中当某个条件为真时进行循环相反。

到目前为止，所看到的代码都是由短小简单的示例组成，它们可以直接放在 Web 页面的 Code Behind 文件中，例如在前面看到的 Page_Load 或按钮单击事件的处理程序中。然而，在现实世界的 Web 站点中，可能对结构化和组织代码的需要会更多。下一节将介绍几种结构化和组织方式。

5.4 组织代码

当开始向 Web 站点添加很多页面时，几乎肯定会有可以在多个页面中重用的代码。例如，可能有一些在多个文件中都需要的从 web.config 文件中读取设置的代码。或者，从不同页面中发送带有用户详细信息的电子邮件。因此，需要找到一种使代码集中的方式。在 ASP.NET 4 Web 站点中可以使用函数与子例程(下面将会讨论)。要使这些函数与子例程对站点中的所有页面可用，需要在一个特殊位置(以后会介绍)创建它们。

5.4.1 方法：函数与子例程

函数与子例程(sub)非常相似；两者都允许创建一个可重用的代码块，这些代码块可以从应用程序的其他位置调用。函数与子例程之间的区别在于函数能返回数据而子例程不能。而它们的相同之处在于函数和子例程都能称为方法(method)。在本章的最后一部分讨论面向对象编程时会再次提到这个术语。

为了使函数与子例程更有用，可以对它们参数化。即可以传递能在函数或子例程内使用的额外信息。函数和子例程通常采用下面的格式：

VB.NET

```
' Define a function
Public Function FunctionName ([parameterList]) As DataType

End Function

' Define a subroutine
Public Sub SubName ([parameterList])

End Sub
```

```
C#
// Define a function
public datatype FunctionName([parameterList])
{
}

// Define a subroutine
public void SubName([parameterList])
{
}
```

以 **Public** 开头的第一行被称为是方法签名(method signature)，因为它定义函数的外观，还包括函数的名称和参数。**Public** 关键字(在 C#中是 **public**)称为访问修饰符，定义了其他 Web 页面或代码文件可以在什么范围内看到此方法。本章后面会详细讨论这个问题。目前，应知道 **Public** 的可见性最大，因此从外部调用代码时能见到此方法。

函数的名称后面跟着圆括号，括号中可以包含一个可选的参数列表。这些代码示例中的斜体示例应当用实际代码中真正的值替换。方括号([])之间的部分是可选的。为了使它更具体一点，下面是一些函数和子例程的示例：

```
VB.NET
Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
    Return a + b
End Function

Public Sub SendEmailMessage(ByVal emailAddress As String)
    ' Code to send an e-mail goes here
End Sub
```

```
C#
public int Add(int a, int b)
{
    return a + b;
}

public void SendEmailMessage(string emailAddress)
{
    // Code to send an e-mail goes here
}
```

在这些代码示例中，很明显函数返回一个值，而子例程不返回值。因此，**Add** 方法用 **Return** 关键字(在 C#中全部是小写的 **return**)返回 **a** 和 **b** 的总和。**VB.NET** 中的 **Sub** 和 C#中的 **void** 方法不要求使用 **Return** 关键字，不过可以用它提前退出方法。

最后，函数和子例程都有参数列表，用来定义传送给方法的变量的名称和数据类型。在方法中，可以像访问普通变量一样访问这些变量。**Add** 方法有两个参数：一个表示加号左边的数，一个表示加号右边的数。**SendEmailMessage** 方法只有一个 **String** 类型的参数，用来保存用户的电子邮件地址。

在 **VB.NET** 示例中可以看到，关键字 **ByVal** 在参数列表中位于每个参数前面。这是所有参数的

默认类型，如果省略它，IDE 会自动加上它。与 `ByVal` 对立的是 `ByRef`。这些关键字可以确定将值发送给函数或子例程的方式。当指定为 `ByVal` 时，就制作了变量的一个副本。一旦方法结束，在方法内对副本所做的修改就会立刻消失。相反，当指定为 `ByRef` 时，就会将对变量的引用传递给方法。对接下来的变量所做的修改也会反映在原始变量上。下面的简短示例演示了其工作方式：

```
Public Sub ByValDemo(ByVal someValue As Integer)
    someValue = someValue + 20
End Sub

Public Sub ByRefDemo(ByRef someValue As Integer)
    someValue = someValue + 20
End Sub

Dim x As Integer = 0
ByValDemo(x)

Label1.Text = x.ToString()    ' Prints out 0; A copy of x is sent to ByValDemo,
                              ' leaving the original value of x unmodified.

Dim y As Integer = 0
ByRefDemo(y)

Label1.Text = y.ToString()    ' Prints out 20; A reference to y is sent
                              ' to ByRefDemo so when that method modified
                              ' someValue, it also changed the variable y.
```

C#有类似的结构，也使用 `ref` 关键字。它与 VB.NET 最大的区别在于，当不想使用引用参数时就不需要指定任何内容，而在调用方法时也需要指定 `ref` 关键字：

```
public void ByRefDemo(ref int someValue)
{
    someValue = someValue + 20;
}

int y = 0;
ByRefDemo(ref y);           // Just as in the VB example, y contains 20
                             // after the call to ByRefDemo
```

使用这样的引用参数时要小心；它可能会改变调用代码中的重要变量。这样会导致难以查出的 bug。

为了让站点范围的方法对于 Web 站点内的页面都可访问，应将它们放在集中的位置。Web 站点的 `App_Code` 文件夹是放置代码的极佳位置。

5.4.2 App_Code 文件夹

`App_Code` 文件夹是特殊的 ASP.NET 4 文件夹。它专门用来保存代码文件，如整个站点都要用到的类。仅用于某个页面的代码(如 `Botton` 控件的单击处理程序)应保留在页面的 `Code Behind` 文件中，正如目前所看到的。



注意：App_Code 文件夹是专用于 Web Site Project 这种项目的，在 Planet Wrox 示例 Web 站点使用的就是该项目类型。Web Application Project 项目类型不使用也不支持 App_Code 文件夹。但是，该项目类型允许在许多位置创建代码文件。当使用 Web Application Project 模板构建站点时，建议创建一个中心代码文件夹(例如，可以称为 Code 或 CodeFiles)，来存储所有的代码文件。为了理解本章及后面几章中的示例，一定要按照第 2 章中的介绍使用 Web Site Project。

为了向站点中添加 App_Code 文件夹，在 Solution Explorer 中右击站点的名称，并选择 Add ASP.NET Folder | App_Code 命令。这个文件夹被添加到站点中并得到一个特殊图标：一个上面带有一个小的代码文档的文件夹，如图 5-4 所示。

添加了 App_Code 文件夹后，就可以开始向其中添加类文件。类文件的扩展名与为站点选择的编程语言一致：C#文件的扩展名是.cs，VB.NET 代码的文件扩展名是.vb。在这些类文件内能创建类，而这些类又包含可以安然完成常见任务的方法(函数与子例程)。类将在本章最后一部分详细讨论；目前，主要介绍代码文件中的方法及如何调用这些方法，而不是关注为什么需要先向类中添加代码。

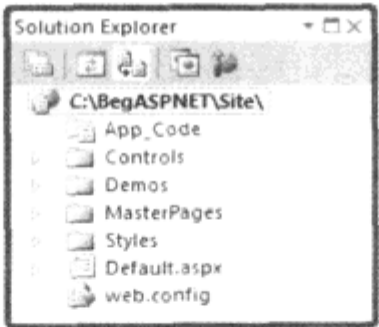


图 5-4

下面的“试一试”练习显示了如何使用 App_Code 文件夹优化在前面的“试一试”练习中创建的计算器。

试一试

优化计算器

在本练习中，将创建一个名为 Calculator 的类，它提供了 4 个方法：Add、Subtract、Multiply 和 Divide。当建立了这个类并能进行必要的计算动作时，修改文件 CalculatorDemo.aspx，以便它能使用新的 Calculator 类。尽管这只是一个示例，但是当需要写大量代码时，将代码从 ASPX 页面移到 App_Code 文件夹中以便被其他应用程序重用，就能大大提高灵活性。这个示例有助于了解一般概念，然而对于帮助理解代码还是显得太短。现在，只需了解计算器的工作原理以及如何调用其方法。在本章后面的面向对象部分，会看到确切的类。

- (1) 如果还未添加 App_Code 文件夹，则首先右击站点并选择 Add ASP.NET Folder | App_Code 命令创建一个 App_Code 文件夹。
- (2) 右击新建的 App_Code 文件夹并选择 Add New Item 命令。
- (3) 在接着出现的对话框中，从 Language 下拉列表中选择适当的编程语言，然后单击 Class。
- (4) 输入 Calculator 作为文件的名称，并单击 Add 按钮。这样会创建一个类文件，其中包含一个名为 Calculator 的类。注意，使用像 Pascal Casing 这样每个单词的首字母都大写的名称来命名类的情况很常见。
- (5) 在定义 Calculator 类的代码行后面，添加下面 4 个方法：

VB.NET

```
Public Class Calculator

    Public Function Add(ByVal a As Double, ByVal b As Double) As Double
        Return a + b
    End Function

    Public Function Subtract(ByVal a As Double, ByVal b As Double) As Double
        Return a - b
    End Function

    Public Function Multiply(ByVal a As Double, ByVal b As Double) As Double
        Return a * b
    End Function

    Public Function Divide(ByVal a As Double, ByVal b As Double) As Double
        Return a / b
    End Function

End Class
```

C#

```
public class Calculator
{

    public double Add(double a, double b)
    {
        return a + b;
    }

    public double Subtract(double a, double b)
    {
        return a - b;
    }

    public double Multiply(double a, double b)
    {
        return a * b;
    }

    public double Divide(double a, double b)
    {
        return a / b;
    }

    public Calculator()
    {
        //
        // TODO: Add constructor logic here
        //
    }
}
```

(6) 接下来，修改 CalculatorDemo.aspx 页面的 Code Behind 文件，以便它能使用刚创建的类。需要进行两次修改：首次需要添加一行创建 Calculator 类的实例的代码，然后需要修改每个 Case 块来使用计算器中的相关计算方法：

VB.NET

```
Dim myCalculator As New Calculator()  
Select Case OperatorList.SelectedValue  
    Case "+"  
        result = myCalculator.Add(value1, value2)  
    Case "-"  
        result = myCalculator.Subtract(value1, value2)  
    Case "*"  
        result = myCalculator.Multiply(value1, value2)  
    Case "/"  
        result = myCalculator.Divide(value1, value2)  
End Select
```

C#

```
Calculator myCalculator = new Calculator();  
switch (OperatorList.SelectedValue)  
{  
    case "+":  
        result = myCalculator.Add(value1, value2);  
        break;  
    case "-":  
        result = myCalculator.Subtract(value1, value2);  
        break;  
    case "*":  
        result = myCalculator.Multiply(value1, value2);  
        break;  
    case "/":  
        result = myCalculator.Divide(value1, value2);  
        break;  
}
```

(7) 保存所有修改并在浏览器中打开页面。计算器仍像以前那样工作；只是这次计算不是在页面的 Code Behind 文件中完成，而是由 App_Code 文件夹中的 Calculator 类完成。

工作原理

在 App_Code 文件夹中创建的文件包含一个名为 Calculator 的类。在本章的最后一部分将介绍关于类的更多知识，现在只需了解类就像对象的定义一样，能够提供可在运行时调用的方法。在这种情况下，Calculator 类的定义包含 4 种进行算术运算的方法。这些方法接受算式左边和右边的参数。各个方法只是完成请求的计算(Add、Subtract 等)，并向调用代码返回结果。

CalculatorDemo.aspx 页面的 Code Behind 文件中的代码首先创建 Calculator 类的一个实例。即基于类的定义在计算机的内存中创建一个对象。为此，它用 New(在 C#中是 new)关键字创建了

Calculator 的一个实例，然后将它存储在变量 myCalculator 中。在本章后面讨论对象时将介绍关于 New 关键字的更多知识。注意这个变量的数据类型是 Calculator，它是类的名称。

```
VB.NET
Dim myCalculator As New Calculator()

C#
Calculator myCalculator = new Calculator();
```

一旦创建了 Calculator 实例，就能调用它的方法。与前面介绍的其他方法一样，Calculator 类的方法接受通过调用代码传递的参数。

```
VB.NET
Case "+"
    result = myCalculator.Add(value1, value2)

C#
case "+":
    result = myCalculator.Add(value1, value2);
    break;
```

然后，Add 方法会将这两个值相加，然后返回 double 类型的结果，存储在变量 result 中。与计算器的第一个版本一样，最终结果显示在带 Label 控件的页面上。

函数和子例程是组织 Web 应用程序的极佳办法。它们允许创建可重用的代码块，能从其他地方方便地调用它们。因为需要多次使用的代码仅定义一次，所以对代码的维护与扩展非常容易。如果在函数中发现 bug，只要修复它在 App_Code 文件中的定义即可，且所有使用该函数的页面都会自动修改过来。除了增加了可维护性外，函数和子例程还使代码更容易阅读，不需要在一个页面内的长列表中滚动，只需调用一个函数并使用返回值即可(如果有返回值的话)。这样就使代码更容易理解，使应用程序中出现 bug 的机会减小。

函数和子例程不是在 NET 项目中组织代码的唯一方式。另一种常用的组织代码的方式是使用名称空间。

5.4.3 使用名称空间组织代码

名称空间可能会让开发新手晕头转向。他们认为名称空间很吓人，他们认为到处都是名称空间，或者看不出使用名称空间的必要性。然而这些感觉都不对，这里只要稍作解释，就可以了解甚至喜欢上名称空间。

名称空间主要是为了解决两个主要问题：组织.NET Framework 和您自己代码中的大量功能，以及避免名称冲突，即两个不同的数据类型采用相同的名称。关于名称空间的一个常见误解是，名称空间与.NET 程序集(.NET Framework 加载和使用的带有.dll 扩展名的文件)有直接的关系，但是事实并非如此。尽管通常会在 System.Web.dll 这样的 DLL 中找到类似于 System.Web.UI 的名称空间，但是在一个 DLL 文件中可以(并且经常会)定义多个名称空间，而一个名称空间也有可能分布在多个程

序集中。稍后在向程序集中添加引用时要记住这一点。

要了解名称空间，打开到目前为止所创建的 ASPX 页面的 Code Behind 文件之一。这时可能看到类似于下面的代码：

```
VB.NET
Partial Class Demos_CalculatorDemo
    Inherits System.Web.UI.Page

C#
public partial class Demos_CalculatorDemo : System.Web.UI.Page
{
```

注意类名定义的后面跟着 Inherits 关键字(在 C#中是冒号)，而该关键字后面跟着 System.Web.UI.Page。后面会介绍这个 Inherits 关键字的用途。在这段代码中，Page 是类(一种数据类型)的名称，它是在 System.Web.UI 名称空间中定义的。通过将 Page 类放在 System.Web.UI 名称空间中，开发人员(以及编译器)就能明白这个类是关于一个 Web 页面的。相反，设想下面的(虚构)类名：

```
Microsoft.Word.Document.Page
```

此代码也是指一个 Page 类。然而，因为它放在 Microsoft.Word.Document 名称空间中，因此很容易看出它是一个 Word 文档页面，而不是 Web 页面。用这种方式就不会将 Web 页面、Word 文档页面弄混淆。这样也有助于编辑器了解指的是哪个类。

名称空间的另一个好处是它们可以帮助找到正确的数据类型。不用在 IntelliSense 列表中显示成千上万个项，而是会显示少数几个最高级的名称空间。当从列表选择一个项并按下句点键(.)时，就会得到另一个相对较短的列表，其中包含类型和位于所选择的名称空间内的其他名称空间。

名称空间只不过是一些简单的容器，可以用句点表示法通过名称引用。它们用来对应用程序中可用的各种数据类型加上前缀。例如，Double 数据类型位于 System 名称空间中，因此它的完全限定名是 System.Double。同样，添加到 Web 页面的 Button 控件位于 System.Web.UI.WebControls 名称空间中，因此它的全名是 System.Web.UI.WebControls.Button。

创建自己的名称空间也十分容易。只要它们不与现有名称冲突，几乎就能构建起自己认为合适的名称空间。例如，可以将 Calculator 类包装在下面的名称空间中(在 App_Code 文件夹中的 Calculator.vb 或 Calculator.cs 中)：

```
VB.NET
Namespace Wrox.Samples

    Public Class Calculator
        ...
    End Class

End Namespace

C#
namespace Wrox.Samples
{
```

```
public class Calculator
{
    ...
}
```

将计算器包装在此名称空间中后，就能创建它的新实例，如下所示：

VB.NET

```
Dim myCalculator As New Wrox.Samples.Calculator()
```

C#

```
Wrox.Samples.Calculator myCalculator = new Wrox.Samples.Calculator();
```

尽管在寻找 Calculator 类时，可以从 IntelliSense 中获得一些帮助，但是很快就会发现输入这些长名称很烦琐。幸运的是，也有解决这种问题的方法。

创建了自己的名称空间后或者要使用一个现有名称空间时，需要让它们在代码中可用。可以使用关键字 Imports(在 VB.NET 中)或 using(在 C#中)实现。例如，为了使 Calculator 类在 Calculator 演示页面中可用，可以向代码中添加下面的名称空间：

VB.NET

```
Imports Wrox.Samples

Partial Class Demos_CalculatorDemo
    Inherits System.Web.UI.Page
```

C#

```
using Wrox.Samples;

public partial class Demos_CalculatorDemo : System.Web.UI.Page
{
```

如果使用的是 C#，就会看到对于像 System 和 System.Web.UI.WebControls 这样的名称空间，ASPX 页面的 Code Behind 文件中默认有若干 using 语句。如果使用的是 VB.NET，就看不到这些引用。因为在 VB.NET Web 站点中，默认名称空间被包括在<namespaces>元素下的计算机的全局 web.config 文件中。

通常，我们知道类的名称，但是不知道该类位于哪个名称空间中。使用 VWD 可以很容易地找到名称空间并添加所需的 Imports 和 using 语句。只需输入所要使用的类的名称，然后将光标放在类名上并按下 Ctrl+. (Ctrl+句点)即可。这样就会看到一个菜单，可从中选择正确的名称空间，如图 5-5 所示。

如果对话框不能够添加 Imports 或 using 语句，那么包含您所寻找的类的程序集就可能无法被项目引用。如果是那样，右击 Solution Explorer 中的 Web 站点并选择 Add Reference 命令。在之后的对话框中可以从.NET 选项卡提供的许多内置的.NET 程序集中选择，也可以使用 Browse 选项卡浏览第三方程序集。一旦添加了引用，就可以通过在类名上再次按 Ctrl+. 组合键来为所寻找的类添加 Imports 或 using 语句。

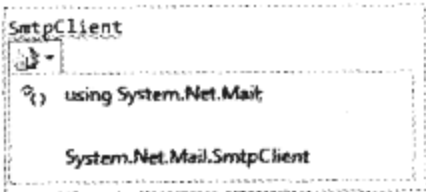


图 5-5

编写大量代码之后，很可能会忘记变量或方法是在哪里声明的，以及它们的用途是什么。因此强烈建议在代码中添加注释。

5.4.4 写注释

无论您是多么简洁的程序员，都可能有一天会看不懂自己所写的代码。几年以后，您编程的方式可能已经改变：学习了新的内容，优化了代码标准并找到了使代码更有效的方式。为了在从现在起的两年后更容易识别和理解现在写的代码，最好对代码进行注释。在代码文件中添加注释的主要方式有两种：内联和作为 XML 注释。

1. 注释内联代码

内联注释直接写在代码语句之间。可以用它们注释现有变量、难懂的循环等。在 VB.NET 中，一次只能在要用作注释的文本前面使用单引号(')注释一行。而要在 C#中注释一行，则需要使用两个斜线字符(//)。此外，在 C#中可以使用/*和*/来注释整块代码，下面的示例介绍了注释的几种不同方法：

VB.NET

```
' Usage: explains the purpose of variables, statements and so on.
' Used to store the number of miles the user has traveled last year.
Dim distanceInMiles As Integer

' Usage: comment out code that's not used (anymore).
' In this example, SomeUnfinishedMethod is commented out
' to prevent it from being executed.
' SomeUnfinishedMethod()

' Usage: End of line comments.
If User.IsInRole("Administrators") Then ' Only allow admins in this area
End If
```

C#

```
// Usage: explains the purpose of variables, statements and so on.
// Used to store the number of miles the user has traveled last year.
int distanceInMiles;

// Usage: comment out code that's not used (anymore).
// In this example, SomeUnfinishedMethod is commented out
// to prevent it from being executed.
// SomeUnfinishedMethod();

// Usage: End of line comments.
if (User.IsInRole("Administrators")) // Only allow admins in this area
{ }

/*
 * This is a block of comments that is often used to add additional
 * information to your code for example to explain a difficult loop. You can
 * also use this to (temporarily) comment a whole block of code.
 */
```

为了使一行代码以注释的形式存在，只需在希望注释开始的地方输入代码字符'或//即可。要使一块代码以注释的形式存在，在文本编辑器中选择它，然后按 Ctrl+K 组合键，再按 Ctrl+C 组合键。类似地，按 Ctrl+K 组合键再按 Ctrl+U 组合键就能取消对选中代码块的注释。

此外，也可以从主菜单中选择 Edit | Advanced | Comment Selection 或 Uncomment Selection 命令，或者在 Text Editor 工具栏上单击这两个命令各自的按钮，如图 5-6 所示。



图 5-6

内联注释通常很适合于文档化代码的少量细节。然而，使用它提供关于代码用途的高级概述也是不错的主意。例如，对于名为 SendEmailMessage 的方法，用一个简短描述解释这个方法的作用以及参数的用处就很不错。这就是 XML 注释发挥作用的地方。

2. 写 XML 注释

XML 注释是作为代码中的 XML 元素(用尖括号<>)添加的，用来描述它的作用、参数、返回值等。VWD IDE 可以帮助写这些注释。所需做的只是将光标放在类或方法前面，并在 VB 中输入'''(3 个单引号)或者在 C#中输入///(3 个正斜杠)。输入这些字符后，IDE 就会插入 summary 和可选参数的 XML 标记，并返回方法的类型。再次以 SendEmailMessage 方法为例。它有两个 String 类型的参数：一个用于要将消息发送到的电子邮件地址，另一个用于邮件正文。应用了 XML 注释后，这个方法应如下所示：

```
VB.NET
''' <summary>
''' Sends out an e-mail to the address specified by emailAddress.
''' </summary>
''' <param name="emailAddress">The e-mail address of the recipient.</param>
''' <param name="mailBody">The body of the mail message.</param>
''' <returns>This method returns True when the message was sent successfully;
''' and False otherwise.</returns>
''' <remarks>Attention: this method assumes a valid mail server is
''' available.</remarks>
Public Function SendEmailMessage(ByVal emailAddress As String,
    ByVal mailBody As String) As Boolean
    ' Implementation goes here
End Function

C#
/// <summary>
/// Sends out an e-mail to the address specified by emailAddress.
/// </summary>
/// <param name="emailAddress">The e-mail address of the recipient.</param>
/// <param name="mailBody">The body of the mail message.</param>
/// <returns>This method returns true when the message was sent successfully;
/// and false otherwise.</returns>
/// <remarks>Attention: this method assumes a valid mail server is
/// available.</remarks>
bool SendEmailMessage(string emailAddress, string mailBody)
```

```
{  
    // Implementation goes here  
}
```

这种注释类型的优秀之处在于，当调用此方法时，在这里输入的注释会显示在代码编辑器的 IntelliSense 工具中(如图 5-7 所示)。

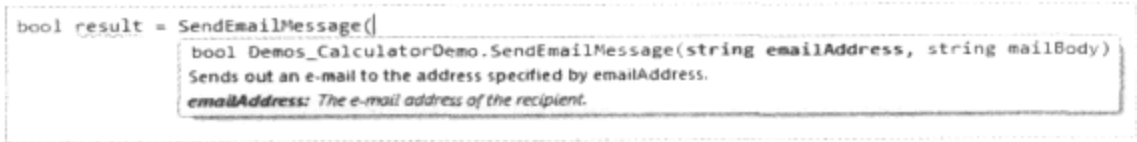


图 5-7

这样您和其他开发人员就能更容易地理解方法及它们的参数的作用。

除了辅助在代码编辑器中的开发外，XML 注释还能用来创建外观漂亮的、如 MSDN 的文档。还有一些第三方工具可以实现这些功能，包括 Microsoft 自带的 Sandcastle(<http://msdn2.microsoft.com/en-us/vstudio/bb608422.aspx>)和 Innovasys 的 Document! X(www.innovasys.com/)。

5.5 面向对象编程基础知识

如果不介绍面向对象(OO)编程，那么介绍使用 ASP.NET 写代码的这一章就不算完整。面向对象或面向对象编程(Object-Oriented Programming, OOP)是一种非常流行的编程技术，它将软件建模为一组互相交互的对象。面向对象编程是.NET Framework 的核心。这个架构内的一切事物基本上都是对象：从简单的事物(如整数)到复杂事物(如 DropDownList 控件、到数据库的连接或者数据驱动的控件)。

由于面向对象编程在.NET 中如此重要，因此熟悉面向对象编程的一般概念就显得尤为重要。不过，也没有必要成为能用 ASP.NET 构建 Web 站点的面向对象编程专家。本节将概述面向对象最重要的术语与概念。这样有助于马上开始面向对象的编程，以便可以在第 6 章开始实际构建有用的应用程序，而不需要在本书中再花上 3 个星期才开始。

5.5.1 重要的面向对象术语

在面向对象的编程中，一切都围绕着对象的概念。事实上，面向对象中的每个事物都是一个对象。但是到底什么是对象呢？类与它们有什么关系呢？

1. 对象

对象是面向对象编程语言中的基本构建块。与在现实世界中一样，面向对象世界中的对象就是一个事物，存储在计算机的内存中。它可以是一个保存某个人年龄的整数，也可以是与另一台计算机上的 SQL Server 的一个开放数据库连接，但是它也可以是更概念上的东西，如一个 Web 页面。在应用程序中，可以使用 New(C#中是 new)关键字创建一个新的对象，正如你所看到的计算器示例一样。创建新对象的过程被称为实例化，而所创建的对象被称为实例。可以实例化复杂的或定制的类型，如 Calculator；也可以实例化简单类型，如 Integers 和 Strings:

VB.NET

```
Dim myCalculator As Calculator = New Calculator()  
  
Dim age As Integer = New Integer()
```

C#

```
Calculator myCalculator = new Calculator();  
  
int age = new int();
```

因为创建 Integer(在 C#中是 int)和 String(在 C#中是 string)这样的简单类型的变量是非常普遍的事，因此编译器允许省略 new 关键字和赋值代码。因此，下面的代码在功能上等价于前面的 age 声明。

VB.NET

```
Dim age As Integer
```

C#

```
int age;
```

本章开头列出的所有数据类型，除 System.Object 外都能不用 New 关键字来创建。
一旦创建了对象的实例，如myCalculator对象，它就可以使用了。例如，可以访问它的方法和属性来对对象做一些有用的事。但是在学习方法与属性之前，需要先了解类。

2. 类

类是对象的蓝图。正如可以用一个蓝图来构建一组相似的房子一样，也可以用一个类来创建该类的多个实例。因此类是应用程序中使用的对象的定义。类的最基本形式如下所示：

VB.NET

```
Public Class ClassName  
  
End Class
```

C#

```
public class ClassName  
{  
}
```

由于这段代码仅定义了一个空类，因此它无法做任何有用的事。为了给它添加一些行为，可以赋予它属性、方法和构造函数。此外，可以让类从一个现有类中继承，使它在功能和行为方面有个基础。下面两节将介绍这些术语。

3. 属性

对象的属性是对象拥有的特征。以 Person 对象为例。Person 对象有何种属性？说出它的很多不同的特征很容易，但最通用的有以下几个：

- 名字

- 姓
- 生日

属性在类中使用 **Property** 关键字(在 VB.NET 中)或者在 C# 中使用类似于方法的属性标题来定义。在这两种语言中，都使用一个 **Set** 块(在 C# 中是 **set**)和一个 **Get** 块(在 C# 中是 **get**)来定义所谓的属性的 **setter** 和 **getter**。当一个对象需要某个特定属性值时访问 **getter**，而 **setter** 用来赋予属性一个值。属性仅提供对存储在对象中的底层数据的访问；它们不包含实际数据。为了存储数据，需要后台变量(**backing variable**)。这通常是在能够存储外部属性的值的类中定义的简单变量。在下面的示例中，变量 `_firstName` 是 `FirstName` 属性的后台变量：

VB.NET

```
Public Class Person
    Private _firstName As String
    Public Property FirstName() As String
        Get
            Return _firstName
        End Get
        Set(ByVal value As String)
            _firstName = value
        End Set
    End Property
End Class
```

C#

```
public class Person
{
    private string _firstName;
    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }
}
```

私有后台变量前面常常要加上一个下划线，后面跟着的第一个单词全部用小写，后面还可以跟着以大写字母开头的更多单词。因此 `FirstName` 属性有一个名为 `_firstName` 的后台变量，`LastName` 有一个名为 `_lastName` 的后台变量等。这样，应用到整个类中的所有变量都被很好地包装到了 **IntelliSense** 列表中。只要在代码中输入一个下划线，就会得到私有变量的一个完整列表。注意，当在函数或子例程中定义变量时，不要使用下划线。

和之前看到的 **Public** 关键字一样，**Private** 也是一个访问修饰符。在本章后面会介绍关于访问修饰符的更多内容。

类中需要属性的主要原因是为了封装数据。其思想是属性允许控制正在被赋值的数据。这样，就能在将数据存储到底层后台变量中之前对它进行有效性验证和操纵。设想应用程序的业务规则之一指出所有名字必须将首字母大写。在非面向对象的语言中，设置名字的开发人员就必须在每次使用名字填充变量时都要记住这个规则。但是在 **OO** 方法中，可以让 `FirstName` 属性负责这个规则，因此其他人不用再关心它。可以在属性的设置器中进行这种数据操作：

VB.NET

```
Set(ByVal value As String)
    If Not String.IsNullOrEmpty(value) Then
        _firstName = value.Substring(0, 1).ToUpper() & value.Substring(1)
    Else
        _firstName = String.Empty
    End If
End Set
```

C#

```
set
{
    if (!string.IsNullOrEmpty(value))
    {
        _firstName = value.Substring(0, 1).ToUpper() + value.Substring(1);
    }
    else
    {
        _firstName = string.Empty;
    }
}
```

上面的代码说明在 VB.NET 和 C#中，value 参数是可访问的(正如方法的参数一样)。Value 参数中包含了要赋值给属性的值。在 VB.NET 中，value 参数是在属性的设置器中显式定义的。C#中没有显式地指定它，但是仍然能访问它。

此代码首先使用方便的 String.IsNullOrEmpty 方法检查正在传递的 value 是否不为 Nothing (C#中是 null)，而且不包含空字符串。

然后，If 块中的代码用 String 类的 SubString 方法(传递值 0 和 1)取走 value 的第一个字母。0 表示子串的开头，而 1 表示必须返回的字符串的长度。字符串索引也是基于 0 的，因此表示开始的 0 和长度 1 有效地返回了 value 参数的第一个字符。然后用 ToUpper()方法将这个字母改为大写。最后，代码会再次用 SubString 取走 value 参数的其余部分，并将合并后的名称返回给后台变量。

现在可以使用代码设置任意大小写的名字。但是，当试图再次访问此名字时，名字总会以恰当的的第一个字符开头：

VB.NET

```
Dim myPerson As Person = New Person() ' Create a new instance of Person
myPerson.FirstName = "imar"            ' Accessing the setter that changes the value

Label1.Text = myPerson.FirstName       ' Accessing the getter that now returns Imar
```

C#

```
Person myPerson = new Person();        // Create a new instance of Person
myPerson.FirstName = "imar";            // Accessing the setter that changes the value

Label1.Text = myPerson.FirstName;      // Accessing the getter that now returns Imar
```

对于不需要任何数据操作或有效性验证的简单属性，可以使用所谓的自动属性。该属性仅在 C#中可用，但是随着.NET 4 的发布，自动属性也能够在 VB.NET 中使用了。使用这些属性，就能使用精简得多的语法，不需要私有后台变量。当编译代码时，编译器会创建必要的后台变量，并且您将需

要引用公有属性。下面是为自动属性编写的 `Person` 类的 `DateOfBirth` 属性：

VB.NET

```
Public Property DateOfBirth As DateTime
```

C#

```
public DateTime DateOfBirth { get; set; }
```

自动属性在 `Visual Basic` 中的实现与其 `C#` 版本相比，有一个好处：可以一次性地声明属性并为其赋值。下面的代码段定义了一个 `CreateDate` 属性，并为其指定当前的日期和时间。

VB.NET

```
Public Property CreateDate As DateTime = DateTime.Now
```

为了给 `C#` 版本中的自动属性指定一个默认值，需要使用构造函数(稍后介绍)。

如果以后发现需要在属性的获取器或设置器中写代码，就可以轻松地在不中断现有应用程序的情况下扩展相关代码块。到那时，属性定义仍然整洁美观，不会把类弄乱。

设置只读和只写属性

有时候设置只读或只写属性相当有用。例如，如果对象的 `ID` 是由数据库自动指定的，它就可能是只读的。如果对象是从数据库中构造的，它的 `ID` 就被指定为私有后台变量。然后就会将公有 `Id` 属性设置为只读来停止调用代码，以防不小心改动它。同样，也可能为了安全性的原因设置只写属性。例如，在 `Person` 对象上可能有一个 `Password` 属性，只能给它指定一个值，以后不能再读它。在类的内部，代码仍然能访问后台变量以使用密码值。另一个可用作只读属性的候选是返回数据组合的属性。如 `Person` 类的 `FullName` 属性，该属性返回的是 `FirstName` 属性和 `LastName` 属性的组合。使用每个属性的设置器来指定数据，但是可以包含一个返回连接后的值的只读属性

`C#` 中的只读或只写属性很简单：只要省略设置器(对于只读属性)或获取器(对于只写属性)即可。`VB.NET` 要稍微复杂一些，它要求显式地指定关键字 `ReadOnly` 或 `WriteOnly`。下面的代码段显示了一个 `VB.NET` 和 `C#` 中都包含的只读的 `FullName` 属性。

VB.NET

```
Public ReadOnly Property FullName() As String
    Get
        Return _firstName & " " & _lastName
    End Get
End Property
```

C#

```
public string FullName
{
    get { return _firstName + " " + _lastName; }
}
```

当试图向只读属性指定一个值时，可能在 `VWD` 中得到一个错误。
与属性相似，对象也可以有方法。

4. 方法

如果说属性是类拥有的东西(它的特征)，那么方法就是类能做的事或可执行的操作。例如，汽

车类的特征有 Brand、Model 及 Color 等。它的方法可以是 Drive()、Brake()及 OpenDoors()。方法为对象提供了行为，使对象能做一些事。

在本章前面讨论使用子例程和函数编写组织代码时，已经看到了很多可用的方法。只要在类的起始和结束元素之间写一个函数或子例程，就可以向类添加一些方法。例如，假设 Person 类有一个 Save 方法，用于将对象本身持久化到数据库中。此方法的签名可能如下所示：

```
VB.NET
Public Class Person
    Public Sub Save()
        ' Implementation goes here
    End Sub
End Class

C#
public class Person
{
    public void Save()
    {
        // Implementation goes here
    }
}
```

如果要调用 Save 方法来让 Person 对象把自身保存到数据库中，就要创建它的一个实例，设置相关属性(如 FirstName)然后调用 Save:

```
VB.NET
Dim myPerson As Person = New Person()
myPerson.FirstName = "Joop"
myPerson.Save()

C#
Person myPerson = new Person();
myPerson.FirstName = "Joop";
myPerson.Save();
```

然后 Save 方法就会知道如何把 Person 保存在数据库中。
注意 Person 类的新实例是用 New(C#中是 new)关键字加上类名来创建的。当激活此代码时，它就会调用对象的构造函数。该构造函数用来创建对象的实例。

5. 构造函数

构造函数是类中的特殊方法，用来帮助创建对象的实例。一旦要创建类的实例，就会运行构造函数，因此它们是将对象初始化为某个默认状态的极佳位置。前面已经介绍过用 New(C#中是 new)关键字创建对象的新实例：

```
VB.NET
Dim myCalculator As Calculator = New Calculator()
```

C#

```
Calculator myCalculator = new Calculator();
```

New 关键字后面跟着对象的构造函数：类的名称。默认情况下，当在 VWD 中创建一个新类时，对于 C#就会得到一个默认构造函数，而 VB.NET 则没有。不过，那并不是真正的问题，因为如果缺少默认构造函数，编译器就会生成一个默认的构造函数。默认构造函数没有参数，在 C#中采用的是类名，而在 VB.NET 中采用的是保留关键字 New：

VB.NET

```
Public Class Person
    Public Sub New()

    End Sub
End Class
```

C#

```
public class Person
{
    public Person()
    {

    }
}
```

虽然这个默认构造函数对于创建类的标准实例已经足够了，但是有时候需要能够预先向类中发送一些信息，这样一旦构造好就可以使用。例如，使用 Person 类，向构造函数传递名字、姓和生日是很有用的，这样可以使数据在以后立即可用。为了使这种情况发生，可以创建一个重载构造函数。重载构造函数或方法实质上是具有相同名称的现有方法的副本，只是方法签名不同。为了让构造函数接受名字和生日，需要使用下面的代码：

VB.NET

```
Public Sub New(ByVal firstName As String, ByVal lastName As String,
               ByVal dateOfBirth As DateTime)
    _firstName = firstName
    _lastName = lastName
    _dateOfBirth = dateOfBirth
End Sub
```

C#

```
public Person(string firstName, string lastName, DateTime dateOfBirth)
{
    _firstName = firstName;
    _lastName = lastName;
    _dateOfBirth = dateOfBirth;
}
```

有了这段代码，就可以创建一个新的 Person 对象：

VB.NET

```
Dim myPerson As Person = New Person("Imar", "Spaanjaars", New DateTime(1971, 8, 9))
```


C#

```
Person myPerson = new Person("Imar", "Spaanjaars", new DateTime(1971, 8, 9));
```

这个构造函数接受传递给它的值，并将该值指定给私有后台变量，这样在该行代码后就会完全初始化 myPerson 对象。

Visual Basic 支持一种略微不同的语法，即使用 Dim myVariable As New ClassName 语法来一次性地声明和初始化一个对象。可使用如下所示的代码实例化 Person 实例，效果与前面的代码是一样的：

```
Dim myPerson As New Person("Imar", "Spaanjaars", New DateTime(1971, 8, 9))
```

除了重载构造函数外，.NET 还提供了另一种创建对象与初始化一些属性的快捷方式：对象初始化器(object initializer)。有了对象初始化器，就可以在声明对象实例的同时提供一些属性的初始值。下面的代码将创建一个 Person 对象并赋予 FirstName 和 LastName 属性一个值：

VB.NET

```
Dim myPerson As New Person() With {.FirstName = "Imar", .LastName = "Spaanjaars"}
```

C#

```
Person myPerson = new Person() { FirstName = "Imar", LastName = "Spaanjaars" };
```

在 VB.NET 中，需要在属性列表前面使用 With 关键字。此外，还需要在各属性名前面加上一个句点(.)作为前缀。除此之外，两种语言的语法基本上是相同的。如果需要很快设置对象的一串属性，并且不用编写构造函数的专用重载版本时，对象初始化器非常有用。

尽管在应用程序中有这个 Person 类很有用，但是还是有时候可能需要 Person 的专用版本。例如，应用程序可能需要像 Employee 和 Student 这样的类。在这种情况下应怎么办？要创建 Person 类的两个副本，然后分别将它们命名为 Employee 和 Student 吗？

虽然这种方法会有用，但是它有一些重大缺陷。最大的问题在于代码重复。如果决定添加一个 SocialSecurityNumber 属性，现在就需要在多个地方添加它：在一般的 Person 类中，以及在 Employee 和 Student 类中。对象继承(面向对象编程的重要支柱)就是为了解决这个问题而设计的。

6. 继承

前面已指出过，System.Object 是.NET 中所有其他数据类型(包括所有内置类型和自定义类型)的父类型，这意味着.NET 中的每个类型(Object 类型本身除外)都继承自 Object 类。继承的一个好处是能在一个高级别(例如 Object 类)上定义某个行为，这个行为可以由继承类自动完成，不需要重复写该代码。在.NET Framework 中，Object 类定义所有其他对象都会继承的一些成员，包括 ToString() 方法。

为了使一个类继承另一个类，需要在 VB.NET 中使用 Inherits 关键字，在 C#中使用冒号(:)，如下面的代码所示，这段代码定义了一个继承自 Person 类的 Student 类。

VB.NET

```
Public Class Student
    Inherits Person
```

C#

```
public class Student : Person
{
}
```

为了理解继承的工作原理，可以再次考虑前面示例中出现的 Person 类。这个类有几个属性，如 FirstName 和 LastName，以及一个 Save 方法。但是，如果它继承自 Object，那么它也有 ToString() 方法吗？当然。图 5-8 显示了继承自 Object 的 Person 类与 Object 类之间的关系。

图 5-8 显示了继承自 Object 类的 Person(箭头指向的是被继承的类),这意味着 Person 实例能够做 Object 类可以做的任何事。如在 Person 对象上调用 ToString()方法:

```
Label1.Text = myPerson.ToString() ' Writes out Person
```

定义在 Object 类中的 ToString()方法的默认行为是指出它自己的类名。在上例中，它意味着 Person 类继承了这个行为，因此指出 Person 作为它的名称。通常，这种默认行为并不够，比如说，如果 Person 能返回它代表的人的全名就会有用得多。可以通过重写 ToString()方法来轻松地实现这一点。重写方法或属性能有效地重定义类从其父类中继承的行为。要重写方法，在 VB.NET 中使用关键字 Overrides，而在 C#中用 override。下面的代码段重定义了 Person 类中 ToString 的行为。

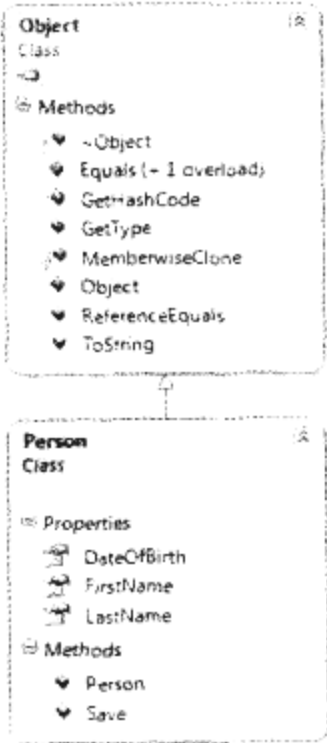


图 5-8

VB.NET

```
Public Overrides Function ToString() As String
    Return FullName & ", born at " & _dateOfBirth.ToShortDateString()
End Function
```

C#

```
public override string ToString()
{
    return FullName + ", born at " + _dateOfBirth.ToShortDateString();
}
```

这样定义了 Person 类中的 ToString()方法后，它就不再返回单词 Person，而是会返回它代表的人的全名:

```
Label1.Text = myPerson.ToString() ' Imar Spaanjaars, born at 8/9/1971
```

注意代码如何使用只读的 FullName 属性来避免再次对合并两个名称的逻辑进行编码。不能只重写自己感觉合适的方法。对于可重写的方法，父类需要使用关键字 virtual(C#)或 Overridable(VB.NET)来标记该成员。

.NET 中的对象继承允许创建对象的一个层次结构，来增强或增加其他对象的功能。这样就可以从通用基类(Object)出发。然后，其他类可以从这个类继承，再增加一些专用的行为。如果需要更多的专用类，可以再从继承自 Object 的类继承，因而创建一个类的层次结构以保持较好的专用性。这个原则适用于.NET Framework 中的很多类，包括 Page 类。您也许没有看出来，实际上在 VWD 中

创建的每个 ASPX 页面都是一个继承自类 `System.Web.UI.Page` 的类。这个 `Page` 类又继承自 `TemplateControl`，它继承自 `Control` 类，`Control` 又继承自 `Object` 类。整个层次结构如图 5-9 所示。最下方是类 `MyWebPage`，它可以是 Web 页面的 `Code Behind` 类，如 `MyWebPage.aspx`。

从图 5-9 中可以看出，`TemplateControl` 是一个抽象类——不能实例化的类；即不能用 `New`(在 C#中是 `new`)来创建它的一个新实例。它仅作为其他可以继承自它的类(如 `Page`)的通用基类。`Page` 和 `Object` 之间的这些类在这个阶段并不真正相关。重要的是页面继承了 `Page` 类拥有的所有行为。所有 ASPX 页面继承自 `Page` 这一事实比一开始想象的更有用。因为它继承自 `Page`，所以就可以自由地获得这个类中定义的大量属性与方法。例如，`Page` 类提供了一个 `Title` 属性，当设置它时最终会以 `<title>` 元素出现在页面中。页面只需设置这个属性即可，作为父类的 `Page` 类会处理余下的事情。

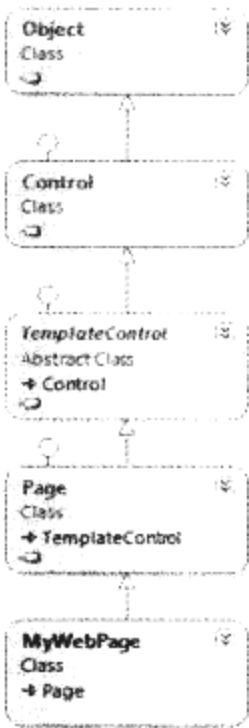


图 5-9

```
VB.NET
Title = "Beginning ASP.NET 4 in C# and VB from Wrox"

C#
Title = "Beginning ASP.NET 4 in C# and VB from Wrox";
```

在第 6 章创建 `BasePage` 类时，将用到继承，`BasePage` 类是在 Planet Wrox Web 站点中创建的多数页面的父类。

在前面的这些示例中，包括 `ToString()` 方法的重写，出现了关键字 `Public`。此外，当创建后台变量时，出现了关键字 `Private`。这些关键字称为访问修饰符，它们确定了代码的可见性。

7. 访问修饰符

本章前面曾提到 OO 的一个核心概念是封装。通过创建函数及属性等成员，创建了一个负责实现的对象。其他对象与这个对象交互时把那些方法与属性作为黑盒对待。即它们传进去一些数据，可以有选择地预期收回一些结果，而对这些方法的具体工作过程并不感兴趣；这就像做广告。因此为了让一个对象屏蔽它的一些内部操作，需要一种控制对类型与成员的访问的方法。可以通过在类、属性或方法名前面指定一个访问修饰符来做到这一点。表 5-7 列出了 C#和 VB.NET 中可用的访问修饰符，并解释了它们的作用。

表 5-7

C#	VB.NET	说 明
public	Public	可以从任何地方访问类或成员，包括当前应用程序之外的代码
protected	Protected	带受保护访问修饰符的代码只能在定义它的类型内或者继承自它的类型内使用。例如，在 <code>Page</code> 类内定义的一个受保护成员对于 ASPX 页面是可访问的，因为它继承自 <code>Page</code>
internal	Friend	将代码的可访问性限制为同一个程序集(assembly)中的其他代码。程序集是一组(可以是一个或多个)编译过的包含可重用.NET 代码的代码文件(可以是.exe 或.dll 文件)

(续表)

C#	VB.NET	说 明
private	Private	只能在定义它的类的内部访问类或成员。例如，在 Person 类中，_firstName 变量只能从 Person 类中访问。其他代码(如 ASPX 页面)无论如何都不能直接访问这个字段，而是需要访问公有 FirstName 属性来获得或设置一个人的名字

在这 4 个访问修饰符中，只有 `protected` 和 `internal`(在 VB 中是 `Protected` 和 `Friend`)能合并起来使用。其他两个修饰符都必须单独使用。通过合并 `protected` 和 `internal`，可以创建只能由当前类和当前程序集中继承自它的任何类访问的成员。

和其他面向对象的概念一样，不需要花费大量时间来指定在代码中的访问修饰符。然而，最好了解它们的存在性以及它们的作用。这样，就可以大致明白为什么有时候类不会显示在 IntelliSense 列表中。如果出现这种情况，很可能是忘记了在类上指定 `public` 访问修饰符(VB.NET 中是 `Public`)。其默认值是 `internal`(VB.NET 中是 `Friend`)，它使得类对在相同程序集(通常是一个带有.dll 扩展名的文件)中的其他类可见，而对程序集外的代码隐藏。在类定义前面添加关键字 `public` 或 `Public` 就能解决这个问题。

5.5.2 事件

本章要讨论的最后一个重要主题是事件。ASP.NET 是一种事件驱动环境，即代码能基于应用程序中发生的特定事件执行。事件由应用程序中的特定对象触发，然后由其他对象处理。.NET Framework 中有很多对象可以触发事件，甚至能向所写的类中添加自己的事件。

为了处理某个对象触发的事件，需要写一个事件处理程序，它实质是上一个有特殊签名的常规方法。可以使用事件布线(wiring)语法将该事件处理程序与事件相关联，不过大部分时候 VWD 会处理它的。当对象(如 Web 页面中的控件)触发一个事件时，可能需要将额外的信息传递给事件处理程序，以向它通知关于引发或影响事件的相关数据。可以用事件参数类(类 `System.EventArgs` 或者继承自它的任何类)发出此信息。

要了解这些术语是如何相互配合的，考虑一下在 Web 页面中单击一个按钮时发生了什么事。当单击它时，浏览器中的客户端按钮引发了一个回发。在服务器上，`Button` 控件看到它在浏览器中被单击了，然后触发了它的 `Click` 事件。就好像按钮说：“大家看啊。我刚刚被单击了。要是你们谁有兴趣，我这里有详细信息。”通常，对按钮的 `Click` 事件感兴趣的代码需要一个事件处理程序来处理单击。可以通过在设计器中双击 `Button` 来创建它的一个事件处理程序。此外，也可以在列出了 `Events` 选项卡的控件的 `Properties` 面板上双击相关事件(如图 5-10 所示)，这个选项卡可以通过按下工具栏上带有闪电图标按钮来打开。

如果双击 `Design` 视图中的控件或者 `Properties` 面板中的事件名，VWD 就会写好事件处理程序的代码。下面的代码段显示了 VB.NET 和 C# 中 `Button` 控件的单击处理程序：

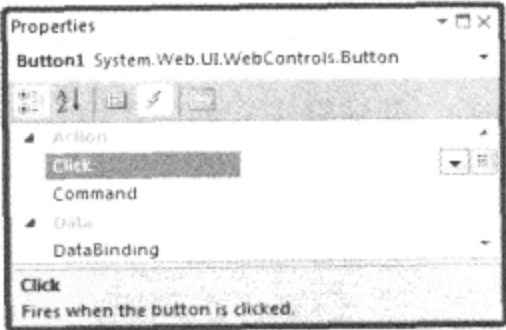


图 5-10

VB.NET

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Button1.Click
End Sub
```

C#

```
protected void Button1_Click(object sender, EventArgs e)
{
}
```

在 VB.NET 示例中，有一个带一些参数的标准方法，后面跟着 `Handles Button1.Click`。这是使 `Button` 控件的 `Click` 事件与 `Button1_Click` 方法关联的事件布线代码。现在，每当单击按钮时，就会执行 `Button1_Click` 内的代码。

C#版本中没有这个 `Handles` 关键字。取而代之的是，在 C#中 `VWD` 向页面的标记中的 `Button` 控件添加了下列突出显示的代码：

```
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" />
```

有了这块标记，编译器就会生成必要的代码将 `Button1_Click` 方法链接到按钮的 `Click` 事件。在运行时将看到完全相同的行为：当单击按钮时，执行 `Button1_Click` 中的代码。

还可以看出这个 `Button1_Click` 事件处理程序有两个参数：一个名为 `sender` 的 `Object` 和另一个名为 `e` 的 `EventArgs`。这是标准的.NET 命名模式，后面跟着生成事件的所有对象。发送者参数包含对触发事件的对象的引用；本例中是 `Button1`。这样万一将多个事件关联到了相同的事件处理程序上时，就能找出是哪个对象触发了事件。

第二个参数是 `EventArgs` 类的实例，它向事件提供了额外的参数。由于单击了按钮后没有别的相关数据要提交，因此使用了简单的空 `EventArgs` 类。然而，在以后的章节中(如介绍数据驱动的 Web 窗体的第 9 章)，将介绍一些激活事件的含较丰富信息的类的示例。

事件的概念是关于面向对象编程部分的最后一个概念。通过这一部分的学习，您一定熟悉了面向对象编程中最重要的术语。在本书后面章节中将介绍这些概念的实用示例。

5.6 关于编程的实用提示

下面列出了关于编程的一些实用提示：

- 总是赋予变量有意义的名称。对于简单的循环计数器，可以用 `i`，尽管 `loopCount` 可能会更好地描述变量的用途。不要在变量前加上单词 `var` 作为前缀。所有变量都是变量，因此在代码中加上 `var` 是画蛇添足。尽可能使用有用的名称，如对私有字段使用 `_firstName` 和 `_categoryId`，而不要用 `strFName` 或 `catI`，对公有属性和类则分别用 `FirstName` 和 `Person`。
- 试验再试验。在使用控件和 ASPX 页面时，了解如何编程的最佳方法是实际操作。输入一些代码并按下 `Ctrl+F5` 组合键看看代码的行为。如果某个地方出错了，编译器会发出滴滴声，并提供关于如何解决问题的有用提示。不要怕把代码弄乱；尽可能进行各种尝试，直到代码能完成您希望它做的事。如果代码不能工作，可以查看第 18 章。该章讨论了有关调试的内容，您可以从中发现有用的提示，从而找到并更正代码中可能出现的许多错误。

- 在写函数或子例程时，尽量最小化代码的行数。通常，超过 40~50 行代码的方法被看作是糟糕的设计。当看到这样的代码时，可以考虑选择将某些部分移到它们自己的例程中。这样可以使代码更容易阅读与理解，使得代码出现 bug 的机会更小。即使某个方法仅使用一次，在一个单独的方法中放一大块代码还是会大大增加代码的可读性和条理性。
- 在代码中写注释时，要描述代码的大意而不要解释明显的语句。例如，下面的注释(实际代码中常常看到)就完全没有价值：

```
Dim loopCount As Integer = 0      ' Declare loopCount and initialize it to zero
```

任何稍有编程经验的人都能看出这段代码的功能。

5.7 本章小结

尽管编程可能很复杂，但是需要了解的基础知识却相当容易掌握。关于编程的有趣事情是，要写出有用的程序并不需要成为专家。只要从简单的 Hello World 示例开始，然后每次增加一些关于代码的知识即可。

本章包括两个主要话题。第一个话题介绍了在.NET 中使用 C#或 VB.NET 编程的有关内容。从中我们可以知道什么是数据类型和变量，并且学习了运算符、决策和循环。我们还了解了如何使用函数、子例程和名称空间编写有条理的代码，以及如何向代码中添加注释。

本章最后一部分介绍了面向对象。虽然面向对象编程本身是一个非常大的主题，但它的基础知识很容易获得。本章介绍了面向对象编程的基本元素：类、方法、属性与构造函数。还介绍了继承，它是面向对象设计的后台驱动力量。

第6章将介绍如何创建外观一致的 Web 页面，在创建作为 Planet Wrox 项目中大多数 Code Behind 类的父类的 BasePage 类时将再次用到继承。

5.8 练习

1. 世界上寿命最长的人能活到 122 岁，用什么样的数据类型存储人的年龄最好呢？如果您提出了更好的存储年龄的数据类型，一定能得到奖励。
2. 下面的代码的作用是什么？

VB.NET

```
DeleteButton.Visible = Not DeleteButton.Visible
```

C#

```
DeleteButton.Visible = !DeleteButton.Visible;
```

3. 假定存在下面的类 Person，则包含字符串属性 StudentId 的一个新类 Student 的代码会是什么样？利用继承创建这个新类。

VB.NET

```
Public Class Person
```



```
Public Property Name As String
End Class
```

```
C#
public class Person
{
    public string Name { get; set; }
}
```

练习的答案见附录 A。

本章要点回顾

类	编程语言中对象的蓝图
集合	特殊的数据类型，可以同时保存多个对象
封装	对外界隐藏类的内部操作和数据，以便更好地管理和保护该数据
实例化	在内存中基于类型定义创建一个新对象的过程
方法	在对象上进行的操作，如汽车类中的 Brake()
名称空间	以层次方式构建类和其他类型的方法
面向对象	编程的一种流行方式，在这里软件被建模为一个能相互影响的对象集
重写	在子类中重定义从父类中继承的某个成员
属性	对象的一个特征，如人的名字

第6章

创建外观一致的 Web 站点

本章要点

- 如何使用母版页和内容页定义 Web 页面的整体外观
- 如何使用集中的基页定义站点中所有页面的共同行为
- 如何创建主题以定义站点的全局外观，让用户在运行时可以选择他们喜欢的主题
- 如何创建外观，以便快速地在站点范围内改变控件布局
- 什么是 ASP.NET 页面的生命周期以及它为什么重要

在构建 Web 站点时应该努力使布局和行为尽可能保持一致。一致性能让站点显得比较专业，而且可以帮助访问者找到浏览站点。幸运的是，ASP.NET 4 和 Visual Web Developer 2010 提供了许多实现一致设计的优秀功能和工具，来帮助立即创建漂亮的页面。

前几章已经介绍过如何使用 VWD、HTML、CSS 和服务控件来可视化地创建 Web 页面。第 5 章介绍了 ASP.NET 编程。本章首次将这些概念结合起来，以展示如何使用编程代码去改变站点的外观。

下一节将介绍如何创建定义页面全局外观的母版页。然后，站点中的 ASPX 页面就能使用该母版页，而不需要重写布局。本章其余部分的介绍都建立在母版页的基础上。

6.1 用母版页创建一致的页面布局

对于大部分 Web 站点来说，当从一个页面转向另一个页面时，只有页面的部分会发生变化。有些部分通常不会改变，包括页眉、菜单和页脚等公共区域。为了创建布局一致的 Web 页面，需要用某种方式在单个模板文件中定义这些相对静态的区域。ASP.NET 在 2.0 之前的版本没有模板解决方案，因此不得不在 Web 站点的每个页面上重复进行页面布局，或者求助于奇奇怪怪的编程技巧。幸运的是，现在有了母版页(master page)，因此不会再出现这种情况。母版页的最大好处是它们可以在单个地方定义站点中所有页面的全局外观。这意味着如果要修改站点的布局——比如要把菜单从左边移到右边——只需修改母版页，基于此类母版页的页面就会自动进行相应的修改。

当 ASP.NET 2.0 中引入母版页时，它们很快得到了开发人员社团的欢迎，他们把它作为 ASP.NET 页面的模板解决方案，因为它们非常容易使用。更让人兴奋的是，VWD 有了极佳的设计时支持，在开发过程中可以在设计时创建和查看页面，而不是只能在运行时从浏览器中浏览。

在某种程度上，母版页看起来就像正常的 ASPX 页面。它包含静态 HTML，比如<html>、<head>和<body>标记，它也可能包含其他 HTML 和 ASP.NET 服务器控件。在母版页内，建立将在所有页面上重复的标记(markup)，比如页面和菜单的总体布局。

然而，母版页并不是真正的 ASPX 页面，不能直接在浏览器中请求；它只能作为实际 Web 页面(称为内容页)所基于的模板。

与第 4 章提到的@ Page 指令不同，母版页使用@ Master 指令将文件标识为母版页。

VB.NET

```
<%@ Master Language="VB" %>
```

C#

```
<%@ Master Language="C#" %>
```

与正常 ASPX 页面一样，母版页可以有一个 Code Behind 文件，用它的 CodeFile 和 Inherits 属性标识。

VB.NET

```
<%@ Master Language="VB" CodeFile="Frontend.master.vb"
    Inherits="MasterPages_Frontend" %>
```

C#

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Frontend.master.cs"
    Inherits="MasterPages_Frontend" %>
```

为了创建内容页可以填充的区域，需要在页面中定义 ContentPlaceHolder 控件，代码类似下面所示：

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
```

尽管通常只需要几个占位符就可以创建灵活的页面布局，但是实际上想创建多少就可以创建多少。

内容页(本质上是正常的 ASPX 文件，不过没有像<html>、<head>、<body>和<form>标记这样的常见代码)用 Page 指令的 MasterPageFile 属性连接到母版页。

VB.NET

```
<%@ Page Title="" Language="VB" MasterPageFile="~/MasterPages/Frontend.master"
    AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default">
```

C#

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master"
    AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default">
```

然后，页面特有的内容将被放到指向相关 ContentPlaceHolder 的<asp:Content>控件内。

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceholder1"
    Runat="Server">
</asp:Content>
```

注意，指向 ContentPlaceholder 的 Content 控件的 ContentPlaceHolderID 属性在母版页中定义。目前它指向默认名称 ContentPlaceholder1，不过在以后的练习中会介绍如何修改它。

当请求页面时，母版页和内容页的标记在运行时会合并起来，经过处理后发送给浏览器。图 6-1 显示了母版页和内容页结合产生最终页面并发送到浏览器的图表。

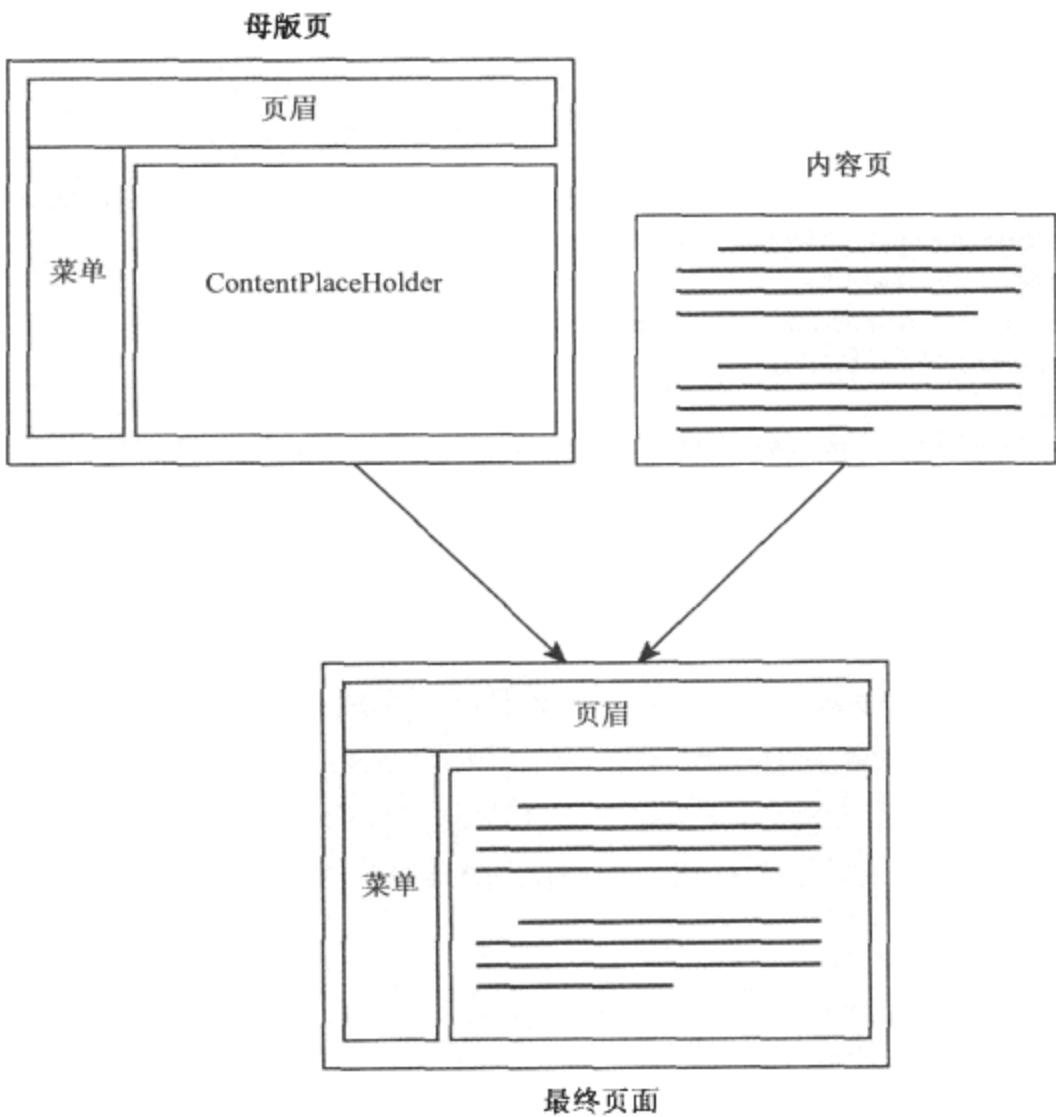


图 6-1

为了看到这个过程是如何实现的，下面介绍如何创建母版页和内容页。

6.1.1 创建母版页

母版页用 Add New Item 对话框添加到站点中。它们可以位于站点中的任何位置，包括根文件夹中，但是从组织性的观点来看，通常比较容易把它们存储在单独的文件夹中。正如正常 ASPX 页面一样，它们支持内联代码模型，也支持 Code Behind 模型。Planet Wrox 项目中所用的母版页却仅使用 Code Behind 模型。在下面的“试一试”练习中，将介绍如何创建一个简单的母版页，并向其中添加一些 HTML 来定义 Web 站点中页面的总体结构。

试一试

创建母版页

- (1) 如果还没有在 VWD 中打开 Planet Wrox 项目，先把它打开。

(2) 在第 2 章曾创建了一个文件夹 MasterPages 来保存母版页，然后向该文件夹中添加了单个的母版页。如果没有做那个练习，现在就添加这个母版页。要添加母版页，创建 MasterPages 文件夹，右击这个新文件夹，选择 Add New Item 命令，并选择 Master Page。确保母版页使用了 Code Behind 模式，而且选择了首选的编程语言。将母版页命名为 Frontend.master。

(3) 在母版页的<form>标记之间添加下面突出显示的代码，替换掉<div>标记与创建母版页时 VWD 添加的 ContentPlaceHolder。注意，除了 Header<div>中的<asp:ContentPlace Holder>元素和<a>元素外，这与第 3 章中添加到 Default.aspx 的代码基本相同。<a>元素把用户带回主页，并且以后会被样式化：

```
<form id="form1" runat="server">
  <div id="PageWrapper">
    <div id="Header"><a href="/" runat="server">Header Goes Here</a></div>
    <div id="MenuWrapper">Menu Goes Here</div>
    <div id="MainContent">
      <asp:ContentPlaceHolder ID="cpMainContent" runat="server">
      </asp:ContentPlaceHolder>
    </div>
    <div id="Sidebar">Sidebar Goes Here</div>
    <div id="Footer">Footer Goes Here</div>
  </div>
</form>
```

确保 MainContent <div>标记内有 ContentPlaceHolder。可以从 Toolbox 中拖一个到页面上，或者用 IntelliSense 的帮助提示直接输入代码。在这两种情况下都应该设置控件的 ID 为 cpMainContent。

(4) 接下来，将母版页切换到 Design 视图中，然后从 Solution Explorer 的 Styles 文件夹中拖动文件 Styles.css 到母版页上。一旦放下文件，VWD 就会更新 Design 视图，并将站点的布局显示为第 3 章中创建的样子。如果设计没有变化，请切换到 Markup 视图并确保页面的页头中有一个指向 CSS 文件的<link>标记：

```
<asp:ContentPlaceHolder ID="head" runat="server">
</asp:ContentPlaceHolder>
<link href="../../Styles/Styles.css" rel="stylesheet" type="text/css" />
</head>
```

现在该页面在 Design 视图中看起来应如图 6-2 所示。

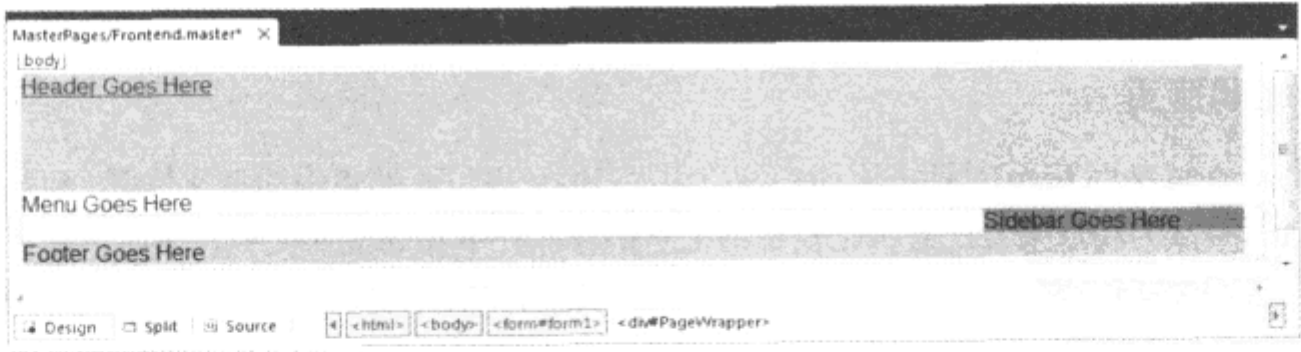


图 6-2

注意，Design 视图的菜单与页脚之间带紫色边框的区域。这是内容页所使用的 ContentPlaceHolder 控件。后面将看到其工作原理。

(5) 现在已创建了母版页，所以可以先保存并关闭这个母版页。

工作原理

在 Visual Web Developer 中，母版页的行为很像正常页面。可以向母版页添加 HTML 和服务控件，可以在 Markup 视图和 Design 视图中管理母版页。当然，最大的区别在于，母版页本身并不是真正的页面：它仅起站点中所有其他页面模板的作用。

在下一部分，将会看到如何将该母版页作为内容页的模板使用。

6.1.2 创建内容页

母版页如果没有内容页来使用它，就没有用处。通常仅有少量几个母版页，却可以有很多内容页。为了将一个内容页基于一个母版页，需要在添加一个新的 Web Form 到站点时，选中 Add New Item 对话框底部的 Select Master Page 选项。或者，可以在页面的 Markup View 中直接在页面上设置 MasterPageFile 属性。本章前面介绍母版页和内容页时已提到过@ Page 指令。

内容页中只能含有映射到母版页中的<asp:ContentPlaceHolder>控件的<asp:Content>控件。而这些控件又可以包含标准标记，比如 HTML 和服务控件声明。因为内容页中的整个标记需要用<asp:Content>标记括起来，所以不容易将现有 ASPX 页面转换为内容页。最容易的方式通常是将要保留的内容复制到剪贴板上，删除原页面，然后将基于母版页的新页面添加到 Web 站点上。添加了该页面后，就可以将标记粘贴到<asp:Content>标记内。通过下面“试一试”的练习可以看到其工作过程。

试一试

添加内容页

本“试一试”练习将说明如何向站点中添加基于以前创建的母版页的内容页。添加了内容页后就能向<asp:Content>区域中添加内容了。

(1) 在前面的练习中向项目中添加了标准 ASPX 页面，项目现在应“升级”为使用新的母版页。由于 VWD 没有将标准页面改为内容页的内置支持，因此需要手工将原 ASPX 页面中的内容复制到新页面中。如果想保留以前添加到 Default.aspx 中的欢迎文本，请将 MainContent<div>标记之间的所有 HTML 复制到剪贴板上(即以前创建的<h1>和两个<p>元素)，然后删除 Solution Explorer 中的页面 Default.aspx。接下来，在 Solution Explorer 中右击 Web 站点并选择 Add New Item 命令。选择适当的编程语言，单击 Web Form，命名为页面 Deault.aspx，然后选中对话框底部的 Place code in separate file 和 Select master page 的复选框，如图 6-3 所示。

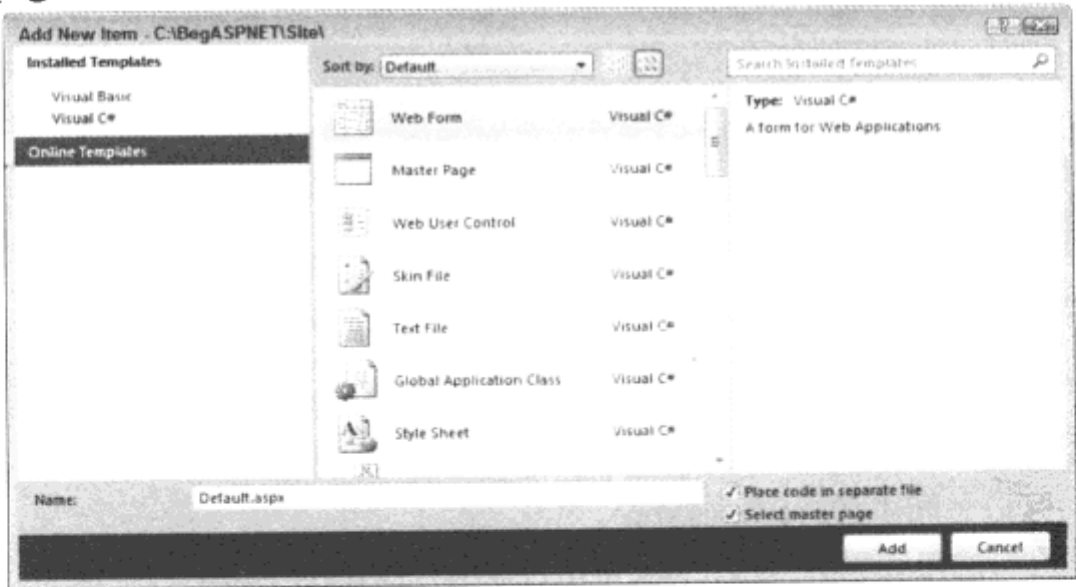


图 6-3

最后，单击 Add 按钮。

(2) 在 Select a Master Page 对话框中(如图 6-4 所示)，单击左边窗格中的文件夹 MasterPages，然后单击右边区域中的 Frontend.master。

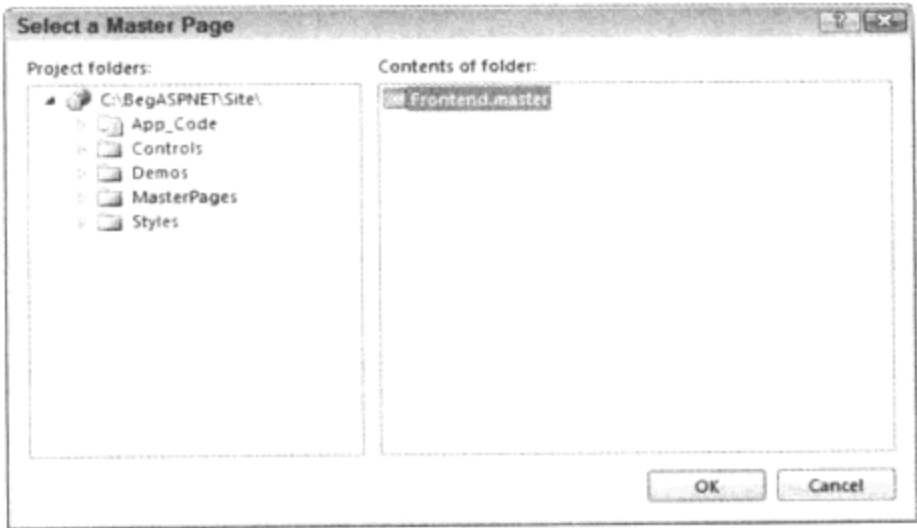


图 6-4

单击 OK 按钮向 Web 站点中添加页面。

在该 VB.NET 示例中，不会得到像标准 ASPX 页面那样带 HTML 的完整页面，而只会看到两个 <asp:Content>占位符：

```
<%@ Page Title="" Language="VB" MasterPageFile="~/MasterPages/Frontend.master"
    AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
</asp:Content>
```

(3) 切换到 Design 视图，这时会发现除了 cpMainContent 的<asp:Content>区域的所有内容都变灰了而且是只读的。图 6-5 显示了该页面的样子。

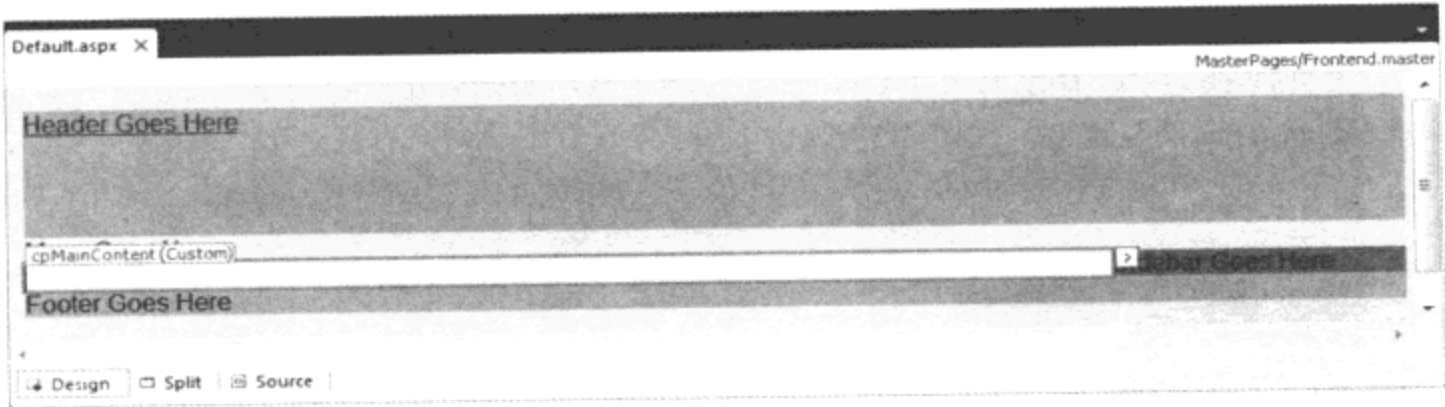


图 6-5

(4) 如果剪贴板上仍然有 Default.aspx 中的旧标记，则在 cpMainContent 占位符内再单击一次并按 Ctrl+V 键(注意：无论是在 Design 视图还是在 Markup 视图中都可以进行该操作)。这样就会向页面中<asp:Contnet>标记之间添加标记(markup)。

(5) 按 Ctrl+S 组合键保存所有修改，再按 Ctrl+F5 组合键在浏览器中打开页面。这时浏览器中显

示的页面应与在 Design 视图中看到的页面非常相似(见图 6-5)。

(6) 现在从浏览器中看一下页面的 HTML。可以通过右击页面并选择 View Source 或 View Page Source 命令来查看。注意浏览器中最终页面的源代码是母版页和内容页的源代码的合并代码。

```
<div id="PageWrapper">
  <div id="Header"><a href=".">Header Goes Here</a></div>
  <div id="MenuWrapper">Menu Goes Here</div>
  <div id="MainContent">
    <h1>Hi there visitor and welcome to Planet Wrox</h1>
    <p class="Introduction">We're glad you're paying a visit to
      <a href="http://www.PlanetWrox.com">www.PlanetWrox.com</a>,
      the coolest music community site on the Internet.
    </p>
    ...
```

前面 4 行来自母版页，而突出显示的 HTML 代码行来自内容页。

(7) 切换回 VWD 并基于母版页在站点根目录中创建一个新页面，命名为 Login.aspx。注意 VWD 是如何记住关于母版页和 Code Behind 的最终设置的(确保它们都被选中)。切换到 Markup 视图并使用文本 Log in to Planet Wrox 在 cpMainContent 占位符内创建<h1>元素。虽然还不需要向这个页面添加任何控件，但它却是第 16 章将要创建的登录功能的基础。如果 MainContent 元素中没有任何内容，Sidebar 将被移动到页面的左边。

(8) 回到页面 Default.aspx 并切换到 Design 视图。在带有 header 和两个<p>元素的欢迎文本下面，创建一个新段落(在 Design 视图中按 Enter 键)并输入一些文本(比如：You can log in here)。注意当 VWD 将先前的类自动应用到新段落时，该段落是如何获得名为 Introduction 的 class 属性的。使用 Apply Styles 窗口的 Clear Styles 选项删除该类，或者手动把它从 Markup 视图的代码中删除。

(9) 突出显示单词 log in 并从主菜单中选择 Format | Convert to Hyperlink 命令。在随后出现的对话框中，单击 Browse 按钮并选择刚创建的页面 Login.aspx。单击 OK 按钮两次。

(10) 保存所有修改并再次按 Ctrl+F5 组合键，以便在浏览器中浏览 Default.aspx 页面。然后单击在上一步中创建的链接。然后应被带到了 Login.aspx 页面。注意，总体布局，如页眉及边条等仍然保持不变。当从一个页面转到另一个页面时唯一变化的是主内容区域中的内容。

工作原理

当在浏览器中请求基于母版页的页面时，服务器会阅读内容页与母版页，将两者合并，处理它们，然后将最终结果发送给浏览器。在本练习的第(6)步中看到了在浏览器中的 HTML 含有这两个文件的标记的请求页面。

当更新或彻底改变站点外观时，有了母版页可以少做不少工作。由于站点的整个设计和布局都是在母版页中定义的，因此想作任何修改时只需要访问那个页面就可以了。所有内容页会自动跟着作相应修改。

1. 母版页详解

到目前为止，已经介绍了带有主内容占位符的母版页。不过如果在 Markup 视图中看一下母版页，将发现页面的 head 部分还有一个 Content Placeholder。


```
<head runat="server">
  <title></title>
  <asp:ContentPlaceholder id="head" runat="server">
  </asp:ContentPlaceholder>
  ...
</head>
```

每当创建一个新的母版页时就会自动添加此占位符。就像对 jQuery 库的引用一样，在内容页中可以用它来添加页面特有的位于页面<head>标记之间的内容，比如 CSS(包括内嵌样式表和外部样式表)和 JavaScript。第 10 和 11 章将介绍关于 JavaScript 和 jQuery 的更多知识。在 Markup 视图中，需要将内容添加到此占位符中，因为它在 Design 视图中是不可见的。

母版页中名为 cpMainContent 的 ContentPlaceholder 本身目前还不包括任何标记。然而，并不一定必须这样空着。只要没有被内容页重写，就可以在那里添加自己的内容作为内容页的默认信息。例如，可以在母版页中使用下面的<asp:ContentPlaceholder>：

```
<asp:ContentPlaceholder ID="cpMainContent" runat="server">
  This is default text that shows up in content pages that don't
  explicitly override it.
</asp:ContentPlaceholder>
```

当基于该母版页创建一个新页面时，一开始在 Markup 视图中看不到此默认信息。然而，可以打开 Content 控件的 Smart Tasks 面板(如图 6-6 所示)，并选择 Default to Master's Content。

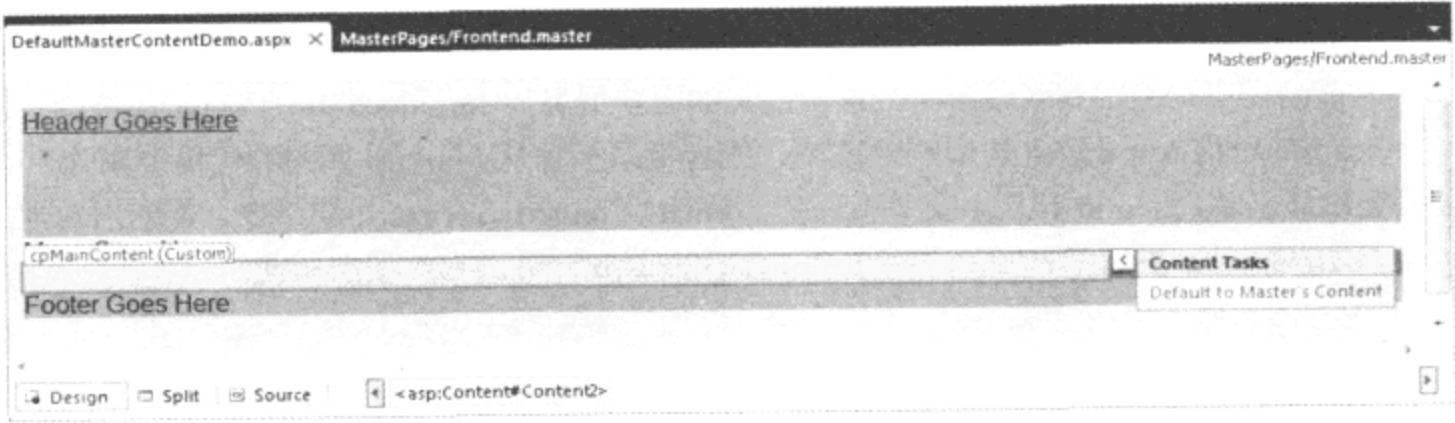


图 6-6

在询问是否希望默认值为母版页内容时，如果单击 Yes 按钮，VWD 就会删除页面 Markup 视图中的整个<asp:Content>控件。然而，当在浏览器中请求页面时，仍将看到母版页中的默认内容。在 Design 视图中，内容仍然是可见的，此刻在设计层面上以只读区域的形式出现。在后面的阶段添加新的 ContentPlaceholder 时带默认内容的母版页就会比较有用。现有页面可以仅显示默认内容，而不需要接触所有这些页面。新的页面可以定义它们自己的内容。

一旦将默认值设置为母版页的内容，就能通过打开 Smart Tasks 面板并选择 Create Custom Content 来再次创建自定义内容。这样就会复制默认内容并向页面中添加空的<asp:Content>控件，可以用自己的内容填充。

2. 嵌套母版页

母版页也可以嵌套。嵌套母版页是基于另一个母版页的母版页。内容页面则可以基于嵌套母版页。如果有一个目标为不同区域仍然需要共享相同外观的 Web 站点，采用嵌套母版页就比较有用。

例如，假设有一个公司网站，分为各个部门。外部母版页定义站点的全局外观，包括公司 logo 和其他品牌元素。然后不同的部门可以有不同的嵌套母版页。例如，销售部的部分可以基于不同于市场部的嵌套母版页，这样各部门就能向它们在站点的部分中加上自己的身份标识。Visual Web Developer 2010 对于 Design 视图中的嵌套母版页有很好的设计时支持，可以显示最终页面的样子。

嵌套母版页的创建很简单：当添加母版页时选中 Select Master Page 复选框，就像向站点中添加正常内容页时一样。然后，在内容页中要重写的位置将<asp:ContentPlaceHolder>控件添加到<asp:Content>控件中。

3. 关于母版页的一些警告

尽管母版页很不错并且能节省不少工作，但是还是有一些警告要知道。

对于初学者，ASP.NET 运行库会改变控件在页面中的客户端 ID。这是用在客户端脚本中的 id 属性，它从浏览器的 JavaScript 中访问控件。在正常 ASPX 页面中，控件的服务器端 ID 通常一对一地插在最终 HTML 中。例如，普通 ASPX 页面中在服务器端的 ID 名为 Button1 的 Button 控件用下列代码定义：

```
<asp:Button ID="Button1" runat="server" Text="Click Me" />
```

最终的 HTML 中客户端 ID，如下所示：

```
<input type="submit" name="Button1" value="Click Me" id="Button1" />
```

然而，<asp:Content>控件内同一个按钮最终却出现下面这样的代码：

```
<input type="submit" name="ctl00$cpMainContent$Button1" value="Click Me"
      id="cpMainContent_Button1" />
```

name 属性已经用自动生成的母版页 ID(ctl00)作前缀，并且 name 属性和 id 属性都包含有 ContentPlaceHolder 控件的 ID(cpMainContent)。

这意味着引用 Button1 的任何客户端代码现在都应引用 cpMainContent_Button1。

注意，这并不仅仅是母版页的问题。在其他情况下也会遇到这种行为；比如，使用用户控件(第 8 章将会介绍)和数据绑定控件(第 13 章及以后章节会讨论)。

第二个警告与第一个警告相关。因为 HTML 元素的 name 和 id 已经改变，所以它们给页面增加了相当可观的大小。对于单个控件来说这可能还不是问题，但是如果有一个含很多控件的页面，恐怕就会影响站点的性能了。对于嵌套母版页来说问题就更严重，因为内容控件都附加到了 ID 后面。同一个按钮在嵌套母版页内最终可能是这样：

```
<input type="submit" name="ctl00$ctl00$cpMainContent$ContentPlaceHolder1$Button1"
      value="Click Me" id="cpMainContent_ContentPlaceHolder1_Button1" />
```

为了使问题减轻，应保持 ContentPlaceHolder 和 Content 控件的 ID 尽可能短。为了提高可读性，本书使用了较长的名称，比如 cpMainContent。然而，在实际站点中可以将它缩短为 cpMC 或 cp1 以节省每次请求时的带宽。



注意：ASP.NET 4 引入了一个新的功能，即 ClientIDMode，以帮助减少有关因改变客户端 ID 而引起的问题的次数。第 8 章将会更多地介绍该功能。

母版页允许在单个位置中定义站点的总体外观。这样可以提高站点的一致性和可维护性。然而，还有另一种提高站点一致性的办法：将页面的行为集中在 Web 站点中。这可以用所谓的基页来实现，具体将在下一节讨论。

6.2 使用集中的基页

第 5 章已经指出，默认情况下所有 ASPX 页面派生自名为 System.Web.UI.Page 的类。这意味着所有页面至少会有在这个类中定义的行为。

然而，在有些情况下，仅有这个行为还不够，需要结合自己的内容。例如，假设需要添加应用到站点中所有页面的行为。不需要将此行为添加到各个单独的页面，可以创建一个公共的基页(base page)。然后站点中的所有页面从这个中间页面中继承，而不从标准 Page 类继承。图 6-7 的左半部分显示了名为 MyWebPage 的默认 ASPX 页面如何直接从 Page 类中继承。该图的右半部分显示了 ASPX 页面从名为 BasePage 的类(这个类又继承自 Page)继承的情况。

- 为了能让页面从这个基页继承，需要做以下两件事：
- 创建一个继承自 Web 站点的 App_Code 文件夹中的 System.Web.UI.Page 的类。
 - 在站点中创建继承自这个基类而不是继承自标准 Page 类的 Web 页面。

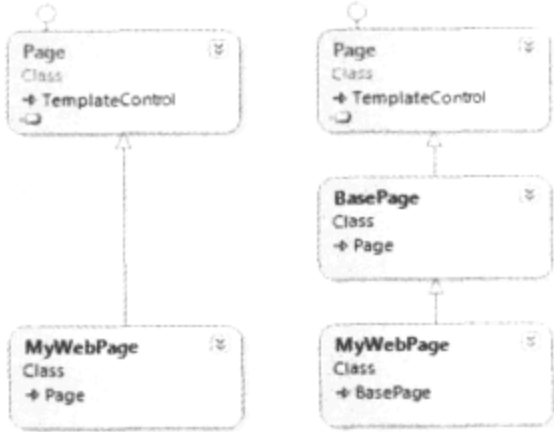


图 6-7

在下面的“试一试”练习中，将在 App_Code 文件夹中创建一个新的基页类。目前，这个类唯一的目的是在运行时检查页面的标题，防止页面使用空名称或无意义的标题，如浏览器中的 Untitled Page。给页面起一个独特、有用的标题，可以使页面很容易地被主要的搜索引擎检索出，因此推荐在 Web 页面中始终含有标题。以编程的方式检查标题相当容易实现，这样可以将主要精力集中在继承这一概念而不是实际代码上。在本章后面讨论主题时将再次修改基页，这次是检索用户对某个主题的首选项。



注意：VWD 的先前版本使用 Untitled Page 作为新 Web 表单的默认标题。但是自 VWD 2008 Service Pack1 发布以来，新的默认标题变成了一个空的字符串。我决定在基页中保留 Untitled Page 选项，以免向新的 ASP.NET 4 Web 站点添加带有不正确标题的老页面。

然而，在可以实现基类之前，需要知道关于 ASP.NET 页面生命周期的更多知识，这是描述浏览器发出请求时处理 Web 页面的过程的另一个重要概念。

6.2.1 ASP.NET 页面生命周期

在思考 Web 服务器如何将页面发送给浏览器，以及把这个过程看作页面的生命周期时，可能会遇到页面生命周期中的几个重要时刻。例如，浏览器最初发出的请求是页面“生命周期”的起点。类似地，当页面将它的整个 HTML 发送给浏览器时，它的生命周期可能看起来就要结束了。然而，在页面的生命周期中还有一些较有趣的事件会继续发生。表 6-1 描述了页面经过的 8 个主要阶段。在这些阶段中，至少会引发一个事件，可以让页面开发人员了解页面的生命周期并在恰当的時刻采取行动。在下面的练习中将看到一个这样的示例。

表 6-1

阶 段	说 明
页面请求 (Page request)	请求 ASPX 页面启动了该页面的生命周期。当 Web 服务器能够并且允许返回页面的一个缓存副本时，整个生命周期就不会执行。在所有其他情况下，页面都会进入开始阶段
开始 (Start)	页面的开始阶段，它获得对 Request 和 Response 等属性的访问权限，用来与页面的环境交互。此外，在这一阶段会引发 PreInit 事件，指示该页面将进入初始化阶段。以后将用这一事件来设置页面的主题
页面初始化 (Page initialization)	在这一阶段，在页面中建立或通过写程序添加的控件将变得可用。在这一阶段 Page 类会激活 3 个事件：Init、InitComplete 和 PreLoad。还是在这一阶段，控件属性在回发过程中再次从 ViewState 和 ControlState 中加载。因此，当修改 DropDownList 中选中的选项并在之后引发一个回发时，就是再次从下拉列表中预先选择正确的选项的时刻，之后可以在服务器端代码中使用该选项。
加载 (Load)	在这一阶段页面引发 Load 事件
有效性验证 (Validation)	在有效性验证阶段，用 Validation 控件验证正在处理的用户输入的有效性。第 9 章将介绍验证器
回发事件处理 (Postback event handling)	<p>在这一阶段，页面中的控件可能引发它们自己的事件。例如，当用户从列表选择了不同的选项时，DropDownList 可能会引发一个 SelectedIndexChanged 事件。类似地，当用户在将文本传送回服务器之前修改了文本时，TextBox 可能会引发 TextChanged 事件。</p> <p>当完成了所有事件处理时，页面会引发 LoadComplete 事件。在这个阶段会引发 PreRender 事件，指示页面将呈现给浏览器。在那以后不久，就会引发 SaveStateComplete 事件以指示页面在将控件的所有相关数据存储在 View State 中</p>

(续表)

阶 段	说 明
呈现 (Rendering)	呈现是控件(以及页面本身)将它们的 HTML 输出到浏览器的阶段
卸载 (Unload)	卸载阶段实际上是一个清理阶段。这是页面和控件能释放资源(比如数据库连接)的时刻。在这一阶段，会引发 Unload 事件，因此可以进行所需的任何清理程序操作

要意识到的一件重要的事情是，所有这些事件都是在服务器端触发的，而不是在客户端。因此，即使在客户端文本框中修改了文本，TextBox 控件的 TextChanged 事件还是会在将该文本回发给页面时在服务器端触发。

现在，您可能会疑惑为什么需要知道这些。最重要的理由是，了解一下页面的生命周期，知道只有在页面生命周期的特定阶段才会进行某些动作。例如，后面即将看到，动态地修改主题必须发生在页面生命周期的早期。为了真正理解 ASP.NET 页面生命周期，需要对控件、状态及事件等多一些了解。因此，在第 15 章介绍激活的所有不同事件以及事件顺序时将再次提到页面生命周期。

在下面的练习中，将使用 Page 类的 PreRender 事件来检查页面标题。由于在很多事件中，页面开发人员能通过编程方式来设置页面的标题，因此检查标题正确性的工作应尽可能放在页面生命周期的后期进行，这就是为什么在这里 PreRender 是最合适事件的原因，从下一个练习中可以看到这一点。

6.2.2 实现基页

实现基页相当容易，所需做的只是向 App_Code 文件夹中添加一个类文件，向其中添加一些代码，这样就可以了。稍微复杂的往往是确保站点中各页面都是继承自这个新的基页而不是继承自标准 System.Web.UI.Page 类。但是，当使用 Code Behind 文件时没有办法配置应用程序自动这么做，因此需要手工修改每个页面。Visual Web Developer 允许导出已经包含该代码的页面模板，从而使这种工作稍稍轻松一些。当在下面“试一试”的练习中添加基页到站点后，将看到如何将页面导出为模板，从而能立即添加使用该基页的文件。

试一试

创建基页

- (1) 在 Solution Explorer 中右击 App_Code 文件夹，并选择 Add New Item 命令。在 Templates 列表中选择 Class，并将文件命名为 BasePage。可以选择喜欢的任何名称，不过 BasePage 清晰地描述了类的用途，因此用这个名称更容易理解它的功能。
- (2) 清除文件的内容，然后向这个类文件中添加下面的代码：

```
VB.NET
Public Class BasePage
    Inherits System.Web.UI.Page

    Private Sub Page_PreRender(
```

```
ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.PreRender

If Me.Title = "Untitled Page" Or String.IsNullOrEmpty(Me.Title) Then
    Throw New Exception(
        "Page title cannot be ""Untitled Page"" or an empty string.")
End If
End Sub

End Class
```

C#

```
using System;
using System.Web;

public class BasePage : System.Web.UI.Page
{
    private void Page_PreRender(object sender, EventArgs e)
    {
        if (this.Title == "Untitled Page" || string.IsNullOrEmpty(this.Title))
        {
            throw new Exception(
                "Page title cannot be \"Untitled Page\" or an empty string.");
        }
    }

    public BasePage()
    {
        this.PreRender += new EventHandler(Page_PreRender);
    }
}
```

(3) 保存文件并关闭它，然后打开以前创建的 Login.aspx 页面。打开它的 Code Behind 文件，然后修改 Inherits 代码(在 C#中是冒号)，因此 Login 页面继承自以前创建的 BasePage 类:

VB.NET

```
Partial Class Login
    Inherits BasePage
    ...
End Class
```

C#

```
public partial class Login : BasePage
{
    ...
}
```

(4) 保存页面，然后按 Ctrl+F5 组合键在浏览器中请求它。如果之前不曾修改页面的标题，则应该会在浏览器中看到如图 6-8 所示的错误消息。

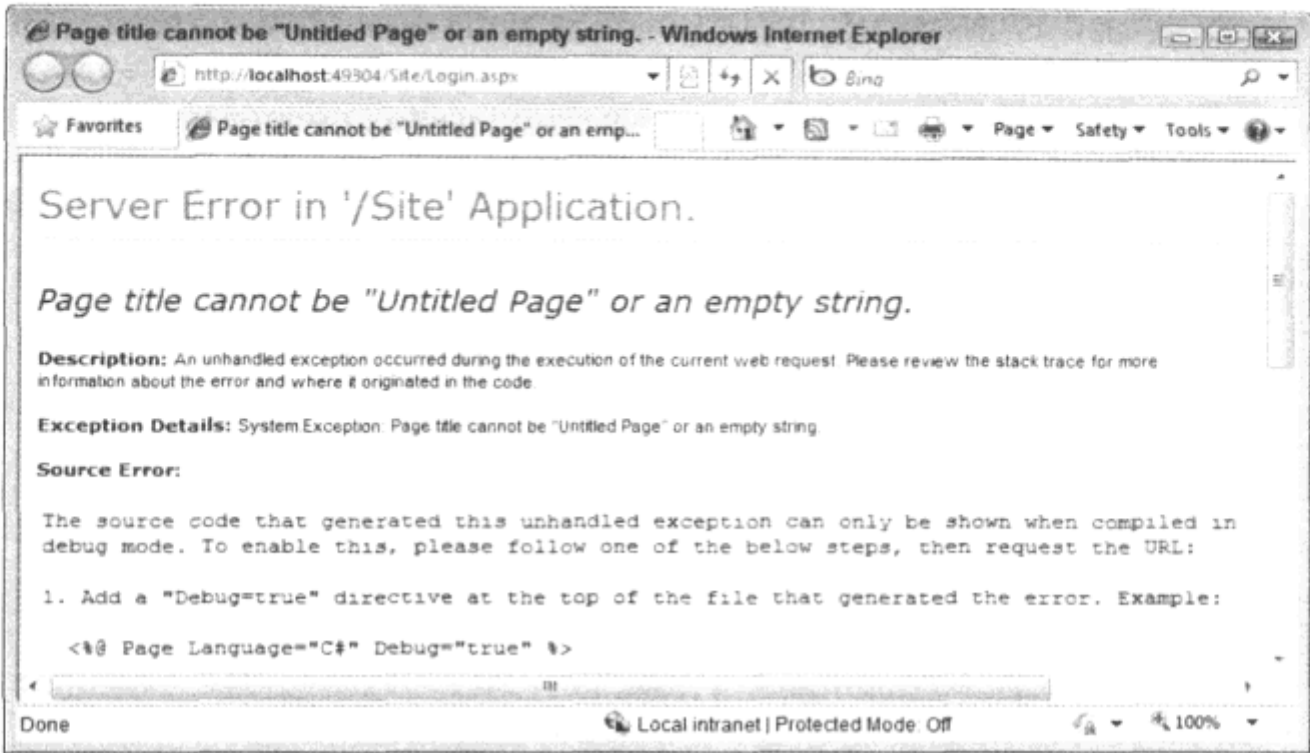


图 6-8

如果没有显示这个通用错误消息，则可能会显示有关 **BasePage** 类源代码的错误消息，页面标题的正确性验证在 **BasePage** 类中进行。

(5) 回到 VWD 并在 Markup 视图中打开登录页面。定位到 @ Page 指令的 Title 属性(如果该属性不在指令里，就将其添加到指令中)，并将它的值设置为 Log in to Planet Wrox。下面的代码段显示了 @ Page 指令的 VB.NET 版本，它的 C#版本与此基本相同：

```
<%@ Page Title="Log in to Planet Wrox" Language="VB"
    MasterPageFile="~/MasterPages/Frontend.master" AutoEventWireup="false"
    CodeFile="Login.aspx.vb" Inherits="Login" %>
```

(6) 在站点中重复第(3)和第(5)步。为了让这个过程快一点，可以通过 Find and Replace 命令快速地用 **BasePage** 替换已经出现的 **System.Web.UI.Page**。确保不要意外地在 App_Code 文件夹本身的 **BasePage** 文件中替换它。为了防止发生这种情况，一定要只在 Code Behind 文件中搜索，如：

- 打开 Replace in Files 对话框(按 Ctrl+Shift+H 组合键或选择 Edit | Find and Replace | Replace in Files 命令)；
- 在 Find What 框中输入 System.Web.UI.Page。在 Replace With 文本框中输入 BasePage；
- 展开 Find Options 部分，并且在 Look at These File Types 文本框中根据所用的语言输入 *.aspx.vb 或 *.aspx.cs。这样就留下了 BasePage 文件，此文件只有一个扩展名.vb 或.cs；
- 单击 Replace All 按钮，然后单击 Yes 按钮以确认 Replace 操作。

(7) 保存对所有打开页面的修改，然后再次浏览到 Login.aspx。如果一切如期进行，应不会出现错误，这时会看到 Login 页面。

记住，这时如果试图访问站点中的所有其他页面，则会抛出一个错误。但是这很容易解决：只要都给它们赋予有效的 Title 即可。对于页面指令中没有 Title 属性的页面，需要手动操作。对于带有空 Title 属性(Title=" ")的其他页面，可以通过在站点中搜索 Title=" "并用诸如 Title="Planet Wrox" 这样的标题来替换该属性，以快速地解决这个问题。不要忘记将 Look At These File Types 重置为*.*。对于页面(除了目前为止已经创建的演示页面之外)而言，最好是给它们设置一个唯一的名称，以便

能够清楚地描述页面所包含的内容。

工作原理

默认情况下，Web 站点中的所有页面都继承自在 System.Web.UI 名称空间中定义的 Page 类。这样就给了它们必需的行为，使它们可以作为能由浏览器请求和由服务器处理的 Web 页面。由于.NET 中的继承模型允许创建一个可以相互继承的类的链，因此可以轻松地在 Web 页面与标准 Page 类之间插入自己的基页类。这一过程通过将 Inherits 语句(在 C#中是冒号[:])改为新的 BasePage 语句来实现。

VB.NET

```
Partial Class Login
    Inherits BasePage
```

C#

```
public partial class Login : BasePage
```

在这个新的 BasePage 类内添加一个事件处理程序，当类激活它的 PreRender 事件时会调用这个事件处理程序。正如前面所提到的，这个事件引发的时间在页面的生命周期中相当迟，当建立了整个页面并准备呈现给客户端时才引发：

VB.NET

```
Private Sub Page_PreRender(
    ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.PreRender

    ' Implementation here

End Sub
```

C#

```
private void Page_PreRender(object sender, EventArgs e)
{
    // Implementation here
}
```

注意，Visual Basic 使用 Handles 关键字通知编译器，将用 Page_PreRender 方法来处理事件。在 C#中，需要手动关联这个事件处理程序。此操作适于在类的构造函数中进行：

```
public BasePage()
{
    this.PreRender += new EventHandler(Page_PreRender);
}
```

上面突出显示的代码的作用与 VB.NET 中的 Handles 关键字的作用相同：它告诉编译器当页面引发 PreRender 事件时运行什么方法。

在事件处理程序内，代码检查当前页面的标题。如果页面的标题仍然是 Untitled Page 或空字符串(添加到 Web 项目的任何新页面的默认值)，那么它就会抛出一个异常。

VB.NET

```
If Me.Title = "Untitled Page" Or String.IsNullOrEmpty(Me.Title) Then
    Throw New Exception(
        "Page title cannot be ""Untitled Page"" or an empty string.")
End If
```

C#

```
if (this.Title == "Untitled Page" || string.IsNullOrEmpty(this.Title))
{
    throw new Exception(
        "Page title cannot be \"Untitled Page\" or an empty string.");
}
```

注意上面代码中的关键字 **Me**(在 VB.NET 中)和 **this**(在 C#中)的用法。这些关键字是区分上下文的，总是引用使用它们的类的实例。在这个示例中，**Me** 和 **this** 引用 **BasePage** 类的当前实例。这个 **BasePage** 实例有一个 **Title** 属性(该属性继承自 **Page** 属性)，可以用来检查不需要的值。如果它仍然包含默认标题(一个空字符串)或文本“Untitled Page”，代码就会触发(或抛出)一个异常。这时会立即停止页面的执行，因此开发人员可以在页面发布给公众之前通过提供一个有效的标题来解决这个问题。第 18 章将介绍关于异常以及如何防止及处理异常的更多知识。

为了在错误消息中显示双引号(")，这两种语言使用了不同的格式。在 Visual Basic 中，需要用双引号。在 C#中，需要在双引号前面用一个反斜杠(\)作前缀，来转义双引号。在这两种情况下，浏览器中最终出现的错误消息都用一对双引号括着，如图 6-8 所示。

由于添加到站点中的每个新页面现在应从这个新的基页继承，因此应创建一个在 Code Behind 文件和标记中已经有正确代码的页面模板，使得更容易从一开始就向站点添加正确的页面。

6.2.3 创建可重用的页面模板

Visual Web Developer 附带有一个优秀的导出模板的工具，适用于大量不同文件类型，包括 ASPX 页面、类文件，甚至 CSS 文件。创建了自定义模板后，就可以一次性定义每个文件中都需要的代码或标记，然后基于这个模板创建新的文件，这样就可以直接给文件一个很好的开头，并使需要输入的代码减少。下面的“试一试”练习显示了如何创建自己的模板。

试一试

创建一个可重用的页面模板

在本练习中将介绍如何创建添加到站点的所有新 ASPX 页面的模板文件。为了避免与当前站点中的现有页面冲突，创建一个新的临时页面并用它来创建模板。稍后就可以删除临时文件。

(1) 向站点中添加一个新的 Web 窗体并命名为 **Temporary.aspx**。确保它使用了 **Code Behind**，使用选择的编程语言，且基于 **MasterPages** 文件夹中的母版页。

(2) 打开这个新页面(按 F7 键)的 Code Behind 并修改 **Inherits** 行(在 C#中是冒号)，使页面继承自 **BasePage** 类而不是 **System.Web.UI.Page**。同样将类的名称 **Temporary** 重命名为 **\$relurlnamespace\$_\$safeitemname\$**：

VB.NET

```
Partial Class $relurlnamespace$_$safeitemname$
    Inherits BasePage
    ...
End Class
```


End Class

```
C#
public partial class $relurlnamespace$_$safeitemname$ : BasePage
{
    ...
}
```

确保不要删除任何现有代码，比如 using 语句或 C#版本中的 Page_Load 方法。

不要担心关于像\$这样的未预期字符的任何编译错误。一旦开始基于这个模板添加页面，\$relurlnamespace\$-\$Safeitemname\$就会被正在添加的页面的名称替换。

(3) 切换到页面的 Markup 视图，将 Inherits 属性从 Temporary 改为 \$relurlnamespace\$_\$Safeitemname\$:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master"
    AutoEventWireup="true" CodeFile="Temporary.aspx.cs"
    Inherits="$relurlnamespace$_$safeitemname$" %>
```

可以不管 CodeFile 属性；每当向站点添加新页面时 VWD 会自动将它改为正确的 Code Behind 文件。

(4) 另外，如果需要的话，可以向希望默认出现在页面中的文件里添加其他代码，比如带版权说明的注释块。

(5) 保存对页面的修改然后选择 File | Export Template 命令。在出现的对话框中，选择 Item Template 选项并从屏幕底部的下拉列表中选择编程语言，如图 6-9 所示。

(6) 单击 Next 按钮并选中列表底部的 Temporary.aspx 前面的复选标记。再次单击 Next 按钮，打开 Select Item References 对话框。

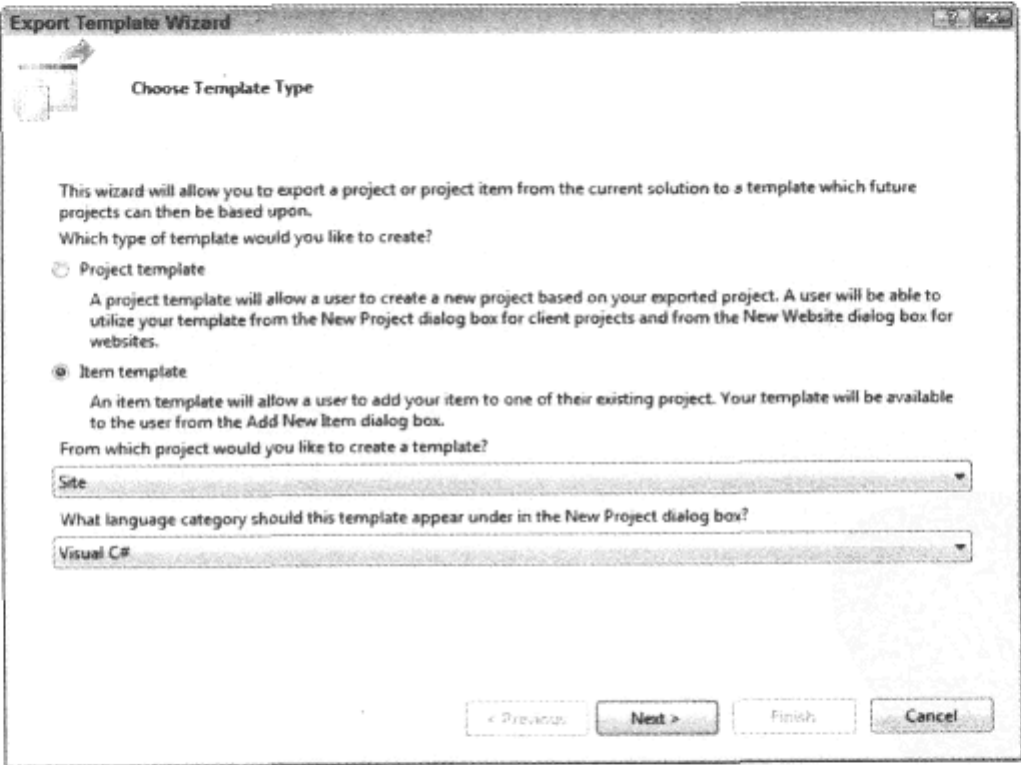


图 6-9

(7) 不需要在 Select Item References 对话框中设置任何内容。如果有一个引用特定程序集(.dll 文件)的 Web 站点，应在这里选择它们，以便下次添加基于该模板的文件时 VWD 好自动添加这些引

用。在这种情况下，再次单击 Next 按钮来到 Select Template Options 屏幕。输入 MyBasePage 作为新模板的名称，也可以输入一个简短描述来说明该模板的用途。图 6-10 显示了最终对话框。

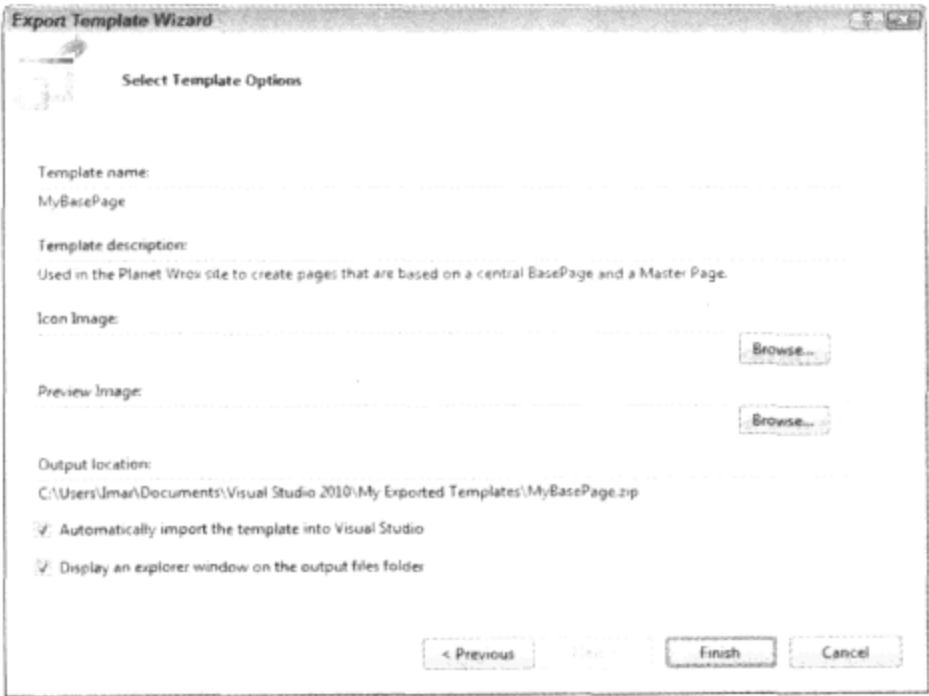


图 6-10

(8) 单击 Finish 按钮完成模板的创建。VWD 会打开 Windows 资源管理器，将新模板显示为一个 ZIP 文件。可以关闭窗口，因为不会用到它。

如果使用 VB.NET 和 C#两种语言来实践本练习，则确保在为第二种语言进行导出之前一定先要重命名结果 ZIP 文件，否则它会被重写。要重命名文件，打开 Windows 资源管理器，找到 My Documents (就是 Windows Vista 和 Windows7 中的 Documents)，然后浏览到 Visual Studio 2010\Templates\ItemTemplates。从中将发现一个名为 MyBasePage.zip 的文件，可以将它重命名为像 MyBasePageCS.zip 这样的名称。注意文件的位置不同于图 6-10 中文件的位置；输出位置只包含可以用作备份的导出模板的一个副本。

(9) 回到 VWD 中，删除创建的临时文件 Temporary.aspx。然后右击 Solution Explorer 中的 Web 站点并选择 Add New Item 命令。注意，自定义模板现在出现在如图 6-11 所示的模板列表中。如果单击它，它甚至会显示以前添加的描述。

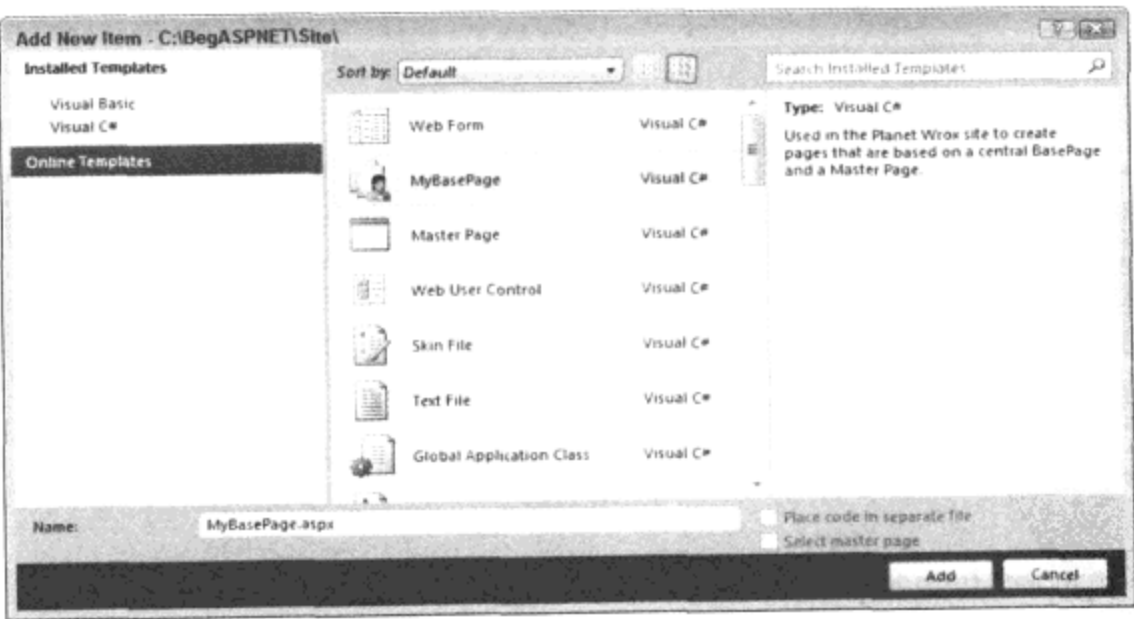


图 6-11

(10) 为页面输入一个新名称，比如 TestPage.aspx，并单击 Add 按钮将它添加到站点中。检查一下文件的标记与 Code Behind，确认\$relurlnamespace\$_\$safeitemname\$已经被重命名为_TestPage 以反映页面的新名称。如果一切看上去无误，就可以删除 TestPage.aspx，因为在 Planet Wrox Web 站点中用不到它。

工作原理

当导出模板时，Visual Web Developer 使用必需文件创建一个 ZIP 文件——在本练习中必需文件是一个 ASPX 文件和一个 Code Behind 文件。然后，这个模板会存储在 Documents 文件夹下的 Visual Studio 2010 文件夹的 ItemTemplates 子文件夹中。ZIP 文件中的有些文件包含占位符 \$relurlnamespace\$_\$safeitemname\$。当用 Add New Item 对话框向基于模板的站点中添加一个新文件时，VWD 会用文件夹(将文件添加到站点的根文件夹后，该文件夹为空)的名称替换\$relurlnamespace\$，用页面的实际名称替换\$safeitemname\$。在本练习中，输入 TestPage.aspx 作为页面的新名称，因此最终在 Code Behind 中得到一个名为_TestPage 的类，这个类继承自全局 BasePage。在两个占位符之间硬编码下划线(_)，只有在将基于该模板的 Web 窗体添加到子文件夹中时才需要下划线。然而，它是类标识符的一个有效开始标记，因此可以放心地将其留在 Web 站点根部的页面中。如果向子文件夹中添加一个文件(如 Demos 文件夹)，则类的名称使用该文件夹的名称作为前缀，这样最后类的名称就是 Demos_TestPage。除了\$relurlnamespace\$_\$safeitemname\$外，还有几个其他占位符也可以使用。在 MSDN 站点 <http://msdn2.microsoft.com> 上搜索短语\$safeitemname\$，就可以找到其他模板参数。

如果需要对导出模板进行修改，可以重新执行整个导出过程，也可以手动编辑 zip 文件中的文件。

有了这个导出模板，现在就可以非常快地向继承自 BasePage 类的站点中添加页面。不再需要手动修改类文件的 Code Behind，也不再需要手动修改页面的标记。

除了母版页和中心 BasePage 类外，还有一些创建 Web 站点外观一致的方法。其中之一就是主题(theme)。

6.3 主题

前面介绍了如何创建母版页来定义站点中页面的全局外观。还介绍了如何通过使用集中的基页来集中页面的行为。然而，还有其他影响站点全局外观的方式：主题与外观。外观在本章以后再介绍，因为它们是主题的一部分，所以需要先讨论主题。

主题是定义页面外观的文件的集合。它通常包含外观文件、CSS 文件和图像。主题在 Web 站点的根文件夹中的特殊 App_Themes 文件夹中定义。在这个文件夹中需要创建定义实际主题的一个或多个子文件夹。在每个子文件夹中，可以有若干组成主题的文件。图 6-12 显示了 Web 站点的 Solution Explorer，定义了两个主题：Monochrome 与 DarkGrey。

每当主题在活动状态时，对主题文件夹中各个 CSS 文件的链接就会自动添加到页面的<head>部分。稍后将看到其工作原理。主题



图 6-12

文件夹中的图像引用自 CSS 文件。它们可以用来修改 Web 站点的公共元素，比如背景图像，或者项目列表或导航列表中使用的图像。

为了创建一个主题，需要做下列事情：

- 如果站点中还没有特殊 App_Themes 文件夹，则需创建一个；
- 对于要创建的每个主题，用主题的名称创建一个子文件夹，如图 6-12 中所示的 Monochrome 或 DarkGrey；
- 可选地，创建一个或多个将成为主题一部分的 CSS 文件。虽然根据主题命名 CSS 文件有助于标识正确的文件，但并不要求一定要这样做。添加到主题的文件夹中的任何 CSS 文件都会在运行时自动添加到页面中；
- 可选地，向主题文件夹中添加一个或多个图像。CSS 文件应当用稍后将介绍的相对路径来引用这些图像；
- 可选地，也可以向主题文件夹中添加一个或多个外观文件。外观允许为之后要在运行时应用的特定控件定义单个属性(比如 ForeColor 和 BackColor)。

执行了这些步骤以后，就能将站点或单个 Web 页面配置为使用此主题。为了能够建立恰当的主题，需要知道有两种类型的主题。

6.3.1 不同类型的主题

ASP.NET 页面有两个不同的设置主题的属性：Theme 属性和 StyleSheetTheme 属性。这两个属性都使用在 App_Themes 文件夹中定义的主题。虽然一开始它们看起来非常相似，但是在运行时它们的行为就不同了。StyleSheetThemes 在页面的生命周期中应用得非常早，在创建页面实例后不久就应用了。这意味着单个页面能通过在控件上应用内联属性来重写主题的设置。因此，例如，带有将按钮的 BackColor 设置为紫色的外观文件的主题可以被页面标记中下面的控件声明重写：

```
<asp:Button ID="Button1" runat="server" Text="Button" BackColor="Red" />
```

另一方面，Theme 属性在页面的生命周期中应用的时间较晚，能有效地重写为单个控件自定义的任何属性。

6.3.2 在 Theme 和 StyleSheetTheme 之间作选择

由于 StyleSheetTheme 的属性能被页面重写，而 Theme 又能再次重写这些属性，两者用于不同的目的。如果想为控件提供默认设置则应设置 StyleSheetTheme。即 StyleSheetTheme 能为控件提供默认值，然后又可以在页面级重写。如果想强制应用控件的外观则应使用 Theme 属性。因为 Theme 中的设置不能再重写，而且它有效地重写了任何自定义设置，因此能确保控件的外观就是在主题中定义的样子。有一个例外：通过将控件上的 EnableTheming 设置为 False 来禁用主题。在本章快结束时将介绍该属性及其作用。

6.3.3 应用主题

要向 Web 站点应用主题，有 3 个不同的选项：在 Page 指令中的页面级、在站点级修改 web.config 文件，以及通过程序来设置主题。

- 在页面级设置主题：在页面级设置 Theme 属性或 StyleSheetTheme 属性很容易，只要设置页面的 Page 指令中的相关属性即可；

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
    Inherits="_Default" Theme="DarkGrey" %>
```

用 StyleSheetTheme 替换 Theme 来应用一个主题，该主题的设置可以由单个页面重写。图 6-13 表明，输入 Theme="", VWD 就会弹出一个列表，其中含有它在 App_Themes 文件夹中发现的所有主题。



图 6-13

- 在站点级设置主题：为了在整个 Web 站点中强制应用一个主题，可以在 web.config 文件中设置主题。要做到这一点，需要将一个 theme 属性添加到<system.web>元素内的<pages>元素中；

```
<pages theme="DarkGrey">
...
</pages>
```

确保全部用小写字母输入 theme，因为 web.config 文件中的 XML 是区分大小写的。

- 通过程序来设置主题：设置主题的第三种也是最后一种方式是通过代码来编程设置。在后面的“试一试”练习中将看到其工作原理。

下面的“试一试”练习说明了主题的工作原理。通过这个练习，您将学习如何创建主题、添加必需的 CSS，然后将应用程序配置为使用新主题。

试一试 为 Web 站点创建一个新主题

在本练习中将创建两个主题：Monochrome 和 DarkGrey。对于每个主题，将添加 CSS 布局，这个布局会自动应用到站点上。还将配置应用程序使用这两个主题之一，然后切换到另一个主题看看有什么区别。

(1) 向 Web 站点中添加特殊 App_Themes 文件夹。为此，在 Solution Explorer 中右击 Web 站点，然后选择 Add ASP.NET Folder | Theme 命令。这样不仅会创建 App_Themes 文件夹，而且默认会立即为名为 Theme1 的主题创建一个子文件夹。输入 Monochrome 作为新名称。现在 Solution Explorer 看起来应如图 6-14 所示。

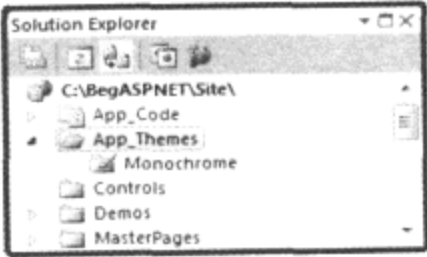


图 6-14

(2) 从 Styles 文件夹中将 Styles.css 文件移到这个新的 Monochrome 文件夹中。可以直接将它拖到新文件夹中，也可以按 Ctrl+X 键剪切文件，然后单击 Monochrome 文件夹，再按 Ctrl+V 键来粘贴它。可以留下空的 Styles 文件夹，因为以后还会用到它的。

(3) 为了在以后清楚地看到 CSS 来自何处，将文件名 Styles.css 改为 Monochrome.css。

(4) 由于主要布局现在已经由主题控制，因此不再需要母版页中指向老 CSS 文件的<head>部分中的<link>元素，所以可以删除它。要删除<link>元素，打开母版页，切换到 Markup 视图并从代码中删除下面突出显示的代码行：


```
<head runat="server">
  <title></title>
  <asp:ContentPlaceHolder ID="head" runat="server">
  </asp:ContentPlaceHolder>
  <link href="../../Styles/Styles.css" rel="stylesheet" type="text/css" />
</head>
```

(5) 下一步是向整个 Web 站点应用主题。为此，从站点的根文件夹中打开 web.config 文件，并直接在<system.web>元素中将 theme 属性添加到<pages>元素内：

```
<system.web>
  <pages theme="Monochrome"></pages>
  ...
```

(6) 要测试主题，保存所有修改然后在浏览器中请求页面 Default.aspx。站点的设计应与它所显示的样子一致。



常见错误：如果看到页面标题无效的错误，返回 Visual Web Developer 并将 Default.aspx 的@ Page 指令的 Title 属性改为“Welcome to Planet Wrox”。如果设计与它显示的样子不一致，则需要浏览器中按 Ctrl+F5 或 Ctrl+R 键。这样会迫使进行一个“硬刷新”，该操作意味着从服务器获得文件的最新版本，而不是获得页面的本地缓存副本。

- CSS 现在不会链接到母版页的 CSS 文件，而是通过在 web.config 文件中设置的主题包括在页面源代码中。要了解它的工作原理，在浏览器中打开页面的 HTML 源代码。在上方应看到下面的代码(为了更好的可读性，已更改了布局)。

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Welcome to Planet Wrox</title>
  <link href="App_Themes/Monochrome/Monochrome.css" type="text/css"
    rel="stylesheet" />
</head>
<body>
```

- 注意，从 Monochrome 主题文件夹到样式表的链接被注入到了页面的<head>中。ASP.NET 运行库会为它在当前活动主题文件夹中找到的每个.css 文件完成这一操作，因此一定要保持主题文件夹干净，避免浏览器包括和下载不必要的文件。还要注意，<link>被添加到了结束标记</head>前面。这样可以确保主题文件包括在可能是自己添加(比如通过母版页)的所有文件夹后面。这与 StyleSheetTheme 属性的工作方式相反。由于这种类型的主题允许重写它的设置，因此在文件最上方导入它，以便给它后面的其他改变页面外观的 CSS 文件腾出空间。

(7) 返回 Visual Web Developer 并在 Design 视图中打开母版页。这时会注意到所有设计信息都已不见，现在 VWD 又显示了页面的基本布局。但是，VWD 没有显示用 theme 属性设置的主题。然而，可以通过设置 StyleSheetTheme 来克服这一局限。为此，再次打开 web.config 文件，定位先前已经创建的<pages>元素并添加下面的属性：

```
<pages theme="Monochrome" StyleSheetTheme="Monochrome"></pages>
```


(8) 保存对 web.config 文件的修改，关闭它后再次打开母版页，切换到 Design 视图。这时将看到 VWD 现在已经向页面应用了正确的样式信息。

(9) 为了向站点中添加另一个主题，在 App_Themes 文件夹下创建一个新文件夹，命名为 DarkGrey。接着，打开下载的本书的代码文件夹。如果跟着本书前言中的指令做了，这个文件夹就位于 C:\BegASPNET\Resources。打开 Chapter 06 文件夹，然后打开 DarkGrey 文件夹。将 Windows Explorer 和 VWD 放在一起，然后从 Windows Explorer 中将 DarkGrey.css 文件拖到 VWD 中的 DarkGrey 主题文件夹中。如果无法拖动该文件，则在 Windows Explorer 中使用 Ctrl+C 键复制它，然后在 VWD 中使用 Ctrl+V 键将其粘贴到正确的文件夹中。Solution Explorer 现在应如图 6-15 所示。



图 6-15

在以后的练习中会添加 CSS 文件引用的图像。

(10) 再次打开 web.config 文件并将<pages>元素中两次出现的 Monochrome 都改为 DarkGrey。再次保存修改并按 Ctrl+F5 键。现在看到的便不再是紫色的 Monochrome 主题，而是应用了 DarkGrey 主题的站点，如图 6-16 所示。如果没有看到菜单，而主内容和边栏都紧靠在一起，那么应确保浏览器窗口有足够的宽度来显示所有的内容。

如果没有看到新的主题，就关闭所有打开的浏览器，确保正确地修改了 web.config 文件并再次打开 Default.aspx 文件。如果还是没有看到主题，就按浏览器中的 Ctrl+F5 键或 Ctrl+R 键迫使其从服务器获取一个新的副本。如果仍然没有显示新的主题，则需要在 Windows 托盘条中右击 ASP.NET Development Server 并选择 Stop 命令。接下来，通过在 VWD 中按 Ctrl+F5 组合键再次请求浏览器中的 Default.aspx 文件。

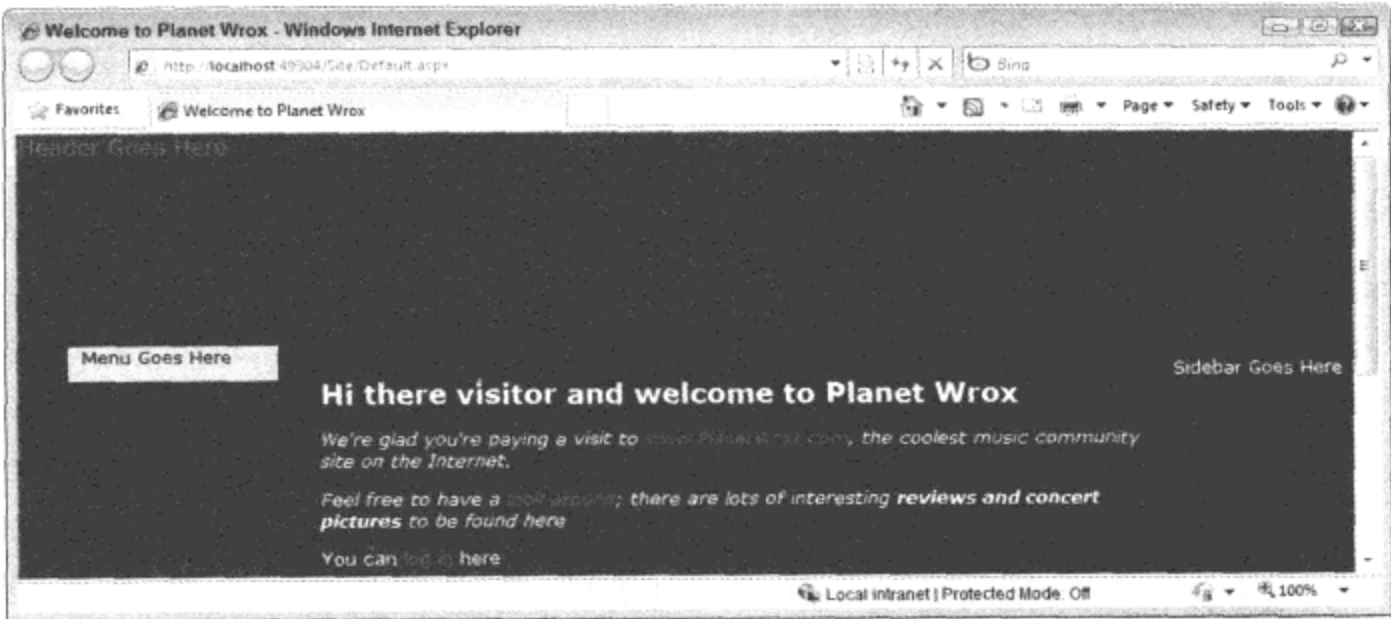


图 6-16

工作原理

在本“试一试”练习中首先通过修改 web.config 文件中的<pages>元素应用了 Monochrome 主题。当运行库发现某个主题是活动状态时，它就会扫描关联的主题文件夹寻找.css 文件，并将它找到的对所有.css 文件的链接包括到页面的<head>部分。在 Monochrome 主题的情况下，它找到了文件 Monochrome.css，并自动将它添加到<head>部分。当将该主题改为 DarkGrey 时会发生同样的过程。链接的样式表通过修改页面中使用的布局和颜色，来影响页面在浏览器中的显示方式。

为了在 VWD 中启用设计时支持，还需要修改 web.config 文件中的 StyleSheetTheme。唯一的不利方面是现在会包括相关 CSS 文件两次：一次为 Theme，另一次为 StyleSheetTheme。由于会包括同一个文件两次，因此它不会影响站点的布局。第二个文件中的所有选择器会否决第一个文件中的选择器。然而，如果觉得这种重复是在浪费 CPU 周期，则在开始使用应用程序时可以删除 web.config 文件中的 StyleSheetTheme 属性。

DarkGrey.css 中的 CSS 文件可以使页面的布局发生很大的变化。如果想知道文件中包含什么 CSS 以及它改变了页面的哪些元素，可以在 VWD 中打开它。其中有大量注释，详细描述了各个选择器。

ASP.NET 主题并不仅仅局限于 CSS 文件。随着下面的学习，您会知道主题也能够包含图像和外观文件。

6.3.4 扩展主题

除了 CSS 文件与外观(本章快结束时介绍)外，主题还可以包含图像。主题图像最普遍的用法是从 CSS 中引用它们。要充分利用图像，就要了解 CSS 如何引用图像。

在设计时，除非给出一个以指示站点根文件夹的正斜杠(/)开头的路径，否则 CSS 选择器引用的图像会相对于 CSS 文件的当前位置搜索。例如，图 6-17 中描述的 App_Themes 文件夹。

为了引用 Monochrome 主题的 Images 文件夹中的 MenuBackground.jpg 文件，可以向 Monochrome.css 中添加下面的 CSS。

```
#MenuWrapper
{
    background-image: url (Images/MenuBackground.jpg);
}
```

如果要引用站点根文件夹的 Images 文件夹中的图像，请使用这个 CSS。

```
background-image: url (/Images/MenuBackground.jpg);
```

注意，图像路径开头的正斜杠指示站点的根文件夹。如果要在不同的主题之间共享相同的图像，后一种语法更有用。只要将它们放在特定主题外面的文件夹中(如根文件夹的 Images 文件夹中)，然后用这种基于根文件夹的语法引用它们即可。只有在将 Web 站点的 Virtual Path 属性也设置成一个正斜杠(/)时，这种引用才管用。第 7 章将更深地入地介绍 URL 引用图像等资源时可以采用的不同形式，以及讨论如何使用 Virtual Path 属性。

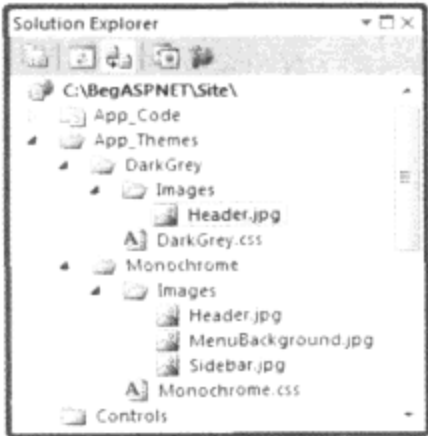


图 6-17

试一试

向主题中添加图像

在本“试一试”练习中，将向站点中添加图像和 CSS 文件来完成这两个主题。图像和 CSS 文件可以从本章的下载代码 zip 文件中得到，下载地址是 www.wrox.com/go/beginningaspnet4。由于本练习将重写 Monochrome 主题中的文件 Monochrome.css，因此如果要对那个文件作任何自定义修改，请先为它创建一个备份。

(1) 打开 Windows Explorer 并导航到从本章的 zip 文件(位置在 C:\BegASPNET\Resources)中提取的文件。打开 Chapter 06 文件夹，然后打开 Monochrome 文件夹。选择 Images 文件夹和 Monochrome.css 文件。

- (2) 从 Windows Explorer 窗口中将选中的文件夹和文件拖(或复制粘贴)到 VWD 中的 Monochrome 主题文件夹中。当出现询问是否重写 Monochrome.css 的提示时，单击 Yes 按钮。
- (3) 重复上面两个步骤，但这次只需要从 Windows Explorer 的 DarkGrey 文件夹中将 Images 文件夹拖动(或复制粘贴)到 VWD 中的 DarkGrey 主题文件夹中。这时 Solution Explorer 看起来应如图 6-17 所示。
- (4) 打开 MasterPages 文件夹中的母版页，并删除<a>标记之间的文本 Header Goes Here。由于页面的 header 部分填充了图像，因此不再需要该文本。
- (5) 通过右击 Default.aspx 页面并选择 View in Browser 命令在浏览器中请求 Default.aspx 页面。现在应该能看到含有 DarkGrey 主题中的图像的 Web 页面了，如图 6-18 所示。



图 6-18

- (6) 返回到 VWD，打开 web.config 文件并再次将<pages>元素的两个主题属性从 DarkGrey 切换为 Monochrome。在浏览器中打开 Default.aspx 页面，将看到带有新主题和图像的页面，如图 6-19 所示。

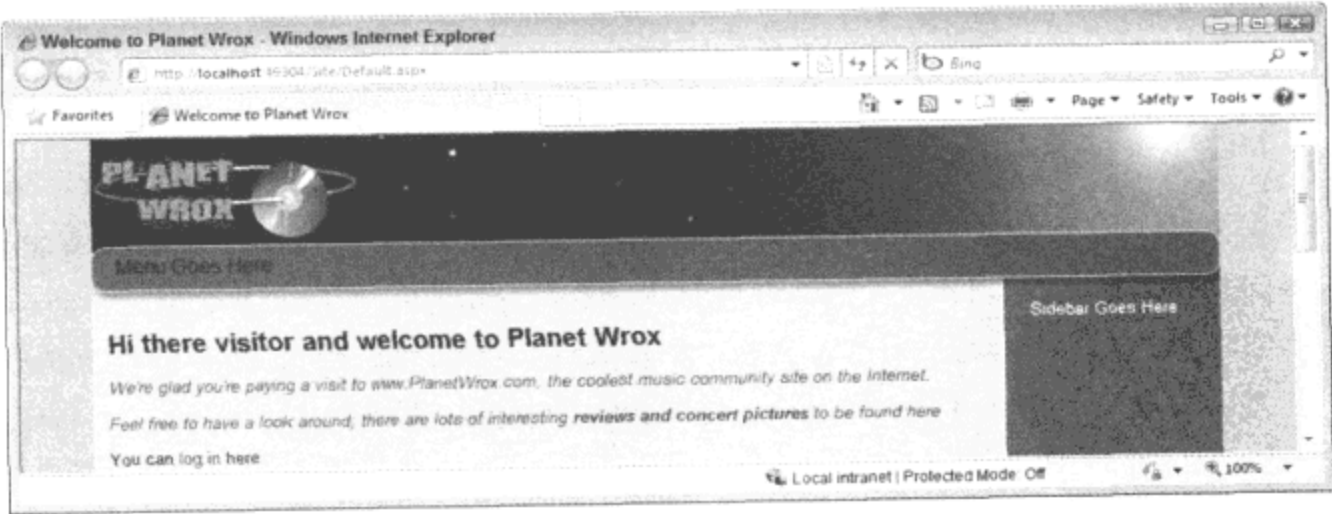


图 6-19

工作原理

从主题的角度来看，在前面“试一试”的练习中没有作太多的改变。正如以前看到的，主题添

加在页面的头部。然而，这次主题中的样式表指向位于主题文件夹中的图像。浏览器读取 CSS 文件，沿着链接找到图像，下载图像然后显示在代码文件中的各种 CSS 选择器所描述的恰当位置。

为这两种主题添加的 CSS 文件都包含大量注释，因此如果要知道 CSS 做了什么，请查看这两个主题文件夹中的文件。

尽管使用主题使页面开发人员能快速改变站点外观甚至是站点的布局，但如果让用户在运行时切换主题将更有用。通过这种方式，用户就能将站点定制为他们喜欢的样子。下一节将介绍如何实现动态切换主题。

6.3.5 动态切换主题


在运行时切换主题有几个好处。例如，可以通过允许用户用喜欢的颜色和布局选择主题来愉悦用户。并不是每个人都喜欢黑色背景白色文字的，因此可以在运行时改变它们将是很多人都喜欢的事。然而，也可以部署主题来帮助视力较弱的用户。创建一个对比鲜明的配色方案和大的字体，人们查看站点时就更轻松。Planet Wrox Web 站点中的主题仅仅改变了颜色和布局等屏幕元素，不过很容易创建这些主题的副本然后修改字体大小和配色方案。

由于使用的是在运行时向页面应用主题的方式，因此需要在页面的生命周期较早的时候设置主题，更确切地说是在 `PreInit` 事件中设置。Web 站点的基页仍然是设置主题的理想位置，因为站点中的所有页面都继承自这个类。

为了允许用户修改主题，可以提供给他们一个下拉菜单，当用户修改列表中的活动选项时，该菜单自动向服务器发一个回发。在服务器上，就会得到从列表中选择的主题，将它应用到页面上，然后将选项存储在 cookie 中，以便在接下来访问 Web 站点时检索它。

cookie 是能存储在用户计算机上的小块文本。存储在 cookie 中的数据仅能发送给首先设置它的服务器，因此其他站点读不到此 cookie。然而，由于 cookie 以纯文本的形式存储在用户的计算机上，因此永远不要将任何敏感数据(如密码)存储在 cookie 中。不过存储首选项主题等不会受到伤害的数据是 cookie 的极佳用法。

在下面两个“试一试”练习中，将介绍如何实现动态切换主题的功能。第一个“试一试”练习介绍如何将母版页修改为允许用户选择主题。这个练习仅仅检索用户选择并存储在 cookie 中的主题名称。第二个“试一试”练习说明了如何在运行时将那个主题应用到继承自 `BasePage` 的每个页面中。



注意：关于 cookie 以及它是否侵犯人的隐私方面有很多争议。一般而言，cookie 是安全的，因为它们只存储服务器设置的已经拥有的数据。如果用户自己没有把数据传送给服务器，cookie 是没法从计算机中窃取敏感数据的。在大多数情况下，cookie 改善了用户的浏览体验，因为它记住了小块数据，不用每次访问页面时都要重新输入。但是，有些大公司(如广告代理商)会使用独特的 cookie 来跟踪用户在 Web 上的轨迹，从而可以大体知道用户都浏览了哪些 Web 站点。为了确保站点访问者理解您拥有和保存了哪些信息，最好向站点中添加一个隐私语句，说明 cookie 的意图和用法，以及任何您可能会保存的个人数据。

试一试

让用户选择主题

在本练习中将向母版页添加一个 DropDownList 控件。这个控件包含一些可用的主题，因此用户可以从选择一个。用户的选择会存储在 cookie 中，以便以后再次使用。最后一步是当用户再次访问页面时在下拉列表中预先选好适当的主题。

(1) 在 Markup 视图中打开母版页，定位到名为 Sidebar 的<div>。删除静态文本 Sidebar Goes Here，用 DropDownList 控件替换它。替换方法是把它从 Toolbox 中拖到两个 div 标记之间。将控件的 ID 由 DropDownList1 改为 ThemeList。在下拉列表前面的换行符(
)后输入一些文本(例如，Select a Theme)来阐明列表的作用。

(2) 切换到 Design 视图，打开控件的 Smart Tasks 面板，并选择 Enable AutoPostBack。

(3) 在同一个 Smart Tasks 面板上，单击 Edit Items 链接并插入两个项：一个是文本 Monochrome，另一个是文本 DarkGrey。

(4) 双击下拉列表为 SelectedIndexChanged 事件建立一个事件处理程序。另外，还可以选择 DropDownList 控件，按 F4 键打开该控件的 Properties 面板，单击那个闪电按钮以切换到 Events 选项卡，然后双击 SelectedIndexChanged。图 6-20 以 Events 模式显示了 Properties 面板。

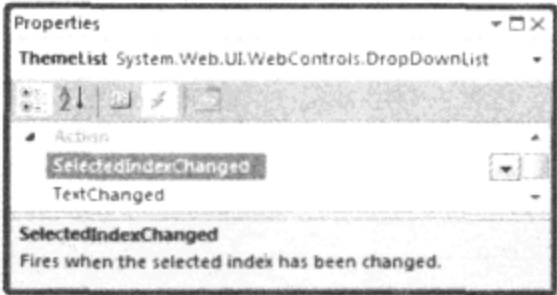


图 6-20

用户在客户端的下拉列表中作出新的选择后，就会激活服务器上 SelectedIndexChanged 处理程序中的代码。在处理程序块内，添加下列突出显示的代码，从列表中检索选中的主题并将其存储到 cookie 中：

VB.NET

```
Protected Sub ThemeList_SelectedIndexChanged(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles ThemeList.SelectedIndexChanged
    Dim preferredTheme As HttpCookie = New HttpCookie("PreferredTheme")
    preferredTheme.Expires = DateTime.Now.AddMonths(3)
    preferredTheme.Value = ThemeList.SelectedValue
    Response.Cookies.Add(preferredTheme)
    Response.Redirect(Request.Url.ToString())
End Sub
```

C#

```
protected void ThemeList_SelectedIndexChanged(object sender, EventArgs e)
{
    HttpCookie preferredTheme = new HttpCookie("PreferredTheme");
    preferredTheme.Expires = DateTime.Now.AddMonths(3);
    preferredTheme.Value = ThemeList.SelectedValue;
    Response.Cookies.Add(preferredTheme);
    Response.Redirect(Request.Url.ToString());
}
```


(5) 还是在母版页的 Code Behind 文件中，当页面加载时将需要再次添加一些代码来从列表中预先选择恰当的项。进行此操作的最佳位置是在 Page 类的 Load 事件中。如果使用的是 C#，Page_Load 处理程序应该总是在那儿。如果使用的是 Visual Basic，可以用两种不同的方式添加该处理程序：在 Design 视图中的任意地方双击页面(在 C#中也可以这么做)，或者从 Code Behind 中文档窗口上方左侧的下拉列表中选择(Page Events)，如图 6-21 所示，然后从第二个下拉列表中选择 Load。这也是为其他控件(像 Button 及 DropDownList 等控件)添加处理程序的好办法。

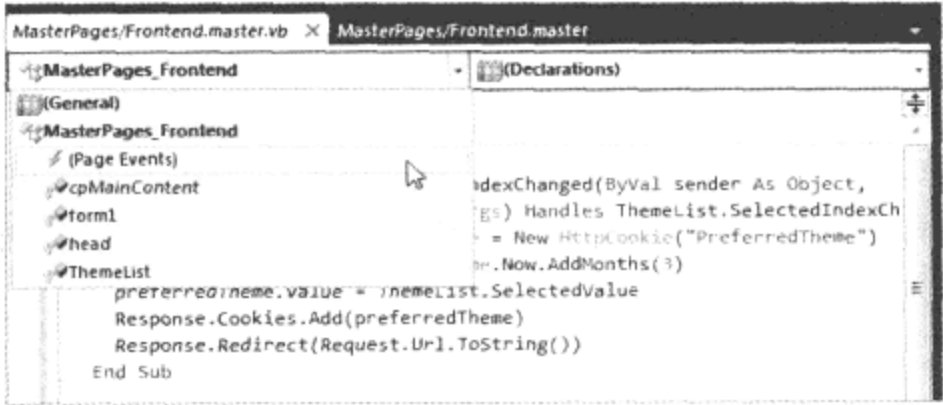


图 6-21

在 VWD 添加的处理程序块内，添加下面的代码：

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    If Not Page.IsPostBack Then
        Dim selectedTheme As String = Page.Theme
        Dim preferredTheme As HttpCookie = Request.Cookies.Get("PreferredTheme")
        If preferredTheme IsNot Nothing Then
            selectedTheme = preferredTheme.Value
        End If
        If Not String.IsNullOrEmpty(selectedTheme) AndAlso
            ThemeList.Items.FindByValue(selectedTheme) IsNot Nothing Then
            ThemeList.Items.FindByValue(selectedTheme).Selected = True
        End If
    End If
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        string selectedTheme = Page.Theme;
        HttpCookie preferredTheme = Request.Cookies.Get("PreferredTheme");
        if (preferredTheme != null)
        {
            selectedTheme = preferredTheme.Value;
        }
        if (!string.IsNullOrEmpty(selectedTheme) &&
            ThemeList.Items.FindByValue(selectedTheme) != null)
        {
            ThemeList.Items.FindByValue(selectedTheme).Selected = true;
        }
    }
}
```



```
}  
}  
}
```

(6) 保存所有修改，然后在浏览器中再次请求 Default.aspx。边栏中的下拉列表应显示已选中列表中的第一项。从列表中选择其他选项，页面就会重新加载。上次在下拉列表中选择项现在应当已在下拉列表中预先选择了，注意这时可以发现看到的一直是同一个主题，因为还没有写应用选中主题的任何代码。在后面的“试一试”练习中将介绍如何写这样的代码。



常见错误：如果出现错误，请确保输入代码时没有拼写错误。如果什么也没有发生(例如，页面没有回发)，请检查一下有没有将 DropDownList 控件上的 AutoPostBack 属性设置为 True。如果根本没有看到任何应用的主题，则要在 web.config 文件中对配置的主题检查<pages>元素，另外还要在两个代码块中检查 cookie 的名称 PreferredTheme 的拼写是否正确。

工作原理

在上面的练习中对母版页作了 3 个重要修改。首先，添加了下拉列表并将 AutoPostBack 设置为 True。这样，从列表选择一个新项后，页面就会将自身提交回服务器。当发生这种情况时，SelectedIndexChanged 处理程序中的代码就会激活。该代码创建了一个可以存储在用户浏览器中的 cookie。要使 cookie 在浏览器会话之间持续存在，需要设置 Expires 属性。在下面的代码示例中，cookie 被设置为从现在起的 3 个月后过期，这意味着在这个时间段后浏览器会自动丢弃它。然而，每当用户选择一个新主题时，这个日期都会延长，又一次将它设置为 3 个月。

VB.NET

```
Dim preferredTheme As HttpCookie = New HttpCookie("PreferredTheme")  
preferredTheme.Expires = DateTime.Now.AddMonths(3)
```

C#

```
HttpCookie preferredTheme = new HttpCookie("PreferredTheme");  
preferredTheme.Expires = DateTime.Now.AddMonths(3);
```

创建了 cookie 后，就可以设置它的 Value 属性。在本例中，DropDownList 的 SelectedValue(包含主题的名称)存储在 cookie 中。然后用 Response.Cookies.Add 将该 cookie 添加到 Cookies 集合中：

VB.NET

```
preferredTheme.Value = ThemeList.SelectedValue  
Response.Cookies.Add(preferredTheme)
```

C#

```
preferredTheme.Value = ThemeList.SelectedValue;  
Response.Cookies.Add(preferredTheme)
```

注意，cookie 被添加到与用户的响应关联的 Response 对象的 Cookies 集合中。以后会看到如何

再次从与用户发出的页面请求相关联的 Request 对象的 Cookies 集合中读取该 cookie。
最后一步是将用户重定向到同一个页面：

VB.NET

```
Response.Redirect(Request.Url.ToString())
```

C#

```
Response.Redirect(Request.Url.ToString());
```

这是必需的，因为不这样的话就不会立即应用新主题。因为主题需要在页面的生命周期的早期设置，所以不能再为当前的请求设置。通过将用户重定向到同一个页面，就发出了一个能够成功应用选中主题的新请求。下面的“试一试”练习显示了如何用代码以编程的方式设置选中的主题。

对母版页做的最后一个改变是修改 Page_Load 处理程序。在这个方法内，声明了 String 变量，这个变量用来查看 Page.Theme 以保存当前的活动主题。它将成为默认主题，如果用户没有保存喜欢的主题的 cookie，将预先从下拉列表中选择这个默认主题。然后代码会查看有没有名为 PreferredTheme 的 cookie。如果存在，就用它的值作为字符串 selectedTheme 一个新值。最后，就会用这个 String 变量在下拉列表中找到这个项并预先选中它。

这样，下拉列表就总是会显示当前配置的站点主题或者用户手动选择的项，即使用户下星期再回到站点时仍是如此。注意 FindByValue 方法在 DropDownList 控件的 Items 集合上的使用。这个方法会返回它找到的项，如果该项不存在则返回 Nothing(在 C#中是 null)。这可以确保当 cookie 中包含不再存在的主题时，代码不会预先选中列表中不存在的项。

有了这种让用户适当选择主题的能力后，下一步就是应用所选择的主题。

正如前面所提到的，主题需要在 PreInit 事件(该事件在页面生命周期的早期发生)中设置。在该事件内可以查看带选中主题的 cookie 是否存在。如果存在，就可以用它的值设置恰当的主题。

试一试

应用用户选中的主题

在本练习中，将修改基页并为 PreInit 事件添加一些代码来设置用户的主题。

(1) 从 App_Code 文件夹中打开 BasePage 文件，并添加下列设置在 PreInit 事件发生期间选中的主题的代码。可以将该代码放在检查页面标题的方法前面或后面。

VB.NET

```
Private Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs) _  
    Handles Me.PreInit  
    Dim preferredTheme As HttpCookie = Request.Cookies.Get("PreferredTheme")  
    If preferredTheme IsNot Nothing Then  
        Page.Theme = preferredTheme.Value  
    End If  
End Sub
```

C#

```
private void Page_PreInit(object sender, EventArgs e)  
{  
    HttpCookie preferredTheme = Request.Cookies.Get("PreferredTheme");  
    if (preferredTheme != null)  
    {
```



```
        Page.Theme = preferredTheme.Value;
    }
}
```

(2) 如果使用的是 C#, 则与前面“试一试”练习中的 PreRender 事件处理程序一样, 需要为 PreInit 事件在类的构造函数中建立一个事件处理程序。这样就可以告知 ASP.NET 运行库哪个方法会处理 PreInit 事件:

```
public BasePage()
{
    this.PreRender += new EventHandler(Page_PreRender);
    this.PreInit += new EventHandler(Page_PreInit);
}
```

(3) 保存对所有打开文档的修改, 然后在浏览器中请求 Default.aspx。页面应当用前面“试一试”练习的下拉列表中最后选择的主题加载。

(4) 从列表中选择一个新的项。页面应重新加载, 并且应显示其他主题。

如果在浏览器中发现页面显示了两个主题的组合, 则回到 VWD, 打开 web.config, 并从<pages>元素中删除 StyleSheetTheme 属性, 而由于 theme 属性将作为新访问者的默认属性, 因此保留它。

工作原理

获得用户喜欢的主题并将它存储在 cookie 中的艰难工作已经完成, 接下来应用主题就非常容易了。PreInit 事件处理程序中的代码会首先确认有没有名为 PreferredTheme 的 cookie。方法是通过将 Get 方法的返回值与 Nothing(在 C#中是 null)作比较。

VB.NET

```
Dim preferredTheme As HttpCookie = Request.Cookies.Get("PreferredTheme")
If preferredTheme IsNot Nothing Then
```

C#

```
HttpCookie preferredTheme = Request.Cookies.Get("PreferredTheme");
if (preferredTheme != null)
```

这里, Request.Cookies 用来从用户的浏览器与请求一起发送的 cookie 中读取信息。如果该 cookie 存在, 就用它的 Value 属性设置恰当的主题:

VB.NET

```
Page.Theme = preferredTheme.Value
```

C#

```
Page.Theme = preferredTheme.Value;
```

因为主题在页面的生命周期的早期设置, 所以这个设置将一直用到页面结束, 从而有效地将主题中定义的外观赋予页面。注意, 当 cookie 包含一个站点中不再存在的主题时, 这个代码会导致页面崩溃。在第 18 章, 您将会学到更多有效处理这种情况的技术。

通过编程设置主题的功能, 可以向用户提供一种快速且容易的方式将页面修改为他们喜欢的样子。这样的主题会影响整个 Web 站点中各个页面的颜色和布局。与母版页结合在一起就产生了影响

整个页面外观的灵活方式。如果能修改页面上的某些控件，也是有用的。例如，可能需要赋予站点中每个按钮相同的外观。这就是 ASP.NET 外观出场的时候了。

6.4 外观

外观是包含标记的简单文本文件，它允许从某个集中位置定义一个或多个服务器控件的外观。它们位于主题文件夹下，是构成 ASP.NET 主题功能的一个重要部分。外观文件(扩展名为.skin)包含控件的服务器端表现元素。然后这些设置会应用到外观所应用到的所有控件上。要了解其工作原理，考虑下面这个定义 Button 控件的外观的示例：

```
<asp:Button BackColor="#cccccc" ForeColor="#308462" runat="server" />
```

用了这个外观定义，站点中的所有按钮的 BackColor 属性都会是#cccccc，ForeColor 属性是#308462。所需做的只是在主题文件夹下面创建一个 skin 文件，向它添加这个标记即可。从那时起，所有按钮都会自动修改。与前面看到的直接设置控件上的属性一样，像 BackColor 和 ForeColor 这样的属性会转换为客户端 HTML 和 CSS。

注意，这个外观标记与按钮标记类似。不过有几个区别。第一个区别是，skin 文件中的控件不能有 ID 属性。ID 用来唯一标识页面中的控件，由于外观是应用到所有控件的，因此没有必要给它一个 ID 属性。另一个区别在于能在标记中设置的属性数目。并不是控件的所有属性都能应用外观。例如，不能通过一个 skin 文件设置 Button 按钮的 Enabled 属性。Microsoft 的 MSDN 文档列出了每个属性是否能应用外观的列表。想知道是否可以对某个属性应用外观的另一种方法是直接试试能不能应用；如果设置 Skin 文件中的属性时不允许设置，就会得到一个运行时错误。

一般来说，影响外观的属性(BackColor、ForeColor、BorderColor 等)可以应用外观，而影响行为的属性(Enabled、EnableViewState 等)不能应用外观。

在用 Add New Item 对话框创建新的外观文件时，会发现服务器端注释块中已经包含了一些文本。可以放心地删除这些注释，因为它们只是提供了一个简短示例来表明外观的工作方式。在单个 skin 文件中可以定义多个控件。然而，从可维护性的角度来看，最好根据所表示的控件来命名各个 skin 文件。例如，可以将按钮的 skin 文件命名为 Button.skin，标签的 skin 文件命名为 Label.skin 等。

不要直接向 skin 文件中的控件属性(从而向页面中的最终标记)应用格式化元素，最好用 CssClass 属性指向 CSS 文件之一中的一个 CSS 类。这样，就更容易进行站点范围的修改，而且不会使最终 HTML 膨胀。对于前面的示例，一个带如下所示的外观定义的文件与该主题的 CSS 文件中的一个类给出的效果是相同的：

```
<asp:Button CssClass="MyButton" runat="server" />

.MyButton
{
    color: #308462;
    background-color: #cccccc;
}
```

6.4.1 创建一个 skin 文件

skin 文件必须直接在主题的文件夹中创建。不能像存储主题的图像那样把它们存储在一个子文件夹中。在下面的“试一试”练习中将看到如何创建一个简单的 skin 文件来修改 Web 站点中所有按钮控件的外观。本书以后的章节将建立于这一知识基础上，在那些章节中会为其其他控件(如 GridView)定义更复杂的外观。

当开始在 skin 文件中输入时，会看到熟悉的 IntelliSense 并没有出现。这使得定义自己的控件及其属性比较困难。然而，有一种简单的折中方法：

- (1) 通过选择 Tools | Options 命令打开 VWD 的 Options 对话框。
- (2) 展开 Text Editor 选项然后单击 File Extension。如果没有看到这些选项，则确保选中了屏幕底部的 Show All Settings 选项。
- (3) 在 Extension 框中输入 skin，然后从 Editor 下拉列表中选择 User Control Editor。
- (4) 单击 Add 按钮，然后单击 OK 按钮关闭 Options 对话框。

从现在起，可以从 skin 文件(如果已经创建了，则首先需要重新打开已有的 skin 文件)中得到 IntelliSense。使用该设置，当打开一个 skin 文件时，会在 Error 列表中得到关于构建提供者(build provider)的警告。可以安全地忽略这个错误，因为即使 VWD 中包含了这些设置，skin 也仍会在运行时正常工作。

试一试

为按钮控件创建外观

为了有效地使用外观，应尽可能地使用 CssClass 属性，而不是应用出现在页面的最终 HTML 中从而增加页面大小和加载时间的内联属性。然而，为了说明如果有添加内联属性的特殊需要时它的工作原理，因此本练习介绍了如何应用这两种方法。

- (1) 在 Monochrome 主题文件夹中，添加一个新的 skin 文件并命名为 Button.skin。通过右击 Monochrome 文件夹并选择 Add New Item 命令添加这个文件。在接着出现的对话框中选择 Skin File 然后输入 Button 作为文件名。
- (2) 删除文件中的所有内容，然后添加下列代码：

```
<asp:Button CssClass="MyButton" BackColor="#7a70a4" runat="server" />
```

注意，这个标记结合了内联属性的样式(BackColor)与指向 CSS 文件中的选择器的 CssClass。如前所述，可以忽略关于缺失构建提供者的警告，因为 skin 文件在运行时会很好地工作。只要关闭了 skin 文件，警告就会消失。

- (3) 从主题文件夹中打开 Monochrome.css 文件，并向文件的末尾添加此 CSS 选择器：

```
.MyButton
{
    color: White;
}
```

- (4) 在 Demos 文件夹中创建一个新的 Web 窗体并命名为 SkinsDemo.aspx。确保基于以前创建的导出模板来创建这个 Web 窗体。赋予页面的 Title 值为 Skins Demo，然后将 Toolbox 中的 Button 拖到页面的 cpMainContent 区域来添加一个 Button 控件。最终代码如下：


```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <asp:Button ID="Button1" runat="server" Text="Button" />
</asp:Content>
```

(5) 保存所有修改，然后在浏览器中请求 `SkinsDemo.aspx`。如有必要，切换到 `Monochrome` 主题。在第(4)步中添加的按钮现在应是紫色背景，上面的文本为白色。如果修改后的颜色没有出现，则确保在下拉列表中选择了合适的主题，并添加 `MyButtonCSS` 类到 `Monochrome` 主题的 CSS 文件中。如果仍然看不到其修改，则按 `Ctrl+F5` 或 `Ctrl+R` 键，强制从服务器的 CSS 文件中获取新的副本。

工作原理

要了解上面的“试一试”练习的工作原理，应在浏览器中看一下页面的 HTML。`Button` 控件已转换为下面的 HTML：

```
<input type="submit" name="ctl00$cpMainContent$Button1" value="Button"
    id="cpMainContent_Button1" class="MyButton" style="background-color:#7a70a4;" />
```

`skin` 文件中的 `CssClass` 和 `BackColor` 属性都添加到了 HTML 中。前者最终出现为按钮上的 `class` 属性，而后者转换为 `style` 属性。CSS 文件中的 `MyButton` 类设定了按钮的白色显示文本，而内联样式则确定按钮的背景色。如果在下拉列表中选择 `DarkGrey` 主题，并再次查看 HTML，将发现没有 `class` 属性和 `style` 属性来给按钮提供默认的 Windows 外观。

可以看出，外观极易使用，可以在很大程度上改变站点中特定控件的外观。但是，如果不希望所有按钮同时变成紫色和白色该怎么办呢？如果需要一个红色背景的特殊按钮该怎么办呢？可以使用已命名外观(named skin)来实现。

6.4.2 已命名外观

已命名外观与正常外观相同，除了它有一个 `SkinID` 集合，允许根据名称引用该外观。然后，ASPX 页面中的控件可以用 `SkinID` 将特定外观应用到控件上。下面的“试一试”练习说明了其工作原理。

试一试

为按钮控件创建一个已命名外观

创建已命名外观最容易的方式是复制现有外观的代码，然后添加一个 `SkinID` 属性。注意，如果复制和粘贴一个外观定义，VWD 会自动添加一个 `ID` 属性(即像前面介绍的将 `skin` 文件连接到 `User Control Editor` 时出现的情况)。这个 `ID` 是不允许出现的，因此一定要删掉它。

- (1) 再次打开 `Button.skin`，复制所有代码，然后将它粘贴在现有标记下面。
- (2) 如果 VWD 添加了一个 `ID` 属性，则删除它和它的值(例如，删除 `ID="Button1"`)。
- (3) 删除 `CssClass` 属性及它的值，将按钮的 `BackColor` 修改为 `Red`，然后将 `ForeColor` 设置为 `Black`。
- (4) 添加 `RedButton` 的 `SkinID` 属性。最终代码应是这样：

```
<asp:Button CssClass="MyButton" BackColor="#7a70a4" runat="server" />
<asp:Button BackColor="Red" ForeColor="Black" SkinID="RedButton" runat="server" />
```


(5) 保存并关闭 skin 文件。

(6) 打开 SkinsDemo.aspx 并添加第二个按钮。将这个按钮的 SkinID 设置为 RedButton。注意 IntelliSense 可帮助找到正确的 SkinID。这两个按钮的代码如下所示：

```
<asp:Button ID="Button1" runat="server" Text="Button" />
<asp:Button ID="Button2" runat="server" Text="Button" SkinID="RedButton" />
```

(7) 在浏览器中打开 SkinsDemo.aspx。现在应看到两个不同颜色的按钮。如果没有看到不同的颜色，则要确保在浏览器中选择了 Monochrome 主题。

工作原理

已命名外观的工作方式几乎与正常外观相同。然而，使用已命名外观，控件可以指向 skin 文件之一中的特定外观。在 SkinsDemo.aspx 页面中，第一个按钮从默认的未命名外观中得到了设置，而另一个按钮现在得到了 SkinID 设置为 RedButton 的外观。

有了已命名外观，就有了可自由支配的非常灵活的解决方案。可以用正常外观快速改变站点中所有控件的外观。然后用一个已命名外观重写希望看起来不一样的少数控件的行为。

6.4.3 对特定控件禁用主题

如果由于某种原因不想向特定控件应用外观，可以通过设置控件的属性 EnableTheming 来禁用外观，如：

```
<asp:Button ID="Button1" runat="server" EnableTheming="False" Text="Button" />
```

由于 EnableTheming 设置为 False，因此就不会向控件应用外观。而仍然会应用主题的 CSS 文件中的 CSS 设置。

6.5 创建一致页面的实用提示

下面的列表提供了关于创建一致页面的一些实用提示：

- 当创建新的 Web 站点时，总是先添加作为所有其他页面的基础的母版页。即使站点中只有少数几个页面，母版页仍然可以帮助确保整个站点拥有一致的外观。在后面的阶段向站点添加母版页意味着要对现有页面作大量手动修改，因此从一开始就添加母版页可以使以后少做很多工作。
- 一旦发现向复杂控件(如第 7 章将介绍的 TreeView 和 Menu)或者数据识别控件(如第 13 章将介绍的 GridView)添加了样式化信息，就要考虑为它们创建一个外观。可以从单个位置控制所有相似控件布局的这一事实使站点的更新变得更容易。如果要重写几个控件的布局，可以使用有一个 SkinID 的已命名外观，或者通过 EnableTheming 设置为 False 来完全禁用外观。

- 当创建外观或直接在控件上设置样式属性时，则要考虑使用 `CssClass` 属性代替，然后将所有与样式化相关的属性添加到站点或主题的 CSS 中。这样会减小页面的大小，使得以后很容易改变布局。
- Visual Web Developer 的 Export Template 功能可以大大节省时间。它不仅能用来为 ASPX 页面及其 Code Behind 创建模板，而且可以为其他文件(如类和 CSS 文件甚至是一个完整的 Web 站点)创建模板。这样在创建新文件时就有了一个高的起点，不用一次又一次地重复输入同样的内容。

6.6 本章小结

站点中所有页面保持一致的外观有利于使站点看起来更专业和有吸引力。这样还有助于访问者在站点中找到正确的信息，增加他们再次访问站点的可能性。ASP.NET 4 提供了大量帮助创建外观一致的 Web 站点的工具。

ASP.NET 的母版页和内容页用于帮助创建布局，可以在每个基于该母版页的页面中重复使用这个布局。

当母版页定义了一个集中化的外观时，就可以使用 `Base Page` 类来集中页面的行为，例如检查无效的页面标题。

主题用于改变站点中页面的外观以及它们所包含的控件。由于主题包含 CSS 文件、图像和外观，因此可以直接通过应用主题来改变页面的颜色、字体、位置和图像。通过很好地使用一些技术(如已命名外观)和 `EnableTheming` 属性，可以创建一个应用到整个站点的设计，同时不用再为单个控件设计来保持设计的灵活性。

Planet Wrox Web 站点现在开始扩大了。这意味着您和您的访问者会更难找到正确的页面。第 7 章将介绍用户导航站点的若干不同方式，通过导航可以轻松地找到想要查找的页面。

6.7 练习

1. `ContentPlaceHolder` 与 `Content` 控件之间的区别是什么？在各种类型的页面中使用哪个控件？
2. 如何将内容页中的 `Content` 控件与母版页中的 `ContentPlaceHolder` 关联起来？
3. 设想用下面的外观定义创建了一个应用到站点中所有按钮的外观：

```
<asp:Button runat="server" CssClass="MyButton" />
```

该 CSS 类 `MyButton` 设置按钮的背景色为黑色，前景色为白色。为了使页面中的一个特定按钮引人注目，决定给它设置一个红色背景。有哪些方法可以控制这个按钮的外观？

4. 解释设置页面的 `Theme` 属性与 `StyleSheetTheme` 属性之间的区别。
5. 指出设置页面 `Theme` 属性的 3 种不同方式，并解释这些方式之间的区别。

6. 在 Web 站点中实现基页的主要原因是什么？

练习的答案见附录 A。

本章要点回顾

基页	一个类，该类继承自 ASP.NET 页面类，并作为 ASPX 页面的父类
内容页	一个 ASPX Web 窗体，该窗体基于母版页创建其整体外观和布局
Cookie	可以存储在用户计算机中并能从服务器再次访问的文本片段
母版页	一个模板页，可通过该页面来定义内容页
已命名外观	一个 ASP.NET 外观，该外观使用显式的 SkinID 设置并允许通过其 ID 引用它
页面生命周期	当浏览器发出请求时，ASPX 页面所经历的事件周期
外观	一个包含外观设置的集合，该集合能影响浏览器中控件的外观
主题	一个由 CSS 样式、外观和图像组合成的集合，可用来改变站点中页面的外观

第7章

导 航

本章要点

- 如何使用服务器控件和纯 HTML 在站点中到处移动
- 如何在站点中处理页面及其他资源(如图像)
- 如何使用 ASP.NET 的 Menu、TreeView 和 SiteMapPath 导航控件
- 如何通过编程将用户从一个页面导航到另一个页面

当站点包含的页面比较多时，有一个稳固而清晰的导航结构就很重要，这样才能让用户顺畅地浏览站点。使用良好的导航系统，项目中所有没有连接的 Web 页面就会形成一个完整而连贯的 Web 站点。

当考虑导航系统的重要部分时，可能遇到的第一个对象就是菜单。菜单有各种各样的类型与大小，从简单的静态 HTML 链接到复杂的由 CSS 或 JavaScript 驱动的折叠式菜单。但是导航要处理的不仅仅是菜单。ASP.NET 附带了大量有用的导航控件，可以用来立刻建立一个导航系统。这些控件包括 Menu、TreeView 和 SiteMapPath，本章接下来会介绍。

除了 Menu 这样的可视化控件外，导航也需要考虑结构。在组织良好的站点中，用户就容易导航。导航控件用 Websitemap 文件帮助定义站点的逻辑结构。

导航的另一个重要部分发生在服务器端。基于某些条件在 Code Behind 中将用户从一个页面导航到另一个页面是非常普遍的情况。例如，设想一名管理员进入 Web 站点 Management 部分的新 CD 或音乐会评论中。当写完评论后，可能需要重定向到一个新页面上向管理员显示它的全部细节。

本章将介绍如何使用可自由支配的各种导航方法。在学习内置导航控件前，需要了解在站点中处理资源(如 ASPX 页面与图像)的不同方法。

7.1 在站点中移动

让用户从一个页面转移到另一个页面最常用的方式是使用<a>元素。这个元素有一个 href 特性，允许定义要链接到的页面或其他资源的地址。可以在标记之间放置要链接的内容，如文本、图像或

其他 HTML。下面的代码段显示了元素的一个简单示例：

```
<a href="Login.aspx">You can log in here</a>
```

在 Web 页面中使用了这段代码后，用户只要单击文本 “You can log in here”，就会被带到页面 Login.aspx，这个页面应当与包含此链接的页面在同一个文件夹中。

<a>元素有一个服务器端对应物，名为 HyperLink，可以用<asp:HyperLink>在标记中创建。它最终会以<a>元素出现在页面中。这个控件的 NavigateUrl 属性(property)直接映射为<a>元素的 href 特性(attribute)。例如，内容页中的服务器端 HyperLink 如下所示：

```
<asp:HyperLink runat="server" id="LoginLink" NavigateUrl="Login.aspx">
    You can log in here</asp:HyperLink>
```

在浏览器中生成如下的 HTML：

```
<a id="cpMainContent_LoginLink" href="Login.aspx">You can log in here</a>
```

除了 ASP.NET 运行库指定的长 ID 外，这段代码等同于上一个示例。在这两种情况下，href 特性都使用相对 URL 指向页面 Login.aspx。下面将描述相对 URL 与绝对 URL 之间的区别。

7.1.1 理解绝对 URL 与相对 URL

在站点中使用链接的关键是要充分了解 Web 站点内外的资源可以采用的统一资源定位符(Uniform Resource Locator, URL)的不同形式。URL 用来唯一地标识您的或另一个 Web 站点中的资源。这些 URL 用在不同的地方，包括超链接的 href 特性或者指向 CSS 文件的<link>元素，指向一个图像或者一个 JavaScript 源文件的 src 特性和 CSS 属性的 url()值。这些 URL 可以表达为相对 URL 或绝对 URL。两种方法各有优缺点。

1. 相对 URL

在前面的示例中出现了指向相对于使用 URL 的位置的另一个资源的相对 URL。这意味着包含<a>元素的页面和 Login.aspx 页面应放在站点上的同一个文件夹中。要引用其他文件夹中的资源，可以使用下面的 URL。所有示例都基于图 7-1 所示的站点结构。

要将根文件夹中的 Login.aspx 页面链接到 Management 文件夹中的 Default.aspx，可以使用下面这个 URL：

```
<a href="Management/Default.aspx">Management</a>
```

要从 Management 文件夹中的 Default.aspx 页面中引用图像 Header.jpg，可以使用下面的 URL：

```

```

开头的两个句点将一个文件夹导航到根文件夹，然后回到 Images 文件夹中以指向 Header.jpg。对于较深的文件夹层次结构，可以用多个双句点，一个用于要沿站点层次结构向上走的各个文件夹，如这个元素，它可以用来引用 Reviews 文件夹的页面中的同一个图像，Reviews 文件夹

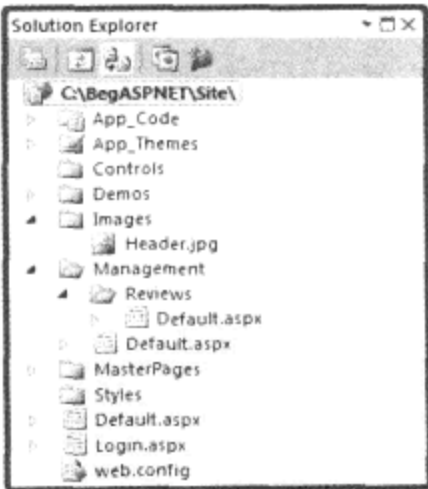


图 7-1

位于 Management 文件夹下：

```

```

相对 URL 的一个好处是，可以将一组文件移到同级的另一个目录下，且不会中断它们的内部链接。然而，同时它们也使得将文件移到站点层次结构中的不同层变得更困难。例如，如果将页面 Login.aspx 移到一个单独的文件夹中，如 Members，那么对 Management 文件夹的链接就会中断。新的 Members 文件夹中就没有了 Management 这一子文件夹，因此 Management/Default.aspx 就不再是有效链接。

要克服这个问题，可以使用基于根文件夹的相对 URL。

基于根文件夹的相对 URL

基于根文件夹的相对 URL 总是以表示站点根文件夹的正斜杠开头。如果仍以对 Management 文件夹的链接为例，它的基于根文件夹的版本就会有如下表示：

```
<a href="/Management/Default.aspx">Management</a>
```

注意 Management 文件夹的前面表示 Web 站点根文件夹的开头的正斜杠。这个链接没有歧义。它总是指向根文件夹的 Management 文件夹中的文件 Default.aspx。有了这个链接，即使将 Login.aspx 页面移到子文件夹也不会中断它；它仍然会指向同一个文件。

服务器端控件中的相对 URL

使用 ASP.NET 服务器控件，就有了另一种可自由支配的引用 Web 站点中资源的方法：可以用符号(~)指向站点的当前根文件夹。要了解这个符号解决了什么问题，需要知道 VWD 创建新 Web 站点并将它们与内置 Web 服务器挂钩的方式。当创建一个新的 Web 站点时，VWD 会默认在内置 Web 服务器(所有页面都在这个服务器中创建)下的独立应用程序文件夹中创建一个站点。因此，假设创建了一个新站点，并按下 Ctrl+F5 键来打开浏览器中的默认页面，最后就得到类似这样的地址：http://localhost:23143/MySiteName。通常，当将站点放在远程服务器上时，就不再需要这个应用程序文件夹。取而代之的是，用户会定位到一个 URL，如 http://www.PlanetWrox.com，并期望能看到这个站点。幸运的是，有一种容易的方式可以阻止 VWD 创建这个独立文件夹：可以用 Web 站点的 Properties 面板将项目的 Visual Path 属性设置为正斜杠(/)。

为了查看这种设置如何影响对文件的引用，下面的“试一试”练习说明了如何创建一个新的 Web 站点，并向它添加一些使用不同的 URL 命名模式的图像。然后改变虚拟路径属性，以便查看该属性有什么作用。

试一试

研究虚拟路径属性的行为

在本“试一试”练习中，将创建一个仅用于本练习的崭新的 Web 站点。本练习结束后可以删除这个 Web 站点。对于本练习，还需要一个图像文件——例如第 6 章中的 Header.jpg；任何其他图像也可以。

(1) 在 VWD 中选择 File | New Web Site 命令创建一个崭新的 Web 站点。

(2) 选择 ASP.NET Empty Web Site 模板，将 Web Location 设置为 File System，并注意到 VWD 提供了末尾为 WebSite1 的路径。如果以前创建了其他 Web 站点而没有给它们指定明确的名称，那

么这个路径可能会是 WebSite2、WebSite3 等。单击 OK 按钮创建站点。

(3) 向站点中添加一个名为 Default.aspx 的新 Web 窗体(使用默认的 Web 窗体选项而不是自定义的模板)，然后向站点的根文件夹中添加一个图像，并命名为 Header.jpg。可以从第 6 章的站点根文件夹中的图像里选择一个，或者采用现有图像并添加到站点中。如果没有把图像重命名为 Header.jpg，那么在下一步中一定要调整一下代码。

(4) 在 Default.aspx 中，向 Markup 视图中添加下列代码以插入 3 个 ASP.NET Image 控件，可以用 3 种不同的方式处理图像。这些图像之间用一个换行符隔开：

```
<asp:Image ID="Image1" runat="server" ImageUrl="Header.jpg" /><br />
<asp:Image ID="Image2" runat="server" ImageUrl="/Header.jpg" /><br />
<asp:Image ID="Image3" runat="server" ImageUrl="~/Header.jpg" /><br />
```

(5) 按下 Ctrl+F5 组合键在浏览器中打开页面。注意浏览器地址栏的地址类似于 http://localhost:49696/WebSite1/Default.aspx。您的端口号和应用程序名可能与这里略有不同，但是重要的是要注意 Web 站点位于 Web 服务器下名为 localhost 的独立文件夹中。还可以发现显示的第二个图像被中断了。这是因为开头的斜杠引用了 Web 服务器的根文件夹，因此会在 http://localhost:49696/Header.jpg 这个位置寻找图像，但在这个位置并没有，因为图像位于 WebSite1 子文件夹中。

在浏览器中打开页面的源代码，看一下 3 个元素：

```
<br />
<br />
<br />
```

前两个 URL 与添加到 ASPX 页面的 URL 完全相同。然而第三个 URL 已经被改为引用该图像的页面所在的同一文件夹中的图像。

(6) 关闭浏览器后回到 VWD，在 Solution Explorer 中单击该 Web 站点的根文件夹，然后按下 F4 键打开 Web 站点的 Properties 面板。将 Virtual Path 由/WebSite1 改为/，如图 7-2 所示。

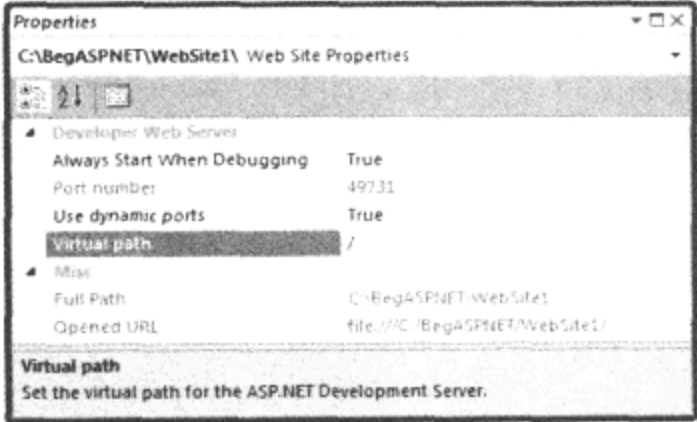


图 7-2

(7) 再次按下 Ctrl+F5 组合键在浏览器中重新打开 Default.aspx 页面。地址栏现在会显示类似这样的地址：http://localhost:49696/Default.aspx。从中可以看出，页面 Default.aspx 现在位于服务器的根文件夹中。因此，3 个图像现在都会正确地显示。

(8) 回到 VWD 中，并创建一个名为 Test 的文件夹。从站点的根文件夹中将 Default.aspx 文件拖到这个新文件夹中，然后在浏览器中请求页面。这次第一个图像会被中断。如果打开 HTML 源代码，将看到这样的代码：


```
<br />
<br />
<br />
```

第一个元素试图找到一个相对于当前文档的图像。由于当前文档位于 Test 文件夹中，而图像位于站点的根文件夹中，因而导致了被中断的图像。另外两个 src 属性指向站点根文件夹中的正确图像。

(9) 现在可以关闭 VWD 中的这个测试项目，并从磁盘中删除它，因为不再需要它了。

工作原理

这个示例演示了用~语法表示站点的根文件夹通常是引用资源(如图像)的最可靠的方式。在本“试一试”练习中看到的 3 种情况中，只有第 3 个图像每次都会正确显示。当服务器控件(如 Image 控件)需要显示一个路径时，它就会找出当前应用程序根文件是什么，并相应地调整路径。当要将文件夹和文件移来移去时，或者当使用 VWD 的附加应用程序文件夹的默认设置来开发站点，并在之后将站点放在生产服务器上(它允许访问服务器实际根文件夹中的文件)时，用这种方法会大大节省时间。

纯 HTML 控件也能通过将它们改为服务器控件来从这种语法中获益。只要向元素中添加一个 runat="server"特性即可。可以输入代码段 runat，然后按一次或两次 Tab 键(取决于 IntelliSense 列表是否打开)以快速地将必需的特性添加到任一元素。有了这个特性，下面的元素就能展示与前面示例中第三个<asp:Image>相同的行为：

```
<br />
```

在纯客户端 HTML 元素和 CSS 文件中不能使用~语法。即，如果没有通过为 img 元素添加 runat="server" 特性使其变为一个服务器端图像，但是却试图使用前面的 src 特性，或者试图在 CSS 文件或代码块中使用~语法时，图像将不会出现在浏览器中。这些情况下应该使用相对链接。

本书的其余部分假定将 Virtual Path 属性设置为正斜杠(/)，以便更容易用不带~语法的 URL 引用文件。如果发现有些资源没有正确地显示，请检查有没有正确地设置 Web 站点的虚拟路径属性。

2. 绝对 URL

与从文档或站点根文件夹的角度引用资源的相对 URL 相反，也可以使用根据完整路径引用资源的绝对 URL。因此，它不是直接引用图像并可选地指定一个文件夹，而是包括了域和协议信息的完整名称(http://前缀)。下面是引用 Wrox Programmer to Programmer 站点(http://p2p.wrox.com)上的 Wrox logo 的示例，在那里可以找到关于本书和其他 Wrox 图书的问题，或者关于编程的一般问题：

```

```

如果要引用自己的 Web 站点之外的资源，就必须使用绝对 URL。使用这样的 URL 时，http://前缀很重要。如果省略了它，浏览器就会在 Web 站点内寻找名为 p2p.wrox.com 的文件夹。

绝对 URL 没有歧义。它们总是指向固定的位置，有助于确保总是引用完全相同的资源，而不管该资源文档位于何处。这样可能会使您认为在任何地方使用绝对 URL 都是理想的，包括引用自己的站点中的资源。然而，事实并非如此。额外的协议与域信息会增加浏览器中页面的大小，使页面的下载不必要地变慢。更重要的是，它增加了修改域名或者在不同的 Web 站点中重用某些功能的难度。例如，如果以前在 www.mydomain.com 上运行了站点，但是现在要转移到 www.someotherdomain.com，

就需要更新整个 Web 站点中的所有绝对 URL。

在开发过程中使用绝对 URL 也会有麻烦。最常发生的事情是在像 `http://localhost` 这样一个 URL 上测试 Web 站点。如果打算让所有图像都指向那个 URL，那么一旦在 `www.PlanetWrox.com` 这样的生产域上输入站点，它们都会被中断。

简言之，谨慎使用绝对 URL。当引用 Web 站点之外的资源时总是需要它们，不过在可能的情况下应首先选择自己的项目中的相对 URL。

7.1.2 默认文档

在 URL 的上下文中还应知道关于默认文档的知识。当浏览像 `www.domainname.com` 这样的站点时，会惊奇地发现出现了一个页面。这是怎么回事呢？每个 Web 服务器都有所谓的默认文档，即当没有提供显式的文档名时浏览器可以请求的一系列文档名。因此，当浏览 `www.domainname.com` 站点时，Web 服务器会扫描请求的目录(在本例中是根文件夹)并处理它在默认文档列表中找到的第一个文件。在大多数 ASP.NET 的情况下，Web 服务器被配置为使用 `Default.aspx` 作为默认文档。因此，当浏览 ASP.NET Web 服务器上的 `www.domainname.com` 时，实际上打开了页面 `www.domainname.com/Default.aspx`。

在创建的链接中，当不需要 `Default.aspx` 时通常应省略它。它会减小页面的大小，但是更重要的是，它使用户输入地址更容易。

现在已经介绍了如何使用 URL 指向文档和其他文件，下面将介绍使用这些 URL 的更高级的控件：ASP.NET 导航控件。

7.2 使用导航控件

ASP.NET 4 提供了 3 个有用的导航工具：SiteMapPath、TreeView 和 Menu。图 7-3 显示了这 3 个导航控件的基本示例，它们没有应用任何样式。

左边的 SiteMapPath 向用户显示了当前页面的路径。如果用户要在站点层次结构中向上走一层或多层，这样会有很大帮助。它还能帮助用户了解他们在何处。TreeView 能显示站点的结构，并允许展开和折叠不同的节点；在图 7-3 中整个树都是展开的。右边的 Menu 控件最初只显示 Home 菜单项。然而，一旦将鼠标移动到菜单项上面，就会出现一个子菜单。在图 7-3 中，这些子元素之一是 Reviews 项，它本身还有子节点。

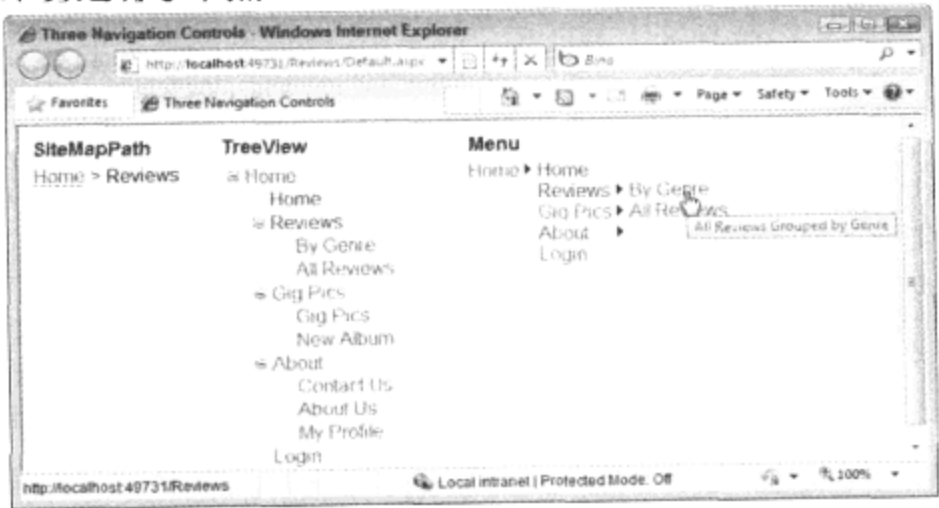


图 7-3

尽管这 3 个导航控件的行为和外观有很大不同，但他们的部分设计是相同的。

7.2.1 导航控件的体系结构

为了使得更容易使用 Menu、TreeView 或 SiteMapPath 显示站点中的相关页面，ASP.NET 使用了一个基于 XML 的文件来描述 Web 站点的逻辑结构。默认情况下，这个文件名为 Web.sitemap。然后站点中的导航控件会用这个文件以有组织的方式表现相关的链接。只要将一个导航控件与这个 Web.sitemap 文件挂钩，就能创建复杂的用户界面元素，如折叠菜单或树型视图。

7.2.2 分析 Web.sitemap 文件

默认情况下，应将站点地图文件命名为 Web.sitemap。这样控件就可以自动找到正确的文件。对于更高级的情况，可以有多个不同名称的站点地图文件，且在向系统提供这些附加文件的 web.config 中有一个配置设置。在大多数情况下，有一个站点地图文件就足够了。站点地图文件的基础版本如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode url="/" title="Home" description="Go to the homepage">
    <siteMapNode url="/Reviews" title="Reviews"
      description="Reviews published on this site" />
    <siteMapNode url="/About" title="About"
      description="About this site" />
  </siteMapNode>
</siteMap>
```

这个站点地图文件中包含一起形成站点逻辑结构的 siteMapNode。在这个示例中，只有一个名为 Home 的根节点，它包含两个子元素：Reviews 和 About。

Web.sitemap 文件中的关键元素

每个 siteMapNode 可以有多个子节点(但是，在 siteMap 元素下只能有一个 siteMapNode)，可以用来创建一个既有广度又有深度的站点结构。本例中的 siteMapNode 元素有 3 个特性集：url、title 和 description。url 特性应指向 Web 站点中的有效页面。可以用上一节介绍的~语法来引用基于应用程序根文件夹的 URL。虽然 ASP.NET 运行库不允许多次指定同一个 URL，但是可以通过添加一个查询字符串使 URL 唯一以绕过这一问题。例如，~/Login.aspx 和~/Login.aspx?type=Admin 会被看作两个不同的页面。本章后面将深入介绍查询字符串。

各种导航控件使用 title 特性显示页面的名称。在以后使用 Menu、TreeView 和 SiteMapPath 控件时将看到关于它的更多信息。description 特性用作导航元素的工具提示。图 7-3 显示了“By Genre”项的工具提示。

导航控件与 ASP.NET 安全机制一起工作。即可以自动隐藏控件(如 Menu)中用户不具有访问权限的元素。安全性将在第 16 章详细介绍。

为了能够使用 Web.sitemap 文件，ASP.NET 使用了 SiteMapDataSource 控件，它在 Toolbox 的 Data 类别下面。当使用 SiteMapPath 控件显示“痕迹导航”时，ASP.NET 会自动找到 Web.sitemap 文件。使用另外两个导航控件时，需要显式地指定一个 SiteMapDataSource 作为 Web.sitemap 文件的中间层。

要创建一个有用的 Web.sitemap 文件，需要向站点中添加一个文件，然后手动向它添加必需的

siteMapNode 元素。尽管有第三方解决方案，但是 VWD 没有自动基于当前站点的结构创建站点地图文件的方式。

试一试

创建 Web.sitemap 文件

本练习将向站点中添加一个新的 Web.sitemap 文件，并向它添加一些 siteMapNode 元素。这个站点地图将作为站点中其他导航控件的基础。如果之前已经设置了 Web 站点的 Virtual Path 属性，就可以跳过第一步。

(1) 再次在 VWD 中打开 Planet Wrox 项目，并在 Solution Explorer 中单击 Web 站点以选中它。按下 F4 键打开 Properties 面板，然后设置 Virtual Path 属性为正斜杠(/)，如图 7-2 所示。从现在起，假定总是用这个基于根文件夹的 URL 运行 Web 站点。

(2) 在 Solution Explorer 中右击 Web 站点，选择 Add New Item 命令，然后单击 Site Map。让默认名称设置为 Web.sitemap 并单击 Add 按钮。最后 Web.sitemap 文件中会出现一个包含两个子节点的根元素。

(3) 修改 Web.sitemap 文件使它包含下列代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode url="/" title="Home" description="Home">
    <siteMapNode url="/Default.aspx" title="Home"
      description="Go to the homepage" />
    <siteMapNode url="/Reviews/Default.aspx" title="Reviews"
      description="Reviews published on this site">
      <siteMapNode url="/Reviews/AllByGenre.aspx" title="By Genre"
        description="All Reviews Grouped by Genre" />
      <siteMapNode url="/Reviews/All.aspx" title="All Reviews"
        description="All Reviews" />
    </siteMapNode>
    <siteMapNode url="/About/Default.aspx" title="About"
      description="About this Site">
      <siteMapNode url="/About/Contact.aspx" title="Contact Us"
        description="Contact Us" />
      <siteMapNode url="/About/AboutUs.aspx" title="About Us"
        description="About Us" />
    </siteMapNode>
    <siteMapNode url="/Login.aspx" title="Login"
      description="Log in to this web site" />
  </siteMapNode>
</siteMap>
```

记住，不需要亲自输入所有这些代码。可以从 Wrox 网站或 www.tupwk.com 网站上下载本章代码文件，从中找到该文件的一个副本。

(4) 保存文件；现在就完成了。



常见错误：确保准确地按照这里显示的代码段输入。要注意一些项，如第一个 Home 元素，它包含其他的子元素，这些子元素的结束标记在代码段的深处。相反，像 By Genre 这样的元素使用的是自动结束标记，而且它们不包含任何子元素。

工作原理

虽然上面的“试一试”练习中没有在 Web.sitemap 文件中添加任何壮观的代码，但是有几件事值得讨论一下。首先，注意站点地图仅包含一个根节点，名为 Home。这是由 Web.sitemap 文件强制实施的，它不允许多于一个根元素。它的不利方面是这个根元素也会成为 Menu 和 TreeView 控件的根项目。在图 7-3 中可以看到 TreeView 的所有子菜单都落在了 Home 节点下面。然而在大多数 Web 站点中，更常见的是让 Home 项与其他项在同一层上。因此，在本“试一试”练习中直接在父节点下添加一个附加 Home 节点使它与 Reviews、About 和 Login 项在同一层上。在以后的练习中将看到如何隐藏控件中的根元素，从而仅显示根节点的第一个子节点及第一个节点的子节点。为了解决 siteMapNode 元素中的 URL 需要唯一这个问题，可以将一个设置为~/，另一个设置为~/Default.aspx。根据 Web 服务器处理默认文档的方式，它最终会指向同一个文件。

Web.sitemap 文件本身用处并不大。需要向站点中添加导航控件来使用站点地图。下一节将介绍如何使用 Menu 控件。之后的几节将深入介绍 TreeView 和 SiteMapPath 控件。

7.2.3 使用 Menu 控件

Menu 控件非常容易使用与调整。要创建一个基础菜单，只需向页面中添加一个 Menu 控件，并将它与一个 SiteMapDataSource 控件挂钩，就可以了。同时 Menu 控件也很灵活，有大概 80 个公有属性(包括所有控件都拥有的共同属性)，可以调整控件的每个视觉方面。表 7-1 列出了菜单用到的最常见的属性。参见 MSDN 在线帮助查看该控件的完整描述。

表 7-1

属 性	说 明
CssClass	允许设置一个应用到整个控件的 CSS 类特性
StaticEnableDefaultPopOutImage	确定是否使用图像指示顶级菜单项上的子菜单的 Boolean 值
DynamicEnableDefaultPopOutImage	确定是否用图像指示子菜单项上的子菜单的 Boolean 值
DisappearAfter	确定将鼠标从菜单项上移开后菜单项仍然可见的时间，以毫秒为单位
MaximumDynamicDisplayLevels	确定控件能显示的子菜单项的级数。有助于非常大的站点地图限制发送给浏览器的项数
DataSourceID	SiteMapDataSource 控件的 ID，为 Web.sitemap 文件中的菜单提供数据
Orientation	确定是使用带隐含子菜单的水平菜单还是使用带折叠子菜单的垂直菜单
RenderingMode	ASP.NET 4 中新增的属性。这个属性用于确定控件是使用表和内联样式，还是使用无序列表和 CSS 样式来显示自身
IncludeStyleBlock	ASP.NET 4 中新增的属性。这个属性使得您可以完全控制控件的样式。当设置为 False 时，ASP.NET 就不添加用于布局菜单的内嵌样式表块，而是由您负责编写 CSS

Menu 控件包含几个以 Static 或 Dynamic 开头的属性。Static 属性用来控制加载页面时出现的主菜单项。因为把鼠标悬停在它们上面时它们不会改变或隐藏，所以认为它们是静态的。子菜单是动态的，因为只有当激活相关主菜单项时它们才会出现。

除了这些属性外，菜单还有几个样式属性，允许改变菜单不同部分的外观。

1. 使用显示模式

早期的 Menu 控件版本因生成的 HTML 而备受批评。在 ASP.NET 2.0 和 3.5 中，Menu 控件使用表和内联样式生成庞大的 HTML 标记。这样除了会不必要地增大页面大小外，还意味着使用 CSS 样式化菜单时将更加困难。幸运的是，在控件上引入 RenderingMode 属性后，这个问题在 ASP.NET 4 中得到了解决。默认情况下，在新的 ASP.NET 4 站点中，这个属性能够确保控件使用元素将自己显示为一个无序列表。可以通过将 RenderingMode 属性设置为 Table 来重写这个行为。即便为了向后与 ASP.NET 3.5 兼容，而将整个 Web 站点配置为以遗留模式显示它自己，也可以将这个属性设置为 List 以强制控件以无序列表形式显示它自己。这可以通过在 web.config 文件中将<pages>元素的 controlRenderingCompatibilityVersion 特性设置为 3.5 来完成：

```
<system.web>
  <pages controlRenderingCompatibilityVersion="3.5" />
```

然而，对于新的 ASP.NET 4 Web 站点来说，如 Planet Wrox 项目，不需要显式地设置这个值，控件就会使用现代的 CSS 以及基于无序列表的菜单。在下一个“试一试”练习中，将看到这个 Menu 控件以及它生成的 HTML。

2. 创建菜单控件的一个基础版本

要查看 Menu 控件是如何操作的，最好首先创建一个基础版本。当了解了它的工作原理以及后台操作后，就可以将菜单样式化为喜欢的样子，这样它就会参与到站点设计的其余部分。

试一试

向站点中添加菜单

在本练习中，将看到如何在使用 Web.sitemap 文件的母版页中添加一个简单的 Menu 控件来构建菜单。Menu 被添加到母版页的 MenuWrapper 区域，并且会水平地表现菜单项。由于采用的是这种方向，因此 Menu 菜单仅适用于 Monochrome 主题。以后会添加一个 TreeView 来表示站点中的页面，并编写一些显示 Monochrome 主题的 Menu 和显示 DarkGrey 主题的 TreeView 的代码。

(1) 在 Markup 视图中打开母版页，并定位到名为 MenuWrapper 的<div>元素。删除占位符文本 Menu Goes Here。如果以前在 MainContent <div>中的 ContentPlaceHolder 标记之间添加了一些默认文本，则删除它们。

(2) 从 Toolbox 的 Navigation 类别中，拖动一个 Menu 放到 MenuWrapperdiv 标记之间。将 Menu 控件的 CssClasss 设置为 MainMenu:

```
<div id="MenuWrapper">
  <asp:Menu ID="Menu1" runat="server" CssClass="MainMenu"></asp:Menu>
</div>
```


(3) 切换到 Design 视图。可能会注意到 Design 视图看起来不再像最终页面。那是因为可能从 web.config 中的<pages>元素中删除了 styleSheetTheme 特性。目前可以让它保持这样。当进行了很多样式化操作后，这就不那么重要了。仍然可以看到 cpMainContent 占位符内的内容将最终出现在浏览器中。

(4) 单击 Menu 控件的灰色箭头打开 Smart Tasks 面板。

(5) 从 Choose Data Source 下拉列表中选择<New data source>。在出现的对话框中单击 Site Map 图标。图 7-4 显示了此时的屏幕。

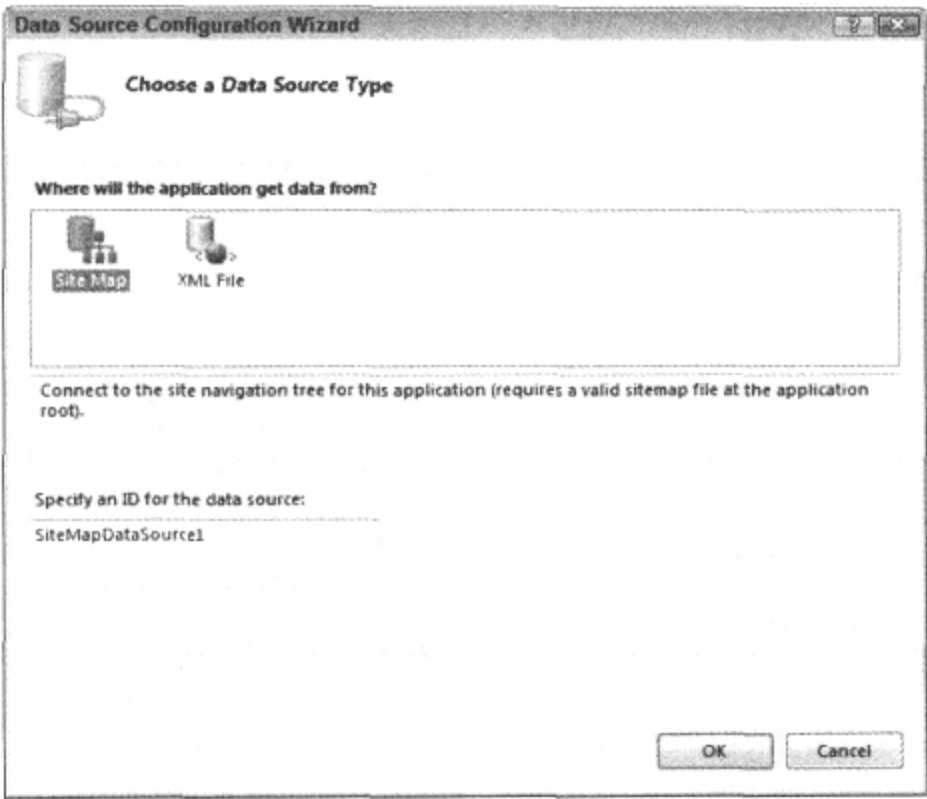


图 7-4

(6) 单击 OK 按钮关闭对话框。

(7) 当返回页面时，Menu 控件现在显示了顶级元素 Home(如图 7-5 所示)。

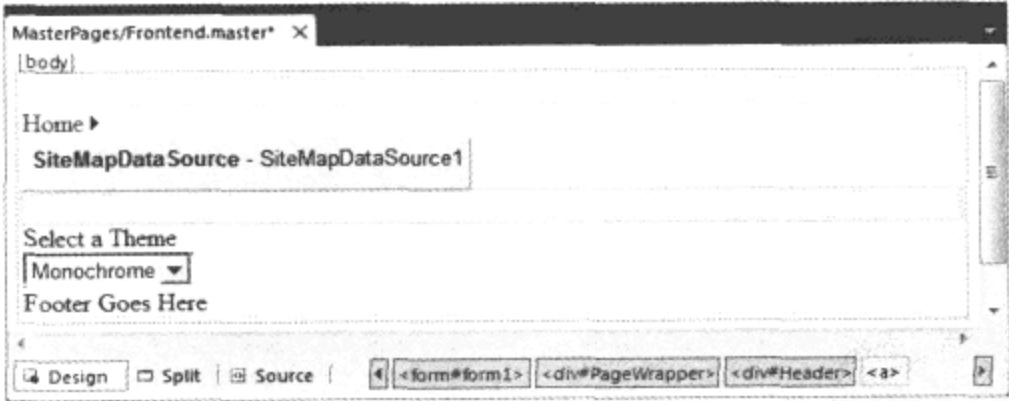


图 7-5

如果 Design 视图看起来不是这样，而是比较像最终页面，则打开 web.config 文件并从<pages>元素中去掉 styleSheetTheme 属性。

(8) 再单击一次 SiteMapDataSource，然后按下 F4 键打开或激活 Properties 面板。将 ShowStartingNode 属性的值由 True 改为 False。注意，一旦这样设置后，设计器中的 Menu 控件就会更新，并显示根元素下的所有直接子菜单：Home、Reviews、About 和 Login。图 7-6 显示了菜单现在的样子。

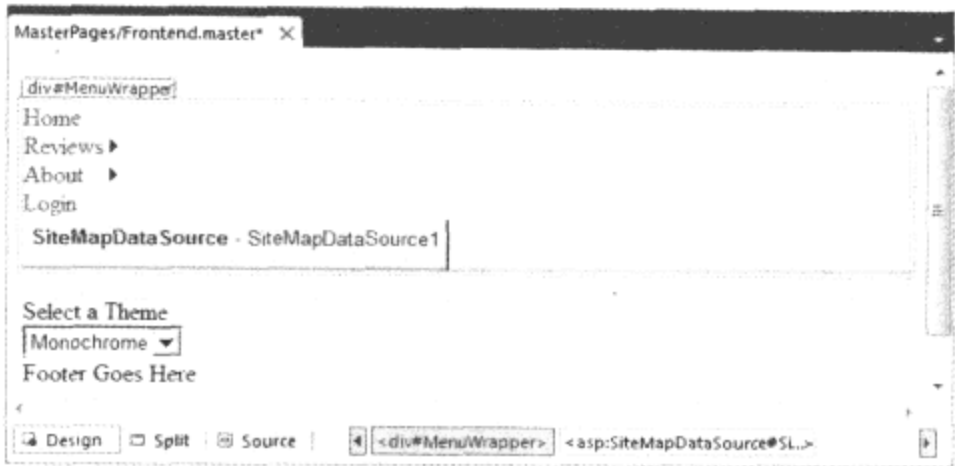


图 7-6

(9) 单击 Menu 控件以选中它，然后使用 Properties 面板对控件的属性作下列修改，如表 7-2 所示。Menu 控件有很多属性，如果将 Properties 面板中的属性列表以字母顺序排序的话，就会发现很容易就能找到它们。为此，可以单击工具栏上的第二个按钮(其上有一个字母 A、一个字母 Z 和一个箭头)。

表 7-2

属 性	值
StaticEnableDefaultPopOutImage	False
Orientation	Horizontal

当准备好这些后，Menu 的代码如下所示：

```
<asp:Menu ID="Menu1" runat="server" CssClass="MainMenu" Orientation="Horizontal"
    DataSourceID="SiteMapDataSource1" StaticEnableDefaultPopOutImage="False">
</asp:Menu>
```

(10) 保存对母版页所做的修改，然后在浏览器中请求 Default.aspx 页面。如有必要，使用 Theme 下拉列表使 Monochrome 成为活动主题。现在应在水平菜单区域中看到菜单。将鼠标悬停在菜单项上，就会看到出现的子菜单，如图 7-7 所示，这里显示的是谷歌 Chrome 浏览器中的页面。

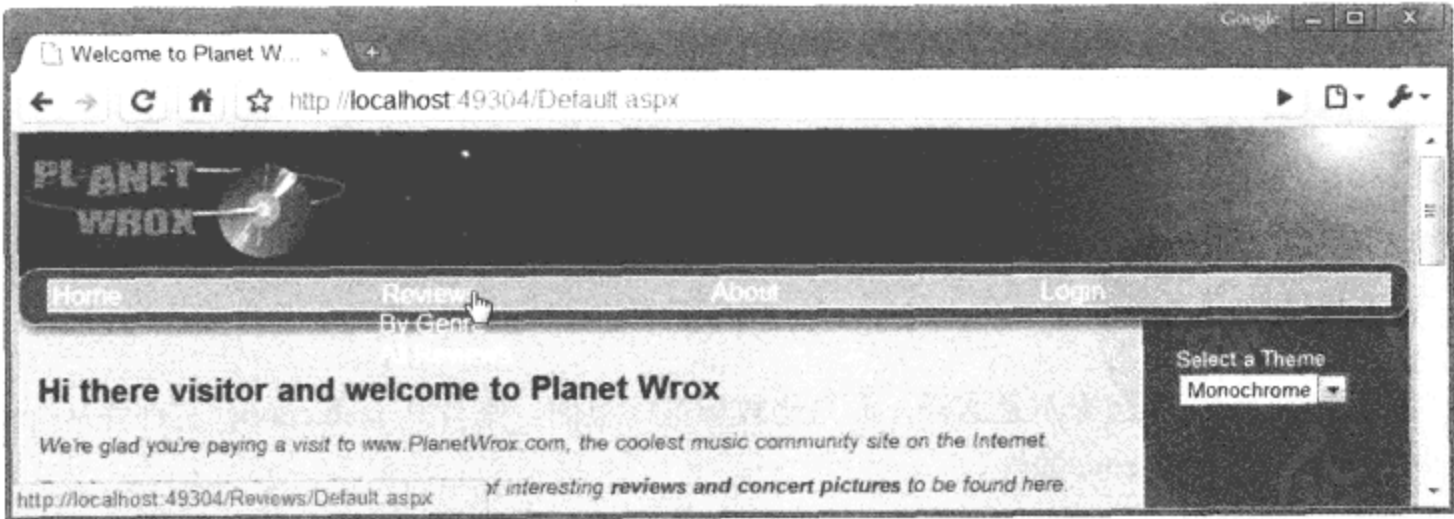


图 7-7

注意，很难看清子菜单项上的文本。这是因为 Monochrome 主题中的 CSS 将菜单区域中的所有锚(anchor)上的文本改成了白色，同时并未设置明确的背景色。了解了 Menu 控件如何工作后，就可

以修改它的样式化信息。

如果 DarkGrey 主题的菜单看起来不太美观，也不要担心。在本章后面会介绍如何实现那个主题的不同 Navigation 控件。

工作原理

当将带菜单的页面发送给浏览器时，Menu 控件会向在同一个母版页中定义的 SiteMapDataSource 请求数据。这个数据源控件会读取 Web.sitemap 文件，然后将 XML 文件传给 Menu 控件。基于层次结构的 XML，Menu 能生成必需的 HTML 和 JavaScript。它用嵌套的子元素为最上边的菜单项生成一个元素，每个元素包含一个或多个菜单项。Menu 控件刚初始化时通过一些 JavaScript 将子菜单隐藏起来。当把鼠标悬停在其中一个主菜单项上时，子菜单就变得可见，这也是通过 JavaScript 来实现的。

如果搜索页面的源代码，查找隐藏或显示菜单的 JavaScript 代码，那是找不到的。那么，在何处用 JavaScript 元素来显示和隐藏相关菜单项呢？答案是在页面的隐秘<script>标记中，如下所示：

```
<script src="/WebResource.axd?d=vxurWY7jjhneEhwnQbmdBEdPSXwLRytjgBhME9lyLool
&amp;t=633925206143355520" type="text/javascript">
```

这个<script>标记引用了一个特殊的 ASP.NET 处理程序，名为 WebResource.axd。查询字符串(URL 中位于查询标记后面的部分)中合适的随机字符通知 ASP.NET 运行库去取一个包含菜单功能的 JavaScript 文件。该文件不在磁盘上，而是由 WebResource.axd 处理程序基于查询字符串即时返回。可以查看这个文件，方法是通过复制 src 属性的值并粘贴到浏览器中 Web 站点的端口号后面(例如，http://localhost:50404)，从而在浏览器中请求它。可以安全地忽略该文件，因为要让菜单正确生效，不需要对它作任何修改。其他控件也会用到 WebResource.axd 语法，如 TreeView 用它来检索在 TreeView 中使用的图像。

除了 JavaScript，在页面顶部还会发现一个 CSS <style>块，它设置了菜单项的默认布局，删除了元素在默认情况下显示的项目列表，并且还删除了菜单中<a>元素的下划线。

为了更好地将 Menu 控件与 Monochrome 主题的现有设计集成在一起，可以用 CSS 来样式化它。

3. 样式化 Menu 控件

Menu 控件提供了许多复杂的样式属性，允许您修改项的外观，如主菜单和子菜单项。也可以定义当这些项在活动状态(被选中)时或将鼠标悬停在它们上方时的样子。每个样式属性都有若干视觉方面的子属性，如字体、颜色和字距等。图 7-8 显示了 StaticMenuItemStyle 的 Properties 面板，它定义了页面首次加载时可见的主菜单项的外观。

大多数属性(如 BackColor、ForeColor 和 Font)都被添加到位于页面顶部的<style>块中，这个页面中包含该 Menu 控件。这样就使它难以在其他页面的设计中或者在其他主题中重用，因此使用 CSS 要好得多。下面的“试一试”练习将给出其工作过程。

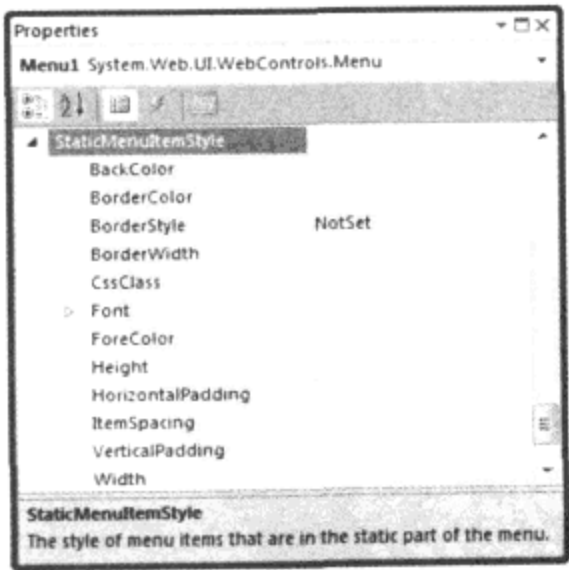


图 7-8

试一试

样式化菜单控件

在本练习中，为 Monochrome.css 文件添加一些 CSS 规则来影响样式化 Menu 控件的方式。在默认情况下，Menu 控件会向菜单项中添加一些 CSS 类(如 level1 和 level2)，这样便很容易在菜单中的各个级别上应用样式。

(1) 从 Monochrome 主题文件夹中打开 Monochrome.css 文件，并添加下面的 CSS 规则。可以省略/*和*/之间的注释，因为它们只是用来描述选择器的作用的。如果不想输入所有这些 CSS 代码，记着，也可以从本书的代码下载文件中得到一个副本，把它从那个文件中复制到您的文件中。对于本章，文件 Monochrome.css 位于 Monochrome 主题文件夹中。注意，CSS 是区分大小写的，因此要准确地输入下面所示的代码：

```
ul.level1
{
    /* Defines the appearance of main menu items. */
    font-size: 14px;
    font-weight: bold;
    height: 19px;
    line-height: 19px;
}

ul.level1 .selected
{
    /* Defines the appearance of active menu items */
    background-color: #BCD1FE;
}

a.level1
{
    /* Adds some white space to the left of the main menu item text */
    margin-left: 5px;
}

a.level2
{
    /* Defines the appearance of the sub menu items */
    background-color: #cccccc;
    padding-left: 8px;
}

a.level1:hover, a.level2:hover
{
    /* Defines the hover style for the main and sub items */
    background-color: #BCD1FE;
}
```

(2) 保存并关闭文件。

(3) 下一步，创建表 7-3 中所示的文件夹和 Web 窗体，在本例和以后的章节中将会用到它们。使用 MyBasePage 模板创建这些新文件。另外，在 Markup 视图中，赋予每个页面一个有意义的 Title 以避免以后出错。

表 7-3

文 件 夹	文 件 名	标 题
/Reviews	Default.aspx	My Favorite Reviews
/Reviews	All.aspx	All Reviews
/Reviews	AllByGenre.aspx	Reviews Grouped by Genre
/About	Default.aspx	About this Site
/About	Contact.aspx	Contact Us
/About	Aboutus.aspx	About Us

(4) 保存所有修改并在浏览器中打开根文件夹中的 Default.aspx 页面。站点菜单现在看起来完善多了，也更符合 Monochrome 主题的其余部分。当将鼠标悬停在主菜单上时，出现的子菜单会在浅灰色背景上显示文本。当将鼠标悬停在子菜单上时，它的背景色会再次发生变化。图 7-9 显示了在 Opera 浏览器中将 hover 样式应用到 By Genre 菜单项上后展开 Reviews 菜单的效果。

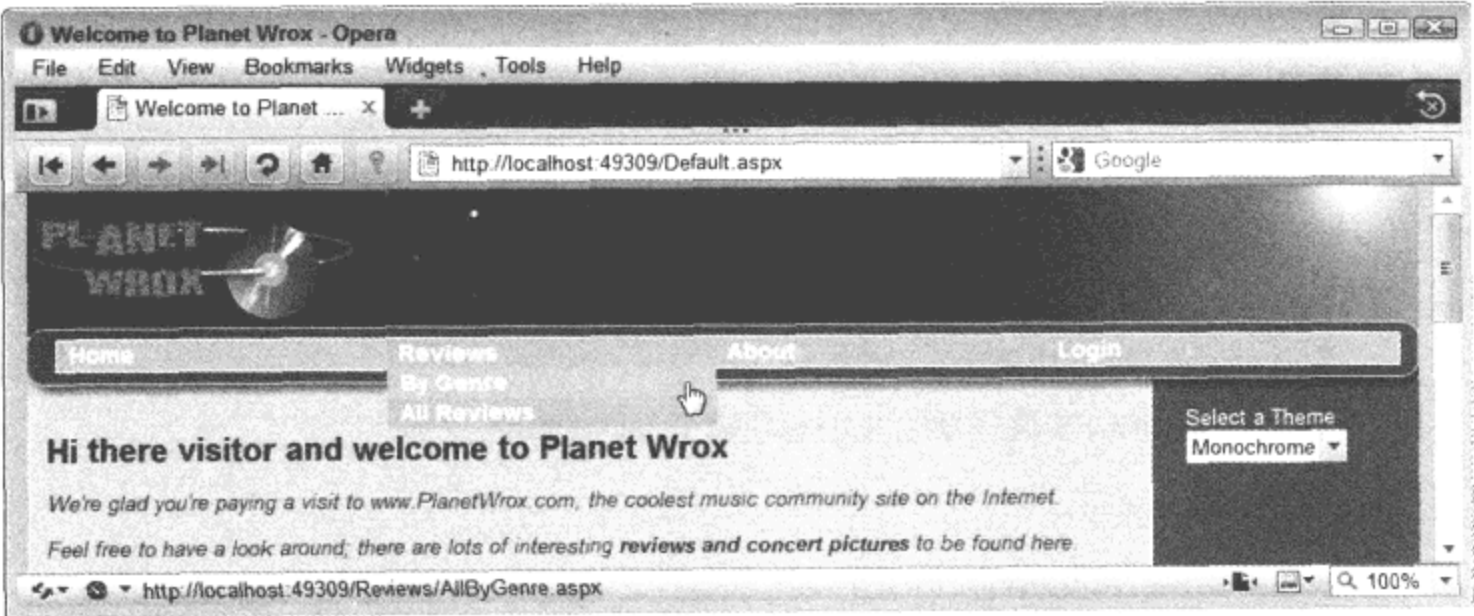


图 7-9



常见错误：如果当导航到创建的新页面之一时出现了错误，请确保都赋予了它们有效的标题。由于它们都继承自基页，因此加载页面时会检查标题。

工作原理

Menu 控件将自己显示为一系列和元素。菜单项本身是一个简单的带有 class 特性的<a>元素，表示它们位于哪一层。如果在浏览器中查看页面的 HTML，会看到如下代码：

```
<ul class="level1">
  <li><a title="Go to the homepage" class="level1 selected"
    href="/Site/Default.aspx">Home</a></li>
  <li><a title="Reviews published on this site" class="level1"
    href="/Site/Reviews/">Reviews</a>
```

```
<ul class="level2">
  <li><a title="All Reviews Grouped by Genre" class="level2"
    href="/Site/Reviews/AllByGenre.aspx">By Genre</a></li>
  <li><a title="All Reviews" class="level2"
    href="/Site/Reviews/All.aspx">All Reviews</a></li>
</ul>
</li>
... <!-- Other menu items go here -->
</ul>
```

由于这段代码是应用了一些 class 特性的纯 HTML 标记，所以可以很容易使用在前面几章中学到的 CSS 技术来样式化这些信息。在第一步中添加的代码使用了许多选择器来样式化菜单中的各个元素。例如，样式化主菜单项的代码如下：

```
ul.level1
{
  font-size: 14px;
  font-weight: bold;
  height: 19px;
  line-height: 19px;
}
```

这段代码被应用到了 level1 中所有带 CSS 类的元素上，也就是说，它被应用到了所有的主菜单项上，如 Home、Reviews 和 About。看一下菜单的 HTML 中的第一个<a>元素，它表示选中的 Home 项。注意它是如何应用到名为 selected 的第二个 class 上的：

```
<a title="Go to the homepage" class="level1 selected"
  href="/Site/Default.aspx">Home</a>
```

之后，通过使用这个 CSS 选择器给选中的项赋予不同的颜色：

```
ul.level1 .selected
{
  background-color: #BCD1FE;
}
```

这个规则同样适用于其他选择器，包括伪类：hover 选择器，当把鼠标悬停到它们上面时，它被应用到<a>元素上。

```
a.level1:hover, a.level2:hover
{
  background-color: #BCD1FE;
}
```

水平模式的 Menu 控件对于 Monochrome 主题是理想的，因为它的功能是水平导航条。对于 DarkGrey 主题，可以用相同的 Menu，并将它的 Orientation 设置为 Vertical。这样会创建一个竖直菜单，其中主菜单项堆叠在另一个菜单项上面，而子菜单会折叠在主菜单的右边。但是如果不用 Menu 控件的话，也可以用一个 TreeView 控件来显示站点地图的层次结构。下一节将讨论这个控件。

7.2.4 使用 TreeView 控件

TreeView 控件能显示项的层次结构列表，与 Windows 资源管理器中的树类似。项可以在包含子

元素的项前面用小加号和减号来展开和折叠。同样地，作为导航站点的一种方式，它是显示 Web 站点的站点地图的理想工具。然而，TreeView 控件使用的数据并不局限于 Web.sitemap 文件。也可以将它绑定到常规 XML 文件上，甚至可以通过编程来创建一个 TreeView 控件或它的项(节点)。

表 7-4 列出了 TreeView 最常用的属性。同样，可以从 MSDN 的在线帮助上得到所有可用属性及它们的说明的详细列表。

表 7-4

属 性	说 明
CssClass	允许设置应用到整个控件的 CSS 类特性
CollapseImageUrl	当单击时折叠起部分树的图像。默认是在上面用一个减号图标
ExpandImageUrl	当单击时展开部分树的图像。默认是在上面用一个加号图标
CollapseImageToolTip	当用户将鼠标悬停在可折叠菜单项上时显示的工具提示
ExpandImageToolTip	当用户将鼠标悬停在可展开菜单项上时显示的工具提示
ShowExpandCollapse	确定 TreeView 中的项是否可以通过单击它们前面的图像来折叠或展开
ShowLines	确定是否用连线来连接树中的各个项
ExpandDepth	确定页面首次加载时展开树中哪一层的项。默认设置是 FullyExpand，即树中的所有项都可见。可用的其他设置是用数值表示要展开的层

TreeView 控件有许多样式属性，允许修改树的不同部分的外观。为了告诉 TreeView 显示哪些项，要将它绑定到 SiteMapDataSource 控件上。下面将演示这个控件。

试一试 用 TreeView 控件构建一个导航系统

在本练习中，向以前创建的 Menu 下方的 MenuWrapper<div>标记中添加一个 TreeView 控件。然后将 TreeView 绑定到与 Menu 相同的数据源。接下来，编写一些代码，根据活动主题显示 Menu 或 TreeView。

- (1) 在 Markup 视图中打开母版页并在 Menu 控件下方添加一个 TreeView 控件，方法是将它从 Toolbox 中拖到页面中。
- (2) 在该控件的起始和结束标记之间添加如下的<LevelStyles>元素：

```
<LevelStyles>
  <asp:TreeNodeStyle CssClass="FirstLevelMenuItems" />
</LevelStyles>
```

在这段代码中，FirstLevelMenuItems 类选择器是在 DarkGrey.css 中定义的，用于在第一层为每个树项创建一些空间。

- (3) 切换到 Design 视图，单击 TreeView 一次，并单击小灰箭头打开 Smart Tasks 面板。从 Choose Data Source 的下拉列表中，选择 SiteMapDataSource1，这是为 Menu 控件创建的数据源控件(如图 7-10 所示)。



图 7-10

一旦选择了数据源，Design 视图中的 TreeView 就会更新；它现在会显示站点地图文件中的正确菜单项。

(4) 打开 TreeView 控件中的 Properties 面板，并将 ShowExpandCollapse 属性设置为 False。

(5) 在文档中的某处单击获得焦点，然后按下 F7 键打开母版页文件的 Code Behind，并定位到以前预先选择 Theme 列表中的主题时用过的 Page_Load 事件。在该代码下面、方法结束之前，添加下面的突出显示的代码，根据当前活动主题显示或隐藏 TreeView 和 Menu 控件。

VB.NET

```
ThemeList.Items.FindByValue(selectedTheme).Selected = True
End If
End If
Select Case Page.Theme.ToLower()
Case "darkgrey"
    Menu1.Visible = False
    TreeView1.Visible = True
Case Else
    Menu1.Visible = True
    TreeView1.Visible = False
End Select
End Sub
```

C#

```
ThemeList.Items.FindByValue(selectedTheme).Selected = true;
}
}
switch (Page.Theme.ToLower())
{
    case "darkgrey":
        Menu1.Visible = false;
        TreeView1.Visible = true;
        break;
    default:
        Menu1.Visible = true;
```

```
        TreeView1.Visible = false;
        break;
    }
}
```

(6) 保存所有修改并在浏览器中打开 Default.aspx 页面。根据当前活动主题，应看到 Menu 或 TreeView 控件。从列表选择一个不同的主题，页面就会重新加载，现在将另一个控件显示为 Web 站点的导航系统(如图 7-11 所示)。

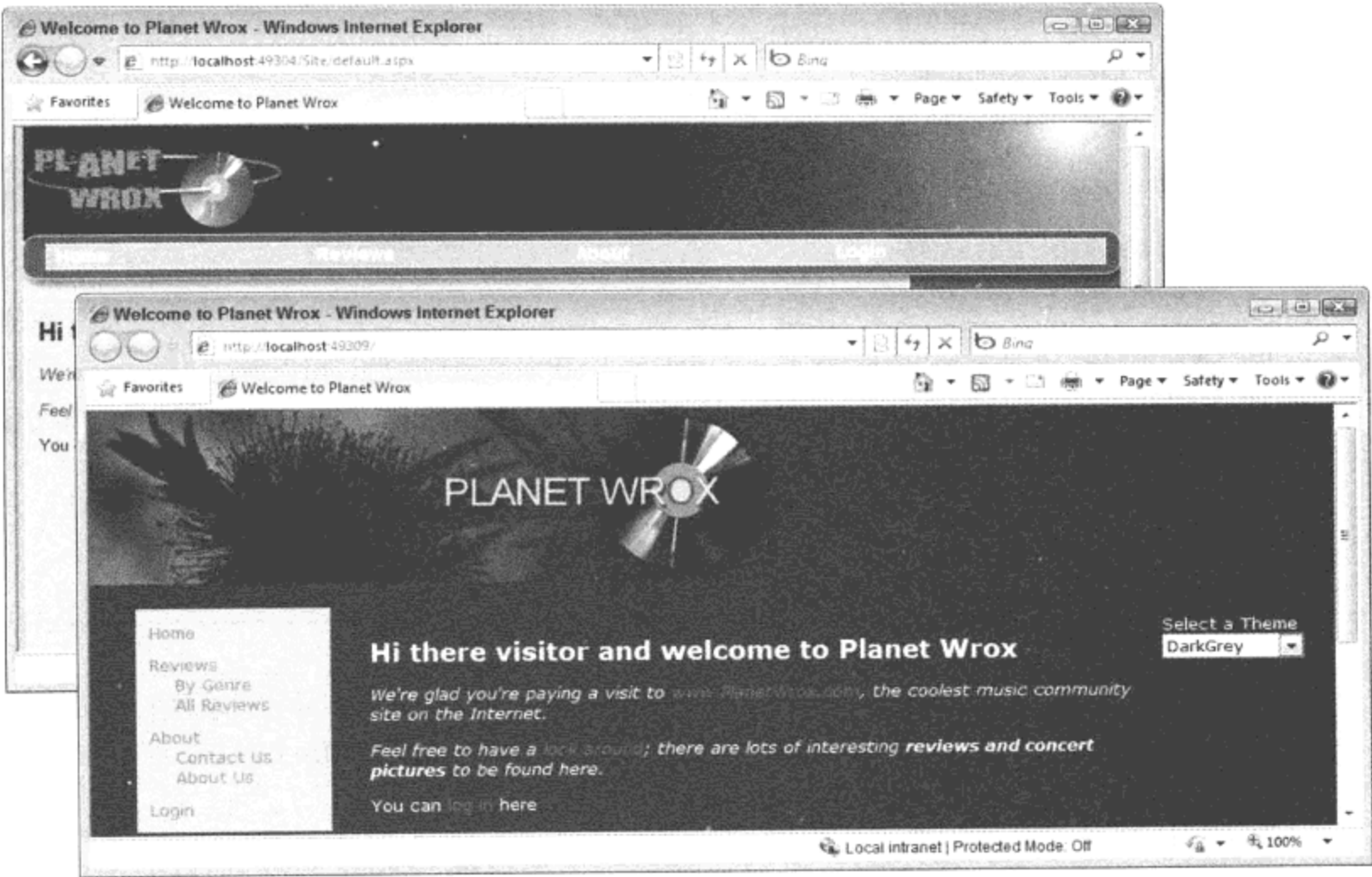



图 7-11



常见错误: 如果在运行 BasePage 类中的代码时出现一个错误,请确保为 web.config 文件中的<pages>元素设置了有效的主题。当从一个页面移动到另一个页面时,如果主题被切换了,请确保这个页面继承自 BasePage 类。如果使用基于定制模板的新页面时,就会出现这种情况。

工作原理

与 Menu 控件一样, TreeView 控件也能从 SiteMapDataSource 控件中获得它的数据, SiteMapDataSource 控件又从 Web.sitemap 文件中得到它的信息。在默认情况下, TreeView 显示加号和减号来表示项可以折叠与展开。对于站点菜单,这可能意义不大,因此可以将 ShowExpandCollapse 设置为 False 有效地隐藏要展开和折叠的图像。TreeView 允许设置许多样式属性,包括 NodeStyle、RootNodeStyle 和 LevelStyles, 通过使用它们能够影响树中各个项的外观。在本例中,是用 LevelStyles

来应用名为 FirstLevelMenuItems 的类, 这个类在第一层的项(如 Home 和 Reviews)的上方添加了一些空间。

母版页的 Code Behind 文件中的代码通过查看 Page 的 Theme 属性来了解当前主题。当 DarkGrey 是当前主题时, 代码就会隐藏 Menu, 然后显示 TreeView。在 Case Else / default 块中则正好相反。这意味着对于 Monochrome 主题和所有将来可能添加的主题来说, TreeView 会被隐藏起来, 并用 Menu 作为导航系统。

使用 TreeView 时遇到的问题与在早期 ASP.NET 版本中使用 Menu 时遇到的相同, 即产生大量的 HTML 标记。但是, 该控件没有 RenderingMode 属性, 因此, 如果使用 TreeView 控件, 将会在处理基于表的 HTML 时遇到麻烦。

到现在为止。我们已经讨论了 3 个导航控件中的两个, 要介绍的最后一个控件是 SiteMapPath 控件。

7.2.5 使用 SiteMapPath 控件

SiteMapPath 控件显示了您在站点结构中的位置。它将自身表现为一系列链接, 常称之为痕迹导航(breadcrumb)。它是一个非常简单但功能强大的控件, 有 50 多个公有属性, 可以通过 Properties 面板来设置这些属性以影响它们的显示方式。与 Menu 和 TreeView 控件一样, 它也有很多样式属性, 用来改变元素(如当前节点、正常节点和路径分隔符)的外观。

表 7-5 列出了 SiteMapPath 控件的几个最常用的属性。

表 7-5

属 性	说 明
PathDirection	支持两个值: RootToCurrent 和 CurrentToRoot。第一个设置显示左边的根元素、正中的中间层, 以及路径右边的当前页面。CurrentToRoot 设置则完全相反, 当前页面显示在痕迹导航路径的左边
PathSeparator	定义路径的不同元素之间要显示的符号或文本。默认是大于号(>), 不过可以将它改为别的符号, 比如竖线()
RenderCurrentNodeAsLink	确定将路径的最后一个元素(当前页面)呈现为文本链接还是纯文本。默认为 False, 这通常没有问题, 因为已经在元素表示的页面上, 所以并不真正需要链接
ShowToolTips	确定当用户将鼠标悬停在路径中的元素上时控件是否显示工具提示(从 Web.sitemap 文件中的 siteMapNode 元素的描述特性中检索)

根据个人的喜好, 通常不需要定义 SiteMapPath 控件的任何样式。在浏览器中的最终页面里, SiteMapPath 主要由锚标记(<a>)和纯文本组成。如果已经为 CSS 文件中的锚建立了一个特定的选择器, SiteMapPath 就会自动将自身显示得与页面中的其他链接一致。

试一试

用 SiteMapPath 控件创建一个痕迹导航

放置 SiteMapPath 控件的一个不错的位置是站点的全局母版页。这样它就会自动在所有页面中变得可见。

(1) 在 Markup 视图中打开母版页，并定位到 MainContent 元素的起始标记。在这个标记后面以及<asp:ContentPlaceHolder>标记的前面从 Toolbox 中拖一个 SiteMapPath 控件。在 SiteMapPath 后面添加两个换行符(
)。最后代码应如下所示：

```
<div id="MainContent">
    <asp:SiteMapPath ID="SiteMapPath1" runat="server"></asp:SiteMapPath><br /><br />
    <asp:ContentPlaceHolder ID="cpMainContent" runat="server">
```

(2) 保存修改然后在浏览器中请求 Default.aspx 页面。注意，页面现在显示了从站点根文件夹(通过 Home 链接标识)到当前页面的路径。单击 Menu 或 TreeView 控件中的几个项来导航站点，这时将看到各个页面的“痕迹导航”发生了变化。图 7-12 显示了在 Internet Explorer 中 All Reviews 页面的“痕迹导航”。All Reviews 页面是 Reviews 的一个子页面。Reviews 都落在 Home 根元素下面。
当导航到这些子页面之一时，可以单击路径的元素以向上走一层或多层。单击图 7-12 中显示的页面中的 Reviews 会将您带回到主 Reviews 页面，而单击 Home 则会把您带回站点的根文件夹。

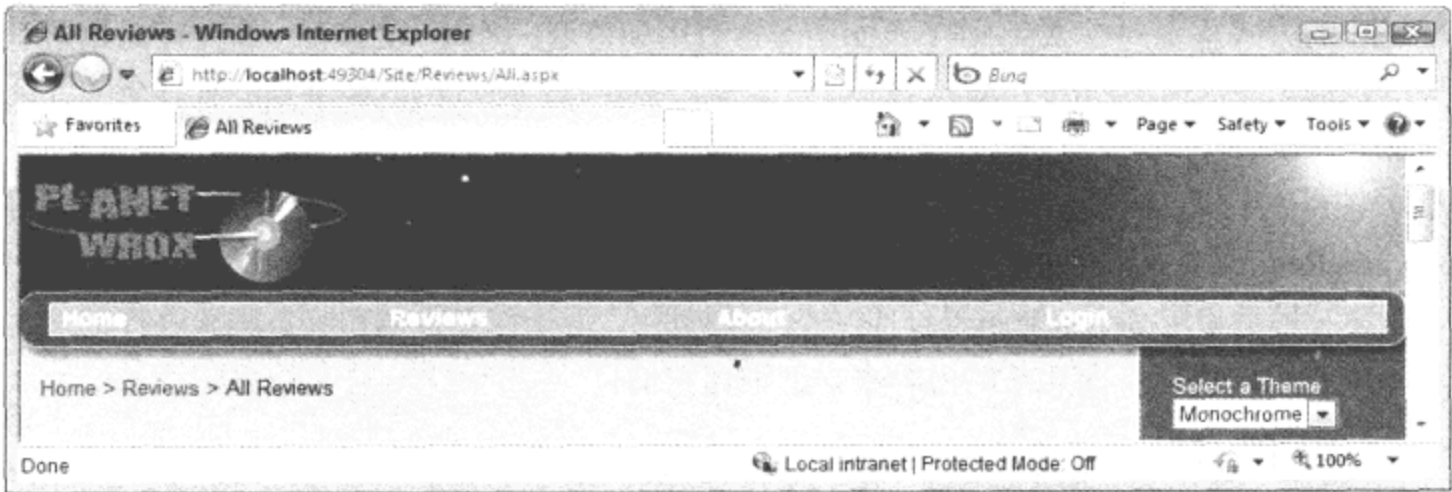


图 7-12

(3) 用 Theme 选择器切换到其他主题。注意，除了链接的颜色是在各个主题的 CSS 文件中定义的外，SiteMapPath 看起来非常相似。

工作原理

SiteMapPath 呈现为一系列包含一个链接或纯文本的元素。下面是图 7-12 中 SiteMapPath 的部分 HTML 代码：

```
<span><a title="Home" href="/">Home</a></span>
<span> &gt; </span>
<span><a title="Reviews published on this site"
    href="/Reviews/Default.aspx">Reviews</a></span>
<span> &gt; </span>
<span>All Reviews</span>
```

前两个元素(Home 和 Reviews)用一个链接(<a>)表示，允许导航到在它们的 href 属性中定义的页面。最后一个元素 All Reviews 只是纯文本。在这些元素之间将看到一个元素，其中包含可以在 PathSeparator 属性中设置的字符。

由于这个分隔符(>)在 HTML 中有一个特定的含义(大于)，它的值被编码为>以确保在浏览器中以纯文本字符出现。

如果在浏览器中查看页面的 HTML，也能看到一个允许跳过链接的<a>元素。<a>元素包含一个小图像，其 width 和 height 属性设置为了 0px，因此它不可见。它对于使用屏幕阅读器的那些有视觉障碍的人非常有帮助，因为它允许用户跳过导航直接阅读页面的内容。TreeView 和 Menu 控件使用了相同的方法避免使用屏幕阅读器在每次加载页面时都要大声读出整个站点结构。

这 3 个导航控件为从客户端浏览站点的导航系统提供了优秀的功能集。Menu 和 TreeView 控件都允许快速地显示站点的整个结构，以使用户能轻松地找到浏览路径。SiteMapPath 控件可帮助用户了解他们在站点中的位置，并给他们一种便利的方式来将页面导航到站点层次结构中的较高层次。

除了从客户端浏览器中导航外，用代码从服务器端将用户导航到不同的页面也非常普遍。其工作原理将在下一节讨论。

7.3 以编程的方式重定向

以编程的方式重定向在 ASP.NET 页面中非常有用和普遍。例如，设想有一个允许向数据库中输入评论的页面。一旦他们单击了 Save 按钮，评论就会被保存，用户也会被带到能看到全部评论的另一个页面。

ASP.NET 支持 3 种通过编写程序将用户重定向到新页面的主要方式：前两种是使用 Response.Redirect 和 Response.RedirectPermanent(ASP.NET 4 中新增的选项)，可以将指令发送到浏览器以取得一个新的页面。而第三种是使用 Server.Transfer，它在客户端执行。由于在客户端和服务端进行重定向的行为相当不同，因此将用下面几节的内容来详细描述它们。

7.3.1 通过编程将客户重定向到不同页面

在每个 ASPX 页面内，用户都具有对 Response 属性的访问权限，这个属性在前面介绍保存选中主题的 cookie 时已介绍过。Response 对象提供了对有用属性和方法(都与从服务器到用户浏览器的响应相关)的访问权限。这些方法中的两个是 Redirect 和 RedirectPermanent 方法。这两个方法会向浏览器发送请求新页面的指令。如果要用户重定向到站点中或者完全不同的 Web 站点中的另一个页面，这个方法非常有用。每个 Redirect 方法能以两种不同的方式使用：

```
Response.Redirect(newUrl)
Response.Redirect(newUrl, endResponse)
Response.RedirectPermanent(newUrl)
Response.RedirectPermanent(newUrl, endResponse)
```

Redirect 和 RedirectPermanent 的区别主要与搜索引擎优化有关。使用 Redirect 是告诉客户端：页面只是被临时移动了。通常使用这个属性基于某些动作将用户重定向到一个新页面上。例如，在填写完联系表单后，可能需要将用户导航到一个显示感谢消息的 ThankYou.aspx 页面上。

RedirectPermanent 用于告诉客户端：页面被永久移动了。当要告诉搜索引擎停止寻找一个旧的页面并索引一个新页面时，它将会很有用。例如，假设站点中有一个不再使用的页面：Index.aspx。搜索引擎可能会一直请求这个页面。如果将下面的代码添加到 Index.aspx 的 Code Behind 文件中，客户端(包括搜索引擎)就会被导航到 Default.aspx 页面。此时，搜索引擎会记录永久性重定向的节点，并会停止对 Index.aspx 页面的请求，定位到 Default.aspx 页面上。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Response.RedirectPermanent("Default.aspx")
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.RedirectPermanent("Default.aspx");
}
```

这些重定向方法的版本有一个名为 `endResponse` 的附加 `Boolean` 参数，当把 `endResponse` 参数以值 `False` 传递时，它允许在重定向动作完成后执行余下的所有代码。这通常并不是必需的，因此最好还是使用第一个版本，在默认情况下使用它来结束响应。

当要将用户导航到不同的页面时，常常想发送一些附加信息。这可以通过查询字符串完成。查询字符串是地址中位于页面名称后面的部分，由一个问号分隔。如下面的 URL：

```
http://localhost:49246/Demos/Target.aspx?CategoryId=10&From=Home
```

整个粗体字部分(在问号后面)就是查询字符串。它由名-值对组成，彼此用表示和的符号(&)分开。在本例中有两对：值为 10 的 `CategoryId` 和值为单词 `Home` 的 `From`。页面(本例中是 `Target.aspx`)能用 `Request.QueryString` 读取这些值。下面的“试一试”练习将说明如何使用查询字符串。

试一试

将用户重定向到另一个页面

为了详细了解重定向的工作过程，本练习显示了如何用 `Response.Redirect` 从一个页面重定向到另一个页面。在本例中，使用一个临时重定向(初始页面在重定向后仍然可以访问)，因此，代码使用的是 `Response.Redirect` 而不是 `Response.RedirectPermanent`。

(1) 在 `Demos` 文件夹中，创建两个基于定制模板的新 Web 窗体，分别命名为 `Source.aspx` 和 `Target.aspx`。将它们的 `Title` 分别设置为 `Source` 和 `Target`。

(2) 在 `Design` 视图中打开 `Source.aspx`，双击页面中 `ContentPlaceHolder` 之外灰色只读区域中的某处建立一个 `Page_Load` 处理程序。在这个处理程序内编写下面的代码将用户重定向到 `Target.aspx` 页面。为了显示如何通过查询字符串传递附加数据以及如何在目标页面中读取它们，代码将会传递一个名为 `Test` 的查询字符串字段，它的值是 `SomeValue`。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Response.Redirect("Target.aspx?Test=SomeValue")
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Redirect("Target.aspx?Test=SomeValue");
}
```


(3) 打开 Target.aspx, 切换到 Design 视图, 并将一个 Label 控件添加到 cpMainContent 占位符中。保持它的 ID 设置为 Label1。双击页面中灰色只读区域中的某处, 然后添加下列代码, 建立一个类似于上一步创建的 Page_Load 处理程序:

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Label1.Text = Request.QueryString.ToString()
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = Request.QueryString.ToString();
}
```

(4) 保存所有修改, 回到 Source.aspx 并按下 Ctrl+F5 组合键在浏览器中打开它。这时看到的不是 Source.aspx 页面, 而是图 7-13 所示的页面。

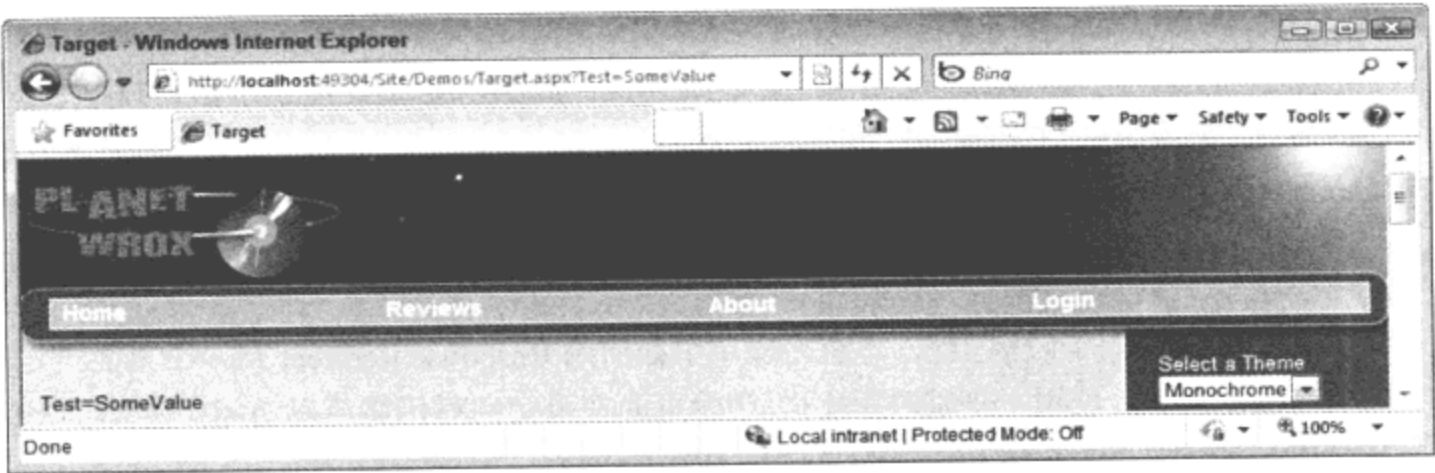


图 7-13

注意地址栏中现在是 Target.aspx?Test=SomeValue, 这是在源页面的 Page_Load 事件处理程序中重定向到的页面。目标页面中的 Label 会显示传递给这个页面的查询字符串。注意, QueryString.ToString()仅包含 Test=SomeValue。地址或问号都不是页面查询字符串的一部分。

工作原理

当使用 Response.Redirect 时, ASP.NET 会发送一个指令到浏览器, 让浏览器取得一个新页面。用技术术语来说, 它发送一个“302” HTTP 状态代码, 告诉浏览器当前页面已经移到了新位置。它也用这个指令发送新 URL, 使浏览器了解接下来去取哪个页面。在本练习中, 新页面是 Target.aspx?Test=SomeValue, 它包含页面名与查询字符串。然后浏览器请求 Target.aspx 页面, Page_Load 事件会被激活, 查询字符串也会显示在页面的标签上。由于这种客户端重定向, 新页面名称和查询字符串被完全提供给了客户端。

如果使用的是 Response.RedirectPermanent, 那么 ASP.NET 发送的将是“301 Moved Permanently”指令。对于常用的浏览器来说, 这样做在行为上并没有什么区别。浏览器之后仍能请求初始页面, 即使它已经被重定向。然而, 对于搜索引擎来说, 该 301 重定向则被解释为“不要再来请求这个页

面”，并且这个页面也不再被索引。

重定向对 URL 的命名模式与在服务器控件中使用的 URL 一样，因此可以重定向到~/Default.aspx 这样的页面来把用户重定向到 Web 站点根文件夹中的 Default.aspx 文件上。

与 Response.Redirect 和 Response.RedirectPermanent 相反，还有一个 Server.Transfer，它重定向到服务器上的另一个页面。

7.3.2 服务器端重定向

如果要发出不同的页面，而又不想修改客户端的地址栏的话，使用服务器端重定向就非常好。这样可以隐藏页面名称和查询字符串的细节，从而产生从用户的角度来看比较干净的 URL。这通常用在所谓的 URL 重写的情况中，用于创建干净的 URL。例如，用户可能请求一个这样的页面：

```
http://www.domain.com/Cars/Volvo/850/T5/
```

服务器在后台可能转换为：

```
http://www.domain.com/Cars/ShowCar.aspx?Make=843&Model=984&Type=7345
```

显然，第一个 URL 更容易理解和在浏览器中输入。它还能让用户猜出符合相同模式的其他 URL。例如，请求下面的页面时就会出现这个情况：

```
http://www.domain.com/Cars/Volvo/V70/R/
```

最后会出现显示 Volvo V70 R 的正确页面。

除了容易理解外，服务器端转换也可能稍稍加快站点的速度。不用向浏览器发送响应让它去取一个新页面，这样会产生一个对页面的新请求，可以直接将用户转到新页面上，以节省一些网络开销。

服务器端转换用 Server 对象实现。正如前面提到的 Request 和 Response 对象提供了关于请求和响应的信息，Server 对象也提供了关于运行页面时服务器的信息。可以用它得到关于服务器名、它的 IP 地址等信息。其中一个方法是执行服务器端转换的 Transfer。

Server.Transfer 只能用来重定向到站点内的其他页面。不能用它将用户转到不同域上的页面。如果试图这么做，ASP.NET 运行库会抛出一个错误。

为了说明 Response.Redirect 和 Server.Transfer 之间的区别，下面的“试一试”练习显示了如何修改 Source.aspx 页面以执行一个 Server.Transfer 操作。

试一试

服务器端重定向

修改重定向代码使它将用户转到另一个页面并不难。所需做的只是用 Server.Transfer 替换 Response.Redirect 即可，下面会演示如何做。

(1) 打开 Source.aspx 页面的 Code Behind 文件，并使用下面的代码行替换 Response.Redirect 所在的那行代码：

```
VB.NET
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Server.Transfer("Target.aspx?Test=SomeValue")
```


End Sub

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Server.Transfer("Target.aspx?Test=SomeValue");
}
```

(2) 保存修改，然后按下 Ctrl+F5 组合键在浏览器中打开 Source.aspx 页面(如图 7-14 所示)。

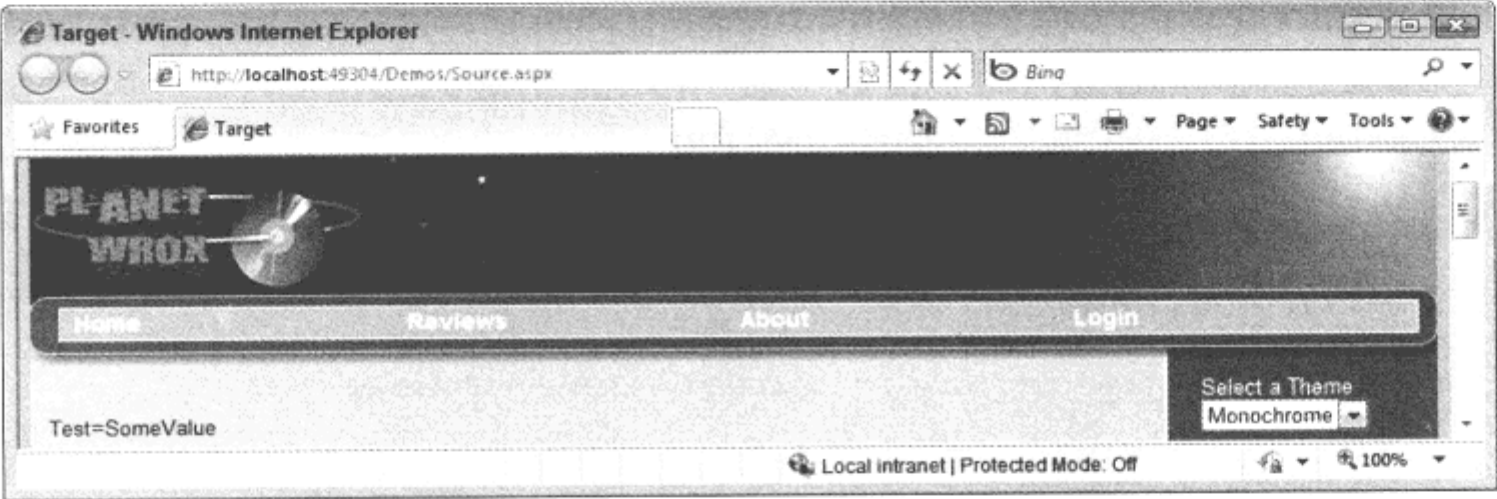


图 7-14

Label 控件显示了从 Source.aspx 页面发送到 Target.aspx 页面的查询字符串值，它演示了这一事实：实际上浏览的是 Target.aspx 页面的输出。然而，浏览器的地址栏并没有改变，仍然显示为 Source.aspx，对用户隐藏了新页面名称和查询字符串值。

工作原理

Server.Transfer 不是指示浏览器去取一个新页面，而是完全在服务器上发生。放弃旧页面的输出，并为要转换到的页面启动新的页面生命周期。然后这个页面生成它的内容，并发送回浏览器，而保持浏览器的地址栏不变。

如果看一下浏览器中的 HTML，将发现表单动作被发送到了新页面，因此会执行针对那个页面发生的任何回发，从而改变地址栏中的地址：

```
<form name="aspnetForm" id="aspnetForm" method="post"
      action="Target.aspx?Test=SomeValue">
...
</form>
```

以编程方式把用户导航到另一个页面，是本章关于导航的最后一部分内容。了解了本章介绍的概念后，就具备了在站点中创建高效导航系统的所有知识：从客户端的浏览器到自己的服务器端代码。

7.4 关于导航的实用提示

下面列出了关于导航的一些实用提示：

- 当开始构建一个知道将来会增长的 Web 站点时，创建一个逻辑结构。不要把所有文件都放在 Web 站点的根文件夹中，而要将相关的文件根据逻辑组合到同一个文件夹中。这样的逻辑组合可以使站点容易管理，用户容易找到他们要找的页面。虽然用 Web.sitemap 文件很容易将页面移到 Menu 或 TreeView 中，但是如果同时使用了编程方式的重定向或转换时就较难了，因为还需要更新服务器端代码以反映新的站点结构。为了创建稳固的页面结构，可以在开始构建站点之前把它画在纸上，或者使用站点地图图表工具，如 Microsoft Visio。
- 尽量限制显示在 Menu 或 TreeView 控件中的主项和子项的数目。如果供用户选择的选项列表过长，用户会迷路或者混淆。
- 当创建用来存储页面的文件夹时，给它们起简短且有逻辑的名称。使用 <http://www.PlanetWrox.com/Reviews> 导航到一个页面比导航到一个名称很长、还包括缩略词和数字的文件夹要直观得多。

7.5 本章小结

本章介绍了 ASP.NET Web 站点中的导航控件。用户不会直接输入 Web 页面的地址，因此重要的是给他们提供一个清楚而直观的导航系统。

创建优秀导航系统的关键基础是要很好地了解 URL 的工作原理。URL 有两种类型：相对 URL 和绝对 URL。相对 URL 用于指向站点中的资源。绝对 URL 允许根据资源的完整位置(包括协议和域信息)指向资源。如果要指向自己的 Web 站点之外的资源，则绝对 URL 是最有用的。

ASP.NET 提供了 Web 站点的用户界面中用到的 3 个导航控件。这些控件允许用户访问站点中的不同页面。Menu 控件显示了带折叠或隐含的子菜单的竖直或水平菜单。TreeView 控件可以用层次结构的方式显示站点的完整结构。SiteMapPath 控件显示了痕迹导航，给用户提供一个可视化线索，指出他们正位于站点中的何处。

除了内置的导航控件外，还可以通过编程方式将用户导航到另一个页面上。ASP.NET 提供了两种主要方法来完成这个工作：在客户端使用 Response.Redirect 和 Response.RedirectPermanent，在服务器端使用 Server.Transfer。重定向方法指示浏览器从服务器中取得一个新的页面，而转换方法就发生在服务器上。

第 8 章将介绍关于 ASP.NET 用户控件的更多知识，可以在 Web 站点的不同页面中重用特定代码与用户界面元素。

7.6 练习

1. TreeView 控件提供了大量样式属性，可以用来修改树中的项。如果要改变树中每个项的背景色，需要修改哪个属性呢？改变背景色的最好方法是什么？
2. 有哪些方法可以通过编程将用户重定向到另一个页面？它们之间的区别何在？
3. 使用 TreeView 控件有两种方式：一种方式是作为带项和子项的列表，单击它们时能折叠或展开；另一种方式是作为显示所有项的静态列表，不能折叠或展开。要禁止用户展开或折叠树中的项，需要设置控件上的什么属性呢？

练习的答案见附录 A。

本章要点回顾

Menu 控件	能够通过使用下拉或折叠的子菜单以水平或垂直方式显示数据的导航控件，包括来自 Web.sitemap 文件的数据
永久性重定向	用于告知客户端(如搜索引擎) 信息的一种机制，在页面被永久性移动时，告知客户端停止请求旧页面
服务器端转换	在服务器端进行的到另一个页面的重定向，不用告知客户端浏览器
SiteMapDataSource 控件	Web.sitemap 文件与导航控件(如 TreeView 和 Menu)之间的桥梁
SiteMapPath 控件	将站点根文件夹中的痕迹导航路径显示到当前页面上的导航控件，可以使用户回到站点的层次结构中
临时重定向	用于将用户重定向到一个临时的新位置的一种机制
TreeView 控件	能够以层次结构的方式显示数据的导航控件，包括来自 Web.sitemap 文件的数据
Web.sitemap	在站点中包含逻辑结构的基于 XML 的文件，该文件用于驱动其他的导航控件

第 8 章

用 户 控 件

本章要点

- 用户控件的概念、外观以及作用
- 如何创建用户控件
- 如何在页面中使用用户控件
- 如何通过向用户控件添加代码逻辑使用户控件更有用

除了第 6 章讨论的母版页、主题与外观外，ASP.NET 4 还有另一个功能，允许创建可重用因而一致的信息块：用户控件。

用户控件可以用来将逻辑上相关的内容和控件组合在一起，然后作为一个单位在内容页、母版页和其他用户控件内使用。用户控件实际上是一种微型 ASPX 页面，其中有一个标记部分，可能还有一个可以在其中编写控件代码的 Code Behind 文件。用户控件的使用与正常 ASPX 页面非常相似，只是有一些细微的区别。

在 ASP.NET 2.0 之前的版本中，用户控件常常用来创建必须出现在站点中每个页面上的可重用的功能块。例如，要创建一个菜单，就会创建一个用户控件，然后将那个控件添加到站点中的每个页面。由于 ASP.NET 对母版页的支持，因此这些情况下不再需要用户控件。这样可以使对站点结构的修改更容易。尽管母版页带来了一些优点，然而学习过本章后您会发现，在 ASP.NET Web 站点中仍然有利用用户控件的空间。

到本章末尾，您将对用户控件是什么及它们的工作方式有一个很好的了解，从而能够创建功能强大、可重用的内容块。

8.1 用户控件简介

用户控件对于封装需要在整个站点中重复使用的标记、控件和代码来说非常有用。在某种程度上，用户控件看起来有一些像服务器控件，它们都可以包含能够在页面中重用的编程逻辑和表现。然而，要做的不是从 VWD Toolbox 中拖动现有的控件，而是需要创建自己的用户控件，然后再将它

们添加到 ASPX 页面上，本章稍后将介绍这些。

母版页允许创建在站点所有页面中显示的内容，但通常需要将内容仅显示在部分页面上而不是所有页面上。例如，您可能想在几个受欢迎的页面上放一条横幅，而不想在主页或其他公共页面上放置。如果没有用户控件，就要为每个需要横幅(图像、链接等)的页面添加横幅代码。当要更新横幅时(例如想使用一个新的图像或链接)，就需要修改使用横幅的所有页面。如果将横幅移到一个用户控件中，并在内容页中使用该控件，那么只要修改此用户控件，使用这个用户控件的所有页面都会自动跟着修改。这样就有了一种创建可重用内容的灵活方式。

用户控件在以下几个方面与正常 ASPX 页面类似：

- 它们有一个可以添加标准标记和服务器的标记部分。
- 可以用 VWD 在 Markup、Design 和 Split 视图中创建和设计它们。
- 它们可以通过内联代码或 Code Behind 文件包含编程逻辑。
- 您具有对基于页面的信息(如 Request.QueryString)的访问权限。
- 它们引发 Page 类引发的事件的一部分(而不是全部)，包括 Init、Load 与 PreRender。

请注意它们也有一些区别。用户控件的扩展名为.ascx，而不是常规的.aspx。此外，不能直接在浏览器中请求用户控件。因此，不能链接到它们。在站点中使用用户控件的唯一方式是将其添加到内容页、母版页或另一个用户控件(已添加到一个页面中)中。

在本章的其余部分，将看到如何创建能显示横幅的用户控件。用户控件对于页面中不同大小的区域能表现为水平或竖直横幅。下一节将介绍如何创建用户控件。其后的小节将介绍如何在 ASPX 页面中使用控件。

8.1.1 创建用户控件

用户控件添加到站点中的方式与添加其他类型的内容相似：通过 Add New Item 对话框。与页面类似，选择编程语言，并确定是否将代码放在单独的 Code Behind 文件中。图 8-1 显示了用户控件的 Add New Item 对话框。

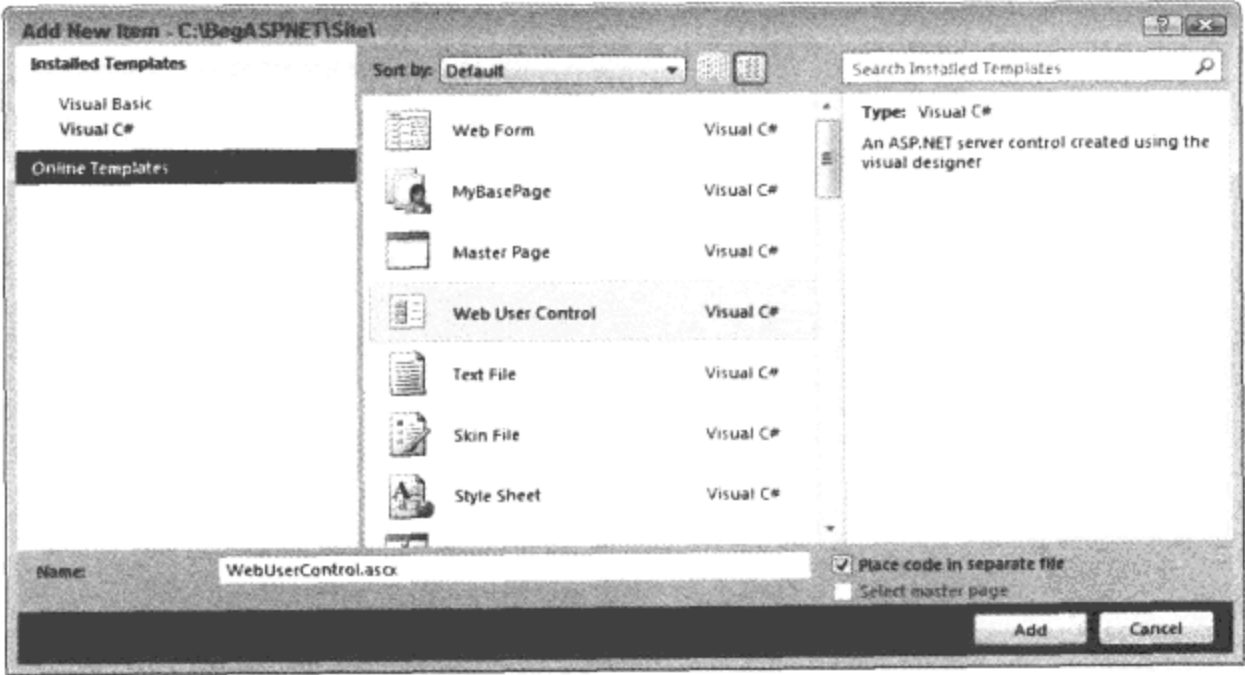


图 8-1

一旦向站点中添加了一个用户控件，它就会自动在文档窗口中打开。容易注意到的第一件事可能是用户控件没有@ Page 指令，而是有一个@ Control 指令，如下面使用 Code Behind 文件的例子所示：

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="WebUserControl.ascx.cs"
    Inherits="WebUserControl" %>
```

这样就将文件标记为用户控件，因此 ASP.NET 运行库知道如何处理它。除此之外，该指令等同于不使用母版页的标准 ASPX 页面。

有了在 VWD 文档窗口打开的用户控件后，就可以使用在前面 7 章创建页面时用到的所有工具。可以从 Toolbox 中向 Markup 和 Design 视图拖动控件，可以用 CSS 窗口改变用户控件的外观和内容，还可以用 Properties 面板修改用户控件中的控件属性。还可以编写响应控件引发的事件的代码。

为了亲自体验用户控件，下面的“试一试”练习显示了如何创建第一个用户控件。在后面的“试一试”练习中将看到如何在站点的 ASPX 页面中使用这个控件。

试一试

创建用户控件

在本练习中将创建一个基础用户控件，显示一个使用 Image 控件的竖直横幅。在本章后面的“试一试”练习中，将看到如何使用这个控件，以及如何添加另一个(水平)图像。然后将向控件添加一些智能功能，以便确定在运行时显示这两个图像中的哪个图像。

对于本练习，需要两个表示横幅的图像，一个是肖像模式，尺寸大约是 120×240 像素，另一个是风景模式，大小为 486×60 像素左右。本书提供的本章的代码下载文件夹 Resources 中有这两个图像，但是也可以自己创建它们。不要在意图像的精确大小，只要它们接近这些尺寸就可以了。

- (1) 在 VWD 中打开 Planet Wrox 站点。
- (2) 如果还没有创建文件夹 Controls，则在站点的根文件夹中新建一个名为 Controls 的文件夹。尽管用户控件可以放在站点层次结构的任何地方，但最好还是把它们放在一个单独的文件夹中，这样比较容易查找和管理。
- (3) 在站点的根文件夹中创建另外一个名为 Images 的文件夹。
- (4) 使用 Windows 资源管理器，打开本章的 Resources 文件夹(如果跟着本书前言的指示操作并下载了本章代码，则该文件夹就是 C:\BegASPNET\Resources\Chapter 08)；如果还没有下载本章代码，可以从 www.wrox.com 网站或 www.tupwk.com 网站上下载它)，然后从 Windows 资源管理器窗口中将文件 Banner120×240.gif 和 Banner486×60.gif 拖(或者复制和粘贴)到第(3)步创建的 Images 文件夹中。如果使用的是自己的图像，也把它们拖到 Images 文件夹中，并给它们起同样的名称。
- (5) 右击 Controls 文件夹并选择 Add New Item 命令。在随后出现的对话框中，单击 Web User Control。选择自己的编程语言，并确保选中了 Place Code in Separate File 选项，如图 8-1 所示。命名文件为 Banner，然后单击 Add 按钮将控件添加到站点中。注意，如果没有输入扩展名.ascx，则 VWD 会自动添加。VWD 会为所有通过 Add New Item 对话框添加的文件类型添加扩展名，因此不需要自己输入。Solution Explorer 现在应如图 8-2 所示。
- (6) 将用户控件切换到 Design 视图中，然后从 Toolbox 的 Standard 类别中拖动一个 Panel 到设计界面上。使用 Properties 面板将 Panel 的 ID 改为 VerticalPanel。
- (7) 从 Toolbox 中拖动一个 Image 控件到 Panel 中。单击选中 Image 控件，然后打开 Properties

面板。定位到 `ImageUrl` 属性并单击省略号按钮，如图 8-3 所示。



图 8-2

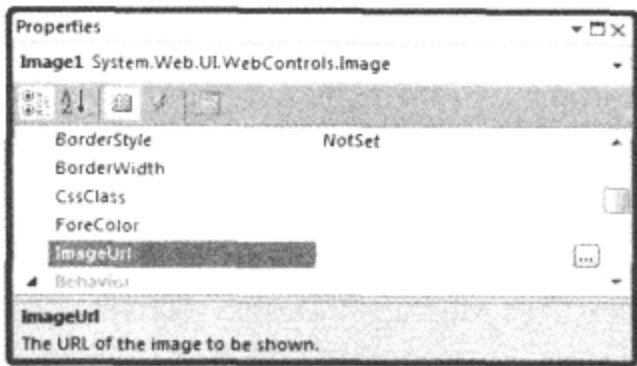


图 8-3

定位到 `Images` 文件夹，选择 `Banner120×240.gif` 图像，单击 `OK` 按钮将它添加到用户控件中。
`Design` 视图现在应如图 8-4 所示。



图 8-4

(8) 还是使用同一个 `Properties` 面板，定位到 `AlternateText` 属性并输入 `This is a sample banner`。有些浏览器(如 `Firefox`、`IE 8` 及其更高版本)只有当图像不能正确显示时才会显示备用文本(显示为客户端 `alt` 特性)。其他浏览器(包括 `IE` 的旧版本)则会在将鼠标悬停在图像上时显示备用文本作为工具提示。

(9) 切换到 `Markup` 视图，当将 `Panel` 控件拖到页面上时，如果它有默认添加的 `Height` 和 `Width` 特性，则把它们都删除。

(10) 将 `Image` 包括到一个标准 `<a />` 元素中，并将它的 `href` 特性设置为 `http://p2p.wrox.com`。可以使用代码段 `a` 插入一个空链接。为此，输入字母 `a` 然后按下 `Tab` 键，`VWD` 就会插入一个链接并允许直接输入 `href` 的值。当再次按下 `Tab` 键时，便会选中链接的内容，可以按 `Del` 键删除这个内容(在这里，链接的内容是 `Image` 控件)。最后，将结束标记 `` 剪切到图像的后边。

(11) 将锚标记(`<a>`)的 `target` 设置为 `_blank`，强制浏览器在单击图像时从新窗口中打开页面。完成以后，整个用户控件的代码应如下所示，除了 `Language` 特性可能设置为 `VB`，在 `VB.NET` 中默认将 `AutoEventWireup` 设置为 `False` 外：

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="Banner.ascx.cs"
    Inherits="Controls_Banner" %>
```



```
<asp:Panel ID="VerticalPanel" runat="server">
  <a href="http://p2p.wrox.com" target="_blank">
    <asp:Image ID="Image1" runat="server" AlternateText="This is a sample banner"
      ImageUrl="~/Images/Banner120x240.gif" />
  </a>
</asp:Panel>
```

(12) 按下 Ctrl+S 组合键保存修改，然后按下 Ctrl+F4 组合键关闭用户控件文件。

工作原理

VWD IDE 中用户控件的这个设计练习与关于页面的练习一样，可以使用拖放、Toolbox、Markup 视图、Split 视图和 Design 视图等。这样就使得用户控件的使用比较容易，因为可以使用所有这些页面开发时用到的熟悉工具。

刚刚创建的控件显示了一个包含在锚元素中的图像。下一节将介绍如何向母版页中添加用户控件，以便让它显示在站点每个页面的边栏<div>中。本章以后的小节中会介绍如何添加可用来在个别内容页中显示水平横幅的图像。

8.1.2 向内容页或母版页中添加用户控件

要能够在内容页、母版页中或者在另一个用户控件中使用一个用户控件，需要执行两个步骤。第一步，需要注册控件，方法是向希望出现用户控件的页面或控件中添加一个@ Register 指令。第二步是向页面添加用户控件的标记，并可以(可选地)在其上设置一些特性。

用户控件的典型@ Register 指令类似这样：

```
<%@ Register Src="ControlName.ascx" TagName="ControlName" TagPrefix="uc1" %>
```

这个指令包含 3 个重要特性，如表 8-1 所示。

表 8-1

特 性	说 明
Src	指向要使用的用户控件。为了在以后的阶段使页面的移动更容易，也可以用~语法指向应用程序根文件夹中的控件
TagName	用在页面的控件声明中的标记名。可以自由地命名这个名称，但通常都让它与控件的名称相同
TagPrefix	容纳用在页面的控件声明中的 TagName 的前缀。正如 ASP.NET 用 asp 前缀指代它的控件一样，它也需要为自己的用户控件提供一个前缀。在默认情况下，这个前缀是 uc，后面跟着一个序号，不过也可以将它改为您自己喜欢的其他内容——如，改为公司名称或者自定义的缩略词

从前边的“试一试”练习中创建的用户控件中可以看到，@ Register 指令如下所示：

```
<%@ Register Src="~/Controls/Banner.ascx" TagName="Banner" TagPrefix="uc1" %>
```

当注册控件时，可以用 TagPrefix:TagName 指令将它添加到页面中，类似于向页面中添加标准

服务器控件的方式。如果给横幅控件指定了@ Register 指令，则需要用下面的标记向页面中添加控件：

```
<uc1:Banner ID="Banner1" runat="server" />
```

这是页面中用户控件所需的最少的代码。注意控件是将 TagPrefix 和 TagName 结合起来定义的。另外两个属性 ID 和 runat 是 ASP.NET 页面中的大多数控件都拥有的标准属性。

幸运的是，大多数情况下，不必亲自输入所有这些代码。当在 Design 视图中从 Solution Explorer 中拖一个用户控件到页面中时，VWD 会自动添加必需的代码。下面的“试一试”练习演示了其工作过程。

试一试

向页面中添加用户控件

在本练习中将向母版页添加用户控件 Banner.ascx，因此它在站点中每个页面的边栏区域中显示一条横幅。

- (1) 从 MasterPages 文件夹中打开 Frontend.master，并将它切换到 Design 视图中。
- (2) 定位到允许选择主题的下拉列表，将光标放在下拉列表后面，按下 3 次 Enter 键创建一些空间。
- (3) 在 Solution Explorer 中，从 Controls 文件夹中将文件 Banner.ascx 拖到刚刚创建的空地方。这时 Design 视图会更新，如图 8-5 所示。

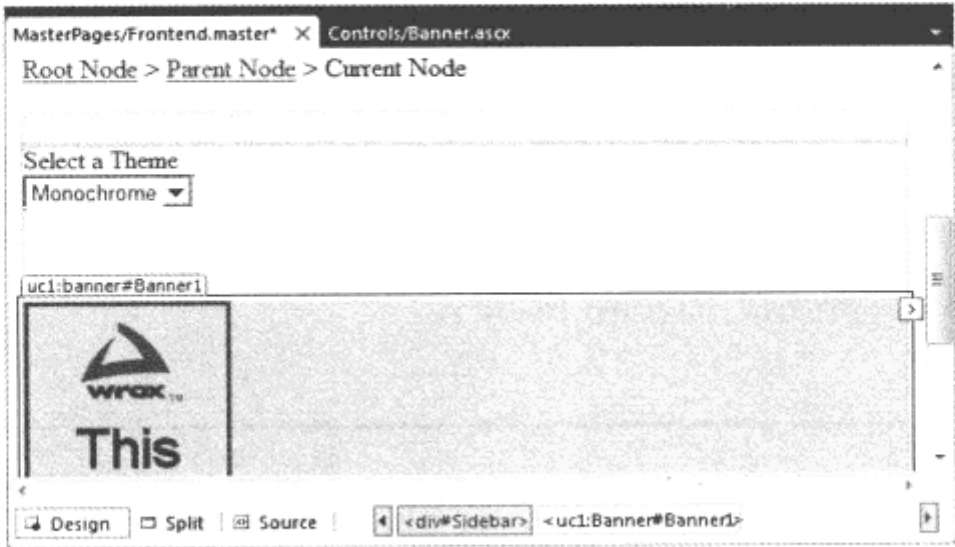


图 8-5

常见错误：如果 Design 视图看起来不像图 8-5 中所示的那样，而是比较接近文件最终出现在浏览器中的样子，那么可能是因为在 web.config 文件中仍然设置了 styleSheetTheme，也可能是因为选中了 View | Visual Aids 或 View | Formatting Marks 菜单中的某些选项。

- (4) 切换到 Markup 视图并定位到文件上方的@ Register 指令。将 src 特性中的两点改为符号(~)：

```
<%@ Register Src="~/Controls/Banner.ascx" TagName="Banner" TagPrefix="uc1" %>
```

- (5) 保存对母版页所做的修改，并关闭它。

(6) 从主题文件夹中打开文件 **Monochrome.css**，并添加如下的 CSS 声明：

```
img
{
    border: 0;
}
```

(7) 把这个声明复制到其他主题中，即添加到 **DarkGrey.css** 文件中。

(8) 保存所有修改，在 **Solution Explorer** 的站点根文件夹中右击 **Default.aspx**，选择 **View in Browser** 命令。

(9) 横幅现在显示在下拉列表的下方。切换到其他主题，会发现出现了同样的横幅。当单击横幅时，会打开一个新窗口，它会将您带到在上一个“试一试”练习中链接到的站点。

工作原理

当将用户控件拖到母版页的设计界面上时，VWD 会执行两个任务：首先添加 **@ Register** 指令，告诉页面到何处寻找用户控件。然后在下拉列表下方添加控件声明。

当加载页面时，ASP.NET 运行库会查看控件声明，并在指定位置注入控件的输出。在本例中，**Panel**、**<a />**元素及 **Image** 被插入到页面的边栏区域中。如果在浏览器中查看页面 HTML，将发现下列代码：

```
</select>
<br /><br /><br />
<div id="Banner1_VerticalPanel">
    <a href="http://p2p.wrox.com" target="_blank">
        
    </a>
</div>
```

Panel 控件被转换成了 HTML **<div />**元素，**Image** 控件被转换成了 ****元素。由于锚元素(**<a />**) 在用户控件中是用纯 HTML 定义的，因此它最后就会显示为所编写的代码。

注意面板的 **id** 从 **VerticalPanel** 改成了客户端 ID—— **Banner1_VerticalPanel**。这对于给 **<div>** 标记指定一个用在客户端脚本中的唯一客户端 **id** 特性来说是必需的。同样，****元素的 **<id>**也是如此。本章后面会用一个小节介绍关于它的更多信息。

通常将 ****元素放在 **<a />**元素内以链接它时，浏览器会在图像四周绘制一个边框。对于未访问过的链接，这个边框通常是蓝色的，而对于以前访问过的链接，边框往往是紫色的。如果要删除这个边框，可以使用下面的 CSS：

```
img
{
    border: none;
}
```

ASP.NET 的早期版本会自动地将 **border-width** 设为 0 像素(使其不可见)以引入边框属性，使之作为 ****标记中的内联样式。由于内联样式表会丢弃内嵌及外部样式表中的设置，因而如果想再次改变它的话，使用自己的 CSS 来影响边框会更加困难。幸运的是，这个问题在 ASP.NET 4 中得以解

决。现在有更多的选择来控制图像四周的边框，但是如果不想让它们使用这里显示的 CSS，则需要显式地删除它们。

当向页面中添加一个用户控件时，VWD 默认会用一个相对路径来引用控件。在这个“试一试”练习中，这个路径首先用两个句点(.)来表示父文件夹，后面跟着 Controls 文件夹，最后是控件的名称：

```
<%@ Register Src="../../../Controls/Banner.ascx" TagName="Banner" TagPrefix="uc1" %>
```

将两个句点改为符号(~)，在站点中移动页面就变得更简单了，因为不管使用控件的页面位于何处，现在 src 特性总是指向应用程序根文件夹中的 Controls 文件夹。

~语法使得带用户控件的页面稍微容易管理一些，还有一种更容易的在站点范围注册用户控件的方式。

8.1.3 用户控件的站点范围注册

如果有一个期望在站点的单独内容页上使用得非常频繁的控件，如前面示例中的横幅，可以在 web.config 文件中全局地注册这个控件。这样，它会变得在整个站点内可用，而不需要在每个页面上注册。下面的“试一试”练习显示了如何做到这一点。

试一试

在 web.config 文件中注册用户控件

在本练习中将在 web.config 文件中注册 Banner.ascx 用户控件。然后可以从母版页中去掉@Register 指令，因为不再需要它了。当修改了 web.config 文件后，向其他页面添加同样的用户控件时就不再需要向页面添加@ Register 指令了。

(1) 打开站点根文件夹中的 web.config 文件。如果熟悉 ASP.NET 的早期版本，则会发现文件 web.config 是空的。不要担心，它所包含的功能并没有从.NET 中删除。放置在 Web 站点的 web.config 文件中的配置信息现在被移到了中心 web.config 和 machine.config 文件中，以应用于整个站点。这样就生成了一个更干净的配置文件，使得更容易找到自己放置在那里的文件。

(2) 定位到在第 6 章中用到的<pages />元素来应用主题，在它的标记之间添加下面突出显示的代码，其中包含一个带有<add />子元素的<controls />元素

```
<pages theme="Monochrome">
  <controls>
    <add tagPrefix="Wrox" tagName="Banner" src="../../../Controls/Banner.ascx" />
  </controls>
</pages>
```

(3) 保存修改并关闭文件。

(4) 再次在 Markup 视图中打开母版页，并定位到边栏区域中的 Banner 控件。将 uc1 改为 Wrox:

```
<Wrox:Banner ID="Banner1" runat="server" />
```

如果用户控件的声明有它自己的结束标记，一定也要更新它(或者把它改成一个自结束标记):

```
<Wrox:Banner ID="Banner1" runat="server"></Wrox:Banner>
```

(5) 在母版页文件中一直向上滚动，并删除@ Register 指令所在的整行代码。

- (6) 保存并关闭母版页。
- (7) 再次在浏览器中打开 Default.aspx。这时会注意到横幅仍然出现在边栏区域中。



常见错误：如果得到一个错误，请确保向 web.config 文件中的正确位置添加了正确的代码，还要确保在母版页的控件声明中将 ucl 改为了 Wrox 并从母版页中删除了 @ Register 指令。

工作原理

如果在母版页中没有 @ Register 指令，ASP.NET 运行库就会扫描 web.config 文件寻找在那里注册的控件。然后它会发现 Wrox:Banner 控件的注册记录，因此它能成功地用 src 属性找到该文件，并将它的内容添加到页面层次结构中。

当向母版页中添加控件注册记录时，移动或重命名控件就变得更加容易。不需要在使用 @ Register 指令的所有页面中查找与替换它，唯一要做的事情就是修改 web.config 文件中的注册记录。当在那里作了修改后，使用控件的所有页面会自动找到新的位置或者用户控件的名称。

虽然用户控件很有用，但它们还有一些地方需要注意。

8.1.4 关于用户控件的警告

前面曾提到过，添加到页面中的 Panel 控件的 ID 被 ASP.NET 运行库修改了。这时不会得到 id 设置为 VerticalPanel 的 <div />元素，而是会得到下面的 id:

```
<div id="Banner1_VerticalPanel">
    ...
</div>
```

在很多情况下，这不是问题，因为通常不需要客户端 id。然而，如果需要从客户端 JavaScript 或 CSS 中访问该控件，那么了解为什么修改这个 id 就很重要了。

1. 理解和管理客户端 ID

在设计时，HTML 页面中的所有元素都要唯一。给它们提供 id 并不是必需的，但是如果提供了，就一定要确保它们是唯一的。为了避免页面的最终 HTML 代码中的冲突，ASP.NET 通过将它们的命名容器的名称作为前缀来确保每个服务器端元素获得一个唯一的客户端 ID。在命名容器内，所有元素都应当有唯一的 ID。当试图添加一个没有唯一服务器 ID 的控件时，VWD 就会发出警告。例如，当试图添加 ID 为 VerticalPanelpanel 的第二个 panel 控件时，就会得到一个错误。但是如果将两个 Banner 控件都放在同一个页面中或者将一个放在母版页中另一个放在内容页中，会怎么样呢？最终可能出现两个 ID 为 VerticalPanel 的客户端 <div />元素。为了避免出现这个问题，ASP.NET 用最接近的命名容器的 ID 作为每个元素的前缀。对于用户控件内的 Panel，这意味着它以母版页中用户控件的服务器端 ID(Banner1)为前缀。

可以用控件的 ClientID 来获得它的完整客户端 id。下面的代码段显示了如何在 Banner.ascx 用户控件内的 Label 控件上显示 Panel 控件的 ClientID:

VB.NET


```
Label1.Text = VerticalPanel.ClientID
```

C#

```
Label1.Text = VerticalPanel.ClientID;
```

运行这段代码，Label 控件的 Text 属性会包含 Panel 的客户端 ID：Banner1_VerticalPanel。第 9 章将介绍更多使用 ClientID 的实用示例。

使用显式 ID，就更容易预知客户端 HTML 元素的最终 id，从而更容易引用 JavaScript 或 CSS 中的那些元素。



注意：由于 ASP.NET 运行库能修改 HTML 元素的客户端 id 特性，因此用 CSS ID 选择器引用元素时可能会有些麻烦。解决这个问题最容易的方式是使用类选择器代替 ID 选择器。在使用嵌入的样式表时，也可以使用控件的 ClientID。

ASP.NET 4 中还引入了一个新的影响客户端 ID 的属性:ClientIDMode。

2. ClientIDMode 简介

从 ASP.NET 4 开始，每个 Web 控件都有一个 ClientIDMode 属性，用于确定创建客户端 ID 的方式。可以用表 8-2 所示的 4 种值中的任意一种来设置 ClientIDMode：

表 8-2

值	说 明
AutoID	与 ASP.NET 的以前版本一样，用于生成 ID 值
Predictable	这个值主要用于数据绑定控件(将在第 13 章及之后章节中介绍)，允许为重复元素创建可预见的客户端 ID。控件的客户端 ID 是通过连接父控件的客户端 ID 和控件自身的服务器 ID 生成的。该 ID 可以使用 ClientIDRowSuffix 属性为每个元素选择性地扩展为一个唯一的值
Static	使用这个值，客户端 ID 与设置的服务器端 ID 将会完全相同。这样可以显式地设置客户端 ID，并给予更多的控制。然而，它并不阻止两次都设置为同一个值，这可能导致客户端的 HTML 中有重复 ID 存在。所以应该谨慎使用
Inherit	使用这个属性，控件将从它的父控件中继承 ClientIDMode 值

当前，Planet Wrox Web 站点并没有从改变现有控件的客户端 ID 属性中获得什么好处，因此现在没有必要改变它们中的任何一个。然而，在后面的章节中，将再次用到 ClientIDMode 属性来创建干净的客户端 ID。

虽然当前横幅用户控件使得在站点中的各个位置显示横幅很容易，但是它还不够智能。它能做的只是显示一个链接图像，仅此而已。为了提高它的可用性，可以向控件添加行为，使它能在站点中的页面上产生不同的行为。

8.2 向用户控件添加逻辑

虽然使用控件创建重复性的内容已经非常有用，但如果向它们添加自定义逻辑，它们就会更加有用。通过向用户控件添加公有属性或方法，可以影响控件运行时的行为。当向用户控件添加一个属性时，它会自动地在正在使用的页面中的控件的 IntelliSense 中和 Properties 面板中变得可用，使得改变外部文件(如页面)中的行为变得更容易。

为了向用户控件中添加属性和方法，可以将它们添加到控件的 Code Behind 文件中。添加的属性可以是各种形式的属性。属性的最简形式看起来就像在第 5 章中看到的属性。对于更高级的情况，需要添加能跨回发维护它们的状态的 View State 属性。下面两个“试一试”练习将介绍如何创建这两种类型的属性。

8.2.1 为属性创建自己的数据类型

为了让横幅更有用，可以向它添加显示为水平横幅的第二个图像。也可以向用户控件添加属性，以确定是显示垂直图像还是水平图像。这一操作可以通过创建类型 System.Byte 的一个数值属性来完成。如 0 表示竖直，1 表示水平。然而，这样很难记住哪个数字代表什么。可以通过创建一个可接受 Horizontal 和 Vertical 这样的值的 String 属性使它更容易记清。然而，在开发时不能检查字符串，因此可能会出现拼写错误，导致抛出一个错误或者显示不正确的横幅。.NET Framework 提供了一个很好的方法解决了这个问题：可以以枚举的形式创建自己的数据类型。使用枚举(enumeration，或简称为 enum)形式可以将数字指派给人性化的文本字符串。然后开发人员使用这种可读的文本，而在后台使用的是数值。下面的代码段显示了枚举的一个基本示例(注意，在 VB 示例中没有使用逗号，但在 C#中必须使用)：

```
VB.NET
Public Enum Direction
    Horizontal
    Vertical
End Enum
```

```
C#
public enum Direction
{
    Horizontal,
    Vertical
}
```

使用这些枚举，编译器会自动将从 0 开始往上计数的数值指派给 Horizontal 和 Vertical 成员。如果愿意，也可以显式地定义数值：

```
VB.NET
Public Enum Direction
    Horizontal = 0
    Vertical = 1
End Enum
```

```
C#
public enum Direction
{
    Horizontal = 0,
    Vertical = 1
}
```

关于枚举最酷事情是可以在代码文件、Properties 面板，甚至在用户控件的代码编辑器中得到 IntelliSense。图 8-6 显示了 C#代码文件中的 IntelliSense。

在图 8-7 中可以看到在 Properties 面板中带有枚举值的相同列表。

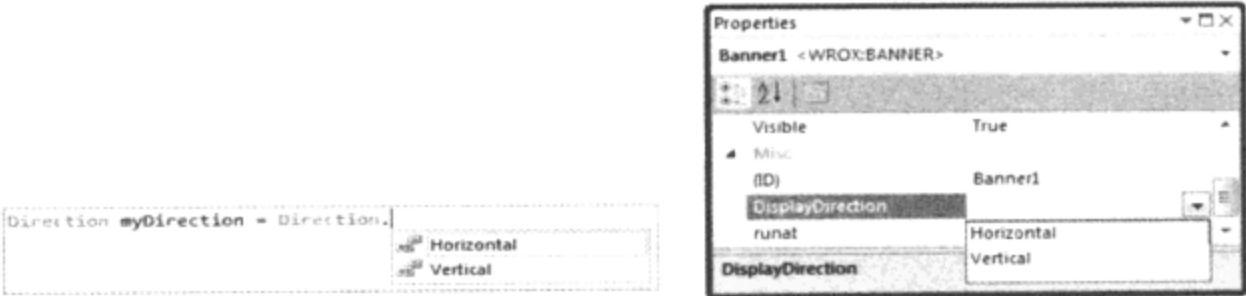


图 8-6 图 8-7

在图 8-8 中可以看到在 Banner 用户控件的代码文件中带有枚举值的相同列表。

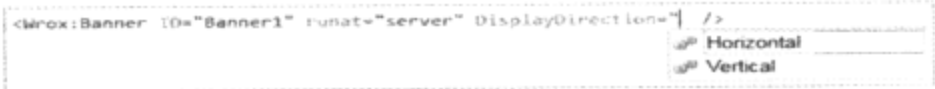


图 8-8

与其他代码文件(如类)一样，应将枚举放在 App_Code 文件夹下面的文件中。如果有多个枚举，可以将它们都存储在同一个文件中，或者为每个枚举创建一个单独的文件。

枚举对于简单而简短的列表来说非常好用。它们可以帮助快速找到正确的项，不需要记住像 0 或 1 这样的数字，而是可以使用人类可读的文本字符串。

下面的“试一试”练习将介绍如何创建一个枚举，并在 Banner 用户控件中使用它。

试一试 创建智能用户控件

在本练习中，将向用户控件中添加第二个横幅。这个横幅会在它自己的面板中显示为水平图像。为了避免同时显示两个横幅，所以要添加一个确定显示哪个横幅的属性。然后使用控件的页面就能定义正确的横幅。

(1) 先创建包含两个不同方向的成员的枚举：竖直与水平。为此，右击 App_Code 文件夹并选择 Add New Item 命令。添加一个名为 Direction 的类文件，并选择编程语言。

(2) 打开文件后，清除它的内容并向其中添加下列代码：

```
VB.NET
Public Enum Direction
    Horizontal
    Vertical
End Enum
```

```
C#
public enum Direction
{
    Horizontal,
    Vertical
}
```

(3) 保存并关闭文件。

(4) 打开用户控件 Banner.ascx 的 Code Behind 文件并添加下面的属性。为了帮助创建这个属性，VWD 提供了一个简便的代码段。要在 C# 中激活该代码段，输入 prop 后按 Tab 键两次。VWD 会添加自动属性的代码结构。在 VB.NET 中，输入 Property 后按 Tab 键两次。但是，这样做会创建一个完整的属性而不是创建一个自动属性(这个“试一试”练习中使用的是自动属性)，因此在这种情况下，最好手动输入代码。插入代码之后，可以再次按下 Tab 键在字段之间移动，分别输入正确的数据类型、属性名。完成后的代码如下：

```
VB.NET
Public Property DisplayDirection As Direction
```

```
C#
public Direction DisplayDirection { get; set; }
```



常见错误：确保在 Page_Load 方法(当使用的是 C# 时)之外，作为结束的 End Class (在 VB.NET 中)或者类的结束花括号(在 C# 中)之前添加属性。

注意，属性的名称为 DisplayDirection，它的类型是 Direction，即之前定义的枚举。

(5) 打开 Banner.ascx 的 Markup 视图，复制整个 <asp:Panel>，并将它粘贴到现有 Panel 下方。将控件命名为 HorizontalPanel，并将图像的 src 设置为 ~/Images/Banner468x60.gif。如果想要浏览这个图像而不直接输入它的路径，把光标放到 Markup 视图中 src 属性值的起始括号后边，然后按下 Ctrl 和空格键。在出现的菜单中选择 Pick URL 命令，这样就可以浏览文件。代码现在应如下所示：

```
<asp:Panel ID="HorizontalPanel" runat="server">
  <a href="http://p2p.wrox.com" target="_blank">
    <asp:Image ID="Image2" runat="server" AlternateText="This is a sample banner"
      ImageUrl="~/Images/Banner468x60.gif" />
  </a>
</asp:Panel>
```

(6) 切换回控件的 Code Behind 文件，并向 Page_Load 处理程序中添加下列突出显示的代码。在 C# 中，处理程序应当已经在那儿了。在 Visual Basic 中，可以从文档窗口上方的左边下拉列表中选择 Page Events，并从右边下拉列表中选择 Load 来建立处理程序，或者在 Design 视图中双击用户控件。

```
VB.NET
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    HorizontalPanel.Visible = False
```



```
VerticalPanel.Visible = False

Select Case DisplayDirection
    Case Direction.Horizontal
        HorizontalPanel.Visible = True
    Case Direction.Vertical
        VerticalPanel.Visible = True
End Select
End Sub
```

```
C#
protected void Page_Load(object sender, EventArgs e)
{
    HorizontalPanel.Visible = false;
    VerticalPanel.Visible = false;

    switch (DisplayDirection)
    {
        case Direction.Horizontal:
            HorizontalPanel.Visible = true;
            break;
        case Direction.Vertical:
            VerticalPanel.Visible = true;
            break;
    }
}
```

- (7) 保存并关闭组成用户控件的两个文件，因为目前对它们的操作已经完成。
- (8) 再次在 Markup 视图中打开母版页文件，并定位到用户控件声明。在 runat="server"属性后添加下面的设置正确图像类型的 DisplayDirection 属性：

```
<Wrox:Banner ID="Banner1" runat="server" DisplayDirection="Horizontal" />
```

IntelliSense 会帮助从列表中选择正确的 DisplayDirection。如果没有出现 IntelliSense，则等待几秒钟，直到 VWD 捕获所有修改。也可以关闭所有文件然后打开母版页，再试一次。

(9) 保存所有修改并在浏览器中请求页面 Default.aspx。注意右边栏区域现在包含了水平图像，其中中断了一点布局，因为图像对于边栏区域来说太宽了。

(10) 切换回母版页并将 DisplayDirection 属性从 Horizontal 改为 Vertical。保存修改并刷新浏览器中的页面。边栏现在应显示竖直横幅，如图 8-9 所示。

(11) 在 Markup 视图中打开 About 文件夹中的页面 AboutUs.aspx。如果没有那个页面，就先创建它。在 cpMainContent ContentPlaceHolder 中添加一些描述您或您的公司的文本，以及您创建该站点的原因。切换到 Design 视图并从 Solution Explorer 中把 Banner.ascx 控件拖到设计界面上刚刚创建的文本的下方。VWD 在母版页中检测到用户控件 Banner1，并将 Banner2 的 ID 赋予刚刚拖放的用户控件。注意在 Design 视图中看到的 4 个横幅：两个来自母版页中的 Banner 控件，另外两个来自 About Us 页面的 Banner 控件本身。在运行期间，其中两个将被隐藏。

(12) 在 Design 视图中选择控件，打开它的 Properties 面板并设置 DisplayDirection 为 Horizontal。

(13) 保存所有修改然后按下 Ctrl+F5 组合键在浏览器中打开 About Us 页面。除了右边栏中的横幅外，现在应当还能看到出现在 Content 区域中的水平横幅。注意，图 8-9 显示了执行步骤(10)后的

结果，这里没有将水平横幅显示在主内容区域。

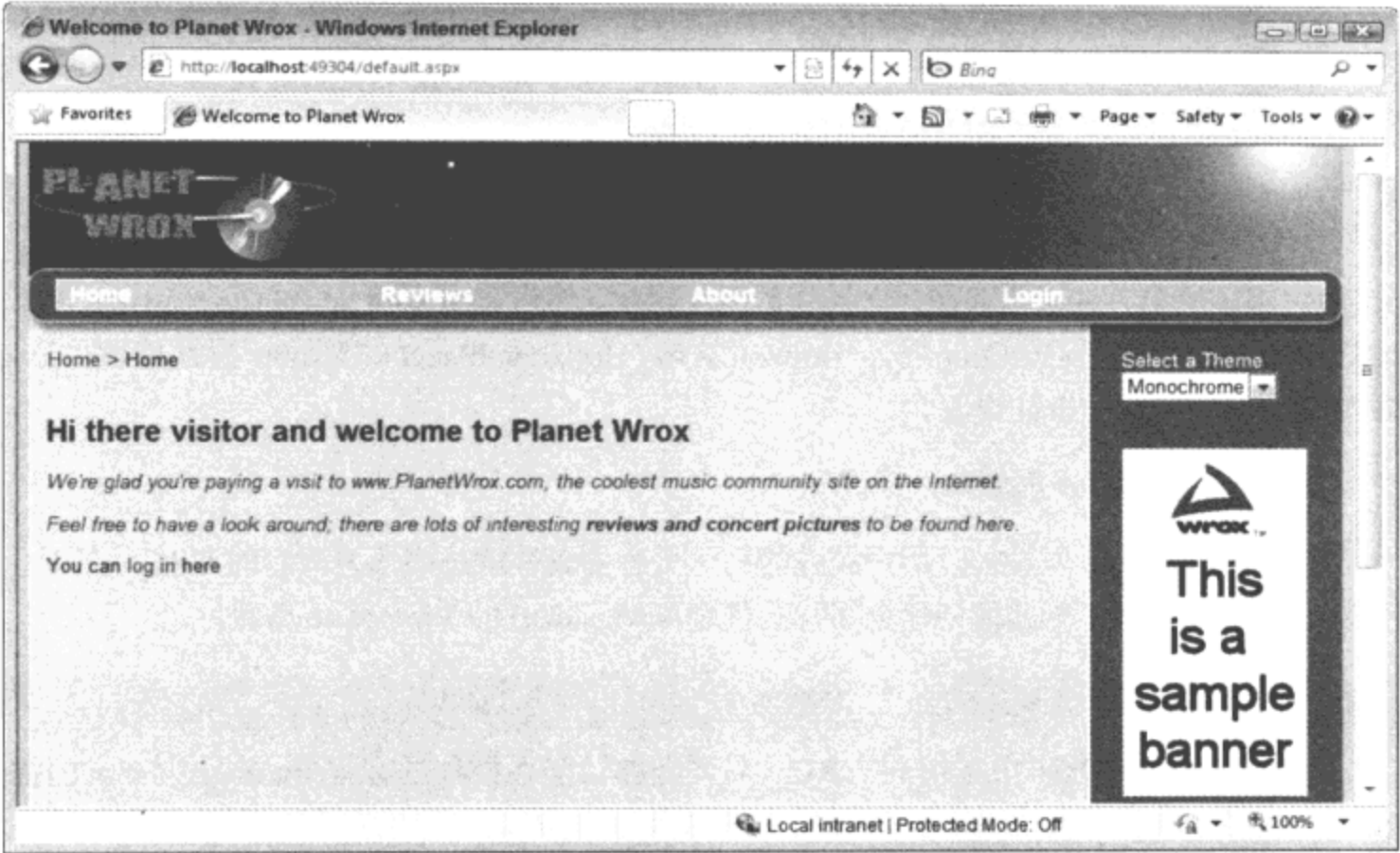


图 8-9

工作原理

DisplayDirection 属性给用户控件提供了一些额外的行为。使用控件的页面现在应像下面这样设置 DisplayDirection:

```
<Wrox:Banner ID="Banner1" runat="server" DisplayDirection="Horizontal" />
```

当 ASP.NET 运行库创建了控件实例后，在控件声明中设置的值就会指派给属性 DisplayDirection。当页面加载时，就会执行用户控件中的代码，如下所示：

VB.NET

```
HorizontalPanel.Visible = False
VerticalPanel.Visible = False

Select Case DisplayDirection
    Case Direction.Horizontal
        HorizontalPanel.Visible = True
    Case Direction.Vertical
        VerticalPanel.Visible = True
End Select
```

C#

```
HorizontalPanel.Visible = false;
VerticalPanel.Visible = false;

switch (DisplayDirection)
```

```
{
    case Direction.Horizontal:
        HorizontalPanel.Visible = true;
        break;
    case Direction.Vertical:
        VerticalPanel.Visible = true;
        break;
}
```

这段代码做的第一件事是隐藏包含图像的两个横幅，然后它用一个 Select Case/ switch 块来确定显示哪个图像。当 DisplayDirection 等于 Horizontal 时，HorizontalPanel 的 Visible 属性被设置为 True。同样的原则也适用于 Vertical 设置。

8.2.2 实现 View State 属性

除了 DislayDirection 属性外，用户控件的另一个有用属性就是横幅链接到的 URL。在下一节中将看到如何实现它，并了解如何创建免回发的名为 NavigateUrl 的 View State 属性。

试一试 实现 NavigateUrl 属性

为了能设置用户从代码中取得的 URL，需要能够访问在控件的标记(markup)中定义的锚标记(anchor tag)。要做到这一点，需要给它指定一个 ID 和 runat="server"属性。也可以将简单的<a>标记改为它的服务器端对应标记：<asp:HyperLink>。为了能通过编程设置新的要添加到 Banner 控件的 NavigateUrl 属性并确保它免回发，需要实现一个 View State 属性。为了说明为什么需要 View State 属性，本练习的前 3 步将修改 About Us 页面，以便能通过编程设置 Banner 控件的 DisplayDirection 属性。然后将导致一个回发，因此能看到失去方向的值，因为它没有在 View State 中维持状态。然后本练习的第二部分显示了如何实现能维持自身状态的 NavigateUrl 属性。

(1) 打开页面 AboutUs.aspx 的 Code Behind 文件，并向 Page_Load 事件处理程序中添加下列突出显示的代码。如果处理程序还不存在，切换到 Design 视图并双击页面上灰色区域的某处。

```
VB.NET
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    If Not Page.IsPostBack Then
        Banner2.DisplayDirection = Direction.Vertical
    End If
End Sub

C#
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        Banner2.DisplayDirection = Direction.Vertical;
    }
}
```

确认页面的 Markup 视图中有 ID 为 Banner2 的 Banner 用户控件，否则相应地更新代码。
(2) 切换到 Design 视图，从页面(不是在母版页中定义的那个页面)中 Banner.ascx 控件上方的

Toolbox 的 Standard 类别中拖出一个 Button 控件。不需要为 Button 控件的 Click 事件写任何代码。

(3) 保存页面并在浏览器中打开它。由于在 Page_Load 中添加的代码，因此页面首次加载时，屏幕底部的横幅会显示为竖直图像。现在单击按钮就会重新加载页面。但是这次，页面会显示水平的图像。因为当 Page.IsPostBack 为 False 时仅在 Page_Load 中设置了 Banner 控件的 DisplayDirection 属性，因此在回发时设置就会丢失，导致横幅恢复回它的默认设置 Horizontal。

(4) 为了避免使用 NavigateUrl 时出现这个问题，需要将它作为一个 View State 属性实现，该属性使用 ViewState 集合作为后台变量存储底层数据。这样，值就存储在 ViewState 中并发送给浏览器，并在每次请求时发送回服务器。要实现该属性，可以向 Banner.ascx 用户控件的 Code Behind 文件中、之前创建的 DisplayDirection 属性下面添加下列代码。

注意，不需要手动输入所有这些代码，因为本书附带的代码下载文件中包含它们。

VB.NET

```
Public Property NavigateUrl() As String
    Get
        Dim _navigateUrl As Object = ViewState("NavigateUrl")
        If _navigateUrl IsNot Nothing Then
            Return CType(_navigateUrl, String)
        Else
            Return "http://p2p.wrox.com" ' Return a default value
        End If
    End Get
    Set(ByVal Value As String)
        ViewState("NavigateUrl") = Value
    End Set
End Property
```

C#

```
public string NavigateUrl
{
    get
    {
        object _navigateUrl = ViewState["NavigateUrl"];
        if (_navigateUrl != null)
        {
            return (string)_navigateUrl;
        }
        else
        {
            return "http://p2p.wrox.com"; // Return a default value
        }
    }
    set
    {
        ViewState["NavigateUrl"] = value;
    }
}
```

(5) 切换到用户控件的 Markup 视图，并向两个链接添加 runat="server"特性。将竖直面板中的链接的 ID 设为 VerticalLink，而另一个链接的 ID 设为 HorizontalLink。代码最后应如下所示：

```
<a href="http://p2p.wrox.com" target="_blank" runat="server" id="VerticalLink">
...
<a href="http://p2p.wrox.com" target="_blank" runat="server" id="HorizontalLink">
```


(6) 切换回用户控件的 Code Behind 文件(按 F7 键)，并修改用户控件的 Page_Load 处理程序，让它也设置锚元素的 HRef 属性。

```
VB.NET
Case Direction.Horizontal
    HorizontalPanel.Visible = True
    HorizontalLink.HRef = NavigateUrl
Case Direction.Vertical
    VerticalPanel.Visible = True
    VerticalLink.HRef = NavigateUrl
C#
case Direction.Horizontal:
    HorizontalPanel.Visible = true;
    HorizontalLink.HRef = NavigateUrl;
    break;
case Direction.Vertical:
    VerticalPanel.Visible = true;
    VerticalLink.HRef = NavigateUrl;
    break;
```

(7) 保存修改并回到 AboutUs.aspx 页面的 Code Behind 文件中。修改代码并将 Banner 控件的 NavigateUrl 属性设置为不同的 URL。应重写设置 DisplayDirection 的代码。

```
VB.NET
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    If Not Page.IsPostBack Then
        Banner2.NavigateUrl = "http://imar.spaanjaars.com"
    End If
End Sub
C#
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        Banner2.NavigateUrl = "http://imar.spaanjaars.com";
    }
}
```

(8) 保存所有修改然后按下 F5 键在浏览器中请求页面 AboutUs.aspx。



注意：一定要刷新浏览器窗口，因此要先关闭可能已经打开的所有窗口。

单击页面左边的水平横幅，这时会弹出一个显示了上一步中设置的 URL 的新窗口。

(9) 关闭这个新窗口，并单击以前添加到 AboutUs.aspx 页面中的按钮，以引发向服务器的一个

回发。重新加载页面后，再次单击横幅图像。这时会被带到与第(8)步相同的站点。这表明了一点：不像以前添加到用户控件的 `DisplayDirection` 属性，`NavigateUrl` 属性现在能跨回发维持它的值了。

工作原理

在前 3 步中演示了非 `View State` 属性的行为。首先在 `Page_Load` 事件处理程序中写了一些通过编程设置 `DisplayDirection` 属性的代码：

```
VB.NET
If Not Page.IsPostBack Then
    Banner2.DisplayDirection = Direction.Vertical
End If
```

```
C#
if (!Page.IsPostBack)
{
    Banner2.DisplayDirection = Direction.Vertical;
}
```

由于对 `Page.IsPostBack` 的检查，因此只有当页面首次加载时才会激活此代码。因为有回发，所以在页面重新加载时不会激活这段代码。当它激活时，它会设置 `Banner` 控件的 `DisplayDirection` 属性，使横幅显示正确的图像。然而，一旦页面被回发，这个值就会丢失，控件的 `DisplayDirection` 会恢复为默认的 `Horizontal`。解决这个问题的一种方法是确保代码在首次和接下来的回发中都会激活。去掉对 `Page.IsPostBack` 的检查足以完成这个任务。然而，这并不总是理想的解决方案。假设打算正确地显示数据库中的方向。由于从数据库中取数据是开销很大的操作，因此打算最小化访问数据库的次数。在这样的情况下，只有当页面首次加载时，开发人员才会取数据，并期望它会在接下来的回发中一直存在。这就是 `NavigateUrl` 属性所做的工作。设置它的值一次，它就一直可用，即使将页面发送回服务器也是如此。这是用 `View State` 属性完成的。

要了解它的工作原理，先看一下该属性的 `setter`：

```
VB.NET
Public Property NavigateUrl() As String
...
Set(ByVal Value As String)
    ViewState("NavigateUrl") = Value
End Set
End Property
```

```
C#
public string NavigateUrl
{
    ...
    set
    {
        ViewState["NavigateUrl"] = value;
    }
}
```

当将一个值指派给 `NavigateUrl` 属性时，它的值存储在 `ViewState` 集合中。可以将 `ViewState`

集合看作一个大包，它允许存储回发后可以再次检索的数据。View State 中的值用一个唯一键标识。在本例中键等于属性的名称，因此很容易看到它们在一起。一旦将一个值指派给一个 View State 属性，它就存储在第 4 章介绍过的页面中的隐藏_VIEWSTATE 字段中。这意味着当页面加载时会将它发送给浏览器，当再次回发页面时会将它发送回服务器。

当回发发生时，会再次激活用户控件的 Page_Load 中的代码。与初次请求一样，代码会访问 Select Case /switch 块中的 NavigateUrl 属性：

VB.NET

```
Case Direction.Horizontal
...
HorizontalLink.HRef = NavigateUrl
...
```

C#

```
case Direction.Horizontal:
...
HorizontalLink.HRef = NavigateUrl;
...
```

NavigateUrl 的值由该属性的 getter 返回：

VB.NET

```
Public Property NavigateUrl() As String
    Get
        Dim _navigateUrl As Object = ViewState("NavigateUrl")
        If _navigateUrl IsNot Nothing Then
            Return CType(_navigateUrl, String)
        Else
            Return "http://p2p.wrox.com" ' Return a default value
        End If
    End Get
    ...
End Property
```

C#

```
public string NavigateUrl
{
    get
    {
        object _navigateUrl = ViewState["NavigateUrl"];
        if (_navigateUrl != null)
        {
            return (string)_navigateUrl;
        }
        else
        {
            return "http://p2p.wrox.com"; // Return a default value
        }
    }
    ...
}
```

这段代码首先尝试从 View State 中取得值，在 VB.NET 中使用 ViewState("NavigateUrl")，在 C# 中使用 ViewState["NavigateUrl"] (用方括号访问集合中的项目)。如果返回的值是 Nothing 或 null，那么 getter 会返回属性的默认值：http://p2p.wrox.com。

然而，如果返回的值不是 Nothing，就会在 VB.NET 中用 CType、在 C# 中用 (string) 将它强制转换为字符串，最终返回给调用代码。最后，从 View State 属性中返回的 NavigateUrl 会被再次指派给锚标记的 HRef 属性，之后它被用作单击图像时用户将被带到的 URL。

8.2.3 关于 View State 要考虑的事项

虽然设计 View State 的目的是为了克服上一个“试一试”练习中指出的维护状态的问题，但是应仔细考虑要不要使用它。存储在 View State 中的值被发送到浏览器中，并在每次请求时发送回服务器。当在 View State 中存储很多值或较大的值时，这样会增加页面的大小，因而会给性能带来负面影响。千万不要将数据库记录这样的大型对象存储在 View State 中；如果需要存储的数据量比较大的话，则每次请求时从数据库中得到刷新数据往往比通过隐藏 View State 字段传递要快。由于 View State 是存储在页面中，之后再以某种方式进行传送，因此，不要用它来保存诸如密码这样的安全数据。

8.3 关于用户控件的实用提示

下面的列表提供了关于使用用户控件的一些实用提示：

- 不要过度使用用户控件。用户控件对于封装重复内容不错，但是它们也会使站点难以管理，因为代码和逻辑包含在多个文件中。如果不确定有些内容是否会在站点的另一部分中重用，先直接把它嵌在页面中。如果有需要，以后总是可以将它移到单独的用户控件中的。
- 保持用户控件集中于单个任务。不要创建可以显示 5 种不同类型的不相关内容，并用一个属性来确定显示哪一个。这样会使控件难以维护与使用。相反，可以创建 5 个轻量级的控件并适当地使用它们。
- 当创建包含应用了样式的标记的用户控件时，不要对包含在用户控件中的服务器控件硬编码样式信息，如 CssClass。而是考虑在用户控件上创建单独的 CssClass 属性，然后用它来设置服务器控件的 CssClass。这样会提高用户控件的可重用性，使它更容易合并不同设计中的控件。

8.4 本章小结

用户控件能大大提高站点的可维护性。不需要在站点中的很多不同页面上重复相同的标记和代码，可以将代码封装到单个控件中，然后在站点的不同区域使用该控件。

为了提高控件的有用性，可以向它们添加行为。创建带属性的控件是常事，属性可以在使用页面时设置，允许修改控件运行时的行为。虽然 View State 属性能解决可能遇到的一些状态问题，但是应仔细考虑是否确实需要它们。由于这些属性增加了页面的大小，因此可能会对站点的性能有负面影响。

可以通过跟踪各个图像被单击的次数进一步改进 Banner 控件。本书的示例站点 Planet Wrox 没有实现这个方法，不过当学习了关于数据库交互的章节后，就很容易自己实现了。

第 9 章将创建另一个作为联系表单的用户控件。通过将表单作为用户控件来构建，就很容易要求用户从站点的不同位置发送反馈。

8.5 练习

- 1. 本章介绍了如何创建标准属性和 View State 属性。两者的主要区别是什么？各自的缺点又是什么？
- 2. 当前，Banner 控件的 DisplayDirection 属性并不能在回发之间维持其状态。请修改该属性的代码，使它能用 ViewState 集合来维持自身状态，与 NavigateUrl 维护其值类似。
- 3. 使用 Direction 这样的自定义枚举优于使用 System.Byte 或 String 这样的内置类型的两个主要好处是什么？

练习的答案见附录 A。

本章要点回顾

@ Register 指令	用来注册用户控件并指向页面、母版页及其他用户控件内的源代码
备用文本	该属性可用来设置图像上的 alt 特性，当图像无法正常显示时，显示备用文本
Machine.config	应用于整个站点的最重要的.NET 配置文件，提供默认的 Web 站点设置
用户控件	可以在页面、母版页或其他用户控件中重用的内容块，存储在文件中，有一个.ascx 后缀和一个可选的 Code Behind 文件
View State 属性	在页面、母版页或用户控件级别使用的属性，其值保存在 View State 中，因而可以在回发过程中一直存在
ViewState 集合	ViewState 集合是 Page 类上的属性，可以用 View State 来存储和检索数据

验证用户输入有效性

本章要点

- 用户输入的概念，为什么要验证用户输入的有效性
- ASP.NET 4 提供了什么来帮助验证用户输入的有效性
- 如何使用内置的有效性验证控件，如何创建标准工具不支持的解决方案
- 如何使用 ASP.NET 发送电子邮件
- 如何读文本文件

到目前为止，本书一直是通过向站点中添加固定大小的页面以及导航菜单来创建一个相当静态的控制布局与内容的 Web 站点。如果能结合动态数据，那么站点就会生动得多。这种数据通常沿着两个方向流动：从服务器发送到终端用户的浏览器，或者数据由用户输入并发送到服务器进行处理或存储。

来自服务器的数据可以从很多不同的数据源检索得到，包括文件与数据库，并常常用 ASP.NET 数据控件表现出来。第 12 章及以后章节将介绍如何访问数据库。

另一种数据流来自用户，被发送给服务器。这种信息的范围相当广，从简单的页面请求与“联系我们”表单到复杂的购物车场景和类似向导的用户界面。这种数据流的底层原则在所有场景中基本相同：用户在 Web 窗体中输入数据，然后提交给服务器。

为了防止系统接收无效数据，在允许系统使用之前要先验证数据的有效性。幸运的是，ASP.NET 4 提供了一个大工具包，使得数据有效性验证变成一个简单的任务。

本章第一部分概述了 ASP.NET 支持的有效性验证控件。其中指出了哪些控件是可用的，如何使用与定制它们，以及在哪些场景中应用它们。

本章第二部分介绍了如何以其他方式使用数据。其中指出了如何通过电子邮件发送用户提交给系统的信息，以及如何用基于文本的模板定制邮件正文。

到本章最后，您将很好地了解到 ASP.NET Web 应用程序的信息流，以及各种验证数据有效性的技术。

9.1 收集用户数据

从严格意义上说，Internet 上的每个 Web 站点都要处理来自用户的输入。通常，可以用许多不同的技术将这种输入发送给 Web 服务器，其中两个最常用的是 GET 与 POST。第 4 章曾简要介绍过这两种方法之间的区别，在 GET 方法中数据被附加到所请求页面的实际地址后面，而使用 POST 方法的话，数据则放在所请求页面的主体中发送。

使用 GET 方法，数据被添加到页面的请求地址中。可以使用第 7 章介绍的方法，即，使用 Request 对象的 QueryString 属性检索它。假设请求下面的页面：

```
http://www.PlanetWrox.com/Reviews/ViewDetails.aspx?ReviewId=34&CategoryId=3
```

对于本例，查询字符串是 ReviewId=34&CategoryId=3。问号用来将查询字符串与地址的其余部分隔开，而查询字符串由&号分开的名/值对组成。每个名字和值又被等号(=)分开。要访问查询字符串中的单个项，可以使用 QueryString 集合的 Get 方法：

VB.NET

```
' Assigns the value 34 to the reviewId variable
Dim reviewId As Integer = Convert.ToInt32(Request.QueryString.Get("ReviewId"))
' Assigns the value 3 to the categoryId variable
Dim categoryId As Integer = Convert.ToInt32(Request.QueryString.Get("CategoryId"))
```

C#

```
// Assigns the value 34 to the reviewId variable
int reviewId = Convert.ToInt32(Request.QueryString.Get("ReviewId"));
// Assigns the value 3 to the categoryId variable
int categoryId = Convert.ToInt32(Request.QueryString.Get("CategoryId"));
```

另一方面，POST 方法用提交给服务器的控件从表单中获得数据。假设有一个带两个控件的表单：一个名为 Age 的 TextBox，用来保存用户的年龄；另一个为 Button，用于将年龄提交给服务器。在 Button 控件的 Click 事件中，可以编写下列代码将用户的输入转换为整数：

VB.NET

```
Dim age As Integer = Convert.ToInt32(Age.Text) ' age now holds the user's age
```

C#

```
int age = Convert.ToInt32(Age.Text);           // age now holds the user's age
```

注意，在本例中不需要像以前使用 QueryString 时那样访问 Form 这样的集合。ASP.NET 使您不用麻烦地手动从提交的表单中检索数据，而是用来自表单的数据填充页面中的各种控件。

只要用户在文本框中输入了一个看上去像年龄的值即可。但是，当用户有意或无意地提交了无效数据时，会发生什么情况呢？如果用户发送的是文本 I am 38 而不是数字 38 该怎么办？当发生这种情况时，代码就会崩溃。当传递给它某种不是表示数字的值时，Convert 类的 ToInt32 方法就会抛出一个异常(错误)。一旦抛出这个异常，页面执行就会完全停止。第 18 章将深入介绍处理异常的有关知识。

为了避免这些问题发生，需要验证发送给服务器的所有数据的有效性。当数据无效时，需要拒

绝它，并确保应用程序能够适当地处理它。

9.1.1 验证 Web 窗体中用户输入的有效性

关心验证用户输入有效性的人常常会说这句话：永远不要相信用户输入。虽然乍一看这样说好像是多疑症似的，但是在所有开放系统中这句话确实重要。即使自认为知道用户是谁，即使完全信任他们，但他们也往往不是能够访问系统的唯一用户。一旦站点发布到 Internet 上，就会成为伺机入侵系统的恶意用户和黑客的潜在目标。除了这些邪恶的访问者外，甚至值得信任的用户也可能在无意间向系统发送错误的数

据。为了尽可能帮助您克服这个问题，ASP.NET 配备了不少有效性验证控件，以帮助在将数据用于应用程序之前验证数据的有效性。下面几小节将介绍如何用标准有效性验证控件确保用户向系统中提交有效数据。

1. ASP.NET 有效性验证控件

ASP.NET 4 有 6 个在 Web 站点中进行有效性验证的控件。其中 5 个控件用来执行实际的有效性验证，而最后一个控件 ValidationSummary 用来向用户提供页面中出现的错误的反馈信息。图 9-1 显示了 Toolbox 的 Validation 类别中的可用控件。

有效性验证控件对于验证用户输入系统的数据的有效性极其有用。可以轻易地将它们与其他控件挂钩，如 TextBox 或 DropDownList；然而，它们也支持自定义有效性验证场景。图 9-2 演示了两个正在运行的有效性验证控件 RequiredFieldValidator 和 RangeValidator，它们能够防止用户提交没有输入必需而有效的数据的表单。

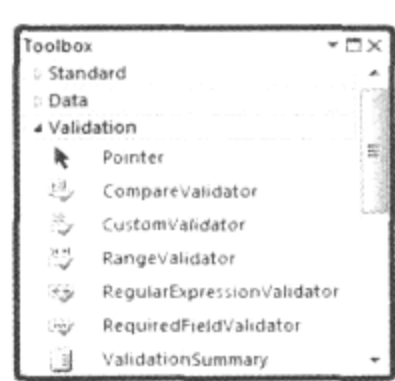


图 9-1

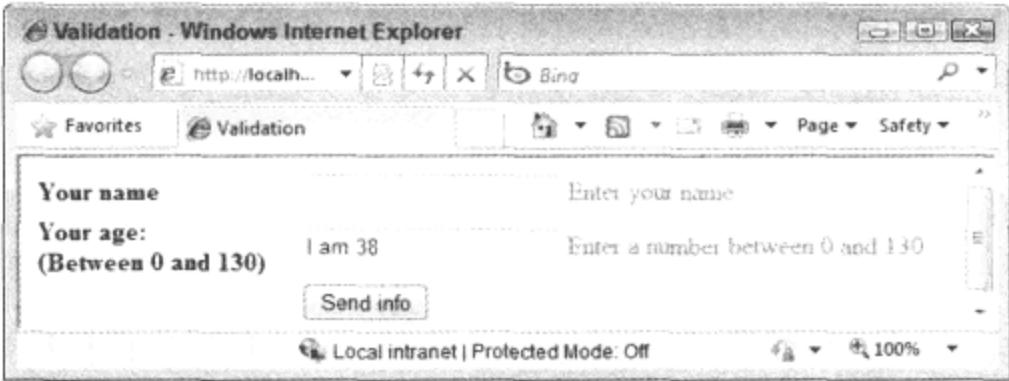


图 9-2

关于有效性验证控件最出色的事情是它们能在客户端和服务端上检查输入。当向 Web 页面中添加一个有效性验证控件时，控件就会呈现在客户端验证关联控件有效性的 JavaScript。大多数启用了 JavaScript 的现代 Web 浏览器(包括 IE、Firefox、Chrome、Opera 和 Safari)都能进行这种客户端有效性验证。同时，有效性验证也可以在服务器上自动进行。这样就容易向用户提供关于使用客户端脚本的数据的即时反馈，从而使 Web 页面在服务器上免受伪数据的侵扰。

2. 关于客户端有效性验证的警告

虽然客户端有效性验证看起来对于防止用户向系统发送无效数据来说似乎足够了，但是永远不要依赖它作为有效性验证的唯一方法。人们很容易在浏览器中禁用 JavaScript，从而使客户端有效性验证例程变得无用。此外，恶意用户能轻易地绕过浏览器中的整个页面直接向服务器发送信息，如

果没有应对的解决方法，服务器会愉快地接受并处理这些不安全的信息。

一般而言，应当将客户端有效性验证看作对用户的好意。它给用户提供了即时反馈，不需要向服务器发送完整的回发，就可以使他们知道他们忘了输入必需的字段，或者输入了不正确的数据。另一方面，服务器端有效性验证是唯一真正的有效性验证方式。这是防止无效数据进入系统的唯一有效方式。

下一节将讨论如何使用有效性验证控件来保护数据。

3. 使用有效性验证控件

要在 ASPX 页面中声明一个有效性验证控件，可以使用以往熟悉的声明语法。例如，要创建在图 9-2 中使用的 RequiredFieldValidator 控件，需要使用下面的代码：

```
<asp:RequiredFieldValidator ID="ReqVal1" runat="server" ControlToValidate="TextBox1"
    ErrorMessage="Enter your name" />
```

为了说明它们的工作过程，下面的“试一试”练习指出了使用一个位于用户控件的联系表单中的 RequiredFieldValidator 的过程。之后将深入讨论各种不同的有效性验证控件。



注意：VWD 提供了大量有用的代码段，可以快速插入类似 Markup 视图中的有效性验证控件那样的控件。在下面的“试一试”练习中，将介绍如何通过 Toolbox、Design 视图和拖放来添加必需的控件，但是了解如何在 Markup 视图中快速添加控件也是非常有用的。例如，要在 Markup 视图中插入一个 TextBox 控件，输入 textbox 并按下 Tab 键，VWD 会自动完成所有控件代码。要插入一个 RequiredFieldValidator 控件，输入字母 req，然后按下 Ctrl+空格键让 VWD 生成完整的单词：requiredfieldvalidator，然后再次按下 Tab 键以插入整个标记。如果直接在设置了 ID 的 TextBox 控件下做了这些工作，那么 VWD 甚至会设置正确的 ControlToValidate 特性。后面这种方式在下一个“试一试”练习中将不起作用，因为这些不同的控件并不是直接相邻的，而是被放置在不同的表单元格中。VWD 仍然会插入 RequiredFieldValidator 的代码，但需要您手动把 ControlToValidate 属性设置为关联的 TextBox 的 ID。

试一试 使用 RequiredFieldValidator 控件

本练习中将创建一个名为 ContactForm.ascx 的用户控件。它可以放在 Web 页面中使站点访问者能留下一些反馈。在后面的“试一试”练习中会通过电子邮件向 e-mail 账户发送响应来扩展此控件。

- (1) 打开 Planet Wrox 项目并在 Controls 文件夹中添加一个新的用户控件。将该控件命名为 ContactForm.ascx。确保它使用了指定的编程语言和一个 Code Behind 文件。
- (2) 切换到 Design 视图并通过选择 Table | Insert Table 命令插入一个表。创建一个 8 行 3 列的表。
- (3) 合并第一行的 3 个单元格。为此，选中这 3 个单元格，右击选择项，并选择 Modify | Merge Cells 命令。
- (4) 在合并后的单元格中输入一些文本，告诉用户他们可以用这个联系表单与您联系。

(5) 在第二行的第一个单元格中输入单词 Name。进入同一行的第二个单元格，拖动一个 TextBox 控件并将其 ID 设置为 Name。进入第二行的最后一个单元格，从 Toolbox 的 Validation 类别中拖动一个 RequiredFieldValidator 控件。最后，进入最后一行的第二个单元格，拖动一个 Button 控件。设置按钮的 ID 将它重命名为 SendButton，并设置它的 Text 属性为 Send。完成后，Design 视图应如图 9-3 所示。



图 9-3

(6) 在 Design 视图中单击 RequiredFieldValidator 一次，然后按下 F4 键打开它的 Properties 面板。设置控件的下列属性，如表 9-1 所示。

表 9-1

属 性	值
CssClass	ErrorMessage
ErrorMessage	Enter your name
Text	*
ControlToValidate	Name

- (7) 保存对用户控件所做的修改，然后关闭它，因为目前对它的处理已完成。
- (8) 为 Monochrome.css 和 DarkGrey.css 这两个主题在 CSS 文件中添加如下的 CSS 声明：

```
.ErrorMessage
{
    color: red;
}
```

保存并关闭这两个文件。


(9) 从 About 文件夹中打开文件 Contact.aspx，切换到 Design 视图。从 Solution Explorer 中将用户控件 ContactForm.ascx 拖到页面中用紫色边框标识的主内容区域。切换回 Markup 视图，将看到下面的控件声明：


```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <uc1:ContactForm ID="ContactForm1" runat="server" />
</asp:Content>
```

- (10) 保存页面然后按下 Ctrl+F5 组合键在浏览器中打开它。如果出现了一个错误，请确保将 TextBox 重命名为了 Name，并将 RequiredFieldValidator 控件上的 ControlToValidate 属性设置为了 Name。
- (11) 保持 Name 文本框空着，并单击 Send 按钮。注意页面并没有提交给服务器。相反，应看到一个红色星号出现在名字字段那一行的右边，表示出现了一个错误。如果星号不是红色的，则按下 Ctrl+F5 或 Ctrl+R 组合键从服务器上获得主题的 CSS 文件的一个新副本，并再次单击 Send 按钮。
- (12) 输入您的名字并再次单击 Send 按钮。页面现在成功地回发给了服务器。

工作原理

通过 ControlToValidate 属性将 RequiredFieldValidator 附加到 TextBox 后，在客户端验证控件有效性的 JavaScript 被发送给了浏览器。



注意：这是第一个需要自己实际编写 JavaScript 的章节。不要有太多的担心，因为不需要编写太多代码。即便以前没有任何 JavaScript 的经验，本例也非常易懂。如果要了解关于 JavaScript 的更多信息，可以参见 Wrox 出版社出版的 *Professional JavaScript For Web Developers*, 2nd Edition(Wrox, ISBN: 978-0-470-22780-0)一书。

RequiredFieldValidator 控件能验证另一个控件的有效性，如 TextBox。验证方法是将另一个控件的值与它自己的 InitialValue 属性作比较，确保另一个控件的值与它不同。在默认情况下，这个属性是一个空字符串，意味着除了空字符串的任何内容都会被看作有效值。每当用户试图通过单击 Send 按钮向服务器提交表单时，该有效性验证控件就会检查它附加到的控件。当文本框仍然为空时，Text 属性会显示星号(由 ErrorMessage CSS 类来格式化)，且表单不会被提交。我们将在本章的后面介绍如何使用和显示 ErrorMessage 属性。当用户在 Name 文本框中输入一些内容时，就会通过有效性验证，从而将页面成功地提交给服务器。

除了 RequiredFieldValidator 控件外，Toolbox 的 Validation 类别还包含若干其他的控件，下面将会讨论。

4. 标准有效性验证控件

Toolbox 中 Validation 类别下控件的名称以 Validator 结束的 5 个有效性验证控件基本上都继承自同一个基类，因此它们有一些共同的行为。这 5 个有效性验证控件中的 4 个以相同的方式工作，并包含允许验证关联控件的内置行为。最后一个控件 CustomValidator 允许编写非内置的自定义有效性验证规则。

表 9-2 列出了有效性验证控件共有的以及使用有效性验证控件时经常要用到的属性。

表 9-2

属 性	说 明
Display	这个属性确定隐藏的错误消息是否占用空间。如果将 Display 设置为 Static，错误消息就会占用屏幕空间，即使在隐藏时也是如此。这类似于前面章节介绍的 CSS 设置 visibility: hidden。Dynamic 设置使用 display: none 隐藏错误消息，直到错误消息必须显示为止。如果设置为 None，错误消息就根本看不到了。如果使用本章后面将要介绍的 ValidationSummary，这会很有用
CssClass	这个属性允许设置应用到错误消息文本的 CssClass 特性
ErrorMessage	这个属性保存用在 ValidationSummary 控件中的错误消息。当 Text 属性为空时，也用 ErrorMessage 值作为出现在页面上的文本
Text	Text 属性用作有效性验证控件显示在页面上的文本。它可以是一个星号(*)以表示出现一个错误，也可以是像“Enter your name.”这样的文本
ControlToValidate	这个属性包含需要验证有效性的控件的服务器 ID
EnableClientScript	这个属性用于确定控件是否提供客户端的有效性验证。默认为 True
SetFocusOnError	这个属性确定客户端脚本是否将焦点放在产生错误的第一个控件上。其默认设置为 False
ValidationGroup	有效性验证控件可以组合在一起，允许对选中的控件进行有效性验证。同一个 ValidationGroup 中的所有控件都会被同时检查，这意味着如果控件不是这个控件组的一部分，就不对它进行有效性验证。例如，假设有一个逻辑页面，其中有一个 Login 按钮，以及用于输入用户名和密码的字段。同时该页面也可能包含允许搜索站点的搜索框。使用 ValidationGroup，就可以让 Login 按钮验证用户名与密码框的有效性，而搜索按钮仅触发搜索框的有效性验证
IsValid	通常在设计时不会设置这个属性，不过在运行时它提供关于是否通过了有效性验证测试的信息

Text 和 ErrorMessage 属性之间的区别

乍一看，这两个属性的作用似乎是一样的。它们都可以用来以错误消息的形式向用户提供反馈。但是当与 ValidationSummary 控件结合起来使用时，两者之间就有了细微的区别。当同时设置这两个属性时，Validation 控件显示 Text 属性，而 ValidationSummary 控件则显示 ErrorMessage。图 9-4 显示了一个示例 Login 页面，它有两个 RequiredFieldValidator 控件。这两个有效性验证控件的 Text 属性都设置成了星号(*)，给用户提供一个可视化线索，表明有一个问题。然后控件下方的 ValidationSummary 控件就会显示完整的 ErrorMessage 属性。

我们已经了解了 RequiredFieldValidator 的工作原理，下面几节将介绍其余 3 个标准有效性验证控件。然后在后面一节会说明如何使用 CustomValidator 和 ValidationSummary 控件。

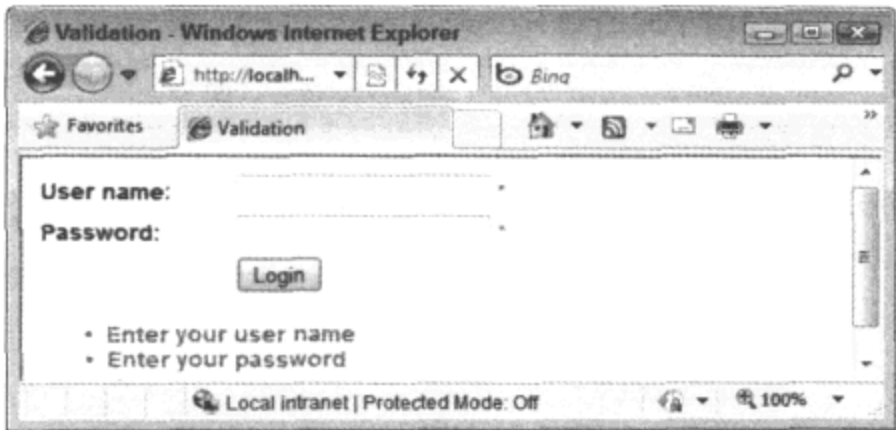


图 9-4

RangeValidator 控件

RangeValidator 控件允许检查一个值是否落在特定的范围内。这个控件能检查字符串、数字、日期和货币等数据类型。例如，可以用它来确保一个数字位于 0 到 10 之间，或者一个选中的日期位于今天和下两个星期之间。表 9-3 列出了它最重要的属性。

表 9-3

属 性	说 明
MinimumValue	这个属性确定可接受的最小值。例如，当检查 1 和 10 之间的整数时，将这个属性设置为 1
MaximumValue	这个属性确定可接受的最大值。例如，当检查 1 和 10 之间的整数时，将这个属性设置为 10
Type	这个属性确定有效性验证控件检查的数据类型。这个值可以设置为 String、Integer、Double、Date 或 Currency 来检查各自的数据类型

下面的示例演示了如何使用 RangeValidator 控件确保输入到 Rate 文本框中的值是一个介于 1 到 10 之间的整数：

```
<asp:RangeValidator ID="RangeValidator1" runat="server"
  ControlToValidate="Rate" ErrorMessage="Enter a number between 1 and 10"
  MaximumValue="10" MinimumValue="1" Type="Integer" />
```

RegularExpressionValidator 控件

RegularExpressionValidator 控件允许检查在控件的 ValidationExpression 属性中设置的正则表达式的值。正则表达式提供了一个紧凑的语法，允许搜索文本字符串中的模式(pattern)。正则表达式是一个复杂的主题，不过幸运的是，VWD 配备了几个内置表达式，使验证电子邮件地址及邮政编码这样的值的有效性变得比较容易。如果想了解关于正则表达式的更多内容，请参见清华大学出版社引进并出版的《正则表达式入门经典》(ISBN：978-7-302-18382-2)一书。

下面的示例演示了如何使用 RegularExpressionValidator 控件确保用户输入的值是一个电子邮件地址：

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
  ControlToValidate="Email" ErrorMessage="Enter a valid e-mail address"
```



```
ValidationExpression="\w+([-+.']\w+)*@\w+([-.\]\w+)*\.\w+([-.\]\w+)*" />
```

CompareValidator 控件

CompareValidator 控件能用来比较一个控件的值与另一个控件的值。它通常用在注册表单中，用户必须输入密码两次，以确保两次输入的密码相同。也可以不与另一个控件作比较，而是与一个常量值比较。

表 9-4 列出了 CompareValidator 控件的其他属性。

表 9-4

属 性	说 明
ControlToCompare	这个属性包含验证器要与之比较的控件 ID。当设置了这个属性，ValueToCompare 就无效了
Operator	这个属性确定比较操作的类型。例如，当 Operator 设置为 Equal 时，两个控件都必须包含验证器认为有效的同一个值。类似地，还有一些选项，如 NotEqual、GreaterThan 和 GreaterThanEqual，用来执行不同的有效性验证操作
Type	这个属性确定有效性验证控件检查的数据类型。这个值可以设置为 String、Integer、Double、Date 或 Currency 来检查各自的数据类型
ValueToCompare	这个属性允许定义一个要比较的常量值。它通常用在必须输入 Yes 这样的单词的协议中，表示同意某些条件。只要将 ValueToCompare 设置为单词 Yes，并将 ControlToValidate 设置为要验证有效性的控件，就可以了。当设置了这个属性，请确保清除 ControlToCompare 属性，因为否则的话会优先采用那个属性

下面的示例演示了如何使用 CompareValidator 控件确保两个文本框中包含一样的密码：

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ControlToCompare="ConfirmPassword" ControlToValidate="Password"
    ErrorMessage="Your passwords don't match" />
```

在下面的“试一试”练习中将看到这些控件中的大部分控件的用法，除了 RangeValidator 控件。然而，它的用法与其他有效性验证控件的用法相似，因此在需要它时很容易将它添加到 Web 页面或用户控件中。

试一试

扩充联系表单

在前面的“试一试”练习中，通过创建一个包含一个表的用户控件和几个让用户输入他们的名字的控件，我们学习了联系表单的基础。在本练习中，将扩充这个表单，添加一些电子邮件地址、家庭电话号码和办公电话号码的字段；使用有效性验证控件来确保电子邮件地址是有效格式，并且至少填写了两个电话号码中的一个。为了确保用户输入正确的电子邮件地址，会要求他们输入两次。如果不喜欢这个行为，只需删除带有第二个电子邮件地址的文本框的那一行，并忽略 CompareValidator 即可。

- (1) 再次从 Controls 文件夹中打开 ContactForm.ascx 页面，并将它切换到 Design 视图中。
- (2) 在第二列中，再拖动 5 个文本框到名字文本框与 Save 按钮之间的空表单元格中。从上到下，分别将这些新控件命名为：

- EmailAddress
- ConfirmEmailAddress
- PhoneHome
- PhoneBusiness
- Comments

(3) 将 Comments 控件的 TextMode 属性设置为 MultiLine，然后在设计器中稍微增大控件的宽度和高度，以使用户添加注释。

(4) 在添加 TextBox 控件的那一行的第一个单元格中，添加图 9-5 所示的文本。

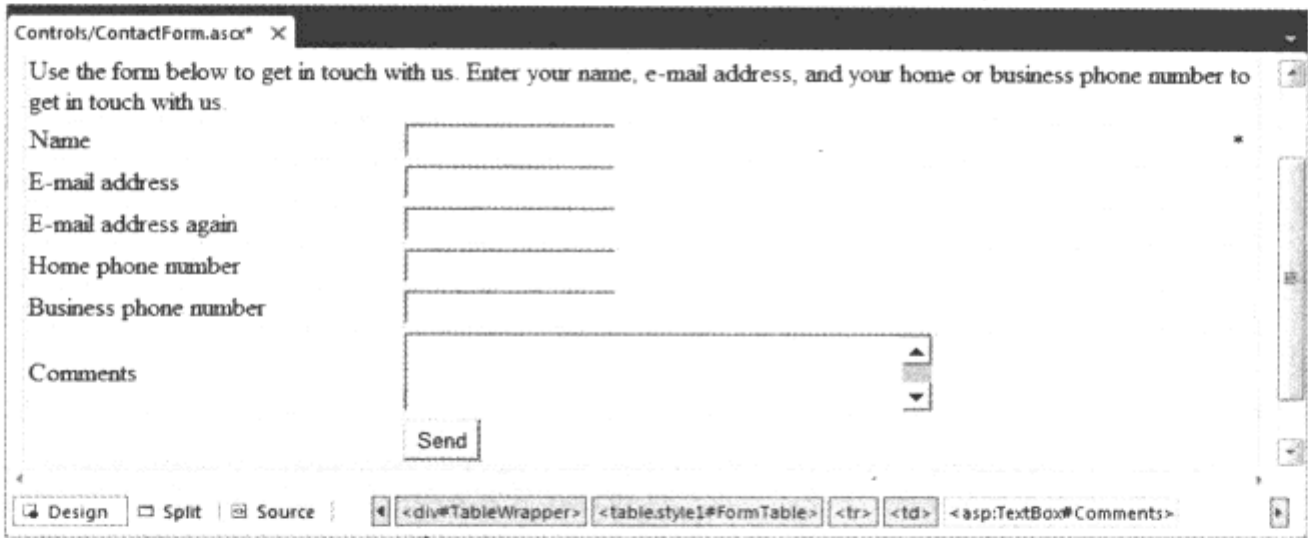


图 9-5

(5) 在第一个电子邮件地址那一行的最后一个单元格中，拖动一个 RequiredFieldValidator 和一个 RegularExpressionValidator 控件。在第二个电子邮件地址所在行的最后一个单元格中，拖动一个 RequiredFieldValidator 和一个 CompareValidator 控件。最后，在 Comments 行的最后一个单元格中，拖动一个 RequiredFieldValidator 控件。完成后，表单将如图 9-6 所示。

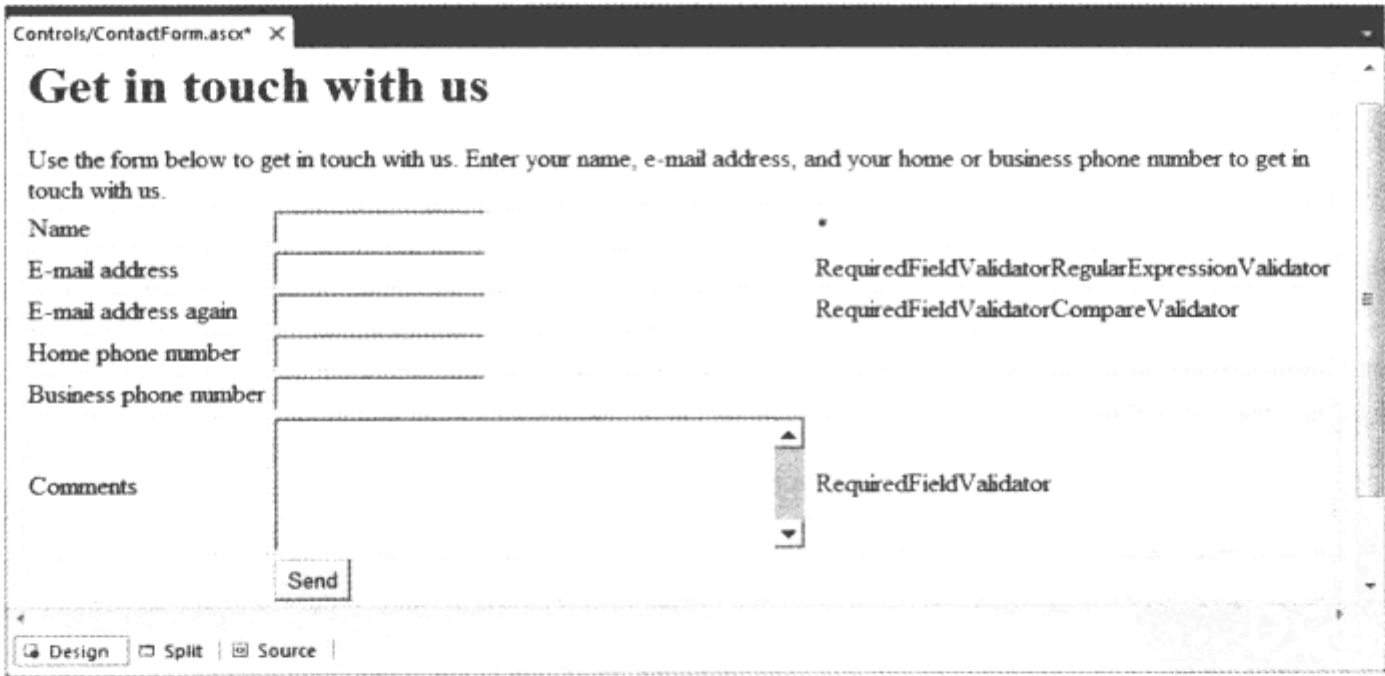


图 9-6

(6) 对于添加的每个有效性验证控件，打开 Properties 面板将 Text 属性设置为一个星号(*)，Display 属性设置为 Dynamic，CssClass 设置为 ErrorMessage。要一次性地设置所有控件，请选中第一个有

效性验证控件，然后按下 **Ctrl** 键并单击其他控件从而将它们都选中。在选中多个控件的情况下修改 **Properties** 面板的话，这些修改将应用于所有选中的控件。

(7) 然后根据表 9-5 设置控件的其他属性。

表 9-5

控 件	需要设置的属性	
RequiredFieldValidator (对第一个电子邮件地址)	ErrorMessage:	Enter an e-mail address
	ControlToValidate:	EmailAddress
RegularExpressionValidator	ErrorMessage:	Enter a valid e-mail address
	ControlToValidate	EmailAddress
RequiredFieldValidator (对第二个电子邮件地址)	ErrorMessage:	Confirm the e-mail address
	ControlToValidate:	ConfirmEmailAddress
CompareValidator	ErrorMessage:	Retype the e-mail address
	ControlToCompare:	EmailAddress
	ControlToValidate:	ConfirmEmailAddress
RequiredFieldValidator (对 Comments 字段)	ErrorMessage:	Enter a comment
	ControlToValidate:	Comments

(8) 仍然在 **Design** 视图中，单击 **RegularExpressionValidator** 一次，打开它的 **Properties** 面板并定位到 **ValidationExpression** 属性。当单击面板中的属性时，面板会显示一个省略号按钮。单击那个按钮会出现一个对话框，可以用来选择一个正则表达式，如图 9-7 所示。

(9) 单击列表中的 **Internet e-mail address**，注意 **VWD** 在 **Validation expression** 框中插入了一个长正则表达式。单击 **OK** 按钮向控件添加属性，并关闭对话框。

(10) 保存所有修改，然后在浏览器中请求 **About** 文件夹中的 **Contact.aspx** 页面。如果得到一个错误，请确保设置了前边介绍过的相关控件上所有的 **ControlToValidate** 属性。

可通过省略必需数据或输入伪数据来了解各种有效性验证控件的反应。只有当输入了所有必需字段并在两个文本框中输入了相同的电子邮件地址时，页面才会提交给服务器。在这一阶段，只会看到指示有问题的红色星号。当看到了这些有效性验证控件的工作过程后，就会知道如何用 **ValidationSummary** 控件向用户提供更多详细信息。

工作原理

与 **RequiredFieldValidator** 控件一样，其他有效性验证控件向客户端提供 **JavaScript**，当单击 **Send** 按钮或修改某个客户端控件的值时就会触发它。**CompareValidator** 控件是通过比较两个不同控件的值来工作的。只有当两个控件含有相同的值时，它才会返回真值。重要的是要意识到当文本框为空时 **CompareValidator** 控件不会触发它的有效性验证代码。因此，将其链接到一个 **RequiredFieldValidator**

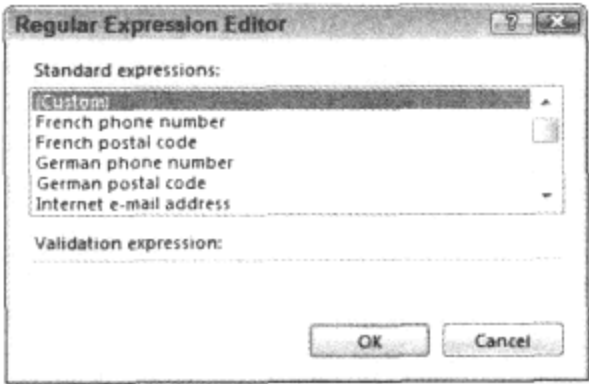


图 9-7

控件上也是非常重要的。这个控件首先确保用户至少输入了一些数据，然后 CompareValidator 控件才确保两个文本框中的内容相同。

RegularExpressionValidator 控件通过检查自己正在验证有效性的数据的模式来工作。如果观察该控件的 ValidationExpression 属性，将发现一个长的、含义不明的字符串。这个模式确保电子邮件地址中包含一些文本，后面跟着一些可选的分隔符，如短划线(-)或句点，再加上一些文本。它也确保地址中有一个@符号，后面跟着一个域名，一个句点，然后还有至少一个表示顶级域名的字符，如.com、.nl 或.co.uk。用这种表达方式，you@yourprovider.com 就会被认为是有效的电子邮件地址。因此，a@a.a 也是有效地址，而 you@you 不是有效的电子邮件地址。

注意，RegularExpressionValidator 控件只是粗略地检查电子邮件地址的语法，因此仍然很可能输入一些不存在的电子邮件地址，而它们只是看起来像有效甚至是无效的电子邮件地址，如 a@a.a。然而，在很多情况下，这种验证器足以过滤掉用户输入电子邮件地址时的错误。

到目前为止所看到的有效性验证控件很容易使用。将它们添加到页面上，设置一些属性，然后它们就会自动地做所有复杂的工作。然而，它们并不是支持可能遇到的任何有效性验证场景。例如，如果想要确保用户至少输入了两个电话号码中的一个时要怎么办？想要将用户在表单中出现的所有错误以完整的列表形式显示给用户时要怎么办？这就是用到 CustomValidator 和 ValidationSummary 控件的时候了。

5. CustomValidator 和 ValidationSummary 控件

CustomValidator 控件允许为客户端(用 JavaScript)和服务端(用 VB.NET 或 C#)编写自定义的有效性验证函数。这样在要验证有效性的数据和要应用的规则方面就有了相当大的灵活性。

ValidationSummary 控件向用户提供了它从单个有效性验证控件的 ErrorMessage 属性中检索到的一个错误列表。它能以 3 种不同的方式显示这些错误：使用一个嵌在页面中的列表、使用 JavaScript 警报框或者同时使用这两种方式。用 ShowMessageBox 和 ShowSummary 属性控制这个设置。此外，DisplayMode 属性可以修改表现错误列表的方式。默认设置是 BulletList，其中每个错误都是项目列表中的一个项，但是其他选项是 List(没有项目符号)和 SingleParagraph。

下面的“试一试”练习将指出如何编写客户端和服务端有效性验证方法，以及如何使用 ValidationSummary 控件。

试一试

编写客户端和服务端的有效性验证方法

本练习介绍了如何在页面中使用 CustomValidator 控件确保用户至少输入了两个电话号码中的一个。这种有效性验证是在客户端和服务端上发生的。此外，还将说明如何用 ValidationSummary 控件向用户提供有关他们在表单中出现的错误的反馈。

(1) 回到 VWD 中的 ContactForm.ascx 用户控件，并将它切换到 Design 视图中。右击其中含有 Button 控件的那一行(右击单元格而不是按钮)，并从上下文菜单中选择 Insert | Row Below 命令向表中插入新的一行。也可以在该行的单元格中单击以选中它然后按下 Ctrl+Alt+向下箭头键来插入一行。

(2) 选择刚刚插入的那一行的 3 个单元格，右击它们，并选择 Modify | Merge Cells 命令创建一个跨全部 3 列的单元格。

(3) 从 Toolbox 的 Validation 类别中，拖一个 ValidationSummary 控件到这个新创建的单元格中，并将其 CssClass 属性设置为 ErrorMessage。

(4) 在 Home phone number 的文本框后面的空单元格中，拖动一个 CustomValidator 控件并设置表 9-6 所示的属性：

表 9-6

属 性	值
CssClass	ErrorMessage
Display	Dynamic
ErrorMessage	Enter your home or business phone number
Text	*
ClientValidationFunction	ValidatePhoneNumbers

(5) 在 Design 视图中双击 CustomValidator 控件，让 VWD 为事件 ServerValidate 编写一个事件处理程序。并向该事件处理程序中添加下列代码：

VB.NET

```
Protected Sub CustomValidator1_ServerValidate(ByVal source As Object,
    ByVal args As System.Web.UI.WebControls.ServerValidateEventArgs) _
    Handles CustomValidator1.ServerValidate
    If Not String.IsNullOrEmpty(PhoneHome.Text) Or
        Not String.IsNullOrEmpty(PhoneBusiness.Text) Then
        args.IsValid = True
    Else
        args.IsValid = False
    End If
End Sub
```

C#

```
protected void CustomValidator1_ServerValidate(object source, ServerValidateEventArgs
args)
{
    if (!string.IsNullOrEmpty(PhoneHome.Text) ||
        !string.IsNullOrEmpty(PhoneBusiness.Text))
    {
        args.IsValid = true;
    }
    else
    {
        args.IsValid = false;
    }
}
```

(6) 切换到用户控件的 Markup 视图，并在带控件的表之前添加下列 JavaScript 代码块：

```
<script type="text/javascript">
function ValidatePhoneNumbers(source, args)
{
    var phoneHome = document.getElementById('<%= PhoneHome.ClientID %>');
    var phoneBusiness = document.getElementById('<%= PhoneBusiness.ClientID %>');
    if (phoneHome.value != '' || phoneBusiness.value != '')
```

```
    {  
        args.IsValid = true;  
    }  
    else  
    {  
        args.IsValid = false;  
    }  
}  
</script>  
<table class="style1">
```

如果发现 VWD 将起始花括号({)添加到了行的最后，而不是单独成行，则从主菜单中选择 Tools | Options 命令。然后选择 Text Editor | JScript | Formatting，在 New Lines 类别中取消对这两个选项的选择。这只是一个纯粹的个人偏好，无论这个花括号是否单独成行，JavaScript 都会正常运行。

(7) 按下 Ctrl+Shift+S 组合键保存所有修改，然后在浏览器中请求页面 Contact.aspx。注意，如果没有至少输入两个电话号码中的一个，就无法提交表单。还要注意，ValidationSummary 控件显示了一个列表，列出了关于输入表单的数据的所有问题。客户端 JavaScript 函数 ValidatePhoneNumbers 现在确保在将页面提交回服务器之前至少输入了一个电话号码。图 9-8 显示了最终出现在 Google Chrome 浏览器中的页面。

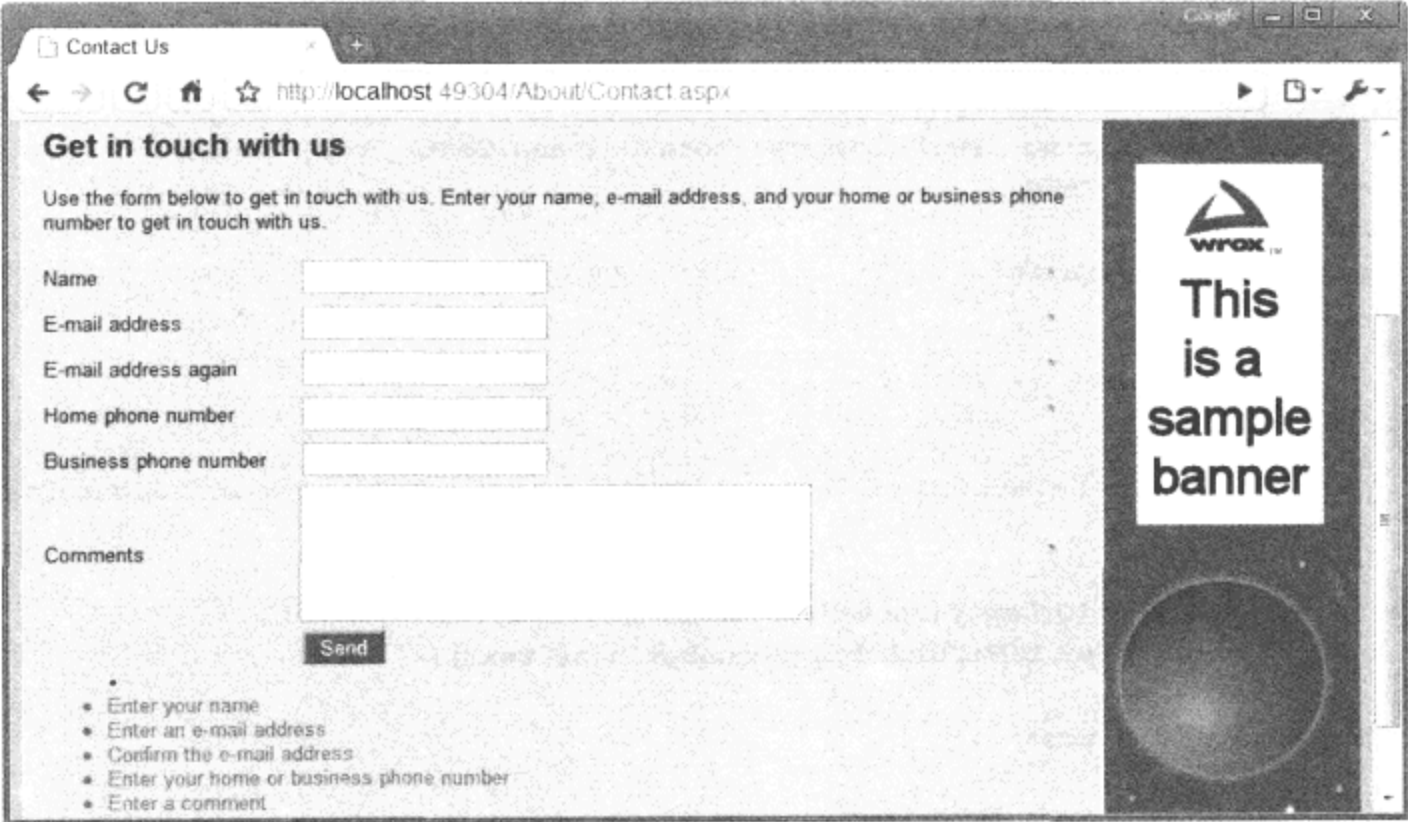


图 9-8

(8) 回到 VWD 中并在 Design 视图中单击 ValidationSummary 控件。在 Properties 面板上，将 ShowMessageBox 改为 True，ShowSummary 改为 False。(小提示：通过双击这个值，可以轻松地在 Properties 面板上的下拉列表中选择下一个项。对于 Booleans 类型的值，这意味着如果双击 False 它就会变为 True，反之亦然)。同样，将它的 HeaderText 属性设置为：“Please correct the following errors before you press the Send button:”。

(9) 再次在浏览器中打开页面，并再单击一次 Send 按钮。注意现在得到的不是含有错误的内联列表，而是一个客户端警告，如图 9-9 所示。这个错误列表出现在 ValidationSummary 的 HeaderText 之前。

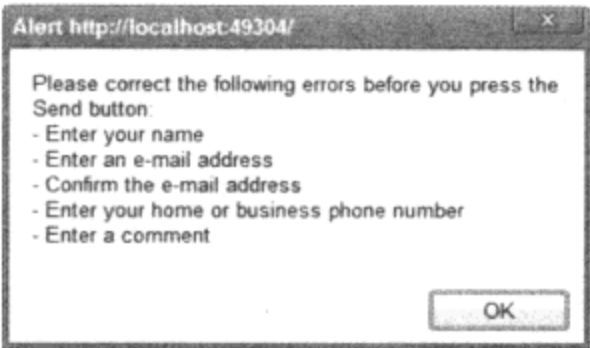


图 9-9

工作原理

当添加了 CustomValidator 控件时，就创建了两个事件处理程序：一个用于客户端，另一个用于服务器端有效性检查，在下面的代码段中两者都用了突出显示的代码：

```
<asp:CustomValidator ID="CustomValidator1" runat="server" ErrorMessage="Enter your
home or business phone number" ClientValidationFunction="ValidatePhoneNumbers"
OnServerValidate="CustomValidator1_ServerValidate"
Display="Dynamic">*</asp:CustomValidator>
```

如果使用的是 VB.NET，就看不到 OnServerValidate 特性，因为它是用 Handles 关键字建立在 Code Behind 文件中的。

当单击 Send 按钮时，在 ClientValidationFunction 中设置的 JavaScript 函数 ValidatePhoneNumbers 是在客户端触发的。这个函数是在用户控件的标记部分定义的，包含两个对电话号码文本框的引用：

```
var phoneHome = document.getElementById('<%= PhoneHome.ClientID %>');
var phoneBusiness = document.getElementById('<%= PhoneBusiness.ClientID %>');
```

注意，调用 ClientID 的代码封装在服务器端的<%= %>块中。这段代码在服务器上运行，然后将控件的 ClientID 返回客户端。如果查看浏览器中 Contact 页面的 HTML，将发现下列代码：

```
function ValidatePhoneNumbers(source, args)
{
    var phoneHome = document.getElementById('cpMainContent_ContactForm1_PhoneHome');
    var phoneBusiness =
        document.getElementById('cpMainContent_ContactForm1_PhoneBusiness');
    if (phoneHome.value != '' || phoneBusiness.value != '')
```

这里可以看到如何将控件的服务器端 ClientID 属性转换为它们的客户端 id 属性。这是比在最终 HTML 中硬编码文本框的 id 特性更好的解决方案，因为它们很容易被 ASP.NET 运行库修改。在前面的章节中已经介绍过如何以及为什么会发生这种情况。

要想让最终的 JavaScript 代码在浏览器中显示得稍短一些、可读性更好一些，可以使用第 8 章介绍的 ClientIDMode 属性来指定电话号码控件的 ID 不变。由于在一个页面中不可能有两个 ContactForm 用户控件，因而可以肯定地假设如果指定客户端控件 ID 不变的话，就不可能会有两个客户端控件有着相同的名称。为此，需要把这两个控件的 ClientIDMode 属性设置为 Static，如下所示：

```
<asp:TextBox ID="PhoneHome" runat="server" ClientIDMode="Static" />
...
```



```
<asp:TextBox ID="PhoneBusiness" runat="server" ClientIDMode="Static" />
```

由于现在指定控件 ID 值不变，因而它们在最终的 HTML 中如下所示：

```
var phoneHome = document.getElementById('PhoneHome');
var phoneBusiness = document.getElementById('PhoneBusiness');
```

由于控件有了不变的客户端 ID，因而还可以在用户控件的 JavaScript 中一起清除 ClientID 属性，并直接使用下面的代码：

```
var phoneHome = document.getElementById('PhoneHome');
var phoneBusiness = document.getElementById('PhoneBusiness');
```

这些可能很容易输入和使用，但需要付出一定代价，即如果重命名这些服务器控件中的任何一个，代码就会中断却不出现错误消息和警告。因此，在运行时最好还是使用 ClientID 属性获得控件的客户端 ID。

最后，客户端 ID 被传递给 document 对象上的 JavaScript 函数 getElementById，来获得对 JavaScript 中相应的文本框的引用。然后代码会分析这两个文本框控件的 value 属性。如果其中之一不是空字符串，则有效性验证就成功了。但是，ValidatePhoneNumbers 方法如何将有效性验证成功与否的信息反馈给有效性验证机制呢？当 ASP.NET 有效性验证机制调用 ValidatePhoneNumbers 方法时，它会传递两个参数：source(它是对 HTML 中实际 CustomValidator 的引用)和 args。args 对象会提供一个 IsValid 属性，用来判断有效性验证成功与否：

```
if (phoneHome.value != '' || phoneBusiness.value != '')
{
    args.IsValid = true;
}
else
{
    args.IsValid = false;
}
```

运行了这段代码，如果两个文本框均为空，IsValid 就会被设置为 false，因此有效性验证不成功，从而会中止提交表单。如果两个文本框中至少有一个包含值，IsValid 就会被设置为 true。在本例中，没有使用 source 参数，但是可以用它来突出显示或根据它是否有效来修改有效性验证控件。

在服务器上，CustomValidator 控件调用服务器端有效性验证方法，它执行的是同样的检查：

VB.NET

```
If Not String.IsNullOrEmpty(PhoneHome.Text)
    Or Not String.IsNullOrEmpty(PhoneBusiness.Text) Then
    args.IsValid = True
Else
    args.IsValid = False
End If
```

C#

```
if (!string.IsNullOrEmpty(PhoneHome.Text) ||
    !string.IsNullOrEmpty(PhoneBusiness.Text))
{
    args.IsValid = true;
}
```

```
    }
    else
    {
        args.IsValid = false;
    }
}
```

通过在客户端和服务端上检查数据，确保了系统只接受有效数据。即使浏览器不支持 JavaScript(可能是因为用户故意将它关闭了)，在服务器上仍然会检查数据。然而，重要的是要意识到在使用提交给服务器的数据之前仍然要检查页面的有效性。这可以通过检查页面的 IsValid 属性来实现：

```
VB.NET
If Page.IsValid Then
    ' OK to proceed
End if
```

```
C#
if (Page.IsValid)
{
    // OK to proceed
}
```

当页面中或活动 ValidationGroup 中的所有控件都有效时，IsValid 属性返回 True。通过在使用数据之前检查服务器上的 IsValid 属性，可以依据有效性验证控件确保数据的有效性，即使用户关闭了浏览器中的 JavaScript，并且没有进行任何客户端检查就将表单发送给了服务器。在本章后面讨论电子邮件的发送时将再次看到 IsValid 属性的使用。

除了目前介绍的有效性验证控件外，ASP.NET 还提供了其他的有效性验证机制，稍后将会介绍。

9.1.2 理解请求有效性验证

通过设计，每当页面上有一个控件包含看上去像 HTML 标记的内容时，ASP.NET 页面就会抛出一个异常。例如，当在联系表单中输入<h1>Hello World</h1>或<script type="text/javascript">alert('Hello World')</script>作为注释文本框的内容时，就会看到图 9-10 中所示的错误消息。

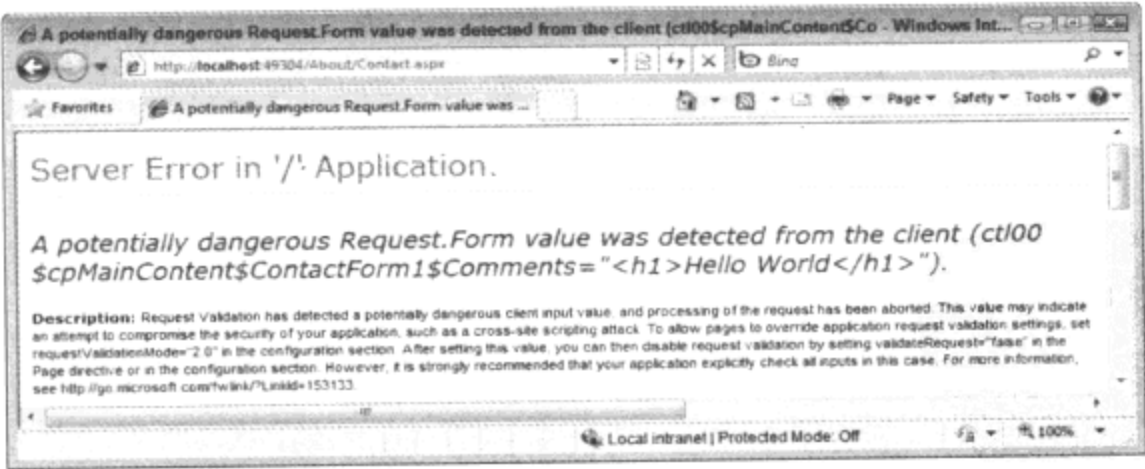


图 9-10

ASP.NET 运行库这样做的目的，是为了防止用户输入可能使 Web 站点设计或运行混乱的 HTML 标记或 JavaScript 代码。如果确定要允许用户输入 HTML，可以通过将 @ Page 指令中的 ValidateRequest 特性设置为 False 来禁用请求有效性验证：

```
<%@ Page .... Inherits="Contact" Title="Contact Us" ValidateRequest="False" %>
```

将这个设置为 False 后，用户就能输入 HTML 而不会引起错误。当要将 ValidateRequest 设置为 False 时，一定要确保确实想允许用户输入 HTML。

9.2 在服务器上处理数据

用户在 Web 窗体中输入的信息通常不是使 Web 站点成为交互式、数据驱动系统的唯一数据。在大多数 Web 站点中，也有一些来自其他数据源的信息，如数据库、文本、XML 文件和 Web 服务。此外，也有一些数据从系统中发送出去。例如，每当有人通过联系页面提交信息时，可能是想要向 Web 站点的拥有者发送一封电子邮件，或者可能想在对 Web 站点添加了一个新功能或者作了新评论时通知人们。对于这些场景，重要的是要了解 ASP.NET 4 如何允许发送电子邮件。下一节将讨论这个主题。

9.2.1 从 Web 站点中发送电子邮件

从 ASP.NET 页面发送电子邮件的代码的编写相当简单。在 System.Net.Mail 名称空间内有很多对象，使得电子邮件消息的创建与发送非常容易。这些类允许创建新消息，在 To、CC 和 Bcc 字段中添加地址，添加附件，当然还能发送消息本身。

表 9-7 描述了从 .NET 应用程序中发送电子邮件时通常要使用的 4 个类。

表 9-7

类	说 明
MailMessage	这个类表示要发送的消息。它有一些设置发送消息内容的属性，如 Subject 和 Body，设置地址的 To、CC 和 Bcc 属性，以及向消息添加文件的 Attachments 集合
MailAddress	这个类表示电子邮件中使用的发送者和接收者地址。它有几个重载的构造函数，允许设置电子邮件地址并显示名称
Attachment	这个类表示能附加到 MailMessage 中的文件。当构造一个 Attachment 实例时，可以传递要发送的文件名。然后用它的 Attachments 集合的 Add 方法向 MailMessage 添加附件
SmtpClient	这个类用来发送实际消息。在默认情况下，这个类的实例检查 web.config 文件中用于发送电子邮件的 SMTP 服务器(它使用的是简单邮件传送协议 SMTP)以及用于发送电子邮件时可选的用户名与密码等的设置

1. 配置 Web 站点发送电子邮件

虽然发送电子邮件的代码相当容易，但是配置应用程序和网络却往往需要一些技巧。用来发送电子邮件的计算机必须能访问 SMTP 服务器，不管是通过本地网络还是通过 Internet。在大多数情况下，应使用电子邮件客户端(例如 Microsoft Outlook)也使用的 SMTP 服务器。如果要将站点托管到一个外部 Web 站点上，则需要使用它们提供的 SMTP 服务器。如果不能确定 SMTP 服务器，请联系网络管理员或 ISP。

当有了 SMTP 服务器的地址后，就能在 web.config 文件的<system.net>元素中全局地配置它。当

使用的是来自 ISP 的 SMTP 服务器时，有如下配置设置：

```
<system.net>
  <mailSettings>
    <smtp deliveryMethod="Network" from="Your Name &lt;you@yourprovider.com>";">
      <network host="smtp.yourprovider.com" />
    </smtp>
  </mailSettings>
</system.net>
...
</configuration>
```

必须添加<system.net>元素作为 web.config 文件的根元素<configuration>的直接子元素。在<system.net>内添加一个包含<smtp>元素的<mailSettings>元素。最后，<network>元素就有了一个指向 SMTP 服务器的 host 特性。

<smtp>元素接受一个可选的 from 特性，用来以格式 Name <E-mail Address>设置发送者的名称和电子邮件地址。因为 XML 中的尖括号<和>字符有特殊意义，所以需要<和>转义它们。当通过编程发送电子邮件时，可以重写这个 From 地址，具体请见下一个“试一试”练习。

如果 ISP 要求在发送电子邮件之前进行身份验证或者希望使用一个不同的端口号，可以向<network />元素中添加如下信息：

```
<smtp deliveryMethod="Network">
  <network host="smtp.yourprovider.com" userName="UserName" password="Password"
    port="587" />
</smtp>
```

一些邮件服务器(如 Gmail 支持的邮件服务器)要求使用 SSL，SSL 是一种加密技术，用于加密发送到邮件服务器的数据以提高安全性能。在 ASP.NET 4 之前的版本中，必须在代码中编程激活 SSL。幸运的是，在<network />元素上有了 enableSsl 特性后，这不再是个问题。要使用 Gmail 服务器或其他任何需要使用 SSL 的邮件服务器，可以使用如下所示的<network />元素：

```
<network enableSsl="true" host="smtp.gmail.com" password="YourPassword"
  userName="YourAccountName@gmail.com" />
```

不要忘记输入用户名和密码，而在 Gmail 中则需要输入完整的 Gmail 电子邮件地址。

在开发过程中，有一种更容易的方法来处理应用程序发出的邮件，即直接将邮件放到本地磁盘的文件夹中。为此，创建一个文件夹 C:\TempMail。您需要亲自创建这个文件夹，因为它不会被自动创建。然后将<smtp />元素配置成如下所示：

```
<smtp deliveryMethod="SpecifiedPickupDirectory">
  <specifiedPickupDirectory pickupDirectoryLocation="C:\TempMail" />
</smtp>
```

在 web.config 文件中进行了这些设置后，消息并没有被发送到网络上，而是作为物理文件(带有.eml 扩展名)放在了在 pickupDirectoryLocation 特性中配置的文件夹中。可以在邮件客户端(如 Windows Vista 系统中的 Windows Mail 或是 Windows Live Mail(可以从网上下载))中读取这些文件。在网络版本上进行开发的时候，本人喜欢这种设置，因为邮件是即时到来的，而且不会打乱电子邮件账户或收件箱。

参考在线 MSDN 文档(在 <http://tinyurl.com/yz59sb4>)以了解关于<mailSettings>元素可以接受的不同设置。

2. 创建电子邮件消息

为了创建与发送电子邮件消息，需要采用 4 个步骤。首先，需要创建 MailMessage 类的一个实例。然后通过添加正文和主题来配置消息。下一步是提供消息的发送者和接收者的信息，最后需要创建 SmtpClient 类的一个实例来发送消息。下面的“试一试”练习说明了如何在代码中完成这 4 步。

试一试

发送电子邮件消息

在本练习中将在 Demos 文件夹中创建一个简单页面。这个页面中的代码创建一个电子邮件消息，当页面加载时发送。在后面的“试一试”练习中会修改联系表单，使它能通过电子邮件发送用户的响应。

- (1) 在 Demos 文件夹下面创建一个名为 Email.aspx 的新文件。确保它基于自己的基页模板，使它有正确的母版页，并自动从 BasePage 继承。将页面的 Title 改为 E-mail Demo。
- (2) 按 F7 键切换到 Code Behind 文件，并且在文件上方的类定义之前，添加下面的语句，使 System.Net.Mail 名称空间中的类对代码可用：

VB.NET

```
Imports System.Net.Mail
```

C#

```
using System.Net.Mail;
```

- (3) 向 Page_Load 处理程序中添加下列代码。如果使用的是 VB.NET，需要先在文档窗口上方用两个下拉列表来配置处理程序，或者在 Design 视图中双击页面：

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Dim myMessage As MailMessage = New MailMessage()
    myMessage.Subject = "Test Message"
    myMessage.Body = "Hello world, from Planet Wrox"
    myMessage.From = New MailAddress("you@yourprovider.com", "Sender Name")
    myMessage.To.Add(New MailAddress("you@yourprovider.com", "Receiver Name"))

    Dim mySmtpClient As SmtpClient = New SmtpClient()
    mySmtpClient.Send(myMessage)
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MailMessage myMessage = new MailMessage();
    myMessage.Subject = "Test Message";
    myMessage.Body = "Hello world, from Planet Wrox";
    myMessage.From = new MailAddress("you@yourprovider.com", "Sender Name");
    myMessage.To.Add(new MailAddress("you@yourprovider.com", "Receiver Name"));
}
```



```
SmtplibClient mySmtplibClient = new SmtplibClient();
mySmtplibClient.Send(myMessage);
}
```

修改带有电子邮件地址和姓名的两行代码，将 From 和 To 地址设置为自己的地址。如果只有一个电子邮件地址，可以让发送者和接收者使用相同的地址。

(4) 打开 web.config 文件并一直向下滚动。在结束标记</configuration>前面添加下列设置：

```
<system.net>
  <mailSettings>
    <smtp deliveryMethod="Network"
      from="Your Name <you@yourprovider.com>";
    <network host="smtp.yourprovider.com" />
  </smtp>
</mailSettings>
</system.net>
</configuration>
```

不要忘记将 smtp.yourprovider.com 改为您的 SMTP 服务器的名称。同样，一定要在 from 特性中输入您的姓名和电子邮件地址。如有必要，像前面显示的那样向<network>元素中添加 userName、password 和 port 特性。如果使用 Gmail 或另一个需要使用 SSL 的服务器来发送电子邮件，那么<network />元素应如下所示：

```
<network enableSsl="true" userName="YourAccountName@gmail.com"
  password="YourPassword" host="smtp.gmail.com" />
```

当使用了 SSL 时，检查站点中有关端口号的特定请求，典型的端口号包括 465 和 587。

(5) 保存所有修改并切换回 Email.aspx，然后在浏览器中请求该页面。稍后，应在本练习第(3)步指定的地址或本地接收文件夹中收到一封电子邮件消息。



常见错误：如果得到一个错误，则要检查一些地方。首先，确保在 web.config 中输入了正确的 SMTP 服务器。可能需要与 Internet 提供商或网络管理员交流，得到正确的地址或者得到用户名与密码。还要确保使用的邮件服务器确实允许发送消息。如果得到 “The SMTP server requires a secure connection or the client was not authenticated,” (SMTP 服务器需要一个安全连接，否则客户不能通过身份验证)这样的错误，您的提供商可能要求您登录或使用 SSL 来保障安全连接。如果是这种情况，检查 web.config 中的用户名、密码和端口号，或者尝试像以前所示的那样设置<network />元素的 enableSsl 特性。

最后，如果得到 “The specified string is not in the form required for an e-mail address,” (指定的字符串不是电子邮件地址要求的形式)这样的错误消息，则检查是否在 web.config 文件中的 from 特性中输入了一个有效的电子邮件地址。当忽略 @符号或存在其他语法错误时，便会得到这样的错误消息。

即便无法在本地计算机上发送邮件，也总是可以使用 SpecifiedPickupDirectory 发送选项来在本地计算机上存储这个文件。

工作原理

向 Code Behind 文件中添加了下面的 Imports 或 using 语句：

VB.NET

```
Imports System.Net.Mail
```

C#

```
using System.Net.Mail;
```

这个语句用来使此名称空间中的类在代码中可用，而不需要在它们前面加上它们的完整名称空间作为前缀。例如，允许创建下面这样的 MailMessage 实例：

VB.NET

```
Dim myMessage As MailMessage = New MailMessage()
```

C#

```
MailMessage myMessage = new MailMessage();
```

如果没有 Imports 或 using 语句，就需要较长的代码：

VB.NET

```
Dim myMessage As System.Net.Mail.MailMessage = New System.Net.Mail.MailMessage()
```

C#

```
System.Net.Mail.MailMessage myMessage = new System.Net.Mail.MailMessage();
```

在 Imports / using 语句后面，代码创建了一个新的 MailMessage 对象，并设置了它的 Subject 和 Body 属性。然后代码会指派电子邮件消息的发送者与接收者的地址：

VB.NET

```
myMessage.From = New MailAddress("you@yourprovider.com", "Sender Name")
myMessage.To.Add(New MailAddress("you@yourprovider.com", "Receiver Name"))
```

C#

```
myMessage.From = new MailAddress("you@yourprovider.com", "Sender Name");
myMessage.To.Add(new MailAddress("you@yourprovider.com", "Receiver Name"));
```

MailMessage 的 From 属性的类型是 MailAddress，因此可以直接指派一个新的 MailAddress。MailAddress 类的构造函数接受电子邮件地址和友好名称作为字符串，因此可以在一行代码内创建与指派 From 地址。

MailMessage 类的 To 属性是一个集合，因此不能直接指派一个 MailAddress。而是需要使用 Add 方法指派一个地址。这也允许通过调用 To.Add 方法多次来添加多个接收者，每次都在一个不同的 MailAddress 实例中传递。以类似的方式用 CC 和 Bcc 属性指派电子邮件消息的抄送(carbon copy)和密件抄送(blind carbon copy)字段的电子邮件地址。

代码的最后两行将实际消息发送出去：

VB.NET

```
Dim mySmtpClient As SmtpClient = New SmtpClient()  
mySmtpClient.Send(myMessage)
```

C#

```
SmtpClient mySmtpClient = new SmtpClient();  
mySmtpClient.Send(myMessage);
```

当调用 Send 方法时,SmtpClient 会扫描 web.config 文件寻找已配置的 SMTP 服务器或本地 Drop 文件夹。然后它会联系那个服务器并传送消息或保存在本地文件夹中。

在前面的“试一试”练习中，电子邮件消息的正文文本是硬编码的。这并不总是最好的解决方案，因为在想修改文本时就必须先扫描并修改代码。最好是使用一个基于文本的模板。下一节将进行讨论。

9.2.2 从文本文件中读取数据

.NET Framework 配置了一些方便的类与方法，使得文件的使用非常轻松。例如，位于 System.IO 名称空间中的 File 类允许从文件中读写、创建与删除文件以及到处移动文件。这个类仅包含一些静态方法，即不必先创建该类的实例。而可以直接调用 File 类上的方法。例如，要读取一个文本文件的全部内容，可以使用下面的代码：

VB.NET

```
Dim myContents As String = System.IO.File.ReadAllText("C:\MyFile.txt")
```

C#

```
string myContents = System.IO.File.ReadAllText(@"C:\MyFile.txt");
```

在本示例中，C#中的文件名用了前缀@符号，来避免使用一个额外的反斜杠(\)给每个反斜杠加上前缀。在 C#中，反斜杠有特殊意义(用于转义其他含有特殊意义的字符)，因此要在字符串中使用它，一般需要用另一个反斜杠作为它的前缀。使用@符号则告知编译器，它应将发现的每个反斜杠作为一个字面值，从而忽略这个字符的特殊意义。它还会保留字符串中所有的换行符。

表 9-8 列出了 File 类最常用的允许使用文件的方法。

表 9-8

方 法	值
AppendAllText	这个方法向一个文本文件附加一个指定字符串。如果该文件不存在，就先创建它
Copy	这个方法将一个位置的文件复制到另一个位置
Delete	这个方法从磁盘中删除指定文件
Exists	这个方法检查磁盘上是否存在指定文件
Move	这个方法将指定文件移到不同的位置
ReadAllText	这个方法用于读取文本文件的内容
WriteAllText	这个方法将字符串的内容写到一个新建文件中。如果目标文件已经存在，就会重写它

这些方法可以用于各种用途。例如，当用户上传了一个文件，可以用 Move 方法将它移到不同

的文件夹中。此外，当要清除不再需要的上传文件时，使用 Delete 方法。

ReadAllText 方法可以用来读取文本文件的全部内容。例如，当通过电子邮件发送文本时，可以将电子邮件的正文文本存储在一个文本文件中。当打算发送电子邮件时，调用 ReadAllText 方法并将这个方法返回的内容指派给电子邮件的正文。下面的“试一试”练习说明了具体的操作过程。

试一试

从 ContactForm 用户控件中发送邮件

本练习指出了如何使用电子邮件将联系表单中的用户数据发送到自己的收件箱中。作为电子邮件消息的正文，代码在包含占位符的文本文件中读取信息。这些占位符用表单中的实际用户数据填充，然后通过电子邮件发送。

(1) 首先在 Web 站点的 App_Data 文件夹中添加一个新文本文件。如果还没有 App_Data 文件夹，则在 Web 站点上右击，并选择 Add ASP.NET Folder | App_Data 命令。通过右击 App_Data 文件夹并选择 Add New Item 命令创建文本文件。然后单击文本文件并将其命名为 ContactForm.txt。

(2) 在这个文本文件中输入下列文本，包括括在一对双井号中的占位符：

```
Hi there,

A user has left the following feedback at the site:

Name:                ##Name##
E-mail address:      ##Email##
Home phone:          ##HomePhone##
Business phone:      ##BusinessPhone##
Comments:            ##Comments##
Save and close the file.
```

(3) 打开 ContactForm.ascx 用户控件的 Code Behind 文件，并在文件上方导入下面的名称空间(不包含注释)：

```
VB.NET
Imports System.IO          ' Provides access to the File class for reading the file
Imports System.Net.Mail    ' Provides access to the various mail related classes

Partial Class Controls_ContactForm
    Inherits System.Web.UI.UserControl
```

```
C#
using System.IO;           // Provides access to the File class for reading the file
using System.Net.Mail;     // Provides access to the various mail related classes

public partial class Controls_ContactForm : System.Web.UI.UserControl
```

(4) 切换到 Markup 视图，并向带几个服务器控件的表中添加 runat="server"和 id="FormTable"属性。以这种方式，当提交表单时可以编程隐藏整个表。为此，定位到起始表标记并如下修改它：

```
<table class="style1" runat="server" id="FormTable">
```

(5) 向下滚动至文件末尾，在结束标记</table>后面，添加一个名为 Message 的标签。将它的 Text 属性设置为 Message Sent。通过将 Visible 属性设置为 False 来隐藏该标签：


```
</table>
<asp:Label ID="Message" runat="server" Text="Message Sent" Visible="False" />
```

(6) 将控件切换到 Design 视图中，然后将 ValidationSummary 的 ShowSummary 设置回 True，并将 ShowMessageBox 设置为 False。然后，双击 Send 按钮。在 VWD 添加的事件处理程序内，添加下列代码：

```
VB.NET
Protected Sub SendButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles SendButton.Click
    If Page.IsValid Then
        Dim fileName As String = Server.MapPath("~/App_Data/ContactForm.txt")
        Dim mailBody As String = File.ReadAllText(fileName)

        mailBody = mailBody.Replace("##Name##", Name.Text)
        mailBody = mailBody.Replace("##Email##", EmailAddress.Text)
        mailBody = mailBody.Replace("##HomePhone##", PhoneHome.Text)
        mailBody = mailBody.Replace("##BusinessPhone##", PhoneBusiness.Text)
        mailBody = mailBody.Replace("##Comments##", Comments.Text)

        Dim myMessage As MailMessage = New MailMessage()
        myMessage.Subject = "Response from web site"
        myMessage.Body = mailBody

        myMessage.From = New MailAddress("you@yourprovider.com", "Sender Name")
        myMessage.To.Add(New MailAddress("you@yourprovider.com", "Receiver Name"))

        Dim mySmtpClient As SmtpClient = New SmtpClient()
        mySmtpClient.Send(myMessage)

        Message.Visible = True
        FormTable.Visible = False
    End If
End Sub
```

```
C#
protected void SendButton_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        string fileName = Server.MapPath("~/App_Data/ContactForm.txt");
        string mailBody = File.ReadAllText(fileName);

        mailBody = mailBody.Replace("##Name##", Name.Text);
        mailBody = mailBody.Replace("##Email##", EmailAddress.Text);
        mailBody = mailBody.Replace("##HomePhone##", PhoneHome.Text);
        mailBody = mailBody.Replace("##BusinessPhone##", PhoneBusiness.Text);
        mailBody = mailBody.Replace("##Comments##", Comments.Text);

        MailMessage myMessage = new MailMessage();
        myMessage.Subject = "Response from web site";
        myMessage.Body = mailBody;

        myMessage.From = new MailAddress("you@yourprovider.com", "Sender Name");
```

```
myMessage.To.Add(new MailAddress("you@yourprovider.com", "Receiver Name"));

SmtpClient mySmtpClient = new SmtpClient();
mySmtpClient.Send(myMessage);

Message.Visible = true;
FormTable.Visible = false;
}
}
```

再次，请确保用自己的值替换了 MailMessage 的 From 和 To 属性中的电子邮件地址。

(7) 保存所有修改，并再次在浏览器中请求 Contact.aspx 页面。输入详细信息并单击 Send 按钮。

这时将出现文本 Message Sent。

(8) 检查将电子邮件发送到的电子邮件账户，过一会儿，应收到一封电子邮件消息，类似于图 9-11 所示。

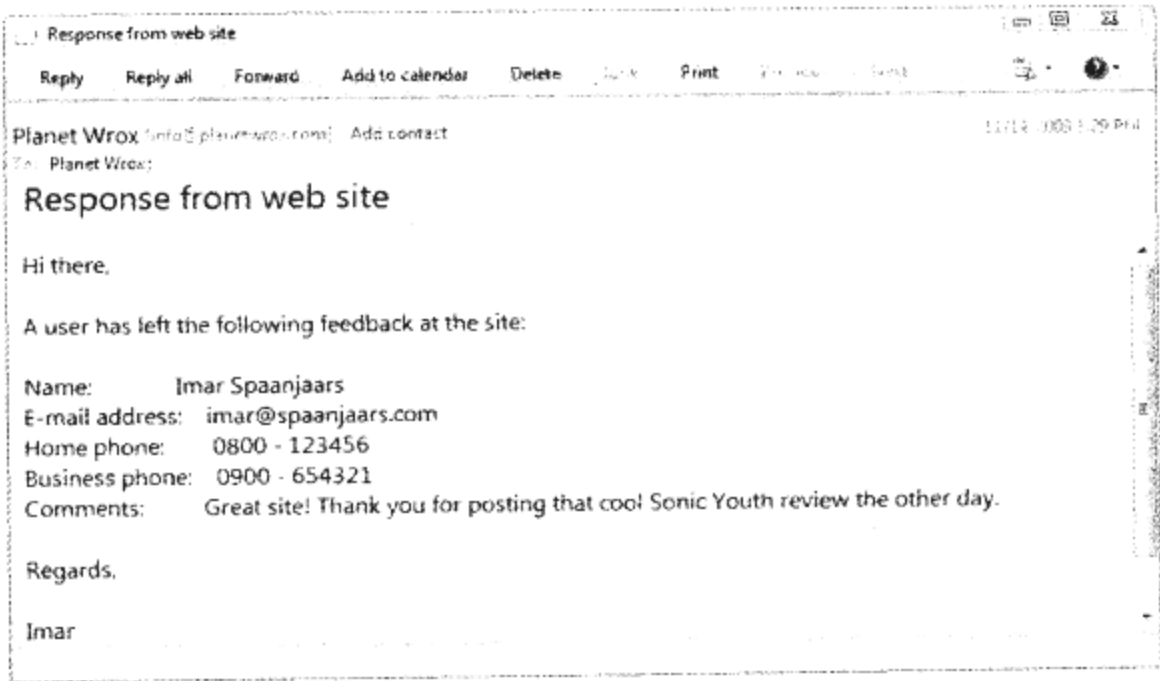


图 9-11

工作原理

本“试一试”练习的邮件发送部分与前面创建的演示页面十分类似。然而，它们的区别在于邮件消息正文文本的来源。不是在 ContactForm 控件的 Code Behind 文件中硬编码邮件正文，而是将文本移到了一个单独的文件中。这个文件包含了一些占位符，在运行时用用户的详细信息进行替换。要立刻读取整个文件，可以使用下面的代码：

VB.NET

```
Dim fileName As String = Server.MapPath("~/App_Data/ContactForm.txt")
Dim mailBody As String = File.ReadAllText(fileName)
```

C#

```
string fileName = Server.MapPath("~/App_Data/ContactForm.txt");
string mailBody = File.ReadAllText(fileName);
```

第一行使用了 Server.MapPath 将一个虚拟路径转换为它的物理对应路径。通过使用虚拟路径，

很容易将站点移动到不同的位置，因为它不依赖于任何硬编码的路径。Server.MapPath("~/App_Data/ContactForm.txt")返回一个物理路径，如 C:\BegASPNET\Site\App_Data\ContactForm.txt。然后将这个路径用在 File 类的 ReadAllText 方法中，打开文件并返回它的内容，然后将内容赋给 mailBody 变量。



注意：每次需要时就读取文件并不十分有效。在第 15 章中将介绍如何高速缓存文件的内容，这样就不必在每次请求时都读取它。

然后，代码调用 String 类的 Replace 方法许多次，用用户在联系表单中输入的详细信息替换消息正文中的静态占位符。Replace 方法的返回值(带替换字符串的新文本)将被重新赋给 mailBody 变量。在最后一次调用 Replace 方法后，mailBody 中就不再包含占位符，而是包含了用户的详细信息：

VB.NET

```
mailBody = mailBody.Replace("##Name##", Name.Text)
...
mailBody = mailBody.Replace("##Comments##", Comments.Text)
```

C#

```
mailBody = mailBody.Replace("##Name##", Name.Text);
...
mailBody = mailBody.Replace("##Comments##", Comments.Text);
```

Replace 方法是区分大小写的，因此如果发现有些占位符没有被正确地替换，请确保在代码中和邮件正文中使用了相同的大小写。

占位符括在一对双井号字符(##)之间。井号字符是任意选择的，但是有助于标识占位符，最小化不小心替换一些可能出现在实际消息中的文本的风险。

除了 Replace 方法外，还可以使用 String.Format 来格式化消息。Format 方法接受包含一个数值型占位符(包含在花括号中)和几个数值(与占位符中使用的数值有关)的字符串来替换占位符。第 10 章将介绍关于这个方法的更多知识。

一旦建立了消息正文，就将它指派给 MailMessage 对象，然后通过 SmtpClient 发送，与以前的“试一试”练习中看到的一样。

当在联系表单中填写了详细信息并单击了 Send 按钮时，可能会注意到一些页面闪烁，这是因为页面先提交给了服务器，然后用成功消息重新加载。这种页面闪烁可以用第 10 章将介绍的 Ajax 技术最小化或完全去除。

9.3 关于验证数据有效性的实用提示

下面的列表提供了关于验证数据有效性的实用提示：

- 总是要验证所有用户输入的有效性。每当要将一个 Web 站点发布到 Internet 上时，就失去了控制它的用户的能力。为了防止恶意用户向系统中输入伪数据，总是要使用 ASP.NET 的有效性验证控件验证用户输入的有效性。

- 总是在有效性验证控件中提供有用的错误消息。可以向 ErrorMessage 属性指派错误消息并让 Text 属性为空，也可以用 ValidationSummary 控件显示一个错误消息列表。
- 考虑用有效性验证控件的 CssClass 特性将错误消息的样式定义移到一个单独的 CSS 文件中，而不是直接在有效性验证控件上设置它们。
- 每当编写发送电子邮件消息的代码时，都要考虑将电子邮件的正文移到一个单独的文本文件中然后存储到 App_Data 文件夹中，因为这样会使得应用程序更加容易维护。
- 当在文本或 XML 文件中存储数据时，总是将它们存储在专门为这个目的设计的 App_Data 文件夹中。这样，所有数据文件就可以很好地打包在一起。更重要的是，默认情况下 Web 服务器会阻止访问这个文件夹中的文件，因此站点访问者不能直接请求它们。
- 当发送电子邮件进行测试时，总是将它们发送到一个现有的有效地址中。即使像 asdf@test.com 这样的看起来是无效的地址，也可能是真实存在的并且在被监视，从而导致通过电子邮件发送的敏感数据(如密码)的丢失。
- 考虑在开发过程中使用 SpecifiedPickupDirectory 作为 SMTP 邮件的 deliveryMethod。这样就不需要在整个网络上发送消息，从而得到一个快速的响应和一个更简洁的收件箱。

9.4 本章小结

用户输入是大多数交互式 Web 站点的重要方面。它来自 Web 站点的不同源：本章中创建的联系表单、查询字符串以及其他源。为了防止用户向系统中输入无效数据甚至是危险内容，在使用数据前验证所有输入的有效性是很重要的。

ASP.NET 4 配备的有效性验证控件最大的好处是它们工作在客户端和服务端上，允许创建响应表单，使用户可以立即得到他们产生的错误的反馈，而不需要进行完整的回发。同时，数据在服务端上进行验证，可确保来自不支持 JavaScript 的客户端的数据也是有效的。

为了存储用户提交给站点的信息，有两种方法。数据可以存储在数据库或文本文件中，或者通过电子邮件发送。后一种方法对于联系表单特别有用，因此当有人在 Web 站点中写了评论时，可以立即得到警报。发送电子邮件是使用 System.Net.Mail 名称空间中的类来实现的。这些类允许创建电子邮件消息，添加主题、正文、发送者和接收者信息，然后通过 SmtpClient 类发送消息。

9.5 练习

1. 为了让 ContactForm.ascx 用户控件更可重用，可以在其上创建一个字符串属性，如 PageDescription，以允许设置使用该控件的页面的名称。然后将这个字符串添加到该页面的控件声明中。最后，可以将该描述添加到所发送消息的主题中。这样，就能看到是从哪个页面中调用了联系表单。需要写什么代码实现上述内容呢？
2. 为什么在处理数据时检查 Page 的 IsValid 属性的值是如此重要呢？如果忘记进行这种检查会发生什么情况？
3. MailMessage 类的 To 与 From 属性之间的行为有什么区别？
4. 当使用一个 CustomValidator 控件时，可以在客户端和服务端上编写有效性验证代码。如何告

知 ASP.NET 运行库在有效性验证处理期间调用什么客户端有效性验证方法？

5. 如何把 CustomValidator 例程中有效性验证的成功或失败情况告知有效性验证机制呢？
练习的答案见附录 A。

本章要点回顾

客户端有效性验证	在客户端浏览器中进行的有效性验证。主要是对客户负责并提供快速反馈
File 类	包含允许使用文件的方法，包括读写文本文件
Format 方法	String 类的一个方法，用于使用其他值替换字符串中的数值型占位符
正则表达式	一种紧凑灵活的语法，用于在其他字符串中查找文本字符串
Replace 方法	String 类的一个方法，用于使用另一个值替换字符串中的某个值
服务器端有效性验证	在服务器端进行的有效性验证。当可以避开客户端有效性验证时，总是需要使用服务器端有效性验证来保护数据
SMTP 服务器	负责接受和发送电子邮件的服务器
SSL	在两台计算机之间加密(从而可以保护)数据流的技术
System.Net.Mail 名称空间	用于电子邮件类(如 MialMessage、MailAddress 和 SmtplibClient)的名称空间
有效性验证控件	一组 ASP.NET 服务器控件，可以在客户端和服务器端验证用户输入的数据的有效性

本章要点

- 使用 UpdatePanel 控件避免页面闪烁
- 了解启用 Ajax 功能的 ScriptManager 控件
- 使用 UpdateProgress 控件通知用户 Ajax 操作的进程
- 创建客户端脚本可以访问的 Web 服务和页面方法
- 使用客户端 ASP.NET AJAX Library

毫无疑问，过去几年 Web 开发中最大的炒作就是 Ajax。虽然驱动 Ajax 的技术已经出现一段时间了，但直到 2005 年初它才有了正式名称。Ajax(Asynchronous JavaScript And XML)允许客户端 Web 页面通过异步调用与服务器交换数据。由 Ajax 驱动的最普遍的功能可能是无闪烁页面，它允许执行到服务器的回发，而无需刷新整个页面。

要使用 Ajax 功能增强 Web 站点，可以选择不同的 Ajax 架构。其中许多架构都能提供一组功能和工具，包括用于在浏览器和服务器端激活 Ajax 的客户端 JavaScript 架构。虽然 ASP.NET 有许多不同的 Ajax 架构可用，但最著名的一个是 Microsoft ASP.NET AJAX，因为它被完全集成在 .NET 4 Framework 和 VWD 2010 中。ASP.NET AJAX 的一个很酷的特性是，它与其他客户端架构(包括 jQuery，第 11 章将介绍它)具有很好的互操作性，因此您可以不必仅仅局限于 ASP.NET AJAX。

Microsoft ASP.NET AJAX 不只提供无闪烁回发。除了实现无闪烁页面的控件之外，Microsoft ASP.NET AJAX 还提供了更多的服务器控件来创建富交互式的且有响应的用户界面。

除了基于服务器控件的部分之外，ASP.NET AJAX Framework 还配备了富客户端架构。这个架构使得 JavaScript 能够与服务器通信，还允许使用同样运行的直观代码模型访问整个客户端页面，不管针对哪个浏览器。

到本章结束时，应当全面理解 ASP.NET AJAX Framework 提供的各种服务器控件。另外，还应该基本理解在 ASP.NET 世界中创建 Web 服务和页面方法的方式，以及使用客户端 AJAX Framework 从客户端 JavaScript 代码中调用它们的方法。

10.1 AJAX 简介

本书第 1 章介绍了浏览器如何与服务器进行交互。浏览器使用 GET 或 POST 方法(第 4 章和第 9 章已介绍)请求页面。服务器处理该页面，并回发产生的 HTML 代码。之后，浏览器解析该 HTML 代码并将页面呈现给用户，并有选择地下载任意的资源，如图像、脚本文件和 CSS 样式表。当用户之后与页面交互时(例如，通过单击按钮来提交一个已填好信息的联系表单)，页面被回发给服务器，之后浏览器中会再次加载整个页面。图 10-1 中左图演示了上述过程。

尽管上述模型已经在 Web 页面中使用很多年了，但是它仍然存在一些大的缺点。首先，因为整个页面是在回发后被加载的，因此发送到浏览器的 HTML 代码量要远大于浏览器所需要的。回想一下在第 9 章创建的联系表单。用户刚提交完联系表单，服务器就显示了一个带有文本 Message Sent 的 Label 控件，这是通过完全加载一个隐藏了表单控件的新页面并显示上述文本消息实现的。即使页面的剩余部分没有改变(菜单、侧边栏、页脚等)，也仍然需要将它们从服务器发送至客户端。理想情况下，您可能只希望发回那些发生了改动的 HTML 代码。就联系表单而言，需要发送的可能与文本 Message Sent 一样少。图 10-1 中右图展示了上述过程。服务器没有将整个页面作为一个响应来发送，而是发送了一个部分响应(只包含文本 Message Sent 的内容)，之后浏览器使用该响应来更新页面中发生了变化的部分，并保留页面的剩余部分。

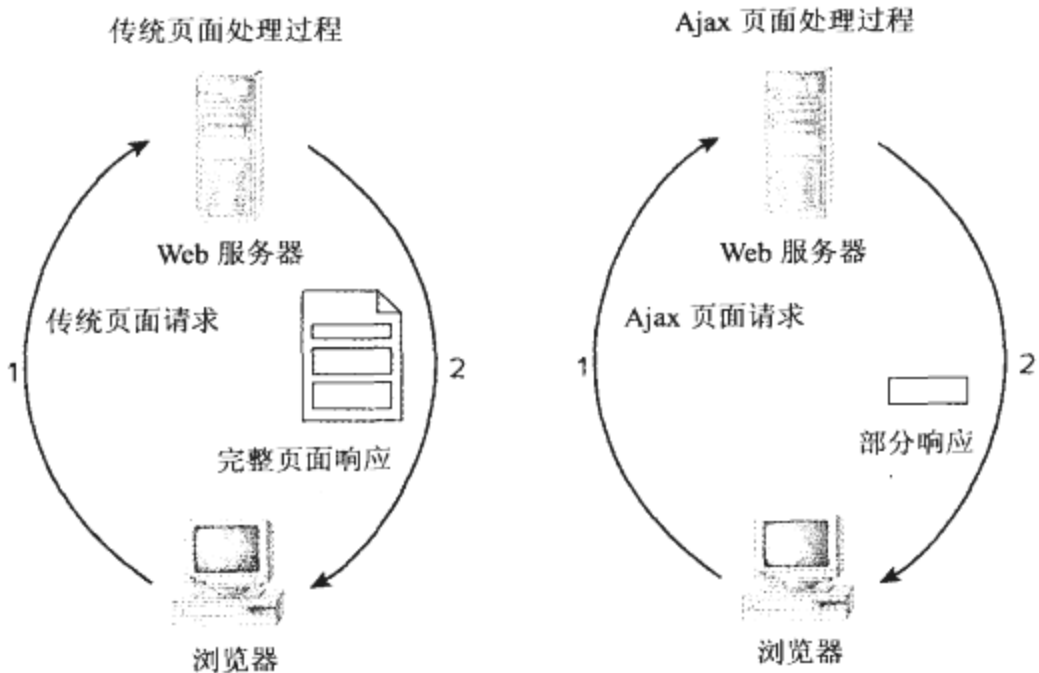


图 10-1

加载整个页面的第二个缺点与浏览器显示页面的方式有关。由于整个页面被替换掉，因此浏览器不得不关闭旧的页面，再打开新的页面。这样做会使页面“闪烁”，从而使页面失去对用户的吸引力。可以使用 Ajax 技术解决上述两个问题，本章后面将会介绍。

Ajax 涉及的概念已经讨论了许多年。Internet Explorer 5 及之后版本的 IE 浏览器都带有 XMLHttpRequest 对象，允许使用 JavaScript 与服务器进行通话，从而发送和接收数据。但是，人们还在使用其他技术来仿真现在被称为 Ajax 的行为，包括 Macromedia Flash、iframe 元素或隐藏框架。

然而，当引入术语 Ajax 之后它才被广泛使用。为了走在时代前列，Microsoft 开始构建 ASP.NET AJAX——现在已经完全集成到了 ASP.NET 和 VWD 2010 中的 Ajax Framework。这种架构提供了许

多优势，作为 Web 开发人员，可以利用它们创建有响应的应用程序。

特别是，通过 ASP.NET AJAX 可以：

- 创建无闪烁页面，它们允许刷新部分页面，而不需要全部重载，也不会影响页面的其他部分。
- 在这些页面刷新过程中给用户反馈。
- 更新部分页面，使用计时器按计划调用服务器端的代码。
- 访问服务器端 Web 服务和页面方法，使用它们返回的数据。
- 使用富客户端编程架构访问和修改页面中的元素，访问代码模型和典型系统(与.NET Framework 的系统类似)。

ASP.NET AJAX 包括两个重要的部分：ASP.NET AJAX 服务器控件和客户端 ASP.NET AJAX Library。在本章剩余部分介绍如何使用 ASP.NET AJAX Framework 创建富交互式 Web 应用程序时会用到它们。注意 Ajax 本身是一个宽泛的对象，仅用一章不可能全面地讨论它。如果想更深入地学习 ASP.NET AJAX，请参考清华大学出版社引进并出版的《ASP.NET 3.5 AJAX 高级编程》一书(ISBN: 978-7-302-21358-1)。虽然本书讨论的是 Ajax 之前的版本，但书中的许多概念现在仍然适用。

ASP.NET AJAX 的优点是它很容易上手。创建无闪烁的页面就是将一些控件从 Toolbox 中拖到页面上。在理解了 Ajax Framework 的基础之后，就可以继续探讨更高级的主题，例如调用 Web 服务和使用富客户端架构与页面交互来扩充自己的知识。

10.2 在项目中使用 ASP.NET AJAX

ASP.NET AJAX 完全集成到了 ASP.NET 和 VWD 中，这意味着可以立刻开始使用它。对于在 VWD 中创建的每个新的 ASP.NET 4 Web 项目来说，它们的 Ajax 功能都已经被激活了。此外，Toolbox 中包含一个 AJAX Extensions 类别，该类别中包含许多要在页面中使用的与 Ajax 相关的控件。VWD 还对 ASP.NET AJAX 提供了大力支持，它为服务器端的控件和客户端的 JavaScript 代码(编写该代码是为了与客户端页面和在服务器端运行的代码进行交互)提供了 IntelliSense。

10.2.1 创建无闪烁页面

为了避免 ASPX 页面中的完整回发并且只更新部分页面，可以使用 UpdatePanel 服务器控件。要让这个控件正确运行，还需要一个 ScriptManager 控件。如果打算在多个 ASPX 页面中使用 Ajax 功能，可以将 ScriptManager 控件放置在母版页中，因此它在基于这个母版页的所有页面中都可用。每个页面只能包含一个 ScriptManager 控件，因而如果已经添加了一个到母版页，就不能再添加另外一个到内容页。要从内容页中访问在母版页内定义的 ScriptManager 控件，可以使用稍后介绍的 ScriptManagerProxy。在 Toolbox 的 AJAX Extensions 类别中可以找到这些以及其他与 Ajax 相关的服务器控件，如图 10-2 所示。

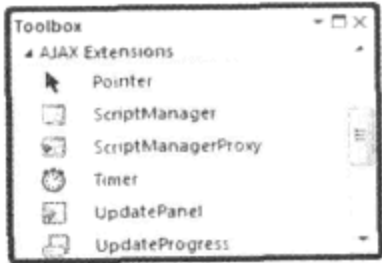


图 10-2

下面两节将介绍 UpdatePanel 和 ScriptManager 控件。之后介绍如何在 Planet Wrox Web 站点的页面中使用这些控件。后面几节将介绍 UpdateProgress、Timer 和 ScriptManagerProxy 控件。

1. UpdatePanel 控件

UpdatePanel 控件是创建无闪烁页面的关键组件。在它最基础的应用中，只要用控件包装要更新的内容，并将 ScriptManager 控件添加到页面就行了。当 UpdatePanel 内的某个控件产生到服务器的回发时，只会刷新 UpdatePanel 里面的内容。

为了说明 UpdatePanel 控件解决的问题和它在客户端页面中的行为，下面的“试一试”练习提供了使用该面板避免回发过程中页面闪烁的简单示例。

试一试 给页面添加 UpdatePanel 控件

在本练习中，要将 Label 和 Button 控件添加到页面。当单击浏览器中的按钮时，Label 控件的 Text 属性会更新为服务器的当前日期和时间。要避免通常与回发有关的页面闪烁，可以将控件包装在 UpdatePanel 中来观察该控件是如何影响页面行为的。

- (1) 打开 VWD 中的 Planet Wrox 项目。
- (2) 在 Demos 文件夹中，使用定制模板创建一个名为 UpdatePanel.aspx 的新 Web 窗体。给页面提供 UpdatePanel Demo 的 Title。
- (3) 将新页面切换到 Design 视图窗口，并将 Label 和 Button 控件从 Toolbox 拖入 cpMainContent 占位符中。如果 ContentPlaceHolder 变得与 Label 一样小，请在 Label 上释放 Button。这样 Button 就放在了 Label 前面，但如果再次将 Label 拖到 Button 的上方，两者就会改变位置。
- (4) 使用 Properties 面板清除 Label 控件的 Text 属性。要做到这一点，请右击 Properties 面板中的 Text 属性，然后选择 Reset。
- (5) 双击 Design 视图窗口中页面的灰色和只读区域，为其 Load 事件建立处理程序，并将下列代码添加到 VWD 添加的处理程序中：

```
VB.NET
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Label1.Text = System.DateTime.Now.ToString()
End Sub
```

```
C#
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = System.DateTime.Now.ToString();
}
```

(6) 保存所有修改，并按下 Ctrl+F5 组合键在浏览器中打开页面。Label 显示当前日期和时间。多次单击 Button 控件。注意每次单击这个按钮时，页面就会闪烁然后刷新，并显示更新后的日期和时间。现在来查看浏览器使用的 HTML 代码(右击浏览器中的页面，并选择 View Source 或 View Page Source)。注意页面包含一个服务器发送来的附带有日期和时间的 span 元素。

(7) 关闭浏览器，回到 VWD，并将页面 UpdatePanel.aspx 切换到 Markup 视图中。在 Label 控件前保留一定的空间，然后输入 updatepanel 并按下 Tab 键。VWD 会在 UpdatePanel 内插入代码，并插入<ContentTemplate>元素。

(8) 接下来，剪切结束标记</ContentTemplate>和</UpdatePanel>，并将它们粘贴到第(3)步中创建

的按钮下面。应该使用下面的标记结束(但是控件可能缺失 ID 特性):

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:Label ID="Label1" runat="server"></asp:Label>
    <asp:Button ID="Button1" runat="server" Text="Button" />
  </ContentTemplate>
</asp:UpdatePanel>
```

(9) 在 UpdatePanel 的起始标记之前, 请从工具箱的 AJAX Extensions 类别中拖放一个 ScriptManager 控件。还有一种方法是输入 sm, 然后按下 Tab 键, 使用代码段来插入 ScriptManager 控件。代码如下所示(尽管 ScriptManager 控件缺少 ID 特性, 并且在使用代码段时会使用一个自结束元素):

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" runat="Server">
  <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
  <asp:UpdatePanel ID="UpdatePanel1" runat="server">
```

(10) 保存修改, 然后再次在浏览器中请求页面。多次单击该按钮, 将标签更新为当前的日期和时间。注意现在没有页面闪烁。似乎只是更新了页面上的标签而已。如果再次在浏览器中查看源代码, 将会看到包含第一个请求的日期和时间的 span 元素。通过单击按钮所添加的对标签的更新并不是 HTML 源代码的一部分, 因为 Ajax 框架会自动将其添加到浏览器的内部 HTML 代码中。

工作原理

通过将内容封装在 UpdatePanel 中, 可以在页面上(要刷新这个页面而又不影响整个页面)定义一个区域。在前面的示例中, UpdatePanel 内的 Button 控件会产生一个回发, 然后刷新这个定义了控件的区域。它只刷新封装在 UpdatePanel 内的那部分页面, 而不是替换掉整个页面, 从而实现页面的无闪烁加载。

如果对从服务器发送到浏览器的数据进行分析(可使用网络分析工具, 如 Fiddler, 该工具可以从 <http://fiddlertool.com> 上下载), 则会发现只有非常有限的数据被发送到了客户端, 而不是发送了整个页面(大约 12KB)。只有如下数据被发送:

```
1|#||4|233|updatePanel|cpMainContent_ctl00|
<span id="cpMainContent_Label1">11/14/2009 8:46:21 PM</span>
<input type="submit" name="ctl00$cpMainContent$Button1" value="Button"
id="cpMainContent_Button1" class="MyButton" style="background-color:#7A70A4;" />
|0|hiddenField|__EVENTTARGET||0|hiddenField|__EVENTARGUMENT||0|hiddenField|__LASTFOCU
S||1600|hiddenField|__VIEWSTATE|/wEPDwUJMzU5Mjc3OTcxNDQWAmYPZBYCAgMPZBYKAgUPPCsADQI
...
asyncPostBackControlIDs||0|postBackControlIDs||46|updatePanelIDs||tctl00$cpMainCont
ent$ctl00,cpMainContent_ctl00|0|childUpdatePanelIDs||45|panelsToRefreshIDs||ctl00$cp
MainContent$ctl00,cpMainContent_ctl00|2|asyncPostBackTimeout||90|16|formAction||Updat
ePanel.aspx|16|pageTitle||UpdatePanel
Demo|119|scriptBlock|ScriptPath|/ScriptResource.axd?d=p4-mCo3_3Zdf8FVdrAYB5ByMKQUtQlr
0TxvHwcFKKzRKu5tGLKlOgeOQvo0myHomX3hZm2cb3pq-S-yKeS2I5w2&t=el96b05|
```

注意, 这里删除了一大块内容, 包括中间部分用三个句点表示的页面的许多视图状态, 以节省一些空间。如果查看这个响应, 便能看出更新后的 Label 和 Button 控件的 HTML 代码。这两个控件定义在 UpdatePanel 控件的<ContentTemplate>元素内。ASP.NET AJAX 使用剩余文本来维持页面的状

态(使用 `_VIEWSTATE` 字段)和了解页面中放置响应的位置。尽管仍需要加载很多数据，但是相比整个页面(大约 12KB)来说，所加载的内容已经很少了。这样可以带来更快的响应以及更好的用户体验。

在步骤(10)中，当查看浏览器中页面的源代码时，您会注意到页面包含的仍然是最初的源代码，而不是 Ajax 框架修改的更新后的源代码。这使得在某些时候创建、测试和调试 Ajax 应用程序会变得很困难，因为无法真正地看到发送至浏览器的数据。幸运的是，许多工具可用来帮助解决这个问题。除了前面提到的 Fiddler 工具外，还建议您看一下 Microsoft Internet Explorer Developer 工具栏。Internet Explorer 8 就带有该工具栏，可以通过 Tools | Developer Tools 菜单选项访问它。对于更早版本的 Internet Explorer 来说，可以从 <http://tinyurl.com/IEDevBar> 上下载独立的安装程序。

用于调试的另一个很好的工具是 Firebug，它可以与 Firefox 浏览器很好地集成，可以从 <http://getfirebug.com> 上下载 Firebug。

在这个“试一试”练习中，使用了两个重要的 AJAX Extensions 控件。本练习中直接放在 `UpdatePanel.aspx` 页面内的 `ScriptManager` 控件，是正确运行大多数 Ajax 功能的必要条件。该控件是客户端页面与 Ajax 框架之间的桥梁，负责完成像正确注册在浏览器中使用的 JavaScript 文件这样的任务。然后使用 `UpdatePanel` 定义希望更新的区域，而不需要重新加载整个页面。下面几节将更详细地讨论这两个控件。

2. 进一步研究 UpdatePanel 控件

当单击按钮时，只更新了页面上的 `UpdatePanel` 控件及其内容(如前面的“试一试”练习所述)。这是 `UpdatePanel` 的默认行为，其中在 `<ContentTemplate>` 元素内定义的其他服务器控件只刷新它内部的内容。然而，`UpdatePanel` 能做的还不只这些，下一节将会介绍。

UpdatePanel 控件的常见属性

表 10-1 列出了 `UpdatePanel` 控件的一些重要属性，它们能够影响 `UpdatePanel` 的行为。

表 10-1

属 性	说 明
ChildrenAsTriggers	这个属性确定位于 <code>UpdatePanel</code> 内的控件能否刷新 <code>UpdatePanel</code> 。其默认值是 <code>True</code> ，如前面的“试一试”练习所示。当将这个值设置为 <code>False</code> 时，必须将 <code>UpdateMode</code> 设置为 <code>Conditional</code> 。注意，当设置为 <code>False</code> 时， <code>UpdatePanel</code> 内定义的控件仍然会引发到服务器的回发；它们只是不再自动更新面板而已
Triggers	<code>Triggers</code> 集合包含 <code>PostBackTrigger</code> 和 <code>AsyncPostBackTrigger</code> 元素。如果要想实现完整的页面刷新，那么就可以用第一个；而如果要使用在面板之外定义的控件更新 <code>UpdatePanel</code> ，那么第二个就很有用
RenderMode	这个属性可以设置为 <code>Block</code> 或 <code>Inline</code> 来表示 <code>UpdatePanel</code> 将它自己呈现为 <code><div></code> 元素还是 <code></code> 元素
UpdateMode	这个属性确定总是刷新控件(<code>UpdateMode</code> 设置为 <code>Always</code>)还是只在某些条件下刷新控件，例如，当 <code><Triggers></code> 元素内定义的某个控件引发回发时(<code>UpdateMode</code> 设置为 <code>Conditional</code>)
ContentTemplate	尽管在 <code>UpdatePanel</code> 的 <code>Properties</code> 面板中不可见，但 <code><ContentTemplate></code> 是 <code>UpdatePanel</code> 的一个重要属性。它是一个容器，可以将控件放置在该容器里面作为 <code>UpdatePanel</code> 的子控件。如果忘记了这个必需的 <code>ContentTemplate</code> 属性，VWD 会发出一条警告

在本章后面的练习中会介绍 UpdatePanel 的更多内容。

UpdatePanel 警告

UpdatePanel 看似(确实)很有用，其实使用它的代价很高。虽然 UpdatePanel 只有在刷新部分页面时才出现，但是整个页面(及其所有的表单数据)仍然要回发给服务器。在服务器上，页面仍旧要经历其正常的生命周期，然后发回用来更新页面的 HTML 代码。然而，发回的数据采用的并非是最佳的数据格式，因为该数据包含一些系统开销数据(ASP.NET AJAX 需要的，用来理解如何解释该数据)。这意味着 UpdatePanel 会带有一些与表单 post、页面处理及网络流量有关的开销数据。本章后面将介绍一些通过客户端代码从服务器端获取数据以及向服务器端上传数据的方法，这些方法能够最小化上述数据开销。

如前面的“试一试”练习所述，UpdatePanel 控件能够刷新部分页面。在 UpdatePanel 之内或在它之外定义的控件可以刷新 UpdatePanel。然而，为了正常运行，UpdatePanel 还需要一个管理客户端 JavaScript 的 ScriptManager 控件。

3. ScriptManager 控件

ScriptManager 控件是客户端页面和服务器之间的桥梁。它管理脚本资源(客户端使用的 JavaScript 文件)，负责部分页面的更新(如前所述)，处理与 Web 站点的交互，例如 Web 服务和 ASP.NET 应用程序服务(如成员、角色和配置文件)。第 16 章和第 17 章将从服务器的角度进一步讨论这些服务。

如果认为只在一小部分页面上需要 Ajax 性能，那么通常可以将 ScriptManager 控件直接放置到内容页中。在前面的“试一试”练习中，已经简单了解了其工作原理。但是，也可以将 ScriptManager 控件放置在母版页中，这样它便在整个站点中都可用。本章稍后的“试一试”练习中将这样做。

ScriptManager 类有许多属性，其中大多数都用于高级场景。在很多情况下，例如刚刚看到的使用 UpdatePanel 更新部分页面，不需要改变 ScriptManager 类的任何属性。而在有些情况下，需要改变或设置它的某些属性。表 10-2 列出了 ScriptManager 控件的一些常见属性。

表 10-2

属 性	说 明
AllowCustomErrorsRedirect	这个属性确定 Ajax 运行过程中出现的错误是否会导致加载自定义的错误页面。默认是 True；设置为 False 时，错误在浏览器中显示为 JavaScript 通知窗口，或者在禁止调试时对客户端隐藏。注意，如果没有配置任何自定义错误页面，错误就总是显示为 JavaScript 通知，而不管这个设置的值是什么。第 18 章将详细讨论设置自定义错误页面和调试应用程序
AsyncPostBackErrorMessage	如果没有使用自定义错误页面，这个属性允许自定义错误消息，当发生 Ajax 错误时，用户可以看到这条错误消息。它允许对用户隐藏脏细节，并给他们提供更友好的错误消息
EnablePageMethods	这个属性确定是否允许客户端代码调用页面内定义的方法。后面将讨论其工作原理
EnablePartialRendering	这个属性确定 ScriptManager 是否支持使用 UpdatePanel 控件呈现部分页面。除非想阻止整个页面的部分更新，否则应该将它设置为 True

(续表)

属 性	说 明
EnableCdn	若该属性设置为 True，ASP.NET 将会包含微软的 Content Delivery Network 站点上(而不是自己的服务器上)的客户端框架文件的链接。这样可以节省一些带宽，并且如果用户已经从使用这些文件的其他站点那里获取了这些文件的高速缓存副本的话，这样做还可能会提高页面首次加载时的速度
MicrosoftAjaxMode	这个属性可用于确定是否包含 Microsoft AJAX 客户端库。该属性允许使用 ScriptManager 控件完成与服务器相关的任务(如注册客户端脚本)，而不需要在页面中嵌入客户端框架
Scripts	ScriptManager 控件的<Scripts>子元素允许添加客户端在运行时必须下载的其他 JavaScript 文件
CompositeScript	和<Scripts>元素一样，<CompositeScript>元素也允许添加其他的 JavaScript 文件。但是，在<CompositeScript>元素下注册的文件都被合并为一个单独的、可下载的文件，从而可以减小网络开销并提高页面的性能
Services	<Services>元素允许定义客户端页面能够访问的 Web 服务。本章第二部分将介绍如何使用 Web 服务

虽然使用 UpdatePanel 和 ScriptManager 已经足以创建无闪烁页面，但 ASP.NET AJAX 提供了更多控件来增强用户在启用了 Ajax 的 Web 站点中的体验。改进用户体验的方法之一是使用 UpdateProgress 控件，本章稍后将会讨论。另一种选择是使用 Timer 控件，也在本章稍后讨论。

10.2.2 给用户提供反馈

尽管回发通常导致视觉问题，但它们也有一大优点：用户可以看到正在发生的事情。UpdatePanel 使这变得有点困难。在正在发生的情况已经发生之前，用户没有视觉线索。因此要告诉用户在处理他们的请求时再等待几秒种，可以使用 UpdateProgress 控件。

UpdateProgress 控件

可以使用 AssociatedUpdatePanelID 属性将 UpdateProgress 控件连接到 UpdatePanel 上。当关联的 UpdatePanel 控件忙于刷新时，就会显示它在<ProgressTemplate>元素中定义的内容。通常要在模板中放入像“Please wait”这样的文本或动画图像(也接受其他标记)来让用户知道正在发生的事情。

除了 AssociatedUpdatePanelID 和<ProgressTemplate>属性之外，UpdateProgress 控件还提供了表 10-3 所示的常用属性。

表 10-3

属 性	说 明
DisplayAfter	该属性确定控件在显示其内容之前等待的时间(以毫秒为单位)。当刷新时间非常短暂以至于通知消息过多时，这非常有用。默认值是 500 毫秒，也就是半秒钟
DynamicLayout	该属性确定控件隐藏时是否占用屏幕空间。它直接映射到前面见过的 CSS display: none;或者 visibility: hidden;属性

在下面的“试一试”练习中，将学习如何结合 UpdatePanel、ScriptManager 和 UpdateProgress 让

ContactForm 用户控件无闪烁。

试一试

无闪烁页面——将它放在一起

在本练习中，要修改以前创建的用户控件 ContactForm.ascx，并将整个控件包装在一个 UpdatePanel 中，因此当输入消息并单击 Send 按钮时，页面不会执行完整回发。为了帮助用户理解发送消息时页面是忙碌的，可以给控件添加一个 UpdateProgress 面板。在这个控件内部可以放置一个 GIF 动画图像(可以从本书的下载代码中得到它)。同样，也可以登录 www.ajaxload.info 创建自己的动画图像。

(1) 在 Markup 视图下，从 Controls 文件夹中打开用户控件 ContactForm.ascx，然后使用 <ContentTemplate> 将整个 <table> 元素和控件底部的 Label 包装到 UpdatePanel 中。直接在 Markup 视图中输入代码，使用一个代码段，或者从工具箱中拖动控件，都可以完成这项任务。请确保将 UpdatePanel 的 ID 设置为了 UpdatePanel1。最终应获得以下代码：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <table class="style1" runat="server" id="FormTable">
      ....
    </table>
    <asp:Label ID="Message" runat="server" Text="Message Sent" Visible="false" />
  </ContentTemplate>
</asp:UpdatePanel>
```

(2) 保存对控件的修改，然后打开 MasterPages 文件夹中的 Frontend.master 文件。在 PageWrapper 的起始标记 <form> 和 <div> 标记之间添加一个 ScriptManager 控件，方法是将它从 Toolbox 拖动到页面的源代码中。最终应获得如下代码：

```
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
    <div id="PageWrapper">
```

(3) 保存对母版页的修改，然后关闭它。

(4) 打开在前面“试一试”练习中创建的 UpdatePanel.aspx 页面，并删除 ScriptManager 元素。由于在母版页中声明了这个控件，因此不能在基于这个母版页的页面内重新定义它。保存修改，并关闭页面。

(5) 在浏览器中打开 About 文件夹中的 Contact.aspx 页面，然后填充联系表单。注意，只要单击 Send 按钮，这个表单就会消失，并由说明消息已发送的标签所取代。和前面的示例一样，当页面重载和显示文本“Message Sent”时，页面没有闪烁。

(6) 为了让用户在消息传递给邮件服务器时保持同步更新，应当给页面添加一个 UpdateProgress 控件。在这个控件内部，可以添加一个动画图像和一些文本，以告知用户消息正在发送。要添加图像，请使用 Windows 资源管理器找到文件夹 C:\BegASPNET\Resources，从该文件夹中可以获取与本书同步的文件。打开 Chapter 10 文件夹，然后打开 Monochrome 文件夹。将 PleaseWait.gif 文件从 Windows 资源管理器拖动到 App_Themes 下 Monochrome 主题的 Images 文件夹中。重复这一过程，

将 PleaseWait.gif 文件从 DarkGrey 文件夹拖动到其各自主题的 Images 文件夹中。图 10-3 显示了这两个图像的最终状态。

(7) 打开 Monochrome.css 文件，一直向下滚动至末尾，添加下面的规则：

```
.PleaseWait
{
    height: 32px;
    width: 500px;
    background-image: url(Images/PleaseWait.gif);
    background-repeat: no-repeat;
    padding-left: 40px;
    line-height: 32px;
}
```



图 10-3

- (8) 将完全相同的规则复制到 DarkGrey 主题的 DarkGrey.css 文件中。
- (9) 切换回 ContactForm.ascx 用户控件，并在文件末尾 UpdatePanel 的结束标记下，从 Toolbox 的 AJAX Extensions 类别中拖动一个 UpdateProgress 控件。将它的 AssociatedUpdatePanelID 设置为 UpdatePanel1， UpdatePanel 的 ID 之前已在页面中定义。
- (10) 在<UpdateProgress>标记之间创建一个<ProgressTemplate>，在这个模板中，创建一个<div>元素，并将它的 class 特性设置为 PleaseWait，该特性是在第(7)步中创建的 CSS 类。在<div>元素中，输入一些文本告诉用户等待一会儿。应该得到下面的代码：

```
</asp:UpdatePanel>
<asp:UpdateProgress ID="UpdateProgress1" runat="server"
    AssociatedUpdatePanelID="UpdatePanel1">
    <ProgressTemplate>
        <div class="PleaseWait">
            Please Wait...
        </div>
    </ProgressTemplate>
</asp:UpdateProgress>
```

(11) 要模拟发送消息时的长时间延时(因此可以看见 UpdateProgress 控件)，可以将下面的代码行添加到控件的 Code Behind 文件中，就在以发送电子邮件的方法改变控件可视性的代码行之后。

```
VB.NET
Message.Visible = True
FormTable.Visible = False
System.Threading.Thread.Sleep(5000)

C#
Message.Visible = true;
FormTable.Visible = false;
System.Threading.Thread.Sleep(5000);
```

(12) 保存所有修改，并再次打开 About 文件夹中的 Contact.aspx 页面。填充所需的细节，单击 Send 按钮。在按下这个按钮之后不久，就会看到 UpdateProgress 控件以下面的形式显示文本和动画图像，如图 10-4 所示。不久之后， UpdateProgress 和整个表单将会消失，并得到 Message Sent 文本。

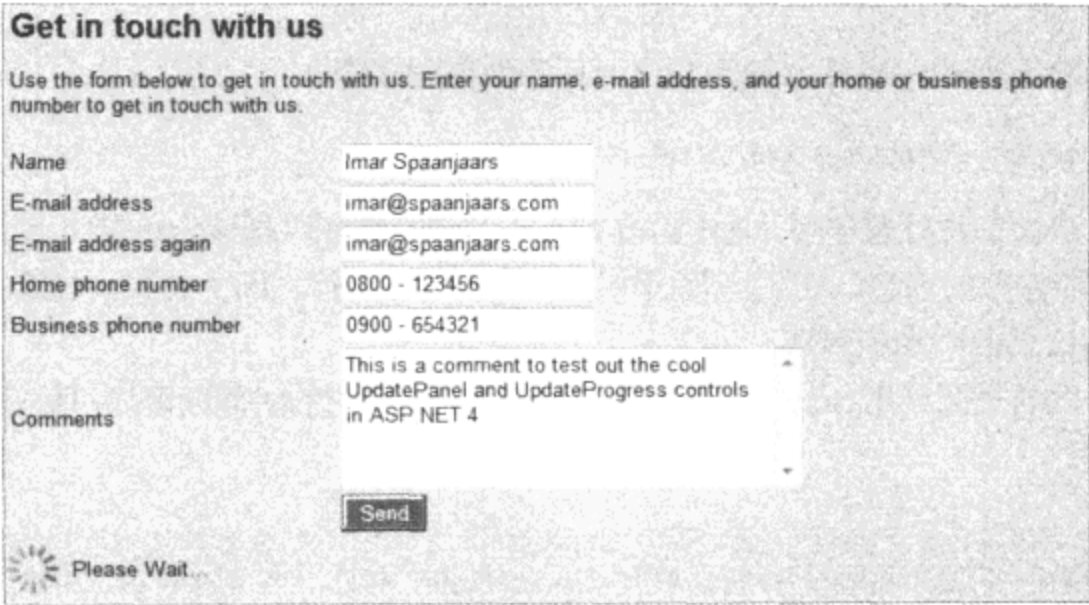



图 10-4



常见错误：如果没有看到上面描述的行为，则浏览器可能使用的是过时的 CSS 文件版本。可以按 Ctrl+F5 组合键或 Ctrl+R 组合键从服务器获取最新版本。还有一种选择，就是清空浏览器的高速缓存。

工作原理

使用用户控件中的 UpdatePanel，在 UpdatePanel 的 ContentTemplate 内的所有内容都会依据回发更新，同时又不会影响页面的其他部分。这样，就可以使用服务器控件隐藏表单，并使用 Message Sent 标签取代它，而不会导致任何页面闪烁。

要告知用户他(她)的消息正在发送，也可以给站点添加一个 UpdateProgress 控件。在默认情况下，当附加给 Ajax UpdatePanel 的刷新时间超过 500 毫秒(半秒)时，就会显示这个控件。该控件的 <ProgressTemplate>元素包含一个简单的<div>元素，其 class 设置为 PleaseWait。可以将下面的 CSS 规则添加到这些主题的两个 CSS 文件中：

```
.PleaseWait
{
    height: 32px;
    width: 500px;
    background-image: url(Images/PleaseWait.gif);
    background-repeat: no-repeat;
    padding-left: 40px;
    line-height: 32px;
}
```

这个代码段首先将 Update 消息的大小设置为 500 像素宽、32 像素高。这足以覆盖内容块的宽度，并为更长的消息提供了足够的空间。

然后代码添加了动画图像作为背景图像。为了防止在背景中重复使用图像，repeat 属性应该设置为 no-repeat。然后将左内边距属性设置为 40 像素。这将<div>中的文本向右移动，让它紧邻动画图像。最后，将文本的 line-height 设置为 32 像素，这个高度与整个<div>相同。这样便可以将<div>

元素内的整个文本块垂直居中，并使其与动画图像排列在一起。

最后，可以将下面的代码行添加到发送消息的处理程序中：

```
System.Threading.Thread.Sleep(5000);
```

这个代码将页面的执行暂停 5 秒(传递给 Sleep 方法的数字以毫秒为单位)，因此可以仔细检查 UpdateProgress 控件中的消息。在生产代码中，应该删除这一行，因为它极大地降低了页面的执行速度，而又没有给页面添加任何值。

除了用户触发的页面使用 Send 按钮更新之外，还能以指定的时间间隔通过编程触发页面刷新，如下面一节所述。



注意：将服务器端的功能封装到 UpdatePanel 中时，可能有时会很难发现是否产生了错误以及错误消息的内容。例如，当发送电子邮件失败时，无法看到真正的错误消息，因为该消息被隐藏在了 JavaScript 中。为了使一出现错误就能够很容易地看到该错误消息，可以从页面中暂时删除 UpdatePanel，或者使用服务器端注释标记`<%--和--%>`取消其结束和起始标记的注释。

```
%--<asp:UpdatePanel ID="up1" runat="server"><ContentTemplate>--%>
... Existing content goes here
<%--</ContentTemplate></asp:UpdatePanel>--%>
```

10.2.3 使用 Timer 控件

Toolbox 的 AJAX Extensions 类别中的 Timer 控件对于重复执行服务器端代码非常有用。例如，可以使用它每 5 秒钟更新一次 UpdatePanel 的内容。UpdatePanel 的内容的来源各不相同，例如论坛上最新发布的帖子的数据库或新闻站点上的新闻条目，带有可用于浏览器中滚动广告的信息的 XML 文件，某股票 Web 服务提供的股票行情等。

Timer 控件的用法非常简单。该控件按照指定的时间间隔激活其 Tick 事件。在这个事件的事件处理程序内，可以执行认为合适的任何代码。下面的代码段演示了简单的 UpdatePanel 和 Timer 控件的标记，该标记可以放在基于母版页的内容页中(因为母版页已经包含了所需要的 ScriptManager 控件)：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate>
    <asp:Label ID="Label1" runat="server"></asp:Label>
    <asp:Timer ID="Timer1" runat="server" Interval="5000" OnTick="Timer1_Tick" />
  </ContentTemplate>
</asp:UpdatePanel>
```



注意：注意当使用 VB.NET 时，不需要 Timer 控件上的 OnTick 处理程序，因为在 VB.NET 中是使用 Code Behind 文件中的 Handles 关键字来处理 Tick 事件的。

当 Timer 控件“做记号”时，便会激活 Tick 事件，可以使用如下代码处理：

VB.NET

```
Protected Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Timer1.Tick
    Label1.Text = System.DateTime.Now.ToString()
End Sub
```

C#

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    Label1.Text = System.DateTime.Now.ToString();
}
```

在浏览器中运行该代码时，每隔 5 秒就会更新一次标签当前的日期和时间。如果希望使该操作更慢或更快，则需要调整其 Interval 属性，该属性以毫秒为单位来指定时间。

使用自动更新面板的场景以及使用按钮单击刷新内容的能力非常普遍。自动刷新面板给用户提供了来自服务器的最新日期信息的非插入方法。同时，如果用户想在他们选择的任意时刻刷新数据，只要单击这个按钮就行了。从编码的角度来看，它们没有什么不同。Timer 的 Tick 事件处理程序和 Button 的 Click 事件处理程序调用的是相同的代码(更适合封装到一个单独的方法中)。

关于 Timer 控件的更多信息，可以参看 <http://tinyurl.com/TimerClass> 上的 MSDN 文档。

本书已经对 ASP.NET AJAX Framework 必须提供的大多数重要的服务器端控件进行了介绍。接下来将讨论如何在支持 Ajax 的 Web 页面中使用 Web 服务和页面方法，并介绍客户端 JavaScript 框架。在讨论 Web 服务和客户端框架的过程中，还将介绍如何使用 ScriptManagerProxy，它是 Toolbox 中 Ajax Extensions 类别下的最后一个控件。

10.3 在 Ajax Web 站点中使用 Web 服务和页面方法

从启用了 Ajax 的 ASP.NET Web 站点中调用 Web 服务和页面方法的能力是对 Web 开发工具箱的极大补充。能够调用 Web 服务或页面方法意味着可以从客户端代码中更容易地访问服务器端的数据，从而为全面回发提供了另一种选择。下一节将讨论 Web 服务，之后的一节还将会深入探讨 ASP.NET 的页面方法。

在能够在自己的应用程序中创建和使用 Web 服务之前，需要重点了解 Web 服务的定义，以及如何在 ASP.NET 项目中定义它。

10.3.1 Web 服务的定义

从本质上说，Web 服务是可以在 Internet 上调用并能够随意地将数据返回调用代码的方法。这使它们非常适用于在不同系统间交换数据。因为 Web 服务建立在一致和易于理解的标准之上，所以它们能够很方便地在不同类型的平台和系统之间交换数据。例如，使用 Web 服务就很容易在 Microsoft Windows 上运行的 ASP.NET Web 站点和在 Linux 上运行的基于 PHP 的站点之间交换数据。同时，它也可以在 ASP.NET Web 站点和使用 JavaScript 的客户端浏览器之间交换数据。

在 Planet Wrox 项目中，Web 服务只用于让浏览器中的客户端页面与服务器会话和交换数据。

因此在这个站点中，服务器和客户端都在相同的 Web 项目中——一个在客户端执行(调用 Web 服务器的 JavaScript)，另一个驻留在服务器端(Web 服务本身)。从安全的角度来看，这是最简单的解决方案，因为两者彼此相互信任。

如果要想让客户端页面与不同域上的 Web 服务会话，就需要在浏览器中设置安全性以允许这个操作。另外，也可以使用 Web 服务让两个服务器或其他应用程序(例如桌面应用程序)彼此通信。在这种情况下，一个应用程序(Windows 桌面应用程序、PHP 或传统 ASP Web 应用程序甚至是另一个 ASP.NET Web 服务)与网络上的 ASP.NET Web 服务交互并交换数据。然而这两种情况都超出了本章讨论的范围。

在 Planet Wrox 项目中创建的 Web 服务与已经创建的普通方法类似。不同之处是该 Web 服务需要使用 WebMethod 特性装饰每个方法。其功能就像小标记或者可以粘贴到代码元素上的标签(例如方法、属性等)，它将某块代码标记得很特别，因此与特性代码交互的其他代码可以看见特性代码中包含的特性，并依据这些信息进行决策。不必过于担心这一点，因为使用 Web 服务时不需要亲自读取这些特性。所需做的就是将这个特性粘贴在方法上，将它转换为 Web 方法。例如，要将返回字符串的标准方法转换为 Web 方法，可以应用下面的特性：

VB.NET

```
<WebMethod()>
Public Function HelloWorld(ByVal yourName As String) As String
    Return String.Format("Hello {0}", yourName)
End Function
```

C#

```
[WebMethod]
public string HelloWorld(string yourName)
{
    return string.Format("Hello {0}", yourName);
}
```

在 C#中，使用方括号括起特性；而在 VB.NET 中，则使用尖括号括起特性。还可能会遇到下面这样的例子：为了满足 VB.NET 之前版本的需要，而在 VB.NET 特性后添加一个空格和一个下划线。即便使用下划线非常有效，也要尽量避免使用它。

正确使用这个特性，就可以告知 ASP.NET 运行库：确实希望提供该方法作为能够从客户端代码调用的 Web 方法。这也允许在同一个类中创建不会自动提供为 Web 服务的其他方法，从而允许灵活地决定为外部世界开放什么资源。

除了将方法标记为 Web 方法的 WebMethod 特性之外，通常将该方法保存在扩展名为.asmx 的文件中，放置在从 System.Web.Services.WebService 继承的类中。下一节将介绍其工作原理。

10.3.2 创建 Web 服务

使用 VWD 创建 Web 服务非常简单。与其他所有文档类型一样，VWD 也附带有 Web 服务模板。可以使用 Add New Item 对话框给站点添加 Web 服务。然后可以修改服务，并在 Web 浏览器中使用 ASP.NET 运行库自动创建的标准测试页面测试它。当 Web 服务正确运行时，可以从客户端 JavaScript 代码中调用它，如下面的“试一试”练习所述。

试一试

创建 Web 服务

本练习中将创建一个简单的名为“Hello World”的 Web 服务。该服务接受您的名字作为输入参数，并返回友好的个性化问候。对于这个 Web 服务而言，没有太多实际的用法。然而，由于该服务本身非常简单，因此更容易将重点放在基本概念上。

(1) 在站点的根文件夹下创建一个名为 WebServices 的新文件夹，将站点内的所有 Web 服务分组到一个文件夹中。这不是必需的，但是却有助于组织站点。

(2) 接下来，右击该新文件夹，选择 Add New Item 选项。单击 Web Service 选项(不要误选 WCF Service，否则之后创建的会是一个不同类型的服务)，确保选择了喜欢的编程语言和 Place Code in Separate File 选项，命名该服务为 NameService，如图 10-5 所示。

(3) 单击 Add 按钮将服务添加到站点。注意将 .asmx 文件添加到 WebServices 文件夹的同时也要将 Code Behind 文件(.vb 或.cs)放入站点的 App_Code 文件夹，如图 10-6 所示。

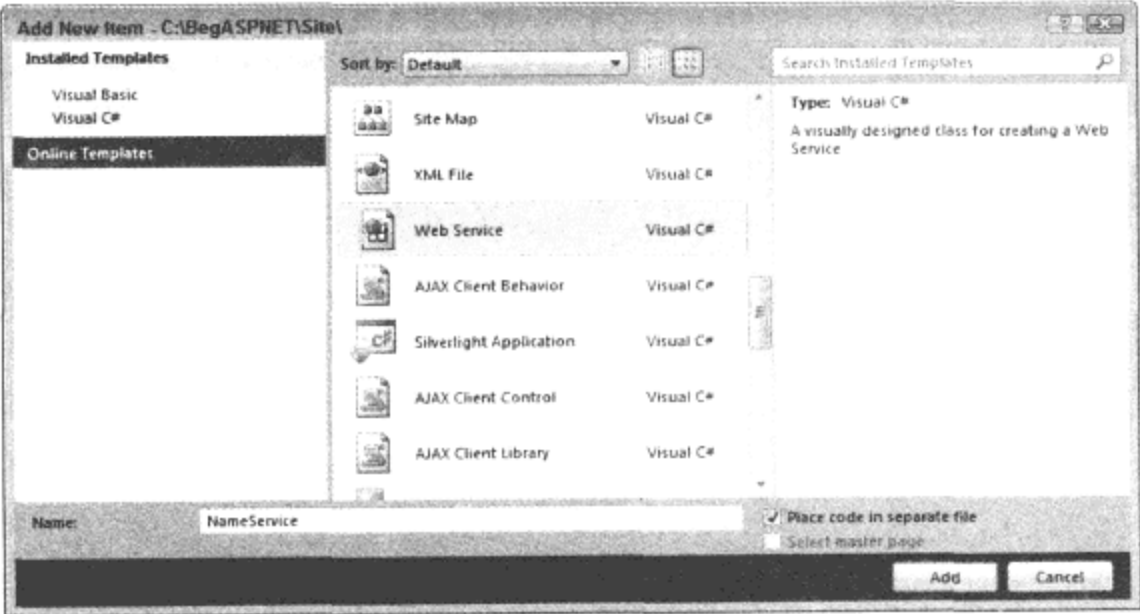


图 10-5

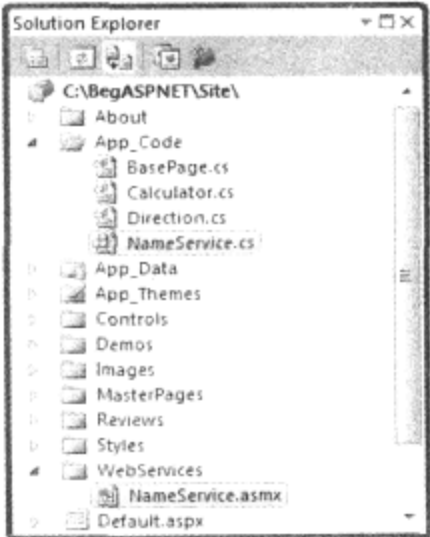


图 10-6

(4) 打开 App_Code 文件夹中的 NameService Code Behind 文件并修改 HelloWorld 方法的代码，让它接受字符串并返回个性化问候。应该得到如下代码：

```
VB.NET
<WebMethod()>
Public Function HelloWorld(ByVal yourName As String) As String
    Return String.Format("Hello {0}", yourName)
End Function

C#
[WebMethod]
public string HelloWorld(string yourName)
{
    return string.Format("Hello {0}", yourName);
}
```

(5) 保存所有修改，右击 Solution Explorer 中的 NameService.asmx 文件，选择 View in Browser 选项。浏览器完成加载之后，就会得到一个页面，它列出了在 NameService.asmx 服务中定义的所有

公共 Web 服务。在这个练习中，只能看到 HelloWorld，如图 10-7 所示。

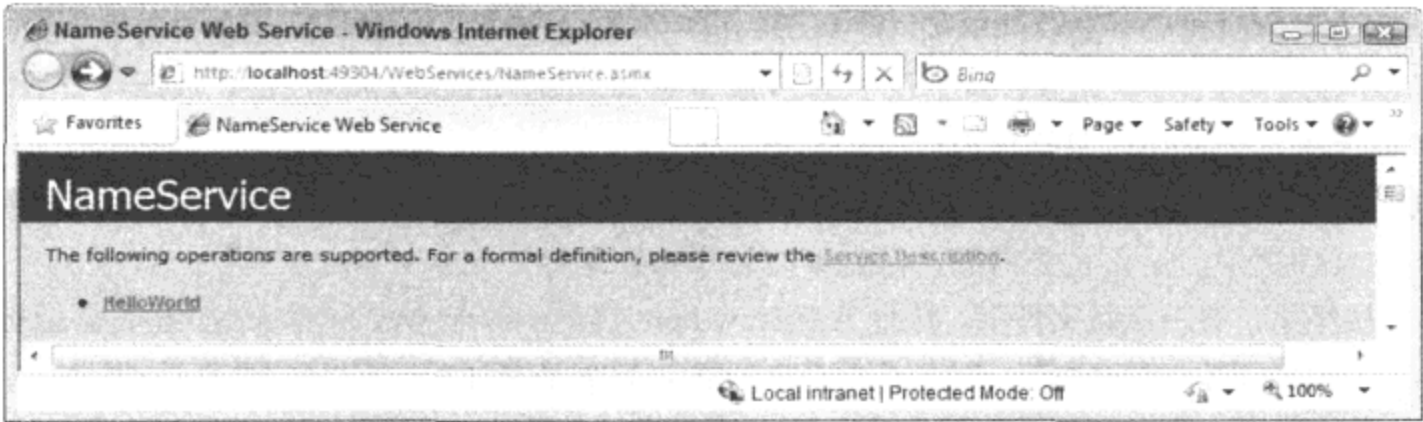


图 10-7

(6) 单击 HelloWorld 链接，就会登录到能够测试该服务的页面。在 yourName 字段内输入您的名字，然后单击 Invoke 按钮。打开一个新窗口(如图 10-8 所示)，它显示了由 Web 服务返回的 XML。



图 10-8

工作原理

Web 服务实质上是能够在网络(例如 Internet 或局域网)上调用的方法。它们能够让应用程序彼此通信和交换数据。其基本消息格式是 XML，如图 10-8 所示，它显示了 HelloWorld 方法的结果。

为项目添加 Web 服务时，并不是文件中的所有方法都会自动变成 Web 可调用的方法。要将方法提供为服务，还需要应用 WebMethod 特性。

VB.NET

```
<WebMethod()>
Public Function HelloWorld(ByVal yourName As String) As String
```

C#

```
[WebMethod]
public string HelloWorld(string yourName)
```

使用这个特性，外部系统就可以看见这个方法，因此外部系统可以访问它。在提供服务的计算机的浏览器中打开.asmx 文件时，将自动获得一个测试页面，可以用来测试服务。在 HelloWorld 服务中，提交名字、单击 Invoke 按钮将名字作为参数发送给服务。服务的响应就是将您的名字添加到欢迎消息中，然后使用 String.Format 将它作为字符串返回：

VB.NET

```
Public Function HelloWorld(ByVal yourName As String) As String
    Return String.Format("Hello {0}", yourName)
```

End Function

```
C#
public string HelloWorld(string yourName)
{
    return string.Format("Hello {0}", yourName);
}
```

正如第 9 章所介绍的，`String.Format` 方法使用字符串，该字符串中包含括在一对花括号({})内的数字占位符。对于每个数值，提供一个字符串值作为后续参数。在前面的示例中只有一个占位符，但是可以使用多个参数轻松地扩充对 `Format` 方法的调用。例如，如果要用姓和名字格式化字符串，最好使用下面的代码：

```
VB.NET
Return String.Format("Hello {0} {1}", firstName, lastName)
```

```
C#
return string.Format("Hello {0} {1}", firstName, lastName);
```

`String.Format` 方法让字符串更易于理解。不是使用&或+杂乱地串联字符串，只需要在字符串中定义占位符，然后在运行时提供值就行了。

最后，Web 服务方法返回欢迎消息字符串。Web 服务运行库负责将返回值发送给调用代码；本例中的测试页面将返回值显示为原始的 XML 字符串。之后其他的框架和编程语言就可以解析并使用这个 XML 字符串，就像本章后面将要介绍的 ASP.NET AJAX 使用 JavaScript 一样。

虽然这只是一个简单示例，但这里的概念也适用于复杂的 Web 服务(它们可以交换复杂的敏感数据，而不只是交换简单字符串)。很明显，测试页面只用来测试服务是否正确运行。在现实的 Web 服务中，数据通常由其他代码使用，例如 Web 应用程序或客户端 JavaScript。下一节将介绍后者如何运行。

10.3.3 在 Ajax Web 站点中使用 Web 服务

在 ASP.NET AJAX 之前，从客户端浏览器中调用 Web 服务和使用它们返回的数据需要写许多代码；特别是想让它所有主流浏览器(例如 IE 和 Firefox)中运行时。幸运的是，Ajax Framework 隐藏了所有复杂操作和需要使用 Web 服务的代码。需要做的就是将特性添加到 Web 服务中，将它标记为脚本可以调用的服务。然后在 `ScriptManager` 控件中注册服务，并写几行 JavaScript 来调用服务和接收它的返回值。这只对在自己的 Web 站点内定义的服务有用，如下所述。如果要调用不在同一个域上的服务作为调用它们的页面，还需要写其他代码。这超出了本书的范围，但清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C#和 VB.NET(第 7 版)》一书(ISBN: 978-7-302-23524-8)详细讨论了调用外部 Web 服务的方法。

下一节将介绍配置 Web 服务以便客户端脚本可以调用它们的方法。在下面的“试一试”练习中，将介绍如何使用这些知识以及如何从客户端页面中调用 Web 服务。

1. 配置 Web 服务

前面已经介绍过如何通过添加特性将方法标记为 Web 方法。这样可以将方法提供给外部世界。

要让客户端脚本也能看见 Web 服务，需要将一个特性添加到服务类中。如果查看 App_Code 文件夹中的 NameService 类，就会发现模板中已经添加了这个特性，但却让它以注释的形式存在。

VB.NET

```
' <System.Web.Script.Services.ScriptService()> _
```

C#

```
// [System.Web.Script.Services.ScriptService]
```

删除注释标记，将整个服务提供为客户端脚本服务。

2. 配置 ScriptManager 控件

从本章前面一节可以知道，ScriptManager 控件几乎是所有与 Ajax 相关的操作中必不可少的组件。它注册客户端 JavaScript 文件(这些文件由 Ajax 架构和您自己随意使用)，负责使用 UpdatePanel 更新部分页面，处理与在 Web 站点中定义的 Web 服务之间的交互。可以将 ScriptManager 添加到单个页面或添加到母版页中，让它变得在整个站点上都可用。

使用 Web 服务时，还需要告知 ScriptManager 要给客户端脚本提供 Web 服务。有两种方法可以实现：

- 在母版页中的 ScriptManager 中。
- 在使用 Web 服务的内容页中使用 ScriptManagerProxy 类。

要在全部或大多数页面中使用 Web 服务，最好在母版页的 ScriptManager 中声明 Web 服务。要这样做，可以给 ScriptManager 控件提供一个<Services>元素，该元素再包含指向公共服务的一个或多个 ServiceReference 元素。例如，要让创建的 NameService.asmx 服务在站点内的所有页面中可用，可以将下面突出显示的代码添加到母版页：

```
<asp:ScriptManager ID="ScriptManager1" runat="server">  
  <Services>  
    <asp:ServiceReference Path="~/WebServices/NameService.asmx" />  
  </Services>  
</asp:ScriptManager>
```

通过在母版页中引用该服务，它变得对于基于这个母版页的所有页面都可用。这也意味着每个页面都要下载运行这个服务所需的 JavaScript 文件。如果页面根本没有使用 Web 服务，这也会浪费带宽和资源。因此，对于只在一些页面上使用的服务，最好引用页面本身的服务。在没有使用母版页(里面有 ScriptManager)的正常页面上，可以直接将 ScriptManager 添加到 Web 窗体中。然而，如果使用的母版页有自己的 ScriptManager(在 Planet Wrox Web 站点中有这样的情况)，就要使用 ScriptManagerProxy 控件。因为在一个页面中只能有一个 ScriptManager，所以不能在使用母版页(里面有 ScriptManager)的内容页中添加另一个 ScriptManager。因此，ScriptManagerProxy 将作为内容页和母版页中的 ScriptManager 之间的桥梁，使得在哪里注册服务具有极大的灵活性。

正确配置了 ScriptManagerProxy 之后，就可以将完全相同的<Services>元素添加给它，就像使用 ScriptManager 本身一样。

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">  
  <Services>
```



```
<asp:ServiceReference Path="~/WebServices/NameService.asmx" />
</Services>
</asp:ScriptManagerProxy>
```

下面的“试一试”练习说明了如何使用 ScriptManagerProxy 从客户端代码注册和访问 Web 服务。

试一试

从客户端代码调用 Web 服务

在这个练习中，要在 ScriptManagerProxy 控件中注册 Web 服务，让它只在一个页面中可用。另外，还要修改服务，让脚本可以访问它的方法。最后，要写一些客户端 JavaScript 代码，通过它访问服务，然后显示其返回值。

(1) 需要做的第一步是将 ScriptService 特性添加到服务类中，将它标记为可被客户端脚本调用。为此，可以打开文件 NameService.vb 或从 App_Code 文件夹中打开 NameService.cs 文件，并取消对定义这个特性的行的注释。应该得到如下代码：

```
VB.NET
<System.Web.Script.Services.ScriptService()> _
<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class NameService
    Inherits System.Web.Services.WebService
```

```
C#
[System.Web.Script.Services.ScriptService]
public class NameService : System.Web.Services.WebService
{
```

(2) 之后，修改 WebService 特性的 Namespace 属性。在默认情况下，名称空间如下所示：

```
VB.NET
<WebService(Namespace:="http://tempuri.org/")> _
```

```
C#
[WebService(Namespace = "http://tempuri.org/")]
```

在 Web 服务开发过程中这个名称很好，但在将其放入生产环境后，该名称就应该能真实地反映服务的唯一名称，以避免与具有相同名称或类型的其他服务发生冲突。如果有自己的域名，可以改变名称空间，例如 http://www.yourdomain.com/。如果没有自己的域，也不要担心。将 Namespace 设置为默认值 http://tempuri.org/也可以。

(3) 下一步是创建使用该服务的页面，然后使用 ScriptManagerProxy 控件注册它。在 Demos 文件夹中添加一个新 Web 窗体，并将它命名为 WebServices.aspx。确保这个页面建立在定制的 BasePage 模板基础上，这样它就会有正确的母版页设置，并继承自 App_Code 文件夹中的 BasePage 类，然后给它赋予一个如 Web Services Demo 这样的 Title。添加这个页面之后，将 ScriptManagerProxy 控件从 Toolbox 的 AJAX Extensions 类别中拖动到 cpMainContent 占位符的标记中。

(4) 在 ScriptManagerProxy 元素内，添加<Services>元素，让它再包含 ServiceReference，其 Path 设置为前面创建的 NameService。注意，只要输入 Path=，IntelliSense 就会显示一个文件列表，从而

可以帮助选择正确的文件。单击列表底部的 Pick URL，定位到 WebServices 文件夹中的服务文件。在 WebServices.aspx 页面中应该得到如下代码：

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" runat="Server">
  <asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
    <Services>
      <asp:ServiceReference Path="~/WebServices/NameService.asmx" />
    </Services>
  </asp:ScriptManagerProxy>
</asp:Content>
```

(5) 在<ScriptManagerProxy>的结束标记下方，添加一个 Input (Text)和一个 Input (Button)，方法就是从 Toolbox 的 HTML 类别中拖动它们。通过使用纯 HTML 元素而不是 ASP.NET 服务器控件，可以看到要写的代码在客户端执行。将文本框的 id 设置为 YourName，将按钮的 id 设置为 SayHello。将按钮的 value 设置为 Say Hello。应该得到下面的标记：

```
</asp:ScriptManagerProxy>
<input id="YourName" type="text" />
<input id="SayHello" type="button" value="Say Hello" />
```

(6) 在这两行下面，使用下面的代码添加客户端 JavaScript 代码块：

```
<input id="SayHello" type="button" value="Say Hello" />
<script type="text/javascript">
  function HelloWorld()
  {
    var yourName = $get('YourName').value;
    NameService.HelloWorld(yourName, HelloWorldCallback);
  }

  function HelloWorldCallback(result)
  {
    alert(result);
  }

  $addHandler($get('SayHello'), 'click', HelloWorld);
</script>
```

(7) 按 Ctrl+Shift+S 组合键保存全部修改，然后在浏览器中请求页面 WebServices.aspx。输入您的名字，然后单击 Say Hello 按钮。如果一切正常，应该收到来自 Web 服务的问候消息，重复您的名字。图 10-9 显示 Apple 的 Safari 中的通知窗口。

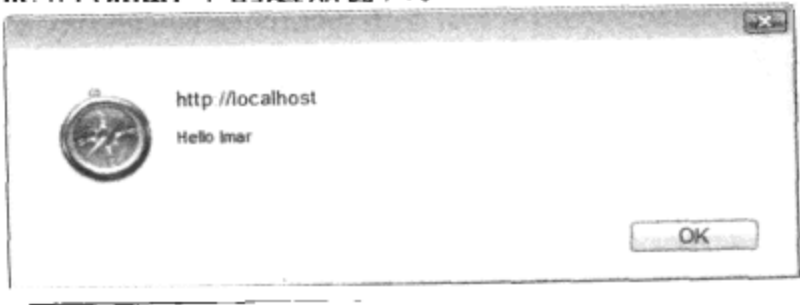


图 10-9



注意：如果收到的是错误而不是消息框，或者看到在屏幕左下角有一个小的黄色三角形，请确保像在此代码段中那样输入了 JavaScript。JavaScript 区分大小写，因此要确保所有大小写字母正确。同时还要保证在第(6)步添加的 JavaScript 块位于前面定义的输入框和按钮之后。最后，要确保到 Web 服务的路径与 .asmx 服务文件的实际路径相匹配，并且已经在服务类中应用了 ScriptService 特性。

工作原理

在本示例中使用的 Web 服务与在前面的“试一试”练习中测试页面中使用的 Web 服务几乎完全相同。唯一不同是 ScriptService 特性，它将服务标记为可被客户端脚本代码访问。

要将服务提供给应用程序中的客户端脚本，需要先注册它。可以在母版页内 ScriptManager 的 <Services>元素中完成。在母版页中注册 Web 服务的不利方面是：在站点内的每个页面中都要引用其客户端 JavaScript。对于只是偶尔使用的服务，最好将 ScriptManagerProxy 添加到特定页面，并在那里注册服务。在页面中，控件 ScriptManagerProxy 的外观和操作与标准控件 ScriptManager 很相似。但是，ScriptManagerProxy 控件实际上只是一个 proxy 类，该类可以将其所有的设置传递给母版页中真正的 ScriptManager 控件。使用 ScriptManagerProxy 控件建立<Services>元素，如下所示：

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
  <Services>
    <asp:ServiceReference Path="~/WebServices/NameService.asmx" />
  </Services>
</asp:ScriptManagerProxy>
```

所需做的就是通过设置 Path 属性来引用服务。与到目前为止在本书中看到的其他服务器端 URL 一样，可以使用~语法来查阅应用程序的根文件夹。

注册了服务之后，它就在客户端代码中可用。注意，VWD 中的 IntelliSense 非常智能，它能够发现已经定义和注册的 Web 服务。只要在客户端脚本块中输入 NameService(后面跟一个点)，IntelliSense 就会再次运行，并显示它找到的公有方法。图 10-10 显示了在 IntelliSense 列表中突出显示的 HelloWorld 方法。

这就很容易找到在站点内定义的正确服务。这对 Visual Studio 以前的版本是一个巨大改进，以前的版本在 IntelliSense 列表中只有固定数量的与 JavaScript 相关的选项。从 VWD 2008 开始，IntelliSense 实际上就已经能够查看代码，并用在代码中找到的正确的变量名、方法、服务等填充 IntelliSense 列表。在 VWD 2010 中，Microsoft 通过提高性能和 IntelliSense 列表中显示项的精确度，来更进一步地改进 IntelliSense。

要理解实际页面如何运行和访问 Web 服务，可以查看<script>块中的代码。

需要查看的第一块代码是 HelloWorld 方法。



图 10-10


```
function HelloWorld()
{
    var yourName = $get('YourName').value;
    NameService.HelloWorld(yourName, HelloWorldCallback);
}
```

首先，这个代码引用前面创建的文本框。在通常情况下，使用纯 JavaScript 时应该使用 document.getElementById("YourName")来获得文本框。然而，客户端 Ajax Framework 提供了名为\$get 的快捷方式，它能够实现相同功能。一旦引用了文本框，就可以访问它的 value 属性，从而获得用户输入的名称。

然后使用下面的代码将这个名称发送到 Web 服务方法 HelloWorld:

```
NameService.HelloWorld(yourName, HelloWorldCallback);
```

调用 HelloWorld 的第一个参数是 Web 服务方法所需的参数：保存您的名字的字符串。第二个参数 HelloWorldCallback 是对另一个 JavaScript 方法的引用，当服务返回其结果时，就触发这个方法。这称为成功回调方法，因为它是在服务已经成功返回其信息之后调用的方法。

对 Web 服务的调用特意设计为异步调用。这意味着对服务的调用在单独的线程中进行，HelloWorld 方法存在的时间很短。因为 Web 服务可能需要花很长时间响应，所以需要指定当它从服务返回时负责处理响应的方法。这里这个方法被命名为 HelloWorldCallback，但也可以给它起任何喜欢的名称。

除了这个成功的回调之外，也可以添加 Web 服务因为某些原因失败(例如因为网络连接失败或因为服务抛出异常)时触发的其他回调。在那种情况下，对 HelloWorld 的调用如下所示：

```
NameService.HelloWorld(yourName, HelloWorldCallback, ErrorCallback);
```

ErrorCallback 函数如下所示：

```
function ErrorCallback(error)
{
    alert(error.get_message());
}
```

传递给这个方法的 error 参数(该参数是一个 WebServiceError 对象)有方便的方法和属性来显示有关异常的信息。在这个示例中，可以使用 get_message()捕获在服务器上发生的原始异常。要了解关于 WebServiceError 对象更多的内容，可以通过访问 <http://tinyurl.com/ner882> 查看官方的 ASP.NET AJAX 文档。

如果一切按计划进行，对 HelloWorld 的调用就会触发 Web 服务方法 HelloWorld。这个方法接收名称，并返回一条友好的欢迎消息，和前面看到的一样。当 Web 服务返回值时，就会调用 HelloWorldCallback 方法。这个方法有 result 参数，它用于保存 Web 服务的返回值。

```
function HelloWorldCallback(result)
{
    alert(result);
}
```

在前面的练习中，result 是一个简单字符串。因此可以使用 alert(result)直接在 JavaScript 通知窗口中显示结果。

在其他情况下，`result` 参数可能保存更复杂的对象，它们提供对其属性的访问。ASP.NET AJAX Framework 最大的优点是，完成处理这些复杂对象的大部分工作对用户来说是透明的。如果从 Web 服务返回了像 `Person`、`Order` 或 `Review` 这样的复杂对象，ASP.NET AJAX Framework 会自动让这个对象在客户端脚本中可用，而不需要编写自定义的代码将对象从服务器站点实例转换为客户端 JavaScript 能够理解的内容。这个过程涉及一个概念，即 JSON(JavaScript Object Notation)。JSON 是一种极其简洁的方法，它描述的是需要从一台计算机转移到另一台计算机的对象(本例中是从服务器转移到客户端计算机上)。要了解关于 JSON 用法的更多内容，可以查看 <http://json.org> 或者在 MSDN 站点中搜索 JSON。

需要考察的最后一件事情是这一切首先是怎样开始的。当单击这个按钮时，就会自动触发客户端 `HelloWorld` 函数。但这是怎样实现的呢？答案是对 `$addHandler` 的神秘调用：

```
$addHandler($get('SayHello'), 'click', HelloWorld);
```

`$addHandler` 实际上是 Ajax 框架中定义的 `Sys.UI.DomEvent` 类的 `addHandler` 方法的一个快捷方式。可以用它在页面中注册这些对象特定事件的事件处理程序。这与到目前为止在 VB.NET 和 C# 服务器端代码中看到的事件处理程序类似。

在本示例中，再次使用 `$get` 来获得对按钮的引用。然后它的 `click` 事件就关联到了 `HelloWorld` 方法上。这意味着，无论什么时候单击这个按钮，都会执行 `HelloWorld` 函数内的代码。

和 `$get` 方法一样，`$addHandler` 方法是跨浏览器注册事件的一个方便的快捷方式。虽然可以为如 `Button` 单击(没有客户端 Ajax 框架)这样的事件注册事件，但 `$addHandler` 方法更容易以清晰、简洁和跨浏览器的方法注册事件，还提供了一些 `IntelliSense` 的额外功能。

`$get` 和 `$addHandler` 最大的好处是可以在任何站点中使用它们。所需做的就是母版页或内容页内包含 `ScriptManager` 控件，并准备使用客户端框架。不需要在页面内使用 Web 服务或其他与 Ajax 相关的控件。如果愿意，甚至不需要 ASP.NET 就可以使用客户端库。可以直接使用任何其他 Web 环境中的客户端 JavaScript 文件，包括纯 HTML 和 PHP 页面。这使得可以更容易地用最少的代码来编写高级的 JavaScript 功能。

很明显，本章介绍的 `NameService` 几乎没有现实用途。然而，本章介绍的 Web 服务的原则也能够很容易地应用于更复杂的服务，允许只使用几行代码就能从客户端 JavaScript 中访问服务器上的数据。

在第 18 章讨论调试时，还会看到 `NameService`。在那一章将逐行分析代码，因此会明白执行哪些代码以及以什么顺序执行。

尽管 Web 服务非常有用并且很容易创建，但是可能需要花费一点时间。有时，可能只需要发送和接收传送到页面的以及从页面中发出的当前所使用的一些信息。可以使用页面方法实现上述目的，下面我们介绍该内容。

10.3.4 页面方法简介

页面方法和 Web 服务有一些共同之处。对于两者，都可以使用很少的代码在客户端调用。可以向它们发送数据，并接收回发的数据。另外，当调用它们时，可以定义成功和失败回调方法。两者的不同之处在于，页面方法直接在现有的 ASPX 页面内定义，而不是在单独的 ASMX 服务文件中定义。只能从页面运行的脚本中调用页面方法。这使得当需要将那些小的、简单的限制在当前页面

内的功能时，页面方法成为理想选择。

要启用页面方法，需要将 ScriptManager 控件的 EnablePageMethods 属性设置为 True。不能在 ScriptManagerProxy 类上设置该属性，因此需要直接在 ScriptManager 控件上进行设置。就 Planet Wrox Web 站点来说，ScriptManager 控件放置在母版页内。一旦启用了页面方法，就可以按照下面的步骤设置和使用它：

(1) 在需要处理的页面的 Code Behind 文件中创建一个公共的静态方法(VB.NET 中称之为共享方法)。需要在这个方法中应用[WebMethod]属性(VB.NET 中是<WebMethod()>)。该方法可以有选择地接收(通过方法的参数)和返回一些数据。

(2) 编写需要的 JavaScript 来调用页面方法，并使用其结果。

在下面的“试一试”练习中您将了解其工作原理。



注意：静态方法是一个可以应用到类中而非类的一个实例中的方法。第 5 章中创建了一个带有 4 个实例方法的 Calculator 类。这意味着为了使用这些方法，需要首先使用 new 关键字(在 VB.NET 中是 New)创建类的一个实例。如果将方法改成静态方法，则可以在 Calculator 类上直接地调用这些方法，如下所示：

VB.NET

```
Dim result As Integer = Calculator.Add(4, 5) ' result is now 9
```

C#

```
int result = Calculator.Add(4, 5); // result is now 9
```

静态方法通常用于那种不需要用对象来保存自身状态的实用方法，但是接下来将看到，如果希望实现页面方法，仍需要静态方法。静态方法和实例方法各自服务于不同的目的，它们通常不容易进行交换。就 Calculator 类(该类不保留方法调用间的状态)而言，静态方法也是一种很好的选择。

试一试

从客户端代码调用页面方法

在本练习中，要修改 WebServices.aspx 页面，添加第 2 个按钮以调用一个页面方法。为了能轻松地比较在服务器上调用代码的两种技术，这里创建的页面方法与先前调用的 Web 服务类似。

(1) 在 Markup 视图下打开母版页 Frontend.master，并将 ScriptManager 控件的 EnablePageMethods 特性设置为 True：

```
<asp:ScriptManager ID="ScriptManager1" runat="server"
    EnablePageMethods="True"></asp:ScriptManager>
```

(2) 打开 Demos 文件夹中 WebServices.aspx 页面的 Code Behind 文件，在 Demos_WebServices 类中添加下面的服务器端方法：

VB.NET

```
<WebMethod()>
Public Shared Function HelloWorld(ByVal yourName As String) As String
    Return String.Format("Hello {0}", yourName)
End Function
```



```
C#
[WebMethod]
public static string HelloWorld(string yourName)
{
    return string.Format("Hello {0}", yourName);
}
```

注意，这个代码与 Web 服务中定义的代码几乎一样，包括 WebMethod 特性也是一样的。唯一的区别在于关键字 Shared(C#中是 static)。

WebMethod 特性无法被直接识别出来。为了解决这个问题，可以在页面顶部其他语句的下面输入如下的 using 或 Imports 语句：

```
VB.NET
Imports System.Web.Services

C#
using System.Web.Services;
```

还可以在代码中单击一次特性，然后按 Ctrl+.(Ctrl+句点)键，打开一个带有推荐选项的列表，并从中选择第一个选项来插入代码。

(3) 切换到 Markup 视图，为之前创建的 HTML 按钮创建一个副本，将其 id 特性设置为 SayHelloPageMethod，并修改按钮的 value 属性，以便更好地描述按钮的用途，如下所示：

```
<input id="SayHelloPageMethod" type="button"
      value="Say Hello with a Page Method" />
```

(4) 为按钮的客户端 click 事件创建一个事件处理程序，该处理程序与之前所创建的相似。以 HelloWorldPageMethod 作为客户端方法进行调用：

```
$addHandler($get('SayHelloPageMethod'), 'click', HelloWorldPageMethod);
```

(5) 实现 HelloWorldPageMethod 方法，如下所示：

```
function HelloWorldPageMethod()
{
    var yourName = $get('YourName').value;
    PageMethods.HelloWorld(yourName, HelloWorldCallback);
}
```

可以在同一脚本块中的 HelloWorld 函数下直接添加该方法。注意，这里不需要编写新的回调方法以处理 HelloWorld 的调用返回值。可以很容易地重用 Web 服务示例中创建的方法，因为该方法所做的仅仅是通知返回值。

(6) 保存所有的改动，按 Ctrl+F5 组合键在浏览器中运行页面。输入自己的名字并单击 Say Hello with a Page Method 按钮。然后，就可以看到与 Web 服务示例中一样的消息。



常见错误：如果得到一个关于 PageMethods 的尚未定义的错误，则要确保在方法的签名中添加了 static 或 Shared 关键字，并确保已在母版页的 ScriptManager 控件上将 EnablePageMethods 设置为 True。

工作原理

从客户端代码的角度来看，除了实际调用方法的代码外，几乎所有的代码都与 Web 服务示例中的相同。现在需要调用 PageMethods.MethodName(而不是 ServiceName.MethodName)来调用一个页面方法，这里的 PageMethods 是一个确定的名称，它指的是 ASP.NET AJAX JavaScript 的实现。当单击按钮时，ASP.NET AJAX Framework 运行必要的代码来调用在 Code Behind 文件中定义的方法。因为将方法标记成了静态，所以 ASP.NET 运行库不需要创建 Page 类的一个实例(因而也就不需要经历其整个生命周期)，而可以只调用方法。这样做可以得到一个来自方法的更快且更有效的响应。但是需要注意一个警告：因为方法是静态的并应用到了类中(不是该类的实例中)，因此无法访问页面方法中的实例成员，如页面中定义的控件。

对于发送和检索那些不需要完整回发或 Web 服务开销的少量信息来说，页面方法是一个理想的选择。可以在各种情况下使用页面方法，包括发送用户数据(如用户名称、喜好、访问的页面等)、从数据库或 Web 服务中获取最新数据等。

在上述 Web 服务和页面方法的示例中，我们使用客户端 ASP.NET AJAX Library 获得对按钮控件的引用，并使用\$get 和\$addHandler 分别启动了它们各自的单击处理程序。除了 Web 服务和页面方法外，ASP.NET Ajax 框架还提供有其他方法，正如下面将要看到的。

10.3.5 客户端 ASP.NET AJAX Library

起初您可能没有意识到客户端 ASP.NET AJAX Library 的重要性，但它的功能非常强大，并且能够提供浏览器中纯 JavaScript 所缺少的许多功能。Microsoft 从 .NET Framework 中吸取了许多好的功能，然后将它们转换为客户端 JavaScript 对应的功能。这就意味着可以在支持 JavaScript 的 Web 站点中使用熟悉的 .NET 概念，即使 JavaScript 不支持这些概念。

到目前为止，我们只介绍了客户端 ASP.NET AJAX Library 提供的一些方法和对象。然而，\$get 和\$addHandler 只是冰山一角。客户端库包括 6 个顶级的名称空间(包括根名称空间 Sys)和一个全局名称空间，它们允许访问 30 多个类，而这些类具有数百种有用的方法，可用来帮助创建富客户端 Web 界面。

例如，可以使用客户端框架建立和处理页面事件(如 load 和 unload)，使用 WebRequest 类对其他 Web 页面发出请求，根据 CultureInfo 类的代码中客户端的文化背景设置来呈现不同格式的数据，执行数据绑定(绑定的数据源有多种，如可以将 Web 服务绑定到用户的界面控件中，从而显示并编辑它们)，以及其他情况。

全局名称空间中包含了一些成员和类型，它们扩展了 JavaScript 原先设计的特性。还包含了在 JavaScript 中发现的类型，如 Array、Boolean、Error、Number、Object 和 String，这些类型已经被扩展为包含模仿 .NET Framework 的行为。表 10-4 列出了这些类型的一些常用的方法。

表 10-4

类 型	方 法	用 途
String	format	使用指定的格式字符串和参数返回一个格式化的字符串。例如： var greeting = String.format('Hello {0}', name);
	endsWith	用于确定一个字符串是否以另一个字符串开始或结束。例如：
	startsWith	var isSonic = 'Sonic Youth'.startsWith('Sonic');

(续表)

类 型	方 法	用 途
String	trim trimEnd trimStart	删除字符串前面和后面的空格。例如 var trimmed = ' Sonic Youth '.trim();// results in Sonic Youth
Boolean	parse	将包含有文本 true 或 false 的字符串转换成真正的布尔值。出现其他值时则抛出一个异常。例如： var isTrue = Boolean.parse('true');
Date	format	可以获得日期的不同的格式化字符串表示。例如： alert(new Date().format('f'));// Alerts a date like Monday, 22 March 2010 23:52

客户端 ASP.NET AJAX Library 很大,所以并不总是能很容易地找到需要的方法、类或名称空间。为了更有效地使用客户端库,可以使用一些很好的资源。第一个是 IntelliSense,它可以在不同的类型中提供可用成员,它通过查找指定给类型的值来完成这个工作。例如,在图 10-11 中可以看到,VWD 推测变量 name 的类型是字符串,这是因为该变量提供了与字符串相关的方法,如 trim 和 trimstart,这两个方法已经被客户端库添加到 String 类型中了。

如果指定一个数字给变量,就会得到一个完全不同的列表,如图 10-12 所示。

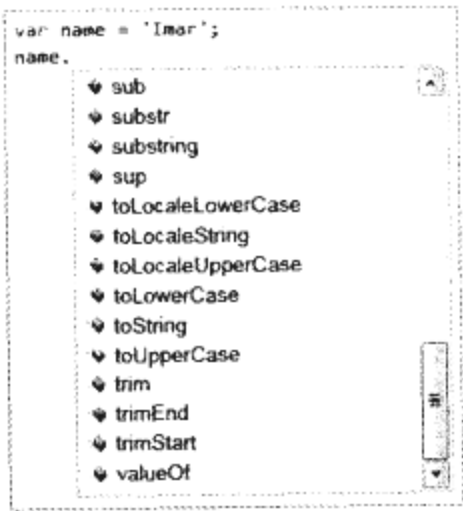


图 10-11

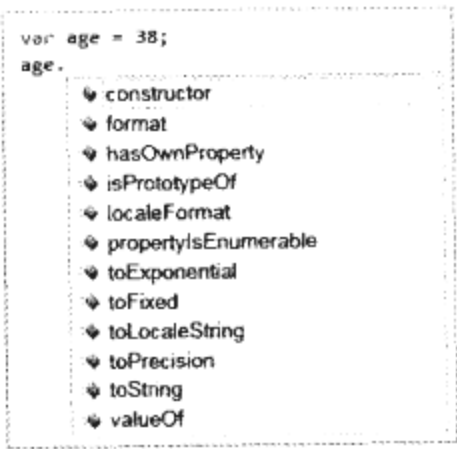


图 10-12

许多成员已经被文档化,因此还可以得到一个很好的提示文本,来说明如何使用这些成员。如果没有看到正确的方法出现,则请确保页面(或母版页)包含一个 ScriptManager 控件。如果没有该控件,客户端 JavaScript 框架将无法使用,因此 VWD 就会禁止使用浏览器的最终页面中不可用的功能。有时 IntelliSense 很可能没有在列表中显示所期望的成员,但是这并不能说明这些方法是不可用的。因为 JavaScript 不是一种强类型语言,所以 VWD 需要做许多的解析、推断和猜测工作来提供正确的选项。当没有看到期望的选项时,请确保更正了代码中全部的错误,保存所有的修改,然后关闭并再次打开文件。在很多情况下,这样做可以解决问题。

另一个很好的资源是 MSDN 站点上的官方 ASP.NET AJAX 文档,可以在 <http://tinyurl.com/AjaxClient4> 上找到它。在这里可以找到一个完整的文档列表,该列表包含不同的名称空间及其所包含的所有类型和成员。

10.3.6 这仅仅是开始

ASP.NET AJAX 包含的内容还有很多，无法在一章中介绍完。虽然 ASP.NET AJAX 的服务器和客户端部分可能是最大的、使用最多的功能，但其他的功能也是值得介绍的，包括：

- **ASP.NET AJAX 控件工具箱：**一个支持 AJAX 的控件的庞大集合，带有诸如日历扩展器和能自动完成的文本框这样的功能。可以在 www.asp.net/ajax/AjaxControlToolkit/Samples/ 上查看控件工具箱。
- **成员和角色服务：**ASP.NET AJAX 允许与客户端代码中的服务(如成员和角色服务)直接进行交互。在第 16 章中会看到关于成员和角色服务的更多内容，但是不会深入探讨 AJAX 功能。
- **客户端数据绑定：**使用客户端数据绑定功能，可以创建一个有吸引力的数据驱动的 Web 应用程序，该应用程序允许用户从客户端管理数据。用户可以通过 Web 服务使用服务器上的数据，而不需要回发整个页面。通过智能模板引擎，可以使用标准的 HTML 和 CSS 轻松地定义页面的外观，而 AJAX 关心的是所有的数据处理函数。要了解客户端数据绑定功能，可以访问 URL 地址为 <http://tinyurl.com/ClientDataBinding> 的 MSDN 站点。
- **Microsoft 通过借鉴开发人员社区里的内容，积极地开发了 Microsoft ASP.NET AJAX。**可以在官方的 ASP.NET AJAX 网站上(www.asp.net/ajax)发现更多新的改进、文本说明、录像及示例代码。

10.4 有关 Ajax 的实用提示

记住下面这些提示有助于更好地使用 ASP.NET AJAX：

- 由于 UpdateProgress 面板的内容只在 Ajax 页面更新过程中才可见，所以很难设计它的内容。只有在产生到服务器的回发后的几秒或更短的时间内才能看到内容。为了让设计 UpdateProgress 面板变得更容易，应该首先设计 UpdateProgress 面板外面的消息。例如，在本章的“试一试”练习中，应该将<div id="PleaseWait">移动到所有控件之外，从而让它总是可见。然后就可以修改<div>的 HTML 和 CSS，直到它正确显示为止。然后可以将<div>移回 UpdateProgress 面板中，因此就只在部分页面更新过程中显示它。
- 无论什么时候使用 UpdatePanel，都要考虑添加相关的 UpdateProgress 控件。即使因为 UpdatePanel 刷新太快，没有看到这种需求，也值得为慢计算机或慢网络上的人添加 UpdateProgress 控件。在页面方便并且可见的区域(例如在 Footer 区域)中给母版页添加一个 UpdateProgress 控件会更好。不要设置 AssociatedUpdatePanelID，让进程面板显示所有的 Ajax 回调。这样，就不需要在站点不同的区域中使用大量不同的等待指示器。
- 不要滥用 UpdatePanel 控件。在许多情况下，当使用 UpdatePanel 控件时，会感觉应用程序的性能增加了，即使实际性能相同。这是一件好事，因为用户会认为应用程序比没有 UpdatePanel 控件时运行得更快。然而，使用太多的 UpdatePanel 控件会让用户感到迷惑，特

别是当他们不需要 UpdateProgress 控件告诉他们正在发生的事情时。可以考虑使用 Web 服务和页面方法来替代，因为它们可以减少总开销以及需要转移的数据。

10.5 本章小结

Ajax 是一种广泛而且十分有趣的技术，可以给站点增加许多功能。它可以分为两个不同的部分：服务器端控件和客户端 JavaScript Framework。

在服务器上， UpdatePanel 控件允许立即创建无闪烁页面，而 ScriptManager 控件则作为服务器和客户间的桥梁，负责完成诸如注册必需的客户脚本这样的任务。AJAX Extensions 工具箱中还包 括 ScriptManagerProxy、UpdateProgress 和 Timer 控件。

除了这些非常有用的服务器端控件之外，ASP.NET AJAX Framework 也带有一些富客户端框架， 它们允许只使用一些代码行就能访问站点中的 Web 服务和页面方法。Web 服务和页面方法都可以用 于在服务器中交换数据，而不需要限制或刷新用户界面。这个框架提供了数百个有用的方法，它们 可以用于浏览器中页面的各个方面。所需做的就是 在母版页或内容页中包含 ScriptManager 控件，并 在自己的 Web 页面中使用完整的客户端框架。

虽然 Ajax 本身是一种非常引人注目的技术，但它在更丰富的数据驱动场景中更有用。例如， 当排序、过滤或翻页数据时，使用从数据库返回的记录周围的 UpdatePanel 控件来避免闪烁，可以 极大地提升用户的浏览体验。在第 12 章将介绍如何使用数据库。使用从本章学到的 Ajax 知识，您 可以快速创建无闪烁的、数据库驱动的 Web 页面。

10.6 练习

- 1. Toolkit 的 AJAX Extensions 类别定义一个 ScriptManager 和一个 ScriptManagerProxy。解释这两 个控件之间的区别，并解释何时要使用 ScriptManager，何时使用 ScriptManagerProxy。
- 2. 如何让用户知道部分页面更新正在进行？
- 3. 要将站点内的方法提供为客户端脚本可以调用的 Web 方法，需要创建一个特殊类并应用两 个属性。需要创建哪个类？应用哪些特性？
- 4. 根据页面方法，在页面中展示和使用方法的步骤是什么？

练习的答案见附录 A。

本章要点回顾

AJAX	Asynchronous JavaScript And XML，它是一个术语，是用于创建无闪烁 Web 页面以及从客户端 代码与服务器进行交互的一组技术的集合
特性	一个代码元素，可以应用于其他元素中(如类和方法)来改变其含义或行为
页面方法	一个服务器端静态方法(VB.NET 中是共享方法)，在页面中定义，可以从客户端脚本中调用

ASP.NET 4 入门经典——涵盖 C# 和 VB.NET(第 6 版)

(续表)

ScriptManager 控件	Microsoft ASP.NET AJAX Framework 的一个核心组件, 负责管理客户端脚本文件和服务器端的 Ajax 行为
ScriptManagerProxy 控件	内容页与母版页中定义的 ScriptManager 控件之间的桥梁
UpdatePanel 控件	一个控件, 通过只更新在其<ContentTemplate>元素中定义的内容来创建无闪烁页面
UpdateProgress 控件	一个面板(<div>或), 可以在一个 Ajax 操作执行期间异步显示
Web 服务	一个方法, 可以在 Internet 或局域网中被其他应用程序调用

第 11 章

jQuery

本章要点

- jQuery 简介
- 如何使用 jQuery 增强页面，包括添加丰富的视觉效果和动画
- 如何使用可用的插件扩展 jQuery

前面的章节中介绍了 JavaScript，这是在客户端编写脚本和与客户端的 Web 页面元素进行交互所运用的实际标准语言。虽然给出的示例比较简单，但是 JavaScript 的功能其实要丰富得多，也强大得多。尽管如此，它仍然存在一些不足之处。JavaScript 存在的一个问题是，并不是所有的浏览器都以相同的方式解释它。编写的大部分 JavaScript 代码都可以在主流浏览器中运行，但是浏览器对代码的解释和表现出的行为存在一些细微的差别，所以很难编写能够在所有主流浏览器中表现出相同行为的代码。而且，JavaScript 还缺少能够使日常的 JavaScript 编码变得更加轻松的实用功能。例如，它提供了可以找到页面上的特定元素的内置方法(如第 9 章所述，这个方法为 `getElementById`)和找到特定 HTML 标记的全部元素的内容方法(`getElementsByTagName`)，但是它缺少像 `getElementsByClassName` 这样的用来找出应用到特定类的元素列表的功能。第 10 章介绍的客户端 ASP.NET AJAX 库可以帮助解决这样的一些问题，但是在有些场合它也无法满足要求。

幸运的是，Internet 开发社区一直在致力于开发一种在后台使用 JavaScript 的框架，这种框架扩展了 JavaScript 的功能，同时也提供了非常丰富的功能集，用来帮助构建人员构建交互式的客户端 Web 页面。他们已经开发出了许多 JavaScript 库(其中大多数都是免费的)，包括：

- Prototype(<http://prototypejs.org>)
- Scriptaculous，这是 Prototype 的一个增件(<http://script.aculo.us>)
- Ext JS(<http://extjs.com>)
- Dojo(<http://dojotoolkit.org>)

有一个框架得到了大量的关注，它就是 jQuery。jQuery 最早由 John Resig 在 2006 年 1 月开发和发布，现在已经成长为一个备受欢迎的客户端框架。Microsoft 也注意到 jQuery 功能强大，并决定在自己的产品中附送这个框架。最初，jQuery 随 Microsoft ASP.NET MVC 框架一起提供，现在 Visual

Studio 和 Visual Web Developer 2010 中也包含了这个框架。

11.1 jQuery 简介

jQuery 库的主要关注点一直是简化访问 Web 页面元素的方法、帮助处理客户端事件、提供视觉效果(如动画)支持，以及使得在应用程序中使用 Ajax 变得更加简单。在 2006 年 1 月，John Resig 公布了 jQuery 的第一版，然后在 2006 年 8 月正式发布了 jQuery 1.0。后来又陆续发布了许多版本，目前 jQuery 1.4.1 是最新的稳定版本。



注意：由于对 jQuery 的开发仍在持续进行，所以很可能当您阅读完本书的时候，更新的版本已经发布出来了。为了避免由于 jQuery 库发生重大改变而出现版本问题，建议使用随本书代码一起提供的 jQuery 版本。这样可以保证您练习并掌握本章以及后续章节中的所有示例。当掌握了 jQuery 以后，您就可以从 <http://jquery.com> 上下载并使用最新版本。一定要查看您下载的版本中包含的改动记录，以便了解与本书使用的版本相比，新版本包含的新功能和改动的功能。

可以从 jQuery 的官方网站 <http://jquery.com> 上下载 jQuery 的最新版本。该网站不但提供了可以下载的文件，还提供了文档、FAQ、教程和其他有助于更好地利用 jQuery 的信息。除了从 jQuery 的网站上下下载库文件以外，还有一种获得 jQuery 文件的方法：使用 ASP.NET Web 站点模板创建的任意新 Web 站点都包含一个 Scripts 文件夹，其中已经包含了必要的 jQuery 文件。但是，在第 2 章中，您基于 ASP.NET 空 Web 站点模板建立了 Planet Wrox Web 站点，这个模板不包含这些 jQuery 文件。后面将会讨论如何手动向 Web 站点添加 jQuery 文件。

因为 jQuery 库会增加网页的大小，所以应该明确决定是否在 Web 站点中包含它。当决定把 jQuery 库添加到 Web 站点中时，还需要作出一些选择。

11.1.1 选择引用 jQuery 的位置

要在 Web 站点中包含 jQuery，有几种选项可供选择：

- 只在需要 jQuery 的网页或者用户控件中添加对 jQuery 库的引用。
- 在 Web 站点的母版页中添加对 jQuery 库的引用，从而使所有的页面都可以使用该 jQuery 库。

两种方法各有优缺点。只在需要 jQuery 库的页面上添加对它的引用可以减小页面大小。当用户浏览没有使用 jQuery 的页面时，就不需要下载 jQuery 库文件。而当他们下载了库文件以后，浏览器就会缓存库文件的一个副本，从而使得在以后访问页面时，不需要再次下载这些文件。

在 Web 站点的母版页中添加对 jQuery 库的引用十分方便，因为所有基于该母版页创建的页面都会自动获得对 jQuery 的访问权。但是，这会对 Web 站点第一个页面的性能造成一些冲击，因为需要从服务器上下载库文件。

因为 jQuery 库很小，所以一般要在母版页中包含它。

除了添加 jQuery 文件的位置以外，还有一些关于如何包含库文件的选项。

11.1.2 包含 jQuery 库的不同方式

因为 jQuery 库由一个使用 JavaScript 代码编写的文件组成，所以可以使用标准的<script>语法在页面、用户控件或母版页中嵌入对 jQuery 库的引用：

```
<script src="FileName.ext" type="text/javascript"></script>
```

必须使用一个独立的结束</script>标记，因为如果使用自结束标记，一些浏览器无法正常运行代码。

我选择将所有的客户端脚本文件存储到 Web 站点根目录下的 Scripts 文件夹中，所以对 jQuery 库(jQuery-1.4.1.min.js)的引用如下所示：

```
<script src="/Scripts/jquery-1.4.1.min.js" type="text/javascript"></script>
```

也可以将引用嵌入到在第 10 章已经添加到母版页的 ScriptManager 控件中。ScriptManager 控件有一个<Scripts>子元素，可以用来注册将会添加到浏览器的最后一个页面的 JavaScript 文件。在 ScriptManager 中注册的 JavaScript 文件的最简形式如下所示：

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Scripts>
    <asp:ScriptReference Path="~/Scripts/jquery-1.4.1.min.js" />
  </Scripts>
</asp:ScriptManager>
```

另外一种方法是使用 Microsoft 的内容传送网络(Content Delivery Network, CDN)或 Google Code 引用 jQuery 库的在线版本。更多信息可以访问 Microsoft 的 CDN 站点 www.asp.net/ajax/cdn，或者 Google 的 API 站点 <http://code.google.com/apis/ajaxlibs/>。

使用外部库的在线版本的优势在于可以提升服务器的性能并降低带宽。因为站点的访问者很可能已经在访问另外一个站点的时候下载了共享脚本，所以当它们访问您的站点的时候可能就不需要再次下载这些文件了。

在下面的“试一试”练习中，将向 Planet Wrox Web 站点的母版页添加 jQuery 库 1.4.1。在设置好 jQuery 库后，本章的其余部分将会讨论 jQuery 的工作原理，以及如何在 Planet Wrox Web 站点中利用 jQuery。

试一试

第一个 jQuery 页面

在本例中，将向母版页添加 jQuery 库，从而使站点内的所有页面都可以访问它。记住，即使在您阅读本书时已经有新的 jQuery 版本发布，我们仍然强烈建议您使用随本书一起提供的版本(也可以从 Wrox 网站上下载该版本)。这可以保证所有示例的行为与描述一致。

- (1) 首先在 Visual Web Developer 中，向站点的根文件夹中添加一个新的 Scripts 文件夹。
- (2) 接下来，在 Windows 资源管理器中找到并打开您解压的本书附带的代码的文件夹。如果按照本书前言中的说明进行操作，那么这个文件夹的位置为 C:\BegASPNET\Resources。打开 Chapter 11 文件夹，然后将带有.js 扩展名的 3 个 JavaScript 文件从 Windows 资源管理器中拖动到第(1)步在 VWD 中创建的 Scripts 文件夹中。jquery-1.4.1.min.js 是实际的 jQuery 库，而 jquery-1.4.1-vsdoc.js 是 IntelliSense

文档文件，只在 VWD 内使用。jquery.updnWatermark.js 包含 jQuery 的一个插件，在本章即将结束时将会讨论这个文件。现在，Solution Explorer 应如图 11-1 所示。

(3) 接下来应该向站点的母版页添加 jQuery 库，从而使站点内的全部页面都可以访问 jQuery 库。为此，打开 MasterPages 文件夹下的 Frontend.master 文件，并在必要时切换到 Markup 视图。找到 ScriptManager 控件，添加下面突出显示的标记：

```
<asp:ScriptManager ID="ScriptManager1" runat="server"
EnablePageMethods="true">
    <Scripts>
        <asp:ScriptReference
Path="~/Scripts/jquery-1.4.1.min.js" />
    </Scripts>
</asp:ScriptManager>
```

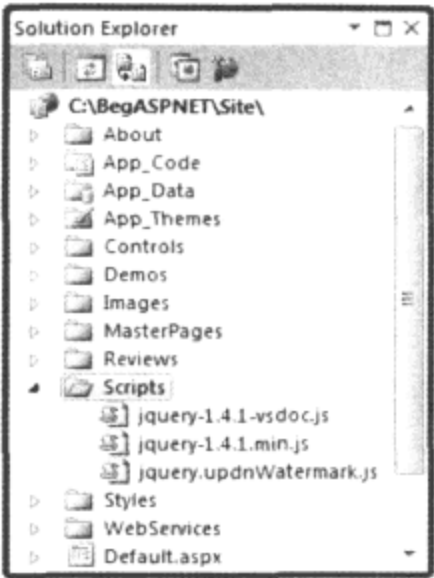


图 11-1

如果 ScriptManager 还没有独立的结束标记，则应该添加一个 (并删除开始标记的斜杠(/))，否则不能正确地添加代码。

(4) 保存并关闭母版页，因为现在还不需要对它执行其他操作。

(5) 要想尝试 jQuery 库，可以在 Demos 文件夹中根据定制的模板创建一个全新的 Web 窗体。将新窗体命名为 jQuery.aspx，并将它的 Title 设为 jQuery Demo。

(6) 当在 Markup 视图中打开新页面后，向 cpMainContent 的 Content 代码块中添加如下代码：

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" runat="Server">
    <input id="Button1" type="button" value="button" />
    <script type="text/javascript">
        $(document).ready(function() {
            $('#MainContent').css('background-color', 'green')

            $('#Button1').click(function() {
                $('#MainContent').css('background-color', 'red')
                .animate({ width: '100px', height: '800px' })
            });
        });
    </script>
</asp:Content>
```

和其他许多编程语言一样，JavaScript(以及 jQuery)对缺少引号、大括号和小括号十分敏感，所以一定要完全按照这里显示的代码进行输入。或者，可以从本书附带的源代码中的 jQuery.aspx 页面中复制并粘贴这些代码。

注意，在输入代码时，“IntelliSense”将会弹出，帮助您完成代码，并通过工具提示提供关于各种方法和参数的信息。如果没有弹出“IntelliSense”，那么确保一定要把正确的<Scripts>元素添加到母版页中。然后，保存并关闭全部已打开的文档，然后重新打开 jQuery.aspx。

(7) 将改动保存到页面中，然后按下 Ctrl+F5 组合键以便在浏览器中打开该页面。注意，MainContent div 的背景色变成了绿色。单击按钮，注意背景色变成了红色，而 MainContent 元素的大小也发生了改变，最终变成了 100 像素宽，800 像素高。



常见错误：如果出现了错误，或者没有看到动画，那么一定要确保正确地在页面中添加了指向 jQuery 的链接。另外，检查代码中是否存在输入错误。

工作原理

虽然本例中显示的效果并不十分醒目，但是后台仍然完成了大量的工作。为了理解它的工作原理，首先回顾一下母版页，因为正是在这里添加了对 jQuery 库的引用：

```
<asp:ScriptManager ID="ScriptManager1" runat="server" EnablePageMethods="true">
  <Scripts>
    <asp:ScriptReference Path="~/Scripts/jquery-1.4.1.min.js" />
  </Scripts>
</asp:ScriptManager>
```

这段代码说明脚本管理器包含一个指向 jQuery 库的 script 元素。如果在浏览器中查看页面的 HTML 源代码，那么应该看到如下所示的 script 元素：

```
<script src="../../../Scripts/jquery-1.4.1.min.js" type="text/javascript"></script>
```

这行代码说明浏览器从 Scripts 文件夹中下载 jquery-1.4.1.min.js 文件，从而使页面可以访问 jQuery 库提供的全部功能。

接下来需要查看的是 jQuery 演示页面的代码。首先，添加了一个标准的<script>块，其中可以包含 JavaScript。在这个块中，添加了一些在浏览器加载页面完成后触发的 jQuery 代码。页面就绪后，起始大括号({)和结束大括号(})之间的代码将会执行：

```
<script type="text/javascript">
  $(document).ready(function() {
    // Remainder of the code skipped
  });
</script>
```

因为 jQuery 代码与页面上的元素进行交互，所以经常需要等待整个页面加载完成，然后才能通过编程操作元素。添加这样的 jQuery 代码是将代码执行延迟到页面就绪后的一种标准做法。在本章后面可以看到更多这样的代码，包括访问“文档就绪函数”的便捷方式\$(document).ready。

页面就绪后执行的代码包括两个部分。代码的第一行将 MainContent div 的背景色设为绿色：

```
$('#MainContent').css('background-color', 'green')
```

这行代码获取对 MainContent div 元素的引用，然后调用 css 方法将背景色修改为绿色。第 10 章提到过\$get 方法，它通过自己在客户端 ASP.NET AJAX 库中的 id 获取对页面中某个元素的引用。在本例中，\$('#MainContent')是 jQuery 中与之等效的方法，但是在后面将会看到，它的功能强大得多。

代码的第二部分为添加到页面的 HTML 按钮建立一个单击处理程序，类似于第 10 章使用的 addHandler。在该单击处理程序中，可以看到一些代码将 MainContent div 的背景色修改为红色，还有一些代码使用流体动画改变其宽和高：

```
$('#Button1').click(function() {
    $('#MainContent').css('background-color', 'red')
    .animate({ width: '100px', height: '800px' })
});
```

同样，在本章后面将会更详细地学习关于 jQuery 如何找到按钮和 div 元素，以及 css 和 animate 方法的工作原理的知识，所以如果现在不能理解这些代码，也不必过于担心。

当在浏览器中单击按钮时，MainContent 的背景色就会变成红色，而它的宽和高将会分别变成 100 像素和 800 像素。

在输入 jQuery 代码时，能注意到“IntelliSense”会提供帮助。输入\$(后，将会立即显示一个工具提示，列出可以传递给这个函数的信息。类似地，“IntelliSense”可以帮助您找到并完成 css 方法和需要传递给该方法的各种参数，如图 11-2 所示(在显示“IntelliSense”列表下的工具提示时，该图使其更适合本书的宽度)。

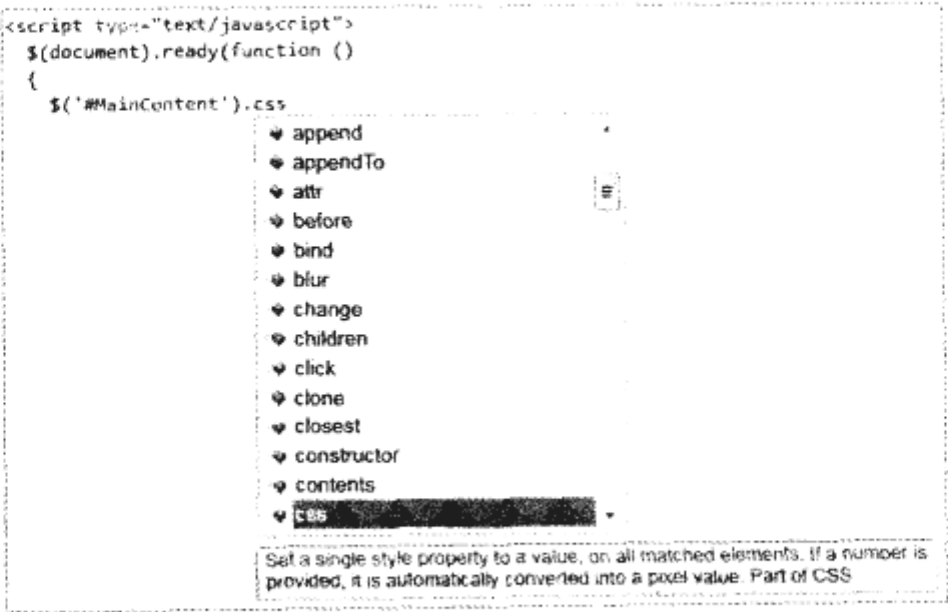


图 11-2

jQuery 的“IntelliSense”通过添加到站点的一个额外的文件 jquery-1.4.1-vsdoc.js 发挥作用。VWD 遍历解决方案，查找并分析以 vsdoc.js 结尾的文件，然后使用在这些文件中找到的文档来构建“IntelliSense”列表。

使用独立的文档文件的原因是减小原始 jQuery 库的大小。如果没有该文档，那么库的大小只有 70KB；而如果带有该文档，那么文件的大小增加到了 230KB。

只有在 VWD 中进行开发的时候才应该使用 vsdoc 文件，在页面中绝不应该包含一个指向该文件的链接，因为与实际库文件相比，这么做没有任何价值。因为文档文件特别大，所以如果使用该文件而不是实际的库文件，则是在浪费宝贵的带宽和时间。

现在已经看到了 jQuery 的实际应用，接下来需要更好地理解它的可能性和语法。

11.2 jQuery 语法

要想理解和使用 jQuery，需要掌握一些基础知识。首先，需要更深入地理解 jQuery 的核心功能，包括前面看到的 \$ 函数，以及 \$ 函数的 ready 方法。通过使用 ready 方法，当浏览器完成页面的加载后，可以执行一些代码。接下来，需要学习 jQuery 的 selector 和 filter 语法，这样就可以通过自己指定的条件在页面中查找元素。当获得一个指向页面中一个或多个元素的引用后，就可以对它们应用多种方法，比如前面提到的 css 方法。还需要知道关于 jQuery 事件(event)的一些知识，因为它们允许向 HTML 元素可能触发的事件(例如 click、onmouseover 等)附加行为。在接下来的几节中将会讨论主要的 jQuery 主题。


11.2.1 jQuery Core

您编写的大部分 jQuery 代码将在浏览器完成页面加载后执行。等到页面完成 DOM 加载后再执行代码十分重要。DOM(Document Object Model，文档对象模型)是 Web 页面的一种分层表示，包含所有 HTML 元素、脚本文件、CSS、图像等的一个树形结构。DOM 总是与您在浏览器中看到的页面保持一致，所以如果借助编程修改 DOM(例如，使用 jQuery 代码)，那么这种修改将在浏览器中显示的页面上反映出来。如果过早执行 jQuery 代码(例如，在页面的最顶端)，那么 DOM 可能还没有加载完成脚本中引用的全部元素时就产生了错误。幸运的是，可以使用 jQuery 中的 ready 函数，将代码的执行推迟到 DOM 就绪。前一个“试一试”练习中已经展示了如何使用 ready 函数，但是这里再次显示该函数，以便您对它有更好的理解。

```
$(document).ready(function() {  
    // Code added here is executed when the DOM is ready.  
});
```

当页面准备就绪，可以执行 DOM 操作时，添加到起始和结束大括号之间的全部代码都将执行。jQuery 也提供了 ready 函数的一个快捷方式，从而使得编写当 DOM 就绪后触发的代码变得更加容易。下面的代码段与前一个示例的效果相同：

```
$(function() {  
    // Code added here is executed when the DOM is ready.  
});
```



注意：在 jQuery 库本身完成加载以前，不能执行任何引用 jQuery 库的代码，这一点十分重要。因为指向 jQuery 库的链接是由 ScriptManager 在 <body> 标记之后添加的，所以之后需要在页面中找到一个位置。一个不错的位置是在母版页中定义的 </body> 结束标记的附近。

因为 jQuery 代码通常是专门针对每个页面编写的，所以应该只在需要 jQuery 代码的页面的结尾添加代码。为了简化这项任务，可以在母版页中添加一个 ContentPlaceHolder，使其主要用于这个目的。使用这个母版页的页面就有了一个方便的地方来编写 jQuery 代码。下一个“试一试”练习中将展示如何进行这种操作。

在前一个 jQuery 示例中，展示了选择页面中的 MainContent div 和按钮的代码。不过其实 jQuery

提供了很多选项，用来从页面中选择特定的元素。接下来就将讨论这些选项。

11.2.2 使用 jQuery 进行选择

在 jQuery 中，可以使用美元符号(\$)作为在页面中查找元素的快捷方式。找到并返回的元素称为匹配集(matched set)。`$`方法的基本语法如下所示：

```
$('Selector Here')
```

在引号(可以使用单引号或者双引号，只要在两端使用相同的类型即可)之间，输入一个或多个选择器，接下来就将讨论这方面的内容。`$`方法返回 0 个或多个元素，然后可以使用多种 jQuery 方法操作返回的这些元素。例如，要向所有 `h2` 元素应用 CSS，可以使用 `css` 方法：

```
$('h2').css('padding-bottom', '10px');
```

这行代码使页面中的所有二级标题具有 10 像素高的底内边距。许多 jQuery 方法的优点在于，除了应用某些设计或行为，它们会再次返回匹配集。这样就可以对相同的匹配集调用其他方法。这个概念称为链接(chaining)。在这种概念中，使用一个方法的结果作为另外一个方法的输入，从而产生一个效果链。例如，下面的代码首先修改了页面中全部二级标题的字体，然后在 5 秒钟内让它们淡出，直至不可见：

```
$('h2').css('font-size', '40px').fadeOut(5000); // timeout is in milliseconds
```

在了解了 selectors 和 filters 的工作原理之后，将会学到像 `animate` 和 `fadeOut` 一样更多不同的视觉效果。

1. 基本选择器

jQuery 选择器使您能够找到页面的文档对象模型中的一个或多个元素，以便向它们应用各种类型的 jQuery 方法。使用 jQuery 选择器的优点在于您已经知道它们的工作原理。jQuery 的设计者并没有开发出一种新技术来查找页面元素，而是使用了您已经熟悉的基于选择器的现有语法 CSS。还记得第 3 章讲到的 CSS 选择器吗？可以在 jQuery 中使用完全相同的选择器。

通用选择器

和对应的 CSS 选择器一样，通用选择器会匹配页面中的全部元素。要将页面中每个元素的字体系列设置为 Arial，可以使用下面的代码：

```
$('*').css('font-family', 'Arial');
```

ID 选择器

和对应的 CSS 选择器一样，这个选择器通过 id 查找和获取元素。例如，要为名为 `Button1` 的按钮设置 CSS 类，可以使用下面的代码：

```
$('#Button1').addClass('NewClassName');
```

当这行代码使用 `addClass` 方法设置 CSS 类时，将会遵循标准的 CSS 规则。这意味着要想使这行代码生效并修改按钮的外观，需要通过外部 CSS 文件或者嵌入式样式表将 `NewClassName` 类提供给页面。如果想要回顾一下不同的 CSS 样式表类型，可以参考第 3 章。



注意：jQuery 的 `$('#Button1')` 和 ASP.NET AJAX 的 `$get('Button1')` 都会获得对 id 为 Button1 的一个元素的引用。那么应该选择哪一个方法呢？一般来说，当对结果应用任意 jQuery 方法(例如 css 方法)时，都应该使用 jQuery 的 `$` 方法。而在操作单个元素(本例中的 Button1 就属于这种情况)，并且想要修改该元素的某个标准属性(例如将其 value 设置为其他文本)时，则可以使用 `$get` 代替。在这种情况下也可以使用 jQuery 的 `$`，但是因为所有的 jQuery 选择器都返回一个对象集合，所以需要通过索引方式，使用 `[0]` 或 `get(0)` 得到第一个(在本例中也是唯一一个)元素。下面的 3 个示例具有等效的功能，它们都将 Button1 的值设置为 Click Me:

```
$get('Button1').value = 'Click Me';
$('#Button1')[0].value = 'Click Me';
$('#Button1').get(0).value = 'Click Me';
```

元素选择器

这个选择器获得与特定的标记名相匹配的 0 个或多个元素的引用。例如，下面这行代码将所有二级标题的文本颜色设置为蓝色。

```
$('h2').css('color', 'blue');
```

类选择器

类选择器获得与特定的类名相匹配的 0 个或多个元素的引用。考虑下面的 HTML 代码段：

```
<h1 class="Highlight">Heading 1</h1>
<h2>Heading 2</h2>
<p class="Highlight">First paragraph</p>
<p>Second paragraph</p>
```

注意，4 个元素中有两个元素都有一个名为 Highlight 的 CSS 类。下面的 jQuery 代码将第一个标题和第一个段落的背景色修改为红色，而保持其他元素不变：

```
$('.Highlight').css('background-color', 'red');
```

分组和合并选择器

和 CSS 一样，可以分组和合并选择器。下面的分组选择器修改页面中全部 h1 和 h2 元素的文本颜色：

```
$('h1, h2').css('color', 'orange');
```

通过使用合并选择器，可以找出被其他一些元素包含着的特定元素。例如，下面的 jQuery 只修改 MainContent 元素中包含的段落，而保持其他所有段落不变：

```
$('#MainContent p').css('border', '1px solid red');
```

为了理解 jQuery 选择器以及可以对匹配集应用的效果，下一个“试一试”练习显示了如何使用一些选择器，以及如何对匹配集应用动画。在本章后面的部分中，将会更加详细地解释各种动画。

现在，只需集中讲述 jQuery 代码的选择器部分。

试一试

使用基本选择器

在这个练习中，首先向主母版页添加一个额外的 ContentPlaceHolder 控件，从而使得向页面添加客户端 jQuery 代码变得更加简单。然后编写一些 jQuery 来试验各种选择器。

(1) 打开 MasterPages 文件夹中的 Frontend.master 文件，并确保它位于 Markup 视图中。

(2) 在页面底部附近，结束标记 </form> 之前，从工具箱中拖出一个 ContentPlaceHolder 并放到这里。将其 ID 设置为 cpClientScript。代码应该如下所示：

```
<div id="Footer">Footer Goes Here</div>
</div>
<asp:ContentPlaceHolder ID="cpClientScript" runat="server">
</asp:ContentPlaceHolder>
</form>
```

(3) 保存并关闭母版页，因为现在不需要对它执行其他操作。

(4) 在 Demos 文件夹中创建一个新的演示页面，并将其命名为 BasicSelectors.aspx。同样，基于您自己的模板创建这个页面，并给它添加一个有意义的标题。将页面切换到 Design 视图中，在底部找到 cpClientScript 占位符，打开它的 Smart Task 面板，然后选择 Create Custom Content 命令。

(5) 切换到 Markup 视图，并将下面的 HTML 代码添加到 cpMainContent 占位符中(不要误把这些代码添加到刚才添加的占位符中)：

```
<h1>Basic Selectors</h1>
<div class="SampleClass">I am a div.</div>
```

(6) 将下面这些演示所有 6 个基本选择器的 jQuery 代码添加到第(4)步创建的 cpClientScript 占位符中。

```
<asp:Content ID="Content3" runat="server" ContentPlaceHolderID="cpClientScript">
<script type="text/javascript">
$(function()
{
    $('*').css('color', 'Green'); // Universal
    $('#Footer').css('border-bottom', '2px solid black'); // ID
    $('h1').bind('click', function() { alert('Hello World') }); // Element
    $('.SampleClass').addClass('PleaseWait').hide(5000); // Class
    $('#Footer, #Header').slideUp('slow').slideDown('slow'); // Grouped
    $('#Sidebar img').fadeTo(5000, 0.1); // Combined
});
</script>
</asp:Content>
```

(7) 在浏览器中保存全部改动和请求页面。现在，所有的文本都是绿色的，页脚的位置有一个额外的边框，还可以看到 Please Wait 动画图标显示然后消失，页眉和页脚缓慢消失然后重新显示，以及在 5 秒钟的时间段内，边栏的横幅变为几乎透明。如果单击 Basic Selectors 标题，将会弹出一个窗口显示 Hello World。

2. 工作原理

添加大量动画会使页面看上去很有趣。但是一般不推荐将所有这些功能同时添加到页面中，否则会吓跑大部分用户。但是，对于这个演示而言，这样做的效果却很好，因为可以从中看到jQuery的一些强大功能。您已经见过所有6个选择器，但是可能还不熟悉对它们的匹配集执行的代码。第一个选择器选择页面中的全部元素，然后应用css方法来将它们的字体颜色修改为绿色。然后ID选择器获取唯一的一个元素，并调用相同的css方法来应用边框。第三个示例使用元素选择器找出h1元素，然后动态绑定一个click处理程序，这样单击标题时，大括号之间的代码将会执行。

第4个选择器演示了类选择器，并说明了如何通过类名找到元素。注意搜索的CSS类并不一定是样式表中定义的现有CSS类。当找到元素后，addClass方法就会向它们添加一个新类，在本例中为PleaseWait，它将应用一个转盘图像作为div元素的背景。之后hide方法将在5秒钟的时间范围内再次隐藏它们。

第5个选择器是分组选择器，用来找出页脚和页眉。之后slideUp方法缓慢地降低这些元素的高度，直到它们完全消失。在这个过程中，它会记录这些元素的初始尺寸，从而当再次调用slideDown的时候，就知道把元素恢复到什么尺寸。

最后一个选择器使用一个合并选择器来找到右边栏的横幅图像。当它找到图像的时候，就会将其不透明度设置为0.1(10%)，从而逐渐地使它淡出(5秒内)以致于逐渐看不见它。

在本章后面的部分中，将会看到jQuery提供的更多样式和动画方法。现在只需要理解选择器语法，能够在页面中引用元素即可。

通常，能在页面中选择项还不够。例如，当选择表中的行时，您可能不希望一次选择所有的行，而是希望只选择奇数行或者偶数行，以便对表应用“斑马纹”效果，使奇数行和偶数行具有不同的颜色。这时候就需要筛选器发挥作用了。

3. 基本筛选器

在jQuery中，可以使用筛选器进一步过滤选择器得到的结果集。这就为您可以找到特定的元素带来了大量的可能性，比如第一个元素、最后一个元素、所有奇数行元素、所有偶数行元素、所有的标题或者特定位置的项。下一个“试一试”练习之后的表列出了最常用的基本筛选器，并给出了一个使用它们的示例。为了能够理解这些示例和后面的许多示例，需要动手完成这个练习，它为大部分jQuery示例建立了一个测试页面。

试一试

建立一个jQuery演示页面

在这个练习中，将创建一个全新的演示页面。借助于使用这个页面，可以试验本章中的许多示例，而且只需要替换一行代码。

(1) 基于定制模板创建一个新页面，命名为jQueryDemos.aspx。然后，在cpMainContent占位符中添加下面的HTML代码段：

```
<h1 title="First Header">First Header</h1>
<table id="DemoTable">
  <tr><td>Row 1 Cell 1</td><td>Row 1 Cell 2</td></tr>
  <tr><td>Row 2 Cell 1</td><td>Row 2 Cell 2</td></tr>
  <tr><td>Row 3 Cell 1</td><td>Row 3 Cell 2</td></tr>
```



```
<tr><td>Row 4 Cell 1</td><td>Row 4 Cell 2</td></tr>
<tr><td>Row 5 Cell 1</td><td>Row 5 Cell 2</td></tr>
</table>
<h2>Second <span style="font-style: italic; font-weight: bold;">Header</span></h2>
<input id="Button1" type="button" value="button" />
<input id="Text1" type="text" />
<input id="Checkbox1" type="checkbox" />
<input id="Checkbox2" type="checkbox" />
```

您不需要手动输入所有这些代码。相反，可以使用 VWD 编写其中大部分代码。一定要利用好 Table 菜单和工具箱的 HTML 分类。

(2) 在 cpMainContent 的下面为 cpClientScript 添加一个 Content 块，并输入下面的代码：

```
</asp:Content>
<asp:Content ID="Content3" runat="server" ContentPlaceHolderID="cpClientScript">
<script type="text/javascript">
    $(function()
    {
        // Examples go here

    });
</script>
</asp:Content>
```

(3) 使用下面的代码替换//Examples go here 一行，以便测试您的设置：

```
$('#DemoTable').css('background-color', 'green');
```

(4) 保存修改，并按下 F5 键在浏览器中打开该页面。如果一切正常，那么表的单元格的背景色应该变为绿色。

(5) 关闭浏览器，返回到 VWD 中。按下 Ctrl+Z 组合键撤消上一次修改，直到再次看到//Examples go here 一行，然后保存页面。

工作原理

在这个“试一试”练习中，创建了一个简单的 Content 块来保存 jQuery 代码。然后定义了一个在浏览器加载 DOM 完成后触发的代码块。在这个代码块中，您编写了一个简单的选择器来选择 id 为 DemoTable 的表，然后使用 jQuery 的 css 方法修改其背景色。

在表 11-1 中，列出了 jQuery 的基本筛选器。记住，可以使用表中给出的代码示例替换//Examples go here 一行，从而试验每个示例。然后保存页面，并在浏览器中加载页面以便查看代码的效果。

表 11-1

筛 选 器	用 途
:first :last	用于选择匹配集中的第一个和最后一个项。下面的示例将表的第一行和最后一行的背景色设置为红色： <pre>\$('#DemoTable tr:first').css('background-color', 'red'); \$('#DemoTable tr:last').css('background-color', 'red');</pre> 首先，使用#DemoTable 找到表。然后使用 tr 找到表的全部行。最后，使用:first 和:last 筛选器找到第一行和最后一行

(续表)

筛 选 器	用 途
:odd :even	用于选择匹配集中的奇数行或者偶数行。下面的示例将表的奇数行的背景色修改为红色。因为计数是从零开始的，所以实际上将会看到第 2 行和第 4 行的背景色发生了改变(因为它们的索引分别为 1 和 3)。 \$('#DemoTable tr:odd').css('background-color', 'red');
:eq(index) :lt(index) :gt(index)	按照索引匹配元素。:eq(equals)根据索引返回一个元素，而:lt(less than)和:gt(greater than)则分别返回小于或者大于给定索引的项。示例如下所示： // Changes the color in the first row (with an index of 0) \$('#DemoTable tr:eq(0)').css('color', 'green'); // Changes the last two rows (rows 1, 2 and 3, // with an index of 0, 1 and 2 respectively, are skipped.) \$('#DemoTable tr:gt(2)').css('color', 'green'); // Changes the text color of the first two rows to green. \$('#DemoTable tr:lt(2)').css('color', 'green');
:header	找到页面中的全部标题(从 h1 到 h6)。示例如下所示： \$(':header').css('color', 'green');

要想查看所有基本筛选器的完整列表,可以阅读jQuery 文档,网址为 <http://api.jquery.com/category/selectors/>。

4. 高级筛选器

除了刚才看到的基本筛选器以外, jQuery 还支持其他很多筛选器, 它们可以用来根据项包含的文本、是否可见、以及它们包含的任意属性获取项。另外, 还有一些筛选器可以获得表单元素(例如按钮、复选框、单选按钮等), 以及大量可以用来选择子元素、父元素、兄弟元素和后代元素的选择器。表 11-2 列出了最常用的筛选器。jQuery 的在线文档提供了完整的列表, 以及演示它们的工作原理的完整示例。

表 11-2

筛 选 器	用 途
:contains(text)	通过包含的文本匹配元素。示例如下： \$(td:contains("Row 3")).css('color', 'green'); 如果省略 td, 那么整个表都会变成绿色。这是因为表本身也会被匹配(它的一个子表包含文本 Row 3), 所以颜色将应用到整个表上, 从而使得每个单元格的文本变为绿色
:has(element)	匹配至少包含一个给出元素的元素。示例如下： \$(':header:has("span")').css('color', 'green'); 这将只匹配 h2, 因为它是标题(:header), 并包含 span 元素(has("span"))
[attribute]	基于给定属性匹配元素。示例如下： // Matches the button and the text box as both // have a type attribute \$('[type]').css('color', 'green'); 需要在文本框中输入一些文本来查看绿色的字体

(续表)

筛 选 器	用 途
[attribute=value]	基于一个属性和该属性的值匹配元素。示例如下： // Matches just the text box \$('[type=text]').css('color', 'green');
:input :text :password :radio :checkbox :submit :image :reset :button :hidden :file	这些选择器可以用来匹配特定的客户端 HTML 表单元素。例如，可以使用分组选择器把查找按钮和文本框的代码段重写如下： \$(':button, :text').css('color', 'green'); 可以使用这些筛选器来实现一些奇妙的效果。例如，要想编写一些功能来选中一个表单中的全部复选框，可以使用下面的代码： \$(':checkbox').attr('checked', true); 要想取消选择全部复选框，可以传递 false 作为 attr 方法的第二个参数

虽然这些功能很强大，但是如果不能操作它们的结果，那么它们也就没有什么实际价值。下一节将讨论如何修改匹配集中的项的外观和行为。

11.3 使用 jQuery 修改 DOM

有了匹配集之后，就需要对它执行一些操作。例如，您可能希望对匹配集中的项应用 CSS 类或者样式。或者，您可能希望向它们附加一些行为，如添加一个 click 处理程序，从而使得项被单击时可以触发一些代码。您已经看到了使用 css 和 bind 方法实现这种功能的示例，但是 jQuery 提供的功能远不止此。在接下来的两节中，您将看到如何操作各种 CSS 方法，并将学习如何建立事件处理程序。在随后的“试一试”练习中，将结合运用学到的知识，使用 jQuery 增强联系表单用户控件。

11.3.1 CSS 方法

jQuery 以几种不同的方式支持 CSS。首先，可以使用 css 方法来检索特定的 CSS 值(比如某个项的颜色)，以及设置一组元素的一个或多个 CSS 属性。其次，诸如 addClass、removeClass、toggleClass 和 hasClass 等方法可以用来修改或检查对元素应用的 CSS 类。再次，有几种方法可以用来修改元素的尺寸和位置。本书只是讨论了最常用的方法，但是可以在 jQuery 的 Web 站点查看完整的方法列表。这里的示例再一次使用了与前面相同的 HTML 代码段，所以如果您想要在自己的浏览器中尝试这些代码，就很容易理解这个示例。

css(name,value)
这个方法用来设置某个匹配元素上的特定的 CSS 属性。name 参数是引用一个 CSS 属性的名称

(比如 border、color 等)，value 定义了想要应用的样式。下面的示例修改了 h1 元素的背景色：

```
$('#h1').css('background-color', 'green');
```

css(name)

这个方法基于传递给它的属性检索特定的 CSS 值。下面的示例将提示'italic'，因为这是二级标题的 span 元素的 font-style。

```
alert($('#h2 span').css('font-style'));
```

可以在 jQuery 脚本中使用这个值；例如，它可以用来在斜体和普通字体之间切换 font-style，或者将多个元素设置为相同的类型。

css(properties)

这是一个功能强大的方法，因为它可以用来同时设置匹配元素的多个属性。下面的示例将表中所有单元格的颜色修改为红色，将它们的内边距设为 10px，并将它们的字体修改为 Verdana。注意每个属性和属性的值由冒号(:)分隔，而每个属性/属性值对之间由逗号分隔。完整的属性集包含在一对花括号({})之间：

```
$('#DemoTable td').css({'color' : 'red', 'font-family' : 'Verdana',  
                        'padding' : '10px'});
```

addClass、removeClass 和 toggleClass

addClass 和 removeClass 方法分别用来在元素中添加和删除类。和普通的 CSS 一样，使用这些方法，比使用 css(properties)方法进行内联 CSS 赋值更好。这样就更容易在一个集中的位置定义 CSS 类，从而使得它们更易于维护和重用。下一个示例显示了如何将异构类赋给 h2 元素：

```
$('#h2').addClass('PleaseWait');
```

如果希望再次删除类，则可以调用 removeClass 方法，如下所示：

```
$('#h2').removeClass('PleaseWait');
```

如果类还不存在，则 toggleClass 方法将分配一个类；否则，它将删除类。

所有这 3 个方法都允许传递多个类，各个类之间使用空格分隔。

结合使用这些 CSS 方法，就在修改页面元素的外观上拥有了极大的控制权。可以在此之上更进一步，使用 jQuery 丰富的事件系统，通过代码向元素分配和删除各种处理程序。

11.3.2 处理事件

事件是许多编程语言中常用的一种技术。在前面的章节中已经看到了.NET 事件的应用，您使用这些事件处理了 Button 控件的 Click 事件或 Page 的 Load 事件。在这一点上，JavaScript 和 DOM 也不例外，它们在许多地方都提供了事件。例如，许多 HTML 元素(比如使用 input type="button"定义的按钮)都有一个 click 事件，在单击的时候触发。同样地，它们还有 onmouseover 和 onmouseout 事件，当鼠标经过它们或者离开它们的时候触发。通常，当在标记中直接定义事件时，它们如下所示：

```
<input type="button" onclick="alert('Hello');" value="Click Me" />
```

不要编写内联触发的代码(在本例中为 alert 函数)，而是应该将它们指向自己可以编写的 JavaScript 函数。下面的示例调用了一个虚构的 SayYourName 函数：


```
<input type="button" onclick="SayYourName();" value="Click Me" />
```

在第 10 章中，您看到了 ASP.NET AJAX 框架的 `addHandler` 方法使这个任务变得更加简单、更加灵活，因为您可以在一个独立的代码块中建立处理程序。

jQuery 则更进一步，不仅仅允许将事件挂钩到单个元素上，还允许将事件挂钩到整个匹配集上。这种功能极为强大，因为只用几行代码，就可以将处理程序绑定到大量的元素上。例如，考虑具有许多行的一个表。为了使表格的外观美观一些，可以应用一种叫做“活动项跟踪”的技术，使得当鼠标移动到某个项上时，该项就改变颜色。如果不使用 jQuery，就需要对表的每一行编写 `onmouseover` 和 `onmouseout` 事件。这显然会显著增加页面最终的 HTML 代码量。而使用 jQuery，则只需要使用下面的代码(这里仍然使用了前面的 HTML 代码示例)：

```
$(function()
{
    $('#DemoTable tr')
        .bind('mouseover', function() { $(this).css('background-color', 'yellow') });
});
```

这些代码找出 `#DemoTable` 元素中的全部表行，然后动态分配一个函数，当鼠标悬停在每一行上时，将会调用该函数。如果将鼠标悬停在行上，那么背景将会改变颜色。但是如果移走鼠标，那么新颜色将会保留下来。为了解决这个问题，可以使用 jQuery 的链接(chaining)概念，即 jQuery 方法的结果会返回一个匹配集，所以可以对该结果应用其他函数。要将 `onmouseout` 绑定到一个新函数，只需要对 `bind` 的第一次调用返回的值再次调用 `bind`：

```
$('#DemoTable tr')
    .bind('mouseover', function() { $(this).css('background-color', 'yellow') })
    .bind('mouseout', function() { $(this).css('background-color', '') });
```

注意，在前一个示例中结束行的分号移动到了最后一行。这样，第二个 `bind` 就绑定到了前一次对 `bind` 的调用上。这一点不太容易看出来，因为在这里由于本书的宽度原因，代码放到了两行中，但是这些代码实际上等同于如下所示的代码：

```
$('#DemoTable tr').bind('mouseover', ...). bind('mouseout', ...);
```

这些代码完成三项工作：首先，使用 `$('#DemoTable tr')` 找出表中的全部行。它在返回的匹配集中调用 `bind`，以便动态挂钩一些行为，当鼠标移动到某一行上时，就会触发这些行为。然后，对第一次调用 `bind` 返回的匹配集再次调用 `bind`，以便当鼠标从该行移走的时候重设背景色。注意，代码中将颜色设置为一个空字符串('')，以便删除 CSS 背景属性，这样就可以再次看到原来的背景。

在这个示例中，还有一个重点地方需要注意，即设置背景色的方式：

```
$(this).css('background-color', 'yellow')
```

在本例中，`this` 关键字指的是应用该项的元素：在本例中就是表行。使用 `$(this)` 将得到 jQuery 匹配集(包含单个元素)，可以对其应用常规的 jQuery 方法，比如 `css` 方法。或者也可以不是以 jQuery 而是对 `this` 元素执行标准的 JavaScript 方法。

```
this.style.backgroundColor = 'yellow'
```

表行(和其他许多 HTML 元素)都有一个 `style` 属性，可以用来通过编程方式修改 CSS 样式。您

可能以为应该使用 `style.background-color` 来修改颜色，但是在 JavaScript 中并非如此。在 JavaScript 中，短划线(-)不是一个有效的标识符，所以在 JavaScript 中，所有的短划线都将从属性名中删除。而且，原来紧跟在短划线后面的字母将变为大写方式。所以，CSS 中的 `background-color` 在 JavaScript 中就变成了 `backgroundColor`，`font-family` 就变成了 `fontFamily`，等等。当试图通过 JavaScript 和 jQuery 动态设置 CSS 信息时，一定要牢记这些命名规则。

11.3.3 jQuery 的各种功能

在进入下一个“试一试”练习并实际应用新学到的所有 jQuery 的知识之前，还需要了解 jQuery 中的几个重要的函数。首先看 `each`。

`each` 方法迭代(或循环遍历)一个集合。当需要对匹配集中的项应用某种行为，但是无法使用一个 jQuery 函数完成设置时，使用 `each` 方法是一种非常好的选择。需要把希望对每一项执行的函数作为参数传递给 `each`。下面的这个 `each` 示例通过循环遍历匹配集中的每一项，然后调用 `alert`，将每个表单元格的内容显示了出来。同样，可以使用 `jQueryDemos.aspx` 页面试验这些代码。

```
$('#DemoTable td').each(function()
{
    alert(this.innerHTML);
});
```

另外两个重要的方法是 `parent` 方法和 `prev` 方法。这些方法用在 DOM 遍历中，允许在文档树中上下移动，找出低于、高于某一项或者与该项位于同一层次的元素。

`prev` 方法选择匹配元素的直系兄弟。其工作原理如下面的代码所示：

```
alert($('#DemoTable td:contains("Row 1 Cell 2")').prev()[0].innerHTML);
```

这个警报的结果就是“Row 1 Cell 1”。\$选择器首先根据其包含的文本选择了第一行的第二列的表单元格。之后 `prev` 方法返回它的第一个兄弟：其左侧的单元格。因为匹配集是一个集合，即使它只包含一项也是如此，所以仍然需要使用索引器(使用[0])来引用第一项。然后表单元格将显示一个 `innerHTML` 属性，它将返回该单元格的内容。

最后，看一下 `parent`：

```
alert($('#DemoTable td:contains("Row 1 Cell 2")').parent()[0].innerHTML);
```

如果在演示页面中运行这行代码，您将得到如图 11-3 所示的结果。

如果没有看到完全相同的 HTML，那么要确保以 `alert` 开头的一行是文档就绪函数中唯一的一行，因为其他示例可能影响到表的 HTML。

选择器与前一示例相同。`parent` 函数指向围绕匹配的表单元格的 `<tr>`。警报 `innerHTML` 将返回该行包含的两个单元格的 HTML。

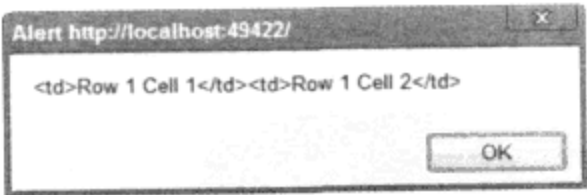


图 11-3

11.3.4 使用 jQuery 时常犯的错误

取决于使用的 jQuery 功能的复杂程度，编写代码可能十分具有挑战性。您需要正确地使用小括号、中括号、逗号等。当发现 jQuery 代码无法运行或者表现出异常行为时，检查代码是否存在下面

列出的问题。

1. ID 选择器无效

很可能是忘记了在它的前面包含一个#符号。`$(‘DemoTable’)`并不会选择 id 为 DemoTable 的表，而`$(‘#DemoTable’)`才会选择该表。ID 选择器无效的另外一种情况是大小写有误。ID 选择器是区分大小写的，所以`$(‘#DemoTable’)`与`$(‘#demotable’)`是不同的。

2. 即使使用#符号，ID 选择器依然无效

出现这种情况时，很可能是客户端元素没有您希望查找的 id。是不是 ASP.NET 运行库修改了客户端 id，使用它的父元素的 id 给它添加了前缀？如果是这样，尝试将相关控件的 ClientIDMode 设置为 Static，这样 id 的可预测性更好。另外，确保代码在文档中的 ready 函数中执行，因为很可能您希望查找的元素还不可用。

3. 没有代码可以运行

检查小括号、大括号和引号。每种符号的开始符号和结束符号的数量必须相等。
在接下来的“试一试”练习中，将会使用许多在前面介绍过的 jQuery 选择器、筛选器和方法。

试一试 结合使用

用户可以填写的表单常常具有水印，其中每个文本框中通常以灰色或者斜体显示出关于应该输入什么内容的说明。图 11-4 显示了一个带有水印的表单。当用户把焦点放到该字段上时，这些文本就将消失。当使用传统的 JavaScript 时，向表单中的每个文本字段添加这种文本是一项繁重的任务。但是使用 jQuery 则可以轻松地完成这项工作，如下所示。

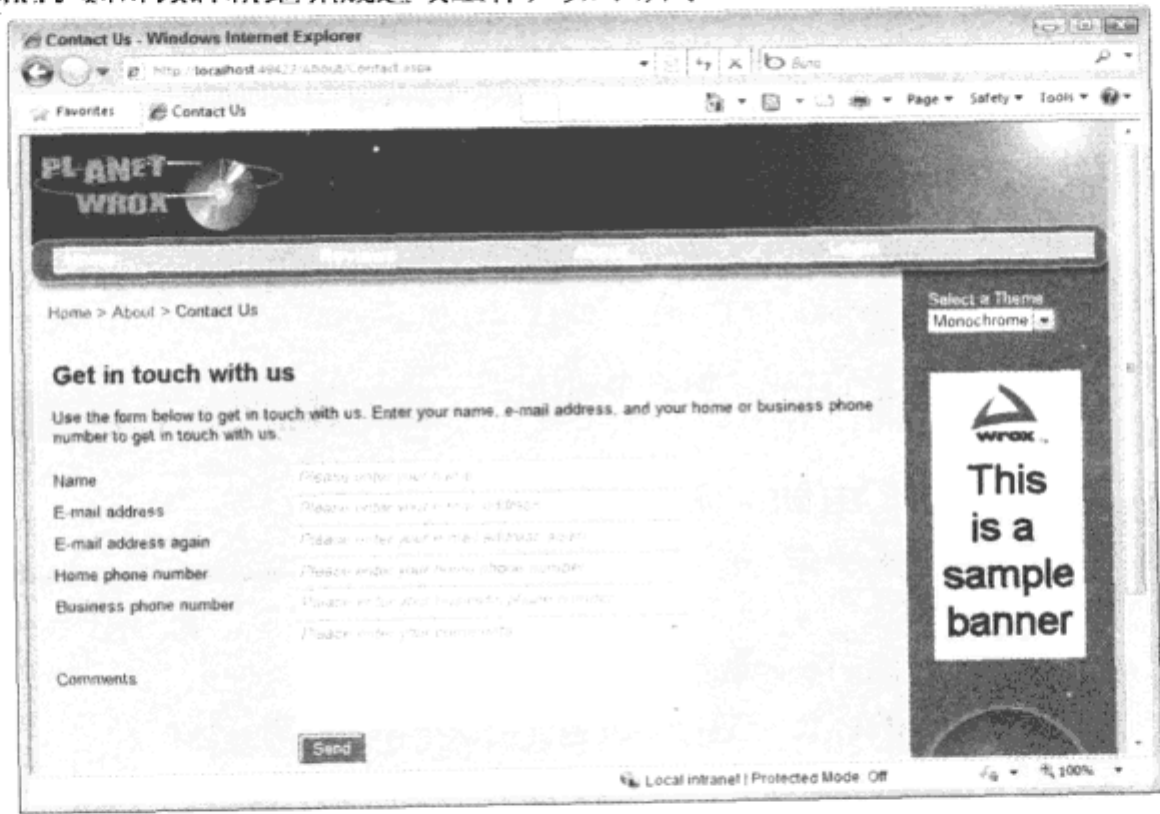


图 11-4

(1) 打开 Controls 文件夹中的用户控件 ContactForm.ascx，并确保它位于 Markup 视图中。

(2) 在顶部的<style />块中，添加下面的声明：

```
.Watermark
{
    font-style: italic;
    color: Gray;
}
```

不要直接向用户控件添加这个选择器，而是将其添加到每个主题的 CSS 文件中。如果您仍然那么做，那么不要忘记在两个 CSS 文件中都添加它。对于本例来说，添加到什么地方并不重要。

(3) 滚动到控件的标记的末尾，在 UpdateProgress 结束标记以下，使用下面的 jQuery 代码添加一个<script>代码块：

```
<script type="text/javascript">
$(function()
{
    $(':input[type=text], textarea').each
    (
        function()
        {
            var newText = 'Please enter your ' +
                $(this).parent().prev().text().toLowerCase().trim();
            $(this).attr('value', newText);
        }).one('focus', function()
        {
            this.value = '', this.className = ''
        }).addClass('Watermark').css('width', '300px');
    });
</script>
```

看上去很奇怪，但是不用担心。在后面的“工作原理”部分，将逐行解释这些代码。如果不希望输入所有这些代码，那么可以从本书附带的源代码中进行复制。

(4) 保存修改，并按下 Ctrl+F5 组合键。您将被带到站点的主页，因为在浏览器中不能直接请求用户控件。从 Menu 控件(当使用单色主题时)或者 TreeView 中单击 Contact Us 项。您将看到如图 11-4 所示的页面。

(5) 单击 Name 字段，将光标放到该字段中。注意，原来显示的文本将会消失，如图 11-5 所示的 Name 字段。



图 11-5

(6) 完成表单，并单击 Send 按钮。邮件将会正常地发送(或者存储在临时邮件文件夹中)。

工作原理

就现在而言，除了看上去很复杂的 jQuery 操作以外，本例中使用的操作基本上都是标准操作。但是如果分析这些操作，那么它并不像看上去那么复杂。下面将逐行讨论这些代码，并解释其工作原理。

```
$(function()
{
    ...
});
```

首先是标准的文档就绪函数。您已经多次看到这个函数。在它定义的函数中，代码首先获得所有文本框和文本区域的匹配集，然后调用匹配集的 each 方法：

```
$('.input[type=text], textarea').each
(
    function()
    {
        var newText = 'Please enter your ' +
            $(this).parent().prev().text().toLowerCase().trim();
        $(this).attr('value', newText);
    })
```

注意，这里使用属性选择器(使用 input[type=text]只选择文本框，而没有选择按钮。因为注释字段将在浏览器中作为<textarea>显示，而不是作为<input type="text" />显示，所以也需要在选择器中包含它，这可以通过使用分组选择器很简单地完成。

each 方法内的代码将对匹配集中的每一项进行调用。在本例中，这意味着它们将被调用 6 次：表单中的每个文本框和 textarea 一次。在 each 方法内，可以使用 this 指针，它指向当前迭代的项。在第一次迭代中，就是 Name 文本框，而在第二次迭代中则变成了 e-mail address 文本框，以此类推。

这些代码的思想是，文本框最终将显示文本框前面的标记中的文本，并在前面加上文本“Please enter your”作为前缀。这是通过下面的代码实现的：

```
var newText = 'Please enter your ' +
    $(this).parent().prev().text().toLowerCase().trim();
```

首先，\$(this)用来引用当前的文本框，正如在前面动态修改表单元格的背景色时那样。代码对这些结果调用 parent，从而得到包含文本框的表单元格的引用。prev 方法返回该表单元格的前一个包含文本的兄弟，也就是它前面的单元格。最后，text().toLowerCase().trim()得到单元格的文本(例如 Name)，将它们转换为小写形式，然后调用 trim()来删除周围的空格。如果感到困惑，就看一下图 11-6，图中显示了这个过程的 4 个阶段，以及每一步选择了什么。



图 11-6

each 代码块的最后一行将新文本赋予文本框的 value 属性：

```
$(this).attr('value', newText);
```

最终结果是文本框现在包含类似于 “Please enter your name” 这样的文本。

和其他方法一样，each 返回匹配集，从而使得对该匹配集调用更多的 jQuery 代码变得简单。one 方法正是利用了这一点：

```
$('.input[type=text], textarea').each(
  function()
  {
    ...
  }).one('focus', function()
  {
    this.value = '', this.className = ''
  }).addClass('Watermark').css('width', '300px');
```

与前面用来挂钩按钮的单击事件处理程序的 bind 类似，one 动态地将一个函数分配给元素的一个事件。one 的一个真正的优点在于，当第一次调用它之后，它就会再次自动删除处理程序。对于本例来说，这一点很有用，因为您并不想在用户每次单击一个文本框的时候就清空文本框。因为如果是这样，那么当用户输入实际数据并且把焦点再次放到该字段以便进行修改的时候，就无法清空字段。

one 函数用来将 focus 事件(当单击文本字段或者使用 Tab 键切换到文本字段时触发)挂钩到某些代码以便清空文本字段的值，然后将 CSS 类动态地重设为一个空字符串。您可能认为 this.class 也可以起到同样的效果。但是，class 是 JavaScript 中的保留字，所以要引用元素中的 CSS class 属性，需要使用 JavaScript 中的 className。

同样，one 返回一个匹配集，可以用来链接另外一个方法 addClass，后者将 Watermark CSS 类分配给全部匹配的元素以修改字体外观。注意，只有在页面第一次加载的时候才会发生这种情况。之后，当控件再次获得焦点时，它的类会被重设，但是不会再次被分配 Watermark。

最后一个链接的调用是 css 方法，它将所有输入控件的 width 设为 300 像素。并不是一定要这么做，而且使用普通的 CSS 可能效果更好，但是对于这个演示来说，这是确保所有控件具有相同宽度并且这个宽度足以容纳它们包含的文本的一种便捷的方法。后面的一个“试一试”练习将通过分配一个合适的 CSS 类来解决这个问题。

虽然这个示例很好地介绍了 jQuery 的诸多功能和特性，但是在一个实际的站点中可能并不希望使用这些代码，因为现在的这些代码具有以下的问题：

- 如果文本框已经包含一个 CSS 类，那么当用户单击文本框时就会清空它。可以通过还原焦点处理程序中的类来解决这个问题，但是这并不是理想的解决办法。
- 这个方法依赖于控件前面包含逻辑文本的标记。如果修改 Comments 标记，使其读作“Feel free to let us know what you think of this site”，那么就会得到一个难看的水印文本。最好能够根据各个控件定义要显示的文本。
- 通常，如果控件丢失焦点并且用户还没有输入文本，那么水印将会重新显示在文本上。但是在本例中却并非如此。
- 客户端验证不会被正确地触发，因为验证框架认为必填字段已经填充。

幸运的是，jQuery 提供了另外一种优秀的方法，可以通过当前的水印来解决这些问题。这种方法就是插件，在讨论动画之后，本章将会介绍插件。

11.4 使用 jQuery 的效果

在本章前面的一个示例中，您看到了如何使用 slideUp 和 slideDown 来逐渐地隐藏和显示元素。但是这只是 jQuery 提供的诸多效果和动画方法中的两种方法。表 11-3 列出了最常用的一些方法。同样，建议您使用 jQueryDemos.aspx 页面试验所有这些示例。

表 11-3

方 法 名	用 途
show() hide()	通过递减 height、width 和 opacity(使它们变为透明)隐藏或者显示匹配元素。两种方法都允许定义固定的速度(慢、中、快)或者一个定义动画持续时间(单位为毫秒)的数字。示例如下： <pre>\$('#h1').hide(1000); //Hide the heading in 1 second \$('#h1 蒙古自治区').show(1000); //Make it reappear in 1 second</pre>
toggle()	toggle 方法在内部使用 show 和 hide 来改变匹配元素的显示方式。即，可见元素将被隐藏，不可见元素将会显示。示例如下： <pre>\$('#h1').toggle(2000); // hide or show the h1 in 2 seconds</pre>
slideDown() slideUp() slideToggle()	类似于 hide 和 show，这些方法隐藏或显示匹配元素。但是，这是通过将元素的 height 从当前尺寸调整为 0，或者从 0 调整为初始尺寸来实现的，所以会"slide up"或"slide down"元素。slideToggle 方法会展开隐藏的元素，卷起可见的元素，从而可以使用一个动作重复地显示和隐藏元素。示例如下： <pre>\$('#h1').slideUp(1000); \$('#h1').slideDown(1000); \$('#h1').slideToggle(1000);</pre>
fadeIn() fadeOut() fadeTo()	这些方法通过修改匹配元素的不透明度显示或隐藏它们。fadeOut 将不透明度设置为 0，使元素完全透明，然后将 CSS display 属性设置为 none，从而完全隐藏元素。fadeTo 允许指定一个不透明度(0 到 1 之间的一个数字)，以便决定元素的透明程度。全部 3 个方法都允许定义一个固定速度(慢、中、快)，或者一个定义了动画持续时间(单位为毫秒)的数字。示例如下： <pre>\$('#h1').fadeOut(1000); // dissolve the h1 in 1 second \$('#h1').fadeIn(1000); // h1 reappears in 1 second \$('#h1').fadeTo(1000, 0.5); // fade to semi-transparent</pre>

(续表)

方 法 名	用 途
animate()	<p>在内部，<code>animate</code> 用于许多动画方法，例如 <code>show</code> 和 <code>hide</code>。但是，也可以在外部使用它，从而使您可以更灵活地以动画方式显示匹配元素。通过使用 <code>animate</code> 方法，可以指定许多动画显示的属性。考虑下面这个示例：</p> <pre>\$('h1').animate({ opacity: 0.4, marginLeft: '50px', fontSize: '50px' }, 1500);</pre> <p>这段代码接受一个 <code>h1</code> 元素，将其字体大小设置为 50 像素，将其不透明度设置为 0.4 以使元素半透明，并将其左页边距设置为 50 像素，从而在 1.5 秒钟的时间内平滑地进行动画显示。<code>animate</code> 方法的第一个参数是一个对象，它保存一个或者多个想要动画显示的属性，每个属性之间都以逗号分隔。注意，需要使用 JavaScript 的 <code>marginLeft</code> 和 <code>fontSize</code>，而不是 CSS <code>margin-left</code> 和 <code>font-size</code> 属性。只能动画显示接受数值的属性。也就是说，可以使用 <code>margin</code>、<code>fontSize</code>、<code>opacity</code> 等属性，但是不能使用 <code>color</code> 或 <code>fontFamily</code> 这样的属性</p>

现在，从简单的选择器到复杂的动画选项，您已经看过了最重要的 jQuery 概念，接下来将使用这些知识来使 Planet Wrox 站点的联系表单更具吸引力。

试一试

动画显示联系表单

在这个练习中，将应用两个主要的动画：一个动画在用户提交联系表单时触发，它会逐渐地卷动表单直至其隐藏起来。另一个动画用来显示和隐藏 Message Sent 标记，从而吸引更多注意直至动画完全消失。

(1) 首先，在 Markup 视图下的 `ContactForm.ascx` 控件中的 Message Sent 标记下面添加一个额外的文本段落。这个段落将在表单提交后显示，并且保持可见，即使 Message Sent 文本已经隐藏起来。在段落标记中添加一些文本，以感谢用户的响应。在段落中添加 `id` 和 `runat="server"` 属性(这样就可以在 Code Behind 文件中针对它进行编程)，并设置其 `Visible` 属性为 `False`。最后，将段落前面的 Label 控件中的 `CssClass` 属性设置为 `Attention`。最后得到的代码如下所示：

```
<asp:Label ID="Message" runat="server" CssClass="Attention" Text="Message Sent"
Visible="False" />
<p runat="server" id="MessageSentPara" visible="False">Thank you for your message.
    We'll get in touch with you if necessary.</p>
```

(2) 将保存表单控件的整个表包装到 `<div>` 中，并设置其 `id` 为 `TableWrapper`。使用 `slideUp` 和 `slideDown` 调整表的大小并不简单，但是通过把表包装到 `div` 中，就可以调整这个元素的大小：

```
<div id="TableWrapper">
<table class="style1" runat="server" id="FormTable">
...
</table>
</div>
```


(3) 在用户控件的顶部，@ Control 指令的下面，添加如下代码：

VB.NET

```
<% If False Then %>
<script src="../../../Scripts/jquery-1.4.1-vsdoc.js" type="text/javascript"></script>
<% End If %>
```

C#

```
<% if (false) { %>
    <script src="../../../Scripts/jquery-1.4.1-vsdoc.js" type="text/javascript"></script>
<% } %>
```

按照设计，VWD 不能为用户控件中的 jQuery 和其他脚本库提供 IntelliSense 功能。这是因为它不知道控件将显示在哪个页面或者母版页中。因为脚本引用是添加到这些页面中的，所以用户控件不知道它们的存在。前面的代码使用了一种新颖的技巧。VWD 认为您在向这个用户控件添加一个 jQuery 引用后，它会跟随这个链接并读取文档，从而提供 IntelliSense。但是在运行时不会包含文件，因为它被包装在一个总是返回 false 的 if 语句中。注意，这会在 Error 列表中产生一个关于不可达代码的警告，但是可以安全地忽略这个警告。使用这个 if 块的原因在于代码会变得不可达，从而使得浏览器中的最终 HTML 无法引用这个文件。

(4) 向下滚动到控件标记的末尾，在水印代码最后一行的下面添加如下的 jQuery 代码，这些代码在表单即将提交时触发：

```
}).addClass('Watermark').css('width', '300px');
$('#form').bind('submit', function()
{
    if (Page_IsValid)
    {
        $('#TableWrapper').slideUp(3000);
    }
});
```

不要重写现有文档就绪函数的结束代码});，而是将这些代码添加到现有的代码块中。

(5) 在 jQuery 文档就绪函数的结束</script>标记之前，结束大括号、小括号和分号之后，添加下面突出显示的 JavaScript 代码：

```
});
function pageLoad()
{
    $('.Attention').animate({ width: '600px' }, 3000).
        animate({ width: '100px' }, 3000).fadeOut('slow');
}
</script>
```

pageLoad 方法(它是客户端 ASP.NET AJAX Library 的一部分)的用途与 jQuery 的文档就绪方法相同，但是有一个例外：当发生部分页面更新后，它也会触发。而由于用户控件中的 UpdatePanel 方法，当您提交表单后就会发生部分页面更新。

(6) 切换到 Code Behind 文件，并添加下面的代码行，当提交表单和发送电子邮件消息后，就会显示如下所示的文本段落：

VB.NET

```
Message.Visible = True
MessageSentPara.Visible = True
FormTable.Visible = False
```

C#

```
Message.Visible = true;
MessageSentPara.Visible = true;
FormTable.Visible = false;
```

- (7) 保存全部修改并关闭用户控件，因为现在已经完成了对它的处理。
- (8) 打开主题文件夹中的 Monochrome.css 文件，并在文件的底部添加下面的 CSS 声明：

```
.Attention
{
    border: 4px solid red;
    padding: 10px 0;
    width: 100px;
    margin: auto;
    display: block;
    text-align: center;
}
```

- (9) 重复前一个步骤，但是现在向 DarkGrey 主题的 CSS 文件添加相同的声明。

- (10) 保存修改，然后在自己的浏览器中请求 Planet Wrox 站点。从 Menu 或 TreeView 控件中选择 Contact Us 项，填写表单，然后单击 Submit 按钮。注意，在单击按钮后不久，表单就会逐渐卷起，直到完全消失。图 11-7 显示了 slideUp 操作进行到一半的表单。



图 11-7

- (11) 当页面完成加载后，就会显示 Message Sent 标记和感谢文本。注意，标记的文本逐渐增加，直到占据整个内容宽度，然后再次收缩，直到完全消失。图 11-8 显示了正在调整大小的消息。

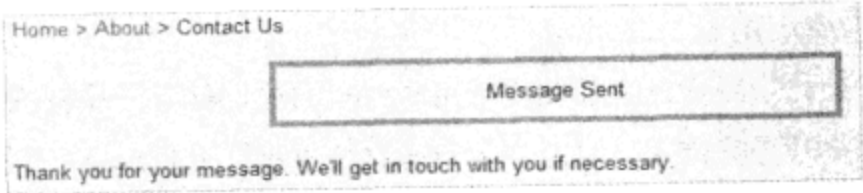


图 11-8

工作原理

这个“试一试”练习中包括的许多步骤处理的都是已经见过的操作：添加 div 元素、类和 CSS 声明。但是，在本例中还是有一些 jQuery 和一些 Ajax 代码值得注意。首先，查看添加到 jQuery 文

档就绪函数中的代码：

```
$( 'form' ).bind( 'submit', function()
{
    if (Page_IsValid)
    {
        $( '#TableWrapper' ).slideUp( 3000 );
    }
});
```

第一行动态地将一些代码绑定到表单的 submit 事件上。这样，当用户单击 Send 按钮时，其余代码就会触发。检查 Page_IsValid 很有必要，这样可以避免用户在某个地方出错时卷起表单。例如，用户可能忘记填写电话号码，但是却单击了 Send 按钮。此时，这个按钮会尝试提交表单，从而触发表单的 submit 事件。ASP.NET 客户端框架截获该事件并验证表单，从而显示一个带有错误消息的警告框。但是，即使表单无效，它仍然可以处理其他提交事件处理程序，包括您的代码建立的处理程序。如果不提前采取措施，那么不管是否有效，表单都会卷起。这样，用户就无法完成表单。但是，可以检查 Page_IsValid，当表单包含无效数据时，它就被设置为 false。只有当 Page_IsValid 返回 true 时，slideUp 方法才会隐藏表单。在 3 秒钟(3000 毫秒)的时间段内，表单会平滑地卷起，直到不可见。

然后，服务器代码将会运行并发送消息，如第 10 章所述。当发送电子邮件消息后，服务器代码就会发送回 Message Sent 标记和文本段落。之后，这个消息将会通过代码动画显示，代码中结合使用了 ASP.NET AJAX 和 jQuery：

```
function pageLoad()
{
    $( '.Attention' ).animate( { width: '600px' }, 3000 ).
        animate( { width: '100px' }, 3000 ).fadeOut( 'slow' );
}
```

本例中没有使用 jQuery 的 \$(function(...)) 在页面加载的时候触发代码，而是使用 ASP.NET AJAX 的 pageLoad，这有一个特殊的原因：这个事件是页面在第一次加载时，以及每次回发(不管是否是部分回发)后由 Ajax 框架触发的。这一点很重要，因为只有当单击按钮造成部分回发后，Message Sent 文本才会变为可用。注意，pageLoad 也会在第一次请求 ContactUs.aspx 时触发。但是，在这种情况下，页面中没有 Label 具有 Attention 类，所以 \$('.Attention') 会产生一个空匹配集，从而不会显示动画。

这里执行的代码相对简单。首先，通过使用 animate({width:'600px'},3000)，动画显示的消息将是 600 像素宽。动画的持续时间为 3 秒。3 秒以后，另一个链接方法就将消息设置回 100 像素宽。最后，fadeOut 方法用来淡出消息，之后消息就将完全消失。

虽然一些代码看上去十分复杂，但是使用 jQuery 在页面上应用一些特殊效果仍然相对简单。不能对所有的页面或表单滥用这些技术，但是当合理使用它们时，jQuery 动画确实可以让 Web 站点熠熠生辉。

如果现在您已经认为 jQuery 实在太棒了，那么我完全同意。但是 jQuery 的功能远不止如此。jQuery 社区积极地开发出了数百种实用的插件，可以毫不费力地把它们添加到页面中。在下一部分，我们将讨论从哪里获得这些插件，并展示了如何实现这样的一个插件，用来替代并增强本章前面实现的水印效果。

11.5 扩展 jQuery

插件确实使 jQuery 更加强大。为了编写重复编写相同的功能，jQuery 提供了一个灵活的插件体系结构，使得插件作者(也可以是您本人)编写的功能可以很容易地通过包含一个或多个 JavaScript 文件以及调用一个或者多个方法进行重用。

jQuery 插件非常流行，以至于 jQuery 站点专门用了一部分空间来提供插件：<http://plugins.jquery.com/>。在编写本书时，有许多类别(例如菜单、Ajax 功能、表单和拖放功能)的几百种插件可用。除了 jQuery.com 上的插件存储库以外，Internet 上还有其他许多插件。

通常，包含和使用 jQuery 插件涉及到以下 4 个步骤：

- (1) 进入 <http://plugins.jquery.com/>，找到并下载想要使用的插件。
- (2) 在项目中包含插件。在 Planet Wrox 网站中，这意味着将下载的.js 文件添加到 Scripts 文件夹中。
- (3) 在一个页面中添加对插件文件的引用。如果需要大量使用该插件，则把它添加到母版页中。否则，将插件添加到需要使用它的页面或者用户控件中。不管把插件添加到哪里，都要确保最终的 HTML 在加载主 jQuery 库之后加载它。
- (4) 编写一些代码来使用插件。具体的代码取决于使用的插件。一般可以在线找到插件的示例或者文档，或者插件可能提供了 readme 文件。

在接下来的“试一试”练习中，将执行这些步骤，包含 Ting Zwei Kuei 开发的 Watermark 插件。可以在 <http://plugins.jquery.com/project/updnWatermark> 上找到该插件。在本章的第一个“试一试”练习中，您已经把本例使用的插件文件添加到了站点中，所以没必要再次下载它。

试一试

使用插件扩展 jQuery

在这个练习中，将使用现成的插件代替在前一个“试一试”练习中编写的水印代码。本例中需要执行的操作是在用户控件中包含插件文件，添加一些 CSS，然后调用一个方法。另外，还需要提供想要在控件中显示的文本。其他所有功能都将得到自动处理。

- (1) 再次在 Markup 视图中打开 ContactForm.ascx，找到</UpdateProgress>结束标记。在 Solution Explorer 中的 Scripts 文件夹中，将文件 jquery.updnWatermark.js 直接拖动到 UpdateProgress 结束标记的后面，以便像下面这样插入一个指向.js 文件的 script 元素：

```
</asp:UpdateProgress>  
<script src="../../Scripts/jquery.updnWatermark.js" type="text/javascript"></script>
```

- (2) 删除在前面的“试一试”练习中为水印功能编写的代码。需要删除所有以\$(‘:input [type=text],textarea’).each 开头的元素，直到(包括)对 addClass 的调用。不要误删卷起表单的代码。

```
$(‘:input[type=text], textarea’).each  
... // rest of the code is omitted  
}).addClass(‘Watermark’).css(‘width’, ‘300px’);
```

- (3) 下面列出了对插件的 updnWatermark 方法的 jQuery 调用。使用这些代码替代前面删除的代码：


```
$.updnWatermark.attachAll({ cssClass: 'Watermark' });
```

(4) 滚动到文件的顶部，直到看到访问者姓名的 TextBox 控件。将其 ToolTip 属性设置为想要在文本框中显示的水印文本。插件会选择这些文本(最终在客户端中包含在 title 属性中)，并将其显示在控件中。在这里，向设置为 InputBox 的 TextBox 添加 CssClass:

```
<asp:TextBox ID="Name" runat="server" ToolTip="Enter your name"
    CssClass="InputBox"></asp:TextBox>
```

(5) 为页面中的其他 5 个 TextBox 控件(包括用于备注的字段)重复操作这个步骤，每次在 ToolTip 属性中提供一个逻辑文本，并设置 CssClass 属性。

其实不必逐个控件地设置 CssClass 属性。可以在 Design 视图中选择全部文本控件(使用 Ctrl+单击)，然后在 Properties 面板中为全部控件立刻设置该属性。

(6) 从 Comments 文本框中删除 Width 属性，因为将使用 CSS 设置它的宽度。

(7) 向上滚动控件，并删除前面添加的.Watermark CSS 声明。

(8) 保存修改，然后关闭用户控件文件。

(9) 向两个主题文件中添加下面的 3 个 CSS 声明。

```
.Watermark
{
    position: relative;
    width: 0;
    height: 1.25em;
    vertical-align: top;
}

.Watermark label
{
    position: absolute;
    left: 0;
    top: 2px;
    white-space: nowrap;
    color: #999;
    padding-left: 4px;
    height: 1.25em;
    vertical-align: middle;
}

.InputBox
{
    width: 300px;
}
```

如果您已经对向两个主题文件添加相同的 CSS 声明感到厌倦，那么可以考虑添加存储在 Styles 文件夹添加中的一个集中 CSS 文件。然后在母版页中链接这个样式表，使两个主题都可以使用它。这样只需要管理共享样式一次。在第 14 章将介绍如何实现这个目标，所以可以将这种修改推迟到第 14 章。

(10) 保存全部修改，然后在浏览器中打开 Planet Wrox 站点。导航到 ContactUs.aspx 页面，注意水印文本已经添加到了控件中，如图 11-9 所示。

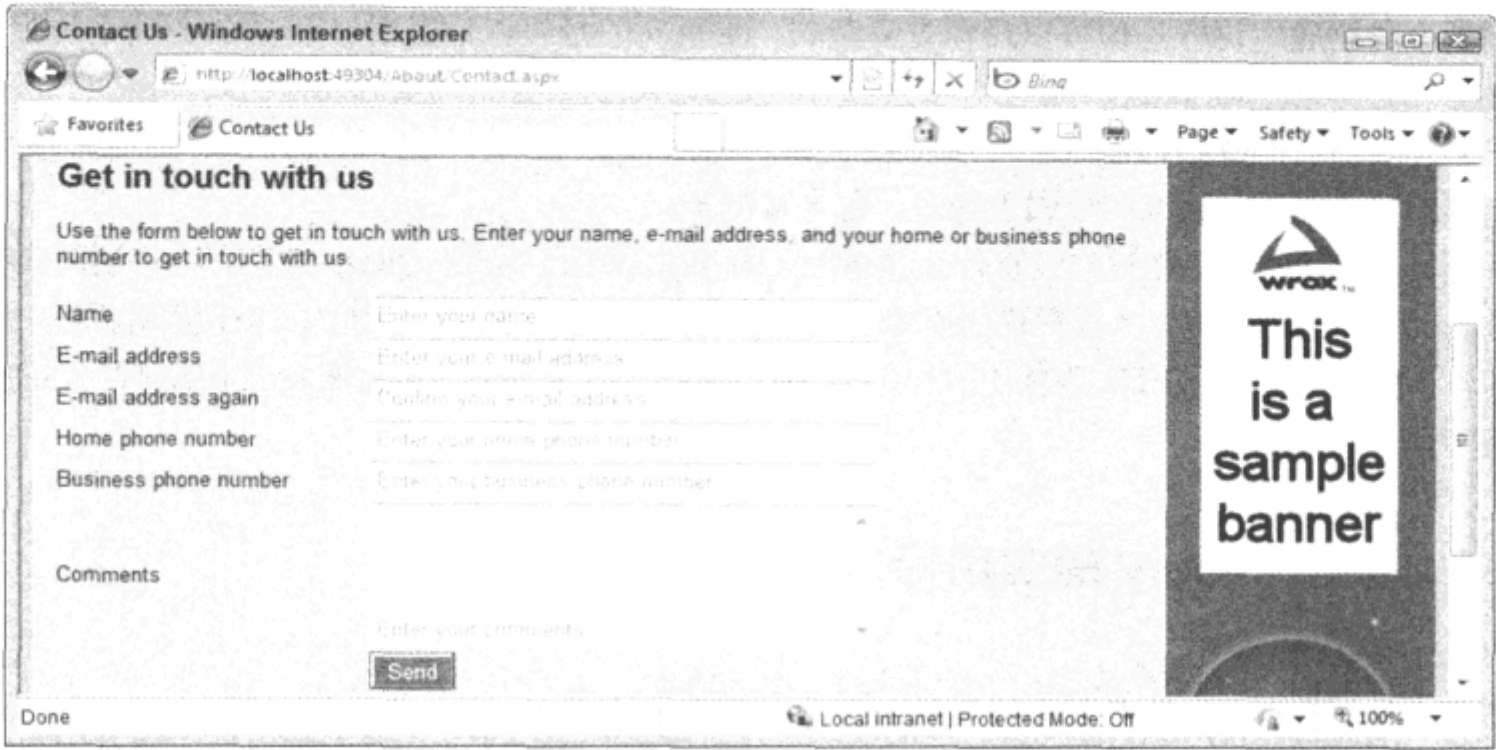


图 11-9

单击控件，然后文本将会消失。不在控件中输入值并按下 Tab 键，文本将会重新显示。最后，完成全部必要字段，然后单击 Send 按钮，以确认表单的行为仍然与前面相同。

工作原理

这个“试一试”练习让人激动的地方在于，我们不必关心它的工作原理。只需要使用插件，执行代码，然后就可以完成练习。就像您不必理解 Visual Web Developer 或者 jQuery 的内部工作原理就可以使用它们一样，通常也不需要知道 jQuery 插件的工作原理。但是，有一点十分重要，即您必须能够阅读和理解支持文档，并且能够正确地使用插件。

在本例中，操作都十分简单。要想使插件将水印文本添加到文本框中，需要设置想添加水印的每个控件的 ToolTip。标记中的 ToolTip 属性将映射到最终 HTML 中的 title 属性上，如下面这个 Name 字段的示例所示：

```
<input name="ctl00$cpMainContent$ContactForm1$Name" type="text" class="InputBox"
      id="cpMainContent_ContactForm1_Name" title="Enter your name" />
```

然后需要像下面这样调用插件的主方法：

```
$.updnWatermark.attachAll({ cssClass: 'Watermark' });
```

注意，不需要选择任何项。只需要调用 jQuery 对象的 updnWatermark 方法(使用\$快捷方式)，而不必指定任何选择器。然后 updnWatermark 方法将扫描表单，查找具有 title 属性的表单字段。可以选择传入一个定义了文本标记的表示的 CSS 类。在本例中，.Watermark 和.Watermark label 选择器定义了文本框上显示的水印文本的位置和颜色。

这个水印插件很智能，它并不会把文本放到文本框中。相反，它会快速创建一个标记，并将其放到文本框上，从而使文本看上去包含在文本框中。这个解决方案的智能之处在于，因为实际的字段没有改变，所以控件的验证仍然与以前一样。验证控件会看到一个空文本框，如果没有输入有效数据，但是却想提交表单，验证控件就会触发一个验证弹出窗口。

强烈建议在掌握了 jQuery 之后浏览 jQuery 站点的插件目录。这里提供的插件几乎可以解决您

遇到的各种问题。这样不但可以避免重复劳动，更重要的是可以让您用自己不曾想到的方式扩展 jQuery 和客户端 JavaScript。

另外，在访问 jQuery 网站时，一定要检查 <http://docs.jquery.com/> 上提供的文档。您将发现，jQuery 所提供的功能比本章演示的要强大得多，也丰富得多。

到目前为止，都是在操作没有什么静态特征的 Web 页面。虽然 jQuery 允许在客户端创建动态修改的页面，但是 Web 站点上提供的内容仍然是静态的。为了解决这个问题，可以使用数据库，接下来的 4 章就将讨论这个问题。在第 12 章，将全面介绍数据库，其后的几章将深入讨论如何在 ASP.NET 环境中操作数据。

11.6 关于 jQuery 的实用提示

为了最大程度地利用 jQuery，应该遵循下面的提示：

- 不断试验。一开始时，jQuery 是一个有点奇怪的技术，不太容易掌握，这主要是因为它所使用的大量大括号和小括号。但是，通过不断尝试，您可以很快就成为 jQuery 高手。
- 访问 jquery.com 网站。除了提供许多显示 jQuery 的功能的示例的优质文档之外，还可以在站点上找到大量关于使用 jQuery 的文章和链接，包括提供视频内容的站点的链接。
- 投入一些时间浏览插件目录。您可能不会马上用到插件，但是应该知道在需要时它们可以提供的功能。

11.7 本章小结

本章介绍了 jQuery，这是一种非常流行的开源客户端 JavaScript 框架，可以用来与文档对象模型进行交互。

本章首先介绍了 jQuery 的下载地址和将其添加到 Web 站点中的方法。然后提供了一个 jQuery 的示例，并在之后介绍了可以用来在页面中找到相关元素的 jQuery 选择器和筛选器。

本章的第二部分致力于讲解可以对匹配集应用效果和动画的多种 jQuery 方法。您了解了如何使用 `css` 这样的方法来操作 CSS 设置，使用 `parent` 和 `prev` 方法导航匹配集中的项，以及如何通过事件，在发生某些动作(例如单击按钮或者提交表单)时触发代码。

在本章即将结束时，您学习了如何使用 jQuery 中的众多动画方法，使页面外观更具吸引力，并且交互性更好。此外还了解了如何使用插件来扩展 jQuery。

11.8 练习

1. 假设希望向用户提供这样一种功能，使他们可以隐藏页面的特定区域。例如，在用户单击按钮时，可以向他们提供一个指向隐藏或者显示边栏元素中的大横幅的链接。实现这项功能需要哪些 jQuery 方法？如果能找到一种方法，将触发代码的文本从 `Hide` 修改为 `Show`，或者从 `Show` 修改为 `Hide`，那么可以得到额外加分。

2. slideUp 和 slideDown 有什么区别？两种方法都接受什么重要参数？

3. jQuery 的文档就绪函数(使用\$(function()...)定义)和 ASP.NET AJAX pageLoad 方法有什么区别？如何利用这种区别？

练习的答案参见附录 A。

本章要点回顾

链接	这个概念是指一个方法的结果用作另一个方法的输入，从而创建一个效果链
筛选器	允许进一步细化 jQuery 的匹配元素集
jQuery	一种流行的客户端 JavaScript 框架，可以简化 DOM、视觉效果、事件处理和 Ajax 功能的使用
匹配集	jQuery 选择器返回的元素集
插件	允许使用第三方程序员编写的现成行为扩展 jQuery
选择器	与 CSS 类似的语法，用来使用 jQuery 在页面中查找元素
水印	向表单控件添加灰色或者变暗的文本提示，以便帮助用户理解需要输入什么内容

本章要点

- 数据库的概念以及 ASP.NET 页面通常使用什么数据库
- SQL 的概念，以及如何使用它来操纵数据
- 什么是数据库关系？为什么说它们很重要
- VWD 中什么工具可用于管理表这样的对象？如何使用这些工具

清楚在 ASP.NET 4 Web 站点中如何使用数据库同了解 HTML 和 CSS 一样重要。没有它几乎不可能构建一个现代的、功能完全的 Web 站点。数据库是非常有用的，因为它允许通过结构化的方式来存储和检索数据。数据库最大的好处是能够在运行时被访问，这就意味着在 VWD 中，您将不再局限于在设计时所创建的相对静态文件。您可以使用数据库存储评论、音乐流派、图片、用户信息(用户名、电子邮件地址、密码等)、有关阅读您的评论的日志信息、新文章等，然后可以通过 ASPX 页面访问这些数据。

这就给表示的数据以及表示数据的方式提供了很大的灵活性，使您能够创建高度动态的 Web 站点，来适应访问者的喜好和站点必须提供的内容，甚至用户的角色或用户所拥有的访问权限。

为了成功地在 ASPX 页面中运用数据库，本章将教您如何使用一种名为 SQL(Structured Query Language，结构化查询语言)的查询语言来访问数据库。这种语言允许检索和操纵存储在数据库中的数据。您还将了解如何使用 VWD 中的数据库工具创建表和查询。

尽管 ASP.NET 和 .NET Framework 提供了很多工具和技术，可以让您在不必深入掌握类似 SQL 这样的底层概念的情况下运用数据库，但仍然有必要了解它们。一旦知道了如何访问数据库，就会发现可以更容易地理解其他的技术，例如 ADO.NET 实体框架(将在第 14 章进行介绍)，这一技术使得直接从代码中访问数据库操作更加简单。

在接下来的章节中，您将运用本章所学到的知识。在第 13 章，您将学习如何使用内置控件处理数据库中的数据。在第 14 章，您将学习如何使用 ADO.NET 实体框架作为数据库上的附加层，使用最少的代码以面向对象的方式访问数据。第 15 章是最后一章介绍数据的章节，将会介绍数据处理的高级技术。

在接下来的几节中，您将了解数据库的概念以及有哪些可用的不同数据库。

12.1 数据库的概念

简单来说，数据库就是以易于访问、管理和更新的形式排列的数据的集合。对于本书以及您将构建的 Web 站点来说，假定数据以电子格式存储在数据库中也是比较可靠的。

最为流行的一种数据库为关系数据库(**relational database**)。这种数据库常用于 Web 站点中，也将用于本书后续部分。不过，关系数据库并不是唯一的数据库类型。还有其他类型的数据库，包括平面文件数据库、对象关系数据库和面向对象数据库，但它们在 Internet 应用程序中不常见。

关系数据库中有表(**table**)的概念，其中数据以行和列的形式存储，如同电子表格一样。表中的每行包含存储于其中的记录项的完整信息，而每列包含表中记录项的特定属性的信息。

术语“关系”指的是数据库中不同表相互关联的方式。它不是将相同的数据一遍遍地复制，而是在其自己的表中存储重复的数据，然后从其他表中创建与该数据的关系。看一下图 12-1 中的一个名为 **Review** 的简单表。该表存储了 Planet Wrox Web 站点上呈现的 CD 或音乐会评论。

在图 12-1 中可看到，每条评论都指派到一种音乐流派，如 **Pop**、**Indie Rock** 和 **Techno**。不过，如果想将流派 **Techno** 重命名为 **Hardcore Techno** 这样的名称时该怎么办？这需要更新所有指派了这一流派的记录。如果存在存储流派的其他表，还需要访问那些表并手动进行这一修改。

图 12-1

更好的解决方案是使用一个单独的表，例如一个名为 **Genre** 的表。这个表可以存储流派的名称和唯一标识每种流派的 ID(例如一个序号)。接着建立 **Review** 表与 **Genre** 表的关系，并且只存储其 ID 而不是整个名称。图 12-2 显示了这一修改的概念模型。

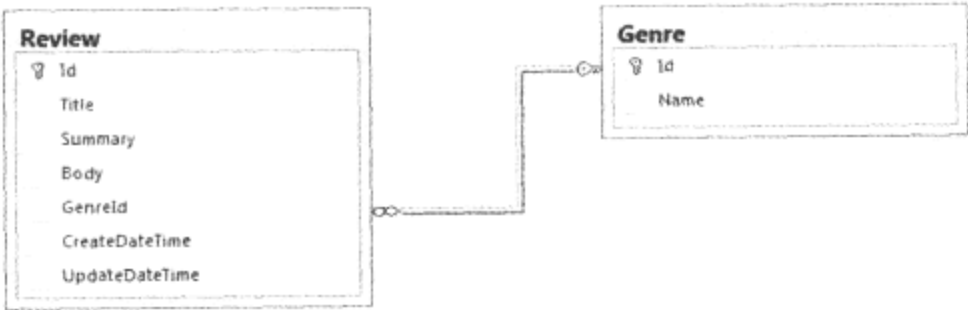


图 12-2

由于已将流派的 ID 存储在了 **Review** 表中，因此重命名流派就很简单。所需做的就是修改 **Genre** 表中的流派的名称，与该流派相关的所有表就会自动作修改。本章后面将介绍如何在关系数据库中创建和使用关系。

12.2 不同类型的关系数据库

在 ASP.NET 项目中可以使用多种不同类型的数据库，包括 Microsoft Access、SQL Server、Oracle 和 MySQL。不过，在 ASP.NET 4 Web 站点中最常用的数据库是 Microsoft SQL Server。本书主要使用 Microsoft SQL Server 2008 Express Edition，因为它是随 VWD 免费提供的，有着许多创新性的功能。而且，由于其数据库引擎与 SQL Server 2008 商业版的相同，因而可在开发周期的后续阶段轻松地升级到那些版本。附录 B 中详细描述了升级方法。即使已经安装了不同版本的 SQL Server 2008，还是推荐再安装一个 2008 Express 版本。这使您可以很容易理解本书以及 Internet 上的许多示例，这些示例通常都假设您使用的是 Express 版本。

为了处理数据库中的数据，SQL Server(和所有其他关系数据库系统)支持 SQL 查询语言。

12.3 运用 SQL 处理数据库数据

要使数据进出数据库，需要使用 SQL(Structured Query Language，结构化查询语言)。这是用于查询关系数据库的实际语言，几乎所有的关系数据库系统都能理解它。SQL 有许多明确的标准，其中最流行的是 ANSI 92 SQL 标准。除了这一标准支持的语法外，许多数据库供应商都扩展了该语言，从而使 SQL 提供只有应用于他们自己的系统时才有的更多灵活性和功能，但代价是降低了与其他系统的互操作性。

Microsoft SQL Server 2008 数据库也不例外，它支持 ANSI 92 SQL 标准中定义的大部分语法。在这一标准之上，Microsoft 添加了一些专用扩充，合起来称为 T-SQL(Transact SQL)。本书后续部分都还使用 SQL 这一术语。

在下面几节中，将介绍如何将 SQL 用于 SQL Server 2008 数据库检索和操纵数据库中的数据。不过，在编写第一个 SQL 语句前，需要先知道如何连接到数据库。下面的“试一试”练习将介绍如何连接到随本书下载代码提供的样例数据库。

试一试

连接 SQL Server 样例数据库

在这个练习中，将学习如何从 VWD 中连接和处理数据库。本章的代码下载中有一个包含了一些表的样例数据库。要从 VWD 中访问数据库，用于登录 Windows 计算机的账户需要至少具有对数据库所在文件夹的读、写权限。如果作为管理员登录，那就很符合这一要求。如果在下一“试一试”练习中访问数据库时出错了，则可参考 19.3.2 节的内容来获得配置正确权限的详细指导。

(1) 对于此“试一试”练习，需要一个新的空白 Web 站点，这可以通过选择 VWD 中的 File | New Web Site(或 File | New | Web Site)命令后选择 ASP.NET Web Site 模板来创建。不要使用到目前为止一直在采用的 Planet Wrox Web 站点，否则，在使用同名称的数据库时会遇到麻烦。可以将这个新的 Web 站点保存到一个文件夹中，如 C:\BegASPNET\DatabaseTest。如果使用了一个不同的位置，请记住这个位置，因为在第(2)步中将会用到它。

(2) 在创建好新的 Web 站点后，请确保您的账户具有足够的写入其 App_Data 文件夹的权限。由于 VWD 和内置 Web 服务器是在您用于登录 Windows 的账户下运行的，所以要确保该账户具有正确的权限。为此，打开 Windows 资源管理器(不是 VWD 的 Solution Explorer)，定位到文件夹

C:\BegASPNET\DatabaseTest。右击 App_Data 文件夹并选择 Properties 命令。切换到 Security 选项卡，确保账户(或指派的组)至少具有 Modify 权限，如图 12-3 所示。

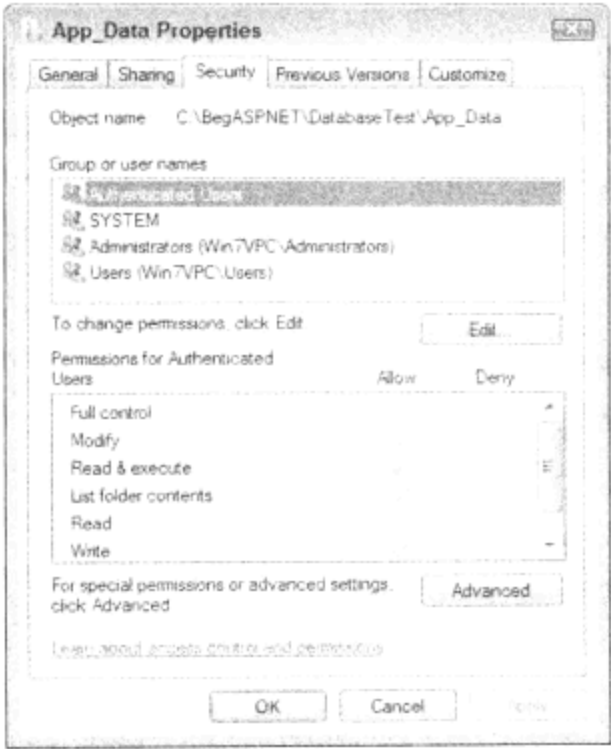


图 12-3

如果没有出现 Security 选项卡或是账户或用户组未列出，可以参考 19.3.3 节中的名为“配置文件系统”的试一试练习以获取将账户添加到这个权限列表的详细信息。

(3) 用 Windows 资源管理器打开位于 C:\BegASPNET\Resources 下的 Resources 文件夹。如果没有此文件夹，可参考本书前言部分的介绍来了解如何获取本书附带的代码。接着打开 Chapter 12 文件夹，选择文件 PlanetWrox.mdf。将 VWD 和 Windows 资源管理器并排安放，然后将该文件从 Windows 资源管理器拖至 VWD 中的 Web 站点的 App_Data 文件夹下。注意，如果拖放不起作用，可以通过复制和粘贴来完成这个工作。这个.mdf 文件是实际的数据库，包含了实际的表和记录等。当开始使用数据库时，磁盘上的 App_Data 文件夹中会出现一个.ldf 文件。SQL Server 通过它来跟踪对数据库所作的改动。如果使用的是 SQL Server 2005 Express Edition，则应该从 Sql2005 文件夹中取得数据库。

(4) 在 VWD 中双击 Solution Explorer 中的数据库文件，打开 Database Explorer(在 Visual Studio 商业版中称为 Server Explorer)中的数据库。

(5) 可以展开连接的数据库来访问其对象，如它包含的表和列，如图 12-4 所示。

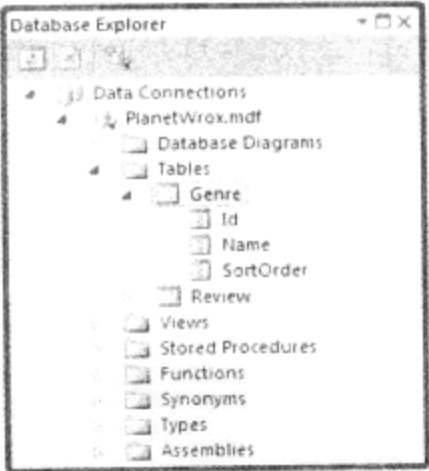


图 12-4

工作原理

要从 VWD 中访问数据库，需要在 Database Explorer 窗口的 Data Connections 下进行注册。在大多数情况下，添加数据库与添加数据库到 App_Data 文件夹并双击它一样简单。不过，如果系统上的安全性较严格，或是不能连接到 Web 站点中的实际 SQL Server 数据文件，那么连接数据库就会有些困难。如果出现这样的情况，可以参考 19.3.2 节内容和附录 B，来获取配置系统的更多详细信息。可以将 App_Data 文件夹只用于在服务器上使用的数据，例如数据库和文本文件。由于 Web 服

务器阻止远程浏览器访问此文件夹，因此不能用它存储必须由客户端下载的图像。

如果有了与 Database Explorer 中的数据库的连接，就可以处理该数据库中的对象。注意，在下一个“试一试”练习中，将介绍如何将有关数据库连接的信息(称为连接字符串(connection string))存储到 Web 站点的 web.config 文件中，以便通过代码和 ASP.NET 控件来使用数据库。

不过，首先要了解如何访问和修改数据库中的数据。

12.4 使用 SQL 检索和操纵数据

与数据库交互时，需要花大量时间检索和操纵数据。这些操作可归结为 4 种不同类型，即 CRUD (4 个单词首字母的缩略词)——Create(创建)、Read(读取)、Update(更新)和 Delete(删除)。

由于这些数据操作很重要，下面几节将详细介绍如何使用它们。

12.4.1 读取数据

要从数据库中读取数据，一般使用一些不同的概念。首先，需要表明要从查询的表中检索什么列。这是通过 SELECT 语句完成的。需要使用 FROM 关键字表明想从什么表中选择数据。然后需要筛选数据，确保只返回您感兴趣的记录。可以使用 SQL 语句中的 WHERE 子句筛选数据。最后，可以使用 ORDER BY 子句排序结果。

1. 选择数据

要从一个或多个数据库表中读取数据，可以使用 SELECT 语句。其最基本的形式如下：

```
SELECT ColumnName [, AnotherColumnName] FROM TableName
```

这里，方括号中的部分是可选的。例如，要从 Genre 表中检索所有行并只选择它们的 Id 和 Name 列，可以使用下列 SQL 语句：

```
SELECT Id, Name FROM Genre
```

紧跟 SELECT 语句的是一个用逗号分隔的列名列表。这里可以有一列或多列。也可以使用星号(*)来表明想返回所有列，而不用显式指定列名。不过，SELECT *通常被看作是个较差的编程实践，因此最好显式地指定想检索的每列。

在 FROM 关键字后，指定想从中检索数据的表的名称。前面的示例只显示了一个表(Genre 表)，不过在后面将看到也可以使用连接(join)指定多个表。



注意：尽管 SQL 语言是不区分大小写的，不过通常还是将像 SELECT 和 FROM 这样的关键字大写。另外，在本书中采用了 Pascal Casing 格式——其中新单词是大写的，如表名、列名等。例如创建具体评论的日期和时间是存储在 Review 表的名为 CreateDateTime 的列中。

2. 筛选数据

要筛选数据，可以使用 WHERE 子句，通过它可以表明匹配数据的条件。例如，要检索 Grunge

流派的 ID，可以使用下列 SQL 语句：

```
SELECT Id FROM Genre WHERE Name = 'Grunge'
```

注意，单词 **Grunge** 被括在单引号中。如果是筛选数据或是将值发送至 INSERT 或 UPDATE 语句(允许创建新的记录或修改已有记录，后面会作解释)，那么对文本数据类型和日期都有这一要求。不过，不能将它们用于数值型或布尔型，因此要获得 ID 为 8 的流派的名称，要使用下列语句：

```
SELECT Name FROM Genre WHERE Id = 8
```

上面两个示例演示了 WHERE 子句的使用，它将等于运算符用于精确匹配。不过，也可以将其他运算符用于不同的条件。表 12-1 列出了可在 WHERE 子句中使用的一些流行的比较运算符。

表 12-1

运 算 符	说 明
=	只有在比较的左侧与右侧值相等时，等于运算符才匹配
>	当比较的左侧值大于右侧值时，大于运算符匹配
>=	当比较的左侧值大于等于右侧值时，大于等于运算符匹配
<	当比较的左侧值小于右侧值时，小于运算符匹配
<=	当比较的左侧值小于等于右侧值时，小于等于运算符匹配
<>	不等于运算符与等于运算符相反，当比较的左侧值与右侧值不同时才匹配

为了结合多个 WHERE 条件，SQL 语言支持大量逻辑运算符，如 AND 和 OR。另外，它还支持其他用于搜索文本和指定范围的运算符。表 12-2 列出了一些逻辑运算符并说明了其用途。

表 12-2

运 算 符	说 明
AND	允许连接两个表达式。例如下列 WHERE 子句： ...WHERE Id > 20 AND Id <30 将返回 ID 在 20~30 之间(不包括 20 和 30)的所有记录
OR	允许定义多个条件，只需匹配其中一个条件(可允许多个匹配)。例如下列 WHERE 子句： ...WHERE Id = 12 OR Id = 27 将返回 ID 为 12 或 27 的记录
BETWEEN	允许指定一个想匹配的值的范围，有下限和上限。例如： ...WHERE Id BETWEEN 10 AND 35 将返回 ID 在 10~35 之间(如果数据库中有的话，也包括 10 和 35)的所有记录
LIKE	用于确定一个值是否匹配某个特定模式。可以使用通配符来匹配值的特定部分，如用%来匹配具有 0 个或多个字符的字符串，用下划线(_)来匹配任意单个字符。例如，下列 WHERE 子句： ...WHERE Name LIKE '%rock%' 将返回名称中含有 rock 的所有流派，包括 Indie Rock、Hard Rock 等

如果不存在与 WHERE 子句匹配的记录，那么此时不会得到一个错误，而是得到没有结果的信息。在用 WHERE 子句定义了筛选要求后，如果想改变从数据库返回的数据的顺序。这时可以使用 ORDER BY 子句。

3. 排序数据

ORDER BY 子句出现在 SQL 语句的末尾，可包含一个或多个列名或表达式，也可包括 ASC 或 DESC 来决定记录是以升序(ASC，默认关键字)或以降序(使用 DESC)排列。

例如，要检索 Genre 表中的所有流派并按它们的名称以字母升序排列，可以使用下列 SQL 语句：

```
SELECT Id, Name FROM Genre ORDER BY Name
```

由于升序是默认顺序，因此不需要显式指定 ASC 关键字，如果您想这样做当然也可以。下面的示例在功能上与前一示例相同。

```
SELECT Id, Name FROM Genre ORDER BY Name ASC
```

如果想返回同样的记录，但使它们以 SortOrder 列降序排列，可以使用下列语法：

```
SELECT Id, Name FROM Genre ORDER BY SortOrder DESC
```

注意，从 SortOrder 列的例子可以看到，在 ORDER BY 语句中如何按列排序并不是 SELECT 语句的一部分。因此，即使某个特定的列不是最终结果集的一部分，仍可以以它进行排序。

在下一个“试一试”练习中，将介绍如何对样例数据库执行大量查询，从而使我们很好地了解不同的查询如何影响从数据库返回的结果。

试一试

从样例数据库中选择数据

在这个“试一试”练习中，使用前一个“试一试”练习中连接的数据库。该数据库只用于本章中的示例，所以不用担心会陷入困境。

(1) 打开 Database Explorer(或 Visual Studio 付费版本中的 Server Explorer)，方法是选择 View | Database Explorer。定位到之前添加的 Data Connection，展开至 Tables 节点。在其中可看到两个表，Genre 和 Review，如图 12-5 所示。

(2) 右击 Genre 表并选择 Show Table Data 命令。在文档窗口中将看到一个包含 Genre 表中所有可用流派的列表，如图 12-6 所示。

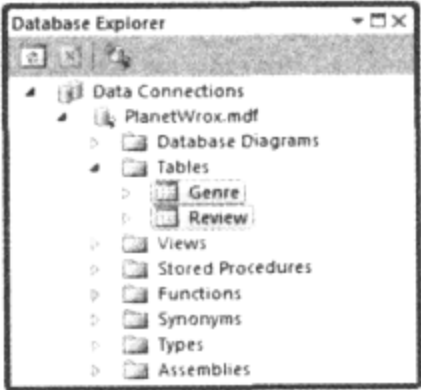


图 12-5



图 12-6

注意，这并不是 Genre 表中所有记录的列表。它实际上是您打开窗口时，执行的一个 SQL

SELECT 查询的结果。要查看出现这一列表所涉及的查询，请确保显示了 Query Designer(查询设计器)工具栏，如图 12-7 所示。如果该工具栏不可见，可右击已有工具栏并单击 Query Designer。

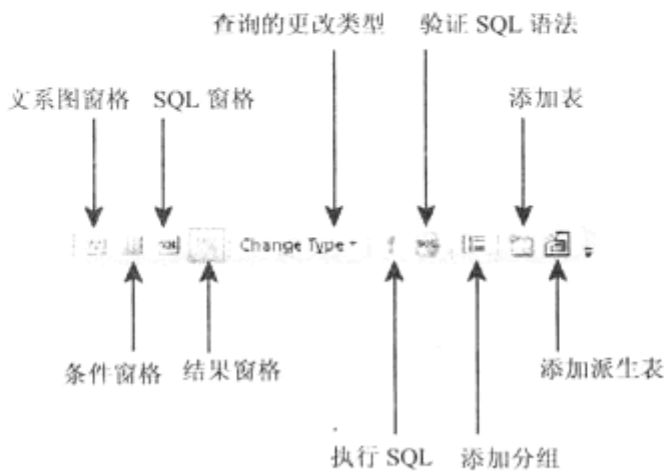


图 12-7

在该工具栏上，单击关系图窗格、条件窗格和 SQL 窗格按钮分别打开这些窗口。工具栏上的前 4 个按钮现在是按下的状态，文档窗口现在被分为 4 个区域，每个区域对应于这 4 个按钮中的一个。图 12-8 显示了带有 4 个窗格的整个文档窗口。

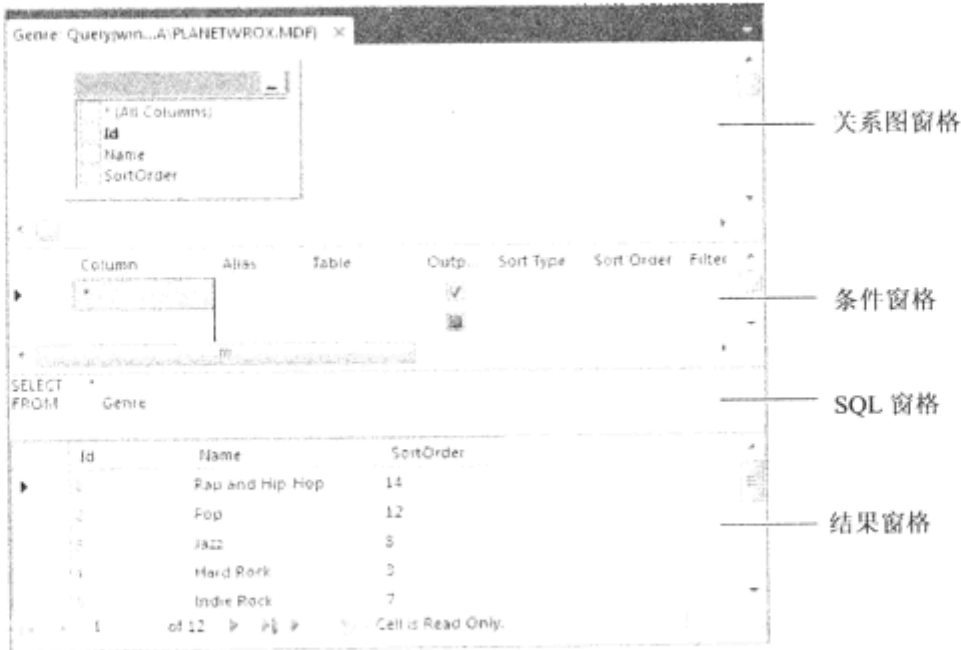


图 12-8

SQL 窗格显示了用于检索显示在结果窗格中的流派的 SQL 语句。在这里，SQL 语句读取 `SELECT * FROM Genre` 检索表中的所有列和记录，但可以很容易地修改它。

(3) 在 SQL 窗格中，将光标放在单词 `Genre` 的正后方，按一下 `Enter` 键，然后输入 `WHERE Id > 4`。完整的 SQL 语句最终如下所示：

```
SELECT * FROM Genre WHERE Id > 4
```

在 SQL 窗格中，查询被分成多行。这很不错，因为 SQL 允许将语句分散到多行，而不需要续行符。

(4) 为了确保 SQL 语句是有效的，单击工具栏上的“验证 SQL 语法”按钮，然后修正 SQL 语句可能包含的错误。接着，单击“执行 SQL”按钮(该按钮上有一个红色惊叹号)或按 `Ctrl+R` 组合键。在这两种情况下，都会执行 SQL 语句，从而使结果窗格更新为显示所有 ID 大于 4 的流派。

(5) 除了显示结果外，VWD 还改变了查询方式。它显式地显示了表中的所有列，而不是使用 SELECT *语句。现在看一下关系图窗格：图 12-8 中对话框的上部显示了整个表。在关系图窗格中，通常可以通过选择和不选择列名来决定它们最终是否出现在查询中。取消选定 SortOrder 列(注意不要改变条件窗格中 Output 列的复选标记)。注意，它也从条件窗格(如图 12-9 所示)和 SQL 窗格的 SQL 语句中移除了。

Column	Alias	Table	Output	Sort Type	Sort Order	Filter
Id		Genre	<input checked="" type="checkbox"/>			> 4
Name		Genre	<input checked="" type="checkbox"/>			

图 12-9

(6) 看一下条件窗格。它显示选择了两列。在 Filter 列显示了筛选所有 ID 大于 4 的流派的表达式，如图 12-9 所示。

在这个窗格中，可以在无需编写大量代码的情况下修改查询。要了解如何应用一个附加的筛选器，可在 Name 行的 Filter 单元格中输入 LIKE '%rock%'。这就限制结果为包含单词 rock 和 ID 大于 4 的所有流派。如果再次按下 Ctrl+R 键，则结果窗格就会更新以反映在查询中所做的更改。

(7) 要确定排序顺序，可以使用 Sort Type 列。这可以针对可视列(例如，那些列的 Output 复选框被选中，且在最终的结果集中)，也可以针对其他列。要按 SortOrder 列排序，则单击一下 Name 下边的单元格。该单元格将发生改变并显示一个下拉列表。从该下拉列表中选择 SortOrder。单击或按 tab 键离开该字段，VWD 将在 Output 列中放置一个复选标记。可以单击该复选标记从输出中再次删除该列，使之仍可用于排序和筛选，但不要在查询结果中出现。不过对于本练习来说，可以选择该列。

(8) 在 Sort Type 列中，从下拉列表中选择 SortOrder 为 Descending。最终的条件窗格如图 12-10 所示。

Column	Alias	Table	Output	Sort Type	Sort Order	Filter
Id		Genre	<input checked="" type="checkbox"/>			> 4
Name		Genre	<input checked="" type="checkbox"/>			LIKE '%rock%'
SortOrder		Genre	<input checked="" type="checkbox"/>	Descending	1	

图 12-10

在使用关系图和条件窗格修改的同时，VWD 会不断更新 SQL 窗格。最终的 SQL 语句现在也包括了额外的 WHERE 子句和 ORDER BY 语句：

```
SELECT Id, Name, SortOrder
FROM Genre
WHERE (Id > 4) AND (Name LIKE '%rock%')
ORDER BY SortOrder DESC
```

(9) 再次按 Ctrl+R 组合键(或单击工具栏上的 Execute SQL 按钮)，结果窗格将显示 Genre 表中匹配条件的记录，如图 12-11 所示。

Genre: Query(win...A\PLANETWROX.MDF) X			
	Id	Name	SortOrder
▶	9	Alternative Rock	9
	5	Indie Rock	7
	7	Rock	2
1 of 3			

图 12-11

注意，记录现在按 `SortOrder` 列的降序排列。

工作原理

VWD 中的 Query Designer 是一个非常有用的工具，可以创建针对数据库的新查询。不是在 SQL 窗格中手动编码整个 SQL 语句，而是使用关系图和条件窗格可视地创建查询。当然，也可以使用 SQL 窗格对 VWD 生成的 SQL 代码作手动调整。

最后执行的查询返回包含单词 `rock` 和 `ID` 大于 4 的所有记录。在第(8)步显示的查询有一个 WHERE 子句，它由两部分组成：第一部分限制了返回的记录的 `ID` 大于 4；第二部分筛选包含文本 `rock` 的记录。这两个条件是通过使用 `AND` 关键字同时应用的，因此只有 `ID` 大于 4 并且其名称中含有单词 `rock` 的记录被返回。实际返回的是 `Alternative Rock`、`Indie Rock` 和 `Rock` 流派，而没有 `Hard Rock` 流派，因为其 `ID` 为 4。

最后，结果集以 `SortOrder` 列的降序排列，使用的语法为 `ORDER BY SortOrder DESC`。注意，`SortOrder` 是一个随意选择的名称。可以很容易地给该列起一个不同的名称，或按一个不同的列(如 `Name` 列)进行排序，以字母顺序检索流派。

在这个例子中，可以看到如何从单个表中检索数据。不过，在大多数实际应用程序中，要从多个表中获取数据，这些表以某种方式相关。可以使用 `JOIN` 关键字在 SQL 语法中定义此关系。

4. 连接数据

查询中的 `JOIN` 允许表示一个或多个表之间的关系。例如，可以使用一个 `JOIN`，从 `Review` 表中查找在某个特定流派中发布的所有评论，然后从 `Review` 表中选择一些列，包括该流派的 `Name`。

`JOIN` 的基本语法类似于下列突出显示的代码：

```
SELECT
    SomeColumn
FROM
    LeftTable
INNER JOIN RightTable ON LeftTable.SomeColumn = RightTable.SomeColumn
```

第一部分是查询的标准 `SELECT` 部分(之前已看到过)，而第二部分引入了关键字 `INNER JOIN` 来表达两个表之间的关系。此查询只返回 `LeftTable` 表在 `RightTable` 表中有相应记录的记录。例如，要返回一个评论的 `Id` 和 `Title`，以及它所属的流派的名称，可使用下列 SQL 语句：

```
SELECT
    Review.Id, Review.Title, Genre.Name
FROM
    Review
INNER JOIN Genre ON Review.GenreId = Genre.Id
```

注意，在 `SELECT` 语句中，每列都用表名作为前缀。这清楚地表明了引用的表，避免了在多个表中有类似列名时的冲突(例如两个表中都有 `Id` 列)。

除了只返回匹配记录的 `INNER JOIN` 外，还可以使用 `OUTER JOIN`。`OUTER JOIN` 允许从一个表中检索记录，而不管它们在另一表中是否有匹配记录。下列示例将返回系统中所有流派的一个列表，以及每个流派中的评论：

```
SELECT
```



```
Genre.Id, Genre.Name, Review.Title
FROM
  Genre
LEFT OUTER JOIN Review ON Genre.Id = Review.GenreId
```

对于指派给一个流派的每条评论，会返回一行，其中包含该评论的 Title。不过，即使流派未被评论，也会返回该行。如图 12-12 所示。



图 12-12

在图 12-12 中，可以看到对于 Review 表中指派给 Indie Rock 流派的每条评论，该流派重复了多次。而像 Punk 这样的流派，只有一个评论与之相连，因此它们只列出一次。最后 Rock 和 Grunge 流派，没有与之相关的评论。不过，由于 SQL 语句使用了 LEFT OUTER JOIN，所以仍会返回这两种流派(列于 JOIN 左侧)。没有返回评论的 Title，该列现在包含的是一个 NULL 值，表明没有相关的评论。

除了 LEFT OUTER JOIN，还有 RIGHT OUTER JOIN，它返回位于 JOIN 右侧的表中的所有记录。LEFT OUTER JOIN 和 RIGHT OUTER JOIN 语句非常类似，但大多数情况下看到的是前者。

另外，还有其他一些连接，包括交叉连接(cross join)和 self join(自连接)。要获得这些类型的连接的详细信息，可参阅清华大学出版社引进并出版的《SQL Server 2008 编程入门经典(第 3 版)》一书(ISBN: 978-7-302-21432-8)。

在接下来这个“试一试”练习中将演示如何使用一个非常常见的连接类型——INNER JOIN。

试一试

连接数据

要连接两个表的数据，需要在代码中编写一个 JOIN 语句。为了帮助编写该代码，VWD 在您添加表到关系图窗格时会尝试添加一个 JOIN。不过，有时这个 JOIN 是不正确的，因此需要对其进行手动修改。

(1) 仍然在测试站点中，在 Database Explorer(或 Server Explorer)窗口中，右击 Review 表并选择 Show Table Data 命令。将看到出现了表中的所有评论。接着，单击 Query Designer 工具栏上的按钮，分别打开关系图窗格、条件窗格和 SQL 窗格。

(2) 右击关系图窗格中位于 Review 表旁的一个打开区域，选择 Add Table 项。或者从主菜单中选择 Query Designer | Add Table 命令。

(3) 在打开的对话框中单击 Genre 表，然后单击 Add 按钮。最后，单击 Close 按钮。

(4) VWD 生成的 SQL 语句如下所示：

```
SELECT * FROM Review
INNER JOIN Genre ON Review.GenreId = Genre.Id
```

VWD 会正确检测出数据库中定义在 Review 表的 GenreId 列和 Genre 表的 Id 列之间的关系，从而应用正确的 JOIN 语句。稍后将会介绍如何自定义这些关系。

(5) 要了解如何不直接编写代码而创建 JOIN，需要手动重建 JOIN。首先在关系图窗格中右击两表之间的连线并选择 Remove 命令。SQL 语句现在包含了一个 CROSS JOIN。

(6) 然后，单击关系图窗格中 Review 表的 GenreId 列，将它拖放至 Genre 表的 Id 列。一旦释放鼠标键，VWD 就会在 SQL 窗格中创建一个新的 INNER JOIN，代码与前面所见的代码完全一样。

(7) 在条件窗格中，单击包含星号(*)的第一行的左边页边距选取整行，然后按 Delete 键或右击左边页边距并选择 Delete 命令。这样就从 SQL 语句中删除了星号。或者，可以直接从 SQL 窗格中删除星号。

(8) 在关系图窗格中，复选 Review 表的 Id 和 Title 列以及 Genre 表的 Name 列。

(9) 最后，按 Ctrl+R 组合键执行查询。文档窗口将如图 12-13 所示，在屏幕底部的结果窗格中显示了查询结果。

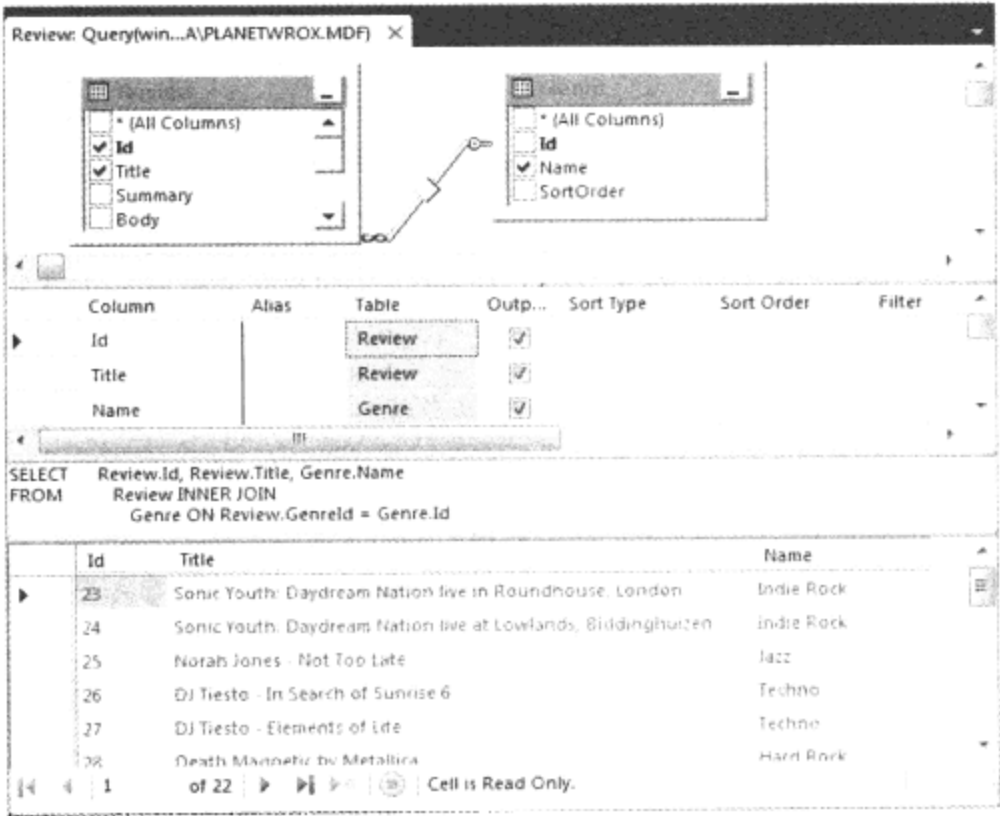


图 12-13

工作原理

通过在 SQL 语句中使用 JOIN，可以告知数据库如何将记录相关。在这个示例中，创建了 Review 表的 GenreId 列和 Genre 表的实际 Id 之间的关系：

```
SELECT
    Review.Id, Review.Title, Genre.Name
FROM
    Review
INNER JOIN Genre ON Review.GenreId = Genre.Id
```

通过此 JOIN 操作，可以从多个表中检索数据，并在一个单独的结果集中呈现它们。SQL Server 的查询处理器为每条评论返回正确的流派名，如图 12-13 所示。

除了选择数据外，还需要能够将数据插入到数据库中。这是使用 INSERT 语句完成的。

12.4.2 创建数据

要将新记录插入到 SQL Server 表中，可以使用 INSERT 语句。它有一些不同的形式，但最简单的形式如下所示：

```
INSERT INTO TableName (Column1 [, Column2]) VALUES (Value1 [, Value2])
```

和 WHERE 子句一样，需要将字符串和日期值括在单引号中，但可以在 SQL 语句中直接输入数字和布尔值。下列代码段显示了如何在 Genre 表中插入新行：

```
INSERT INTO Genre (Name, SortOrder) VALUES ('Tribal House', 20)
```

在创建好一些数据后，您可能想再次编辑它，这时可以使用 UPDATE 语句。

12.4.3 更新数据

要更新表中的数据，可以使用 UPDATE 语句：

```
UPDATE TableName SET Column1 = NewValue1 [, Column2 = NewValue2] WHERE  
Column3 = Value3
```

在 UPDATE 语句中，使用 Column = Value 结构表示特定列的新值。可以有多个这样的结构，其最大数目就是表中的列数。为了限制更新的记录项数，可以像之前选择数据一样使用 WHERE 子句。

下面这个示例更新了之前用 INSERT 语句插入的记录。它将 Name 设置为 Trance，将 SortOrder 更新为 5，使记录项在排序列表中稍提升了些。它还在 WHERE 子句中设置了新记录的唯一 ID(这个例子中是 13)，从而限制被 UPDATE 语句影响的记录数。

```
UPDATE Genre SET Name = 'Trance', SortOrder = 5 WHERE Id = 13
```

显然，您可能还需要删除已有记录。毫无疑问，SQL 语言使用 DELETE 语句来完成。

12.4.4 删除数据

和 SELECT 和 UPDATE 语句一样，可以在 DELETE 语句中使用 WHERE 子句来限制删除的记录数。这个 WHERE 子句通常非常重要，因为如果没有它，很可能会删除整个表而不只是一小部分记录。

在编写 DELETE 语句时，不需要指定任何列名。所需做的只是指明想从哪个表中删除记录，并用 WHERE 子句限制删除的记录数(这是可选的)。下面这个示例删除了在前两个例子中插入并更新的记录。

```
DELETE FROM Genre WHERE Id = 13
```

如果去掉上面的 WHERE 子句，则所有的记录都会从表中删除。
在接下来的这个“试一试”练习中将看到这些 SQL 语句的作用。

试一试

处理样例数据库中的数据

在这个练习中，将实际运用前面所学的知识。通过一系列步骤，可以看到如何在 Genre 表中创建一条新记录，再次选择它来找到其新的 ID，使用 UPDATE 语句更新它，最后从数据库中删除该流派。尽管示例本身看起来相当简单，但却集中体现了 SQL 的工作原理。如果您通过本节理解了这些示例，那么就能在本章和后面的章节中运用其余的 SQL 语句。

(1) 在您的临时测试站点中打开 Database Explorer 窗口，定位到数据库中的 Genre 表。右击它并选择 Show Table Data 命令。如果该表已经用旧式查询打开，那么需要按 Ctrl+F4 组合键先将它关闭。这样就删除了已有的 SQL 语句。

(2) 单击 Query Designer 工具栏中的前 3 个按钮，分别打开关系图窗格、条件窗格和 SQL 窗格。

(3) 在关系图窗格中，选择 Name 和 SortOrder 列。请确保未选择 Id 列，如图 12-14 所示。

由于 ID 列的值是由数据库自动生成的，因此我们无法显式地在 INSERT 语句中为它们提供值。

(4) 单击 Query Designer 工具栏上的 Change Type 按钮，然后选择第 3 个选项：Insert Values。SQL 窗格中的查询会得到更新，现在包含一个 INSERT 语句的模板：

```
INSERT INTO Genre (Name, SortOrder) VALUES (,)
```

(5) 在 VALUES 的括号之间，输入流派的名称(在单引号之间)并为之排序，用逗号隔开：

```
VALUES ('Folk', 15)
```

(6) 按 Ctrl+R 组合键执行该查询。系统将出现一个对话框，告知该操作影响了一行记录，如图 12-15 所示。

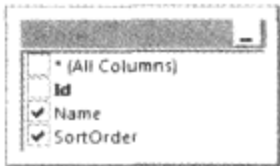


图 12-14

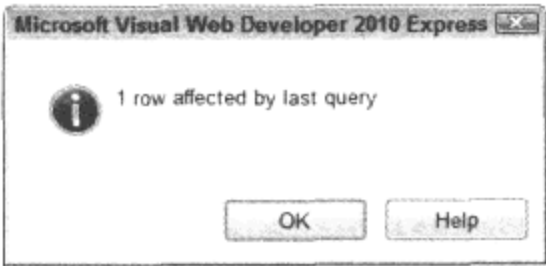


图 12-15

(7) 单击 OK 按钮关闭该对话框。

(8) 从 SQL 窗格中清除整个 SQL 语句(可以使用 Ctrl+A 组合键选择整个 SQL 语句，然后按 Delete 键删除此 SQL 语句)，并用下列代码(选择所有流派并以降序排列)替换：

```
SELECT Id, Name FROM Genre ORDER BY Id DESC
```

(9) 按 Ctrl+R 组合键执行此 SELECT 语句。结果窗格会显示一个流派列表，刚插入的那个流派位于列表最上面。注意新插入记录的 ID。如果之前没有插入任何记录的话，这个 ID 应为 13，如果是其他值也无妨。

(10) 再次单击工具栏上的 Change Type 按钮，这次选择 Update 命令。完善 VWD 创建的 SQL 语句，如下所示：

```
UPDATE
  Genre
SET
  Name = 'British Folk',
  SortOrder = 5
```

```
WHERE
  Id = 13
```

不要忘了用第(9)步确定的 ID 替换 SQL 语句中的 13。

(11) 再次按 Ctrl+R 组合键执行该查询，将出现一个对话框通知一条记录被修改。

(12) 再次清除 SQL 窗格，输入下列查询并按 Ctrl+R 组合键执行。

```
SELECT Id, Name FROM Genre WHERE Id = 13
```

用第(9)步确定的记录的 ID 替换 WHERE 子句中的 Id。将看到出现了更新的记录。

(13) 单击 Query Designer 工具栏上的 Change Type 按钮并选择 Delete 命令。VWD 将修改 SQL 语句，以删除 ID 为 13 的记录。

```
DELETE FROM Genre WHERE (Id = 13)
```

(14) 按 Ctrl+R 组合键执行该查询并从数据库中删除记录。单击 Ok 按钮关闭确认对话框。

(15) 为了确认记录已被删除，可再次单击 Change Type 按钮并选择 Select 命令。然后在关系图窗格中选择 Genre 表的一个或多个列，并再次按 Ctrl+R 组合键。可以看到这次没有返回记录，可确认新插入的流派已从数据库中删除。

工作原理

在这个简短的练习中，执行了 CRUD 这 4 项操作，从中可看到数据在 SQL Server 数据库中从创建到删除的整个过程。

首先是 INSERT 语句：

```
INSERT INTO Genre (Name, SortOrder) VALUES ('Folk', 15)
```

这在 Genre 表中创建了一条新记录。在下一节将看到，Genre 表的 Id 列是标识列(identity column)，这意味着每条新记录都会被自动指派一个新的、有序的 ID。

为了检索该 ID，使用一个带有 ORDER BY 子句的 SELECT 语句，该子句以 ID 降序排列，因此最新的 ID 位于列表最上面。像这样在一个繁忙的应用程序中检索新 ID 是不可靠的，结果有可能是其他记录的 ID。本书后面将介绍如何以可靠的方式检索 ID，但就本练习而言，ORDER BY 方法很适合。

通过新的 ID，执行 UPDATE 语句来修改新插入流派的 Name 和 SortOrder。如果只想用 UPDATE 语句更新一列，如只想更新 Name 列，则可以轻松地忽略其他列。例如，下面的 UPDATE 语句只更新 Name 列，而使其他列保持原值：

```
UPDATE
  Genre
SET
  Name = 'British Folk'
WHERE
  Id = 13
```

在本练习的最后，执行一个 DELETE 语句来删除新记录。在执行 DELETE 或 UPDAE 语句时指定一个 WHERE 子句通常很重要，这样可以避免清除整个表或将所有记录的名称设置为相同的值。

```
DELETE FROM Genre WHERE (Id = 13)
```


此 SQL 语句只删除 ID 为 13 的记录。如果记录存在，就被删除。如果记录不存在，也不会引发错误，但 VWD 中的对话框会显示没有影响任何记录。这个例子中没有用到括号，但是当 WHERE 子句中有多个条件时，使用它有助于确定优先级。

到目前为止，我们已经介绍了如何处理数据库中的已有表。不过，了解如何使用关系创建新表也很重要。这将在下一节讨论。

12.5 创建自己的表

使用 VWD 的内置数据库工具创建 SQL Server 2008 数据库表很容易。在下一节简要介绍了 SQL Server 2008 或更高版本中可随意使用的数据类型之后，我们将介绍如何在数据库中创建自己的表。

12.5.1 SQL Server 中的数据类型

和 Visual Basic .NET 和 C#这样的编程语言一样，SQL Server 数据库也使用不同的数据类型来存储数据。SQL Server 2008 大约支持 30 多种不同的数据类型，大部分与.NET 中使用的类型类似。表 12-3 列出了最常用的 SQL Server 数据类型和说明，以及它们在.NET 中对应的数据类型。

表 12-3

SQL 2008 数据类型	说 明	.NET 数据类型
bit	以 0/1 格式存储布尔值(1 表示 True, 0 表示 False)	System.Boolean
char / nchar	包含固定长度的文本。如果存储的文本短于定义的长度，就用空格填充。nchar 以 Unicode 格式存储数据，允许存储用各种语言编写的数	System.String
datetime	存储日期和时间	System.DateTime
datetime2	与 datetime 类型相似，但具有更高的精度和范围	System.DateTime
date	存储日期，没有时间元素	System.DateTime
time	存储时间，没有日期元素	System.TimeSpan
decimal	允许存储较大的小数	System.Decimal
float	允许存储较大的小数	System.Double
image	允许存储大的二进制对象，如文件。尽管其名称暗示了只可用它存储图像，但事实并非如此。可用它存储任何类型的文档或其他二进制对象	System.Byte[]
tinyint	用于存储 0~255 之间的整数	System.Byte
smallint	用于存储 - 32 768~32 767 之间的整数	System.Int16
int	用于存储 - 2 147 483 648~2 147 483 647 之间的整数	System.Int32
bigint	用于存储 - 9 223 372 036 854 775 808 ~9 223 372 036 854 775 807 之间的较大整数	System Int64
text / ntext	用于存储较多的文本	System.String

(续表)

SQL 2008 数据类型	说 明	.NET 数据类型
varchar / nvarchar	用于存储变长的文本。nvarchar 以 Unicode 格式存储数据，这样就可以存储用各种语言编写的数据库	System.String
uniqueidentifier	存储全局唯一标识符	System.Guid

要查看 SQL Server 2008 支持的所有数据类型的完整列表，请参见 <http://tinyurl.com/SqlDataTypes> 站点上的 MSDN 文档。

其中的一些数据类型允许指定最大长度。在定义 char、nchar、varchar 或 nvarchar 类型的列时，需要指定字符长度。例如，nvarchar(10)最多可存储 10 个字符。从 SQL Server 2005 开始，到 SQL Server 2008 之前的 SQL Server 版本，这些数据类型都允许指定 MAX 为最大值。通过 MAX 说明符，可以在单个列中最多存储 2GB 的数据。对于大段的文本，像评论主体部分，应该考虑使用 nvarchar(max) 数据类型。如果清楚某列(像邮政编码或手机号)的最大长度或想显式限制其长度，则可以指定这一长度。例如，评论的标题应存储于 nvarchar(200)的列中，限制最大字符数为 200。

12.5.2 了解主键和标识列

为了唯一标识表中的记录，需要建立一个主键(primary key)。主键由表中一个或多个列组成，包含一个唯一的跨所有记录的值。如果将一列标识为主键，那么数据库引擎就可以确保最终不会出现具有相同值的两个记录。主键可以由单个列(例如，包含了表中每条记录的唯一数值的数字列)组成，也可以跨多个列，这些列组合起来构成整条记录的唯一 ID。

SQL Server 也支持标识列。标识列是一个数字列，其值是在插入新记录时自动生成的。它们通常用作表的主键。在下一节中创建自己的表时将看到这一运用。

一般并不需要给每个表都指定主键，不过这样做可以使数据库编程人员的工作变得简单，因此建议还是为表添加主键。

使用 VWD 的数据库工具可以很容易地创建表、主键和标识列，这些内容将在下一个“试一试”练习中看到。

试一试

在表设计器中创建表

在本练习中，将为添加到 Planet Wrox 项目的新数据库添加两个表。应该在前面章节中创建的 Planet Wrox Web 站点中执行这些练习。可以关闭并删除在本章开始时创建的测试站点，因为我们已经不再需要它了。本练习假定已经为本地数据库创建了表(存储在 App_Data 文件夹中)。如果使用的是一个独立的远程 SQL Server 数据库，则不能使用 VWD Express 这么操作，而是需要从 www.microsoft.com/sql/express 站点上下载 SQL Server Management Studio 的 Express 版本。

(1) 在 VWD 的 C:\BegASPNET\Site 文件夹下打开 Planet Wrox Web 站点，右击 App_Data 文件夹并选择 Add New Item 命令。在打开的对话框中，单击 SQL Server Database，输入名称 PlanetWrox.mdf，然后单击 Add 按钮将该数据库添加到站点中。Database Explorer(或 Server Explorer)将会自动显示新的数据库。如果没有显示，可双击 Solution Explorer 中的 PlanetWrox.mdf。

(2) 在 Database Explorer 中，右击 Tables 节点并选择 Add New Table 命令，如图 12-16 所示。

(3) 在打开的对话框中，可输入构成表定义的列名和数据类型。为 Genre 表创建 Id、Name 和 SortOrder 3 列，结果如图 12-17 所示。

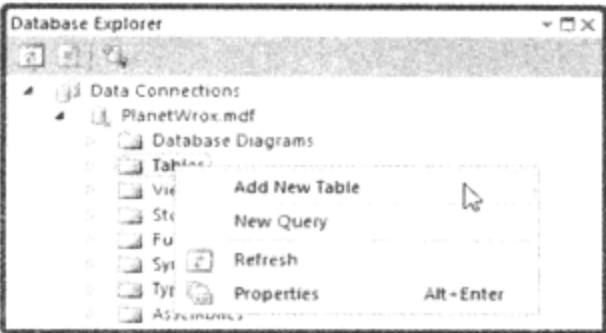


图 12-16

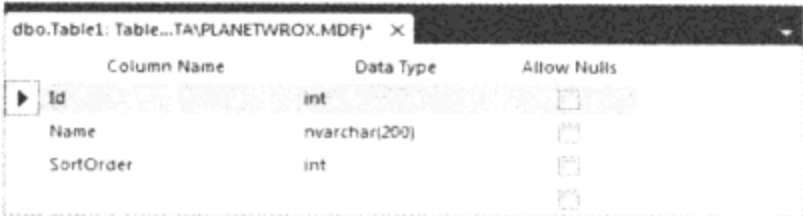


图 12-17

确保对于这 3 个数据项都没有选择 Allow Nulls 列。因为这一列决定了字段是可选的还是必选的。在 Genre 表中，这 3 项都是必选的，所以要取消选择 Allow Nulls 列。

(4) 接着，单击 Id 单元格左侧的页边距(在图 12-17 中用黑色箭头标识)选取整行，然后在 Table Designer(表设计器)工具栏(如图 12-18 所示)上，单击左侧第 2 个按钮(其上有一个黄色钥匙图标)，将 Id 列作为主键。

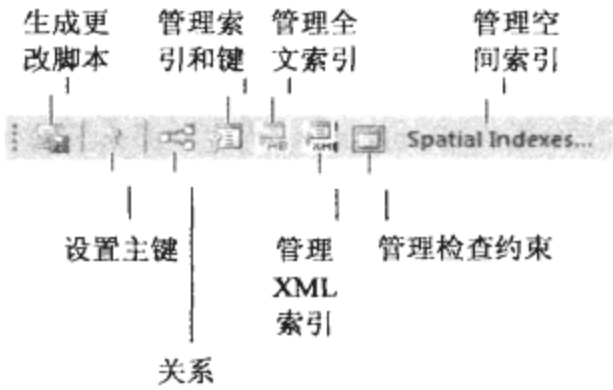


图 12-18

(5) 在表定义之下，可看到 Column Properties 面板，它类似于 VWD 中的 Properties 面板。在 Id 列仍处于选中的状态下，在 Column Properties 面板上稍向下滚动，直至看到 Identity Specification 项。展开它并设置 Is Identity 为 Yes，如图 12-19 所示。

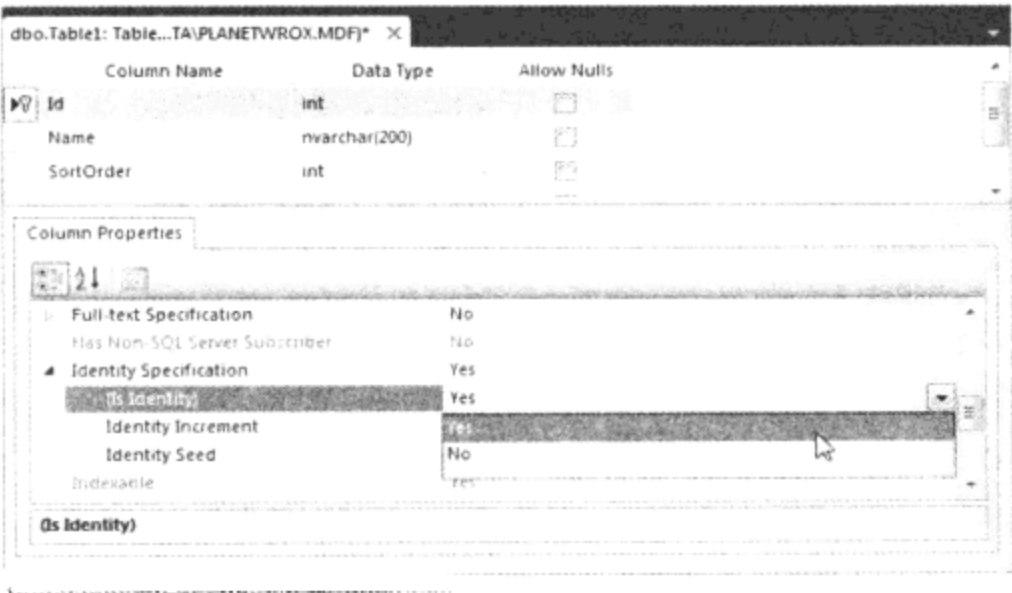


图 12-19

- (6) 按 Ctrl+S 组合键保存修改。会弹出一个对话框要求为该表提供一个名称。输入 Genre 作为表名并单击 OK 按钮以应用更改。
- (7) 按照第(2)、(3)步创建另一个表，但在这个表中要包含表 12-4 列出的对 Planet Wrox Web 站点上的 CD 和音乐会评论所作的存储规范。

表 12-4

列 名	数 据 类 型	是否允许为空	说 明
Id	int	否	表的主键和标识
Title	nvarchar(200)	否	包含评论的标题
Summary	nvarchar(max)	否	包含评论的简短小结或摘要
Body	nvarchar(max)	是	包含评论的整个主体文本
GenreId	int	否	包含评论所属的流派的 ID
Authorized	bit	否	确定评论是否得到管理机构授权发布。未授权的评论将不在 Web 站点上显示
CreateDateTime	datetime	否	创建评论的日期和时间
UpdateDateTime	datetime	否	评论最后被更新的日期和时间

- (8) 将 Id 列再次作为主键，像第(4)步和第(5)步那样设置其 Is Identity 属性为 Yes。
- (9) 单击 CreateDateTime 列，然后在 Column Properties 面板上的 Default Value or Binding 字段下输入 getdate()作为它的值，如图 12-20 所示。

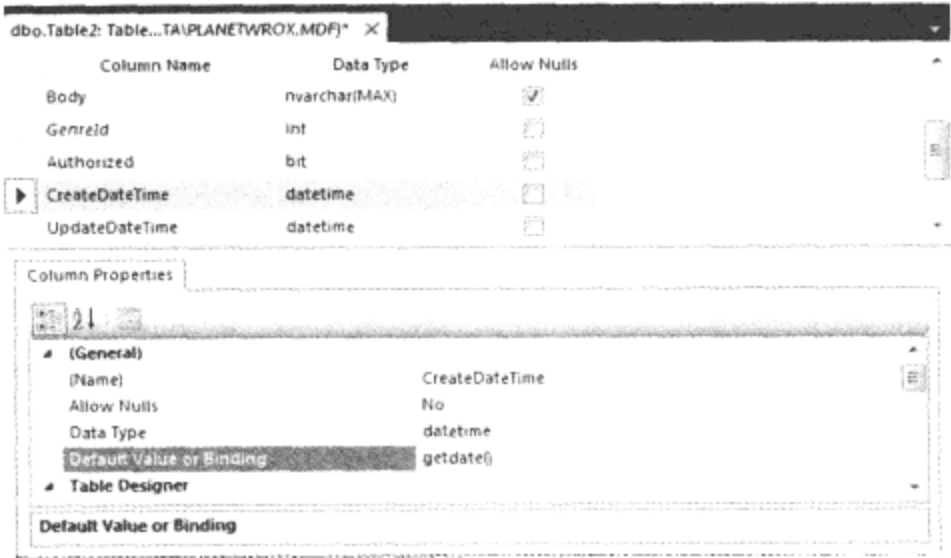


图 12-20

- (10) 对 UpdateDateTime 列重复第(9)步。
- (12) 在完成之后，按 Ctrl+S 组合键保存该表并将其命名为 Review。

工作原理

VWD 中的 Table Designer 非常简单。您只需输入新的列名，为该列定义数据类型，这差不多就是所有的工作了。而 Genre 和 Review 表中的一些列，如 Id 列，则需要多一点的工作。对于这些列，将 Is Identity 设置为 Yes。这意味着 SQL Server 会自动为插入的每条新记录指派一个新序号。在默认

情况下，表中的第一条记录的 ID 为 1，后续记录的 ID 依次加 1。可以通过对 Identity Specification 元素中的 Identity Increment 和 Identity Seed 进行设置，来改变列的默认行为。

也可以给 Review 表的 CreateDateTime 和 UpdateDateTime 列指派默认值(default value)。默认值是在您没有在 SQL 语句中显式提供的情况下由数据库插入的。也就是说，如果 INSERT 语句中没有包含 CreateDateTime 和 UpdateDateTime 列的值，数据库会自动插入默认值。在前一个“试一试”练习中，这个默认值是 getdate()，它会自动插入当前日期和时间。这种方法使得可以很容易地跟踪发表的评论。在随后的章节中我们将会学习如何在更新评论后更新 UpdateDateTime 列。

如果您不确定是否正确遵循了所有这些步骤，可以看一下随本章源代码提供的数据库。它包含了正确的表，其中有一些记录。要查看数据库，可以在 VWD 中创建一个新的临时 Web 站点，将数据库文件拖至 App_Data 文件夹中。然后就可以使用 Database Explorer 访问该数据库了。与本章前面一样，将会在 Resources 文件夹中看到 SQL Server 2005 的一个特定版本。

除了之前看到的用 SELECT 和 JOIN 语句仅在自己的 SQL 查询中定义的关系外，还可以在数据库中创建关系。下一节中将介绍此关系的用处和如何在数据库中进行创建。

12.5.3 创建表之间的关系

考虑一下到目前为止创建的表。创建了一个 Genre 表，其中 Id 列唯一标识流派记录。还创建了一个带有 GenreId 列的 Review 表。显然，这个列应包含指向 Genre 表中的记录的 Id，这样可以知道评论属于哪个流派。现在，假设从有一些评论与之连接的 Genre 表中删除一条记录。如果没有关系存在，那么数据库会让您进行这一操作。不过，这会导致很大的麻烦。如果您现在尝试显示该流派及评论，将会失败，因为已经不再有匹配的流派了。同样，如果想列出系统中按流派分组的所有评论，也将会遗漏属于已删除的那个流派的评论。

为了避免这类问题，使数据库保持良好的、一致的状态，可以在两表之间创建关系。在建立了正确的关系后，数据库将阻止无意间从一个仍有其他记录与之关联的表中删除记录。

除了保护数据外，关系也使得数据模型更清晰。如果通过关系图(下一“试一试”练习中将使用)查看数据库，则会发现表间的关系有助于更好地理解表的连接方式和它们表示的数据。

可以通过在一个表的主键和另一个表的列之间创建关系来进行关系的定义。在另一个表中的列通常被称为外键(foreign key)。在 Review 和 Genre 表的例子中，Review 表的 GenreId 列指向 Genre 表的主键列 Id，这就使 GenreId 成为外键。在下个“试一试”练习中，将介绍如何在两表间创建关系，然后执行一个 SQL 语句，显示关系如何有助于保护数据。

试一试 在两表之间创建关系

在两表间可视化地添加关系之前，需要为数据库添加一个关系图。关系图是一个可帮助理解和定义数据库的可视化工具。在关系图上，可以将表的一列拖至另一个表中来创建关系。在本练习中，将在 Review 表和 Genre 表之间创建关系。

(1) 再次打开 Planet Wrox Web 站点的 Database Explorer，右击 Database Diagrams 元素(在图 12-16 中可见)并单击 Add New Diagram。如果是首次将关系图添加到数据库中，可能会出现一个对话框，询问您是否希望 VWD 将您作为数据库的所有者。单击 Yes 按钮继续。即使没有看到这个提示也不必担心，它并不影响随后的创建。这一提示后面可能还会有另外一个提示，指明为了运用关系图，VWD 需要创建一些必需的对象。再次单击 OK 按钮继续。

- (2) 在随后的 Add Table 对话框中，选择在前面的“试一试”练习中创建的两个表(在单击每项的同时按住 Ctrl 键)，单击 Add 按钮将表添加到关系图中，然后单击 Close 按钮关闭 Add Table 对话框。
- (3) 若有必要，通过拖放排列关系图中的表，使它们互相靠近。
- (4) 在 Genre 表上，单击 Id 列(它应包含黄色钥匙图标，表明这是表的主键)的左页边距，然后将它拖至 Review 表的 GenreId 列上并释放鼠标键。
- (5) 弹出的两个对话框允许自定义关系的默认值。在最上边的窗口中，确认从表 Genre 中选中 Id 作为主键表，并在表 Review 中选中 GenreId 作为外键表。单击 OK 按钮关闭上面的窗口并确认选中的两列是在关系中。在剩下的对话框中(如图 12-21 所示)，注意如何将 Enforce Foreign Key Constraint 设置为 Yes。这一属性确保了如果还有评论与 Genre 表关联，那么就不能从该表中删除记录。单击 OK 按钮并关闭此对话框。

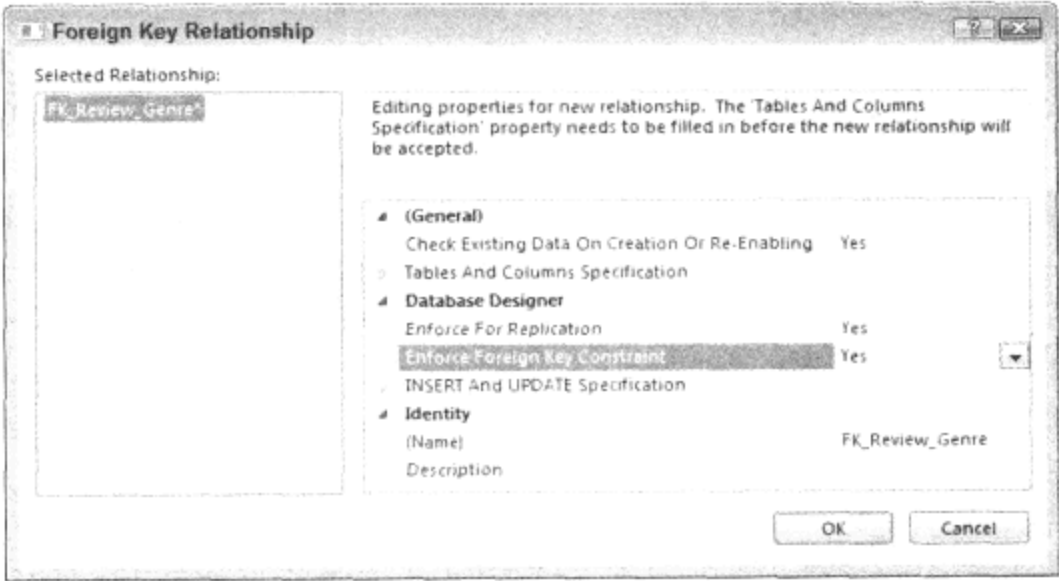


图 12-21

- (6) 现在，关系图窗口在两表之间显示一条线。在 Genre 表这一边，可以看到用一个黄色钥匙图标表明该表包含关系的主键。而在另一边，可以看到用无穷符号(∞)表明 Review 表有许多使用相同 GenreId 的记录。图 12-22 所示就是这个关系图。

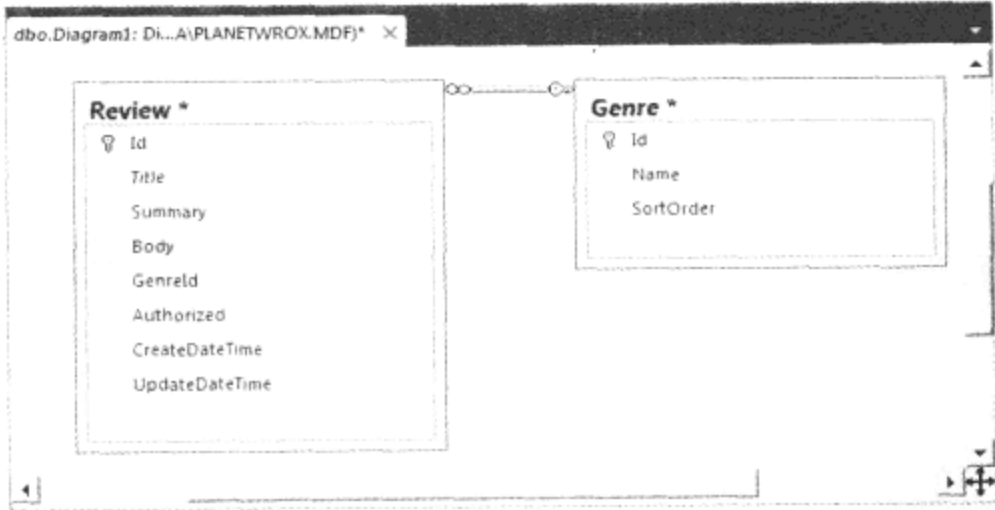


图 12-22

注意，这两个表之间的线并不一定总是指向正确的列。有时这很容易使人混淆，因为您可能认为实际上关联的是其他的列。为了确认关系中的列，可右击两个表之间的线并选择 Properties。Tables

And Columns Specification 选项会显示哪个列和表位于关系中，如图 12-23 所示。

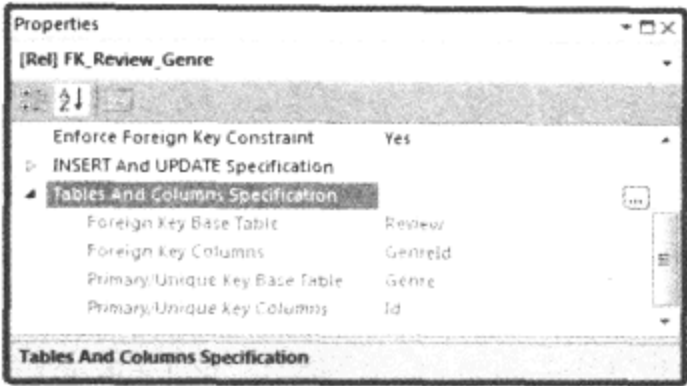


图 12-23

(7) 按 Ctrl+S 组合键保存对关系图的更改。可以将名称设置为其默认值 Diagram1 或是输入一个更具描述性的名称，如 Reviews and Genres，然后单击 OK 按钮。接着会出现一个警告，表明您将对 Review 和 Genre 表作更改。单击 Yes 按钮应用这一更改。

(8) 回到 Database Explorer，右击 Genre 表并选择 Show Table Data 命令。通过输入 Name 和 SortOrder 两列输入一些不同的流派。如果在 SortOrder 字段中按 Tab 键离开当前行，则该行就插入到了数据库中，并且 Id 列会用一个唯一的序号填充。最终得到的列表如图 12-24 所示。

Genre: Query(win...A\PLANETWROX.MDF) X			
	Id	Name	SortOrder
	2	Pop	12
	3	Jazz	8
	4	Hard Rock	3
	5	Indie Rock	7
	6	Punk	1
	7	Rock	2

图 12-24

(9) 使用 Show Table Data 命令从 Database Explorer 中打开 Review 表并输入一些评论记录。对于 GenreId，提供一些在向 Genre 表中插入记录时得到的新 ID。现在可以编辑 Title、Summary、Body 字段并设置 Authorized 为 True。记住，不必在日期列输入值。如果不输入，数据库会自动插入默认值。要注意您不能在 Id 列插入值。因为这个列是一个标识字段，数据库会自动提供其值。如果出现日期列缺值的错误，那么要确保在前一个“试一试”练习中输入了正确的默认值。完成一行输入后，单击该行的外部(例如其下面空的新行)，然后按 Ctrl+R 组合键将该行插入到表中。得到的记录列表将如图 12-25 所示，当然，列的内容可能有所不同。

Review: Query(win...A\PLANETWROX.MDF) X						
	Id	Title	Summary	Body	GenreId	Authorized
	36	Travelling the Face of the Globe by Oi Va Voi - A g...	Oi oi oi, the new album	NULL	4	True
	37	Battle for the Sun by Placebo - Possibly the best al...	Talks about the new PL...	NULL	5	True
	38	Sonic Youth: The Eternal - Takes time to get used ...	The latest SY CD	NULL	5	True

图 12-25

(10) 再次右击 Genre 表并选择 Show Table Data 命令。单击 Query Designer 工具栏上的“SQL 窗格”按钮，然后使用同一工具栏上的 Change Type 按钮创建一个删除查询。修改该查询，使之如下

所示：

```
DELETE FROM Genre WHERE Id = 5
```

这段代码将尝试删除 Indie Rock 流派。然而，由于存在与它相关的评论，因此该删除动作将不会执行。要确保 WHERE 子句中的 Id 与第(9)步中用于链接评论的流派 ID 中的一个相匹配。按 Ctrl+R 组合键执行该查询。VWD 现在将显示如图 12-26 所示的对话框，而不是从 Genre 表中删除记录。

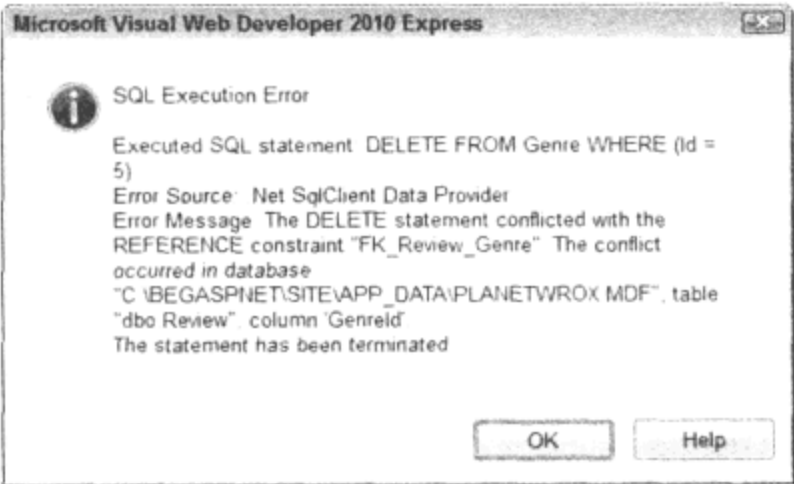


图 12-26

工作原理

如果在两表间创建了关系，那么在试图插入、更新或删除数据时，数据库将强制这一关系。在这个例子中，Review 表中的记录有一个位于 Genre 表中的流派。在试图从 Genre 表中删除一条记录时，由于数据库知道这一流派被 Review 表中的一条记录所使用，所以取消了删除操作。在第 15 章将会介绍如何在自己的 Web 站点上处理这种情况，以及如何向用户显示友好的错误提示。

现在我们已经了解了处理数据库的基本概念，第 13 章将介绍如何使用可用的 ASP.NET 数据控件来使用数据库。

12.6 有关数据库的实用提示

下面给出了一些有关处理数据库的实用提示：

- 由于数据库通常是 Web 站点的核心所在，所以需要仔细考虑其设计。特别重要的是，首先要有一个好的设计，然后在其上进行站点构建。如果有大量要访问数据库的页面，那么就较难对数据模型作更改，如删除表或重命名列。
- 始终要考虑表的主键。笔者喜欢给每个表指派一个 Id 列。基本数据类型为 int，作为标识列，它会自动给每条记录提供唯一的 ID。也可以考虑使用 uniqueidentifier 数据类型，它确保了跨数据库或应用程序边界的唯一性。
- 给数据库对象(如表和列)提供合乎逻辑的名称。避免使用像空格、下划线和破折号这样的字符。像 GenreId 这样的名称就比 colGen_ID_3 易读。
- 不要使用 SELECT * 从数据库中获取所有列。如果使用 SELECT *，一般总是想选取实际需要的多列。通过显式指定要检索的列，可以使意图更清晰，同时还可提高查询性能。

- 在适当时总是在表间创建关系。尽管查询在本章中看到的评论和流派时，表之间并没有关系，查询工作运行得也很好，但关系可帮助保证数据的质量。通过适当的关系，可以最小化出现孤立或不正确数据的可能性。

12.7 本章小结

处理数据库的能力是对 Web 开发技能的一项很好的补充。如今，大部分现代的动态 Web 站点都使用数据库，所以了解如何处理它们很重要。

要访问和操纵关系数据库中的数据，可以使用名为 SQL(Structured Query Language, 结构化查询语言)的语言。在其他元素中，该语言定义了 4 个重要的关键字，允许对数据库执行 CRUD (Create\Read\Update\Delete)操作。

SELECT 语句允许从一个或多个表中检索数据。要访问多个表，可以使用一种可用的 JOIN 类型定义表之间的关系。要限制查询返回的记录数，可以使用 WHERE 子句。而为了对查询返回的结果进行排序，可以使用 ORDER BY 子句。要在数据库中创建新记录，可以使用 INSERT 语句，还可使用 UPDATE 语句改变已有记录。最后，要删除不再需要的记录，可以使用 DELETE 语句。和 SELECT 和 UPDATE 语句一样，DELETE 也可采用可选的 WHERE 子句来限制被删除的记录数。

本章第二部分讲述了如何使用内置数据库工具创建新表以及表之间的关系。另外，还介绍了两表之间的关系如何保护数据不被破坏或孤立。

尽管本章的重点是需要编写用来访问数据库的 SQL，但第 13 章将讲到，在很多情况下，VWD 都能生成几乎全部的代码使数据库的访问变得很容易。当然，扎实掌握 SQL 知识可以帮助您理解和调整 VWD 所生成的代码。

12.8 练习

1. 试图从 Genre 表(该表在 Review 表中有一些匹配的记录)中删除一条记录失败的原因是什么？
2. 用 DELETE 语句尝试从 Review 表(该表将其 GenreId 设置为 Genre 表中已有流派的 Id)中删除一条记录成功的原因是什么？
3. 假设您要清理数据库，从 Review 表中删除所有 Id 小于或等于 100 的所有记录，该如何编写该 SQL 语句？
4. 假设您想删除 ID 为 4 的流派。但在删除之前，您想将指派给这一流派的评论重新指派给另一个 ID 为 11 的流派，应该采用怎样的 SQL 语句来完成这一操作？
5. 编写一条将 Rock 流派更新为可以读取 Punk Rock 流派的 SQL 语句。至少存在两种编写 WHERE 子句的方法来实现这个语句。

练习的答案见附录 A。

本章要点回顾

CRUD	用于处理数据库中数据的 4 种基本 SQL 操作—— Creat(创建)、Read(读取)、Update(更新)和 Delete(删除)
外键	在一个表中标识的指向另一个表中主键的列，用于强制参照完整性
标识列	指派给新记录的自动序号
连接	允许通过在查询中描述两个或多个表之间的关系来查找相关数据
主键	在一个表中用于唯一标识该表中记录的一个或多个列
关系数据库	数据库的一种，在这种类型的数据库中，数据是存储在单独的像电子表格这样的表中，并且表之间还可以相互引用
关系	在一个或多个表中定义的关系，有助于强制参照完整性
表	数据库中的一种对象，用于存储数据

本章要点

- 使用 GridView、DetailsView 和 SqlDataSource 等控件显示、插入、编辑和删除数据
- 使用 ASP.NET 有效性验证控件创建一个丰富的界面，使用户在插入和编辑数据的同时仍然能够维护数据完整性
- 将连接字符串存储到应用程序中以使它们易于更新的最好方法

本章将学习如何使用随 ASP.NET 发布的流行数据控件来显示、插入、更新和删除数据。除了运用可用于在 Web 页面中显示和编辑数据的可视控件外，也将学习如何使用充当数据库和 ASPX 页面之间桥梁的 SqlDataSource 控件。

首先要介绍的是可用的数据控件。

13.1 数据控件

为了有效地处理系统中的数据，ASP.NET 提供了两组数据感知(data-aware)控件：数据绑定控件(data-bound control)和数据源控件(data source control)。

第一组中包含了可用于显示和编辑数据的控件，如用户界面中的 GridView、Repeater 和 ListView 控件。数据源控件用于从数据源(如数据库或 XML 文件)中检索数据，然后将这一数据提供给数据绑定控件。图 13-1 显示了 Toolbox 的 Data 类别中可用的数据控件的完整列表。

下面 3 节是对 Data 类别下所有控件的一个概述。在本章剩余部分将详细介绍其中的一些控件以及如何使用它们。

13.1.1 数据绑定控件

图 13-1 所示的 Toolbox 中前 7 个控件就是所谓的数据绑定控件。使用它们在 Web 页面上显示和编辑数据。GridView、DataList、ListView 和 Repeater 都可以同时显示多条记录，因此它们通常被称为列表控件(List Control)。而 DetailsView 和 FormView 设计为一次显示一条记录。DataPager 是为

ListView 控件提供分页功能的辅助控件。

1. 列表控件

由于 ASP.NET 提供了多个用于显示记录列表的控件，您可能想知道何时选用哪种工具。GridView 是一个非常多功能的控件，它支持自动分页(记录被划分到多个“页面”中)、排序、编辑、删除和选择。它像一个带有行和列的电子表格那样呈现数据，其中每行包含一个完整的记录。尽管有许多种可以样式化这些行和控件的外观(第 15 章会作详细介绍)的方法，但不能从根本上改变表现数据的方式。另外，GridView 并不允许直接在底层数据源中插入记录。

图 13-2 所示为一个典型的 GridView。

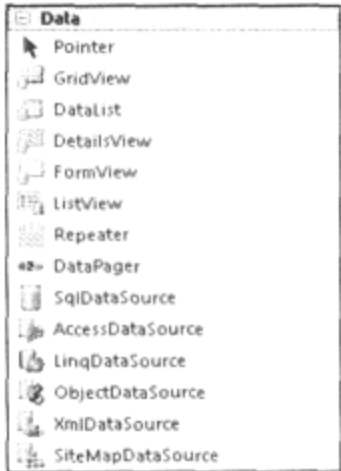


图 13-1

	id	Title	Date Created
Edit Delete Select	9	The National - Boxer	11/29/2007 10:10 PM
Edit Delete Select	10	P.J. Harvey - White Chalk. Wow! really something new	12/12/2007 1:08 PM
Edit Delete Select	11	Interpol - Our Love to Admire	12/12/2007 1:29 PM
Edit Delete Select	12	Yeah Yeah Yeahs - The Is Is EP	12/20/2007 10:36 PM
1 2 3 4 5 6			

图 13-2

DataList 控件使得不仅可以像 GridView 那样以行表现数据记录，也可以以列的形式表现，从而可以创建一种矩阵形式的数据表现方法。另外，它也允许通过一组模板定义数据的外观。不过，它不支持分页和排序，不允许插入新记录或更新、删除已有记录。

Repeater 控件在输出到浏览器的 HTML 方面提供了最大的灵活性，因为该控件本身并不添加任何 HTML 到页面输出中。因此，它常用于 HTML 有序列表或无序列表(和)，以及其他列表形式(在其中，您不希望有不必要的 HTML 与您自己的相混合)。可以通过控件提供的大量模板定义整个客户端标记(markup)。不过这种灵活性的代价很高：该控件没有分页、排序和修改数据的内置功能。在第 14 章中将有 Repeater 控件的更多介绍。

ListView 是在 ASP.NET 3.5 中引入的，它最好地合并了 GridView、DataList 和 Repeater。ASP.NET 4 中对 ListView 做出了一些改动，使它更易于使用。类似于 GridView，它支持数据编辑、删除和分页。像 DataList 那样，它支持多列和多行布局，而且还像 Repeater 那样允许完全控制控件生成的标记。在第 14 章中，将介绍有关 ListView 控件的更多信息。

ASP.NET 4 使用一个 ClientIDRowSuffix 属性扩展了列表控件，该属性允许指出一些特定的列，这些列的值用来基于数据库中的数据创建唯一的客户端 ID。为此，需要将在前面章节中看到的 ClientIDMode 属性设为 Predictable。

2. 单项控件

DetailsView 和 FormView 控件有些类似，因为它们都只能一次显示一条记录。DetailsView 使用内置的表格格式显示数据，而 FormView 使用模板来定义数据的外观。图 13-3 所示就是一个简单的、

基于模板的 DetailsView 的例子。

Id	1
Title	Sam's Town by The Killers - Really the best album it
Summary	Is the new album by the Killers - Sam's Town, really the best album of the last 20 years? Read this review to find out!
CreateDateTime	2007/11/23
Update Cancel	

图 13-3

ASP.NET 4 中对 FormView(以及将在第 16 章看到的一些登录控件)添加了一个新的 RenderOuterTable 属性。当把这个属性设为 True(它的默认值为 False, 所以需要显式设置为 True)时, 控件不会生成包装的 HTML <table>元素。这样就会生成更少的代码和更清晰的 HTML。这两种控件都允许为不同的情形定义模板, 如数据的只读显示和数据插入及更新。本章后面将介绍如何定制这些模板。

3. 分页控件

另外一个有用的控件是 DataPager, 它可以在其他控件上分页。目前, 它只能用于扩展 ListView 控件, 但随着 .NET Framework 未来版本的发布, 这一情况将会改观。第 14 章将讨论 ListView 和 DataPager 控件。

要使数据绑定控件显示有用的内容, 需要为它们指派数据源(data source)。要将这一数据源绑定到控件上, 有两种主要的方法: 可以将数据赋予控件的 DataSource 属性, 或是使用一个单独的数据源控件。在后续章节中, 将了解如何使用 DataSource 属性。下面主要介绍不同的数据源控件。

13.1.2 数据源控件

Toolbox 的 Data 类别下包含了 7 个不同的可用于绑定数据到数据绑定控件的数据源控件。XmlDataSource 和 SiteMapDataSource 用于绑定分层的、基于 XML 的数据到这些控件。在第 7 章创建站点地图时曾介绍过如何使用 SiteMapDataSource。

AccessDataSource 控件用于在 Web 页面中显示 Microsoft Access 数据库的数据。它非常简单, 在某种程度上类似于 SqlDataSource 控件, 因为它允许处理来自数据库的数据。但不同之处在于它只针对 Microsoft Access 数据库进行优化。

ObjectDataSource 控件允许将数据绑定控件连接到应用程序中的单独对象。不是将数据感知的控件直接连接到数据库, 而是将带有自定义对象的单独层的数据与它们绑定。如果您想了解该控件的更多信息, 可以参阅清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C#和 VB.NET (第 7 版)》一书(ISBN: 978-7-302-23524-8)。

最后 3 个控件是 SqlDataSource、EntityDataSource 和 LinqDataSource。本章和第 14 章将会分别讨论前两个控件。LinqDataSource 用作 LINQ to SQL 的数据源。LINQ to SQL 是一种类似于第 14 章将会讨论的 ADO.NET Entity Framework 的技术。因为 Microsoft 现在大力推广的是 Entity Framework, 而不是 LINQ to SQL, 所以本书不会讨论 LinqDataSource 控件。

QueryExtender 控件用作 LinqDataSource 和 EntityDataSource 控件的增件, 因为它可以用来创建丰富的过滤界面, 从而能够搜索特定的数据, 而不必手动编写大量代码。

13.1.3 其他数据控件

Toolbox 中的最后一个控件是 Chart 控件。它最初是作为 Visual Studio 2008 的一个增件发布的，但是现在已经完全集成到了 Visual Studio 2010 中。它用来绘制从简单的条形图到 3D 饼图和折线图的各种图形。这个控件不在本书的讨论范围之内，所以这里不再详细介绍它，但是可以从 <http://tinyurl.com/nsnbvv> 上找到一系列详细讨论该控件的文章。

在下一节将会看到如何使用 SqlDataSource 和 GridView 从数据库中检索和显示数据。后面的部分和章节将更深入地讨论其他数据控件。

13.2 联合使用数据源和数据绑定控件

SqlDataSource 控件允许快速创建可操作的、数据库驱动的 Web 页面。不需要编写大量代码，就可以创建能执行 CRUD(Create、Read、Update 和 Delete，即创建、读取、更新和删除数据)这 4 种操作的 Web 页面。尽管其名称暗示了只可以访问 Microsoft 的 SQL Server 数据库，但事实并非如此。该控件还可以访问其他数据库，如 Oracle 或 MySQL。

13.2.1 使用 GridView 显示和编辑数据

为了使您了解如何一起使用 SqlDataSource 控件和数据绑定控件，接下来的“试一试”练习创建了一个非常简单的数据驱动的 Web 页面，该页面允许管理存储在数据库的 Genre 表中的记录。本章假定有一个 PlanetWrox.mdf 数据库，其 Genre 和 Review 表位于 App_Data 文件夹中。还假定了它们每个表都包含一些记录。如果您没有遵循第 12 章中的步骤，或是并不能确定是否正确地执行了第 12 章介绍的步骤，则从本章代码下载(位于 C:\BegASPNET\Resources\Chapter 13)的 Resources 文件夹中获取 PlanetWrox.mdf 文件的副本，并将它放置在 Planet Wrox Web 站点的 App_Data 文件夹中，覆盖已有数据库(如果已有)。如果您自己的副本中没有包含大量的评论和流派记录，那么复制一份 PlanetWrox.mdf 数据库文件也是个不错的主意。这样就有了一组很好的样例记录可供使用。

试一试

使用 GridView 和 SqlDataSource 控件

在本练习中，首先构建 Web 站点的 Management 部分，这将是在 Web 站点上管理像评论和流派这些内容的主要入口点。现在，您在本部分所创建的页面可供 Web 站点的所有用户使用，但第 16 章将显示如何阻止不是管理员的用户访问这一文件夹。

通过本练习，您将了解如何从 Database Explorer(Visual Studio 商业版中的 Server Explorer)中拖放一个表到页面上，用 VWD 创建一个 Web 用户界面，通过为 GridView 和 SqlDataSource 自动生成所需的代码来管理数据库中的项。在本书后面的练习中，将了解如何手动产生这一行为，从而对生成的代码能有更多的控制。

(1) 从 VWD 的 C:\BegASPNET\Site 下打开 Planet Wrox Web 站点。

(2) 右击 MasterPages 文件夹，选择 Add New Item 项，为站点添加一个名为 Management.master 的新母版页。确保使用了正确的编程语言，并且它不基于已有的母版页。另外，通过检查 Place Code

in Separate File 选项，确保它使用了 Code Behind 文件。

(3) 将<form>元素中的 HTML 改为下列代码，创建两个互相紧挨着的浮动的<div>元素。一个<div>元素包含 Management 部分的一个简单的列表菜单，另一个<div>元素包含 ContentPlaceHolder 控件，允许内容页提供自定义内容：

```
<form id="form1" runat="server">
<div>
  <div style="width: 200px; float: left;">
    <ul>
      <li><a href="~/Management/Default.aspx" runat="server">
        Management Home</a></li>
      <li><a href="~/Management/Genres.aspx" runat="server">Manage Genres</a></li>
    </ul>
  </div>
  <div style="width: 750px; float: left;">
    <asp:ContentPlaceHolder ID="cpMainContent"
      runat="server"></asp:ContentPlaceHolder>
  </div>
</div>
</form>
```

可以忽略 VWD 显示的缺少页面的警告，因为后面会添加它们。保存并关闭该母版页。

(4) 添加一个新文件夹到该站点的根文件夹中，命名为 Management。右击该新文件夹，选择 Add New Item 命令，创建一个名为 Default.aspx 的新 Web 窗体。通过选中 Select Master Page，确保该页面基于刚才创建的 Management.master 文件。添加一些文本到 cpMainContent 内容块中，欢迎用户来到 Web 站点的 Management 部分：

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" runat="Server">
  <h1>Planet Wrox Management Section</h1>
  <p>Welcome to the Management section of this web site. Please choose an item
    from the menu on the left to continue.</p>
</asp:Content>
```

使该页面的标题为 Planet Wrox-Management-Home。

(5) 在 Management 文件夹中创建另一页面并命名为 Genres.aspx。将其基于同一个母版页，然后将其标题改为 Planet Wrox-Management-Genres。

(6) 将页面切换到 Design 视图并确保 Database Explorer(或 Server Explorer)窗口打开。如果没有看到 Planet Wrox 数据库列出，可以参阅第 12 章的第一个“试一试”练习，其中解释了如何建立连接。记住，在针对本章的 Resources 文件夹中有一个数据库包含了本章将使用的表。

(7) 展开 PlanetWrox.mdf 数据库至 Tables 节点，然后从 Database Explorer 中拖放 Genre 表到 Genres 页面的 cpMainContent 区域。VWD 会自动创建一个 GridView 和一个 SqlDataSource。

(8) 在为 GridView 控件自动打开的 Smart Tasks 面板上(如果没有打开，可单击该控件右上角的灰色箭头或右击该控件并选择 Show Smart Tag 命令)，选择所有可用选项，如图 13-4 所示。

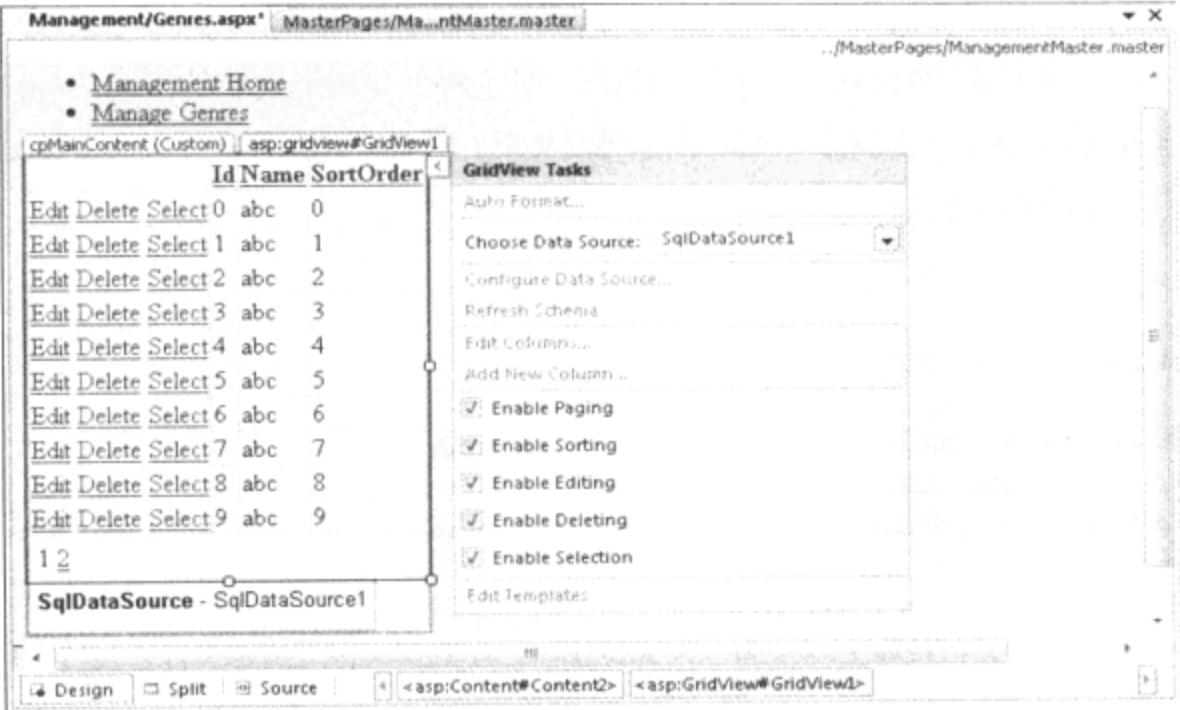


图 13-4

(9) 右击 Solution Explorer 中的 Management 文件夹并选择 Add New Item 命令。选择 Web Configuration File 并单击 Add 按钮，添加一个只应用于 Management 文件夹的 web.config 文件。在打开的文件中，在<system.web>下面添加一个<pages>元素，并将 theme 设置为空字符串，从而禁用站点的整个 Management 部分的主题：

```
<configuration>
  <system.web>
    <pages theme=""></pages>
  </system.web>
</configuration>
```

(10) 保存所有的更改，然后在浏览器中请求 Management 文件夹中的 Genres.aspx 页面(如图 13-5 所示)。结果将看到一个带有 Genre 表中流派的网格。左列的链接可用来编辑、删除和选择相关的流派。注意，不可以删除有一个或多个评论与之相连的流派。如果这样尝试，会得到一个错误。第 15 章更深入地研究了改变用户界面(UI)来禁用 Delete 链接的方法，这样用户就不会意外地点击它们。

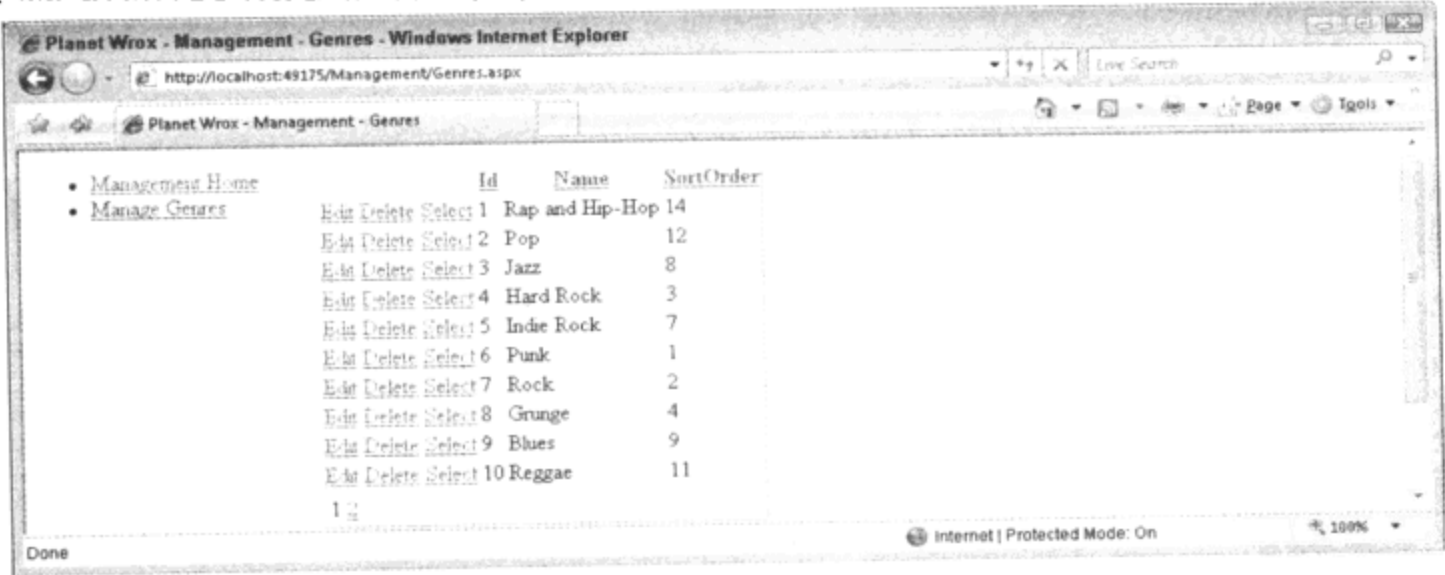


图 13-5

如果评论列表位于左侧菜单之下，则可能需要使浏览器窗口稍宽一些。

(11) 可以单击图 13-5 中可见的列标题，如 Name 和 SortOrder，以便在该列上对网格中的数据
进行排序。如果再次单击同一标题，数据就会以相反的顺序排列。

(12) 单击某一流派的 Edit 链接，更改出现在文本框中的名称并单击 Update 按钮。这时，GridView
将显示新的名称。

工作原理

您在本例中没有编写多少代码，但通过拖放数据库表获得了大量功能。要想知道其工作原理，
可看一下 VWD 生成的源代码。首先，看一下 SqlDataSource 控件的标记：

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%%$ ConnectionStrings:PlanetWroxConnectionString1 %>"
    ProviderName="
        <%%$ ConnectionStrings:PlanetWroxConnectionString1.ProviderName %>"
    DeleteCommand="DELETE FROM [Genre] WHERE [Id] = @Id"
    InsertCommand="INSERT INTO [Genre] ([Name], [SortOrder]) VALUES (@Name,
        @SortOrder)"
    SelectCommand="SELECT [Id], [Name], [SortOrder] FROM [Genre]"
    UpdateCommand="UPDATE [Genre] SET [Name] = @Name, [SortOrder] = @SortOrder
        WHERE [Id] = @Id">
<DeleteParameters>
    <asp:Parameter Name="Id" Type="Int32" />
</DeleteParameters>
<InsertParameters>
    <asp:Parameter Name="Name" Type="String" />
    <asp:Parameter Name="SortOrder" Type="Int32" />
</InsertParameters>
<UpdateParameters>
    <asp:Parameter Name="Name" Type="String" />
    <asp:Parameter Name="SortOrder" Type="Int32" />
    <asp:Parameter Name="Id" Type="Int32" />
</UpdateParameters>
</asp:SqlDataSource>
```

这里有一些值得注意的事情。首先，可以看到 ConnectionString 和 ProviderName 特性指向已在
web.config 文件中定义的连接字符串。在下一节中将看到更多与之相关的内容，包括对用于连接字符
串的<%%\$ %>语法的说明。

接着是 4 个命令，它们每个都包含一个 SQL 语句用于 CRUD 操作中的一种：INSERT、UPDATE
和 DELETE 命令包含参数(用@前缀符号进行标识)。在运行时，控件被要求执行相关的数据操作，
这些参数被运行时值所取代。SqlDataSource 控件跟踪*Parameters 集合中的相关参数。例如，
<DeleteParameters>元素包含了一个单独的参数表示流派的 Id(主键)：

```
<DeleteParameters>
    <asp:Parameter Name="Id" Type="Int32" />
</DeleteParameters>
```

注意在 SQL 语句中，参数的 Name 与参数排在一起：

```
DeleteCommand="DELETE FROM [Genre] WHERE [Id] = @Id"
```


而 `SqlDataSource` 控件自己在这阶段并不做多少工作。它需要一个数据绑定控件来告诉自己执行什么数据操作。在这个“试一试”练习中，这个数据绑定控件是 `GridView`，使用下列代码定义：

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True" AllowSorting="True"
    AutoGenerateColumns="False" DataKeyNames="Id" DataSourceID="SqlDataSource1"
    EmptyDataText="There are no data records to display.">
    <Columns>
        <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
            ShowSelectButton="True" />
        <asp:BoundField DataField="Id" HeaderText="Id" ReadOnly="True"
            SortExpression="Id" />
        <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name" />
        <asp:BoundField DataField="SortOrder" HeaderText="SortOrder"
            SortExpression="SortOrder" />
    </Columns>
</asp:GridView>
```

这个 `GridView` 控件包含了一些重要的特性。首先，`DataKeyNames` 特性告诉 `GridView` 数据库中记录的主键是什么。它需要这一信息来唯一标识网格中的记录。

`DataSourceID` 特性指向您之前所看到的 `SqlDataSource` 控件，而 `AllowPaging` 和 `AllowSorting` 在 `GridView` 上启用它们的相关功能。

在 `<Columns>` 元素下，可看到许多字段。首先是 `CommandField`。它是 `GridView` 中的一列，使用户为应用 `CommandField` 的行执行一个或多个动作。它在浏览器中表示为一个或多个文本链接或按钮。在这个例子中，`ShowDeleteButton`、`ShowEditButton` 和 `ShowSelectButton` 都被设置为 `True`。这就为网格提供了图 13-5 所示的功能。当单击其中一个由 `CommandField` 创建的链接时，它们会在服务器上触发一个命令。例如，单击 `Edit` 链接会使 `GridView` 进入编辑(Edit)模式，这样就可以编辑选择的记录。注意，单击 `Select` 链接看上去并没有改变 `GridView`。第 15 章将介绍如何创建 `GridView` 的样式，这样就可以显著地改变控件的外观，包括在视觉上区分选择的行与其他行。

如果想让 `GridView` 呈现按钮而不是链接，需要设置 `ButtonType` 为 `Button`：

```
<asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
    ShowSelectButton="True" ButtonType="Button"></asp:CommandField>
```

其他 3 个字段就是所谓的绑定字段(bound field)，它们直接通过 `DataField` 特性映射到数据库中 `Genre` 表的列，因此 `GridView` 知道数据的显示位置。

`GridView` 和 `SqlDataSource` 控件联合起来检索和修改底层数据源中的数据。为了了解其工作原理，这里提供了一个在浏览器中请求 `Genres` 页面并修改某个流派时发生的事件的摘要。

- (1) 在浏览器中请求页面，然后该页面开始其生命周期。
- (2) `GridView` 知道它用来检索和显示数据，因为它有一个指向 `SqlDataSource` 控件的 `DataSourceID` 特性。它联系这个数据源控件并向其请求数据。然后 `SqlDataSource` 连接数据库并触发其 `SelectCommand`，`SelectCommand` 是一个 SQL 语句，该 SQL 语句从数据库中的 `Genre` 表中选择 `Id`、`Name` 和 `SortOrder`：

```
SelectCommand="SELECT [Id], [Name], [SortOrder] FROM [Genre]"
```

- (3) 在 `SqlDataSource` 从数据库中接收到请求的数据时，它将它们转交给 `GridView`，后者用在

<Columns>元素中建立的绑定字段为数据创建一个 HTML 表。对于 CommandField 列中的每个 Edit、Delete 和 Select 链接，GridView 通过在 View State 中存储页面中显示的每条记录的唯一 ID，从而实现了对这些 ID 的跟踪。

(4) 只要一单击 Edit 链接，页面就开始回发。GridView 通过查看相关的 DataKeyName，并从 View State 中检索记录的 ID，就可以知道单击的是哪一行。接着它再次请求 SqlDataSource 触发 SelectCommand，从数据库中获取最新数据，然后使选择的行进入编辑模式，这样您就可以更改相关详细信息了。在单击 Update 链接时，GridView 从 TextBox 控件中收集新值，然后再次与 SqlDataSource 联系。

(5) 对于 SqlDataSource 的<UpdateParameters>元素中的每个参数，GridView 都提供一个值。它从选择的行中检索流派的 Id，然后从页面的 TextBox 控件中检索新的 Name 和 SortOrder 值。

(6) 运用 Id、Name 和 SortOrder 的相关数据，SqlDataSource 对数据库执行其 UpdateCommand 命令：

```
UpdateCommand="UPDATE [Genre] SET [Name] = @Name,
                [SortOrder] = @SortOrder WHERE [Id] = @Id"
```

每个带有@前缀符号的参数都用 GridView 提供的值填充。发送到数据库的 SQL 语句如下所示：

```
UPDATE [Genre] SET [Name] = 'New Name', [SortOrder] = 1 WHERE [Id] = 1
```

表名和列名被括在一对方括号([])中。这一做法在这里不是必需的，但如果列名包含空格，或者表名或列名匹配保留字时，这样做就很有用。

(7) 最后，GridView 通过再次要求 SqlDataSource 执行其 SelectCommand 命令，刷新页面上的数据。这样，GridView 就用所作的更新显示最新的数据。

其他两个命令的工作方式与此相同，它们将自己的 SQL 命令发送到数据库中。

在练习的最后，添加一个新的 web.config 文件到 Management 文件夹中并重新设置应用于 Management 部分的所有页面的主题(theme)。删除主题后，可以更容易地集中于 Management 部分的功能，而不会被布局问题所分心。在第 15 章将为 Management 文件夹再创建一个主题并将它应用于 Management 文件夹下的 web.config 文件。那样的话，管理页面将获得一个不同于前端页面的外观。

现在已经看到了如何显示、编辑和删除数据，接下来应该学习如何使用 DetailsView 控件在数据库中插入新记录。

13.2.2 使用 DetailsView 插入数据

与使用 GridView 控件显示、更新和删除数据一样，使用 DetailsView 控件插入数据也是非常简单的。同样，DetailsView 支持大量可以自定义控件不同状态下外观的模板。例如，它有<FooterTemplate>、<HeaderTemplate>和<PagerTemplate>元素来定义控件上部和下部的外观。另外，还有一个<Fields>元素，可用来定义在控件中出现的行，与 GridView 的<Columns>元素很相似。

DetailsView 可以在一些不同的模式下显示数据。首先，它可以以只读模式显示已有的数据。另外，它可用于插入新数据和更新已有数据。通过将 DefaultMode 属性分别设置为 ReadOnly、Insert 和 Edit，可以控制 DetailsView 的模式。接下来将了解如何配置 DetailsView 和设置 DefaultMode 属性。

试一试

使用 DetailsView 控件插入数据

在本练习中，将了解如何使用 DetailsView 控件向 Genre 表中插入新记录。和 GridView 的示例一样，这一练习不需要您编写任何代码。您所要做的就是拖放一些控件，设置一些属性。显然，这样的免写代码的页面有其局限性，很少能用于更高级的场景。因此，本章后面将介绍如何扩展和自定义这些控件。

- (1) 回到 VWD 中的 Genres.aspx 页面并确保其在 Design 视图下。
- (2) 从 Toolbox 的 Data 类别中拖动一个 DetailsView 到 GridView 的下面。如果无法放到 GridView 的下面，但是可以放到 SqlDataSource 控件的上面，那就将它放到 SqlDataSource 上。VWD 将在 SqlDataSource 的标记之前添加被拖放控件的标记。
- (3) 如果控件的 Smart Tasks 面板没有自动打开，那么就打开它，并通过从 Choose Data Source 下拉列表中选择名称将它与 SqlDataSource 关联。
- (4) 在同一 Smart Tasks 面板中，选择 Enable Inserting 项。
- (5) 按 F4 键打开控件的 Properties 面板，然后定位到 Behavior 类别中的 DefaultMode 属性。将该属性设置为 Insert。DetailsView 的代码现在如下所示：

```
<asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"
    DataKeyNames="Id" DataSourceID="SqlDataSource1" DefaultMode="Insert"
    Height="50px" Width="125px">
  <Fields>
    <asp:BoundField DataField="Id" HeaderText="Id" InsertVisible="False"
        ReadOnly="True" SortExpression="Id" />
    <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name" />
    <asp:BoundField DataField="SortOrder" HeaderText="SortOrder"
        SortExpression="SortOrder" />
    <asp:CommandField ShowInsertButton="True" />
  </Fields>
</asp:DetailsView>
```

- (6) 保存对页面的更改，然后按 Ctrl+F5 组合键在浏览器中打开它。在 GridView 下，将看到可用于插入新流派的控件(如图 13-6 所示)。

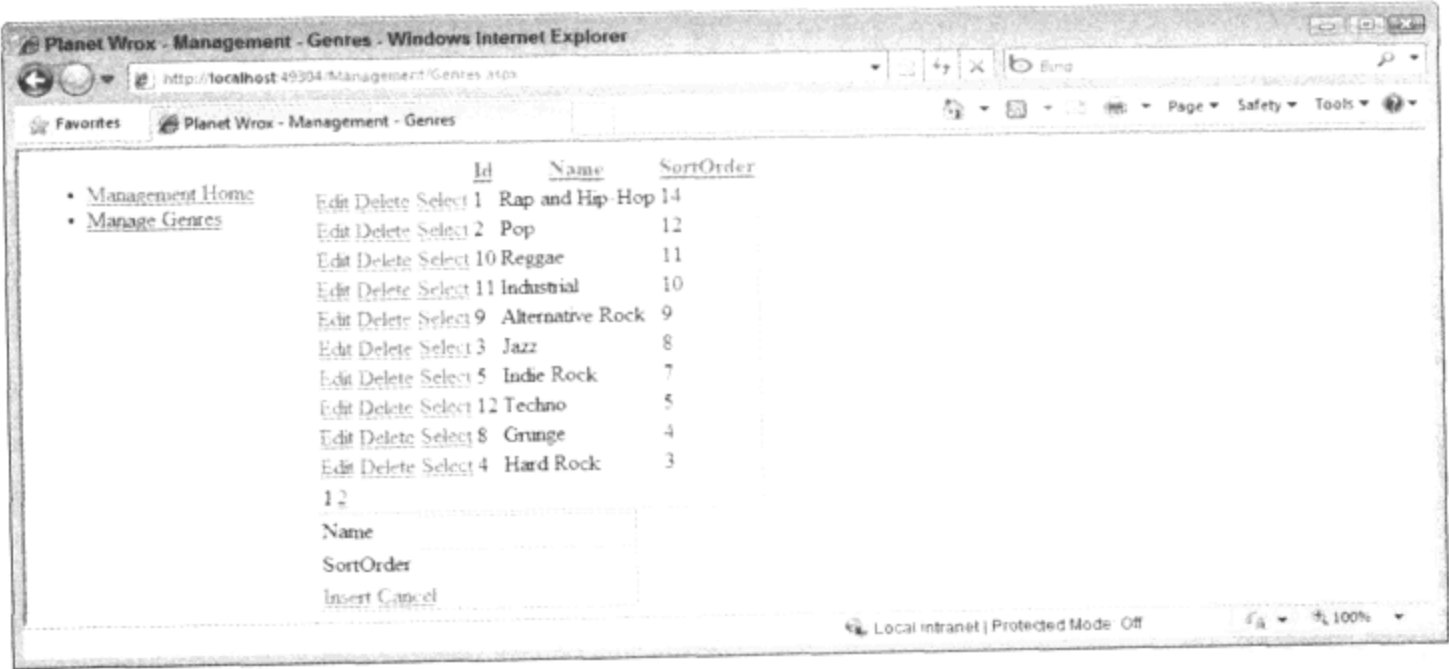


图 13-6

(7) 插入一个新流派，如 Disco 或 Dance。确保输入了名称和排序顺序(为一个数字)，然后单击 Insert 链接。您可能需要单击屏幕底部 Pager 栏的一个数字进入 GridView 的最后一页分页，以便查看新记录。

工作原理

和其他数据绑定控件一样，通过设置 DataSourceID 属性将 DetailsView 与数据源控件相关联。由于页面上已经有了一个可用的 SqlDataSource 控件，所以只需重用它。DetailsView 对于只读、插入和编辑模式提供了不同的视图。通过将 DefaultMode 设置为 Insert，可以强制控件切换至 Insert 模式，这意味着您自动获得了一个用于输入流派详细信息的 UI，以及 Insert 和 Cancel 链接。DetailsView 控件实际上非常智能。如果将它指向 SqlDataSource 控件，它就可以识别 DataKeyNames 属性，并将它设置为 Id:

```
<asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"
    DataKeyNames="Id" DataSourceID="SqlDataSource1" ...
```

它还了解 Id 列是数据库的标识列，因此通过将 InsertVisible 设置为 False，将它在图 13-6 中的 Insert 屏幕中隐藏。由于数据库是自动生成这一 ID 的，所以让用户输入其值是毫无意义的。

在您输入一些值并单击 Insert 链接后，就会发生与使用 GridView 更新类似的过程。DetailsView 从页面的控件(Name 和 SortOrder)中收集相关信息，并将它们转发给 SqlDataSource。SqlDataSource 控件将这些新值传递给 INSERT 语句的参数，然后将该命令发送给数据库，从而将新记录插入到 Genre 表中。如果在没有输入名称或者排序顺序的情况下单击 Insert 链接，就会得到一个错误。在本章和后面的章节中，将看到如何修改数据绑定控件来包含有效性验证功能。

在本章前面将 Genre 表放到 Genres.aspx 页面上时，VWD 不仅为您创建了一些控件，还在 web.config 文件中存储了关于数据库的信息。下一节将解释其工作原理，以及为什么这种行为很重要。

13.2.3 在 web.config 文件中存储连接字符串

在首次将 Genre 表放在页面上时，VWD 会创建一个 SqlDataSource 控件。为了告知该控件要访问什么数据库，它还在 web.config 文件中的<connectionStrings>元素下创建了一个连接字符串，并将 SqlDataSource 指向该连接字符串。web.config 中的设置如下所示:

```
<connectionStrings>
  <add name="PlanetWroxConnectionString1" connectionString=
    "Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\PlanetWrox.mdf;
    Integrated Security=True;Connect Timeout=30;User Instance=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

然后该 SqlDataSource 访问这个连接字符串:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%%$ ConnectionStrings:PlanetWroxConnectionString1 %>"
```

这一代码使用表达式语法(expression syntax)来引用 web.config 中的连接字符串。它有效地向 web.config 文件请求了侦听名称 PlanetWroxConnectionString1 的连接字符串。

除了使用<%\$ %>将控件值绑定到像连接字符串这样的资源的表达式语法之外，还会遇到使用<%# %>的类似语法。这被称为数据绑定表达式语法(data-binding expression syntax)，允许将控件值绑定到来自数据源(如数据库)的数据中。在本章和后面两章将介绍更多关于数据绑定表达式语法的内容。

将连接字符串存储到 web.config 中是一个非常好的实践。这样，可以将连接字符串集中于一个单独的位置，在数据库发生改变时(例如从开发环境切换到生产服务器)可以更容易地修改它们。不要将连接字符串直接存储到 Code Behind 文件中或页面的标记部分。如果那样的话，当必须更改连接字符串时，就需要费力地在站点的所有页面中寻找连接字符串。

到目前为止所使用的 SQL Server Express Edition 支持在需要时即时运用与 SQL Server 相连的数据库。看一个实际的连接字符串，了解其工作原理：

```
Data Source=.\SQLEXPRESS;  
AttachDbFilename=|DataDirectory|\PlanetWrox.mdf;  
Integrated Security=True;  
Connect Timeout=30;  
User Instance=True
```

该连接字符串由 5 部分构成，它们都位于 config 文件中的一行内。第一部分的值包含了数据源来标识所针对的 SQL Server，可再分为两个小部分。反斜杠前面的部分(这里是句点)标识服务器。句点表示本地计算机(尽管也可以使用(local))。另外，可以在此输入服务器的名称来代替句点。句点后面的部分只有在使用 SQL Server 的命名实例(named instance)的时候才需要。由于可在同一计算机上安装多个 SQL Server 实例，所以实例名称用于标识一个唯一的 SQL Server 实例。在安装 SQL Server 2008 Express 时，会获得默认实例名 SqlExpress。因此，要在本地计算机上引用 SQL Server 2008 Express 实例，其格式为\SqlExpress 或(local)\SqlExpress。

AttachDbFileName 包含了到 SQL Server Express 数据库的路径。|DataDirectory|占位符在运行时展开为 App_Data 文件夹的完整物理路径。因此，在页面加载和 SqlDataSource 需要连接到数据库时，它将打开 C:\BegASPNET\Site\App_Data\中的 PlanetWrox.mdf 文件。作为 AttachDbFileName 的另一选择，你也会在其他连接字符串中看到 Initial Catalog。Initial Catalog 只是指向所使用的 SQL Server 上可用的数据库，如 PlanetWrox。附录 B 中对此作了更多介绍。

连接字符串的最后 3 个部分与安全性和超时有关。通过 Integrated Security，Web 服务器使用的账户用于连接数据库。在 VWD 和内置 Web 服务器中，这一账户是用于登录计算机的账户。如果使用的是 IIS，这一账户则是一个特殊的 ASP.NET 账户，在 Windows 的大多数版本上都称为 Network Service。介绍部署的第 19 章和解释如何连接到 SQL Server 的附录 B 深入讨论了与安全相关的问题。

Connect Timeout 定义了应用程序等待连接初始化完成的时间。当超时时间到达，但是仍然没有建立起连接时，就会产生错误。

到现在为止，您已经学习了第 12 章中引入的大部分数据库概念。了解了创建(使用 Insert 模式下的 DetailsView)、读取(使用 SelectCommand 和 GridView)、更新(在 GridView 中内联和使用 UpdateCommand)和删除(使用 GridView 中的删除选项和 DeleteCommand)操作。而且，知道了可以只使用一个设置在 GridView 中启用排序。我们还没有介绍的内容是筛选(filtering)，这是一种限制表现在页面上的数据的方式。在下一节中，将介绍如何创建一个筛选器来显示属于某个特定流派的评论。这一工作在名为 Reviews.aspx 的新页面的 Management 部分进行，这一页面是管理 Web 站点中

评论的主要入口点。后续部分将基于这一部分，逐步为 Reviews 页面扩展更多有用的功能。

13.2.4 筛选数据

正如在第 12 章所学到的，数据筛选是用 WHERE 子句完成的。幸运的是，VWD 和 ASP.NET 提供了很多使创建筛选变得非常容易的工具。为了筛选数据，SqlDataSource 控件(和其他数据源控件)有一个<SelectParameters>元素，允许在运行时提供用于筛选的值。这些值可来自于不同的源，如表 13-1 所示。

表 13-1

参 数	从何处检索值
ControlParameter	页面中的控件，如 DropDownList 或 TextBox
CookieParameter	存储在用户计算机上并随每条请求发送到服务器的 cookie
FormParameter	已提交给服务器的表单上的值
Parameter	多种源。通过这一参数，通常可通过代码设置其值
ProfileParameter	用户配置文件上的属性。第 17 章将详细介绍 ASP.NET 配置文件
QueryStringParameter	查询字符串字段
SessionParameter	存储在会话(这是在用户访问站点时为特定用户存储的数据)中的值

由于这些参数的行为或多或少有些类似，因此可在自己的代码中轻松地使用它们。一旦理解了如何使用它们中的一个，就可以快速使用其他参数。在下一“试一试”练习(使用带有所有流派的 DropDownList 筛选属于所选流派的评论列表)中，将介绍如何使用 ControlParameter。

试一试

建立筛选器

为了使长的数据列表易于管理，以更小的字节块的方式向用户提供它们会是个不错的主意。例如，需要用列表表示数据库中的评论，好的做法是让用户按流派进行筛选。将带有流派的 DropDownList 置于带有评论的 GridView 上，来将评论限制为属于选择的流派就是不错的方案。接下来将看到如何进行构建。

- (1) 在 Management 文件夹中创建一个名为 Reviews.aspx 的新 Web 窗体，确保它使用 Code Behind，并且基于新的 Management 母版页。将该页面的 Title 改为 Planet Wrox-Management -Reviews。
- (2) 在母版页的 Management 部分中添加一个到该页面的链接。

```
<li><a href="~/Management/Genres.aspx" runat="server">Manage Genres</a></li>
<li><a href="~/Management/Reviews.aspx" runat="server">Manage Reviews</a></li>
</ul>
```

- (3) 回到 Reviews.aspx 页面并切换至 Design 视图。从 Toolbox 的 Standard 类别中，拖动一个 DropDownList 控件到该页面。在其 Smart Tasks 面板上，选择 Enable AutoPostBack，然后单击 Edit Items 链接。插入一个项并将其 Text 设置为 Make a selection，然后清除其自动插入的 Value。
- (4) 从 ListItem Collection Editor 对话框返回后，下拉列表的 Smart Tasks 面板仍打开着。单击 Choose Data Source 项并从屏幕顶部的下拉列表中选择<New data source>。将出现如图 13-7 所示的

Data Source Configuration Wizard。

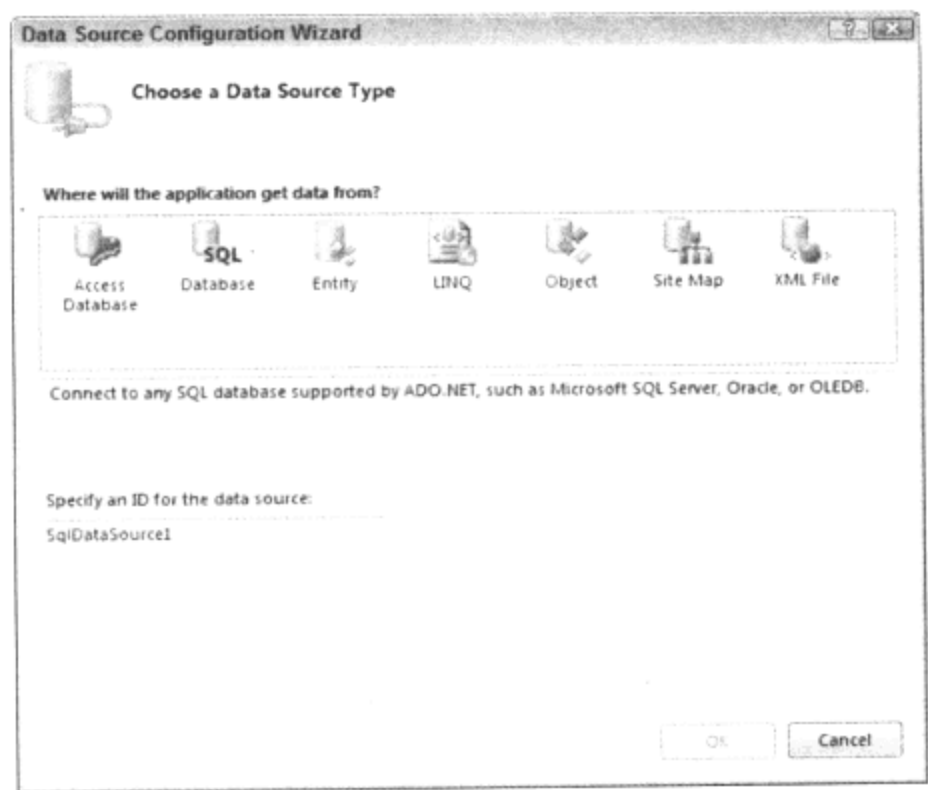


图 13-7

(5) 单击 Database 项，使其 ID 设置为 SqlDataSource1，并单击 OK 按钮。

(6) 在随后的对话框中，从下拉列表中选择名为 PlanetWroxConnectionString1 的连接字符串并单击 Next 按钮。

(7) 选中 Specify columns from a table or view 单选按钮。同时要确保从表名下拉列表中选择 Genre，然后选择 Columns 部分中的 Id 和 Name 列。单击 ORDER BY 按钮，从 Sort By 下拉列表中选择 SortOrder 并单击 OK 按钮。完成设置后，Configure Data Source 向导如图 13-8 所示。

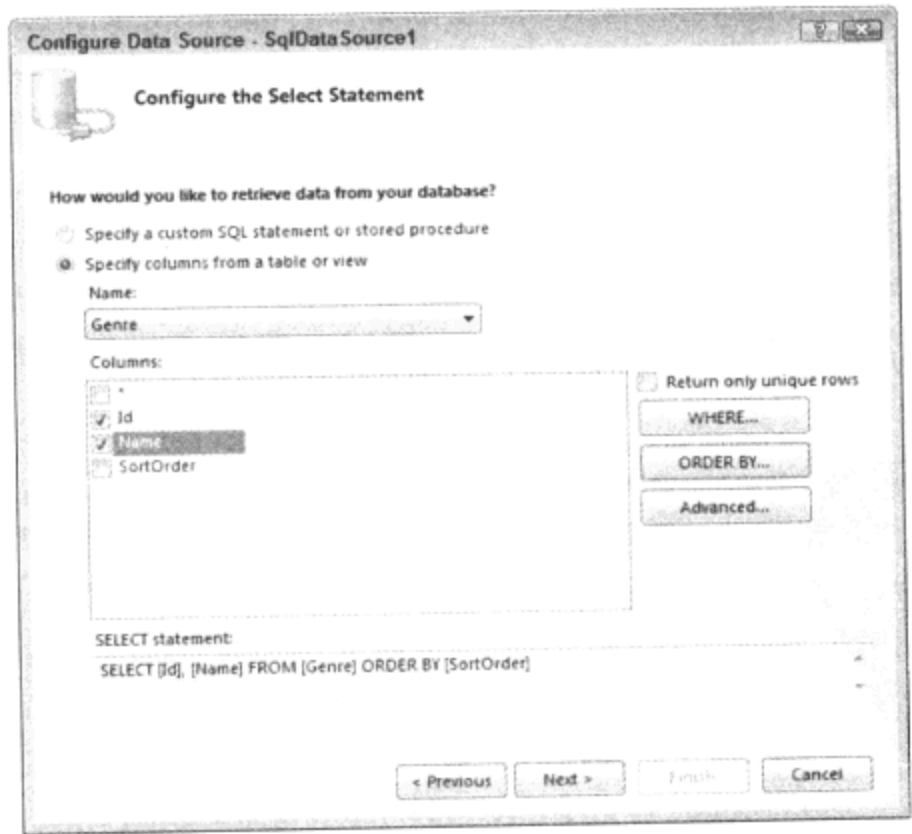


图 13-8

(8) 单击 Next 按钮，然后是 Finish 按钮，让 VWD 创建正确的 SqlDataSource。返回到下拉列表的 Data Source Configuration Wizard，在其中可以为流派建立一个显示在下拉列表中的字段和一个作为列表中底层值的字段。选择 Name 作为要显示的字段，并使第二个下拉列表设置为 Id。结果如图 13-9 所示。

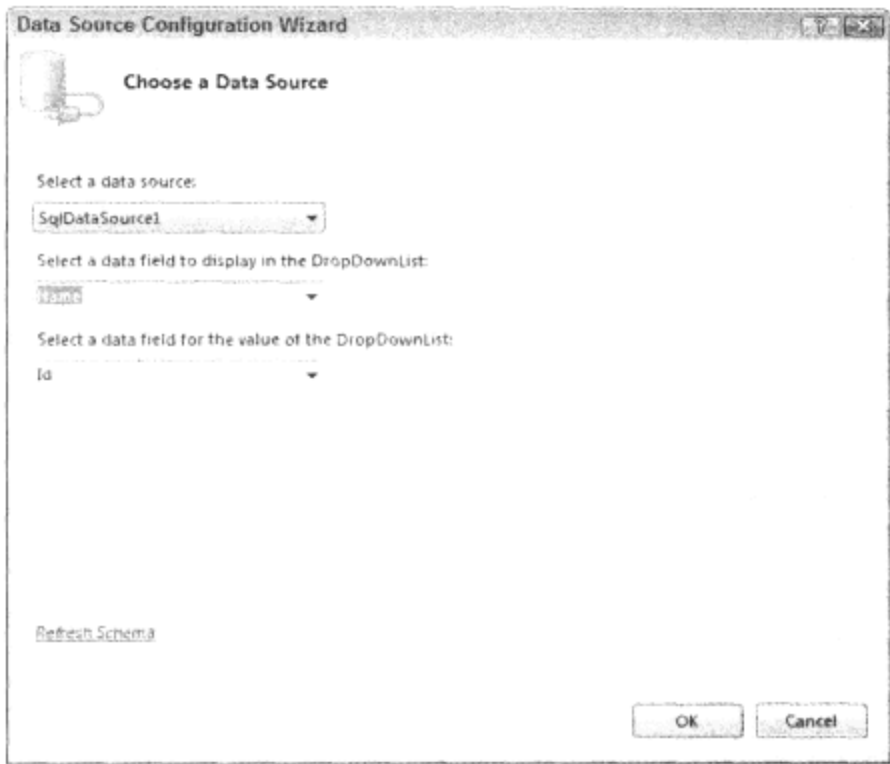


图 13-9

(9) 单击 OK 按钮关闭该对话框完成下拉列表的数据源的建立过程。

(10) 在 Design 视图中仍选择 DropDownList 控件，按 F4 键打开其 Properties 面板，设置属性 AppendDataBoundItems 为 True。切换至 Markup 视图，如果指示用户选择列表项的静态 ListItem 并没有 Value 特性，就手动添加并将其设置为一个空字符串。最终代码如下所示：

```
<asp:DropDownList ID="DropDownList1" runat="server" DataSourceID="SqlDataSource1"
    DataTextField="Name" DataValueField="Id" AppendDataBoundItems="true"
    AutoPostBack="True">
    <asp:ListItem Value="">Make a selection</asp:ListItem>
</asp:DropDownList>

<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%= $ConnectionString:PlanetWroxConnectionString1 %>"
    SelectCommand="SELECT [Id], [Name] FROM [Genre] ORDER BY [SortOrder]">
</asp:SqlDataSource>
```

(11) 保存所有更改，然后按 Ctrl+F5 组合键在浏览器中打开该页面。将看到一个带有数据库中所有流派的下拉列表，并且这些流派按照它们的 SortOrder 列进行排序。一旦从列表中选择一个新的流派，页面就回发给服务器。不会发生其他事情，因为没有给 DropDownList 控件连接任何逻辑，不过下一“试一试”练习中会作介绍。

工作原理

在本练习的最后，所得到的代码和之前练习中将 GridView 拖至页面上时 VWD 自动创建的代码

类似。有一个数据绑定控件(DropDownList)，它从数据源控件(SqlDataSource 控件)获取数据。所不同的是建立的方式在生成的代码方面提供了更大的灵活性。不是依赖于 VWD 为数据库的所有列生成一个 SQL 语句，而是只选择需要的两列。另外，由于 SqlDataSource 并不要求对数据源作更新，所以只需要提供一个 SelectCommand。也可以使用 ORDER BY 按钮控制添加到列表的项的顺序。

在建立了 SqlDataSource 控件之后，将其返回的数据显示在 DropDownList 控件中就相当简单了。首先使用 DataSourceID 特性将 DropDownList 指向正确的数据源，然后建立 DataTextField 和 DataValueField 来告诉控件对于控件中显示的文本和底层值要使用什么列。将 AppendDataBoundItems 设置为 True，可以保留在代码中手动添加的项。如果禁用这一设置，则一旦添加数据绑定项，静态文本 Make a selection 就会被清除。

在建立了筛选器后，下一步是创建 GridView，显示所选流派的评论。在下一个“试一试”练习中将作详细介绍。

试一试

应用筛选器

在这个“试一试”练习中，添加另一个从 Review 表中获取其数据的 SqlDataSource。通过创建筛选器(SQL 语句中的 WHERE 子句)，可以限制显示在网格中的项为只属于特定流派的那些评论。使用<asp:ControlParameter>将从上一节中创建的下拉列表中选择的流派发送到 SqlDataSource 控件的 SelectParameters 集合中。

- (1) 切换 Reviews.aspx 页面至 Design 视图，从 Toolbox 的 Data 类别中拖动一个 GridView 到已有 SqlDataSource 控件的上方。GridView 会添加在其上方并打开 Smart Tasks 面板。
- (2) 在 Choose Data Source 下拉列表中，选择<New data source>。在 Data Source Configuration Wizard 中，单击 Database(就像图 13-7 所示的在向导中选择流派一样)并单击 OK 按钮。
- (3) 在随后的对话框中，从下拉列表中选择 Planet Wrox 连接字符串并再次单击 Next 按钮。
- (4) 选择 Name 下拉列表中的 Review 表，然后选中 Columns 列表中的星号(*)选择所有列。
- (5) 单击 WHERE 按钮建立一个使用 SelectParameters 的 WHERE 子句。在随后的对话框中，输入如图 13-10 所示的详细信息。

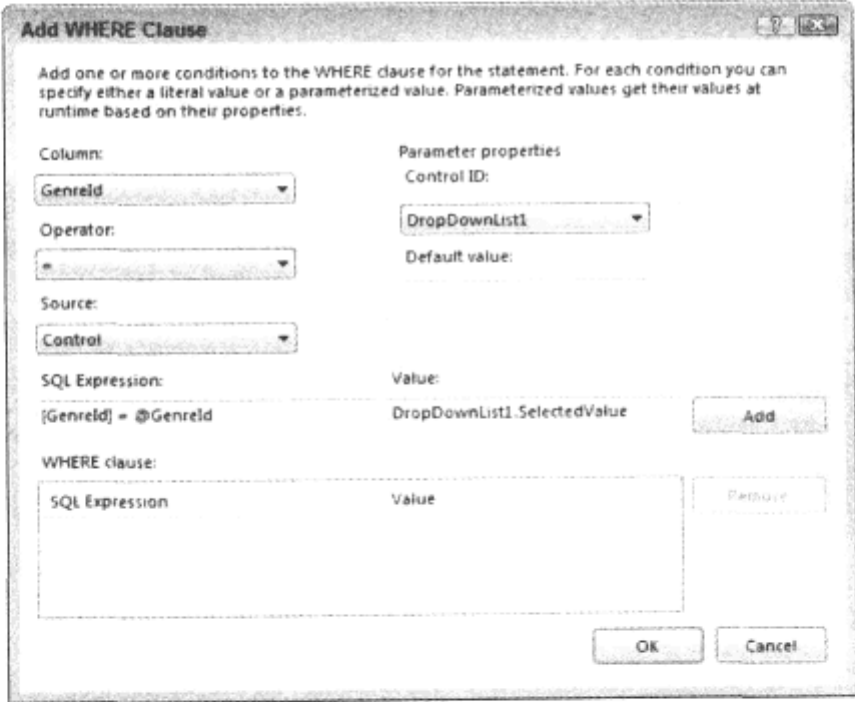


图 13-10

因为某种原因，您的每个控件可能在 Control ID 下拉列表中显示两遍。选择哪个 DropDownList1 选项都可以。

(6) 单击 Add 按钮给位于屏幕底部的 WHERE 子句列表添加选择，然后单击 OK 按钮。

(7) 回到 Configure Data Source 向导中，单击 Next 按钮。要测试查询，可单击 Test Query 按钮。如果参数正确的话，会有对话框弹出要求输入一个值。输入一个在 Genre 表中存在的值并单击 OK 按钮。如果在 Review 表中包含所选择的流派的记录，它们就会显示在 Test Query 窗口中。最后，单击 Finish 按钮关闭向导。如果获得一个有关刷新参数的对话框，那么单击 Yes 按钮更新 Markup 视图中的代码。

(8) 保存所有更改并在浏览器中打开 Reviews.aspx 页面。



常见错误：如果出现错误“input string was not in a correct format”，则要确保将下拉列表中静态 ListItem 的 Value 设置为空字符串(“”)。

(9) 从下拉列表中选择新项。页面就会刷新，并显示属于所选流派的评论。如果页面不刷新，请确保在前一个“试一试”练习中将 AutoPostBack 设置为 True。现在，页面看起来相当乱，因为 GridView 包含许多列，其中一些非常宽(如 Body 列)。在下一个“试一试”练习中将看到如何对此进行调整。

工作原理

本练习的大部分内容与之前显示可用流派列表的练习相同。不同之处在于 SqlDataSource 基于下拉列表中的选择筛选 Review 表的记录的方式。看一下 SqlDataSource 这一部分的代码：

```
<asp:SqlDataSource ID="SqlDataSource2" runat="server"
    ConnectionString="<%= $ ConnectionStrings:PlanetWroxConnectionString1 %>"
    SelectCommand="SELECT * FROM [Review] WHERE ([GenreId] = @GenreId)">
    <SelectParameters>
        <asp:ControlParameter ControlID="DropDownList1" Name="GenreId"
            PropertyName="SelectedValue" Type="Int32" />
    </SelectParameters>
</asp:SqlDataSource>
```

SelectCommand 的 SQL 语句包含了 GenreID 的参数，该参数由 SELECT 语句中的 @GenreId 变量表示。这意味着该 SQL 语句只返回 Review 表中特定流派的记录。在运行时，运行库从 ControlParameter 元素中定义的控件中检索这一参数值。在这个示例中，代码是从 DropDownList1 控件中获取值。VWD 知道，为了从 DropDownList 中获取选择的值，应访问其 SelectedValue 属性，因此将其添加为 ControlParameter 的 PropertyName。如果需要使用不同的属性，可以简单地在 ControlParameter 元素的声明中作修改。

在这一代码建立后，GridView 向 SqlDataSource 请求数据。然后这一数据源向 DropDownList 请求用户在列表中的选择的项。该值插入到 SQL 语句中，并随后者发送到数据库。从数据库返回的结果通过数据源再发送回 GridView，GridView 使用它们在浏览器中创建 HTML 表。

从下拉列表中选择“Make a Selection”时，会得到一个没有记录的空页面。在这里，DropDownList

返回一个空字符串作为其值(在 Value 属性中定义),而在数据库中被转换为 null。这就导致了查询从 Review 表中不返回记录。

到现在为止,都是依靠 VWD 的代码生成工具创建 GridView 和 DetailsView。默认情况下,VWD 为它在数据源中发现的每个列创建一个列(针对 GridView)或一个字段(针对 DetailsView)。它可以智能地识别数据源中一些底层数据类型,因此对于数据库中的布尔(位)字段,可以使用 CheckBoxField,但仅此而已。为了进一步自定义数据控件的外观,需要自定义其 Columns 和 Fields 集合。

13.3 自定义数据控件的外观

默认情况下,GridView 和 DetailsView 自动根据接收的数据呈现列或行。或者,在将控件与数据源相连时,可以让 VWD 创建大量字段或列。但通常你想改变屏幕上显示的内容,使用更少的列、不同的列标题或是不同的控件来显示数据。幸运的是,这可以通过使用 VWD 中的 Fields 编辑器轻松实现。在下一节中,将介绍如何使用这一编辑器创建和修改不同类型的内置列和字段。在这之后的一节中,将介绍如何使用用户定义的模板进一步自定义字段。

配置数据绑定控件的列或字段

在 GridView 和 DetailsView 的<Columns>或<Fields>元素中,可以添加如表 13-2 所示的字段类型。

表 13-2

字段类型	说明
BoundField	这是大部分数据库类型的默认字段。它呈现为只读模式下的简单文本和编辑模式下的 TextBox
ButtonField	这一字段类型呈现为链接或按钮,允许在服务器上执行一条命令
CheckBoxField	这一字段类型呈现为只读模式下的只读复选框,和编辑模式下的可编辑复选框
CommandField	这一字段类型支持建立不同的命令,包括编辑、插入、更新和删除
HyperLinkField	这一字段类型呈现为链接(<a>元素)。可以设置 DataNavigateUrlFields、DataNavigateUrlFormatString 和 DataTextField 等属性来影响超链接的行为。下一个“试一试”练习中将看到有关此字段的更多内容
ImageField	这一字段类型在浏览器中呈现为元素
TemplateField	这一字段类型允许为不同的模板自定义外观,如 ItemTemplate、InsertItemTemplate 和 EditItemTemplate

显然,每种 Field 类型都有着不同的用途,因此可从中选择最合适的。在下一个“试一试”练习中将看到其中一些 Field 类型的更多内容。

试一试

自定义 GridView 列

在这个练习中,将介绍如何在 Reviews.aspx 页面中进行下列操作:

- 使用 Fields 编辑器为带有评论的 GridView 自定义字段;
- 使用 HyperLink 列创建到详细页面的链接,在详细页面上可管理评论的细节内容;
- 格式化已有 BoundField 列的输出;
- 在 Code Behind 文件中使用自定义函数完全控制 TemplateField 中的输出。

后一个练习将介绍如何创建详细页面来插入新的评论和编辑已有评论。

(1) 在 Reviews.aspx 中，进入 Design 视图，为 SqlDataSource2 控件打开 Smart Tasks 面板并单击 Configure Data Source。单击 Next 按钮跳过连接字符串屏幕，然后通过从 Review 表中选择 Id、Title、Authorized 和 CreateDateTime 列完成如图 13-11 所示的屏幕设置。确保 SQL 语句文本框也包含前面创建的 WHERE 子句筛选器。



图 13-11

单击 Advanced 按钮，取消选择第一项，让 VWD 生成 INSERT、UPDATE 和 DELETE 语句的命令。可以取消选中 Optimistic Concurrency 复选框——它检测自记录最后从数据源加载以来的变化。单击 OK 按钮关闭 Advanced SQL Generation Options 对话框，然后单击 Next 按钮，最后单击 Finish 按钮，更新该页面源代码中的 SQL 语句。在被询问是否想为 GridView 重置字段和键时，单击 Yes 按钮。



常见错误：如果 Advanced SQL Generation Options 对话框中的两个选项都以灰色显示，则需要使用 Database Explorer 检查数据库中的表。确保按照第 12 章的介绍，将 Review 表的 Id 列设为主键。

(2) 这时，VWD 已经为 Markup 视图中的 GridView 创建了列。要删除这些项并定义自己的项，可打开 GridView 的 Smart Tasks 面板并单击 Edit Columns。这将打开 Fields 对话框。如果 Selected fields 列表已经包含了项，则首先使用 Delete 按钮(其上有大大的红色 X 标记)清除列表。

(3) 在 Available fields 列表中，选择 BoundField(而不是 CheckBoxField)下的 Authorized，然后单击 Add 按钮将列表项复制到 Selected fields 列表中。对 CreateDateTime 字段重复这一步骤。现在的对话框如图 13-12 所示。

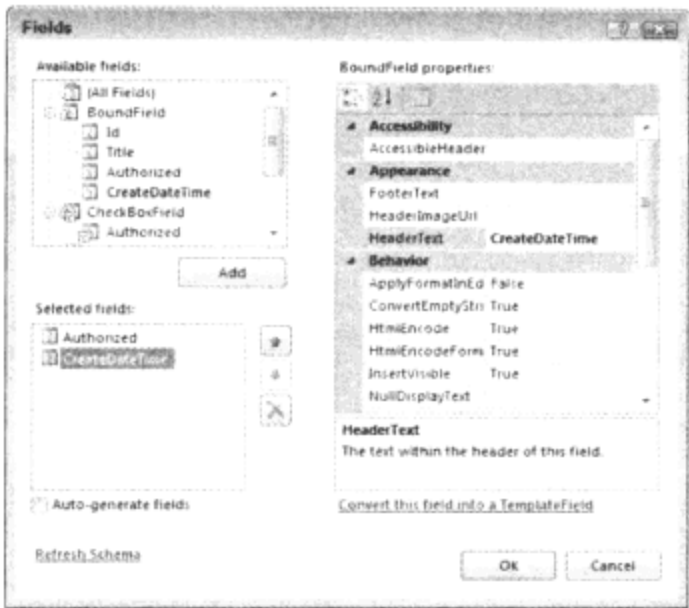


图 13-12

(4) 在屏幕顶部的 Available fields 列表中，选择 HyperLinkField，然后单击 Add 按钮将该项也添加到 Selected fields 列表中。通过单击上箭头两次，将 HyperLinkField 移至列表最上面。然后使用右侧的 Properties 面板，为 HyperLinkField 设置如表 13-3 所示的属性。

表 13-3

属 性	值
HeaderText	Title
DataNavigateUrlFields	Id
DataNavigateUrlFormatString	AddEditReview.aspx?Id={0}
DataTextField	Title

(5) 在 Available fields 列表中，单击 CommandField 并再次单击 Add 按钮。然后使用 Properties 面板将刚插入的列表项的 HeaderText 属性设置为 Delete，将 ShowDeleteButton 设置为 True。这使得在后面就可以使用 GridView 从数据库中删除评论。这时的 Fields 对话框如图 13-13 所示。

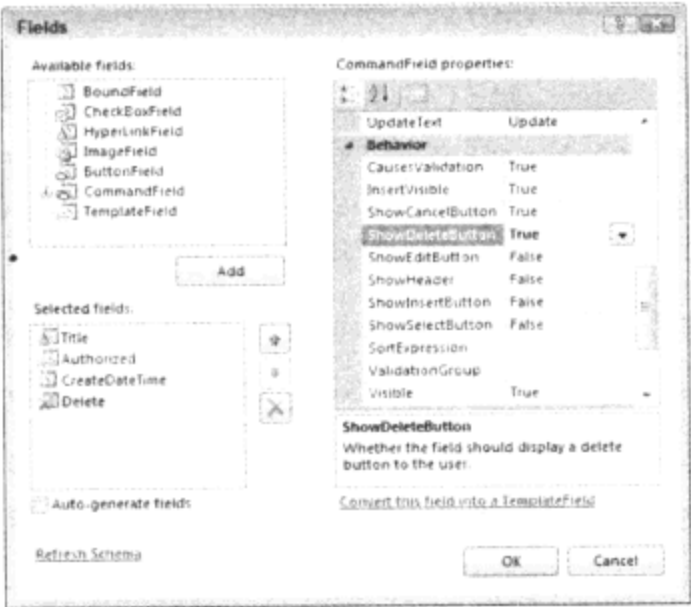


图 13-13

- (6) 单击 Selected fields 列表中的 Authorized 列，然后单击对话框右下角蓝色的 Convert this field into a TemplateField 链接，如图 13-13 所示。
- (7) 单击左侧的 CreateDateTime 列，设置其 DataFormatString 属性为 {0:g}。
- (8) 单击 OK 按钮将更改应用于源代码。
- (9) 切换到 Markup 视图，然后删除 Authorized 字段的 <EditItemTemplate>。GridView 会以只读模式显示评论，因此不需要此模板。
- (10) 修改 Authorized 字段的 ItemTemplate 中的 Label 控件，结果如下所示：

```
<asp:Label ID="AuthorizedLabel" runat="server"
    Text='<%# GetBooleanText(Eval("Authorized")) %>' />
```

- (11) 按下 F7 键切换至页面的 Code Behind 文件，向类文件的顶部(VB.NET 中是在 Inherits 行的后面，C#中是在起始花括号之后)添加下列函数，它将根据传递的布尔值返回文本 Yes 或 No。

```
VB.NET
Inherits System.Web.UI.Page
Protected Function GetBooleanText(ByVal booleanValue As Object) As String
    Dim authorized As Boolean = CType(booleanValue, Boolean)
    If authorized Then
        Return "Yes"
    Else
        Return "No"
    End If
End Function
```

```
C#
public partial class Management_Reviews : System.Web.UI.Page
{
    protected string GetBooleanText(object booleanValue)
    {
        bool authorized = (bool)booleanValue;
        if (authorized)
        {
            return "Yes";
        }
        else
        {
            return "No";
        }
    }
}
```

- (12) 保存所有更改(按 Ctrl+Shift+S 组合键)，然后按 Ctrl+F5 组合键在浏览器中打开 Reviews.aspx 页面。从下拉列表中选择一个流派，将看到一个评论列表出现。注意，现在，Authorized 列显示了文本 Yes 或 No。CreateDateTime 列以短格式显示日期和时间。图 13-14 显示了 Indie Rock 流派的结果。如果没有正确地看到时间，那么切换回 VWD，为 CreateDateTime 的 BoundField 添加一个 HtmlEncodeFormatString 属性，并将其设置为 True：

```
<asp:BoundField DataField="CreateDateTime" DataFormatString="{0:g}"
    HeaderText="CreateDateTime" HtmlEncodeFormatString="True"
    SortExpression="CreateDateTime" />
```

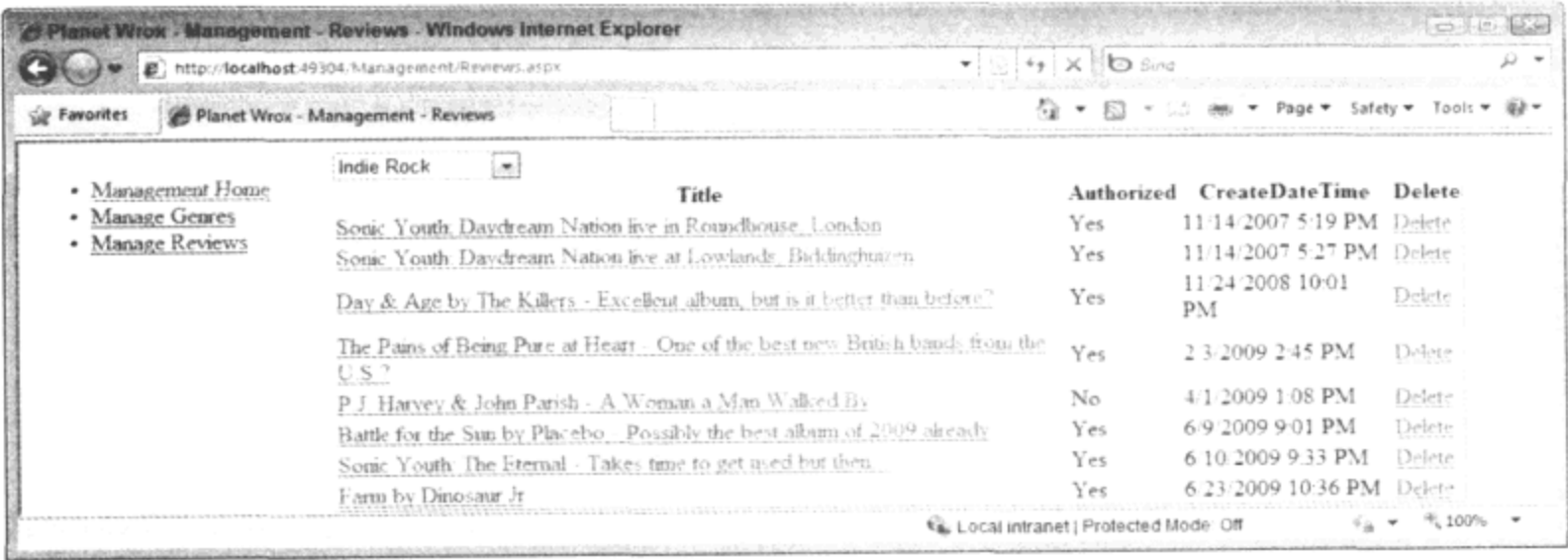


图 13-14

注意，GridView 的第一列中的标题现在链接至一个页面，其中的评论的 ID 传递到查询字符串字段 Id: <http://localhost:1049/Management/AddEditReview.aspx?Id=1>。本章后面将创建这个 Add/Edit 页面。

工作原理

首先为 SqlDataSource 修改 SelectCommand。现在 SQL 语句不是使用 SELECT * 选择所有列，而是包含列的子集，使页面加载稍快一些：

```
<asp:SqlDataSource ID="SqlDataSource2" runat="server"
    ConnectionString="<%%$ ConnectionStrings:PlanetWroxConnectionString1 %>"
    SelectCommand="SELECT [Id], [Title], [Authorized], [CreateDateTime]
    FROM [Review] WHERE ([GenreId] = @GenreId)">
    ...
</asp:SqlDataSource>
```

接着，使用 Fields 对话框修改 GridView 显示的不同字段。使用 HyperLinkField 创建 Title 列：

```
<asp:HyperLinkField DataNavigateUrlFields="Id" DataTextField="Title"
    DataNavigateUrlFormatString="AddEditReview.aspx?Id={0}" HeaderText="Title">
</asp:HyperLinkField>
```

DataNavigateUrlFields 包含了一个您想在 DataNavigateUrlFormatString 属性中使用的字段的用逗号隔开的列表。在这里，只使用了一个字段。为了显示这一字段的值，在 DataNavigateUrlFormatString 属性使用了占位符，如 {0}。例如，ID 为 10 的评论，其 HyperLink 列的 NavigateUrl 属性为 AddEditReview.aspx? Id=10。通过这一设置，{0} 被 DataNavigateUrlFields 属性中第一个字段的值替代了。如果定义用逗号隔开的多个字段，就用 {1}、{2} 等访问它们。

DataTextField 设置为列 Title。这使得 HyperLink 用评论的标题呈现其 Text 特性，如图 13-14 所示。

也可以为 CreateDateTime 列的绑定字段设置 DataFormatString 属性。

```
<asp:BoundField DataField="CreateDateTime" DataFormatString="{0:g}"
    HeaderText="CreateDateTime" HtmlEncodeFormatString="True"
```



```
SortExpression="CreateTime" />
```

`DataFormatString` 允许定义底层数据显示的格式。在这里，小写字母 `g` 用于以短格式(不显示秒)显示日期和时间。在 <http://tinyurl.com/DateFormatters> 站点的 MSDN 文档中，可看到有关不同格式字符串的更多信息。

然后将 `Authorized` 列转换为模板列。在表现内容方面，模板列允许自由显示各种文本。实质上，可以添加任何合适的东西作为列内容，包括 HTML 和 ASPX 控件。在这个“试一试”练习中，您改变了 `Label`，因此它使用数据绑定表达式语法`<%# %>`从自定义函数中获取其文本。

```
<asp:Label ID="AuthorizedLabel" runat="server"
    Text='<%# GetBooleanText(Eval("Authorized")) %>'></asp:Label>
```

这起作用的原因在于两方面。首先看一下 `Eval("Authorized")`语句。这被称为单向数据绑定表达式(one-way data binding expression)，其结果是 `Authorized` 列的值作为对象传递给 `GetBooleanText` 自定义方法。然后这个方法将传入的值转换为 `Boolean` 值，并根据数据库中 `Authorized` 列的值返回 `Yes` 或 `No`。这只是个演示在数据绑定时如何在 `Code Behind` 中调用自定义方法的简单示例。不过，对于更复杂的方法而言，原理是一样的：使用 `Eval("ColumnName")`将一个或多个参数传递给 `Code Behind` 方法。`Code Behind` 中的方法接受这些对象参数，将它们强制转换为合适的类型，然后适当地使用它们。在最后，方法可以返回一个带有任何合适的文本或 HTML 的字符串。

为 `Title` 列所创建的 `HyperLink` 指向名为 `AddEditReview.aspx` 的页面。该页面允许创建新的评论和更新已有评论。在下一节中将介绍如何创建这一页面。

13.4 更新和插入数据

在本章前面，讨论了如何使用 `GridView` 和 `SqlDataSource` 控件进行简单的更新。尽管这种内置的更新行为在许多情况下表现良好，但它并不总是能满足所有需求。

幸运的是，像 `FormView` 和 `DetailsView` 这样的控件允许调整它们的外观，在终端用户处理数据方面提供了更大的灵活性。在下一节中，将介绍如何使用 `DetailsView` 控件为用户提供 一个更简单的界面，以便在数据库中插入和编辑评论。

使用 DetailsView 插入和更新数据

在本章前面，学习了如何创建一个简单的 `DetailsView` 控件，并完全依赖于 VWD 和控件本身在浏览器中呈现相关的用户界面。显然，这一默认行为并不总是能够满足要求。如果您想改变界面中使用的控件，该怎么办呢？例如，对于流派想使用 `DropDownList` 而不是简单的 `TextBox`，想添加一个或多个在第 9 章所学到的有效性验证控件，或是想通过编程的方式管理一些发送到数据库的数据时，该怎么办？通过 `DetailsView` 控件、其基于模板的列以及控件在其生命周期不同阶段触发的各种事件，所有这些都可能实现的。

不过，首先需要学习一下数据绑定控件和数据源控件触发的不同事件。表 13-4 列出了 `DetailsView`、`FormView` 和 `ListView` 提供的并在它们的生命周期中引发的一些事件。`GridView` 也有类似的事件，不过都以 `Row` 打头而非 `Item`。由于 `DataList` 和 `Repeater` 控件本身并不支持数据编辑，因此它们没有任何事件。

表 13-4

事 件	说 明
ItemInserting	该事件在对数据源执行 Insert 命令前触发。这里是更改将发送到数据库的数据的最佳位置
ItemInserted	该事件在对数据源执行 Insert 命令后触发
ItemUpdating	该事件在对数据源执行 Update 命令前触发。这里是更改将发送到数据库的数据的最佳位置
ItemUpdated	该事件在对数据源执行 Update 命令后触发
ItemDeleting	该事件在对数据源执行 Delete 命令前触发
ItemDeleted	该事件在对数据源执行 Delete 命令后触发

这 6 个事件都是在控件生命周期中非常适宜的时候触发的：对数据源执行数据操作之前和之后。在下一个“试一试”练习中将介绍如何使用它们。

试一试 使用 DetailsView 控件管理数据

在本练习中，创建 AddEditReview.aspx 页面，之前在 Reviews 页面中创建了该页面的链接。在该页面中，创建一个 DetailsView，通过实现模板字段自定义其大部分字段，然后处理该控件的一些事件来改变其行为。在完成这些之后，就可以创建、列出、更新和删除 Web 站点中的评论了。

(1) 在 Management 文件夹中，创建一个新的 Web 窗体并命名为 AddEditReview.aspx。将它基于母版页的 Management 部分，选择您的编程语言。设置其 Title 为 Planet Wrox- Management-Insert and Update Reviews。

(2) 切换该页面至 Design 视图并拖动一个 DetailsView 控件到该页面上。在自动打开的 Smart Tasks 面板中，从 Choose Data Source 下拉列表中选择<New data source>。单击 Database 图标，然后单击 OK 按钮。在随后的对话框中，从下拉列表中选择连接字符串并单击 Next 按钮。

(3) 输入如图 13-15 所示的详细信息。

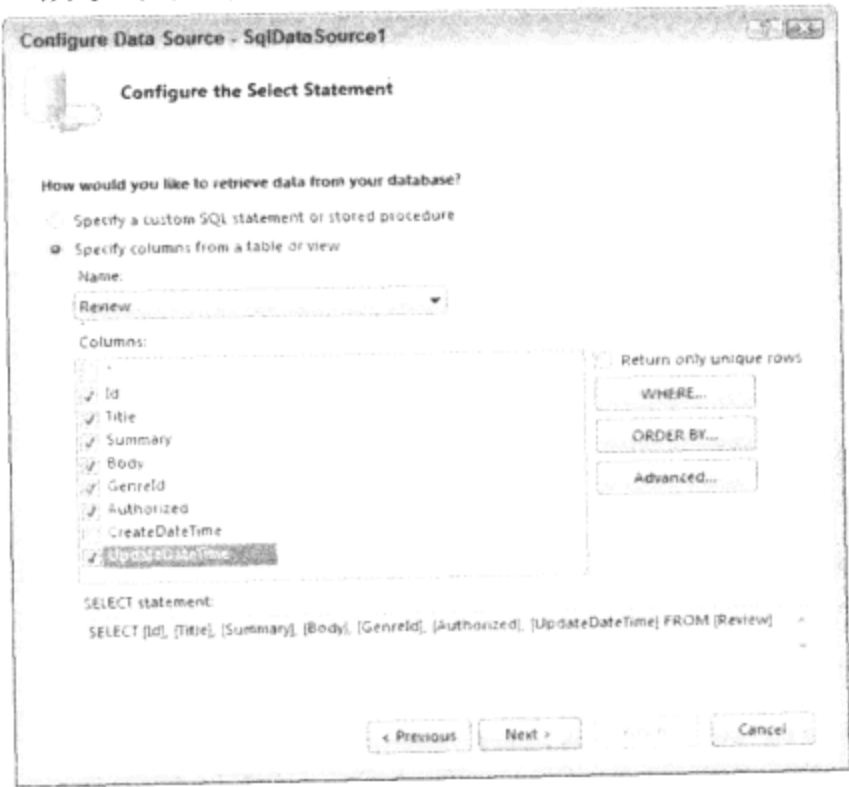


图 13-15

注意，除了 CreateDateTime 字段外，Review 的所有字段都是显式选择的。

(4) 单击 WHERE 按钮，通过完成设置如图 13-16 所示的对话框，建立一个从查询字符串中检索评论 ID 的 SelectParameter。

不要忘记在 QueryString field 文本框中输入 Id。

(5) 单击 Add 按钮将这个参数添加到对话框底部的 WHERE clause 列表中，然后单击 OK 按钮关闭该对话框。

(6) 返回到 Configure Data Source 向导(如图 13-15 所示)中，单击 Advanced 按钮，选择相应选项生成 INSERT、UPDATE 和 DELETE 语句，然后单击 OK 按钮关闭对话框。最后，单击 Next 按钮、Finish 按钮关闭数据源向导。

(7) 在 DetailsView 的 Smart Tasks 面板上(如图 13-17 所示)，选择插入和编辑的选项。

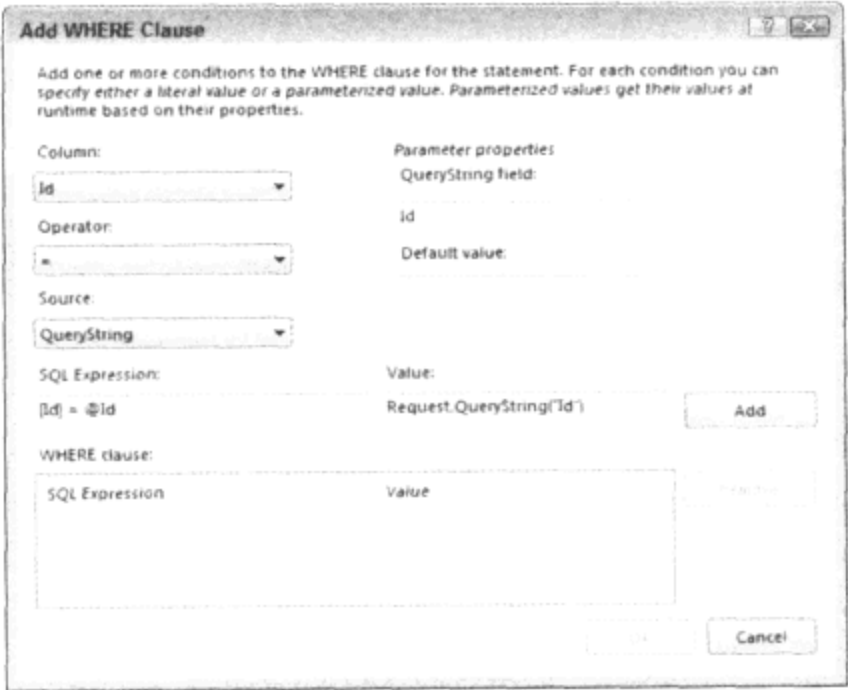


图 13-16

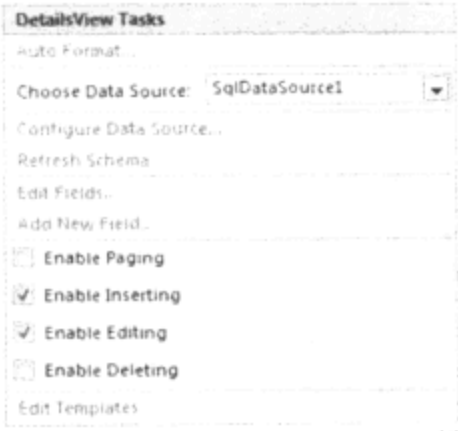


图 13-17

(8) 在 DetailsView 的 Properties 面板上，将 DefaultMode 设置为 Insert。

(9) 在 Design 视图中双击页面的空白区域，建立一个 Page_Load 处理程序并输入下列代码：

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    If Request.QueryString.Get("Id") IsNot Nothing Then
        DetailsView1.DefaultMode = DetailsViewMode.Edit
    End If
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Request.QueryString.Get("Id") != null)
    {
        DetailsView1.DefaultMode = DetailsViewMode.Edit;
    }
}
```

(10) 在 Design 视图中打开 Reviews.aspx 页面并从 Solution Explorer 中将 AddEditReview.aspx 页面拖至该页面的 GridView 之下。这就创建了到该页面的链接，因此可以插入新评论。切换至 Markup 视图，然后将<a>标记之间的文本改为 Insert New Review:

```
<a href="AddEditReview.aspx">Insert New Review</a>
```

(11) 保存所有更改，然后在浏览器中打开 AddEditReview.aspx 页面。将得到控件的默认布局，其中，数据源中的所有列都显示为简单的文本框。如图 13-18 所示填充字段。

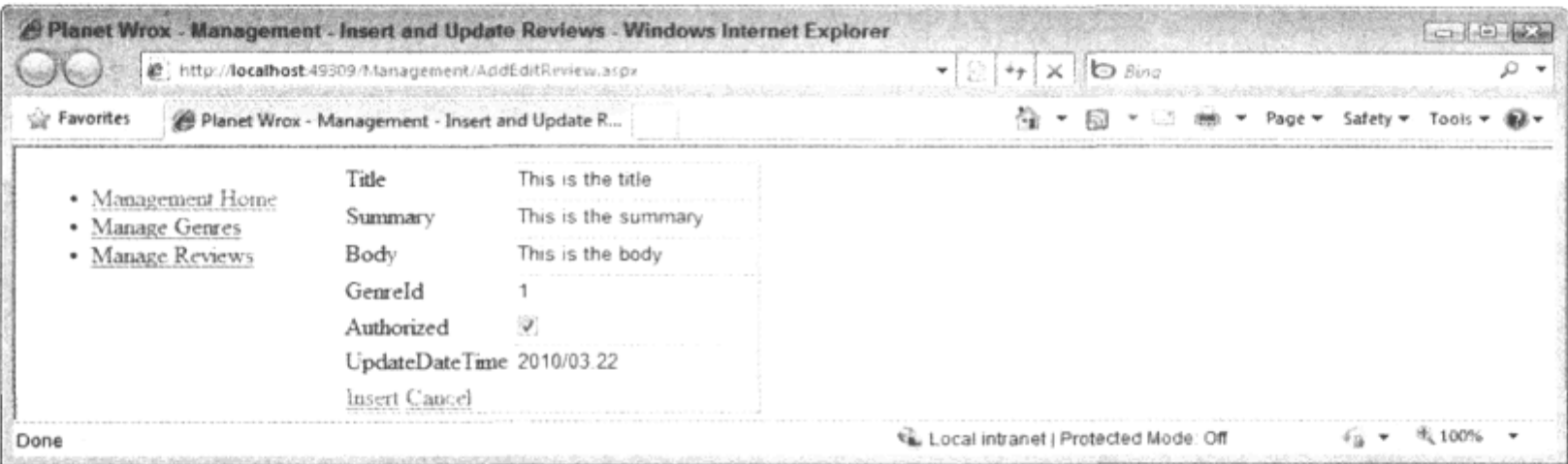



图 13-18



常见错误：如果显示一个空白屏幕，那么请确保将 DefaultMode 设为 Insert。要确保输入的 GenreId 与数据库的 Genre 表中的一个流派相匹配，否则在尝试插入项时会引发错误。同时也要确保使用 yyyy/mm/dd 的格式输入一个有效的日期，其中 y 代表年份，m 代表月份，d 代表哪一天。

单击 Insert 将记录项插入到数据库中。最初，没有太多情况发生，只是控件都被清空了。但通过下列两步，可在 Reviews.aspx 页面中定位这个新评论：

- 单击左侧菜单中的 Manage Reviews 链接。
- 从页面顶部的下拉列表中选择适当的流派。如果使用的是与本章下载一起打包的数据库，则在插入 Review 时使 GenreId 为 1，流派为 Rap and Hip-Hop。

在看到您的评论时，可单击其标题，然后您会被带到 AddEditReview.aspx 页面，在其中可再次修改该评论的详细信息。

工作原理

到现在为止，这一练习的大部分内容都是我们所熟悉的。用于插入的 DetailsView 的工作方式与前面练习中的 DetailsView 一样。所不同的是它们处理更新的方式。Code Behind 中的代码查看查询字符串，如果发现 Id 查询字符串参数，它就将 DetailsView 切换至 Edit 模式。

VB.NET

```
If Request.QueryString.Get("Id") IsNot Nothing Then
```



```
DetailsView1.DefaultMode = DetailsViewMode.Edit
End If
```

```
C#
if (Request.QueryString.Get("Id") != null)
{
    DetailsView1.DefaultMode = DetailsViewMode.Edit;
}
```

在控件处于 Edit 模式下时，它知道要做什么。它调用 `SqlDataSource` 并向其请求数据。然后 `SqlDataSource` 从查询字符串中检索评论的 ID，访问数据库，接着返回正确的评论，并显示在页面上。在随后单击 Update 链接时，`SqlDataSource` 触发其 `UpdateCommand` 将改动发送到数据库。

这一练习为后一练习奠定了很好的基础。在后一练习中，将通过实现带有有效性验证控件的自定义模板来扩展 `DetailsView`，并建立不同的事件处理程序来响应控件的事件。

现在，带有 `DetailsView` 的页面看起来有些单调。如果有了下列功能，它会变得更好、更易于使用。

- 对于 `Summary` 和 `Body` 字段采用文本区域而不是单行文本框
 - 对于流派采用下拉列表，其中填充数据库中的可用流派
 - `UpdateDateTime` 列的自动更新
 - 有效性验证控件避免字段为空
 - 在插入或更新一个记录项后自动重定向到 `Reviews.aspx` 页面
- 接下来的“试一试”练习将演示如何实现所有这些功能。

试一试

自定义 `DetailsView` 和处理其事件

这个练习有点长，包含的步骤也比较多。记住，如果您想比较自己的版本和我的版本，那么总是可以从 `Wrox` 网站上下载这个页面的最终版本。

(1) 确保 `AddEditReview.aspx` 在 Design 视图中，通过单击 `DetailsView` 控件的 Smart Tasks 面板上的 Edit Fields，打开其 Fields 编辑器。定位到 Selected fields 列表中的 `UpdateDateTime` 列，并设置其 Visible 属性为 False。

(2) 单击 Selected fields 列表中的 Title 列，然后单击带有 Convert this field into a TemplateField 文本的蓝色链接。对 `Summary`、`Body` 和 `GenreId` 字段重复这一操作，然后单击 OK 按钮关闭 Fields 对话框。

(3) 切换至 Markup 视图，然后为 `Summary` 和 `Body` 字段的 4 个文本框控件添加 `TextMode` 特性，设置其值为 `MultiLine`。另外，分别设置它们的 Width 和 Height 属性为 500 和 100 像素。确保对 `EditItemTemplate` 和 `InsertItemTemplate` 做了这一工作。最终显示 `Summary` 字段的代码如下所示：

```
<asp:TemplateField HeaderText="Summary" SortExpression="Summary">
    <ItemTemplate>
        <asp:Label ID="Label2" runat="server"
            Text='<%# Bind("Summary") %>'></asp:Label>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:TextBox ID="TextBox2" TextMode="MultiLine" Width="500" Height="100"
```

```
        runat="server" Text='<%# Bind("Summary") %>'></asp:TextBox>
    </EditItemTemplate>
    <InsertItemTemplate>
        <asp:TextBox ID="TextBox2" TextMode="MultiLine" Width="500" Height="100"
            runat="server" Text='<%# Bind("Summary") %>'></asp:TextBox>
    </InsertItemTemplate>
</asp:TemplateField>
```

(4) 在 Title 和 Summary 行的 EditItemTemplate 和 InsertItemTemplate 中添加一个 RequiredFieldValidator。可以直接输入必需代码或是直接从 Toolbox 中拖放它至 Markup 视图中。使用代码段时，这个任务更加简单一些：将光标放到 TextBox 下面的一个空白的新行中，输入 req，按下 Ctrl+空格键完成 requiredfieldvalidator 这个词，并按下 Tab 键。VWD 会插入一个 RequiredFieldValidator，并使用在代码中定义的前一个 TextBox 的 ID 自动为 ControlToValidate 属性赋值。

确保通过设置 ControlToValidate 属性并提供有用的错误消息，将验证器与模板中各自的 TextBox 控件挂钩。依次将所有 4 个 RequiredFieldValidator 控件的 ID 设置为 reqVal1、reqVal2 等，为它们提供唯一的名称。在完成设置之后，Summary 字段如下所示：

```
<asp:TemplateField HeaderText="Summary" SortExpression="Summary">
    <EditItemTemplate>
        <asp:TextBox ID="TextBox2" TextMode="MultiLine" Width="500" Height="100"
            runat="server" Text='<%# Bind("Summary") %>'></asp:TextBox>
        <asp:RequiredFieldValidator ID="reqVal3" ControlToValidate="TextBox2"
            runat="server" ErrorMessage="Enter a summary">
        </asp:RequiredFieldValidator>
    </EditItemTemplate>
    <InsertItemTemplate>
        <asp:TextBox ID="TextBox2" TextMode="MultiLine" Width="500" Height="100"
            runat="server" Text='<%# Bind("Summary") %>'></asp:TextBox>
        <asp:RequiredFieldValidator ID="reqVal4" ControlToValidate="TextBox2"
            runat="server" ErrorMessage="Enter a summary">
        </asp:RequiredFieldValidator>
    </InsertItemTemplate>
    <ItemTemplate>
        <asp:Label ID="Label2" runat="server" Text='<%# Bind("Summary") %>'>
        </asp:Label>
    </ItemTemplate>
</asp:templatefield>
```

Title 和 Body 字段看起来与此类似。Title 字段的 TextBox 没有应用 TextMode、Width 和 Height 属性，而 Body 字段没有 RequiredFieldValidator 属性。除此之外，这些字段都和 Summary 字段非常相似。

(5) 切换至 Design 视图，拖一个新的 SqlDataSource 控件到页面上已有的 SqlDataSource1 的旁边。打开其 Smart Tasks 面板并单击 Configure Data Source。从下拉列表中选择 Planet Wrox 连接字符串并单击 Next 按钮。选择 Genre 表中的 Id 和 Name 列，单击 ORDER BY 按钮并从 Sort By 下拉列表中选择 SortOrder，在 SortOrder 列上建立 ORDER BY 子句。单击 OK 按钮，Configure Data Source 屏幕如图 13-19 所示。



图 13-19

- (6) 依次单击 Next 按钮和 Finish 按钮，关闭 Configure Data Source 向导。
- (7) 在 Design 视图选择新的 SqlDataSource(名为 SqlDataSource2)，使用 Properties 面板将其 ID 改为 GenresDataSource，以便于在页面中识别它。
- (8) 切换至 Markup 视图，定位至 DetailsView 的 GenreId 的 InsertItemTemplate，删除其所有内容(TextBox 控件)。在删除 TextBox 的地方，通过从 Toolbox 拖放至 Markup 视图，添加一个 DropDownList 控件。代码如下所示：

```
<asp:TemplateField HeaderText="GenreId" SortExpression="GenreId">
  <EditItemTemplate>
    <asp:TextBox ID="TextBox4" runat="server"
      Text='<%# Bind("GenreId") %>'></asp:TextBox>
  </EditItemTemplate>
  <InsertItemTemplate>
    <asp:DropDownList ID="DropDownList1" runat="server">
    </asp:DropDownList>
  </InsertItemTemplate>
  ...
</asp:TemplateField>
```

- (9) 切换至 Design 视图，右击 DetailsView 并选择 Edit Template | Field[4]-GenreId 命令，如图 13-20 所示。如果没有看到这个菜单命令，那么需要在带有控件的一个行上(如 summary 行)单击，使 DetailsView 正确地获得焦点。

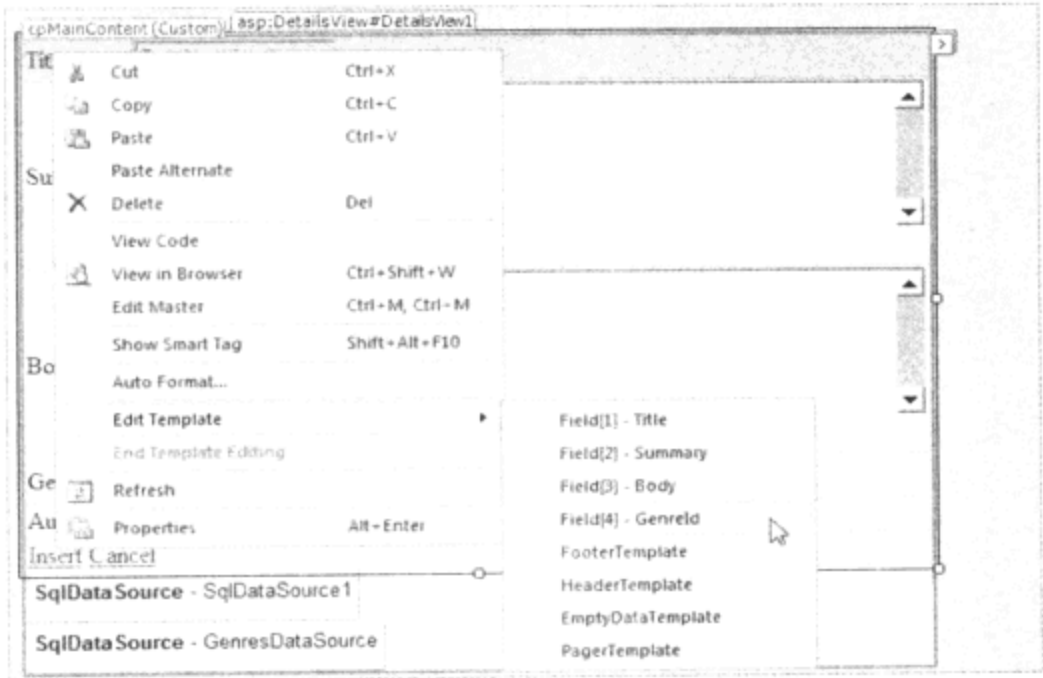


图 13-20

(10) 如果控件处于模板编辑模式下，可以直接访问该 DropDownList。向下滚动一点找到 DropDownList，打开其 Smart Tasks 面板并选择 Choose Data Source。在 Data Source Configuration Wizard 中，从数据源下拉列表中选择 GenresDataSource，从另外两个下拉列表中分别选择 Name 和 Id，如图 13-21 所示。如果 Name 和 Id 没有显示在下拉列表中，则应该单击屏幕底部蓝色的 Refresh Schema 链接。如果列表中没有列出 Data Source，但是列出了 SqlDataSource2，那么要确保按照第(7)步的描述正确地重命名了控件。

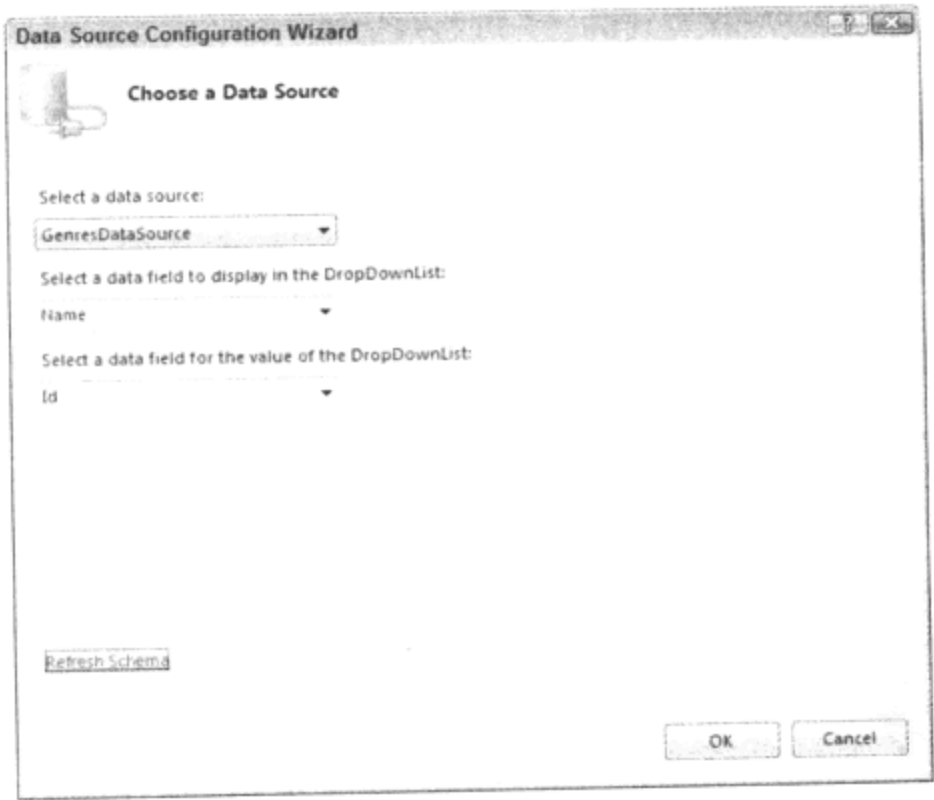


图 13-21

(11) 单击 OK 按钮关闭 Data Source Configuration Wizard。

(12) 回到 DropDownList 控件的 Smart Tasks 面板，单击 Edit DataBindings。在打开的对话框中，单击左侧列表中的 SelectedValue，然后从右侧 Bound to 下拉列表中选择 GenreId，如图 13-22 所示。如果发现屏幕右上角的 Field binding 单选按钮是只读的，则需要单击 Refresh Schema 链接。在参数列表的 Value 字段中输入一个 ID，比如 1，然后单击 OK 按钮。当返回到 DataBindings 对话框时，该单选按钮应该已经启用。

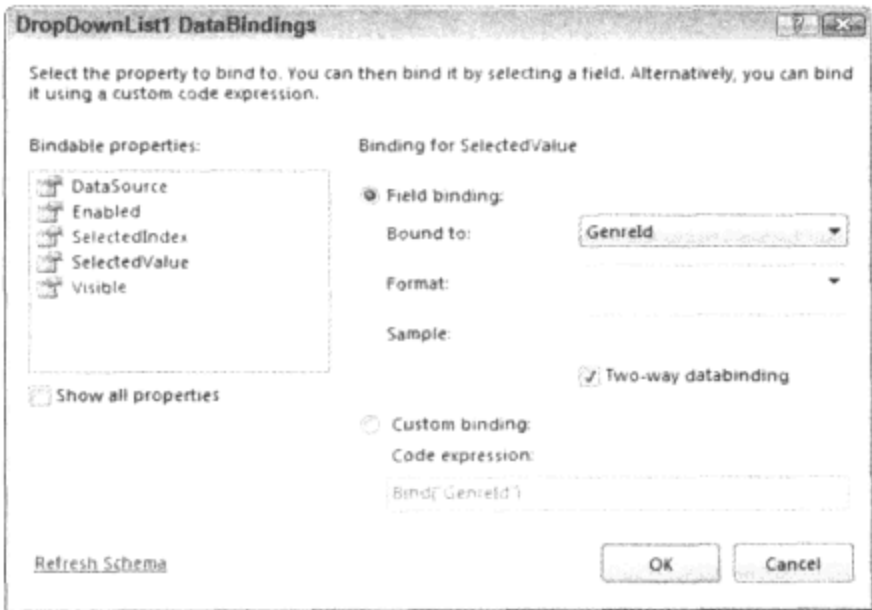


图 13-22

(13) 单击 OK 按钮关闭对话框。现在 Markup 视图中 InsertItemTemplate 的代码如下所示：

```
<InsertItemTemplate>
  <asp:DropDownList ID="DropDownList1" runat="server"
    DataSourceID="GenresDataSource" DataTextField="Name" DataValueField="Id"
    SelectedValue='<%# Bind("GenreId") %>'
  </asp:DropDownList>
</InsertItemTemplate>
```

(14) 复制 InsertItemTemplate 的内容(上一步中突出显示的代码)并粘贴到 EditItemTemplate 中，重写已有的 TextBox 控件。这就为 Edit 模式下的 DetailsView 添加了同样的下拉列表。

(15) 切换至 Design 视图，单击 DetailsView，然后按 F4 键打开 Properties 面板。切换到 Properties 面板的 Events 选项卡，双击下列事件。每次双击一个事件时，VWD 都会切换至页面的 Code Behind，因此需要切换回页面(使用 Ctrl+Tab 组合键)再添加其他事件。

- ItemInserted
- ItemInserting
- ItemUpdated
- ItemUpdating

在完成之后，Properties 面板的事件类别如图 13-23 所示。

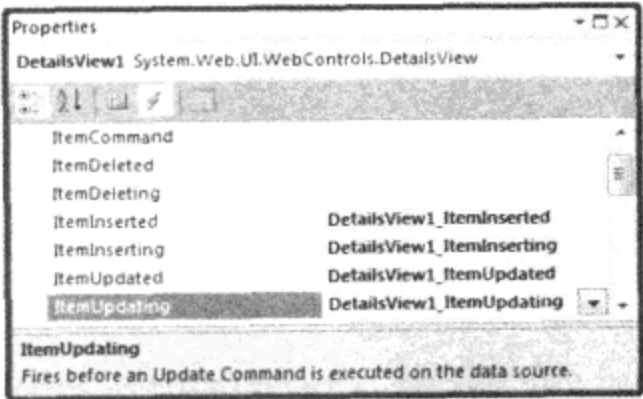


图 13-23

(16) 在 Code Behind 中按如下所示修改代码。注意，ItemInserted 和 ItemUpdated 处理程序调用 EndEditing 方法(也需要添加到代码中)，而 ItemInserting 和 ItemUpdating 设置 UpdateDateTime 的值：

VB.NET

```
Protected Sub DetailsView1_ItemInserted(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.DetailsViewInsertedEventArgs) _
    Handles DetailsView1.ItemInserted
    EndEditing()
End Sub

Protected Sub DetailsView1_ItemInserting(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.DetailsViewInsertEventArgs) _
    Handles DetailsView1.ItemInserting
    e.Values("UpdateDateTime") = DateTime.Now
End Sub

Protected Sub DetailsView1_ItemUpdated(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.DetailsViewUpdatedEventArgs) _
    Handles DetailsView1.ItemUpdated
    EndEditing()
End Sub

Protected Sub DetailsView1_ItemUpdating(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.DetailsViewUpdateEventArgs) _
    Handles DetailsView1.ItemUpdating
    e.NewValues("UpdateDateTime") = DateTime.Now
End Sub

Private Sub EndEditing()
    Response.Redirect("Reviews.aspx")
End Sub
```

C#

```
protected void DetailsView1_ItemInserted(object sender,
    DetailsViewInsertedEventArgs e)
{
    EndEditing();
}

protected void DetailsView1_ItemInserting(object sender,
    DetailsViewInsertEventArgs e)
{
    e.Values["UpdateDateTime"] = DateTime.Now;
}

protected void DetailsView1_ItemUpdated(object sender,
    DetailsViewUpdatedEventArgs e)
{
    EndEditing();
}

protected void DetailsView1_ItemUpdating(object sender,
    DetailsViewUpdateEventArgs e)
{

```



```
e.NewValues["UpdateDateTime"] = DateTime.Now;
}

private void EndEditing()
{
    Response.Redirect("Reviews.aspx");
}
```

(17) 最后，保存所有更改并在浏览器中打开 AddEditReview.aspx 页面。使所有字段为空并单击 Insert 链接。注意，验证控件起作用了，它阻止向服务器发送空值。接着，填入有效值并再次单击 Insert。这次将被带到 Reviews.aspx 页面。从下拉列表中选择评论所在的流派以定位评论，然后单击其标题进行编辑。DetailsView 将显示前面输入的所有值(如图 13-24 所示)。

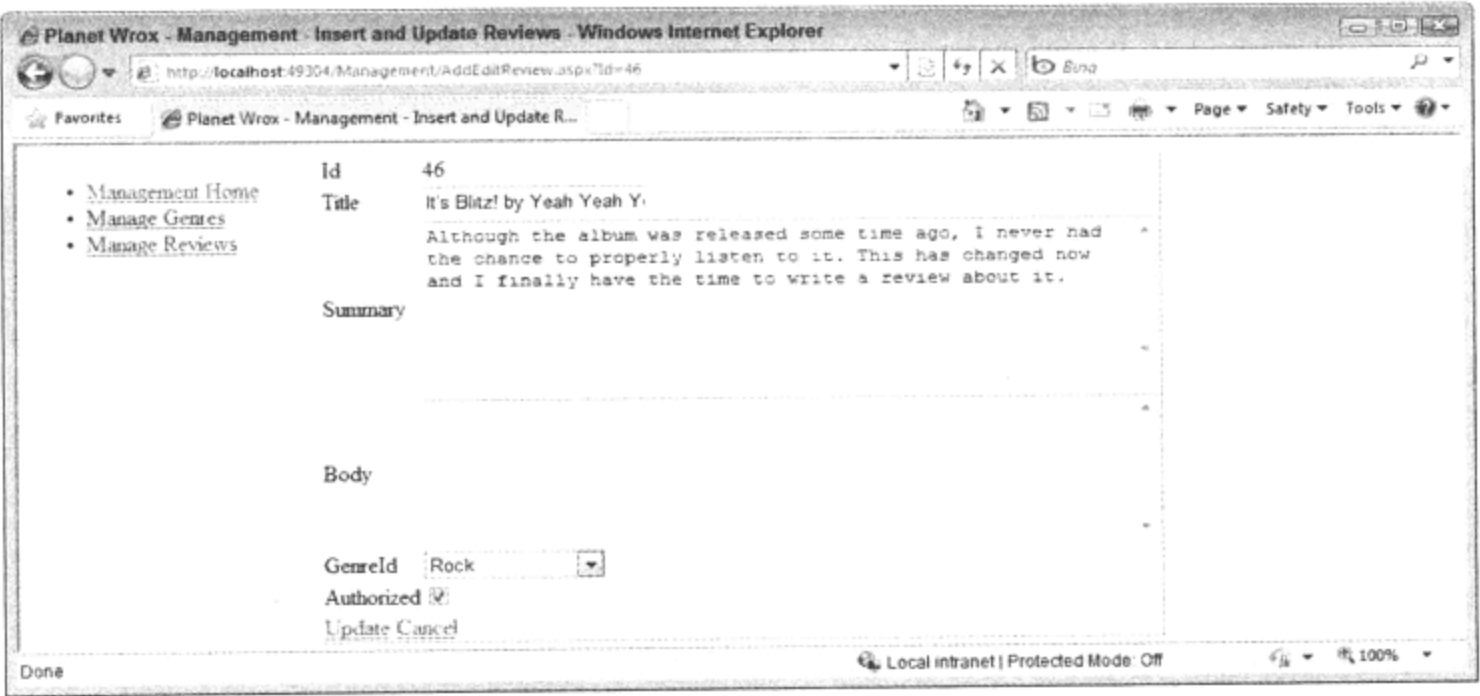


图 13-24

工作原理

DetailsView 和 SqlDataSource 控件做了大部分复杂工作。您创建了不同的模板，允许用户插入新记录和更新已有记录，然后那两个控件完成其余工作。一些有趣的事情可以帮助了解所有这一切的工作原理。首先看一下 Title 列的 InsertItemTemplate:

```
<InsertItemTemplate>
    <asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind("Title") %>'>
    </asp:TextBox>
    <asp:RequiredFieldValidator ID="reqVal2" ControlToValidate="TextBox1"
        runat="server" ErrorMessage="Please enter a title">
    </asp:RequiredFieldValidator>
</InsertItemTemplate>
```

这个代码段中最重要的部分是 TextBox 的 Text 属性被绑定的方式。在前面介绍了使用 Eval 的单向绑定语法，它主要是输出绑定列的值。不过，有了 Bind，会实现更强大的功能。它可以表示 SqlDataSource 的列和页面中的控件之间的双向(two directions)数据绑定。在这个例子中，评论的 Title 列绑定到 TextBox。这意味着在控件必须显示其数据(例如，更新已有记录)时，它知道必须显示评论的 Title。更重要的是在回发时，单击了 Update 链接后，控件仍知道 TextBox 控件和 Title 列的关系。

因此，如果在对页面中的评论作更改后单击 `Update`，`DetailsView` 就收集表单中的所有绑定数据(`Title`、`Summary`、`Body`、`GenreId` 和是否授权)，然后发送至 `SqlDataSource` 控件，该控件对于 `Review` 表中的每个相关列都设有参数：

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
...
<UpdateParameters>
  <asp:Parameter Name="Title" Type="String" />
  <asp:Parameter Name="Summary" Type="String" />
  <asp:Parameter Name="Body" Type="String" />
  <asp:Parameter Name="GenreId" Type="Int32" />
  <asp:Parameter Name="Authorized" Type="Boolean" />
  <asp:Parameter Name="UpdateDateTime" Type="DateTime" />
  <asp:Parameter Name="Id" Type="Int32" />
</UpdateParameters>
</asp:SqlDataSource>
```

最后，`SqlDataSource` 获取所有参数值，将这些值注入到 `UpdateCommand` 中，然后将它们发送到数据库。

这对于 `Review` 表中有表单控件与之相连的列都工作得很好，那么其他列呢？您可能注意到，`CreateDateTime` 不是任何 `SqlDataSource` 命令的一部分。由于数据库会自动插入当天的日期和时间，因此不需要在代码中包括它。`UpdateDateTime` 列有所不同。显然，您不希望用户手动输入此列的值，而应该由系统自动跟踪它。这也就是通过设置其 `Visible` 属性为 `False`，使该控件在用户界面中隐藏的原因。不过，由于 `Insert` 和 `Update` 命令仍希望获得此列的值，所以需要另一种不同的方式插入它。这就要用到 `Inserting` 和 `Updating` 事件了。看一下 `ItemInserting` 事件处理程序，来对其工作原理有个基本的了解：

VB.NET

```
Protected Sub DetailsView1_ItemInserting(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.DetailsViewInsertEventArgs) _
    Handles DetailsView1.ItemInserting
    e.Values("UpdateDateTime") = DateTime.Now
End Sub
```

C#

```
protected void DetailsView1_ItemInserting(object sender,
    DetailsViewInsertEventArgs e)
{
    e.Values["UpdateDateTime"] = DateTime.Now;
}
```

如前所见，`ItemInserting` 事件在 `InsertCommand` 发送到数据库之前触发。和 `UpdateDateTime` 一样，对于表中在用户界面中没有对应的相关控件的列来说，这是提供其(默认)值的最佳位置。这一代码只是将 `UpdateDateTime` 值设置为当天的日期和时间。然后，该值发送到数据库，在那里更新 `Review` 表的 `UpdateDateTime` 列。

`ItemUpdating` 命令的原理与此相同。在此事件中，需要对 `NewValues` 集合而不是 `Values` 集合进行索引，但是原理是一样的。

您可能认为对于 `Insert` 命令，不需要设置 `UpdateDateTime`。因为在插入新记录时，数据库会自

动插入值。不过，要区分插入和更新，需要更多的手动工作。必须从 InsertCommand 中删除列，然后从<InsertParameters>集合中删除列。尽管这本身并不需要大量的工作，但在后面尝试修改 SqlDataSource 的 SQL 命令时会陷入麻烦，因为 Insert 和 Update 命令现在不同步了。像这里一样通过代码简单设置 UpdateDateTime，就可以解决许多这样的问题。

在用 SqlDataSource 控件进行插入或更新时，它会分别触发 ItemInserted 或 ItemUpdated 事件。在这些事件中，EndEditing 方法被调用，它只是将用户带回 Reviews.aspx 页面。

VB.NET

```
Private Sub EndEditing()  
    Response.Redirect("Reviews.aspx")  
End Sub
```

C#

```
private void EndEditing()  
{  
    Response.Redirect("Reviews.aspx");  
}
```

随着对有关 DetailsView 控件所触发的不同事件的讨论，也将结束本章内容的介绍。到现在为止，您已经很好地了解了如何使用 GridView、DetailsView 和 SqlDataSource 控件执行 CRUD 操作。

虽然 SqlDataSource 控件很有用，但是许多开发人员不喜欢使用它。这种控件最大的一个缺点是 SQL 语句直接在 ASPX 页面中结束。如果要改变数据库模式，这就会产生许多问题。因为肯定会需要修改这个控件。在进行修改时，许多地方很可能无法正常工作。例如，如果将 Genre 表的 Name 列重命名为 Description，应用程序就会中断。但是，在运行之前不能注意到整个问题，因为 VWD 不能针对 SqlDataSource 控件中定义的命令文本来检查数据库模式。有一些方法可以解决这个问题。一种是构建强类型对象，并使用 ObjectDataSource 控件。关于这种解决方案的细节，本书不予讨论，但是在我的网站上提供了一系列演示这个概念的文章，您可以登录以下网址来查看该文章：<http://imar.spaanjaars.com/QuickDocId.aspx?quickdoc=476>。这些文章中讨论的概念相当高级，所以最好在阅读完本书之后再研究它们。

另外一种方法是使用 ADO.NET Entity Framework，第 14 章将讨论该主题，并将介绍如何使用 EntityDataSource 控件及 ListView 和 DataPager 控件来执行类似的操作，而不需要在代码中编写嵌套 SQL 语句。

13.5 显示和更新数据的实用提示

下面给出了一些显示和更新数据的实用提示：

- 总是将连接字符串存储在 web.config 文件中。尽管可以很容易地将连接字符串直接存储到页面的 SqlDataSource 控件中，但在后面需要对连接字符串作修改时会比较麻烦。
- 总是考虑为数据输入页面添加有效性验证控件。这使得用户可以更容易地知道哪些数据是必需的，应以哪种格式输入，同时避免系统接收或处理无效的或不正确的数据。

- 如果有较长的数据列表要显示，可以考虑对 GridView 这样的控件进行分页。如果提供包含太多项的列表，那么用户会感到厌烦。通常，10~20 项的页面最适宜。
- 考虑将页面中的控件重命名，而不是使用其默认值。例如，在前一个“试一试”练习中，将 SqlDataSource2 重命名为 GenresDataSource。这使人更容易明白需要从哪个数据源获取流派的信息。如果页面中控件较少，这不是个问题。但一旦页面上内容丰富起来，为控件选择易区分的名称就变得很重要。
- 考虑在 AddEditReview.aspx 中设置 Validation 控件的 CssClass 属性，并把它们挂钩到一个 CSS 类。现在可以在根文件夹下的 Styles 文件夹中的一个样式表中创建它们，并把该文件链接到母版页。在后面的章节中将为 Management 部分创建一个单独的主题。

13.6 本章小结

本章是以您在第 12 章中所学到的通过 SQL 访问数据库的知识为基础的。首先讨论了 VWD Toolbox 的 Data 类别中的多种控件。

这些控件可分为两大类：数据绑定控件和数据源控件。第一类控件(包括 GridView、DetailsView 和 ListView)用于在 Web 页面中显示数据。其中大部分(不是全部)控件也可通过提供插入、更新和删除功能来维护数据。

第二类控件，即数据源控件，没有可视的外观。它们充当用户界面和数据库之间的桥梁。有大量不同的数据源控件，各自提供了对特定数据存储的访问。本章介绍了 SqlDataSource 控件，它可用于从不同的关系数据库中检索数据。

13.7 练习

1. 如果需要创建一个用户界面，使用户显示、筛选、编辑和删除某个数据库的数据，最好使用哪个控件？如何将该控件与数据库相关联？
2. 如果想以下列格式显示数据库中流派的一个简单列表，需要选择哪种控件？

```
<ul>
  <li>Punk</li>
  <li>Hard Rock</li>
  <li>Jazz</li>
  <li>Techno</li>
</ul>
```

3. BoundField 和 TemplateField 之间的区别是什么？它们分别在何时使用？
4. 存储连接字符串的最佳位置是哪里？如何从那里访问连接字符串？为什么不将连接字符串存储在页面中？

练习的答案见附录 A。

本章要点回顾

连接字符串	保存连接到数据库(如 SQL Server)所必需的信息字符串
数据源控件	一组 ASP.NET 控件，用作数据源(数据库、XML 文件等)和数据绑定控件之间的桥梁
数据绑定表达式语法	用来将数据源的值绑定到控件属性(如标签和文本框)的语法。例如： Text='<%# Bind("Title") %>'
数据绑定控件	一组 ASP.NET 控件，可以显示平面和层次数据
表达式语法	一种绑定各种数据源的简洁语法，包括从 web.config 文件到控件属性的各种连接字符串。 例如： ConnectionString= "<%\$ ConnectionStrings:PlanetWroxConnectionString1 %>"
InsertParameters UpdateParameters DeleteParameters	一组参数，用来向数据源控件提供数据以支持插入、更新和删除行为
命名实例	特定 SQL Server 实例的名称。用于区分同一台计算机上的多个 SQL Server 安装版本
SelectParameters	一组参数，可以从其他数据源(查询字符串、cookie 等)获取数据，并且可以用在数据源控件中来筛选数据

第14章

LINQ 和 ADO.NET Entity Framework

本章要点

- LINQ 及其语法
- LINQ 的各种形式及其适用场合
- 如何使用 ADO.NET Entity Framework
- 如何使用新的 EntityDataSource 控件来访问 ADO.NET Entity Framework
- 如何使用 ListView 和 DataPager 控件

本书的上一版介绍的是.NET 3.5，在书中我曾提到过，LINQ 是我在.NET 3.5 中最喜欢的一种新功能。虽然现在 LINQ 已经不再是全新的功能，但是它仍然是我在.NET 4 中最喜欢使用的一种功能。LINQ 是一种与.NET Framework 中使用的编程语言紧密集成的查询语言。LINQ，即语言集成查询 (language-integrated query)，它使得可以像使用 SQL 查询数据库的数据那样从.NET 编程语言中查询数据。事实上，LINQ 语法部分模仿了 SQL 语言，它使得熟悉 SQL 的编程人员更容易上手。

LINQ 有一些不同的实现，允许从 VB.NET 或 C#代码中访问和查询大量不同的数据源，包括您自己代码中的集合、XML 文件、.NET DataSet 和数据库。在下一节中，将简要概述主要的几种 LINQ 实现，而其余部分则着重于 LINQ 语法和 ADO.NET Entity Framework(EF)，通过后面这种技术，在使用数据库时不必编写大量代码。ADO.NET Entity Framework 在后台大量使用 LINQ，所以您有一个很好的机会来练习新学习的 LINQ 技能。

14.1 LINQ 简介

使用 LINQ 可以直接通过代码查询多种数据源中的数据。LINQ 之于.NET 编程就像 SQL 之于关系数据库。通过简单的声明性语法，可以查询集合中匹配条件的对象。

LINQ 并不只是.NET Framework 的一个增件。相反，它被设计和实现为.NET 编程语言中的一部分。这意味着，LINQ 被真正集成到.NET 中，为查询数据提供了一个统一的方法，而不管数据的来源。因为它被集成到语言中，而不是特定的项目类型中。LINQ 可用于各种项目，包括 Web 应用程序

序、Windows Forms 应用程序、Console 应用程序等。为了帮助开发人员熟悉 LINQ，其语法基本模仿了 SQL(对于关系数据库来说是最为流行的查询语言)。这意味着 LINQ 具有像 Select、From 和 Where 这样的关键字来从数据源中获取数据。

为了了解 LINQ 查询，这里给出了一个简单的示例，它将显示一个姓名中包含字母 S 的 Wrox 作者的列表：

VB.NET

```
Imports System.Linq
...
Dim authors As String() = New String() {"Hanselman, Scott", "Evjen, Bill",
                                         "Haack, Phil", "Vieira, Robert", "Spaanjaars, Imar"}
Dim result = From author In authors
              Where author.Contains("S")
              Order By author
              Select author
For Each author In result
    Label1.Text += author + "<br />"
Next
```

C#

```
using System.Linq;
...
string[] authors = new string[] { "Hanselman, Scott", "Evjen, Bill",
                                   "Haack, Phil", "Vieira, Robert", "Spaanjaars, Imar" };
var result = from author in authors
              where author.Contains("S")
              orderby author
              select author;

foreach (var author in result)
{
    Label1.Text += author + "<br />";
}
```

尽管这个示例中所用的语法相当简单，但其作用是强大的。给定一个包含作者姓名的字符串的数组，只选择其姓名中包含大写字母 S 的作者，并按照升序排列它们。意料之中的是，在这个示例中，Label 控件显示了我的姓名和 Scott Hanselman，因为只有这两个姓名与 Where 条件匹配。注意，代码如何导入 System.Linq 名称空间。如果发现 IntelliSense 中没有显示某些关键字，或者 VWD 对 LINQ 查询给出了一个编译错误，那么需要检查在代码文件中是否导入了这个名称空间。

当然，这个示例只是一个开始。下面 3 节中讨论的各种 LINQ 类型可以用来对多种数据源创建强大得多的查询。

由于 LINQ 非常强大，又极具潜力，因此它被集成到 .NET Framework 的多个不同地方。下面将介绍不同的 LINQ 实现。

14.1.1 LINQ to Objects

这是语言集成的最基本形式。使用 LINQ to Objects，可以像前面那个例子中那样在 .NET 应用程序中查询集合。当然也不仅限于数组，因为 LINQ 允许查询 .NET Framework 中存在的几乎所有集合。

14.1.2 LINQ to XML

LINQ to XML 是读、写 XML 的一种新的.NET 方法。现在，可以在应用程序中编写直接针对 XML 的 LINQ 查询，而不是使用普通的 XML 查询语言，如 XSLT 或 XPath。

14.1.3 LINQ to ADO.NET

ADO.NET 是 .NET Framework 的一部分，它允许访问数据、数据服务(像 SQL Server)和其他许多不同的数据源。SqlDataSource 控件在后台也使用 ADO.NET。此外，ADO.NET 还常常用在“原始数据访问代码”中，即使用 C#或 VB.NET 编写，而不使用声明性数据控件的数据库连接代码。使用 LINQ to ADO.NET，可以查询与数据库相关的信息集，包括 LINQ to DataSet、LINQ to SQL 和 LINQ to Entities。

LINQ to DataSet 允许对 DataSet(一个表示内存中数据库的类)编写查询。


LINQ to SQL 允许在 .NET 项目中编写针对 Microsoft SQL Server 数据库的面向对象的查询。LINQ to SQL 将 LINQ 查询转换为 SQL 语句，然后再发送到数据库中执行 CRUD 的 4 种操作。在本书的上一版中，这一章整章都在介绍 LINQ to SQL。但是在上一版到这一版的这一段时间中，一些事情发生了变化。Microsoft 已经表示不会再积极开发 LINQ to SQL。在可预见的未来，它仍然将是 .NET Framework 和 Visual Studio 的一部分，但是 Microsoft 可能不会再向它添加新功能了。这是因为 LINQ to SQL 与 Entity Framework(EF)在功能上有很大的重叠。在 LINQ to SQL 中能实现的操作在 EF 中也能实现。但是，与 LINQ to SQL 相比，EF 的功能要强大得多，并且功能要丰富得多。正因为如此，Entity Framework 是比 LINQ to SQL 更好的选择。本章将主要讨论 Entity Framework。

要获得其他实现的更多信息，可访问 LINQ 官方网站，地址为 <http://tinyurl.com/42egdn>。

14.2 ADO.NET Entity Framework 简介

通过使用 ADO.NET Entity Framework(EF)，可以把许多数据库对象(如表)转换成可以在代码中直接访问的 .NET 对象。然后就可以在查询中或者直接在数据绑定中使用这些对象。EF 也允许执行相反操作：首先设计一个对象模型，然后让 EF 创建必要的数据库结构。

使用 EF 十分简单，并且十分灵活。通过使用关系图设计器，可以将表等数据库对象拖放到实体模型中。放到关系图上的对象将成为可用的对象。例如，如果将 Review 表放到关系图中那么就将得到一个强类型的 Review 类。可以使用 LINQ 查询或者其他方式创建这个类的实例。本章后面将对此进行介绍。



注意：ADO.NET Entity Framework 是一个庞大而复杂的主题，本章无法涵盖它的全部内容。要想深入理解 EF，可以参考 Julia Lerman 所著的 *Programming Entity Framework, 2nd Edition*(O'Reilly Media)一书。

当把多个相关的数据库表放到关系图上时，设计人员可以观察到表之间的关系，然后在对象模型中复制这些关系。例如，如果使用某些 LINQ to EF 查询在代码中创建了一个 Review 实例(稍后将会看到)，那么就可以访问它的 Genre 属性，进而可以访问 Name 等属性：

VB.NET

```
Label1.Text = myReview.Genre.Name
```

C#

```
Label1.Text = myReview.Genre.Name;
```

类似地，可以访问特定流派的相关 **Reviews** 集合以便将其绑定到数据绑定控件：

VB.NET

```
Repeater1.DataSource = myGenre.Reviews
```

C#

```
Repeater1.DataSource = myGenre.Reviews;
```

现在还不需要担心实际的语法，本章的后面将会详细介绍它们。对于本节来说，重要的是 EF 在 .NET 应用程序和 SQL Server 数据库之间创建了一个层。Entities Designer 会处理大部分工作，从而允许访问一个可以在应用程序中使用的干净的对象模型。

14.3 将数据模型映射到对象模型

通过使用 EF，可以把数据库项(如表、列和数据库中的关系)映射到应用程序的对象模型中的对象和属性。VWD 提供了很多优秀的工具来尽可能简化这种映射，如下面的“试一试”练习所示。

试一试

简单的 LINQ to Entities 示例

在这个试一试练习中，将看到如何将一个 ADO.NET Entity Data Model 文件添加到项目中，如何将数据库表添加到模型中，以及如何编写简单的 LINQ 查询来访问底层表中的数据。

(1) 打开 Planet Wrox 项目。右击 App_Code 文件夹，选择 Add New Item 命令，然后从 Installed Templates 下选择使用的编程语言。接着单击 ADO.NET Entity Data Model，输入名称 PlanteWrox，然后单击 Add 按钮将其添加到项目中。如果在列表中没有看到该项，检查右击的是 App_Code 而非其他文件夹，如 App_Data。

(2) 在显示的对话框中，确保选中 Generate from Database，然后单击 Next 按钮。

(3) 在 Choose Your Data Connection 这一步中，确保从下拉列表框中选择了 PlanetWroxConnectionString，并且选中了在 web.config 文件中存储设置的复选框。对话框现在看起来如图 14-1 所示。

单击 Next 按钮进入 Choose Your Database Objects 对话框。

(4) 在这个对话框中，展开 Tables，然后取消选择 Genre 和 Review 表。保持 sysdiagrams 表的未选中状态。这是 SQL Server 在内部使用的一个表，在 Planet Wrox 模型中不需要使用它。如果使用了 VWD 的英文版本，那么会有一个在模型中自动将所有名称转换为单数或者复数形式的选项，保持该选项的选中状态。对于其他语言的版本，需要手动执行这个操作，稍后将会介绍。最后，确保选中了在模型中包含外键列的选项。在本章后面将看到这个选项的用途。单击 Finish 按钮将模型添加到站点中。

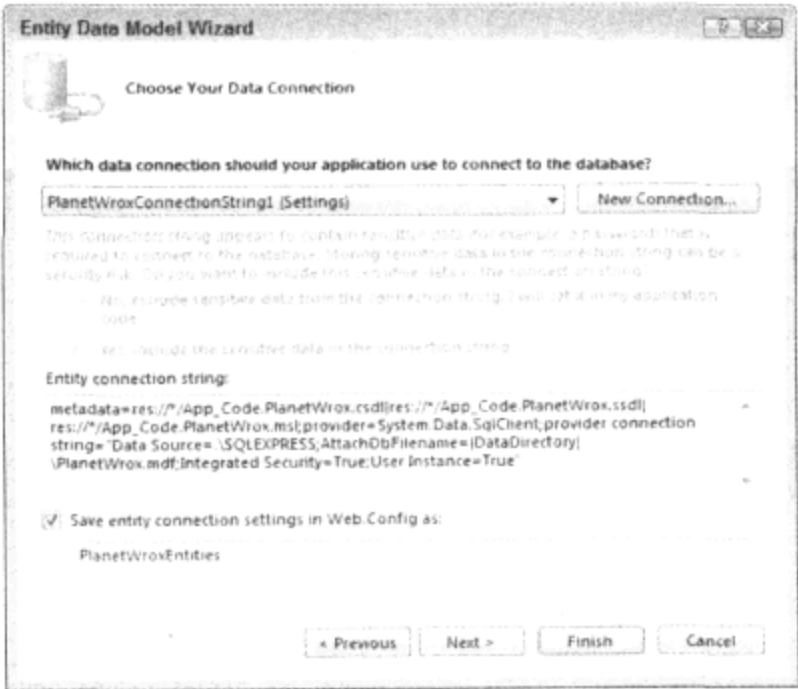


图 14-1

(5) VWD 将添加一个 PlanetWrox.edmx 文件，然后在主编辑器窗口中打开 Entity Designer，如图 14-2 所示。

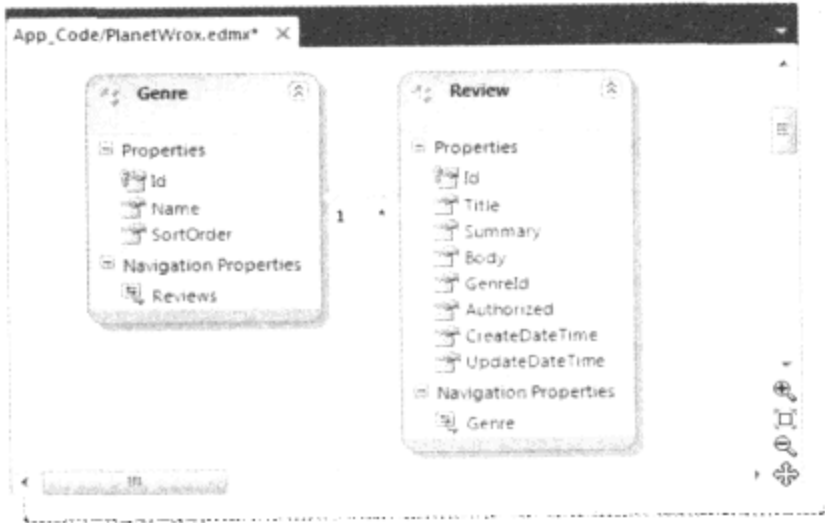


图 14-2

这个 Entity Designer 显示了基于数据库表生成的类。VWD 绘制一条连接两个类的线，表示它知道您在第 12 章创建的底层表之间的关系。如果没有看到这条线，或者没有在 Genre 类的底部看到 Reviews，或者没有在 Reviews 类的底部看到 Genre，那么要确保按照第 12 章的介绍设置了数据库，或者从本书附带代码的 Chapter 13 文件夹中获得数据库的一个副本。

(6) 如果使用的是 VWD 的非英文版本，那么需要手动将实体集和属性修改为复数形式。为此，在设计器中单击 Genre 类，按下 F4 键打开其 Properties 面板，并将 Entity Set Name 从 Genre 改为 Genres。为 Review 类重复这个过程，并将 Entity Set Name 改为 Reviews。最后，单击关系图中 Genre 类的 Review 属性(在图 14-2 中位于 Navigation Properties 标题的下面)，然后按 F2 键将该项重命名为 Reviews。因为一个 Review 只属于一个 Genre，所以不需要将 Review 类的 Genre 属性改为复数形式。

(7) 保存并关闭关系图。

(8) 打开 Reviews 文件夹中的 All.aspx，切换到 Design 视图，并从 Toolbox 中将 GridView 拖动到页面上。如果没有这个页面，那么现在就基于定制模板创建它。

(9) 双击页面的灰色只读区域，使 VWD 为页面的 Load 事件建立一个处理程序，然后添加如下

代码:

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Using myEntities As New PlanetWroxEntities()
        Dim allReviews = From review In myEntities.Reviews
            Where review.Authorized = True
            Order By review.CreateDateTime Descending
            Select review
        GridView1.DataSource = allReviews
        GridView1.DataBind()
    End Using
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
    {
        var allReviews = from review in myEntities.Reviews
            where review.Authorized == true
            orderby review.CreateDateTime descending
            select review;

        GridView1.DataSource = allReviews;
        GridView1.DataBind();
    }
}
```

注意，当输入 PlanetWroxEntities 时会立即显示一个错误，因为它是定义在作用域之外的一个名称空间中的。解决这个问题有两种方法。如果使用的是 VB.NET，那么可以输入 Imports PlanetWroxModel，如果使用的是 C#，那么可以在文件顶部输入 using PlanetWroxModel;。另外，可以单击 PlanetWroxEntities 一次，然后按下 Ctrl+. (Ctrl+句点)来打开一个对话框，从中选择解决问题的方法。图 14-3 显示了使用 C#语言的情况。选择第一项以后，VWD 会自动添加必要的 Imports/using 语句。

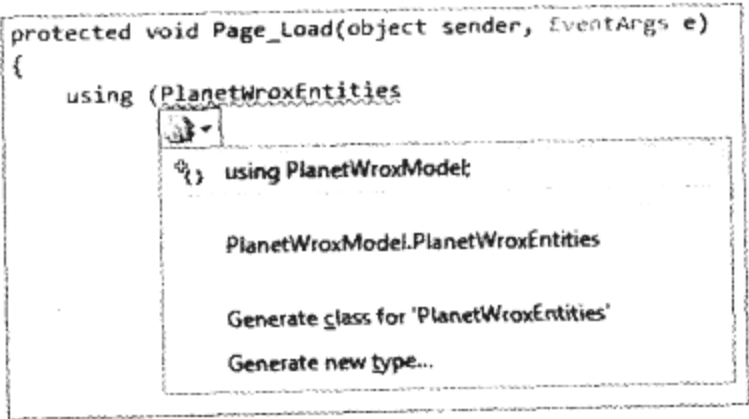


图 14-3

(10) 保存所有更改，然后按下 Ctrl+F5 键打开页面。这将显示一个充满评论的页面，其中评论来自于数据库中的 Review 表，如图 14-4 所示。

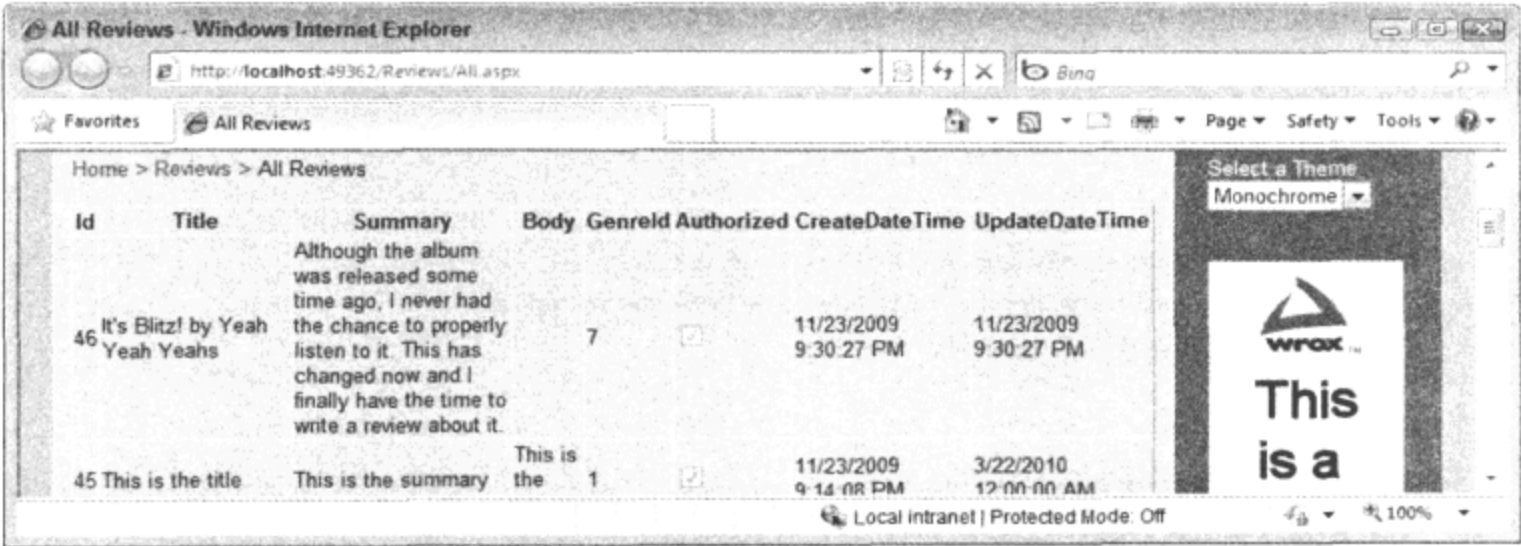


图 14-4

这个页面看上去有些混乱，因为数据是以 GridView 的格式显示的。在后面的“试一试”练习中将会讨论如何改进网格和数据的布局。

工作原理

EF 提供了一个对象关系设计器(可以通过 VWD 访问)，允许基于数据库的表创建一个可通过代码访问的对象模型。只要将表拖至该设计器，VWD 就会创建可用于访问数据库中底层数据的代码，而无需自己编写大量代码。拖至设计器上的类存储在.edmx 文件和其 Code Behind 文件中。设计器文件(PlanetWrox.edmx 文件的 Code Behind)包含了一个继承自ObjectContext 的类，而ObjectContext 是 EF 中提供对数据库进行访问的主要实体。在前一个示例中，这个类为 PlanetWroxEntities(按.edmx 文件命名)，使用它访问添加到关系图中的表的数据。尽管通常不需要查看生成的代码，但可以打开 PlanetWrox.designer.vb 或 PlanetWrox.designer.cs 进行查看。

这个设计器可以智能地检测数据库中的关系，因此也可以在代码中创建必要的关系，如图 14-2 所示。该模型定义两个主要的对象类型，即 Review 和 Genre，与之分别对应的集合是 Reviews 和 Genres。这些集合称作实体集(entity set)。注意，设计器的英文版本已经正确地将 Review 和 Genre 表的名称复数化(分别是 Reviews 和 Genres)，这样可以更容易地区分集合(Reviews)和对象实例(Review)。对于 VWD 的其他语言版本，必须使用 Entities Designer 来自己应用这种逻辑。

在生成模型后，对其执行 LINQ 查询，从而从底层数据库中获取数据。要访问这些数据，需要一个ObjectContext 类的实例，这在代码的 Using 块中创建。Using 块(C#中的 using)中包装的代码用于创建在用完后必须释放(从内存中清除)的变量。由于 myEntities 变量保存了(或没有)到 SQL Server 数据库的连接，因此将使用它的代码包装到 Using 块中是个好主意，这样对象会在块的末尾销毁。然后，这个 myEntities 对象提供可在查询中使用的数据(如评论和流派)。

VB.NET

```
Using myEntities As New PlanetWroxEntities()  
    Dim allReviews = From review In myEntities.Reviews  
                      Where review.Authorized = True  
                      Order By review.CreateDateTime Descending  
                      Select review  
    ...  
End Using
```

End Using

C#

```
using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
{
    var allReviews = from review in myEntities.Reviews
                      where review.Authorized == true
                      orderby review.CreateDateTime descending
                      select review;
    ...
}
```

可注意到，这一查询类似于在前面几章所学的 SQL。在后台，运行库将这个 LINQ 查询转换为相应的 SQL 查询，并对底层数据库执行这个查询。在这一查询中，From 子句中的 review 变量用于引用查询其他部分(Where、Order By 和 Select)的评论，从而可以指定选择、筛选和排序条件。

必须认识到，EF 使用了一种称为延迟加载(lazy loading)的技术，这意味着直到显式地说明要加载子对象时才会加载它们。这说明在前面的示例中，所查询的 Review 对象的 Genre 属性是 null，不包含任何数据。当代码尝试访问它们时，会通过向数据库执行另外一个查询来加载它们。这样，在不需要这些子对象时，就可以显著提高性能。但是，如果事先知道在代码中需要使用子对象，那么为每一项执行一个单独的 SQL 语句就会带来大量开销。如果是这样，可以在初始查询中预加载对象。为此，需要包含这些对象，并使用 Include 方法包含想要查询的类型。

VB.NET

```
Dim allReviews = From review In myEntities.Reviews.Include("Genre")
                  Where review.Authorized = True
    ...
```

C#

```
var allReviews = from review in myEntities.Reviews.Include("Genre")
                  where review.Authorized == true
    ...1
```

在把它们添加到查询中以后，Review 对象的 Genre 属性就得到了正确地填充。虽然初看上去这并不是很直观，效率好像也不高，但是实际上这是一个很不错的功能。如果在特定的页面中不需要这些额外的 Genre 属性，那么就不会有选择和返回这些对象带来的性能冲击。如果需要它们，所需要添加的只是对 Include 的一个调用。

除了 Reviews 集合外，该模型现在还包含一个 Genres 集合。当想要选择数据库中的全部流派时，可以使用下面这个查询：

```
VB.NET
Dim allGenres = From genre In myEntities.Genres
                 Order By genre.Name
                 Select genre

C#
var allGenres = from genre in myEntities.Genres
                 orderby genre.Name
                 select genre;
```


除了这两个单独的对象和它们的集合以外，它们都有引用对方类型的属性。例如，Review 实例有一个 Genre 属性，该属性提供了有关指派给该评论的流派的其他信息。Genre 实例也有一个 Reviews 集合属性，使您可以访问该流派中的所有评论。后面将介绍如何使用这些属性。

从这个“试一试”练习的第一个查询使用的关键字中，可很容易地看出该查询所做的工作：它获取一个系统中已授权的所有评论的列表，并将它们按创建日期以降序排列。然后查询的结果被赋给 allReviews 变量。注意，在两种语言中，都可以将查询展开到几行中，以便提高可读性。并不是必须这么做，但是建议这么做，因为这样更容易理解和维护查询。

这里您可能会注意到一些陌生的语法。例如，VB.NET 示例没有使用 As 子句来定义变量的类型。类似地，C#代码段使用了新的 var 关键字，同样也没有类型名。尽管可能无法从这两种语言的代码段中作出推断，但变量 allReviews 仍是强类型的，不只是一个未定义类型的变量。



注意：强类型指的是变量的类型在声明变量时即被显式定义。当定义了变量的类型后(在 VB 中使用 Dim，在 C#中使用类型名)，就不能在运行时进行修改。强类型语言(如 C#和 VB.NET)有许多优点，包括能够在编译时检查使用的类型，而这是弱类型编程语言所无法做到的。

因为代码没有指出 allReviews 的类型(示例则使用了 Dim 或 var)，.NET 需要一种不同的解决方案来确定变量类型。这是通过运用一个类型推理(type inference)的概念实现的，其中编译器可通过查看赋值的右侧推断变量的类型。在这里，编译器明白将从查询中返回一个 Review 对象列表，于是正确地将 allReviews 变量指定为泛型(generics type)，在 VB.NET 语法中是 IQueryable(Of Review)，在 C#中是 IQueryable<Review>。尽管这初看起来令人有些不安和感到不可思议，但如果您只是将它视作“可在查询中访问的一组 Review 对象”，那就很容易理解了。

然后，这些 Review 对象被指派给 GridView 的 DataSource 属性。在前面的章节中，已经了解了如何使用 DataSourceID 属性将像 GridView 这样的控件连接到像 SqlDataSource 这样的数据源控件上。而通过使用 DataSource 属性，可以指派控件后面用于构建 UI 的实际数据。

VB.NET

```
GridView1.DataSource = allReviews
GridView1.DataBind()
```

C#

```
GridView1.DataSource = allReviews;
GridView1.DataBind();
```

通过在 GridView 上调用 DataBind()，可命令该控件在页面上显示单独的 Review 对象。由于 GridView 控件的 AutoGenerateColumns 属性默认情况下为 True，它就会为它在 Review 对象上发现的每个属性创建一个列。后面将介绍如何自定义该控件和指派给 DataSource 属性的数据。

在下一节中，将学习更多关于 LINQ 查询语法的内容，该语言推动了.NET 4 查询能力的提高。

14.4 查询语法

前一个示例中所看到的查询是非常简单的：它向系统请求了所有已授权的评论，并以排序顺序返回它们。不过，LINQ 的查询能力远高于此。在本节，我们将学习更多的可用于查询对象模型的 LINQ 查询语法。记住，LINQ 语法并不是专门为 Entity Framework 设计的。下面介绍的大多数 LINQ 概念也适用于其他 LINQ 实现，如 LINQ to Objects 和 LINQ to ADO.NET。

14.4.1 标准查询操作符

LINQ 支持大量的查询操作符——可用于选择、排序或筛选从查询返回的数据的关键字。尽管本章所有示例是在 EF 的背景下讨论的，但也可将它们应用到其他 LINQ 实现中。接下来，将通过示例概述一些最为重要的标准查询操作符。每个示例都使用对象模型和之前创建的名为 myEntities 的ObjectContext对象作为查询的数据源。

1. Select

Select 关键字(C#中是 select)用于从查询的源中检索对象。在这个示例中，可看到如何选择已有类型的一个对象。在本章后面，将看到如何动态定义新的对象类型。

VB.NET

```
Dim allReviews = From r In myEntities.Reviews
                  Select r
```

C#

```
var allReviews = from r in myEntities.Reviews
                  select r;
```

这一示例中的变量 r 称为范围变量(range variable)，它只在当前查询中可用。通常在 From 子句中引入范围变量，然后在 Where 和 Select 子句再次使用它来筛选数据，表明要选择的数据。尽管对于它可采用任意的名称，但通常看到的都是单个字母的变量，如 r (表示 Review)，或所查询集合的单数形式(在前面的示例中是 review 而不是 r)。

2. From

尽管 From 子句(C#中是 from)不能算是标准查询操作符，因为它并不对数据进行操作而是指向数据，但它是 LINQ 查询中的一个重要元素，因为它定义了查询所执行的集合或数据源。在前面的例子中，From 子句表明查询必须对 EF 中的 myEntities 对象所提供的 Reviews 集合执行。

3. Order By

运用 Order By(C#中的 orderby，没有在 VB.NET 中使用的空格)，可以对结果集合中的项进行排序。Order By 后面紧跟着的是可选的用来指定排序顺序的 Ascending 或 Descending 关键字(C#中是 ascending 和 descending)。可以通过逗号分隔来指定多个条件。下列查询返回一个流派列表，首先按 SortOrder 列以降序排列，然后按 Name 列以升序(默认排序方式为升序)排列。

VB.NET

```
Dim allGenres = From g In myEntities.Genres
                  Order By g.SortOrder Descending, g.Name
                  Select g
```

C#

```
var allGenres = from g in myEntities.Genres
                 orderby g.SortOrder descending, g.Name
                 select g;
```

4. Where

和 SQL 中的 WHERE 子句一样，LINQ 中的 Where 子句(C#中的 where)允许筛选查询返回的对象。下列查询返回所有授权的评论：

VB.NET

```
Dim allReviews = From r In myEntities.Reviews
                  Where r.Authorized = True
                  Select r
```

C#

```
var allReviews = from r in myEntities.Reviews
                 where r.Authorized == true
                 select r;
```

注意，Where 子句使用了所选编程语言的标准等于运算符：VB.NET 中的 “=”，以及 C#中的 “==”。

5. Sum、Min、Max、Average 和 Count

这些聚集运算符允许在结果集中的对象上进行数学计算。例如，要检索所有评论的数量，可执行下列查询：

VB.NET

```
Dim numberOfReviews = (From r In myEntities.Reviews
                        Select r).Count()
```

C#

```
var numberOfReviews = (from r in myEntities.Reviews
                       select r).Count();
```

注意，Count 方法被应用于整个结果集。因此，需要将整个语句括到括号中，然后再调用 Count 方法。没有括号的话，就会出错。这个示例中的 numberOfReviews 变量将被推断为整型，包含 Review 表中评论的数量。

6. Take、Skip、TakeWhile 和 SkipWhile

Take 和 Skip 允许在结果集中作子选择。这很适用于分页情况，其中只检索当前页面的记录。Take 从结果集中获取所请求数量的元素，然后忽略其余的；而 Skip 则相反，它跳过请求数量的元素，

然后返回其余的。

在 EF 中，Take 和 Skip 操作符也被转换为 SQL 语句。这意味着分页是在数据库级发生的，而不是在 ASP.NET 页面中。这大大增强了查询的性能，特别是对于一些较大的结果集更是如此，因为不是所有的元素都必须从数据库转移到 ASP.NET 页面中。

要想使用 Skip，必须在跳过指定数量的记录之前，向查询中添加一个 Order By 子句(C#中为 orderby)来对结果进行排序。如果不显式地添加 Order By 子句，数据库就可能以无法预知的顺序返回结果，因此在 LINQ 查询中添加 Order By 动作有助于从 Skip 方法获得一致的结果，因为在跳过和获取记录之前，会先对它们进行排序。

下面的例子显示了如何检索第二页的记录，假定页面大小为 10。

VB.NET

```
Dim allReviews = (From r In myEntities.Reviews
                  Order By r.Title
                  Select r).Skip(10).Take(10)
```

C#

```
var allReviews = (from r in myEntities.Reviews
                  orderby r.Title
                  select r).Skip(10).Take(10);
```

和 Count 示例一样，该查询也被括在一对括号中，然后调用 Skip 和 Take 来获取请求的记录。

TakeWhile 和 SkipWhile 查询操作符的工作方式类似，但允许在特定条件满足时获取或跳过一些记录。遗憾的是，在 EF 中无法使用它们，但是通常可以通过给查询添加一个简单的 Where 子句来解决这个问题。

7. Single 和 SingleOrDefault

Single 和 SingleOrDefault 操作符允许返回单个对象作为强类型实例。如果知道查询只返回一条记录，这就很有用；例如，通过其唯一 ID 检索它。下列示例从数据库中检索 ID 为 37 的评论。

VB.NET

```
Dim review37 = (From r In myEntities.Reviews
                Where r.Id = 37
                Select r).Single()
```

C#

```
var review37 = (from r in myEntities.Reviews
                where r.Id == 37
                select r).Single();
```

如果请求的项未找到或是查询返回多个实例，Single 操作符就会引发异常。如果想让该方法返回 null(VB.NET 中是 Nothing)，例如对于未找到的 Review 或 Genre，或是返回相关数据类型的默认值(如 Integer 型的 0、Boolean 型的 False 等)，则使用 SingleOrDefault。

即使数据库中只有一个 ID 为 37 的 Review，如果未调用 Single，则仍会返回一个 Reviews 集合(其中只有一个元素)。通过使用 Single，可强制结果集为所查询类型的单个实例。

8. First、FirstOrDefault、Last 和 LastOrDefault

这些操作符允许返回特定对象序列中的第一个或最后一个元素。和 Single 方法一样，如果集合为空，First 和 Last 就会抛出异常，而 FirstOrDefault 和 LastOrDefault 则返回相关数据类型的默认值。

和 Single 不同的是，当查询返回多个项时，First、FirstOrDefault、Last 和 LastOrDefault 操作符并不抛出异常。

但是，EF 中并不支持 Last 和 LastOrDefault 查询。不过，通过使用 First 和降序排列顺序可轻松实现与之相同的行为。下列代码段显示了如何从数据库检索最早(具有最小的 ID)的和最新的评论。

VB.NET

```
Dim firstReview = (From r In myEntities.Reviews
                   Order By r.Id
                   Select r).First()

Dim lastReview = (From r In myEntities.Reviews
                  Order By r.Id Descending
                  Select r).First()
```

C#

```
var firstReview = (from r in myEntities.Reviews
                   orderby r.Id
                   select r).First();

var lastReview = (from r in myEntities.Reviews
                  orderby r.Id descending
                  select r).First();
```

在执行 First 前按相反顺序对结果集重新排序，就可以得到序列中的最后一条记录。注意在这两种情况中，查询返回的类型是一个真正的 Review 对象，允许直接访问其属性，如 Id 和 Title。

14.4.2 用匿名类型定形数据

到目前为止，在前面几节中看到的查询都返回的是全类型。即，查询返回了一个 Review 实例的列表(例如 Select 方法)、单个 Review 实例(Single、First 或 Last)或是一个数值(如 Count 和 Average)。

不过，通常不需要这些对象的所有信息。图 14-4 显示了一个带有 Review 对象中所有属性的 GridView。为了改进此列表的表示，您可能想省去像 Body 和 Authorized 这样的属性，想用流派名来取代流派 ID。虽然可以告诉 GridView 只显示想看到的列，但是如果能够限制实际的数据，那么会更高效一些。通过匿名类型(anonymous type)可轻松实现这种功能。这是 C#和 VB.NET 语言中的另一种功能。匿名类型是一种不需要像使用其他类型(如类)时那样预先定义名称的类型。而是可以通过选择数据，然后让编译器推断其类型来进行构造。

如果没有定义实际类型并进行命名，那么如何访问这一类型及其属性呢？这将再次用到类型推理，在其中编译器能明白将什么数据赋给变量，然后快速创建一个新的匿名类型。

创建匿名类型很简单：不需要使用像 Select review 这样的语句选择实际的对象，而是可以在 C#中使用 new 关键字，在 Visual Basic 中使用 New With，然后在 一对花括号间定义要选择的属性。

VB.NET

```
Dim allReviews = From myReview In myEntities.Reviews
```

```
Where myReview.Authorized = True
Select New With {myReview.Id, myReview.Title, myReview.Genre.Name}
```

C#

```
var allReviews = from review in myEntities.Reviews
    where review.Authorized == true
    select new { review.Id, review.Title, review.Genre.Name };
```

尽管这一类型是匿名的，不能通过名称直接访问，但编译器仍能推断其类型，对于在查询中选择的新属性提供了完全智能感知(IntelliSense)功能。图 14-5 显示了如何使用 C#中的 var 关键字访问 allReviews 变量中匿名类型的属性。

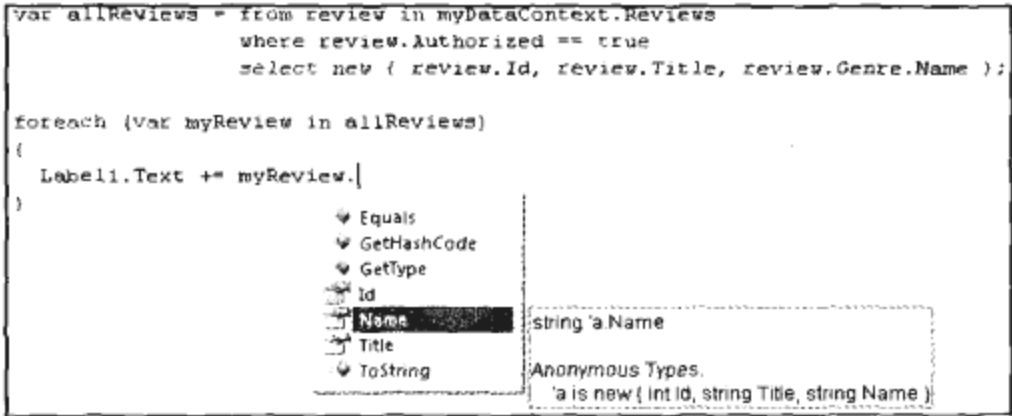


图 14-5

注意，前面的查询访问了 Review 的实际 Genre 属性。除了其 GenreId(定义为数据库的 Review 表中的一列)外，Review 类现在也有一个强类型的 Genre 属性，如前一查询所演示的那样，通过该属性可直接访问流派的属性，如 Name。

除了直接选择已有属性外——如查询中显示的那样选择 Genre 的 Id、Title 和 Name，可以创建属性值并提供不同的名称。例如，下列查询创建了一个新的匿名类型，将 Id 重命名为 Number，限制 Title 为开始的 20 个字符，包含了一个 Boolean 值来表示是否该项已经在数据库中作了更新：

VB.NET

```
Dim allReviews = From myReview In myEntities.Reviews
    Select New With
    {
        .Number = myReview.Id,
        .Title = myReview.Title.Substring(0, 20),
        myReview.Genre.Name,
        .HasBeenUpdated = (myReview.UpdateDateTime >
            myReview.CreateDateTime)
    }
```

C#

```
var allReviews = from myReview in myEntities.Reviews
    select new
    {
        Number = myReview.Id,
        Title = myReview.Title.Substring(0,20),
        myReview.Genre.Name,
        HasBeenUpdated = (myReview.UpdateDateTime >
```

```
        myReview.CreateDateTime)
    );
```

注意 VB.NET 和 C# 之间的差别：在 VB.NET 示例中，新属性的名称 (Number、Title 和 HasBeenUpdated) 都带有点号 (.) 作为前缀。C# 没有这一要求，可以直接写新属性的名称。

选择原始对象中并不存在的额外属性的能力给数据显示提供了很大的灵活性。这一示例只是通过比较 CreateDateTime 和 UpdateDateTime 属性来确定当前评论是否已经更新。然后比较的结果 (True 或 False 的布尔值) 被存储在 HasBeenUpdated 属性中。您可以选择任何内容，包括当前日期和时间、复杂的计算、子字符串或属性的组合等。

在下一个“试一试”练习中，将介绍如何创建一个将 Reviews 集合作为属性的新匿名类型。将使用这一类型创建一个数据库中所有可用流派的列表和每个流派包含的评论。

试一试

运用查询和匿名类型

在这个“试一试”练习中，创建一个页面，其中列出了所有可用流派，每个流派后都有在该流派中发布的评论列表。将使用 Repeater 控件显示流派列表，并用嵌套的 BulletedList 显示其中的评论。完成之后，结果类似于图 14-6 所示。

(1) 打开 Reviews 文件夹中的 AllByGenre.aspx 页面。确保该页面在 Markup 视图中，然后从 Toolbox 的 Data 类别中拖动一个 Repeater 到 cpMainContent 内容占位符的起始和结束标记中。

(2) 在 Repeater 的起始和结束标记之间，创建一个 <ItemTemplate> 元素。在该元素中创建一个 <h3> 元素，后者包含一个 Literal。代码如下所示：

```
<asp:Repeater ID="Repeater1" runat="server">
  <ItemTemplate>
    <h3><asp:Literal ID="Literal1" runat="server"></asp:Literal></h3>
  </ItemTemplate>
</asp:Repeater>
```

(3) 将 Literal 控件的 Text 属性设置为 <%# Eval(" Name ") %>。注意要确保属性的值用单引号括起来，而不是双引号。

```
<asp:Literal ID="Literal1" runat="server"
  Text='<%# Eval("Name") %>'></asp:Literal>
```

(4) 在 <h3> 元素下，从 Standard 类别中拖出一个 BulletedList 控件，并在其上设置如表 14-1 所示的属性。可以直接在 Markup 视图中输入它们或使用 Properties 面板。

表 14-1	
属 性 名	值
ID	ReviewList
DataSource	<%# Eval("Reviews"%)>(确保像如下代码段所示用单引号将属性值括起来)
DataTextField	Title
DisplayMode	Text

得到的控件代码如下所示：


```
<asp:BulletedList ID="ReviewList" runat="server"
    DataSource='<%# Eval("Reviews")%>' DataTextField="Title"
    DisplayMode="Text"></asp:BulletedList>
```

(5) 切换至 Design 视图并双击页面上由母版页定义的只读区域的某个位置，为页面的 Load 事件建立处理程序。在该事件处理程序中，编写下列代码。同样，使用 Ctrl+. 让 VWD 为 PlanetWroxEntities 类插入正确的名称空间。

VB.NET

```
Imports PlanetWroxModel
... ' Class definition goes here
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Using myEntities As New PlanetWroxEntities()
        Dim allGenres = From genre In myEntities.Genres
            Order By genre.Name
            Select New With {genre.Name, genre.Reviews}
        Repeater1.DataSource = allGenres
        Repeater1.DataBind()
    End Using
End Sub
```

C#

```
using PlanetWroxModel;
... // Class definition goes here
protected void Page_Load(object sender, EventArgs e)
{
    using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
    {
        var allGenres = from genre in myEntities.Genres
            orderby genre.Name
            select new { genre.Name, genre.Reviews };
        Repeater1.DataSource = allGenres;
        Repeater1.DataBind();
    }
}
```

(6) 保存对页面的更改，然后在浏览器中打开它。将看到如图 14-6 所示的结果，其中每个流派作为组标题出现在评论列表之上。

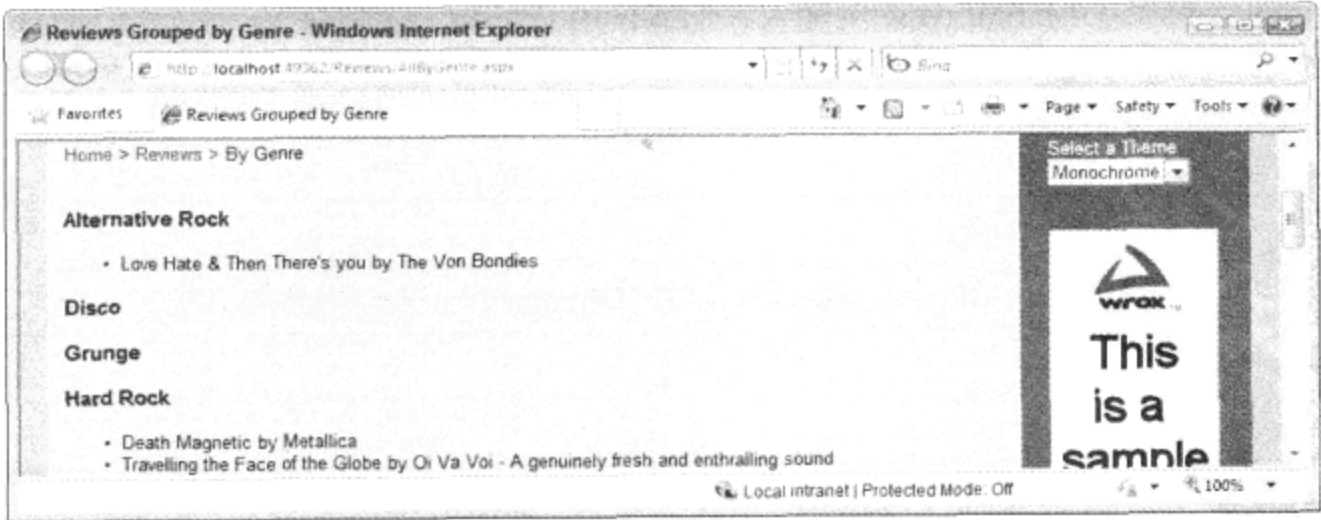


图 14-6

工作原理

在这个“试一试”练习中有两个重点。首先，有一个 LINQ 查询用于从数据库中获取流派和评论。该查询创建了一个新匿名类型，它带有两个属性：作为 String 的 Genre 的 Name 和名为 Reviews 的 Review 对象的集合。这个新匿名类型的类关系图如图 14-7 所示。

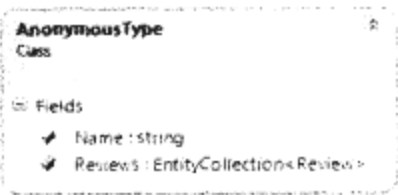


图 14-7

然后这些 Name 和 Reviews 字段用于第二个重要的部分：带有嵌套的项目符号列表的 Repeater 控件。首先看一下 Repeater:

```
<asp:Repeater ID="Repeater1" runat="server">
  <ItemTemplate>
    <h3><asp:Literal ID="Literal1" runat="server"
      Text='<%# Eval("Name") %>'></asp:Literal></h3>
    <!-- BulletedList here -->
  </ItemTemplate>
</asp:Repeater>
```

尽管我们之前未使用过 Repeater 控件，但它看起来很熟悉，因为它的工作方式与其他数据控件类似。在<ItemTemplate>中定义了想对数据源中的每项重复的标记。使用 Eval，可以获得 Title 的值并将它指派给包装在一对<h3>标记中的 Literal。类似的结构用于 BulletedList 来为其提供 DataSource。

```
<asp:BulletedList ID="BulletedList1" runat="server" DisplayMode="Text"
  DataSource='<%# Eval("Reviews") %>' DataTextField="Title" />
```

除了从底层数据项的 Name 指派像 Literal 的 Text 这样的简单属性外，还可以使用 Eval 获取复杂的属性。在这个示例中，Eval(" Reviews ")用于获取当前 Genre 的 Reviews 集合。然后 BulletedList 控件理解如何处理这一数据源并从每个单独的 Review 对象中检索 Title，然后在列表中显示。图 14-8 中的图表显示了每个 Genre 都包含一个或多个评论(其标题显示在流派名下)。

Name (from AnonymousType(0).Name)
Review Title (from AnonymousType(0).Reviews(0).Title)
Review Title (from AnonymousType(0).Reviews(1).Title)
Review Title (from AnonymousType(0).Reviews(2).Title)
Review Title (from AnonymousType(0).Reviews(3).Title)
Name (from AnonymousType(1).Name)
Review Title (from AnonymousType(1).Reviews(0).Title)
Review Title (from AnonymousType(1).Reviews(1).Title)
Name (from AnonymousType(2).Name)
Review Title (from AnonymousType(2).Reviews(0).Title)
Review Title (from AnonymousType(2).Reviews(1).Title)
Review Title (from AnonymousType(2).Reviews(2).Title)

图 14-8

在建立了 Repeater 并定义查询后，需要开始数据绑定过程。这可以通过将查询的结果指派给 Repeater 的 DataSource 属性，然后再调用 DataBind()来实现，如下面的 C#示例所示:

```
Repeater1.DataSource = allGenres;
```

```
Repeater1.DataBind();
```

这两行代码是动态设置：只要调用 `DataBind()`，就执行查询并从数据库中检索相关流派和评论。在这个示例中，流派按它们的 `Name` 排序，但显然也可按其他属性排序，如 `SortOrder`。然后 `Repeater` 循环结果集中的每一项(项是匿名类型的)，使用项的 `Name` 填充带有流派名的 `<h3>` 元素。接着，`Repeater` 将 `Reviews` 集合指派给内部 `BulletedList` 控件的 `DataSource` 属性。这一控件循环遍历可用的 `Review` 实例，使用它们的 `Title` 构建项目列表。在本例中，可以看到没有评论的流派也会显示在列表中。在本章最后的练习部分有一个练习显示了如何隐藏空流派。

尽管要完全了解这些 LINQ 查询和 Entity Framework 背后的原理需要一段时间，但我相信您一定认识到了它们的强大和可访问性。只需要少量几行代码和几个控件，就可以创建强大的数据显示页面。

不过，还可以使用更少的代码创建针对 EF 执行的 LINQ 查询并将它们与 ASP.NET 服务器控件一起使用。下一节介绍新的 `EntityDataSource`、`ListView` 和 `DataPager` 控件时将看到相关内容。

14.5 结合使用服务器控件和 LINQ 查询

到目前为止，我们已经了解了一种将 EF 内 LINQ 查询的结果绑定到 ASPX 页面中的控件的方法：将数据指派给控件的 `DataSource` 属性，然后调用 `DataBind`。这一向控件中输入数据的方法有一些缺点。首先，它不支持直接编辑、更新和删除数据。其次，由于在 `Code Behind` 中定义了数据源，`GridView` 直到运行时才知道给它提供了什么数据，因此没有工具支持建立其列。使用一些新的服务器控件就很容易克服这些缺点，这些服务器控件包括 `ListView` 和 `EntityDataSource` 控件。

14.5.1 在 Entity Framework 中使用数据控件

第 13 章介绍了一些数据控件，如 `GridView` 和 `SqlDataSource`。但是 ASP.NET 4 提供了更多的控件，从而允许使用更少的代码来创建数据驱动的面。其中两个控件为 ASP.NET 页面提供了可视界面，另一个控件用作数据绑定控件和底层数据源之间的桥梁。表 14-2 简要地介绍了这些控件。

表 14-2

控 件	说 明
EntityDataSource	和第 13 章介绍的 <code>SqlDataSource</code> 一样， <code>EntityDataSource</code> 用作数据绑定控件和底层数据源(在这里为 EF)之间的桥梁。
ListView	<code>ListView</code> 控件提供了一个可视界面，允许显示、插入、编辑和删除数据库中的项，从而提供了完整的 CRUD 服务
DataPager	<code>DataPager</code> 与 <code>ListView</code> 结合使用，允许对数据源中的数据进行分页，以分块的方式将数据提供给用户，而不是一次传递全部记录

下面几节将对这些控件作详细介绍，并在后面的一些“试一试”练习中介绍如何使用它们。

1. EntityDataSource 控件

从其名称可看出，EntityDataSource 和 SqlDataSource 及其他数据源控件类似。EntityDataSource 控件之于 EF 就像 SqlDataSource 控件之于基于 SQL 的数据源：它提供了一个声明性的方法来访问模型。和 SqlDataSource 控件一样，EntityDataSource 提供了对 CRUD 操作的轻松访问，另外使数据排序和筛选也变得非常简单。表 14-3 描述了这一新控件的主要属性和功能。

表 14-3

属 性	描 述
EnableDelete EnableInsert EnableUpdate	确定控件是否提供自动插入、更新和删除功能。如果启用，可以结合使用该控件和数据绑定控件(如 GridView 或 ListView)来支持数据管理
ContextTypeName	控件将使用的ObjectContext类的名称。在本书的示例中，这个类型名为PlanetWroxEntities
EntitySetName	想使用的EF关系图中的实体集名，如Reviews
Select OrderBy Where	允许定义EntityDataSource控件对模型触发的查询。每个属性都映射到前面见到过的一个查询操作

和数据绑定控件一起，EntityDataSource 通过 LINQ to EF 提供了对底层 SQL Server 数据库的完全访问。下一“试一试”练习将显示如何在 ASPX 页面中使用该控件。

试一试

一个简单的 EntityDataSource 应用

在这个“试一试”练习中，首先构建 Planet Wrox 的 Gig Pics 功能，在站点的这一部分中，用户可以上传他们喜欢的乐队举办音乐会时的照片。您将了解如何让用户创建一个新的相册作为上传照片的容器。了解如何使用 EntityDataSource 和 DetailsView 创建一个用户界面，允许用户将相册的名称输入系统。在后面的“试一试”练习中，将看到如何将图片添加到这一相册中。

(1) 使用 Database Explorer 窗口将表 14-4、表 14-5 添加到数据库中。可参阅第 12 章了解更多创建表、主键和标识列的详细内容。如果不想手动创建表，可采用随本章的代码下载一起提供的数据库。

表 14-4

PhotoAlbum

列 名	数 据 类 型	说 明
Id	int	相册的唯一 ID(标识和主键)
Name	nvarchar(100)	相册的名称

表 14-5

Picture		
列 名	数 据 类 型	说 明
Id	int	图片的唯一 ID(标识和主键)
Description	nvarchar(300)	描述图片的简短文本
Tooltip	nvarchar(50)	在图片上悬停鼠标时显示的工具提示
ImageUrl	nvarchar(200)	到磁盘上图片的虚拟路径
PhotoAlbumId	int	图片所属相册的 ID

对于这两个表，确保表中的列都不能为空，这是通过取消选择它们的 Allow Nulls 复选框实现的。使 Id 列为主键列，方法是单击它，然后单击 Table Designer 工具栏上的黄色钥匙图标。另外，通过设置 Column Properties 面板上的 Is Identity 属性为 Yes，使该列作为表的标识列。可参阅第 12 章了解如何设置。

(2) 在 Database Explorer 中，打开在第 12 章创建的数据库关系图，向其中添加两个新表(方法是从 Database Explorer 中拖动它们)。如果有必要，并排放置这两个新表。然后将 PhotoAlbum 表的 Id 列拖至 Picture 表的 PhotoAlbumId 列。确认 Primary key table 是 PhotoAlbum，所选列为 Id；以及 Foreign key table 是 Picture，所选列为 PhotoAlbumId，如图 14-9 所示。

(3) 单击 OK 两次应用改动，然后保存并关闭关系图。单击 Yes 确认对两个表所做的改动。

(4) 接着，双击 App_Code 文件夹中的 ADO.NET Entity Framework Model 文件 PlanetWrox.edmx 打开它。右击关系图的空白位置，选择 Update Model from Database 命令。在显示的向导中，展开 Tables，然后选中刚才创建的两个表：PhotoAlbum 和 Picture。单击 Finish 将这两个表添加到模型中。这时的关系图如图 14-10 所示。

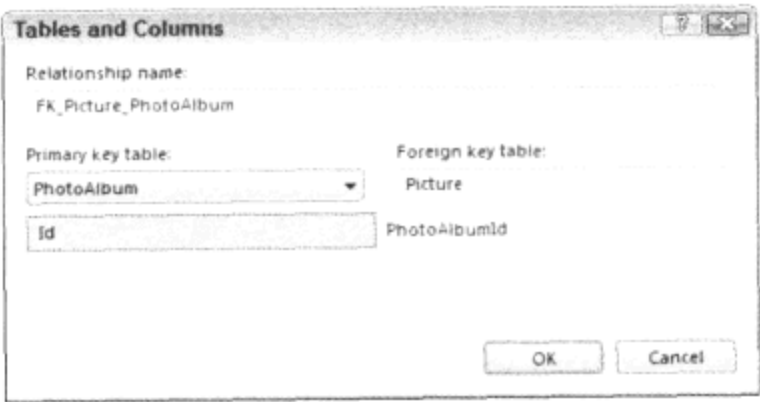


图 14-9



图 14-10

如果使用的是 VWD 的非英文版本，则需要再次复数化实体集和属性的名称。为此，单击 Picture 类，按下 F4 键打开 Properties 面板，并将 Entity Set Name 从 Picture 改为 Pictures。为 PhotoAlbum 类

重复这个过程，并将其 Entity Set Name 改为 PhotoAlbums。最后，在关系图上单击 PhotoAlbum 类的 Picture 属性，按下 F2 将该项重命名为 Pictures。

保存更改并关闭关系图。

(5) 在站点的根文件夹中基于自定义模板创建一个新的 Web 窗体，将它命名为 NewPhotoAlbum.aspx。将页面的标题设置为 Create New Photo Album。

(6) 切换页面至 Design 视图，从 Toolbox 的 Data 类别中拖一个 DetailsView 控件到 cpMainContent 占位符中。打开 DetailsView 控件的 Smart Tasks 面板，并从 Choose Data Source 下拉列表中选择<New data source>。在 Data Source Configuration Wizard 对话框中，单击 Entity 图标并单击 OK 按钮。在 Named Connection 下拉列表中，选择 PlanetWroxEntities。



注意：如果得到一个提示元数据不正确的错误，那么关闭对话框，从页面中删除已有的 EntityDataSource 控件，并从 Toolbox 中手动拖动一个新的 EntityDataSource 控件。在 DetailsView 控件的 Smart Task 面板中，选择新的数据源控件。然后打开 EntityDataSource 控件的 Smart Task 面板，并选择 Configure Data Source。

单击 Next 按钮到达 Configure Data Selection 屏幕，如图 14-11 所示。从 EntitySetName 下拉列表中，选择 PhotoAlbums 项。

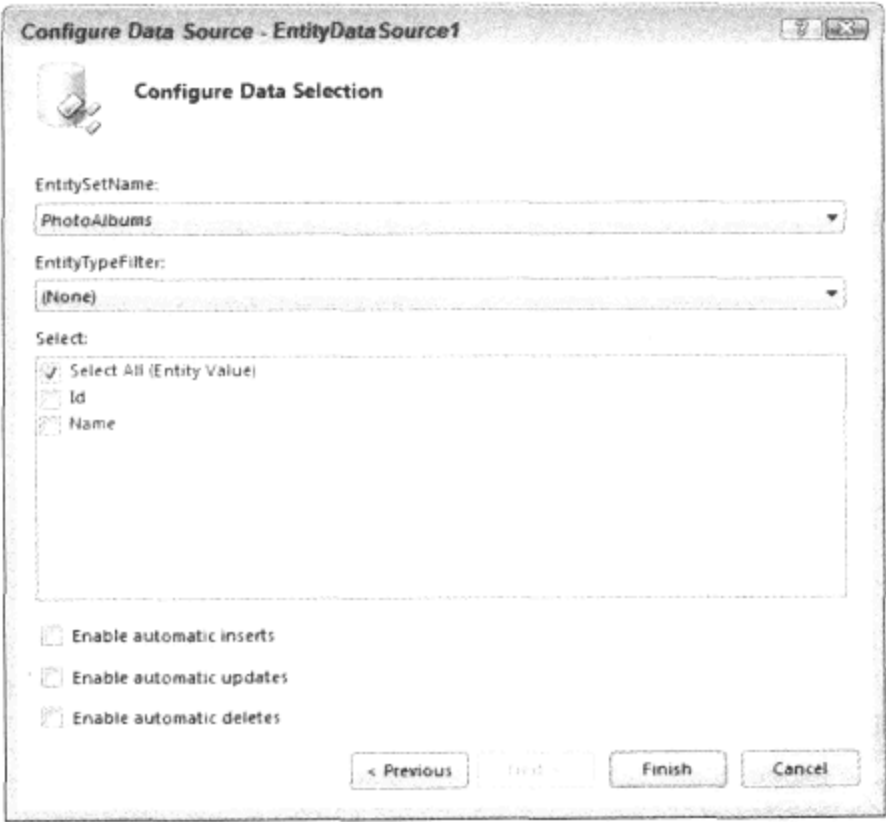


图 14-11

(7) 对于这一练习，需要插入操作，因此取消选择 Enable Automatic Inserts 复选框。单击 Finish 按钮来关闭 Data Source Wizard。

(8) 如果在 DetailsView 中没有看到 Id 和 Name 列，而是看到了一个 Databound Col0，那么需要在控件的 Smart Task 面板中单击 Refresh Schema 链接。

通过在相同的 Smart Tasks 面板上选中 Enable Inserting，为 DetailsView 控件启用插入功能。

(9) 打开 DetailsView 控件的 Properties 面板，将 DefaultMode 从 ReadOnly 改为 Insert。

(10) 切换到 Markup 视图，找到 PhotoAlbum 的 Id 属性的 BoundField，将其 InsertVisible 属性设为 False，这样就不会在插入新相册的时候显示一个要求输入 ID 的文本框。

```
<asp:BoundField DataField="Id" HeaderText="Id"
    SortExpression="Id" InsertVisible="False" />
```

(11) 在 Design 视图中选择 EntityDataSource，打开其 Properties 面板，切换到 Events 选项卡。双击 Inserted 事件，如图 14-12 所示。

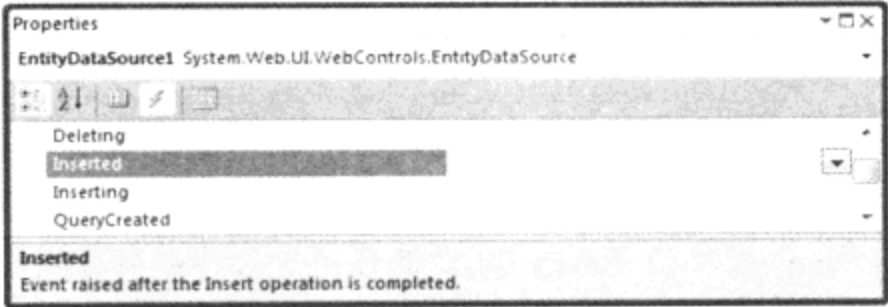


图 14-12

(12) 在代码文件的上部，添加下面的 Imports/using 语句，使实体模型位于作用域内，以便能够访问 PhotoAlbum 等类。

```
VB.NET
Imports PlanetWroxModel
```

```
C#
using PlanetWroxModel;
```

然后，在 VWD 自动添加的事件处理程序中，编写下面的代码，一旦将相册插入到数据库中，就将用户重定向到一个新页面。

```
VB.NET
Protected Sub EntityDataSource1_Inserted(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.EntityDataSourceStatusEventArgs) _
    Handles EntityDataSource1.Inserted
    Dim myPhotoAlbum As PhotoAlbum = CType(e.Entity, PhotoAlbum)
    Response.Redirect(String.Format("ManagePhotoAlbum.aspx?PhotoAlbumId={0}",
        myPhotoAlbum.Id.ToString()))
End Sub
```

```
C#
protected void EntityDataSource1_Inserted(object sender,
    EntityDataSourceChangedEventArgs e)
{
    PhotoAlbum myPhotoAlbum = (PhotoAlbum)e.Entity;
    Response.Redirect(string.Format("ManagePhotoAlbum.aspx?PhotoAlbumId={0}",
        myPhotoAlbum.Id.ToString()));
}
```

(13) 保存所有更改，然后在浏览器中打开 NewPhotoAlbum.aspx。



常见错误：如果得到的页面为空页面，那么要确保将 DetailsView 的 DefaultMode 属性设置为 Insert。

为相册输入一个新名称，比如 Placebo playing live at Rock Werchter 2009，然后单击 Insert 链接。这将得到一个 Resource Not Found 的错误(因为还没有创建 ManagePhotoAlbum.aspx)，但是至少可以在浏览器的地址栏中看到新相册的 ID。

工作原理

这个“试一试”练习首先是添加 Picture 和 PhotoAlbum 表到数据库和 EF 关系图。这些表用于存储相册的数据和它们包含的图片。每个单独的图片都属于其 PhotoAlbumID(指向数据库中 PhotoAlbum 表的 Id 列)引用的一个 PhotoAlbum。Picture 表设计为只保存有关图片的数据；在后面将看到，实际的图片文件将存储到磁盘上。

为了让用户创建新的相册，可向页面中添加一个 DetailsView 控件。要使该控件可用于插入新相册，启用插入功能并设置 DefaultMode 为 Insert。这就强制控件进入插入模式，而不是默认的只读模式。然后，将 EntityDataSource 与用于将相册插入 PhotoAlbum 表中的 DetailsView 相关联。EntityDataSource 控件的代码如下所示：

```
<asp:EntityDataSource ID="EntityDataSource1" runat="server"
    ConnectionString="name=PlanetWroxEntities" EnableFlattening="False"
    DefaultContainerName="PlanetWroxEntities" EnableInsert="True"
    EntitySetName="PhotoAlbums" OnInserted="EntityDataSource1_Inserted">
</asp:EntityDataSource>
```

如果使用 VB.NET，那么代码就不设置 OnInserted 特性。可注意到，这一情况中的 EntityDataSource 很简单：将它指向一个 DefaultContainerName，本例中是 PlanetWroxEntities，这是获取它的数据的主要入口。也可以将 EnableInsert 设置为 True 启用插入功能。另外，设置 EntitySetName，控件可知道使用 EF 关系图中的什么对象。对于简单的插入而言，这就是需要做的全部工作。当页面在浏览器中加载时，DetailsView 呈现一个用户界面，允许输入相册的新名称。在单击 Insert 时，输入的数据被汇集并转发给 EntityDataSource。然后该控件创建一个新的 PhotoAlbum 实例，通过向数据库发送合适的 INSERT SQL 语句将它保存到数据库中。

在许多情况下，这一标准行为是不够的。您需要验证输入数据的有效性或是在将数据发送到数据库之前对实际数据作更改。在后一个“试一试”练习中将图像上传到服务器时将看到后者中的一个示例。

另一个常见的需求是检索新创建项的 ID，它将被发送到下一个页面。本示例使用下列代码来实现：

VB.NET

```
Dim myPhotoAlbum As PhotoAlbum = CType(e.Entity, PhotoAlbum)
Response.Redirect(String.Format("ManagePhotoAlbum.aspx?PhotoAlbumId={0}",
    myPhotoAlbum.Id.ToString()))
```


C#

```
PhotoAlbum myPhotoAlbum = (PhotoAlbum)e.Entity;  
Response.Redirect(string.Format("ManagePhotoAlbum.aspx?PhotoAlbumId={0}",  
                                myPhotoAlbum.Id.ToString()));
```

EntityDataSource 控件的绝妙之处在于它使用强类型对象，其中类型映射到添加至模型关系图中的表。在这里，使用的是 PhotoAlbum 的真正实例，这个类表示 Web 站点中的相册。这样就可以检索在数据源控件的 Inserted 事件中在数据库中插入的相册。e 参数提供了一个名为 Entity 的属性，它包含了对新相册的引用。只要将它强制转换为真正的 PhotoAlbum(使用 VB.NET 中的 CType，或是在 C#中使用小括号括住类名并放在它的前面)，就可以访问 PhotoAlbum 的属性，包括它的由数据库生成(通过 ID 列上的 Identity 设置)的新的 ID，然后存储在 PhotoAlbum 的 Id 属性中。事件处理程序中的最后一行将用户带到下一个页面，并在查询字符串中发送新相册的 ID。

注意，如果没有填写名称字段，但是却单击了 Insert，那么会得到一个错误。第 13 章显示了如何修改 DetailsView 来向其模板中插入验证控件。

现在，可以插入新相册了，下一步自然是添加图片到相册。在下一个“试一试”练习中，将看到如何创建一个带有 ListView 控件的用户界面，允许用户上传新图片到相册中。

2. ListView 控件

到现在为止，您已经了解了一些数据绑定控件的运用，如 GridView，它是非常强大的，它支持数据的更新、删除、排序和分页，但不能插入数据且生成了大量的 HTML 标记。另外还了解了 Repeater 控件，它可对生成的 HTML 作精确的控制，但缺乏其他数据控件所拥有的高级功能，如更新和删除行为、排序和筛选功能。最后还了解了 DetailsView 控件，它允许一次插入或更新一条记录。

ListView 是“最好的”控件，它结合了 GridView 丰富的功能集和 Repeater 对标记的控制功能，并且还具有 DetailsView 的插入行为。ListView 使得可以以不同的格式显示数据，包括网格(像 GridView 那样的行和列)、项目符号列表(类似于本章前面建立 Repeater 的方式)、流格式(其中所有项一个接一个地放在 HTML 中，您可编写一些 CSS 对其进行格式化)。

ListView 通过模板(允许控制 ListView 对其底层数据提供的许多不同的视图)显示和管理其数据。表 14-6 列出了所有可添加为页面标记中 ListView 控件的直接子元素的所有可用模板。

表 14-6

模 板	说 明
<LayoutTemplate>	作为控件的容器。它使得可定义一个放置单独数据项的位置。然后通过 ItemTemplate 和 AlternatingItemTemplate 表示的数据项作为该容器的子元素添加
<ItemTemplate> <AlternatingItemTemplate>	定义控件的只读模式。当一起使用时，它们可以创建一种“斑马纹效果”，其中奇偶行有着不同的外观(通常是不同的背景色)
<SelectedItemTemplate>	允许定义当前活动的或选择的项的外观
<InsertItemTemplate> <EditItemTemplate>	这两个模板允许定义用于插入和更新列表中的项的用户界面。通常，放置文本框、下拉列表和其他服务器控件等到这些模板中，将它们与底层数据源绑定
<ItemSeparatorTemplate>	定义放置在列表中项之间的标记。可用于在项之间添加线、图像或其他标记
<EmptyDataTemplate>	在控件中没有数据显示时显示。可以添加文本或其他标记和控件，告诉用户无数据显示

(续表)

模 板	说 明
<GroupTemplate>	在高级表现场景中使用，其中数据可呈现在不同的组中
<GroupSeparatorTemplate>	
<EmptyItemTemplate>	

尽管这些模板看上去让人觉得需要编写大量代码来使用 ListView，但并非总是如此。首先，Visual Web Developer 2010 根据一些控件(如 EntityDataSource)提供的数据，创建了大部分的代码。其次，并不总是需要所有模板，这就可以最小化控件所需的代码。

除了许多的模板外，ListView 控件还有表 14-7 所示的属性，可通过对其进行设置来影响控件行为。

表 14-7

属 性	说 明
ItemPlaceholderID	放置在 LayoutTemplate 中的服务器端控件的 ID。当该属性引用的控件在屏幕上显示时，将由所有重复的数据项取代。它可以是一个真正的服务器控件，如<asp:PlaceHolder>; 或是一个简单的 HTML 元素，带有一个有效的 ID，其 runat 特性设置为 server(例如<ul runat= " server " id= " MainList " >)。如果没有设置该属性，ASP.NET 会尝试找到 ID 为 itemPlaceholder 的控件并使用该控件
DataSourceID	页面上数据源控件的 ID，如 EntityDataSource 或 SqlDataSource 控件
InsertItemPosition	这一属性的枚举包括 3 个值(None、FirstItem 和 LastItem)，用于确定 InsertItemTemplate 的位置：在列表的开始或末尾，或者不可见

和其他数据绑定控件一样，ListView 也有大量的在控件生命周期的特定时间触发的事件。例如，它有在项插入到底层数据源前后触发的 ItemInserting 和 ItemInserted 事件。类似地，它还有在更新和删除数据前后触发的事件。在第 16 章中将看到处理这类事件的更多内容。

除了刚才所介绍的模板、属性和事件外，ListView 还提供了很多内容。要了解该控件及其成员和行为的详细内容，可参阅 MSDN 文档，网址为：<http://tinyurl.com/mupjot>。

下面的“试一试”练习将介绍如何将这些信息组合到一起。了解如何定义不同的模板和设置相关的属性来控制 ListView 控件的外观。

试一试

用 ListView 控件插入和更新数据

用 ListView 控件插入数据项的操作和 DetailsView 一样简单：将控件指向数据源，然后让 VWD 创建所需的模板。不过，在大多数实际 Web 站点中，这些默认模板并不足够好。可能您想显示的字段比数据源中可用的要少，或是想要在将它们发送至数据库之前进行验证，或是想将数据存储到数据库之外的地方。例如，您可能想将上传的图像存储到磁盘而不是数据库中，然后在数据库表中存储到该文件的引用。因此，下一个“试一试”练习将在构建上传图片到相册的功能时，演示如何自定义 ListView 模板以及处理 EntityDataSource 的 Inserting 事件。

这一练习可让您处理 VWD 自动生成的大量代码。所需做的大部分工作是删除代码而不是添加新代码。如果在某处无所适从或是感觉代码不对劲，可参阅随本章提供的完整源代码(从 Wrox 网站下载)。

(1) 在 Web 站点的根文件夹下，基于自定义模板创建一个名为 ManagePhotoAlbum.aspx 的新 Web 窗体。将其 Title 设置为 Manage Photo Album 并切换至 Design 视图。

(2) 从 Toolbox 中拖动一个 ListView 控件到页面的 cpMainContent 占位符中，然后通过从 Smart Tasks 面板的 Choose Data Source 下拉列表中选择<New data source>(和对 DetailsView 所做的一样)，将它与 EntityDataSource 控件关联。单击 Entity 图标，然后单击 OK 按钮，选择 PlanetWroxEntities 作为命名连接，然后单击 Next 按钮。

在 EntityDataSource 控件的向导的 Configure Data Selection 对话框中，从 EntitySetName 下拉列表中选择 Pictures，如图 14-11 所示。

(3) 选中屏幕底部 3 个复选框中的第一个和最后一个，以启用 EntityDataSource 的插入和删除操作。最后，单击 Finish 按钮关闭 Configure Data Source Wizard。

(4) 回到 Design 视图，选择 EntityDataSource 控件，打开其 Properties 面板，找到 Where 属性，并通过单击该属性的省略号按钮打开其 Expression Editor 对话框。您可能还记得，页面 ManagePhotoAlbum.aspx 通过查询字符串接收相册 ID，所以在这一步将建立一个 QueryStringParameter 把 ListView 筛选为属于指定相册的图片。单击 Add Parameter 按钮，输入名称 PhotoAlbumId，然后按照图 14-13 所示填写 Expression Editor 对话框。

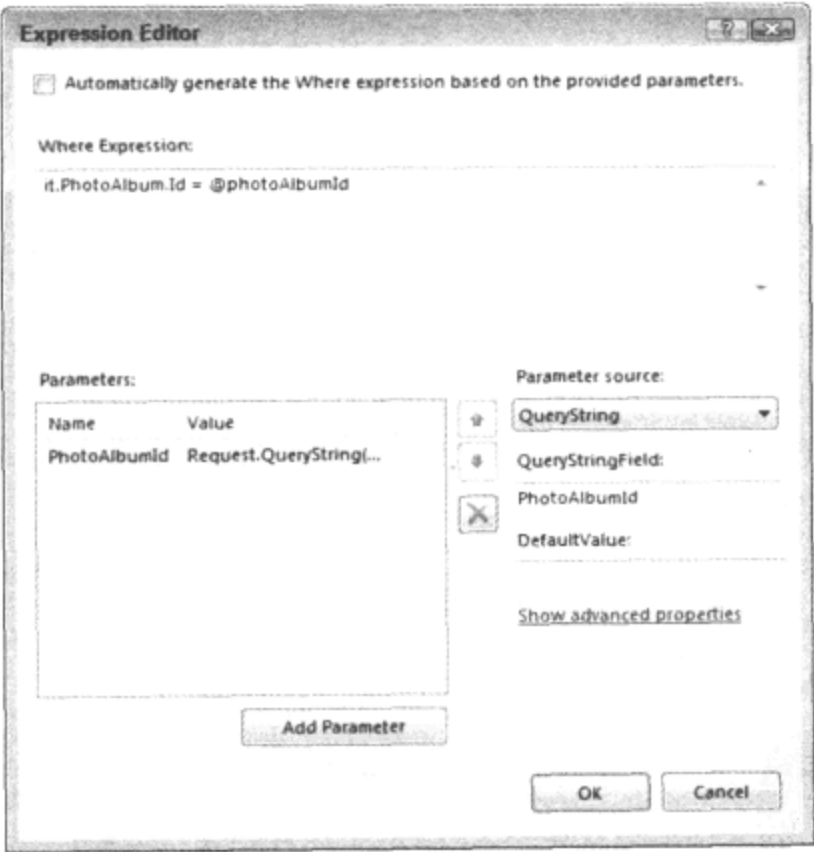


图 14-13

确保在对话框顶部的 Where Expression 文本框中输入 it.PhotoAlbum.Id = @photoAlbumId。接下来，单击 Show advanced properties 链接，然后将参数的 Type 属性改为 Int32。完成以后，单击 OK 按钮关闭对话框。

(5) 回到页面中，ListView 应该显示为一个普通的矩形，如图 14-14 所示，因为还没有提供任何

模板信息。

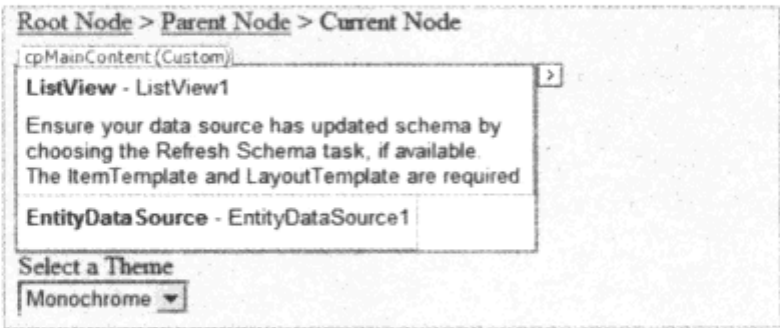


图 14-14

(6) 在 ListView 的 Smart Tasks 面板上选择 Configure ListView。(如果没有看到此链接，则首先单击 Refresh Schema，然后重新打开 Smart Tasks 面板)。出现的对话框允许选择控件的布局、样式和是否启用插入和更新等操作。选择 Bulleted List 作为布局，选中 Enable Inserting 和 Enable Deleting 选项，这时对话框如图 14-15 所示。

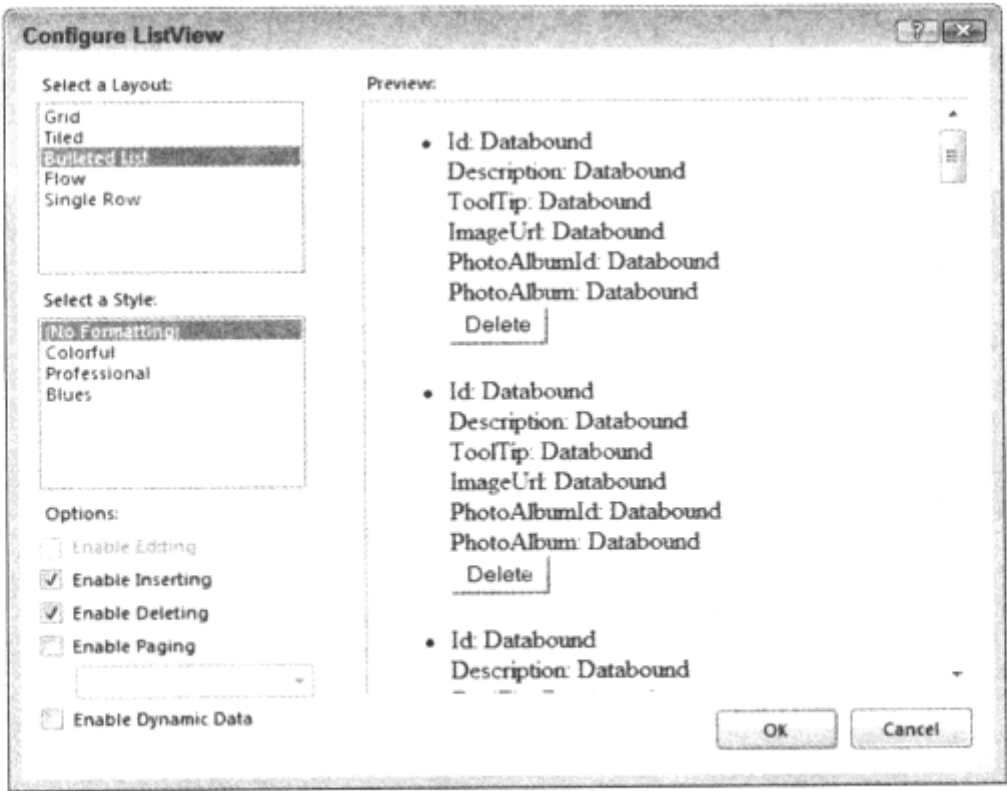


图 14-15

(7) 单击 OK 按钮关闭该对话框。如果有对话框询问是否想重新生成 ListView 控件，则单击 Yes 按钮。

(8) 切换至 Markup 视图并删除下面列出的模板的代码。简单的做法是，单击相关的起始标记，再单击文档窗口底部的标记选择器中的标记，选择整个元素及其内容，然后按 Delete 键。或者，可以使用左侧的加号(+)符号折叠标记，然后选择整行并一次性删除它。

- <AlternatingItemTemplate>
- <EditItemTemplate>
- <EmptyDataTemplate>
- <ItemSeparatorTemplate>
- <SelectedItemTemplate>

(9) 定位到 LayoutTemplate 中的元素并删除它的 ID、runat 和 style 特性。然后添加一个 class

特性并设置其为 ItemContainer。也可以删除 VWD 添加到下面的空<div>元素。<LayoutTemplate> 现在包含下列代码：

```
<LayoutTemplate>
  <ul class="ItemContainer">
    <li runat="server" id="itemPlaceholder" />
  </ul>
</LayoutTemplate>
```

(10) 定位到 ItemTemplate 并删除创建 Id、PhotoAlbumId 和 PhotoAlbum 列的代码行(下列代码段中突出显示的代码)，因为并不需要它们。不过要确保未删除起始标记：

```
<li style="">
  Id:
  <asp:Label ID="IdLabel" runat="server" Text='<%# Eval("Id") %>' />
  <br />
  Description:
  <asp:Label ID="DescriptionLabel" runat="server"
    Text='<%# Eval("Description") %>' />
  <br />
  ToolTip:
  <asp:Label ID="ToolTipLabel" runat="server"
    Text='<%# Eval("ToolTip") %>' /><br />
  ImageUrl:
  <asp:Label ID="ImageUrlLabel" runat="server"
    Text='<%# Eval("ImageUrl") %>' /><br />
  PhotoAlbum.Id:
  <asp:Label ID="PhotoAlbum_IdLabel" runat="server"
    Text='<%# Eval("PhotoAlbum.Id") %>' />
  <br />
  PhotoAlbum:
  <asp:Label ID="PhotoAlbumLabel" runat="server"
    Text='<%# Eval("PhotoAlbum") %>' />
  <br />
  <asp:Button ID="DeleteButton" runat="server" CommandName="Delete"
    Text="Delete" />
</li>
```

(11) 对同样是 ListView 控件一部分的 InsertItemTemplate 重复第(10)步。

比较您的代码和下面的代码，并在必要时修改您的代码。检查 ListView 是否有一个设为 Id 的 DataKeyNames 属性，如果没有则添加它。VWD 有时不会添加这个属性，但是代码却要求具有这个属性。另外，检查模板中是否包含正确的控件。在您的代码中，模板的顺序或空白区域可能有所不同。

```
<asp:ListView ID="ListView1" runat="server" DataKeyNames="Id"
  DataSourceID="EntityDataSource1" InsertItemPosition="LastItem">
  <InsertItemTemplate>
    <li style="">
      Description:
      <asp:TextBox ID="DescriptionTextBox" runat="server"
        Text='<%# Bind("Description") %>' /><br />
```

```

        ToolTip:
        <asp:TextBox ID="ToolTipTextBox" runat="server"
            Text='<%# Bind("ToolTip") %>' /><br />
        ImageUrl:
        <asp:TextBox ID="ImageUrlTextBox" runat="server"
            Text='<%# Bind("ImageUrl") %>' /><br />
        <asp:Button ID="InsertButton" runat="server"
            CommandName="Insert" Text="Insert" />
        <asp:Button ID="CancelButton" runat="server"
            CommandName="Cancel" Text="Clear" />
    </li>
</InsertItemTemplate>
<ItemTemplate>
    <li style="">Description:
        <asp:Label ID="DescriptionLabel" runat="server"
            Text='<%# Eval("Description") %>' /><br />
        ToolTip:
        <asp:Label ID="ToolTipLabel" runat="server"
            Text='<%# Eval("ToolTip") %>' /><br />
        ImageUrl:
        <asp:Label ID="ImageUrlLabel" runat="server"
            Text='<%# Eval("ImageUrl") %>' /><br />
        <asp:Button ID="DeleteButton" runat="server"
            CommandName="Delete" Text="Delete" />
    </li>
</ItemTemplate>
<LayoutTemplate>
    <ul class="ItemContainer">
        <li ID="itemPlaceholder" runat="server" />
    </ul>
</LayoutTemplate>
</asp:ListView>
```

(12) 切换回 Design 视图，选中 EntityDataSource 控件并打开其 Properties 面板。切换至 Events 选项卡并双击 Inserting 事件。像在页面 All.aspx 中执行的操作一样，在页面的顶部为 PlanetWroxModel 名称空间添加一个 Imports 或 using 语句。然后在 VWD 添加的事件处理程序中，编写如下代码：

VB.NET

```
Protected Sub EntityDataSource1_Inserting(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.EntityDataSourceChangingEventArgs) _
    Handles EntityDataSource1.Inserting
    Dim photoAlbumId As Integer =
        Convert.ToInt32(Request.QueryString.Get("PhotoAlbumId"))
    Dim myPicture As Picture = CType(e.Entity, Picture)
    myPicture.PhotoAlbumId = photoAlbumId
End Sub
```

C#

```
protected void EntityDataSource1_Inserting(object sender,
    EntityDataSourceChangingEventArgs e
{
    int photoAlbumId = Convert.ToInt32(Request.QueryString.Get("PhotoAlbumId"));
    Picture myPicture = (Picture)e.Entity;
```

```
myPicture.PhotoAlbumId = photoAlbumId;
}
```

(13) 在 Styles 文件夹中创建一个新的样式表文件，并命名为 Styles.css。用下列代码取代文件中的已有 CSS。

```
.ItemContainer
{
    width: 600px;
    list-style-type: none;
    clear: both;
}

.ItemContainer li
{
    height: 300px;
    width: 200px;
    float: left;
}

.ItemContainer li img
{
    width: 180px;
    margin: 10px 20px 10px 0;
}
```

(14) 保存并关闭该文件。

(15) 从 MasterPages 文件夹中打开站点的 Frontend 母版页(不是 Management 母版页)。切换到 Design 视图，将 Styles.css 文件从 Solution Explorer 拖放到母版页上。这就确保了站点上的所有页面都有了对这个新样式表的引用，而不必考虑使用的主题。

(16) 保存所有更改，关闭所有打开的文件，然后在浏览器中请求 NewPhotoAlbum.aspx。确保不是打开了 ManagePhotoAlbum.aspx，因为它要求 NewPhotoAlbum.aspx 发送的查询字符串。为相册输入一个新名称并单击 Insert 按钮。您将被带到 ManagePhotoAlbum.aspx，在那里可上传新的图片。目前，您所能做的是输入图片的描述、工具提示和伪 URL(只是输入一些文本)；后面将介绍如何对此进行修改，使用户可以上传真正的图片到该 Web 站点。单击 Insert 按钮后，新的项出现在列表中，在 Insert 控件的旁边。在添加一些项后，您将注意到 Insert 控件移到了其他项的下一行，如图 14-16(在 Google Chrome 中显示的 Manage Photo Album 页面)所示。

(17) 对某一项单击 Delete 按钮，该项将自动从列表中删除。

(18) 如果您当前是在 Monochrome 主题下查看站点，则使用下拉列表切换至 DarkGrey。注意，布局稍微有些乱，因为没有足够的空间并排放置 3 个图像。调整的方法很简单。打开 DarkGrey 主题文件夹中的 DarkGrey.css 文件并添加下列 CSS 代码到文件的末尾。

```
.ItemContainer
{
    width: 400px;
}
```

这使得图片列表的宽度改为 400 像素，因此在这一主题中只能显示两列图片。

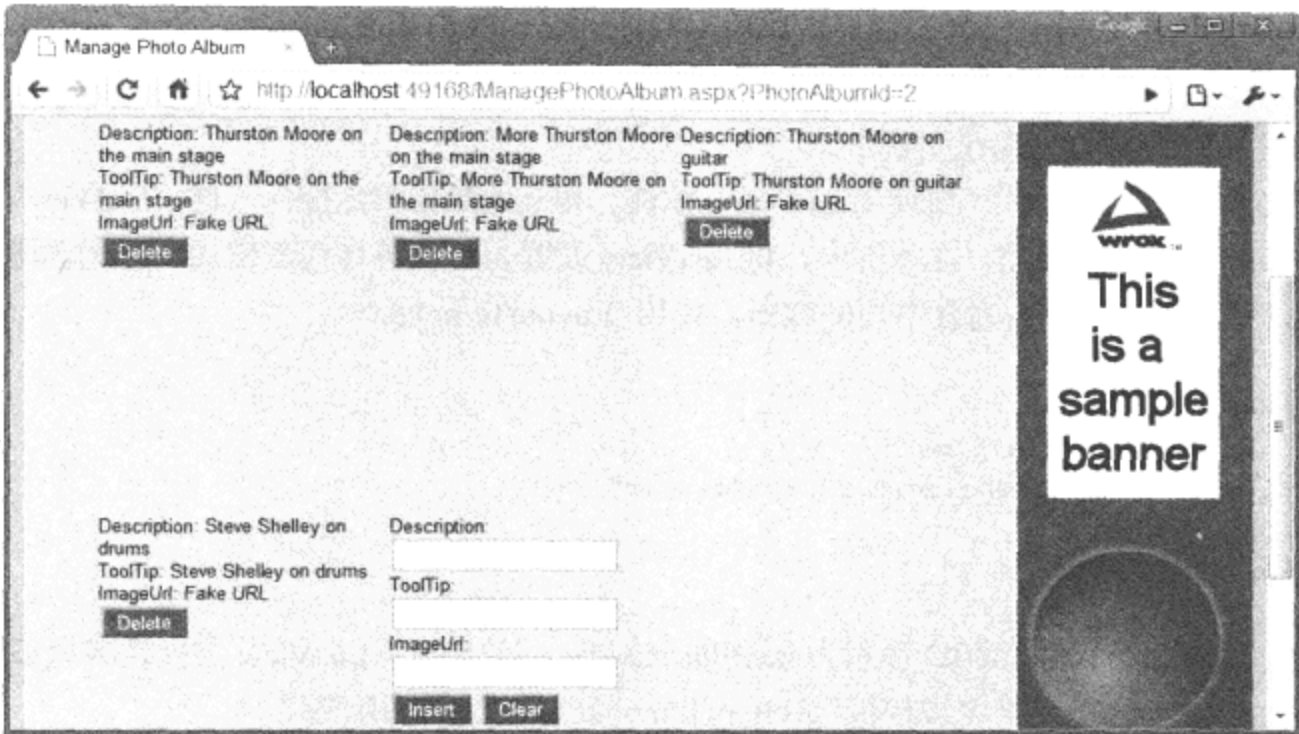


图 14-16

工作原理

这一练习首先是将 EntityDataSource 控件与 ListView 控件相关联。EntityDataSource 控件被配置为使用 Pictures 实体集。正如前面所见，每个图片都通过其 PhotoAlbumId 链接至一个相册。为了使 ManagePhotoAlbum.aspx 页面只显示属于当前相册(由 PhotoAlbumId 查询字符串标识)的那些图片，这里建立了一个 WhereParameter。

```
<asp:EntityDataSource ID="EntityDataSource1" runat="server"
    ConnectionString="name=PlanetWroxEntities"
    DefaultContainerName="PlanetWroxEntities" EnableDelete="True"
    EnableFlattening="False" EnableInsert="True" EntitySetName="Pictures"
    Where="it.PhotoAlbum.Id = @photoAlbumId"
    OnInserting="EntityDataSource1_Inserting">
    <WhereParameters>
        <asp:QueryStringParameter Name="PhotoAlbumId"
            QueryStringField="PhotoAlbumId" Type="Int32" />
    </WhereParameters>
</asp:EntityDataSource>
```

要注意这一标记中的两个重要部分。首先，有一个 WhereParameters 集合，它包含了一个查看查询字符串中的 PhotoAlbumId 字段的 QueryStringParameter。当 EntityDataSource 准备获取其数据时，它从查询字符串中检索参数的值。

第二个重要部分是 EntityDataSource 控件的 Where 特性。它使用 Where 子句限制从数据库返回的项：

```
Where="it.PhotoAlbum.Id = @photoAlbumId"
```

这将获取 Picture 表中具有请求的 PhotoAlbumId 的所有 Pictures。Where 子句中的“it”是一个隐式范围变量，和前面在查询中看到的其他范围变量一样。但是，在这里需要使用“it”，而不能像使用自己的 LINQ 查询时那样创建自己的名称。在运行时，将使用从查询字符串中检索出的实际

PhotoAlbumId 填充 Where 子句，这样可以确保只返回属于当前相册的图片。

在创建新相册后首次加载页面时，其中没有图片。不过，使用 ListView 控件的 InsertTemplate 开始添加项后，它们就会出现在列表中。

为了在页面上显示图片，要使用 ListView 控件。和其他数据绑定控件一样，ListView 可以以结构化的方式显示重复的数据。在本例中，将 ListView 设置为项目符号列表模式，因此它将数据显示为一组元素。为了定义列表中项的容器，使用<LayoutTemplate>。

```
<LayoutTemplate>
  <ul class="ItemContainer">
    <li ID="itemPlaceholder" runat="server" />
  </ul>
</LayoutTemplate>
```

注意，这个元素将其 ID 设置为 itemPlaceholder。这告诉了 ListView 控件在哪里添加单独的图片。在运行时，这一元素将被模板(如<ItemTemplate>)的实际项所取代。

当 ListView 控件需要显示其数据时，它将基于<ItemTemplate>为数据源中的每个数据项创建项。在本例中，每个数据项都是强类型的 Picture 对象，允许访问像 Tooltip 和 Description 这样的属性。

```
<ItemTemplate>
  <li>
    ...
    Tooltip:
    <asp:Label ID="ToolTipLabel" runat="server"
      Text='<%# Eval("ToolTip") %>' />
    ...
    <asp:Button ID="DeleteButton" runat="server" CommandName="Delete" />
  </li>
</ItemTemplate>
```

有了这一代码，数据源中的每一项显示为一系列标签，这些标签显示了图片的相关属性。Eval(PropertyName)用于从对象中检索请求的值，然后显示为 Label 控件的文本。注意，在这一阶段，<ItemTemplate>只显示有关图片的数据。后面将介绍如何上传和显示真正的图片。

注意一下 ItemTemplate 中 Button 控件的 CommandName。它被设置为 Delete，这使得该按钮成为了一个真正的 Delete 按钮。如果单击它，ListView 会判断出单击该按钮针对的是哪个图片，然后命令 EntityDataSource 控件从数据库中删除相关的图片。

添加到 Styles.css 的 CSS 以组织化的方式显示项。将控件的 class 特性设置为 ItemContainer，下列 CSS 就会应用到该列表。

```
.ItemContainer
{
  width: 600px;
  list-style-type: none;
  clear: both;
}
```

第一个声明设置列表整个宽度为 600 像素，而第二个声明从列表的项中删除项目符号。然后，列表中的每项显示在元素中，对此应用下列 CSS：

```
.ItemContainer li
{
    height: 300px;
    width: 200px;
    float: left;
}
```

每个项都变成了 200 像素的强制宽度。float 属性让每个元素互相紧挨着浮动。在 600 像素的父区域内，可以放置 3 个 200 像素的元素，使第 4 个和更多的元素放置在其自己的行中。相比于用 HTML 表显示数据来说，这是很好的替代方法，前者通常需要大量的标记来达到同样的效果。

最后，元素中的每个图像都变成 180 像素的强制宽度，上下页边距为 10 像素，右页边距为 20 像素(在图像间有了一些空间)，左页边距为 0。

```
.ItemContainer li img
{
    width: 180px;
    margin: 10px 20px 10px 0;
}
```

在 DarkGrey 主题中，ItemContainer 的 width 被重写为 400 像素。这样，<div>只可并排显示两个图像。

和其他许多数据绑定控件不同的是，ListView 还支持插入。这是通过定义一个 InsertItemTemplate 实现的，它包含一个或多个绑定到底层对象中的属性的控件。例如，图片的 Description 属性像下面这样进行绑定：

```
<InsertItemTemplate>
    <li>
        Description:
        <asp:TextBox ID="DescriptionTextBox" runat="server"
            Text='<%# Bind("Description") %>' /><br />
        ...
    </InsertItemTemplate>
```

这段代码没有使用 Eval(Property Name)，而是使用了 Bind(Property Name)建立了双向绑定机制。这确保了 ASP.NET 运行库能够判断 Picture 的 Description 属性和 DescriptionTextBox 文本框之间的关系，即使在回发后也是。因此，在输入一些详细内容并单击其 Insert 按钮(其 CommandName 设置为 Insert)后，就构造了新的 Picture 对象，其属性(如 Description、Title 和 ToolTip)用 InsertItemTemplate 中的相关服务器控件的值填充。然后该图片实例被转发给 EntityDataSource 控件，该控件负责在数据库中保存该项，并刷新显示在页面上的图片列表。

当 EntityDataSource 控件即将保存图片时，会触发其 Inserting 事件。在本“试一试”练习的事件处理程序中，将添加一些代码，像下面这样将新的 Picture 实例与 PhotoAlbumId 联系起来：

```
VB.NET
Dim photoAlbumId As Integer =
    Convert.ToInt32(Request.QueryString.Get("PhotoAlbumId"))
```



```
Dim myPicture As Picture = CType(e.Entity, Picture)
myPicture.PhotoAlbumId = photoAlbumId
```

C#

```
int photoAlbumId = Convert.ToInt32(Request.QueryString.Get("PhotoAlbumId"));
Picture myPicture = (Picture)e.Entity;
myPicture.PhotoAlbumId = photoAlbumId;
```

在 Entity Framework 之前的版本中,这种代码并不简单。使用那些版本时,不能设置 PhotoAlbumId,而是需要使用一些晦涩的代码进行检索然后分配整个 PhotoAlbum 属性。幸运的是,在本章前面启用的 Include Foreign Key Columns 选项提供了 Picture 类的 PhotoAlbumId 属性,通过该属性可以直接设置 PhotoAlbum(从查询字符串中检索得到)的 ID。这又会使数据库中的 Picture 与 PhotoAlbum 表中的特定相册关联起来。

显然,让用户直接输入图像的 ImageUrl 不是非常友好。更简单的做法是让用户从本地计算机中选择图像,然后上传到服务器。在下一个“试一试”练习中将介绍如何实现这种功能。

试一试

自定义 ListView 控件的模板

VWD 根据 EntityDataSource 中的信息生成的 ListView 控件的默认模板只适合于最为简单的情况。通常,您需要更多的控制。例如,在 ItemTemplate 中,您可能想显示实际的 Image 控件而非作为文本的普通 ImageUrl 属性。同样,在 InsertItemTemplate 中,您可能想显示文件上传控件而不是一个简单的文本框。在这个练习中,将介绍如何修改标准模板来结合这两种功能。另外,将介绍如何处理 EntityDataSource 控件的 Inserting 事件,将上传文件保存到磁盘,用图像的 URL 更新数据库。

对于这一示例来说,Web 服务器使用的账户(如果使用随 VWD 提供的内置 Web 服务器,就是用于登录计算机的账户)需要有对 GigPics 文件夹(本练习中将创建)的读、写权限。可参阅 19.3.2 节“了解 IIS 中的安全性”,进行权限配置。

(1) 在站点的根文件夹下创建一个名为 GigPics 的新文件夹。这一文件夹将包含用户上传的乐队演出时的图片。

(2) 在 Markup 视图下打开 ManagePhotoAlbum.aspx 页面,并定位到<ItemTemplate>元素。删除显示 ImageUrl 的 Label,用一个 Image 控件取代它,将控件的 ImageUrl 设置为图片对象的 ImageUrl。

```
<asp:Image ID="ImageUrl" runat="server" ImageUrl='<%# Eval("ImageUrl") %>' />
```

删除显示在图像上方的文本 ImageUrl:。

(3) 为了让用户上传图像,需要用 FileUpload 控件取代用于 ImageUrl 属性的 TextBox。还需要再次删除文本 ImageUrl:。在 InsertItemTemplate 中完成这些工作。

```
<asp:TextBox ID="ToolTipTextBox" runat="server"
    Text='<%# Bind("ToolTip") %>' /><br />
<asp:FileUpload ID="FileUpload1" runat="server" />
<br />
<asp:Button ID="InsertButton" runat="server" CommandName="Insert" Text="Insert" />
```

注意,这里不需要将属性和控件绑定。因为上传的图像需要作特殊处理,所以要在页面的 Code

Behind 中编写一些代码，而不是依赖于内置的数据绑定功能。

(4) 将<InsertItemTemplate>中的 Cancel 按钮的 CausesValidation 属性设置为 False:

```
<asp:Button ID="CancelButton" runat="server" CommandName="Cancel" Text="Clear"
    CausesValidation="False" />
```

(5) 类似地，将<ItemTemplate>中的 Delete 按钮的 CausesValidation 属性也设置为 False。

(6) 按 F7 键切换至页面的 Code Behind，用下列代码扩展 Inserting 事件处理程序，将文件保存到磁盘，然后用新位置更新 Picture 实例的 ImageUrl 属性:

VB.NET

```
myPicture.PhotoAlbumId = photoAlbumId

Dim FileUpload1 As FileUpload =
    CType(ListView1.InsertItem.FindControl("FileUpload1"), FileUpload)
Dim virtualFolder As String = "~/GigPics/"
Dim physicalFolder As String = Server.MapPath(virtualFolder)
Dim fileName As String = Guid.NewGuid().ToString()
Dim extension As String = System.IO.Path.GetExtension(FileUpload1.FileName)

FileUpload1.SaveAs(System.IO.Path.Combine(physicalFolder, fileName + extension))
myPicture.ImageUrl = virtualFolder + fileName + extension
```

C#

```
myPicture.PhotoAlbumId = photoAlbumId;

FileUpload FileUpload1 =
    (FileUpload)ListView1.InsertItem.FindControl("FileUpload1");
string virtualFolder = "~/GigPics/";
string physicalFolder = Server.MapPath(virtualFolder);
string fileName = Guid.NewGuid().ToString();
string extension = System.IO.Path.GetExtension(FileUpload1.FileName);

FileUpload1.SaveAs(System.IO.Path.Combine(physicalFolder, fileName + extension));
myPicture.ImageUrl = virtualFolder + fileName + extension;
```

(7) 回到 Markup 视图并为 InsertItemTemplate 添加 3 个验证控件：两个 RequiredFieldValidator 控件与 Description 和 Tooltip 文本框关联，另一个是 CustomValidator，将其 ErrorMessage 设置为 Select a valid .jpg file，将其 ID 设为 cusVallImage。最后，设置 Description 文本框的 TextMode 属性为 MultiLine，并在 Insert 按钮之前输入一个换行符(
)。

最终的代码如下所示:

```
Description:
<asp:RequiredFieldValidator ID="reqDesc" ControlToValidate="DescriptionTextBox"
    runat="server" ErrorMessage="Enter a description." />
<asp:TextBox ID="DescriptionTextBox" runat="server" TextMode="MultiLine"
    Text='<%# Bind("Description") %>' /><br />
Tooltip:
<asp:RequiredFieldValidator ID="reqTooltip" ControlToValidate="ToolTipTextBox"
```

```
runat="server" ErrorMessage="Enter a tool tip." />
<asp:TextBox ID="ToolTipTextBox" runat="server"
    Text='<%# Bind("ToolTip") %>' /><br />
<asp:FileUpload ID="FileUpload1" runat="server" /><br />
<asp:CustomValidator ID="cusValImage" runat="server"
    ErrorMessage="Select a valid .jpg file." />
<br />
<asp:Button ID="InsertButton" runat="server" CommandName="Insert" Text="Insert" />
```

(8) 切换至 Design 视图,选择 ListView 控件,然后双击 Properties 面板的 Event 选项卡中的事件,为其 ItemInserting 事件建立一个事件处理程序。用下列代码完成该事件处理程序:

VB.NET

```
Protected Sub ListView1_ItemInserting(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.ListViewInsertEventArgs) _
    Handles ListView1.ItemInserting
    Dim FileUpload1 As FileUpload =
        CType(ListView1.InsertItem.FindControl("FileUpload1"), FileUpload)
    If Not FileUpload1.HasFile OrElse
        Not FileUpload1.FileName.ToLower().EndsWith(".jpg") Then
        Dim cusValImage As CustomValidator =
            CType(ListView1.InsertItem.FindControl("cusValImage"), CustomValidator)
        cusValImage.IsValid = False
        e.Cancel = True
    End If
End Sub
```

C#

```
protected void ListView1_ItemInserting(object sender, ListViewInsertEventArgs e)
{
    FileUpload FileUpload1 =
        (FileUpload)ListView1.InsertItem.FindControl("FileUpload1");
    if (!FileUpload1.HasFile || !FileUpload1.FileName.ToLower().EndsWith(".jpg"))
    {
        CustomValidator cusValImage =
            (CustomValidator)ListView1.InsertItem.FindControl("cusValImage");
        cusValImage.IsValid = false;
        e.Cancel = true;
    }
}
```

(9) 保存所有更改,然后在浏览器中请求 NewPhotoAlbum.aspx。为相册输入一个新的名称并单击 Insert 链接。通过输入一些描述和工具提示,从硬盘中选择.jpg 图片,然后单击 Insert 按钮,插入一些图片。然后输入另一图片的描述和工具提示,但使文件上传框为空。在单击 Insert 时,会出现错误消息,表明没有上传有效的 JPG 文件,如图 14-17(使用的是 Apple 的 Safari 浏览器)所示。

(10) 单击文件上传控件的 Browse 按钮,找到一个有效的.jpg 文件,然后再次单击 Insert 按钮。现在文件就成功上传了。

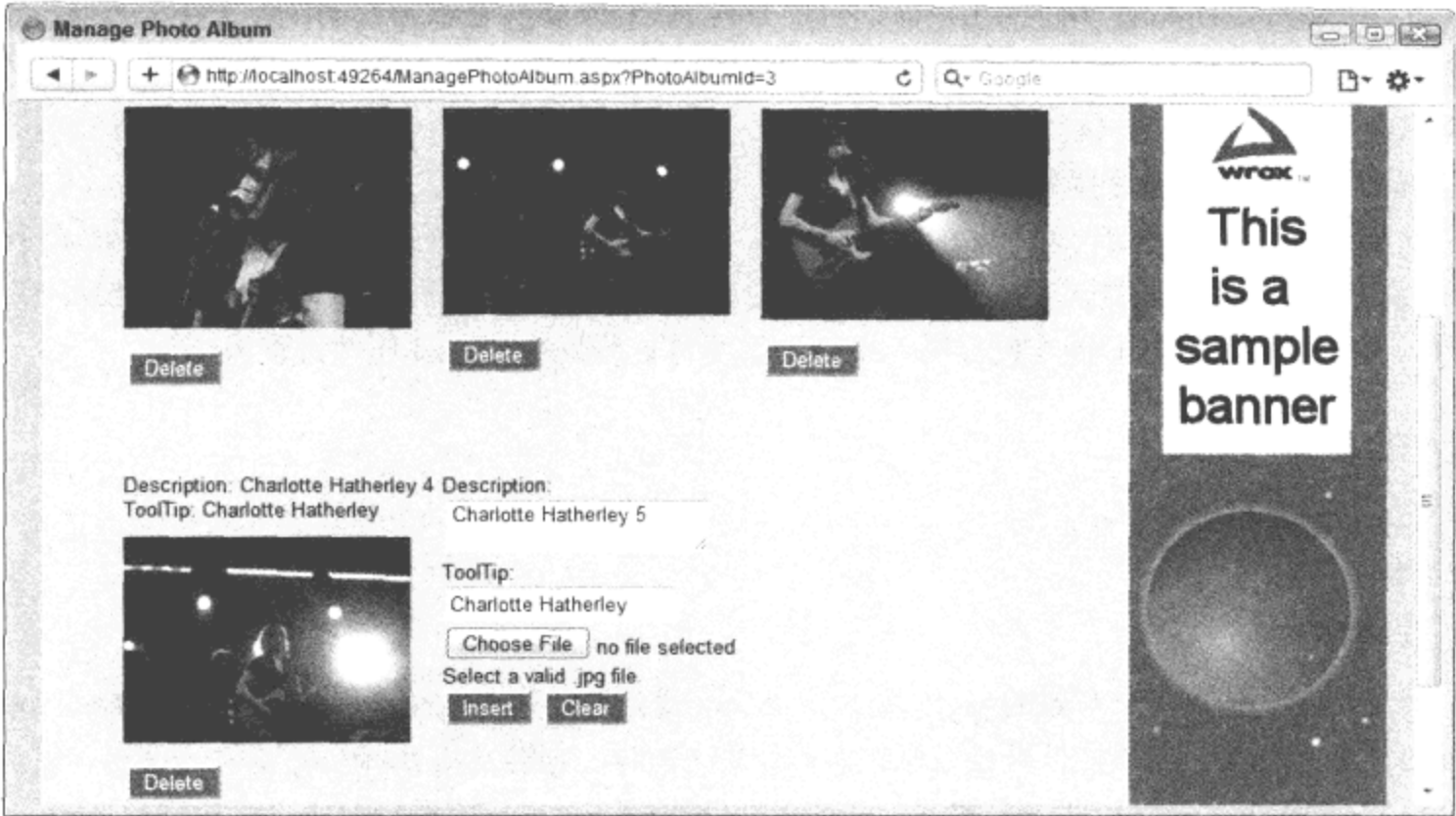


图 14-17

工作原理

在将 Picture 插入数据库的实际过程中并没有改变多少。ListView 控件仍然从页面中搜集所有相关数据，再发送到 EntityDataSource 控件，然后该控件通过 EF 将项插入到数据库中的 Picture 表。所不同的是建立模板和处理 EntityDataSource 和 ListView 控件的事件的方式。首先看一下模板。在 ItemTemplate 中，添加了一个<asp:Image>代替普通文本标签。正如图 14-17 所示，这显示了实际的图像，而不是其 URL。

为了允许用户上传图像，用 FileUpload 控件取代了 InsertItemTemplate 中的 TextBox 控件。另外，添加了一些验证控件，强制用户输入必要的字段。一旦按下 Insert 按钮，页面就回发，且 ListView 控件触发其 ItemInserting 事件。可以在这一事件中执行任何自定义的验证。这一事件处理程序接收的其中一个参数(e 参数)是 ListViewInsertEventArgs 类型的，这是一个提供了关于 ItemInserting 事件的上下文敏感的信息的类。在检测错误时，可以将这个 e 参数的 Cancel 属性设置为 True，来告诉 ListView 控件想取消插入操作。在这一事件处理程序中，添加一些代码来“发现”InsertItem 模板中的上传控件。因为可能会具有名称相同的多个控件(例如 InsertItemTemplate 和 EditItemTemplate 中的 FileUpload 控件)，所以不能直接访问 FileUpload1 控件，而是需要在 InsertItem 对象上使用 FindControl 来搜索该控件。

VB.NET

```
Dim FileUpload1 As FileUpload =
    CType(ListView1.InsertItem.FindControl("FileUpload1"), FileUpload)
```

C#

```
FileUpload FileUpload1 =
    (FileUpload)ListView1.InsertItem.FindControl("FileUpload1");
```

在获得对 FileUpload 控件的引用时，可检查其 HasFile 属性查看是否上传了文件。另外，可检查 FileUpload1.FileName.ToLower().EndsWith(".jpg ")查看是否上传了带有.jpg 扩展名的文件。为了确保只在用户上传文件时执行这一测试，如第 5 章中讲述的那样，代码使用 VB 中的 OrElse 和 C# 中的||来利用 If 语句中的短路逻辑。

如果用户没有上传有效文件，那么就运行 If 块中的代码。它再次使用 FindControl 找到 CustomValidator 控件，设置其 IsValid 属性为 False。这就告诉控件在页面呈现时显示其 ErrorMessage。最后，为了阻止 ListView 继续执行插入操作，需要将 e 参数的 Cancel 属性设置为 True。

```
VB.NET
e.Cancel = True
```

```
C#
e.Cancel = true;
```

如果用户上传了一个有效的.jpg 文件，ListView 就会继续其插入操作，最终对 EntityDataSource 控件实现插入动作。在该控件准备发送插入操作到 EF 时，它触发其 Inserting 事件，使您有最后的机会挂钩到(hook into)进程并查看数据。在该事件处理程序中，使用类似的代码查找对 InsertItem 模板中 FileUpload 控件的引用。然后使用下列代码确定该文件的物理和虚拟文件夹以及其名称和扩展名：

```
VB.NET
Dim virtualFolder As String = "~/GigPics/"
Dim physicalFolder As String = Server.MapPath(virtualFolder)
Dim fileName As String = Guid.NewGuid().ToString()
Dim extension As String = System.IO.Path.GetExtension(FileUpload1.FileName)
```

```
C#
string virtualFolder = "~/GigPics/";
string physicalFolder = Server.MapPath(virtualFolder);
string fileName = Guid.NewGuid().ToString();
string extension = System.IO.Path.GetExtension(FileUpload1.FileName);
```

变量 virtualFolder 保存存储了上传图像的文件夹的虚拟位置(从 Web 站点的根文件夹开始)。使用 Server.MapPath 可以将它转变成物理文件夹。假定项目被放置在默认位置 C:\BegASPNET\Site，那么 physicalFolder 变量将包含 C:\BegASPNET\Site\GigPics。

接着，使用 Guid.NewGuid()生成新的随机文件名。Guid 类生成的文件名可以保证在时间和空间上都具有唯一性。这就指派了类似于 f6d8cd05-2dbe-4aed-868a-de045f9462e3 这样的变量 fileName，确保了是唯一文件名。最后，使用 System.IO 名称空间中的 Path 类的 GetExtension 静态方法检索文件的扩展名。

这时，您已经有了将文件存储到磁盘的所有必需信息，然后更新数据库。使用 FileUpload 控件的 SaveAs 方法可以很容易地将文件存储到磁盘上。

```
VB.NET
FileUpload1.SaveAs(System.IO.Path.Combine(physicalFolder, fileName + extension))
```


C#

```
FileUpload1.SaveAs(System.IO.Path.Combine(physicalFolder, fileName + extension));
```

这一代码将物理文件夹、文件名和扩展名传递到 SaveAs 方法，该方法将文件保存到要求的位置。最后，Picture 实例用新的 ImageUrl 进行更新。

VB.NET

```
myPicture.ImageUrl = virtualFolder + fileName + extension
```

C#

```
myPicture.ImageUrl = virtualFolder + fileName + extension;
```

这指派了~/GigPics/f6d8ed05-2dbe-4aed-868a-de045f9462e3.jpg 这样的 URL 给 ImageUrl 属性，这是上传图像的新的虚拟位置。在插入新图像后，ListView 得到更新，通过使用 Image 控件并将其 ImageUrl 设置为刚上传的图像来显示新的图像。

可以想像，如果为单个相册上传很多图像，页面就很难管理。在前端，用户可能正通过一个缓慢的网络连接访问站点。因此不要将相册中的所有图像显示在一个页面上，可以将相册分开放到多个页面上，允许用户一页页地浏览。在下一节讨论 DataPager 控件时将介绍如何实现这种功能。

3. DataPager 控件

在 ASP.NET 的前几个版本中，分页只是通过一些控件(如 GridView 和 DetailsView)内置的功能实现或是通过手动编写代码实现。在 ASP.NET 3.5 中引入了 DataPager 控件后，这种情况就有了变化。

DataPager 与其他控件不同，它是个单独的控件，可用它来扩展另一个数据绑定控件。目前，.NET Framework 只允许使用 DataPager 为 ListView 控件提供分页功能，但开发社区正在积极地为其他控件(如 GridView)编写实现。

有两种方法将 DataPager 与 ListView 控件关联：可以在 ListView 控件的<LayoutTemplate>中定义它，或是完全在 ListView 的外部定义它。在第一种情况下，DataPager 知道应自动为给哪个控件提供分页功能。在第二种情况下，需要将 DataPager 的 PagedControlID 属性设置为有效 ListView 控件的 ID。下面将介绍如何结合 ListView 配置和使用 DataPager。如果想将 DataPager 控件放到页面不同的地方，例如 Footer 或 SideBar 区域，那么在 ListView 控件的外部定义它将很有用。

试一试

使用 ListView 和 DataPager 控件将数据分页

在这个练习中，创建 Gig Pics 功能的前端页面。到站点的用户可从下拉列表中选择一个可用相册，然后在可分页的列表(通过 ListView 和 DataPager 控件创建)中查看所有可用图片。图 14-20 显示了这一练习的最终结果。

(1) 在站点的根文件夹中，创建一个名为 PhotoAlbums 的新文件夹。在这个文件夹中创建一个新的 Web 窗体，命名为 Default.aspx，确保它基于自定义页面模板。将该页面的 Title 设置为 All Photo Albums。

(2) 切换至 Design 视图并拖放一个 DropDownList 控件到页面上。在该控件的 Smart Tasks 面板上启用 AutoPostBack, 然后通过单击 Choose Data Source, 从第一个下拉列表中选择<New data source> 将它与一个新的 EntityDataSource 控件相关联。单击 Entity 图标, 单击 OK 按钮, 然后确保在 Named Connection 下拉列表选择了 PlanetWroxEntities。单击 Next 按钮, 然后在 Configure Data Selection 对话框中, 从 EntitySetName 下拉列表中选择 PhotoAlbums, 然后选择 Id 和 Name 字段, 如图 14-18 所示。

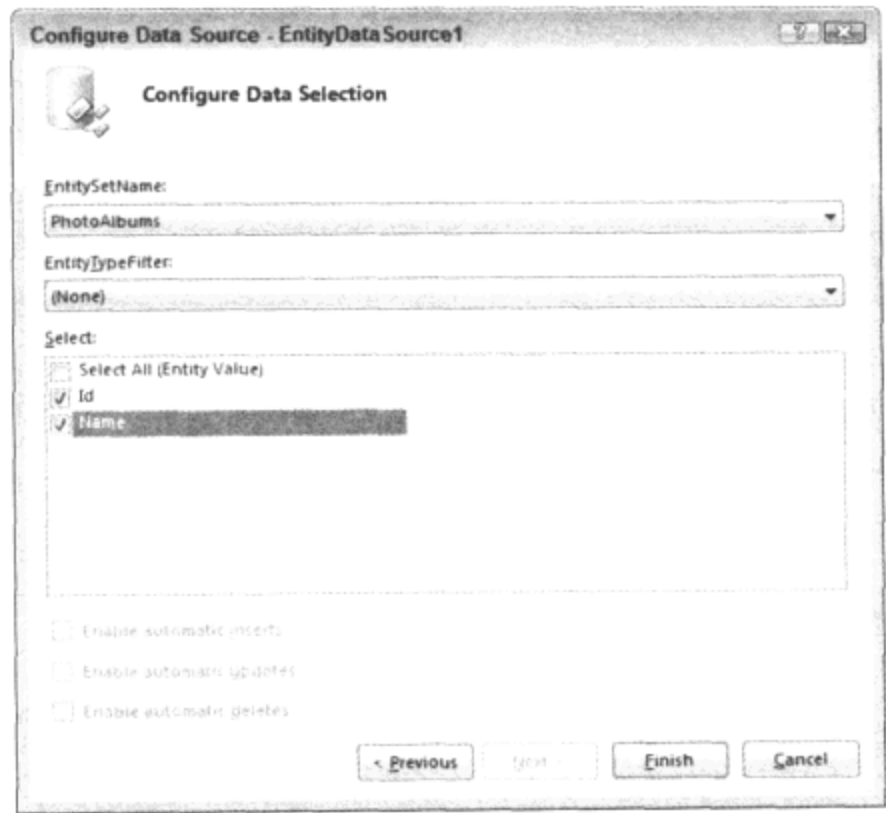


图 14-18

(3) 单击 Finish 按钮关闭 Data Source Wizard, 然后在 Choose Data Source 对话框中, 从 Data Field to Display 下拉列表中选择 Name, 从 Data Field for the Value 下拉列表中选择 Id。如果没有在列表中看到这些项, 则需要单击 Refresh Schema。

(4) 单击 OK 按钮关闭 Data Source Configuration 向导。

(5) 在 DropDownList 下添加一个新的 ListView 控件, 通过从该控件的 Smart Tasks 面板的下拉列表中选择<New data source>, 将它与一个新的 EntityDataSource 连接。单击 Entity 图标, 之后单击 OK 按钮。然后确保从下拉列表选择了 PlanetWroxEntities, 并单击 Next 按钮。在随后的对话框中, 从 EntitySetName 下拉列表中选择 Pictures, 然后单击 Finish 按钮。

(6) 在 Design 视图中选择新的 EntityDataSource 控件(EntityDataSource2), 打开其 Properties 面板, 然后 Where 属性的省略号按钮以打开 Expression Editor 对话框。取消选择对话框顶部的自动生成 Where 子句复选框。单击 Add Parameter 按钮, 并输入参数的名称 PhotoAlbumId。然后从 Parameter source 下拉属性中选择 Control, 并选择 DropDownList1 作为 ControlID。最后, 单击 Show Advanced Properties 链接, 将参数的 Type 改为 Int32。现在, Expression Editor 应该如图 14-19 所示。

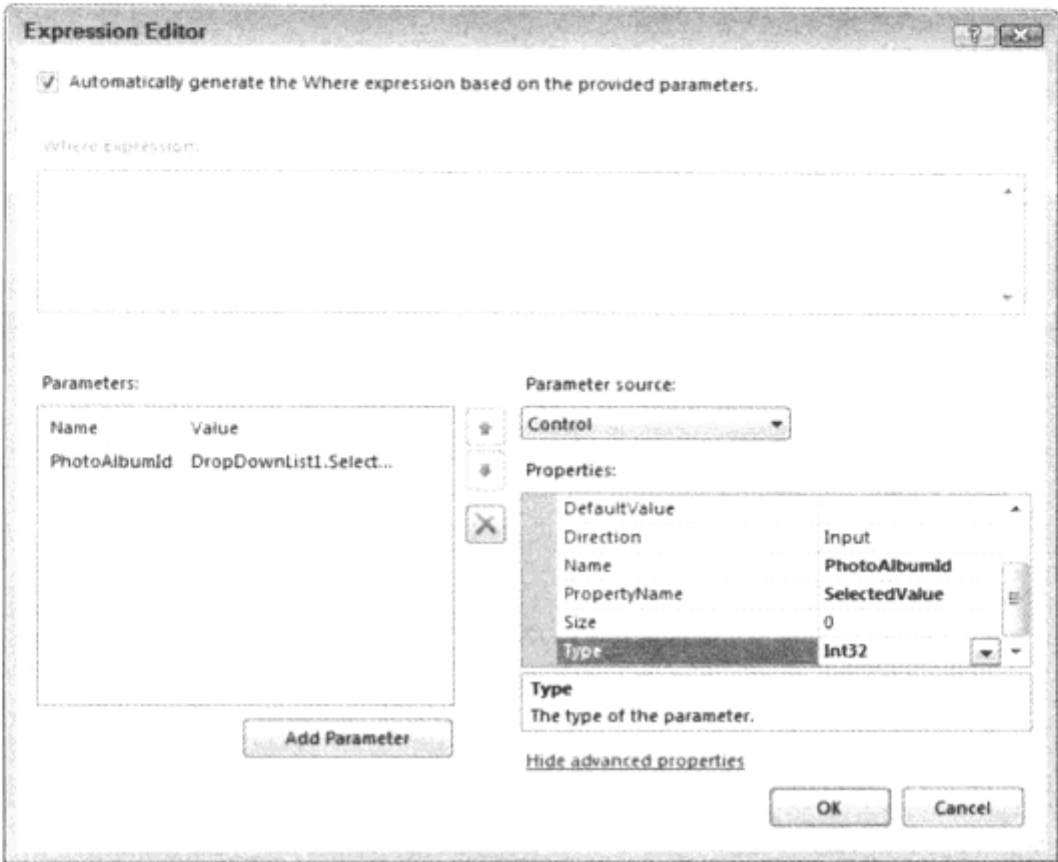


图 14-19

单击 OK 按钮关闭对话框。

(7) 在 ListView 控件的 Smart Tasks 面板上，单击 Configure ListView(如果没有看到该选项，则单击 Refresh Schema 并重新打开 Smart Tasks 面板)。选择 Bulleted List 作为布局，然后在 Options 区域中选择 Enable Paging。其下方的下拉列表默认为 Next/Previous Pager，这对于本练习来说很合适。

(8) 单击 OK 按钮，VWD 会创建一些模板。

(9) 切换至 Markup 视图，删除下列模板和它们包含的代码：

- <AlternatingItemTemplate>
- <EditItemTemplate>
- <InsertItemTemplate>
- <ItemSeparatorTemplate>
- <SelectedItemTemplate>

(10) 从 LayoutTemplate 中的中删除 ID、runat 和 style 特性，然后添加一个 class 特性并设置它为 ItemContainer。定位到 LayoutTemplate 中的 DataPager，然后添加一个 PageSize 特性并设置其值为 3。最后，向空的 style 特性中添加 clear: both;：

```
<LayoutTemplate>
  <ul class="ItemContainer">
    <li id="itemPlaceholder" runat="server" />
  </ul>
  <div style="clear: both;">
    <asp:DataPager ID="DataPager1" runat="server" PageSize="3">
      <Fields>
        <asp:NextPreviousPagerField ButtonType="Button" ShowFirstPageButton="True"
          ShowLastPageButton="True" />
      </Fields>
    </asp:DataPager>
  </div>
</LayoutTemplate>
```

```
</div>
</LayoutTemplate>
```

(11) 修改 ItemTemplate 中的代码，结果如下所示：

```
<ItemTemplate>
  <li>
    <asp:Image ID="Image1" runat="server" ImageUrl='<# Eval("ImageUrl") %>'
      ToolTip='<# Eval("ToolTip") %>' />
    <asp:Label ID="DescriptionLabel" runat="server"
      Text='<# Eval("Description") %>' />
  </li>
</ItemTemplate>
```

这就创建了一个 Image 控件，其 ImageUrl 和 ToolTip 属性绑定到了要绑定的 Picture 对象的相应属性。在浏览器中，将鼠标在图像上悬停时，会出现该 ToolTip。在图像下方，有一个简单的 Label 控件显示图像的 Description。在这个练习中，不需要其他一开始就在模板中定义的属性。

(12) 接着，用 ContentTemplate 元素将整个代码包装到一个 UpdatePanel 的 cpMainContent 内容块中，避免在对图像列表分页时，或是从列表中选择新相册时出现页面闪烁。

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" runat="Server">
  <asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
      <asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
        DataSourceID="EntityDataSource1" DataTextField="Name" DataValueField="Id">
        ...
      </ContentTemplate>
    </asp:UpdatePanel>
  </asp:Content>
```

(13) 打开站点的 Web.sitemap 文件，并为 Reviews 和 About 部分之间的 Gig Pics 部分添加一个主菜单和两个子菜单。

```
</siteMapNode>
<siteMapNode url="~/PhotoAlbums/" title="Gig Pics" description="All Gig Pics">
  <siteMapNode url="~/PhotoAlbums/Default.aspx" title="Gig Pics"
    description="All Gig Pics" />
  <siteMapNode url="~/NewPhotoAlbum.aspx" title="New Album"
    description="Create a new Photo Album with Gig Pics" />
</siteMapNode>
<siteMapNode url="~/About/Default.aspx" title="About"
  description="About this site">
```

(14) 因为添加了另外一个菜单项，所以需要改变 Monochrome 主题下的菜单中每一项的宽度。为此，打开 Monochrome.css，将.MainMenu ul li 选择器的宽度从 200 像素改为 160 像素：

```
.MainMenu ul li
{
  width: 160px;
}
```


(15) 保存所有更改，然后在浏览器中请求 PhotoAlbums 文件夹中的 Default.aspx。从下拉列表中选择 一个相册，然后页面重新加载，以显示该相册中的相关图片。



常见错误：如果从下拉列表中选择新项后什么都没有发生，则需要返回 VWD，确保将 DropDownList 控件的 AutoPostBack 设为 True。

如果相册中没有图片，或是没有足够的图片填充整个页面，从 Gig Pics 菜单中选择 New Album 命令，创建一个新的相册，在其中至少添加 4 个图像。然后单击 Gig Pics 菜单项，从下拉列表中选择新相册。注意，现在就有了一个分页用户界面，允许使用图 14-20 中屏幕底部所示的 First、Previous、Next 和 Last 按钮，向前或向后遍历相册中的图片列表。注意，由于添加了 AJAX 面板，选择和分页操作现在完全没有出现闪烁现象。

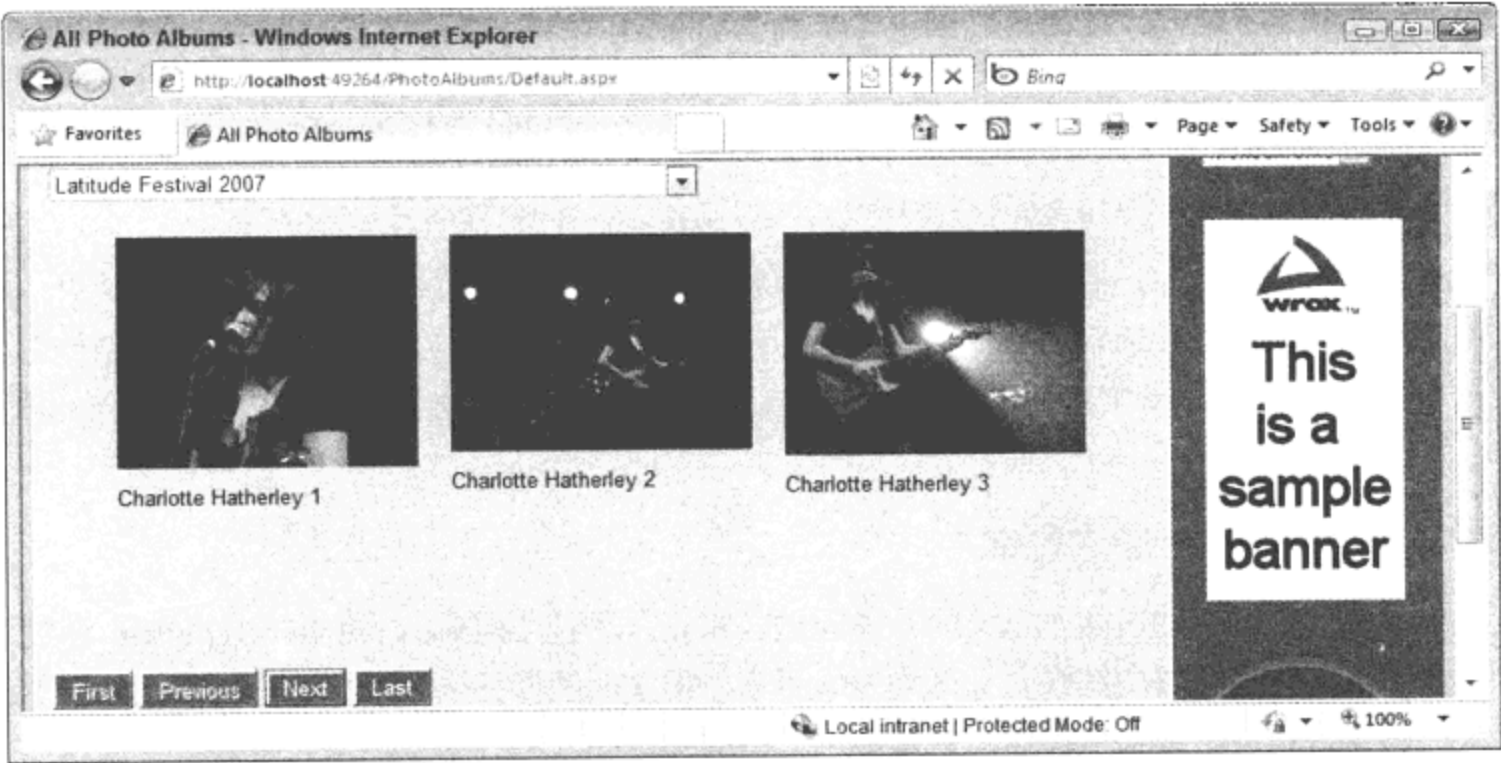


图 14-20



注意：您的前几个相册将会出现问题，因为在创建它们的时候并没有提供图像。如果想要进行清理，则可以从数据库中删除相册(及其相关的图片记录)。在第 16 章将学习如何开发从相册中删除图像的功能。

工作原理

您一定会对本练习中看到的大部分内容感到熟悉。类似于第 13 章中创建 Genres 下拉列表时那样，连接一个 DropDownList 到一个 EntityDataSource 控件。ListView 和其相关的 EntityDataSource 也类似于本章前面所讨论的内容。不过，现在代码使用的 Where 参数查看的是 DropDownList，而不是查询字符串。

```
<WhereParameters>
  <asp:ControlParameter ControlID="DropDownList1" Name="PhotoAlbumId"
    PropertyName="SelectedValue" Type="Int32" />
</WhereParameters>
```

因为在 EntityDataSource 控件上把 AutoGenerateWhereClause 设为 True，所以会基于这些参数快速创建一个 Where 子句。这与前面示例中显式定义 Where 子句不同。对于上一个“试一试”练习中的简单场景，使用自动生成的 Where 子句就可以了。但是对于更加复杂的场景，应该知道可以自己显式指派一个 Where 子句。

当 EntityDataSource 准备从数据库获取数据时，会查看下拉列表的 SelectedValue 属性，然后只检索与请求的 PhotoAlbumId 匹配的图片。

与前面示例最大的不同在于本示例添加了 DataPager。正如本练习演示的那样，分页是自动处理的。您所需要做的就是 在 ListView 的 LayoutTemplate 某处嵌入 DataPager 控件，其余都是自动完成的。如果在 ListView 的外部放置 DataPager，则不要忘了通过设置 PagedControlID 属性将它与 ListView 关联。如果您更喜欢使用链接或图像而非按钮，可以设置 NextPreviousPagerField 元素的 ButtonType 属性为 Link 或 Image。如果喜欢使用数字分页器，则用 NumericPagerField 取代 NextPreviousPagerField 项：

```
<asp:NumericPagerField NextPageText="..." PreviousPageText="..." />
```

在这个练习中，将 PageSize 设为 3，所以很容易填满多个页面，以便查看分页器的效果。在实际应用程序中，PageSize 通常要大一些，比如 10 或者 20。因为布局为 Monochrome 主题使用了一个 3 列布局，为 DarkGrey 主题使用了一个两列布局，所以可能需要选择一个被 2 和 3 整除的值，比如 18。

14.5.2 有关性能的一些注意点

对于本练习，需要知道两个重要的性能问题。首先，尽管 ListView 的 ItemTemplate 通过使用 CSS 设置宽度，使浏览器中图像尺寸改为 180 像素，但实际图像并未改变。这意味着，如果上传一个较大的图像，仍会下载整个图像，只是显示为小的缩略图。更好的做法是在服务器上创建一个真正的缩略图，然后将它发送到浏览器。笔者所编著的 ASP.NET 2.0 Instant Results 一书(ISBN: 978-0-471-74951-6)的 Greeting Cards 一章有大量有关调整服务器上图像大小的示例。

另一个潜在的性能问题是数据分页的方式。运用 DataPager 控件，数据可在 ASPX 页面内部分页。这意味着所有数据将从数据库中检索，然后发送到 EntityDataSource 控件。然后 DataPager 控件选择符合条件的记录并将其显示在当前页面上。这对于最多上百条记录的结果集工作良好。不过，一旦相册或其他集合中的项数超过此数，就会发现页面速度开始变慢。如果那样的话，可能需要考虑一下数据库级的分页。ADO.NET Entity Framework 使用前面讨论过的 Skip()和 Take()方法支持这一场景。

14.6 有关 LINQ 和 ADO.NET Entity Framework 的实用提示

下面列出了一些有关 LINQ 和 ADO.NET Entity Framework 的实用提示：

- 在本章中，介绍了如何创建匿名类型来定形从查询返回的数据。编译器和智能感知工具(IntelliSense)在确定返回的数据和可用的属性方面很有用。所以应该花些时间研究匿名类型，并了解智能感知列表所提供的不同选项。
- 和其他数据访问方法一样(像第 13 章所看到的 SqlDataSource 控件)，尽可能尝试筛选数据。如果您只需要 Jazz 流派中的评论，那么就显式地在代码中或 EntityDataSource 控件中添加 Where 子句，在数据库级限制评论列表。这样就加速了查询和数据检索，提高了应用程序的整体速度。
- 利用匿名类型降低 LINQ 查询的内存消耗。这样，不是检索整个 Review 对象，而是使用 New 关键字动态创建一个新的匿名类型。由于这个新对象只包含真正需要的属性，因而避免了检索完整的对象。

14.7 本章小结

LINQ 是 .NET 4 自带的一项引人注目的、令人兴奋的技术。它在许多数据访问场景中都是—种重要的管道技术，例如使用 ADO.NET Entity Framework 在 ASP.NET Web 应用程序中进行数据库访问。

由于 LINQ 十分重要，它已被集成到 .NET 中的多个不同的地方。LINQ 可用于可以查询内存中的集合的对象。另外，它还可用于 XML、Entities 和 DataSet，这些类型提供了对不同数据存储的访问，但使用相同的、统一的查询语言。LINQ 也用作 ADO.NET Entity Framework 的查询语言。

要在 ASP.NET Web 应用程序中使用 EF，有一些不同的选择。首先，在页面的 Code Behind 文件中编写查询，然后使用控件的 DataSource 属性和 DataBind 方法将结果与数据绑定控件绑定。或者，可以使用 EntityDataSource 控件，这个控件是数据绑定控件和模型之间的桥梁。当与新的 ListView 和 DataPager 控件结合使用时，EntityDataSource 可以用来创建功能全面的 CRUD 页面。

到目前为止，您所看到的数据库驱动的面很乏味。还没有应用任何样式或是提供一些条件格式，其中数据根据其值而作不同的呈现。这可以通过控件样式以及数据绑定和数据源控件的多种事件来实现。第 15 章介绍了如何使用这些样式和事件。

14.8 练习

1. 假定在 Reviews 文件夹中有一个名为 MostRecent.aspx 的页面。该页面显示了最近添加到数据库的 10 个评论。如果只想显示评论的 Title 属性和它所属的流派的名称，该使用怎样的 LINQ 查询？应该使用 Take 方法限制结果集为 10。如果编写获取最后一条评论的相关代码有困难，可参阅本章 14.4.1 节中“First、FirstOrDefault、Last 和 LastOrDefault”部分的内容，其中介绍了如何获取数据库中的最新评论。
2. 与其他数据控件(如 GridView 和 Repeater)相比，ListView 控件的主要优势在哪里？
3. 目前，PhotoAlbums 文件夹中的 Default.aspx 页面只显示了图片的缩略图。如何使用 LINQ 查询在其自己的页面上显示正常尺寸的图片？
4. 从 Photo Album 页面的 ListView 中删除图片时，将只删除数据库记录，而不会删除磁盘上的

图像。使用静态的 `System.IO.File.Delete` 方法可以从磁盘上删除该项。选择 `EntityDataSource` 的合适事件来处理这种情况，并使用可用的 `e.Entity`。

5. 目前，`AllByGenre.aspx` 页面显示了流派的标题，而不管它是否有评论。如何隐藏没有任何评论的流派？使用 `Reviews` 集合的 `Count` 方法来解决这个问题。

练习的答案见附录 A。

本章要点回顾

ADO.NET Entity Data Model 文件	包含将对象模型映射到数据库表的必要信息的文件
匿名类型	没有显式定义，而是动态创建的类型
Entity Framework	一种使用底层数据库创建强类型对象模型的技术，允许与数据库中的数据交互
实体集	实体模型中的对象集合。例如， <code>PhotoAlbum</code> 实例有一个包含相册中的图片的 <code>Pictures</code> 实体集
EntityDataSource 控件	用作 ASPX 页面和 Entity Framework 之间的桥梁的一个 ASP.NET 控件
延迟加载	直到在运行时需要访问数据的时候才从数据库中加载它们的一种技术
LINQ	语言集成查询；.NET Framework 编程语言中支持查询各种数据集合(包括对象、XML 和数据库)的那一部分
范围变量	LINQ 查询中定义的变量，将用在后面的 <code>select</code> 和 <code>where</code> 部分
强类型	一种编程概念，在声明变量时显式定义其类型
类型推理	编译器基于赋给变量的数据确定其类型的一种技术。通过使用这种技术，可以在没有显式定义变量类型的情况下创建强类型变量

本章要点

- 使用样式、主题和外观改变不同数据绑定控件的格式
- 使用数据控件触发的不同事件有条件地改变数据控件的外观
- 通过手动编写数据访问页面的 UI 的代码来完全控制页面的结构和标记
- 使用内置的缓存机制提高 Web 站点的性能

在前面 3 章，我们已经介绍了大量新的概念。第 12 章概述了数据库并特别讲述了 SQL Server 2008 Express Edition，还介绍了基本的 CRUD 操作，它用来创建、读取、更新和删除存储在数据库的数据。第 13 章的重点是使用 SqlDataSource 控件和可用的不同数据绑定控件。第 14 章探讨了 ADO.NET Entity Framework，这是 Microsoft 最新的数据访问策略，用来加速数据访问代码的编写。

为了帮助理解数据访问的核心概念，这 3 章将重点放在了数据源控件及其后台运行的原理上，而对使用数据绑定控件进行数据表示未作详细介绍。显然，在实际的应用程序中，这是不够的，需要以整洁、美观的方式显示数据。

ASP.NET 4 自带的数据绑定控件提供了许多改变数据表示方式的选项。它们允许完全改变所表现数据的设计(字体、颜色、间隔等)。另外，还可以调整这些控件来隐藏特定列、修改列标题，甚至通过编写代码来修改控件的外观。

在接下来的小节中，将看到如何使用多种技术来设置控件的样式。本章后面的章节中将显示如何手动编写数据访问页面的代码，从而提供了更大的灵活性。在本章末尾，还将讨论缓存，这是一种可以提高 Web 站点性能的技术。

15.1 使用样式格式化控件

第 13 章和第 14 章讲述了如何使用众多的 ASP.NET 4 数据绑定控件，学习了如何使用 GridView、Repeater 和 ListView 等控件显示和编辑数据列表，以及如何使用单个记录控件，如 DetailsView。

目前，所依赖的都是控件的内置外观，其结果是令人乏味的普通屏幕显示效果。图 15-1 显示了

在第 13 章创建的 GridView，它用来管理 Planet Wrox 数据库中的流派。

该控件根据浏览器的默认设置显示文本和链接，通常的结果是紫色和蓝色链接，以及像 Times New Roman 这样的默认字体。另外，网格中的列的宽度刚好可显示它们所包含的文本。如果按图 15-2 所示来显示 GridView，则视觉效果会更好。

	<u>Id</u>	<u>Name</u>	<u>SortOrder</u>
Edit Delete Select	1	Rap and Hip-Hop	14
Edit Delete Select	2	Pop	12
Edit Delete Select	3	Jazz	8
Edit Delete Select	4	Hard Rock	3
Edit Delete Select	5	Indie Rock	7
Edit Delete Select	6	Punk	1
Edit Delete Select	7	Rock	2
Edit Delete Select	8	Grunge	4
Edit Delete Select	9	Alternative Rock	9
Edit Delete Select	10	Reggae	11
12			

图 15-1

	<u>Name</u>	<u>Sort Order</u>
Edit	Punk	1
Edit	Rock	2
Edit	Hard Rock	3
Edit Delete	Grunge	4
Edit	Techno	5
Edit	Indie Rock	7
Edit	Jazz	8
Edit	Alternative Rock	9
Edit Delete	Industrial	10
Edit Delete	Reggae	11
12		

图 15-2

Edit 和 Delete 链接的列稍宽了些，使得它与网格的实际内容清楚地分开。Id 列被隐藏了，Name 列也更宽些。页眉、页脚、记录项及交替项以不同颜色显示，使得网格中的数据更容易阅读。因为一些流派带有评论，所以它们的 Delete 链接被禁用。最后，在 Sort Order 标题上添加了一个小三角符号来指出列的排序方向。

通过使用 ASP.NET 样式和数据绑定控件触发的许多事件，可以很轻松地将图 15-1 中单调的 GridView 改为图 15-2 中漂亮的 GridView。在下一节中，将介绍如何将这些样式应用于页面中的单个控件。随后一节中将介绍如何将样式应用到主题中，这样样式就可以被站点某个部分中的所有控件轻松重用。在第 7 章中，已经看到过一些样式，并使用它们样式化 Menu 和 TreeView 控件。不过，由于样式常用于格式化数据绑定控件，所以有必要单独介绍。

15.1.1 关于样式

许多数据绑定控件和导航控件都有大量可用于修改控件外观的样式属性。例如，GridView 控件有一个 RowStyle 属性，允许自定义网格中单个行的外观。同样，DetailsView 有一个 CommandRowStyle 属性，可用于控制存储如 Insert、Delete、Cancel 等命令的命令行的外观。

所有的样式属性最终都继承自 System.Web.UI.WebControls 名称空间中的 Style 类。图 15-3 显示了该类关系图筛选后的视图，其中只显示了其最常用的属性。

顾名思义，Style 类的属性用于改变与该类所应用到的对象的样式相关的信息。其每个属性最终都转换为 CSS 属性或 HTML 特性，如 background-color、border 等。其他样式，如用于 GridView 控件的样式，将添加不同的与布局相关的属性，如控制对齐的不同选项。表 15-1 列出了可用的不同 Style 派生类的最重要的属性。注意，并不是每个属性都可用于每种样式。IntelliSense 工具会显示特定样式中可用的属性。

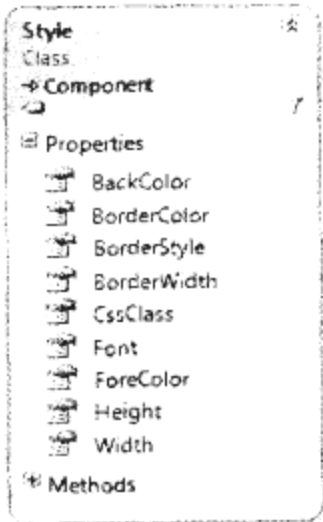


图 15-3

表 15-1

属 性	说 明
BackColor ForeColor	该属性允许改变元素的背景和文本颜色。它们分别映射到 CSS 属性 background-color 和 color
BorderColor BorderStyle BorderWidth	这些属性可以改变样式所应用到的元素的边框。它们直接映射到 CSS 对等属性 border-color、border-style 和 border-width
CssClass	这一属性可以指派一个 CSS 类而不是内联样式信息。相比于单个样式属性，应该优先选择使用 CssClass 属性，因为使用它们可以最小化页面膨胀。在后面的“试一试”练习中将介绍如何使用 CssClass
Font	该属性允许通过不同的子属性(如 Font-Names、Font-Size 和 Font-Bold)设置元素的字体。这些属性最终成为不同的 CSS 字体属性，如 font-family、font-size 和 font-weight
HorizontalAlign VerticalAlign	这些属性最终成为它们所应用到的 HTML 元素上的 align 和 valign 属性，允许控制元素内容的对齐方式。例如，使用 HorizontalAlign 左对齐、居中对齐和右对齐 GridView 的列标题的文本
Wrap	如果设置为 Flase，Wrap 属性最终成为 white-space: nowrap; CSS 声明，它决定内容是否可以换行
Height Width	这些属性允许控制它们所应用到的元素的宽度和高度。它们直接映射到与 CSS 对等的 height 和 width 属性

不同的数据绑定控件尽管共享了一些样式，但都有不同的样式集。表 15-2 列出了 GridView 和 DataList 类的可用样式并描述了它们的用途。其他的数据绑定控件有一些稍微不同的样式，但从这些样式的名称中可知道它们的用途。可以查看 MSDN 文档(网址为：<http://tinyurl.com/lSab9e>)获取对 Style 类的详细描述。另一种好方法是通过使用 Visual Web Developer 的 Auto Format 功能插入大量样式，来学习可用的不同样式。后面将学习如何使用和改进由 Auto Format 功能生成的样式。

表 15-2

样 式	说 明
RowStyle AlternatingRowStyle	这些样式可控制 GridView 中单个行的外观。默认情况下，RowStyle 影响所有行，而 AlternatingRowStyle 只可重写偶数行的 RowStyle(如果设置了 RowStyle)。由于 DataList 并不用于行，但对于更普通的数据项，它有 ItemStyle 和 AlternatingItemStyle 属性，其行为与此多少有些相似
SelectedRowStyle	这一样式可应用于选中的行，将选中的行和未选中的行作不同显示，从而使其易于区分。DataList 控件有一个 SelectedItemStyle 属性，作用与此相同
EditRowStyle	这一样式可应用于编辑模式中的当前行。例如，单击 Management 部分 Genres 页面上 GridView 中某一行的 Edit 链接时，该行切换至编辑模式，然后应用这个 EditRowStyle。为了定义 DataList 中可编辑项的布局，要使用 EditItemStyle
EmptyDataRowStyle	这一样式允许定义在网格与空数据源绑定时显示的行的外观。这一样式可与网格的 EmptyDataText 属性一起使用，该属性包含在无记录时显示的文本；或是与 EmptyDataTemplate 一起使用，该属性允许定义在使用空数据源时显示的自定义模板。DataList 没有针对 EmptyDataRowStyle 的支持

(续表)

样 式	说 明
HeaderStyle FooterStyle	这些样式允许控制 GridView 的页眉和页脚
PagerStyle	这一样式可以影响在启用分页时 GridView 中显示的分页条(pager bar)的外观。DataList 本身并不支持分页，因此没有 PagerStyle
SortedAscendingCellStyle SortedAscendingHeaderStyle SortedDescendingCellStyle SortedDescendingHeaderStyle	这些样式可以用来在按照升序或者降序排序时，改变页眉和整个列的外观。这些样式只在 GridView 控件内可用

一些控件(如 Repeater 和 ListView)则没有内置样式。由于这些控件并不单独为页面提供 HTML，而是让您在这些控件所拥有的大量模板中定义外观，因此拥有单独的样式毫无意义。可以将必要的样式或者类信息添加到模板中定义的元素上。

为了显示如何对控件使用这些样式，下面的“试一试”练习将带领您改善 Management 部分的 Genres 页面中的 GridView 控件的外观。在后一“试一试”练习中，将介绍如何将与样式相关的信息移至主题和 CSS 文件，来提高代码的可重用性和减少在每次请求中发送到浏览器的 HTML 的量。

试一试

应用样式

在本练习中，使用 VWD 的内置格式化功能改变 GridView 控件的外观。将看到 VWD 如何创建必要的样式，它们每个都带有相关的样式化属性集。在后一“试一试”练习中，将介绍如何修改这些样式来使用 CssClass 属性，从而得到一个更易于管理的样式集。

- (1) 从 Planet Wrox 应用程序的 Management 文件夹中打开 Genres.aspx。
- (2) 切换页面至 Design 视图并打开 GridView 控件的 Smart Tasks 面板。确保打开的是 GridView 而不是周围 Content 块的 Smart Tasks 面板。
- (3) 在该面板顶部，单击 Auto Format 链接。
- (4) 从左侧的格式模式中选择 Classic，右侧的 Preview 窗口得到更新，如图 15-4 所示。

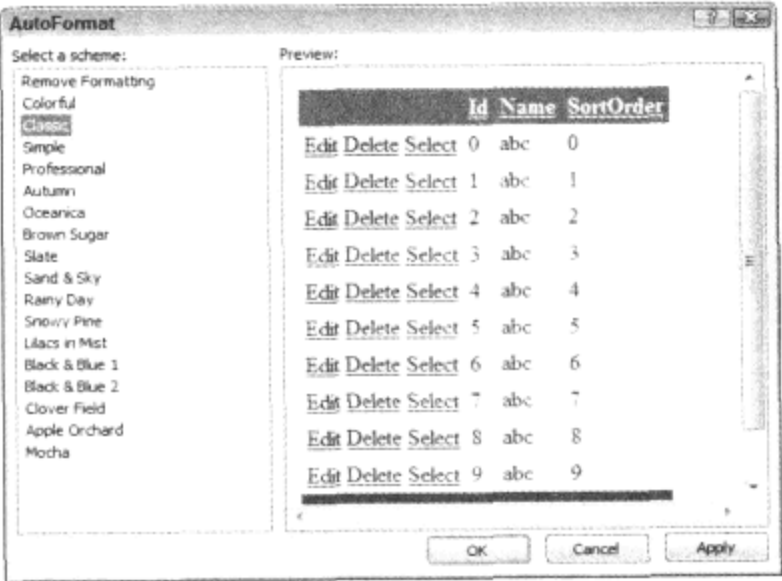


图 15-4

(5) 单击 OK 按钮使 VWD 生成必要的模板。GridView 在 Design 视图中立即更新，显示选择的格式模式。

(6) 切换至 Markup 视图，检查生成的不同样式。将看到下列样式，一些位于<Columns>元素之前，另一些位于其后。

```
<AlternatingRowStyle BackColor="White" />
...
<EditRowStyle BackColor="#2461BF" />
<FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
... Some styles are not shown here to save some space
<SortedDescendingHeaderStyle BackColor="#4870BE" />
```

(7) 保存对页面的更改，并按下 Ctrl+F5 组合键在浏览器中请求该页面。将看到一个应用了所选的格式模式的流派列表。

(8) 右击浏览器中的页面并选择 View Source 或 View Page Source 命令，打开页面的 HTML 源代码。稍往下滚动，直至看到一个其 id 设为 cpMainContent_GridView1 的 HTML 表。可以看到，该表有一个设置文本颜色和边框属性的 style 特性。

```
<table cellpadding="0" cellspacing="4" id="cpMainContent_GridView1"
    style="color:#333333;border-collapse:collapse;">
```

另外，可看到该表的大量子元素(表行和锚元素)都应用了不同的样式设置。例如，奇偶行应用了下列样式：

```
<tr style="background-color:#EFF3FB;"> ... </tr>
<tr style="background-color:White;"> ... </tr>
```

单击 GridView 的页眉几次。页眉和列将改变颜色，以指示列已被排序。

工作原理

第(5)步创建的不同样式元素被转换成对应的 CSS 和 HTML 元素。例如，RowStyle 和 Alternating-RowStyle 将它们的 BackColor 设置为不同的背景色：

```
<RowStyle BackColor="#EFF3FB" />
<AlternatingRowStyle BackColor="White" />
```

当控件呈现其 HTML 时，它将这些背景色应用到项或交替项的表行：

```
<tr style="background-color:#EFF3FB;"> ... </tr>
<tr style="background-color:White;"> ... </tr>
```

GridView 中其他样式的应用原理与此相同。

如果在浏览器中查看页面的源代码，可以看到大量的页面膨胀现象，因为每个单独行都设置了其属性。这就增加了页面大小，特别是有更多的结果显示在 GridView 中时。为了降低页面大小，提高页面性能，需要将样式定义移至页面主题，然后使用 CSS 和 jQuery。下面将作介绍。

15.1.2 合并样式、主题和外观

第 6 章讨论了如何使用母版页、主题和外观创建外观一致的 Web 页面。在建立了基本的主题基础结构后，现在可以很容易地添加应用于整个 Management 部分的新主题。您已经在前面学习了如何创建外观文件来改变按钮的外观；在下面的“试一试”练习中，将重用这一概念，为 GridView 创建外观文件，从而可以一次样式化 Management 文件夹中的所有 GridView 控件。

试一试

创建高级的样式解决方案

在这一“试一试”练习中，将 Genres.aspx 页面中的各种 Style 属性移至一个单独的.skin 文件。还将内联样式信息移至一个单独的 CSS 文件。然后使用 jQuery 进一步分离页面的数据和外观。

(1) 在 Solution Explorer 上，右击 App_Themes 文件夹，选择 Add ASP.NET Folder | Theme 命令，然后输入 Management 作为新主题名称。

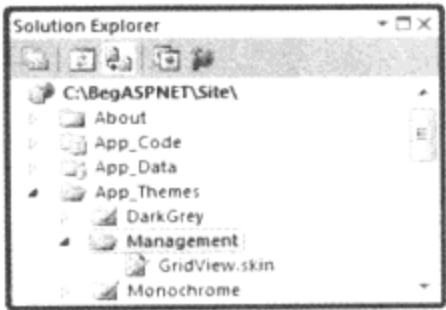


图 15-5

(2) 右击这个新文件夹并选择 Add New Item 命令。添加一个名为 GridView.skin 的外观文件。得到的 Solution Explorer 如图 15-5 所示。

(3) 在 Markup 视图中打开 Genres.aspx 页面并删除在前一个“试一试”练习中创建所有样式，只保留 HeaderStyle、PagerStyle、SortedAscendingHeaderStyle 和 SortedDescendingHeaderStyle。对于剩下的 4 个样式，删除所有特性并用一个 CssClass 特性取代它们，按样式给该特性命名并加上 GridView 前缀。得到的样式如下所示：

```
<HeaderStyle CssClass="GridViewHeaderStyle" />
<PagerStyle CssClass="GridViewPagerStyle" />
<SortedAscendingHeaderStyle CssClass="GridViewSortedAscendingHeaderStyle" />
<SortedDescendingHeaderStyle CssClass="GridViewSortedDescendingHeaderStyle" />
```

如果 VWD 在 CSS 类名下添加了红色错误线，也不用担心。因为 CSS 类还没有定义，所以 VWD 无法找到它们。后面将把它们添加到主题 CSS 文件中，在那里 VWD 依然无法找到它们。不过，它们在运行时工作良好，所以不用担心。

(4) 在代码编辑器中选择所有 4 种样式，然后使用 Ctrl+X 组合键将它们剪切至剪贴板。切换至 GridView.skin 文件，删除所有已有代码(之前看到的注释文本)，粘贴这 4 种样式。

(5) 将 4 个样式包装到一个<asp:GridView>元素中，设置其 runat 特性为 server，其 CssClass 特性为 GridView。不要添加 ID 特性，因为外观文件不需要它。结果代码如下所示：

```
<asp:GridView runat="server" CssClass="GridView">
    ... styles go here
</asp:GridView>
```

(6) 在 Windows 资源管理器中打开本章的资源文件夹(如果按照本书的前言进行操作，那么资源文件夹位于 C:\BegASPNET\Resources\Chapter 15)，选中 Images 文件夹和 Management.css 文件，并按下 Ctrl+C 组合键复制它们。切换到 VWD，单击 App_Themes 下的 Management 文件夹，并按下

Ctrl+V 组合键。和其他两个主题一样，Management 现在有了自己的样式表和 Images 文件夹，如图 15-6 所示。

在后面将会看到，Images 文件夹中的两个图像用来改变 GridView 中排序列的页眉。

(7) 打开前面添加的 Management 文件夹中的 web.config 文件，设置 theme 为 Management:

```
<system.web>
  <pages theme="Management"></pages>
</system.web>
```



图 15-6

(8) 打开 MasterPages 文件夹下的 Management.master 文件，切换到 Markup 视图，将 jquery-1.4.1.min.js 从 Scripts 文件夹中拖放到页面的<head>部分的 ContentPlaceHolder 的下面。VWD 会插入下面的<script>元素:

```
</asp:ContentPlaceHolder>
<script src="../../Scripts/jquery-1.4.1.min.js" type="text/javascript"></script>
</head>
```

(9) 回到 Genres.aspx，在 Markup 视图中，在 GridView 控件的 Columns 元素下面删除 Id 列的绑定字段。用户一般不需要查看用户界面中项的 ID，因为它们通常无意义。删除 Id 列，可以减少页面中的无用数据。将 CommandField 的 ItemStyle-Width 设置为 100px，将 Name 列的 ItemStyle-Width 设置为 200px。最后，设置 CommandField 的 ShowSelectButton 为 False，设置 SortOrder 字段的 HeaderText 为 Sort Order，注意两个单词之间的空格。这时的 GridView 如下所示:

```
<asp:GridView ID="GridView1" runat="server" AllowPaging="True"
  AllowSorting="True" AutoGenerateColumns="False" DataKeyNames="Id"
  DataSourceID="SqlDataSource1" GridLines="None" CellPadding="4"
  ForeColor="#333333" EmptyDataText="There are no data records to display.">
  <Columns>
    <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
      ShowSelectButton="False" ItemStyle-Width="100px" />
    <asp:BoundField DataField="Name" HeaderText="Name"
      SortExpression="Name" ItemStyle-Width="200px" />
    <asp:BoundField DataField="SortOrder" HeaderText="Sort Order"
      SortExpression="SortOrder"></asp:BoundField>
  </Columns>
</asp:GridView>
```

(10) 在 Markup 视图中，向下滚动至页面的末尾，在结束标记</asp:Content>之前，添加下面包装在<script>块中的 jQuery 代码。

```
<script type="text/javascript">
  $(function()
  {
    $('.GridView tr:odd:not(.GridViewPagerStyle)').
      addClass('GridViewAlternatingRowStyle');
  });
</script>
```


(11) 按下 Ctrl+Shift+S 组合键保存全部改动，然后在浏览器中打开 Genres.aspx 页面。现在可以看到图 15-2 中显示的流派列表，但是还看不到禁用的 Delete 链接。后面我们将会添加这个链接。单击 Name 或者 Sort Order 列的标题，对 GridView 中的数据进行排序。注意，GridView 现在在名称的旁边显示了一个小三角来指明排序的方向。

(12) 单击 Management 主菜单中的 Manage Reviews 打开 Reviews 页面。从下拉列表中选择一个流派以显示评论列表。注意，如图 15-7 所示，除了 jQuery 应用的交替行样式以外，现在 Reveiws 列表应用了与之前看到的 Genres 列表一样的样式。



图 15-7

工作原理

本练习中的概念应该还是较为熟悉的。在第 6 章中介绍过如何创建和应用主题和外观，前面的“试一试”练习中也介绍了如何使用各种控件样式。在第 11 章也了解了 jQuery 所涉及的概念。本练习中唯一的新内容就是选择 GridView 中的奇数行，然后动态改变它们的背景色，并使用 not 筛选器跳过页脚的方式。

```
$('.GridView tr:odd:not(.GridViewPagerStyle)')
```

首先，使用选择器 GridView tr:odd 选择表的所有奇数行。然而，根据 GridView 中行的数目，也可能会选择页脚行(其中含有分页控件)，因为页脚也被呈现为一个<tr>。为了避免包含页脚，代码中使用了 not 筛选器，并向它传递一个想要筛选的表达式。在本例中，表达式为 GridViewPagerStyle，因为它是应用到页脚行的类名。jQuery 代码只被应用到了 Genres.aspx 页面，但是可以将它移动到 Management 母版页，或者将它复制到单独的页面中。不管使用哪种方式，都可以消除页面膨胀，因为不需要向 GridView 中的每一行添加一个 style 或 class 特性。相反，可以让 jQuery 找出奇数行和偶数行。如果愿意，可以在 Managment 母版页中创建一个 ContentPlaceHolder，就像在 Frontend.master 文件中所做的那样。

将图像分配到排序后的列标题要求改变一些选择器。首先，为每个排序的标题(升序或者降序)设置一些内边距：

```
.GridViewSortedAscendingHeaderStyle, .GridViewSortedDescendingHeaderStyle
{
    padding-left: 20px;
}
```

这会将标题单元格中的文本略微右移，为图像留出空间。然后，对于升序和降序排序顺序，都

有一个分配图像的单独的选择器。选择器由 ASP.NET 通过向相关的 HTML 元素添加一个 class 特性来应用。下面的代码显示了按照升序排序的一个列的选择器：

```
.GridViewSortedAscendingHeaderStyle
{
    background-image: url (Images/SortAscending.png);
}
```

然后,GridViewHeaderStyle th 选择器阻止背景图像重复显示,并把图像放到顶部附近,并确定背景色和文本对齐方式:

```
.GridViewHeaderStyle th, .GridViewPagerStyle
{
    background-color: #BCD1FE;
    background-repeat: no-repeat;
    background-position: 0 5px;
    text-align: left;
}
```

通过将控件样式声明移至单独的外观文件(这一文件会作为主题的一部分),可以创建一个非常灵活的、可维护的解决方案。如果想查看应用新样式的方式,可以在浏览器中使用 View Source 命令打开页面的源代码。应用的是相关的 class 特性,而不是内联样式。如果想改变 Management 部分中的所有 GridView 控件的布局,所需做的是修改 Management.css 文件中的相关 CSS。如果需要修改其他样式,要先将它们添加到 GridView.skin 文件中。

显然,仍然可以在页面级调整控件。虽然 skin 文件定义了 GridView 的全局外观,但是仍然可以设置列的单个属性,就像对 Genres 页面的 ItemStyle-Width 所做的那样。

尽管样式、外观和主题都是样式化 Web 页面的强大工具,但它们通常都是极端的解决方案。例如,如果创建了 ItemStyle 和 AlternatingItemStyle 元素(而不是像刚才那样使用 jQuery 代码),它们则应用于网格中的每一行。如果只想改变少许行的外观,该怎么办? 如果想根据行持有的实际数据来改变一些行,又该怎么办? 在下一节中,将介绍如何实现条件格式化和事件处理的更多信息。

15.2 处理事件

到目前为止的大部分章节都讨论了 ASP.NET 控件如何触发事件。介绍了如何通过事件处理程序代码(通常将它们添加到页面的 Code Behind 文件中)来处理这些事件。例如,编写代码处理 Button 控件的 Click 事件。另外,在第 14 中,我们学习了如何处理刚好在与数据库作交互之前或之后发生的事件,如 Inserting 和 Inserted。不过,大部分的控件都提供了更多的事件。

深刻理解在控件的生命周期中触发的各种事件以及触发的顺序对于 ASP.NET 开发人员来说是很重要的。通过“挂钩到”控件的生命周期,调整输出的部分,可以创建灵活的、动态的、符合需要的 Web 页面。

为了了解各种事件和它们触发的顺序,下一节介绍了 ASP.NET 控件生命周期中的基本步骤。不需要了解这一过程中触发的每一个事件,而只需看一下最有可能使用的事件。后面几节将介绍如何使用这些事件来改变 Web 页面的行为。

15.2.1 回顾 ASP.NET 页面和控件生命周期

在第 6 章中，学习了页面生命周期中的不同阶段。还学习了不同的事件，如 PreInit、Load、PreRender 和 Unload。除了 ASPX 页面引发的这些事件外，该页面中的其他所有控件也可以引发它们自己的事件。这些事件可能是像 Button 控件的 Click 事件(由用户动作触发)这样简单的事件，也可能是更复杂的事件，如由 EntityDataSource 和 SqlDataSource 这样的控件引发的 Inserting 事件，或者由各种数据绑定控件引发的 DataBound 事件。在下面的练习中将看到许多这样的事件。

试一试 了解运用中的页面和控件生命周期

为了了解在页面或控件生命周期中可挂钩的不同事件及它们触发的顺序，本“试一试”练习演示了如何使用 EntityDataSource 建立一个显示 Genres 表中的一些数据的页面。另外，在该页面上放置一个按钮，可用它触发回发事件来查看其影响。然后，将大量的事件处理程序与页面上控件的一些有趣的事件关联，这样可以了解调用的顺序。

- (1) 在 Demos 文件夹中创建一个名为 Events.aspx 的新文件。确保它基于您自定义的页面模板，这样它就继承自 BasePage。将该页面的 Title 设置为 Events Demo。
- (2) 切换页面至 Design 视图，拖放一个 GridView 至 cpMainContent 占位符中，然后使用该控件的 Smart Tasks 面板将它与一个新的 EntityDataSource 控件关联。使用 PlanetWroxEntities 作为命名连接，并将 EntityDataSource 控件与 Genres 实体集绑定。不需要建立插入、更新或删除行为，也不需要添加 Where 子句。
- (3) 回到 GridView 控件的 Smart Tasks 面板，如果 GridView 没有显示 Id、Name 和 SortOrder 列，就单击 Refresh Schema。然后通过选择第二个复选框启用排序。如果 Id 列的 BoundField 没有 ReadOnly 特性，将特性添加到 Markup 视图中，并设置其值为 True。最后，如果 GridView 没有设为 Id 的 DataKeyNames 特性，就手动添加它。完成后，ASPX 页面中将出现下列代码：

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="Id" DataSourceID="EntityDataSource1" AllowSorting="True">
    <Columns>
        <asp:BoundField DataField="Id" HeaderText="Id"
            ReadOnly="True" SortExpression="Id" />
        <asp:BoundField DataField="Name" HeaderText="Name"
            SortExpression="Name" />
        <asp:BoundField DataField="SortOrder" HeaderText="SortOrder"
            SortExpression="SortOrder" />
    </Columns>
</asp:GridView>
<asp:EntityDataSource ID="EntityDataSource1" runat="server" EntitySetName="Genres"
    ConnectionString="name=PlanetWroxEntities" EnableFlattening="False"
    DefaultContainerName="PlanetWroxEntities">
</asp:EntityDataSource>
```

- (4) 确保在 Markup 视图中。在起始 Content 标记后面，添加下面的标记，创建一个带有一行和两个单元格的表，其中每个单元格都具有一个大标题(h1)和一个分别叫做 NoPostBack 和 PostBack 的 Label 控件。

```
<table>
  <tr>
    <td><h1>NoPostBack</h1><asp:Label ID="NoPostBack" runat="server" /></td>
    <td><h1>PostBack</h1><asp:Label ID="PostBack" runat="server" /></td>
  </tr>
</table>
```

(5) 切换至 Design 视图，在 GridView 下拖放一个 Button 控件，双击它以在 Code Behind 文件中为其 Click 事件建立事件处理程序。

(6) 切换回 Design 视图并双击页面的灰色只读区域，为 Page 控件的 Load 事件建立处理程序。

(7) 再次切换回 Design 视图，单击 GridView，按 F4 键打开其 Properties 面板。切换至 Events 选项卡，然后双击下列事件在 Code Behind 文件中为它们建立事件处理程序。建立每个事件处理程序后，按下 Ctrl+Tab 组合键切换回 Design 视图，这样可以添加下一个事件。

- Sorted
- Sorting
- RowCreated
- DataBinding
- DataBound
- RowDataBound

(8) 重复前一步骤，但这次是为 EntityDataSource 控件建立下面的事件：

- ContextCreating
- Selecting

(9) 确保位于页面的 Code Behind 文件中，在最后一个事件处理程序(但仍在类定义中)下，添加如下方法，该方法根据当前页面请求是否是回发结果来写一些文本到其中一个标签：

VB.NET

```
Private Sub WriteMessage(ByVal handlerName As String)
  If Page.IsPostBack Then
    PostBack.Text &= handlerName & "<br />"
  Else
    NoPostBack.Text &= handlerName & "<br />"
  End If
End Sub
```

C#

```
private void WriteMessage(string handlerName)
{
  if (Page.IsPostBack)
  {
    PostBack.Text += handlerName + "<br />";
  }
  else
  {
    NoPostBack.Text += handlerName + "<br />";
  }
}
```


(10) 对于建立的每个事件处理程序，添加下列代码，该代码调用自定义的方法并给它传递事件的名称，然后由 `WriteMessage` 方法将事件的名称添加到两个 `Label` 控件。不要忘记用实际的事件名代替文本中的事件名。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    WriteMessage("Page_Load")
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    WriteMessage("Page_Load");
}
```

(11) 最后，手动将下列事件处理程序添加到页面。

VB.NET

```
Protected Sub Page_PreRenderComplete(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles Me.PreRenderComplete
    WriteMessage("Page_PreRenderComplete<br />-----")
End Sub
```

C#

```
protected void Page_PreRenderComplete(object sender, EventArgs e)
{
    WriteMessage("Page_PreRenderComplete<br />-----");
}
```

`PreRenderComplete` 事件在 `Page` 控件的生命周期中触发得比较晚，这使得这里非常适合在事件列表的底部放置一条线。这样，可以清楚地看出事件集的所属关系，从而有助于找出在哪次页面加载或回发中触发了哪些事件。

(12) 保存所有更改并在浏览器中打开页面。除了带有可用流派的 `GridView` 外，可以在 `No Postback` 标题下看到事件名列表：

```
Page_Load
EntityDataSource1_ContextCreating
EntityDataSource1_Selecting
GridView1_DataBinding
GridView1_RowCreated
GridView1_RowDataBound
GridView1_RowCreated
GridView1_RowDataBound
...
GridView1_DataBound
Page_PreRenderComplete
-----
```

注意， `RowCreated` 和 `RowDataBound` 事件重复了多次：数据库中的每个流派对应一次，除此

以外还有两次。后面将会看到其原因。单击 GridView 下面的按钮会引发一次回发。NoPostBack 标签没有改变，但PostBack 标签现在显示了下列事件名列表：

```
GridView1_RowCreated
GridView1_RowCreated
...
GridView1_RowCreated
GridView1_RowCreated
Page_Load
Button1_Click
Page_PreRenderComplete
-----
```

单击 GridView 的一个列标题对显示的数据进行排序。注意，第二个标签的文本用第二组事件名扩充。每组事件名之间用 Page_PreRenderComplete 事件处理程序创建的短划线隔开。

工作原理

从技术上讲，这一练习并不复杂。为页面中各种控件建立了一组事件处理程序。在事件处理程序中调用了一个方法，检查页面当前是首次加载还是由于回发而加载。最后，代码用触发事件处理程序的事件的名称更新其中一个 Label 控件。

本练习有意思的地方在于事件发生的顺序。看一下页面首次加载时显示的第一个列表：

```
Page_Load
EntityDataSource1_ContextCreating
EntityDataSource1_Selecting
GridView1_DataBinding
GridView1_RowCreated
GridView1_RowDataBound
GridView1_RowCreated
GridView1_RowDataBound
...
GridView1_DataBound
Page_PreRenderComplete
-----
```

首先触发 Page_Load 事件。然后 GridView 知道它关联到一个 EntityDataSource，并向该控件请求数据。这就导致 ContextCreating 和 Selecting 事件被触发。当 GridView 接收 EntityDataSource 的数据时，它触发其 DataBinding 事件通知它将绑定数据到该控件。然后 GridView 开始创建行。对于数据源中的每个项，它都创建一行，触发 RowCreated 事件，将项的数据绑定到该行，最后调用 RowDataBound。如果仔细地对比 RowCreated 和 RowDataBound 的调用次数进行计数，则会发现它比数据源中的实际项数多调用两次。这是因为当控件创建其标题和页脚行时，也引发了这两个事件。在最后一“试一试”练习中，将介绍如何在事件处理程序中区分这些行。

最后，当 GridView 创建和绑定数据源中的所有行时，它触发其 DataBound 事件。回发的情况很不同。当单击按钮进行回发时，引发下列事件：

```
...
GridView1_RowCreated
GridView1_RowCreated
```

```
GridView1_RowCreated
GridView1_RowCreated
Page_Load
Button1_Click
Page_PreRenderComplete
-----
```

注意，该列表中没有 RowDataBound 或 DataBound 事件，也没有 EntityDataSource。GridView 可以通过 View State 重构整个控件，从而无需再次访问数据库。从 View State 中获取数据时，GridView 仍需要重新创建网格中的每一行，所以仍可看到 RowCreated 事件。在列表最后，可以看到 Page_Load 事件后紧跟着 Button 控件的 Click 事件。要了解和记住的是，用户触发的控件事件，如 Button 控件的 Click 或 DropDownList 的 SelectedIndexChanged，这些事件在 Page 的 Load 事件之后发生。注意，这个 Load 事件并不是 Page 控件生命周期的起点。在 Load 事件之前，Page 已经实例化，并已触发了其 Init 事件。

在该练习的最后，单击列标题排序网格中的数据。这次，GridView 知道它必须对显示的数据进行排序。它自己不能做这一工作，因此它向 EntityDataSource 请求按用户要求的顺序排序的数据的更新副本。和首次加载页面一样，可看到许多 RowCreated 和 DataBound 事件出现。

如果了解其他事件的运用，可重复前一练习的第(7)和第(10)步，为各种事件建立事件处理程序。要了解 View State 的作用，可尝试在控件级别(例如对于 GridView)或页面级别禁用它。在第 18 章将学习一种叫做跟踪的技术，使用这种技术可以为页面中的所有控件找出这些信息，包括执行各种事件所需的时间。

尽管这一“试一试”练习在实际应用中没有什么用途，但是却可以帮助了解各种控件事件和它们的触发顺序。不过，您可以使用同样的原理挂钩到页面，对页面或者页面中的控件作些修改。在下一“试一试”练习中，将介绍如何根据显示的数据，改变数据源中的行的外观。

15.2.2 ASP.NET 页面生命周期和数据控件中的事件

正如前面所讨论的，GridView 对于它添加到输出的每一行都引发了其 RowCreated 和 RowDataBound 事件。可以通过这些事件窥视数据，然后根据数据采取合适的动作。例如，可以使用这些事件验证显示的评论是否是经过授权的。如果未经授权(意味着在前端 Web 站点不可见)，可以改变其外观使其引起注意。使用事件的另外一个例子是当元素没有必要可见或者处于活动状态时，在界面上隐藏或者禁用这些元素。在接下来的“试一试”练习中将介绍如何在 Genres GridView 中禁用 Delete 链接。

试一试 挂钩到 RowDataBound

在这个“试一试”练习中，为 Management 部分的 Genres 页面中的 GridView 控件的 RowDataBound 事件编写事件处理程序。在该事件中，可以诊断绑定到 GridView 行的数据项，从而允许查看流派是否有评论。如果流派有评论，则应用一些代码来禁用 Delete 链接，从而避免用户误删该流派。

(1) 在 Markup 视图中打开 Management 文件夹中的 Genres.aspx 页面，定位到 SqlDataSource 控件。找到 SelectCommand，并对 SQL 语句作如下修改：

```
SelectCommand="SELECT Genre.Id, Genre.Name, Genre.SortOrder,
COUNT(Review.Id) AS NumberOfReviews FROM Genre LEFT OUTER JOIN Review
```



```
ON Genre.Id = Review.GenreId GROUP BY Genre.Id, Genre.Name, Genre.SortOrder"
```

可以将整条 SQL 语句输入到一行中，或者像上面这样将其分开到几行中。

(2) 切换到 Design 视图，打开 GridView 的 Smart Tasks 面板。如果控件给出一个关于缺少 Id 属性的错误，那么就需要单击 Smart Tasks 面板上的 Refresh Schema 链接，并对显示的关于重新生成字段和键来维护当前控件布局的问题选择 No。在 Smart Task 面板上单击 Edit Columns 打开 Fields 对话框。单击 Selected Fields 列表中的 CommandField 项，然后单击对话框右下角的蓝色链接将字段转换成一个 TemplateField。这样，列被扩展为一个模板，使得访问它所包含的控件(如 Delete 链接)更加容易。单击 OK 按钮关闭 Fields 对话框。

(3) 在 Markup 视图中，定位到 delete 链接(其 CommandName 被设为 Delete)，将其 ID 改为 DeleteLink:

```
<asp:LinkButton ID="DeleteLink" runat="server" CausesValidation="False"
    CommandName="Delete" Text="Delete"></asp:LinkButton>
```

(4) 切换至 Design 视图，打开 GridView 的 Properties 面板，选择 Events 选项卡。为 RowDataBound 事件建立一个事件处理程序。

(5) 在 Web 窗体的 Code Behind 文件的顶部，添加下列代码:

```
VB.NET
Imports System.Data

C#
using System.Data;
```

(6) 在 VWD 创建的事件处理程序中，添加下列代码:

```
VB.NET
Protected Sub GridView1_RowDataBound(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs) _
    Handles GridView1.RowDataBound
    Select Case e.Row.RowType
    Case DataControlRowType.DataRow
        Dim myRowView As DataRowView = CType(e.Row.DataItem, DataRowView)
        If Convert.ToInt32(myRowView("NumberOfReviews")) > 0 Then
            Dim deleteLink As LinkButton =
                TryCast(e.Row.FindControl("DeleteLink"), LinkButton)
            If deleteLink IsNot Nothing Then
                deleteLink.Enabled = False
            End If
        End If
    End Select
End Sub

C#
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
```

```
switch (e.Row.RowType)
{
    case DataControlRowType.DataRow:
        DataRowView myDataRowView = (DataRowView)e.Row.DataItem;
        if (Convert.ToInt32(myDataRowView["NumberOfReviews"]) > 0)
        {
            LinkButton deleteLink = e.Row.FindControl("DeleteLink") as LinkButton;
            if (deleteLink != null)
            {
                deleteLink.Enabled = false;
            }
        }
        break;
}
```

(7) 保存对所有打开文件的更改，然后在浏览器中请求 Genres.aspx 页面。从图 15-8 中可以看到，当流派且有评论时，Delete 链接就会被禁用。

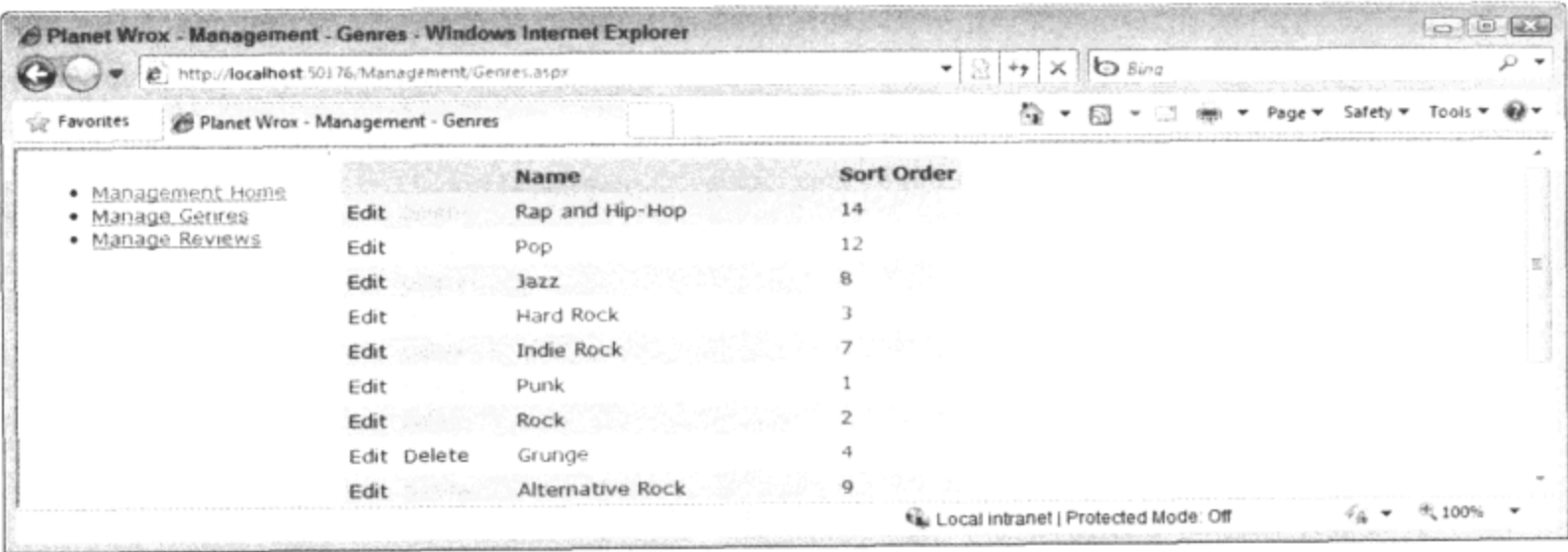


图 15-8

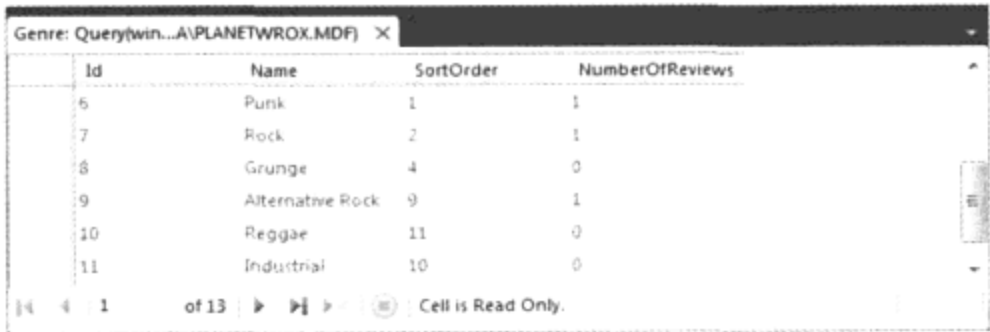
工作原理

本练习尽管较短，但还是演示了挂钩控件的不同事件和改变底层控件显示的有用方法。要了解其工作原理，首先看一下修改后的 SQL 代码：

```
SELECT
    Genre.Id, Genre.Name, Genre.SortOrder, COUNT(Review.Id) AS NumberOfReviews
FROM
    Genre LEFT OUTER JOIN
    Review ON Genre.Id = Review.GenreId
GROUP BY
    Genre.Id, Genre.Name, Genre.SortOrder
```

这段修改后的 SQL 语句将获取 Genre 表中的全部列，并创建一个新列 NumberOfReviews，它包含与每个流派关联的评论的数量。这是通过对 Review 表的 Id 列执行 SQL 函数 COUNT 实现的。因为该 SQL 语句在 Genre 表中的全部唯一列上分组，所以可以得到唯一的记录，其中包括每个流派记录的计数，而不管是否有评论关联到流派上，如图 15-9 所示，其中显示了在 VWD 中这个查询的

结果。



Id	Name	SortOrder	NumberOfReviews
6	Punk	1	1
7	Rock	2	1
8	Grunge	4	0
9	Alternative Rock	9	1
10	Reggae	11	0
11	Industrial	10	0

图 15-9

在执行这个查询时，页面标记中的 GridView 会使用前 3 列，就像这个页面的前一版本那样。但是，也可以访问第 4 列，这可以在 Code Behind 文件中的 RowDataBound 事件中实现。RowDataBound 事件在 GridView 将数据绑定到特定行后针对每一行触发。

VB.NET

```
Protected Sub GridView1_RowDataBound(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.GridViewRowEventArgs) _
    Handles GridView1.RowDataBound
    Select Case e.Row.RowType
        Case DataControlRowType.DataRow
            ...
    End Select
End Sub
```

C#

```
protected void GridView1_RowDataBound (object sender, GridViewRowEventArgs e)
{
    switch (e.Row.RowType)
    {
        case DataControlRowType.DataRow:
            ...
    }
}
```

RowDataBound 事件接受一个 GridViewRowEventArgs 实例，该类提供在这一阶段被绑定的行和数据的信息。其中一个属性是 Row，它表示添加到 GridView 的实际行。该行包含一个 RowType 枚举属性，可对其测试来查看添加的行的类型。这个枚举包含 6 个不同的成员，它们直接映射到 GridView 可包含的不同类型的行：用于普通行和交替行的 DataRow、用于空数据行的 EmptyDataRow、用于放置在 GridView 顶部的页眉行和底部的页脚行的 Header 和 Footer 等。由于需要改变实际数据行的外观，所以 Case 块中的代码只针对普通行和交替行而触发。

在 Case 块中，执行下列代码：

VB.NET

```
Dim myRowView As DataRowView = CType(e.Row.DataItem, DataRowView)
If Convert.ToInt32(myRowView("NumberOfReviews")) > 0 Then
    Dim deleteLink As LinkButton =
        TryCast(e.Row.FindControl("DeleteLink"), LinkButton)
    If deleteLink IsNot Nothing Then
```



```
        deleteLink.Enabled = False
    End If
End If
```

```
C#
DataRowView myDataRowView = (DataRowView)e.Row.DataItem;
if (Convert.ToInt32(myDataRowView["NumberOfReviews"]) > 0)
{
    LinkButton deleteLink = e.Row.FindControl("DeleteLink") as LinkButton;
    if (deleteLink != null)
    {
        deleteLink.Enabled = false;
    }
}
```

DataItem 属性包含对被绑定的数据项对象的引用。在使用 SqlDataSource 控件时，DataItem 被表示为一个 DataRowView，这是封装从数据库返回的行的 .NET 对象。因此，DataItem 被强制转换为 DataRowView 对象，然后它被索引(在 VB.NET 中使用 myRowView("NumberOfReviews")；在 C# 中使用 myRowView["NumberOfReviews"])以获得 NumberOfReviews 列的评论计数。如果计数大于 0，则说明这个流派关联有评论，必须隐藏其 Delete 链接。前面将 CommandField 转换为一个模板字段，该模板字段在代码中为 Delete 链接添加一个显式声明：

```
<asp:LinkButton ID="DeleteLink" runat="server" CausesValidation="False"
    CommandName="Delete" Text="Delete"></asp:LinkButton>
```

使用 FindControl 可以得到对 Delete 链接的一个引用，然后将其转换为一个合适的 LinkButton，设置其 Enabled 属性为 false。因为这些代码也会在 GridView 中的行处于 Edit 模式(通过单击 Edit 链接可以进入此模式)时调用，所以需要检查 deleteLink 是否为 null(VB.NET 中是 Nothing)。如果正在编辑，则 GridView 行不包含 DeleteLink(因为活动的是 EditItemTemplate，不是 ItemTemplate)，因此 FindControl 将返回 null。

在本例中，当通过把 Enabled 设为 False 来禁用 LinkButton 时，ASP.NET 会应用一个 CSS 类 aspNetDisabled：

```
<a id="cpMainContent_GridView1_DeleteLink_0" class="aspNetDisabled">Delete</a>
```

然后可以使用该 CSS 类(位于前面添加的 Management.css 中)设置此禁用链接的样式。将其设置为灰色。

```
a.aspNetDisabled
{
    color : #CCC;
}
```

通过使用这些代码，可以很容易地避免在尝试删除具有相关评论的流派时出现的错误。但是，在对数据源控件执行 CRUD 操作时，并不总是能够避免错误发生。例如，可能尝试删除一开始并没有关联评论的流派。但是，就在删除该流派之前，某个人为该流派插入了一个新评论。此时，当尝试删除该流派时，将会得到一个错误，因为现在该流派关联了一个评论。在这种情况下，数据源控

件允许诊断发生的错误，然后采取必要的措施，例如向用户提供反馈，告诉他们 CRUD 操作没有成功。

15.2.3 处理数据源控件中发生的错误

在第 18 章，将详细介绍如何识别和处理 ASP.NET 页面中发生的错误。该章演示了如何捕获可能在代码中出现的错误，然后通过将它们记入日志或是通知用户来进行处理。由于数据源控件也可以公开错误信息，本章将讨论数据访问错误。

EntityDataSource 和 SqlDataSource 控件都提供了有关可能在某一 CRUD 操作中发生的错误(.NET 中称为异常)的信息。在使用 EntityDataSource 时，在更新数据库后发生的 3 个事件(Inserted、Updated 和 Deleted)都接收一个名为 EntityDataSourceChangedEventArgs 的类的实例，而 Selected 事件则接收一个 EntityDataSourceSelectedEventArgs。而使用 SqlDataSource 时，4 个事件都接收一个 SqlDataSourceStatusEventArgs 实例。图 15-10 显示了这三个 EventArgs 类和它们的属性。



图 15-10

这三个类共享两个重要的属性：Exception 和 ExceptionHandled。第一个属性包含实际发生的异常，或是一切之行良好并无错误发生时的 Nothing 值(在 VB.NET 中)或 null 值(在 C#中)。可以检查这一错误并采取适当行动。例如，可通知用户发生了严重错误或是发送电子邮件给站长告知错误，以便进一步采取适当的行动。

如果决定在数据源控件的事件处理程序中处理错误，应设置对象的 ExceptionHandled 属性为 True。这就告知了 ASP.NET 运行库，您知道发生了异常并适当地处理了它。如果不设置该属性，运行库会转发该异常，最终会显示给用户。

在接下来的这个“试一试”练习中，将介绍如何在 Genres.aspx 页面中使用 SqlDataSourceStatusEventArgs 类。当然，本节采用的原理同样适用于 EntityDataSource 控件引发的事件。

试一试

处理删除行时的错误

在这个“试一试”练习中，将介绍如何处理在 GridView 中删除行时发生的异常。我们将暂时移除禁用 Delete 链接的代码，以便能够删除关联着评论的流派。然后，在用户尝试删除仍有评论与之相连的流派时显示错误消息。这一练习主要是演示如何处理由数据源控件抛出的异常。从终端用户角度来看，像之前练习中那样在不适合删除流派的时候禁用删除链接可以解决大部分情况下的问题，但是在删除流派前，仍有可能会有其他人插入新评论。

(1) 打开 Management 文件夹中的 Genres.aspx。

(2) 切换至 Design 视图并从 Toolbox 中拖动一个 Label 控件到 GridView。这就在 GridView 的上方放置了一个保存错误消息的 Label。将该 Label 的 ID 改为 ErrorMessage 并删除其 Text 属性(右击

Properties 面板中的 Text 属性标签并选择 Reset)。这就从控件的标记中删除了整个 Text 属性及其值。将其 CssClass 属性设置为 ErrorMessage。最后，设置其 EnableViewState 属性为 False，以确保该标签在回发后不维护其文本。最终代码如下所示：

```
<asp:Label ID="ErrorMessage" runat="server" CssClass="ErrorMessage"
           EnableViewState="False"></asp:Label>
<asp:GridView ID="GridView1" runat="server" AllowPaging="True"
```

(3) 从 Management 主题文件夹中打开 Management.css 文件，添加下列样式规则集：

```
.ErrorMessage
{
    color: Red;
    font-weight: bold;
}
```

(4) 切换至 Genres.aspx，确保页面在 Design 视图中，然后单击选中 SqlDataSource 控件。接着打开其 Properties 面板，切换至 Events 选项卡，通过在事件列表中双击 Deleted 事件的名称为它建立事件处理程序。

(5) 在 Code Behind 文件的顶部，添加下列名称空间：

```
VB.NET
Imports System.Data.SqlClient

C#
using System.Data.SqlClient;
```

(6) 在第(4)步中 VWD 添加的事件处理程序中，编写下列代码：

```
VB.NET
Protected Sub SqlDataSource1_Deleted(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.SqlDataSourceStatusEventArgs) _
    Handles SqlDataSource1.Deleted
    If e.Exception IsNot Nothing AndAlso
        TypeOf (e.Exception) Is SqlException Then
        Dim myException As SqlException = CType(e.Exception, SqlException)
        If myException.Number = 547 Then
            ErrorMessage.Text = "Sorry, you can't delete this genre because " &
                "it has associated reviews that you need to delete first."
            e.ExceptionHandled = True
        End If
    End If
End Sub

C#
protected void SqlDataSource1_Deleted(object sender,
    SqlDataSourceStatusEventArgs e)
{
```



```
if (e.Exception != null && e.Exception is SqlException)
{
    SqlException myException = (SqlException)e.Exception;
    if (myException.Number == 547)
    {
        ErrorMessage.Text = @"Sorry, you can't delete this genre because
                               it has associated reviews that you need to delete first.";
        e.ExceptionHandled = true;
    }
}
```

(7) 注释掉在前一个“试一试”练习中添加的代码，以启用 Delete 链接。对于这个练习来说，只需注释掉禁用链接的一行：

VB.NET

```
' deleteLink.Enabled = False
```

C#

```
// deleteLink.Enabled = false;
```

如果想要完全删除这个功能，可以删除整个事件处理程序。此时，不用忘了在 C#中也要从 GridView 的标记中删除该处理程序。

(8) 保存所有更改，然后按 Ctrl+F5 组合键在浏览器中打开 Genres.aspx。尝试删除一个关联有评论的流派，如 Rap and Hip-Hop。这时，ASPX 页面不会删除该流派，而是在 GridView 上方显示错误，如图 15-11 所示。

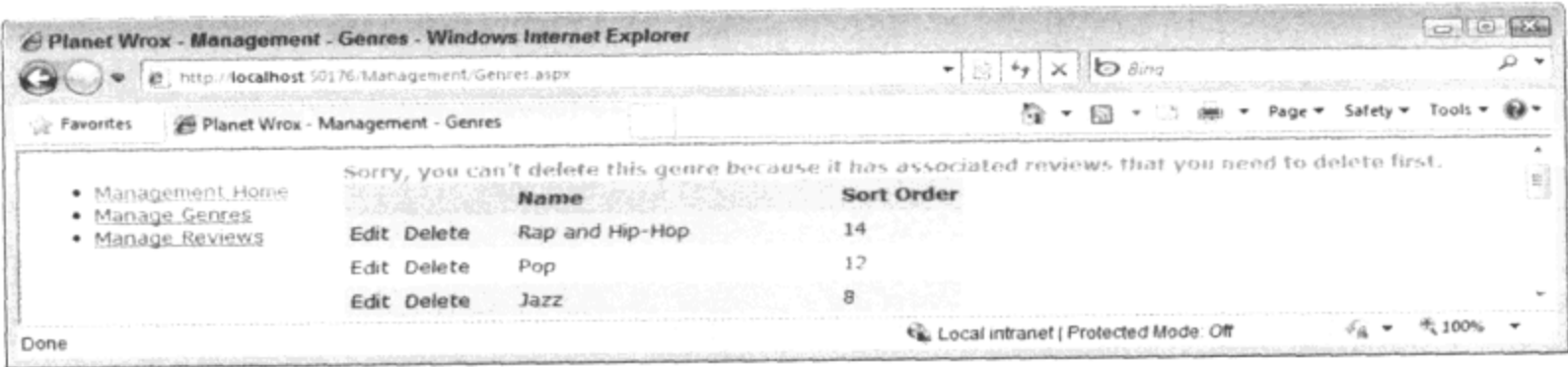


图 15-11

(9) 单击 Management 菜单中的 Manage Reviews 菜单项，然后从下拉列表中选择 Rap and Hip-Hop 流派。删除该流派中的评论，或是编辑它们并将它们指派给一个不同的流派。

(10) 回到 Genres.aspx，再次尝试删除 Rap and Hip-Hop 流派。这次，该流派成功地从数据库中删除。

(11) 为了看到该错误而不进行这一错误处理，可在 Code Behind 文件中注释掉把 ExceptionHandled 设置为 True 的那行代码。保存改动，然后再次在浏览器中打开该页面，尝试删除带有评论的流派。此时将会显示一个如图 15-12 所示的详细的 ASP.NET 错误页面。可注意到，这个错误与在第 12 章结尾尝试手动删除流派时出现的错误很相似。在完成操作后，不要忘了重新启用该行代码。

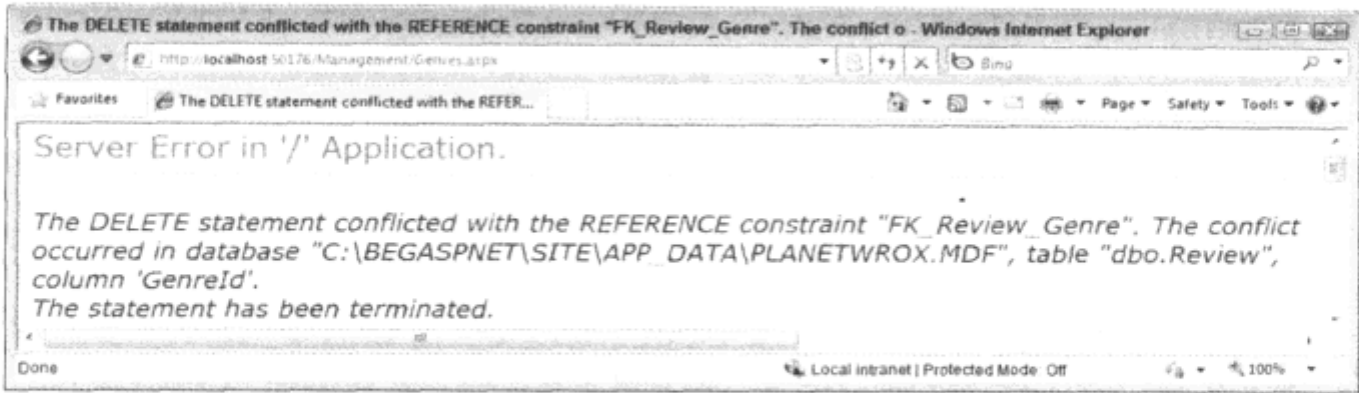


图 15-12

工作原理

在单击图 15-11 中所示的 GridView 中的 Delete 链接时，GridView 在相关 SqlDataSource 控件上触发 Delete 命令。正如在前面几章中看到的，该控件尝试向数据库发送一个 DELETE 语句。然后数据库尝试从数据库中删除请求的流派，但发现无法进行，因为流派有相关的评论。这就导致了外键约束错误(foreign key constraint error)；这意味着流派不能被删除，因为其 ID 是另一个表的外键。

然后，这一外键约束错误从数据库中返回，最终作为 SqlDataSource1_Deleted 处理程序的 e 参数的 Exception 属性。接着，代码检查是否有错误(e.Exception 不为 Nothing/null)，并检查异常的类型是否是 SqlException。

VB.NET

```
If e.Exception IsNot Nothing AndAlso _
    TypeOf (e.Exception) Is SqlException Then
    ...
End If
```

C#

```
if (e.Exception != null && e.Exception is SqlException)
{
    ...
}
```

如果使用的是 SQL Server 数据库，则像 Planet Wrox 示例中的情况，数据库抛出的错误是 SqlException 类型，该类型在本练习中导入的 System.Data.SqlClient 名称空间中定义。这就可以清楚地 将数据库错误与其他错误区分开。

当 SQL Server 抛出异常时，它也传递错误号，该错误号存储在异常的 Number 属性中。要访问该错误号，需要使用下列代码将异常强制转换为真正的 SqlException 类型。

VB.NET

```
Dim myException As SqlException = CType(e.Exception, SqlException)
```

C#

```
SqlException myException = (SqlException)e.Exception;
```

最后，代码检查 Number 属性。如果错误号为 547，则意味着 SQL Server 抛出一个外键约束错误，表明不能删除流派，因为它还有相关的评论。如果是这样，Label 控件的 Text 属性就被设置，最后，代码设置 e.ExceptionHandled 为 True。这就告诉 ASP.NET 运行库，错误已得到处理，用户将

不会得到令人生厌的错误页面，而是在 GridView 的顶部显示友好的错误消息。注意，对于其他异常类型，用户仍会得到默认的 ASP.NET 错误消息屏，这也被称为 Yellow Screen of Death。第 18 章将介绍一些将错误记录到一个集中的位置并向用户提供友好的、可读的错误页面的方法。

错误号 547 看起来像是随意选取的，但这是 SQL Server 针对外键约束异常而返回的。在第 18 章关于调试的内容中，可以学到一些技巧来深入检查抛出的异常，这样可以诊断不同类型异常的 Number 属性。

在前面几章中看到了使用内置的数据控件(如 SqlDataSource 和 EntityDataSource)访问数据库的许多示例。虽然它们很实用，也很易用，但是却并非适合每一种情况。在无法使用这些内置控件时，总是可以手动编写页面代码。下面将介绍这一方面的内容。

15.3 手动编写数据访问代码

在使用数据控件时，最大的一个问题就是它们需要的标记太多。虽然，像 ListView 等控件自动生成了大部分代码，但是页面中仍然会有大量代码。因此，对页面进行大量修改就成了一项既耗时又费力的工作。这些控件的另一个问题是，经常可以发现自己对几乎相同的标记定义了两次：一次用于 Insert 模板，一次用于 Update 模板。使用这些数据控件的最后一个问题是，对于它们创建的标记，开发人员没有完全的控制权。这就使得使用多层下拉绑定控件、AJAX UpdatePanel 控件、图像上传控件等工具创建漂亮复杂的页面变得很困难。为了解决这些问题，可以手动编写页面代码，这样就可以完全控制页面中的标记，以及 Code Behind 文件中的代码。

手动编写代码并不像看上去那么困难，使用这种方法具有很大的灵活性。虽然实际的编写过程因页面而异，但是下面给出了一些基本的操作步骤，遵循这些步骤，可以手动编写 Add/Edit 页面，从而能够使用相同的标记在数据库中输入新项或者更新现有项。

- 通过向允许用户输入新数据和更新现有数据的页面添加大量控件，如 TextBox 和 DropDownList，创建用户界面。
- 向页面添加有效性验证控件，强制用户输入有效数据。
- 在 Code Behind 中，确定是创建新项还是编辑现有项。例如，可以通过查看查询字符串来区分操作。当编辑现有项时，从数据源获取数据项并预填充表单控件。
- 处理 Save 按钮，以便插入或者更新项。当更新现有项时，应该首先从数据库中获取该项，然后使用从表单中获取的新值重写现有的值。最后，将数据项保存到数据存储器中。

在下一个“试一试”练习中，将构建一个实现了这个过程的页面。



注意：在接下来的“试一试”练习中，将手动编写用户界面的代码，并使用 ADO.NET Entity Framework 处理所有的数据访问。另外，使用诸如继承自 DbConnection、DbCommand 和 DbDataReader 的类这样的 ADO.NET 类手动编写与数据库交互的代码也很常见。虽然手动编写与数据库交互的代码需要编写大量代码，但是同时也获得了大量的控制权和灵活性。要想深入理解 ADO.NET，可以参考清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C# 和 VB.NET(第 7 版)》一书(ISBN: 978-7-302-23524-8)或者 *Professional ADO.NET 2*(ISBN: 978-0-7645-8437-4)。

试一试 手动编写数据访问页面的代码

在这个练习中，将创建 AddEditReview.aspx 页面的一个新版本，以便替换当前使用 DetailsView 来处理插入和更新过程的现有页面。在新页面中，将添加一些表单控件，用于输入评论的标题、摘要、正文、流派以及是否获得授权。在页面的 Code Behind 文件中，将使用 PlanetWroxEntities 类来处理所有的数据访问代码。为了保持练习的简练，将不会添加任何验证控件。但是，从第 9 章的讲解可以知道如何使这个页面只接受有效数据。

(1) 首先，使用 Code Behind 文件向站点的 Management 文件夹添加标准的 Web 窗体(不要使用自定义模板)，将其命名为 AddEditReviewHandCoded.aspx。使这个页面基于 Management 母版页，并给它添加一个有意义的标题。

(2) 切换到 Design 视图，选择 Table | Insert Table，然后插入 6 个行，2 个列。向 HTML 表的单元格添加控件，并根据表 15-3 设置它们的属性。

表 15-3

Row	Column 1	Column 2
1	添加一个 Label 控件 Text: Title AssociatedControlID: TitleText	添加一个 TextBox 控件 ID: TitleText Width: 450px AccessKey: T
2	添加一个 Label 控件 Text: Summary AssociatedControlID: SummaryText	添加一个 TextBox 控件 ID: SummaryText Width: 450px AccessKey: U TextMode: MultiLine
3	添加一个 Label 控件 Text: Body AssociatedControlID: BodyText	添加一个 TextBox 控件 ID: BodyText Width: 450px AccessKey: B TextMode: MultiLine
4	添加一个 Label 控件 Text: Genre AssociatedControlID: GenreList	添加一个 DropDownList 控件 ID: GenreList AccessKey: G
5	添加一个 Label 控件 Text: Authorized AssociatedControlID: Authorized	添加一个 CheckBox 控件 ID: Authorized AccessKey: A
6	保留该单元格为空	添加一个 Button 控件 ID: SaveButton Text: Save AccessKey: S

完成以后，页面应如图 15-13 所示。

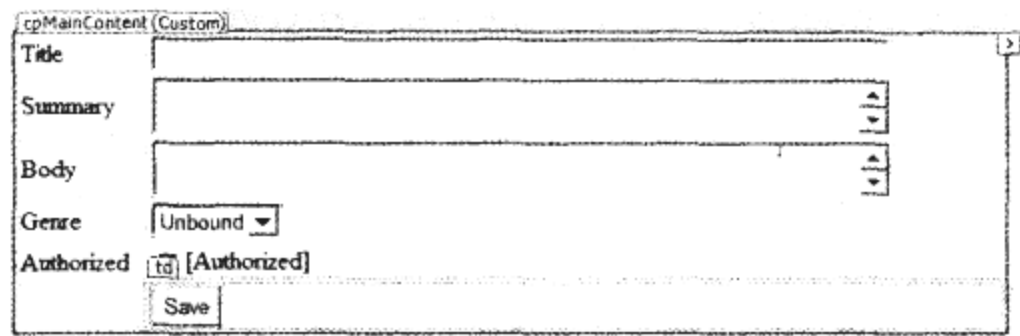


图 15-13

(3) 切换到 Markup 视图，将除了 Summary Label 之外的每个 Label 控件的 Text 属性的首字母放到一对<u>标记内，如下所示：

```
<asp:Label ID="Label1" runat="server" AssociatedControlID="TitleText"
    Text="<u>T</u>itle"></asp:Label>
```

对于 Summary Label，将第二个字母(u)放到一对<u>标记内，因为第一个字母 S 已经用于 Save 按钮。

```
Text="S<u>u</u>mmary"
```

这会为该字母添加下划线，以告诉用户使用哪个访问键来激活相关控件，在这个练习的后面可以看到这一点。

(4) 接下来，将 DropDownList 挂钩到一个新的 EntityDataSource 控件。应该把它绑定到 Genres 实体集，并使用 Id 和 Name 分别作为 DataValueField 和 DataTextField 属性的值。如果在 DataValueField 和 DataTextField 的下拉列表中没有看到 Name 和 Id 属性，可以单击 Refresh Schema 链接。如果不知道如何把控件挂钩到 EntityDataSource，可以参考第 14 章的讨论。完成以后，这两个控件的代码如下所示：

```
<asp:DropDownList ID="GenreList" runat="server" DataSourceID="EntityDataSource1"
    DataTextField="Name" DataValueField="Id" AccessKey="G"></asp:DropDownList>
<asp:EntityDataSource ID="EntityDataSource1" runat="server"
    ConnectionString="name=PlanetWroxEntities"
    DefaultContainerName="PlanetWroxEntities" EnableFlattening="False"
    EntitySetName="Genres"></asp:EntityDataSource>
```

(5) 下一步是编写代码，从数据库获取现有的 Review，以便用户可以进行编辑。当查询字符串包含数据项的 ID 时，页面会假定用户正在编辑该项。否则，页面将假定用户正在创建新评论。

要想建立相关代码，在 Design 视图中双击页面的灰色只读区域，为页面的 Load 事件建立一个处理程序，在页面的顶部为 PlanetWroxModel 名称空间添加一个 Imports/using 语句，然后添加下面突出显示的代码。不要忘记添加在 Page_Load 外部、但是在类定义内部的_id。

```
VB.NET
Imports PlanetWroxModel

Partial Class Management_AddEditReviewHandCoded
    Inherits System.Web.UI.Page
```

```
Dim _id As Integer = -1
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    If Not String.IsNullOrEmpty(Request.QueryString.Get("Id")) Then
        _id = Convert.ToInt32(Request.QueryString.Get("Id"))
    End If
    If Not Page.IsPostBack And _id > -1 Then
        Using myEntities As New PlanetWroxEntities()
            Dim review = (From r In myEntities.Reviews
                           Where r.Id = _id
                           Select r).SingleOrDefault()

            If review IsNot Nothing Then
                TitleText.Text = review.Title
                SummaryText.Text = review.Summary
                BodyText.Text = review.Body
                GenreList.DataBind()
                Dim myItem As ListItem =
                    GenreList.Items.FindByValue(review.GenreId.ToString())
                If myItem IsNot Nothing Then
                    myItem.Selected = True
                End If
                Authorized.Checked = review.Authorized
            End If
        End Using
    End If
End Sub
End Class
```

```
C#
using PlanetWroxModel;

public partial class Management_AddEditReviewHandCoded : System.Web.UI.Page
{
    int _id = -1;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!string.IsNullOrEmpty(Request.QueryString.Get("Id")))
        {
            _id = Convert.ToInt32(Request.QueryString.Get("Id"));
        }
        if (!Page.IsPostBack && _id > -1)
        {
            using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
            {
                var review = (from r in myEntities.Reviews
                              where r.Id == _id
                              select r).SingleOrDefault();
                if (review != null)
                {
                    TitleText.Text = review.Title;
                    SummaryText.Text = review.Summary;
                    BodyText.Text = review.Body;
                    GenreList.DataBind();
                }
            }
        }
    }
}
```



```
        ListItem myItem =
            GenreList.Items.FindByValue(review.GenreId.ToString());
        if (myItem != null)
        {
            myItem.Selected = true;
        }
        Authorized.Checked = review.Authorized;
    }
}
}
```

如果不想输入这些代码,可以从本书附带的源代码(可以从 www.wrox.com 或者 <http://www.tupwk.com.cn/downpage> 上下载)的 Chapter 15 文件夹中找到这些代码。但是,在实际应用程序中,通常也需要输入这些代码,因此与其复制粘贴这些代码,不如找到输入此类代码的最有效方式,即让 IntelliSense 完成大部分工作。

(6) 切换回 Design 视图,双击 Save 按钮为 Button 控件的 Click 事件建立一个处理程序。然后,在 Code Behind 中,为该处理程序添加下面的代码:

VB.NET

```
Protected Sub SaveButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles SaveButton.Click
    Using myEntities As New PlanetWroxEntities()
        Dim myReview As Review
        If _id = -1 Then ' Insert new item
            myReview = New Review()
            myReview.CreateDateTime = DateTime.Now
            myReview.UpdateDateTime = myReview.CreateDateTime
            myEntities.AddToReviews(myReview)
        Else ' update existing item
            myReview = (From r In myEntities.Reviews
                Where r.Id = _id
                Select r).Single()
            myReview.UpdateDateTime = DateTime.Now
        End If

        myReview.Title = TitleText.Text
        myReview.Summary = SummaryText.Text
        myReview.Body = BodyText.Text
        myReview.GenreId = Convert.ToInt32(GenreList.SelectedValue)
        myReview.Authorized = Authorized.Checked

        myEntities.SaveChanges()
        Response.Redirect("Reviews.aspx")
    End Using
End Sub
```

C#

```
protected void SaveButton_Click(object sender, EventArgs e)
{
    using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
```

```
{
    Review myReview;
    if (_id == -1) // Insert new item
    {
        myReview = new Review();
        myReview.CreateDateTime = DateTime.Now;
        myReview.UpdateDateTime = myReview.CreateDateTime;
        myEntities.AddToReviews(myReview);
    }
    else // update existing item
    {
        myReview = (from r in myEntities.Reviews
                    where r.Id == _id
                    select r).Single();
        myReview.UpdateDateTime = DateTime.Now;
    }

    myReview.Title = TitleText.Text;
    myReview.Summary = SummaryText.Text;
    myReview.Body = BodyText.Text;
    myReview.GenreId = Convert.ToInt32(GenreList.SelectedValue);
    myReview.Authorized = Authorized.Checked;

    myEntities.SaveChanges();
    Response.Redirect("Reviews.aspx");
}
}
```

(7) 打开 Management 文件夹中的 Reviews.aspx 文件，将出现两次的 AddEditReview.aspx 替换为 AddEditReviewHandCoded.aspx。其两次出现的位置分别是 Title 的 HyperLinkField 和底部的 Insert New Review 链接。

(8) 通过按下 Ctrl+Shift+S 组合键保存所有未处理的改动。然后在 Solution Explorer 中右击 AddEditReviewHandCoded.aspx 页面，选择 View in Browser 命令。此时，应该会看到一个可以插入新评论的界面，如图 15-14 所示，图中显示的是 Opera 10 中的页面。

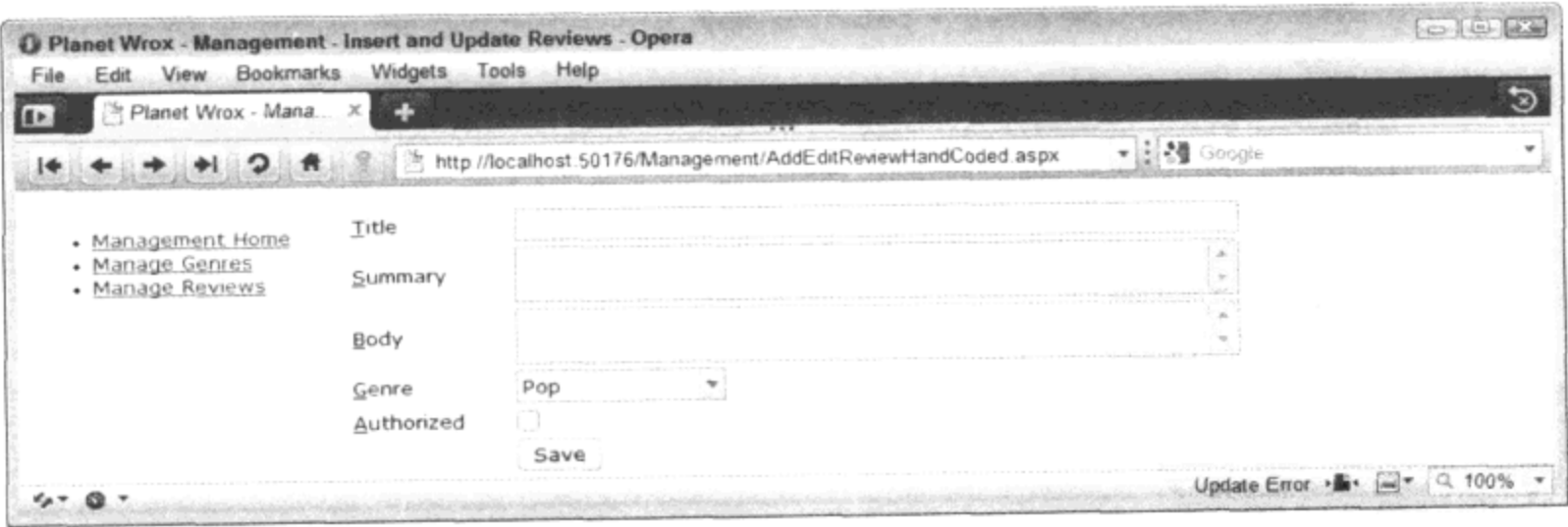


图 15-14

(9) 前面曾经设置了 AccessKey 属性，并给一些字母添加了下划线。结合使用它们可以更方便地访问页面中的控件。在许多浏览器中，按下 Alt 和相关字母键可以使与组合键对应的控件获得焦

点。在本例中，按下 Alt+T 组合键会使 Title 字段获得焦点。通过这种方法，可以在没有鼠标的情况下方便地使用表单。但是，并不是所有的浏览器都使用 Alt+字母。Opera 要求在选择字母以前首先按下 Shift+Esc 键，而 Firefox 则使用 Shift+Alt+Letter。

注意，字母的下划线只是一种视觉表示方法，提示用户可以使用哪种快捷键，就像 Windows 中的程序那样。即使没有下划线，仍然可以使用这种快捷方式。

(10) 输入一个评论，选择一个流派，然后单击 Save 按钮(或者按下 Save 按钮的访问键)。将会再次进入 Reviews 页面。打开流派，定位到评论，然后单击其标题。现在将会打开 AddEditReviewHand-Coded.aspx，其中所有的表单控件都应该已经填写，可以进行编辑。

工作原理

在这个例子中，实际上编写了大量代码，而没有许多使用现成的控件，只是用到了让用户输入一些细节信息和创建流派列表的控件。虽然手动编写代码通常意味着需要做更多的工作，但却可以获得更大的灵活性，并且当操作正确的时候，可以得到更易于维护的页面。在本例中，页面的标记部分要比使用 DetailsView 的前一版本更易于维护。使用这种方法后，就没有了控件上无穷无尽的控件，没有了 InsertItemTemplate 和 EditItemTemplate 之间的重复，也没有了 Code Behind 代码中处理 UpdateDateTime 的棘手的代码。剩下的就是必要的表单控件的一个简单的基于表的表示。就像第 9 章创建的联系信息表单一样，修改这个页面，添加第 9 章介绍的验证控件，以及使用 CSS 改变页面的外观这些操作都很简单。


出现的新内容是在 Label 控件上设置的 AssociatedControlID。当向 Label 控件的这个属性指派一个控件 ID 时，它在浏览器中将变为：

```
<label for="cpMainContent_TitleText"
      id="cpMainContent_Label1"><u>T</u>itle</label>
```

这又会告诉浏览器，当单击 Title 标签的时候，使 Title 文本框获得焦点。服务器控件的 AccessKey 属性已经被赋给它们在客户端的对应元素：

```
<input name="ctl00$cpMainContent$TitleText" type="text"
      id="cpMainContent_TitleText" accesskey="T" style="width:450px;" />
```

当按下相关联的访问键时，浏览器也会使控件获得焦点。



注意：除了自己格式化访问键以外，还可以使用 jQuery 插件完成这项工作。如果对这种解决方案感兴趣，可以看一下 [http://plugins.jquery.com/project/Access Key Highlighter](http://plugins.jquery.com/project/Access+Key+Highlighter) 上的 Access Key Highlighter 插件。

接下来看一下 Code Behind 文件中的代码。首先将讨论如何在数据库中保存一个新评论的表单。然后，将讨论如何从数据库中加载一个现有的评论来预填充表单。

当填写表单的控件时，单击 Save 按钮，SaveButton_Click 方法中的代码将会触发。这些代码首先创建一个新的 PlanetWroxEntities 对象，使用 using 块来使您能够通过 EF 与数据库交互。然后，当_id 变量不包含现有评论的 ID(后面将会看到如何检索 ID)时，将会创建一个新的 Review 实例，并

将其添加到实体对象的 `Reviews` 集合中。当使用 `Insert New Review` 链接创建一个全新的评论时，就会发生这种情况。

VB.NET

```
myReview = New Review()  
myReview.CreateDateTime = DateTime.Now  
myReview.UpdateDateTime = myReview.CreateDateTime  
myEntities.AddToReviews(myReview)
```

C#

```
myReview = new Review();  
myReview.CreateDateTime = DateTime.Now;  
myReview.UpdateDateTime = myReview.CreateDateTime;  
myEntities.AddToReviews(myReview);
```

因为评论需要一个 `CreateDateTime` 属性和一个 `UpdateDateTime` 属性，这段代码将设置它们。注意 `UpdateDateTime` 中填充了 `CreateDateTime`，因此两个属性中将包含相同的日期和时间，表明还没有修改过数据项。

如果 `_id` 变量不包含评论 ID(意味着正在编辑和保存现有评论)，则将使用 LINQ to Entities 查询从数据库中查询 ID。

VB.NET

```
myReview = (From r In myEntities.Reviews  
            Where r.Id = _id  
            Select r).Single()  
myReview.UpdateDateTime = DateTime.Now
```

C#

```
myReview = (from r in myEntities.Reviews  
            where r.Id == _id  
            select r).Single();  
myReview.UpdateDateTime = DateTime.Now;
```

不管是否向该页面传递了一个 ID，在这个阶段 `myReview` 变量都将包含一个评论对象。剩下的代码将通过从相关的表单控件中进行检索来填写评论的属性。对于流派来说，代码直接指派 `GenreId` 属性，而不是查询一个完整的 `Genre` 实例，并把它指派给 `Review` 实例的 `Genre` 属性。之所以能这么做，是因为模型支持外键列(第 14 章介绍了相关内容)。

最后，当完全建立了对象时，代码会调用 `PlanetWroxEntities` 对象的 `SaveChanges` 方法。这会向数据库发送一个 SQL INSERT 或 UPDATE 命令，告诉数据库插入一个新 `Review` 记录，或者更新现有的 `Review` 记录。之后，将使用 `Response.Redirect` 将用户重定向到 `Reviews.aspx` 页面：

VB.NET

```
myEntities.SaveChanges()  
Response.Redirect("Reviews.aspx")
```

C#

```
myEntities.SaveChanges();  
Response.Redirect("Reviews.aspx");
```

显然，在把评论保存到数据库中后，就可以对它进行编辑。在 `Reviews.aspx` 的评论列表中单击现有的一个评论，将会进入查询字符串中包含该评论 ID 的 `Add/Edit` 页面。例如，打开 `http://localhost:1049/Management/AddEditReviewHandCoded.aspx?Id=28` 后，可以编辑 `Metallica` 流派的 `Death Magnetic` 专辑。该页面通过在 `Page_Load` 中使用如下代码检测查询字符串中的 ID：

VB.NET

```
If Not String.IsNullOrEmpty(Request.QueryString.Get("Id")) Then
    _id = Convert.ToInt32(Request.QueryString.Get("Id"))
End If
```

C#

```
if (!string.IsNullOrEmpty(Request.QueryString.Get("Id")))
{
    _id = Convert.ToInt32(Request.QueryString.Get("Id"));
}
```

因为在 `Page_Load` 中这个 `_id` 变量被赋予了一个值，所以它可以用来加载将在表单中显示的现有项，也可以用来在 `SaveButton` 的 `Click` 事件(前面已经看到)中从数据库中获取项。

如果有 ID(`_id` 被赋予了 -1 之外的值)，代码会在 `using` 块内建立一个新的 `PlanetWroxEntities` 实例，并使用下面的 LINQ to Entities 查询 `Review`：

VB.NET

```
Dim review = (From r In myEntities.Reviews
               Where r.Id = _id
               Select r).SingleOrDefault()
```

C#

```
var review = (from r in myEntities.Reviews
               where r.Id == _id
               select r).SingleOrDefault();
```

在数据库中找到 `Review` 后，它的属性将用来预填充表单控件。

VB.NET

```
If review IsNot Nothing Then
    TitleText.Text = review.Title
    SummaryText.Text = review.Summary
    BodyText.Text = review.Body
    GenreList.DataBind()
    Dim myItem As ListItem = GenreList.Items.FindByValue(review.GenreId.ToString())
    If myItem IsNot Nothing Then
        myItem.Selected = True
    End If
    Authorized.Checked = review.Authorized
End If
```

C#

```
if (review != null)
{
```

```
TitleText.Text = review.Title;
SummaryText.Text = review.Summary;
BodyText.Text = review.Body;
GenreList.DataBind();
ListItem myItem = GenreList.Items.FindByValue(review.GenreId.ToString());
if (myItem != null)
{
    myItem.Selected = true;
}
Authorized.Checked = review.Authorized;
}
```

代码在尝试访问评论的属性之前会先检查 `review` 是否是 `Nothing/null`。在本例中，评论为 `null` 的可能性很小，因为我们是通过单击 `Reviews` 页面中的现有项来访问 `Add/Edit` 页面的，所以可以确定该项存在。但是，并不总是这种情况，特别是在面向公众的页面中更不能这么认为。客户端可能保存了某个页面的书签，该书签的查询字符串中有一个特殊的 `ID`。如果之后从数据库中删除了该评论，而用户仍使用老书签访问页面，那么就无法找到页面，导致发生一个 `Null Reference` 异常。

相同的防御性代码编写机制也被用来预选下拉列表中的流派。在这种情况下，可以确定 `Genre` 仍然存在于数据库中，因为 `Genre` 表的 `Id` 列和 `Review` 表的 `GenreId` 之间存在着关系。但是，在尝试从 `DropDownList` 控件中选择一个数据项之前，最好先进行检查，以确定它存在，这样可以避免发生其他 `Null Reference` 异常。因为在这个阶段还没有填充流派的 `DropDownList`，所以需要首先调用 `DataBind()`，强制 `EntityDataSource` 控件获得流派，并把它们添加到 `DropDownList` 中。之后，代码就可以成功地找到并预选合适的数据项。

最后，当单击某个已编辑的对象的 `Save` 按钮后，将触发与插入新项时使用的完全相同的代码。

如果使用验证控件(从第 9 章可以知道，应该这么做)，则需要在保存 `Review` 实例之前检查页面是否有效。

VB.NET

```
Protected Sub SaveButton_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles SaveButton.Click
    If (Page.IsValid) Then
        Using myEntities As New PlanetWroxEntities()
```

C#

```
protected void SaveButton_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
```

这就是使用自己的代码针对 `Entities Framework` 添加和编辑新评论所需做的工作。初看上去，这可能有些让人生畏，因为需要改变自己的思想，忘掉智能控件和它们提供的众多属性与事件，而直接考虑代码。但是，`EF` 使这些工作变得很简单，所采取的大部分操作就是查询实体、复制对象属性的值或者向对象属性赋值，以及通过调用 `SaveChanges` 把改动传回数据库。

显然，这只是个开始。在开始编写自己的代码后，不管是否针对 `EF` 进行编写，面前的可能性都是巨大的。要想了解更多信息，可以参考 `Julia Lerman` 编写的 *Programming Entity Framework Second*

Edition，或者清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C#和 VB.NET(第 7 版)》一书 ISBN：978-7-302-23524-8。

在目前看到的全部数据库示例中，代码在每次请求时都要访问数据库。每次需要显示数据时，就从数据库中获取这些数据。显然，这会浪费时间和 CPU 周期等资源，特别是如果在上次访问后数据并没有改变时更是如此。在本章的最后一节，我们将介绍缓存(caching)技术，这种技术可以显著提高应用程序的响应性及其性能。

15.4 缓存

缓存是提高应用程序性能的一种最好的、通常也是最简单的方法。但是开发人员常常忽视缓存。通过缓存，可以将数据的副本存储到一个可被快速访问的位置。其思想就是从缓存中取出数据比重新生成数据或者从原始数据源中获取这些数据要快。因此，大多数缓存解决方案都将数据存储到内存中，这通常是获取数据的最快方法。.NET 缓存也不例外，它允许将频繁访问的数据存储到计算机内存的一个特定的位置。

通常，缓存原理示意图如图 15-15 所示。

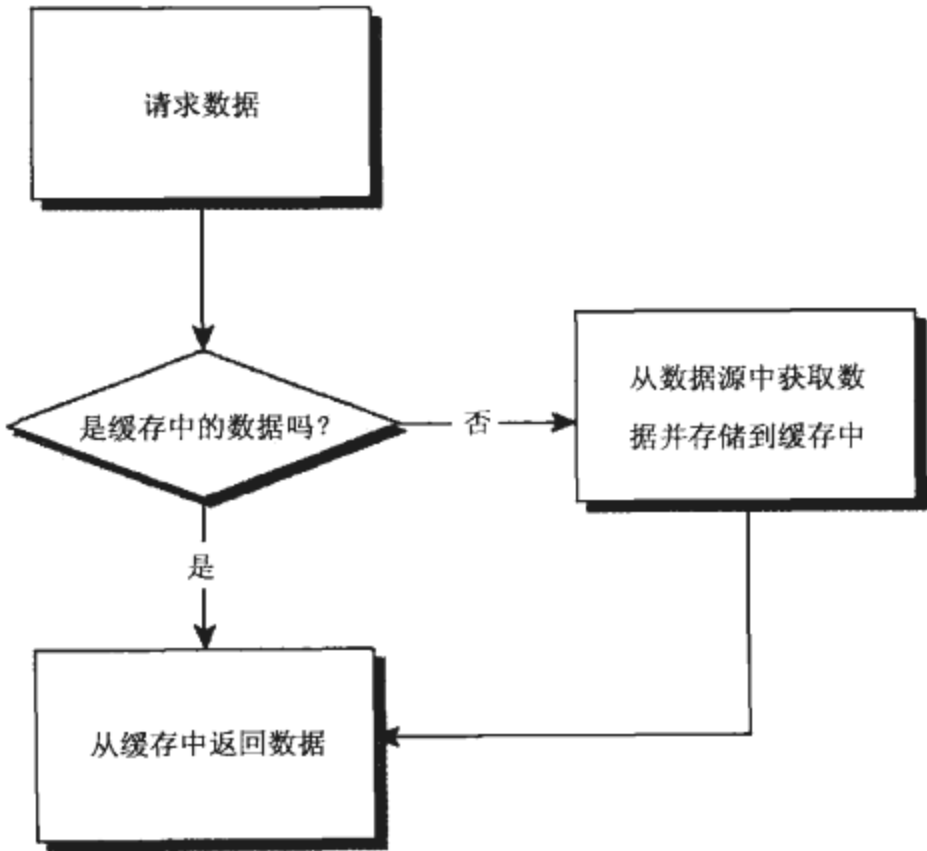


图 15-15

该应用程序从数据库中查询一些数据，例如流派列表。它不是直接访问数据库，而是检查缓存看其是否包含请求的数据。如果包含，就直接从缓存返回数据。如果数据还未存储在缓存中，就从数据源中(如 SQL Server 数据库)检索，并将数据的副本存储在缓存中以备后面的检索使用，最后将数据返回给调用代码。

尽管缓存通常都能较好地改进应用程序，但也有一些需要注意的缺点。下面一节介绍了在使用缓存时会遇到的一些常见问题。之后的一节介绍了在 ASP.NET Web 应用程序中可用的不同缓存机制。

15.4.1 缓存数据的常见问题

在处理缓存的数据时，通常会遇到两种常见的问题：

- 得到陈旧(或者过期)数据。
- 假定数据存在于缓存中，但事实并非如此。

下面将介绍如何避免这两个问题。

1. 避免陈旧数据

陈旧数据是不再与数据源匹配的缓存数据。例如，当为数据库中的所有流派缓存查询结果，并使用这些数据而不是从数据库中获取新数据时，其他用户插入的新流派就可能被忽略。

为了避免出现陈旧数据，需要一种方法来使缓存无效。通过使缓存无效，可将数据项从缓存中删除，然后在下一请求时重新创建数据。使缓存数据无效的方法有很多。首先，可选择设置一个短暂的缓存持续时间。例如，缓存数据库中的 Genres 列表 10 分钟。如果其他用户在这 10 分钟期间插入了新的流派，它将不在 Web 页面上显示。不过，10 分钟后，该列表将从缓存中删除，在下次被请求时用新的数据重新创建。后面将介绍如何使用基于时间的缓存。

使缓存无效的另一种选择是使用缓存依赖项。通过缓存依赖项，可以在缓存的项与原始数据源之间创建关系。在底层数据源改变时，缓存的项无效，这样在它下次被请求时可重新创建。后面将介绍如何使用 CacheDependency。

在使用像 Microsoft SQL Server 这样的数据库时也可以使用缓存依赖项。这意味着，一旦缓存查询的一部分数据改变，缓存项就会失效。数据库缓存机制和失效是较宽泛的高级主题。清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C# 和 VB.NET(第 7 版)》一书(ISBN: 978-7-302-23524-8)的有关缓存的章节中谈到了此内容。

2. 不要依赖缓存中的数据

为了最小化应用程序占用的内存，ASP.NET 中的缓存机制会自动从缓存中删除旧数据和不经常用的数据。因此，不能依赖缓存中的数据。它们会被缓存删除，是因为 ASP.NET 运行库确定它们不经常使用，因而在无谓地占用宝贵空间。当 Web 应用程序或 Web 服务器重启(例如，当修改 web.config 文件时就会发生这种情况)时，会清除整个缓存。数据项也会因为它们的依赖性而被删除。因此，不要依赖于存储在缓存中的项，即使您自己在应用程序生命周期的初期放入了这些数据。本章的后面将介绍如何通过 Cache API(Application Programming Interface，应用程序编程接口；可与程序交互的方法)以编程形式使用缓存。

使用缓存 API 并不是使用缓存的唯一方法。下一节将介绍在 ASP.NET 中缓存数据的不同方法。

15.4.2 在 ASP.NET Web 应用程序中缓存数据的不同方法

可以在 ASP.NET Web 应用程序中部署一些不同的缓存策略，包括输出缓存、使用内置数据源控件缓存和编程缓存。下面将分别进行讨论。

1. 输出缓存

所谓输出缓存，就是缓存呈现页面的最终结果。这意味着在页面首次被请求时，其最终结果就

被添加到缓存中。对于对该页面的后续请求，将发送相同的 HTML。后面这句话非常重要，所以再重复一次：对于对该页面的后续请求，将发送相同的页面。这意味着服务器将不会再次处理该页面，也不会触发 Code Behind 中的自定义代码。来自第一次请求的相同 HTML 将在每次后续请求时返回。

启用输出缓存十分简单，只需在 Page 指令下边添加一个 OutputCache 指令，如下面的 C# @ Page 指令所示(如果使用的是 VB.NET，可以使用相同的代码，但是 @ Page 指令稍有不同)：

```
<%@ Page Title="About this Site" Language="C#"
    MasterPageFile="~/MasterPages/Frontend.master" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="About_Default" %>
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

Duration 是在 ASP.NET 创建页面的新副本之前想要缓存该页面的秒数。在前面的示例中，页面将缓存 1 分钟。

VaryByParam 特性的值 None 告诉.NET 缓存页面的单个版本，而不管传递给它的查询字符串值是多少。虽然对于相对静态的页面，如 Planet Wrox 站点中的 About 页面，这可能很适合，但对于动态页面则不然。例如，假定有一个动态页面根据传递到页面的查询字符串显示评论的细节。第一次请求页面时，可能浏览如下的地址：

```
http://localhost:12345/Reviews/ViewDetails.aspx?Id=23
```

ASP.NET 生成显示 ID 为 23 的评论的页面，然后缓存页面的整个输出。接着请求下面的页面时会发生什么情况呢？

```
http://localhost:12345/Reviews/ViewDetails.aspx?Id=33
```

看到的不是 ID 为 33 的评论，而仍然是 ID 为 23 的评论。为了解决这一问题，ASP.NET 支持缓存页面的特定版本。例如，可以让页面为它检索的每个独特的查询字符串字段缓存页面的副本。这可以通过将指令的 VaryByParam 特性设置为可能的查询字符串或表单值的一个逗号分隔的列表来实现。ASP.NET 将为它找到的每种独特的字段组合缓存页面的副本。例如，考虑这样一个页面，它接受查询字符串中评论的 ID，然后显示评论的细节。要缓存每个独特评论的副本，可以像下面这样向 VaryByParam 特性添加一个 Id：

```
<%@ OutputCache Duration="60" VaryByParam="Id" %>
```

对于显示某个特定评论的细节的页面来说，这样很好。对于每个独特的评论，ASP.NET 都保存一个缓存的副本。这意味着数据库将只在首次请求特定评论时被命中；后续请求将由缓存来处理。

这种输出缓存的一个问题是处理的情况有些极端。虽然根据查询字符串值缓存不同的页面很简单，但是需要编写一些自定义代码来处理像主题这样的其他情况。当首次请求和缓存页面时，将考虑用户的主题。即使后续用户使用了不同的主题，他们仍然看到的是使用最初请求的主题的页面。虽然可以通过编程方式改变这种行为，但是其实还有一种更简单的解决方案：让数据源控件缓存数据，下一节将讨论这些内容。对于没有这种问题的场合(例如没有使用主题)，输出缓存是改进站点性能的一种优秀的机制。

2. 用数据源控件缓存数据

用数据源控件缓存数据的最大好处在于它们只缓存动态的、数据库驱动的数据，而不是整个页面。这样就可以使页面的其他部分保持动态，如横幅模块或欢迎用户的个性化问候。除了 SiteMapDataSource、LinqDataSource 和 EntityDataSource 控件外，所有的数据源控件都支持缓存。

使用数据源控件进行缓存很简单：所需做的就是设置 EnableCaching 属性，然后指定一个 CacheDuration。下列代码段显示了一个缓存数据达 10 分钟的 SqlDataSource 控件。

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" CacheDuration="600"
    EnableCaching="True"></asp:SqlDataSource>
```

使用数据源控件进行缓存的优点是它们可以智能地知道您是否正在对底层数据作更新。因此，如果建立了一个 SqlDataSource 控件为 SelectCommand 缓存数据 20 分钟，但之后使用 InsertCommand、UpdateCommand 或 DeleteCommand 对数据进行了更改，则缓存会自动无效。这只在对同一 SqlDataSource 执行插入、更新或删除命令时才有效。如果有一个页面(如 All.aspx)显示和缓存了评论列表，然后用另一页面(如 Management 文件夹中的 AddEditReview.aspx)插入一条新评论，这就不会起作用。在站点的 Management 部分插入一条新评论后，在缓存到期之前，它不会在 All.aspx 中显示。

除了 CacheDuration 和 EnableCaching 属性外，数据源控件提供了更多缓存选项。清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C# 和 VB.NET(第 7 版)》一书(ISBN: 978-7-302-23524-8)详细介绍了 ASP.NET 4 中的高级缓存功能。

使用数据源控件进行免写代码的缓存在许多情况下是有用的。不过，数据源控件不能适用于所有的情形。如果想缓存从不同数据源获取的数据的结果，该怎么办？如果想要缓存针对 Entity Framework 手动编写代码的页面的结果，或者想缓存需要频繁访问的文本或 XML 文件的内容，又该怎么办？对于这些问题，ASP.NET 支持以编程方式访问缓存。

3. 编程缓存

通过编程缓存，可以使用 VB.NET 或 C#代码将数据项存储到缓存中。显然，后面还可以再次访问它们。要存储数据项，可使用 Add 或 Insert 方法或直接索引 Cache 集合。Add 方法是非常强大和复杂的，允许指定许多选项来确定某数据项缓存多长时间、与其他缓存项相比又有什么优先级，以及将该项从缓存中删除时所依据的条件。

而 Insert 方法更简单。它有一些简单的重载，允许指定缓存项和将其与特定键相关联。在本节前面讨论缓存时看到了这样的一个示例。还有一个重载允许定义依赖关系，当原始数据源发生变化时，这种依赖关系可以用来使缓存项无效。对于缓存不经常改变的文件，这种方法很方便。可以从磁盘上读取一个文件，然后将该文件存储到缓存中，使其与原始文件具有依赖关系。然后就可以从缓存中不断读取文件，而不必重新访问磁盘。当磁盘上的文件改变时，缓存项将自动无效，此时可以再次读取原始源文件并把它存储到缓存中。下面的示例显示了如何修改 ContactForm.ascx 用户控件以在数据发生变化前在缓存中存储和读取数据项。

VB.NET

```
If Page.IsValid Then
    Dim mailBody As String = TryCast(Cache("ContactFormMailBody"), String)
    If String.IsNullOrEmpty(mailBody) Then
```

```
Dim fileName As String = Server.MapPath("~/App_Data/ContactForm.txt")
mailBody = System.IO.File.ReadAllText(fileName)
Cache.Insert("ContactFormMailBody", mailBody, New CacheDependency(fileName))
End If
mailBody = mailBody.Replace("##Name##", Name.Text)

...
End If

C#
if (Page.IsValid)
{
    string mailBody = Cache["ContactFormMailBody"] as string;
    if (string.IsNullOrEmpty(mailBody))
    {
        string fileName = Server.MapPath("~/App_Data/ContactForm.txt");
        mailBody = System.IO.File.ReadAllText(fileName);
        Cache.Insert("ContactFormMailBody", mailBody, new CacheDependency(fileName));
    }
    mailBody = mailBody.Replace("##Name##", Name.Text);

    ...
}
```

注意，代码中创建了一个新的 CacheDependency(需要使用 Imports/using 语句导入 System.Web.Caching 名称空间)，并把它传递给 Insert 方法。CacheDependency 接受它所依赖的文件的文件名。当使用 VWD 或者记事本等改变了磁盘上的文件后，ASP.NET 会从缓存中删除该数据项，这样，下次执行代码时，将从原始源文件中再次读取数据。

要从缓存删除项，可使用接受缓存项的键的 Remove 方法。在使用 Add 或 Insert 方法插入项时会定义该键。

要访问缓存中的项，还有一些可用的选项。首先，可以直接访问 Cache 集合。

```
VB.NET
myReview = TryCast(Cache(myKey), Review)
```

```
C#
Review myReview = Cache[myKey] as Review;
```

这里，使用 myKey 变量中存储的键对 Cache 集合进行索引。
另外，可使用接受键的 Get 方法：

```
VB.NET
myReview = TryCast(Cache.Get(myKey), Review)
```

```
C#
myReview = Cache.Get(myKey) as Review;
```

由于 Get 是一个方法，所以 C#示例对这个缓存键使用了括号，使两个示例看起来更像。
最后，可以使用 Item 属性访问缓存中的项，该属性也接受缓存项的键。

访问缓存中项的这三种方法总是返回一个 object。这意味着，如果知道从缓存中获取的数据的类型，就可以在使用它的属性前将它强制转换为适当的类型(使用 VB.NET 中的 TryCast，或者 C# 中的 as 关键字)。前面两个示例显示了如何首先将从缓存中获取的项强制转换为强类型的 Review 对象。

为了了解如何以编程方式使用缓存，下面的“试一试”练习展示了如何在缓存中插入评论，这样就不必在每次需要它时从数据库中读取。

试一试

使用缓存 API

在这个“试一试”练习中，将看到如何缓存使用 LINQ to Entities 查询从 EF 中获取的 Review 实例，以便在后面可以使用它的键(包含评论的 ID)检索它。

(1) 首先在 Web 应用程序的 Reviews 文件夹中添加一个名为 ViewDetails.aspx 的新页面。确保它基于自定义的模板。不需要设置标题，因为后面将通过编程方式设置。所以，从 Page 指令中删除 Title="" 特性。在 ASP.NET 中有一个存在已久的 bug，当在 Markup 视图中将这个特性设为空字符串时，它会使得对页面的 Title 所作的编程修改无法生效。

(2) 在 Markup 视图中，添加 3 个 Label 控件到 cpMainContent 内容占位符，并进行如下命名：

- TitleLabel
- SummaryLabel
- BodyLabel

删除这 3 个标签的 Text 特性及其值。

(3) 将 TitleLabel 标签包装到<h1>元素中，并将 SummaryLabel 控件的 CssClass 属性设置为 Summary。最终代码如下所示：

```
<h1><asp:Label ID="TitleLabel" runat="server"></asp:Label></h1>
<asp:Label CssClass="Summary" ID="SummaryLabel" runat="server"></asp:Label>
<asp:Label ID="BodyLabel" runat="server"></asp:Label>
```

(4) 切换至 Design 视图并双击页面的只读区域，为 Page_Load 事件建立事件处理程序。

(5) 在页面顶部为 PlanetWroxModel 名称空间添加一个 Imports/using 语句，就像前面对 AddEditReviewHandCoded.aspx 所做的那样。然后为 Page_Load 事件处理程序添加下列代码：

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Dim reviewId As Integer = Convert.ToInt32(Request.QueryString.Get("ReviewId"))
    Dim myReview As Review = TryCast(Cache("Reviews" + reviewId.ToString()), Review)
    If myReview Is Nothing Then
        Using myEntities As New PlanetWroxEntities()
            myReview = (From r In myEntities.Reviews
                Where r.Id = reviewId
                Select r).SingleOrDefault()
        End Using
        If myReview IsNot Nothing Then
            Cache.Insert("Reviews" + reviewId.ToString(), myReview, Nothing,
                DateTime.Now.AddMinutes(20),
                System.Web.Caching.Cache.NoSlidingExpiration)
        End If
    End If
```



```
End Using
End If

If myReview IsNot Nothing Then
    TitleLabel.Text = myReview.Title
    SummaryLabel.Text = myReview.Summary
    BodyLabel.Text = myReview.Body
    Title = myReview.Title
    MetaDescription = myReview.Summary
End If
End Sub

C#
protected void Page_Load(object sender, EventArgs e)
{
    int reviewId = Convert.ToInt32(Request.QueryString.Get("ReviewId"));
    Review myReview = Cache["Reviews" + reviewId.ToString()] as Review;
    if (myReview == null)
    {
        using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
        {
            myReview = (from r in myEntities.Reviews
                        where r.Id == reviewId
                        select r).SingleOrDefault();

            if (myReview != null)
            {
                Cache.Insert("Reviews" + reviewId.ToString(), myReview, null,
                            DateTime.Now.AddMinutes(20),
                            System.Web.Caching.Cache.NoSlidingExpiration);
            }
        }
    }

    if (myReview != null)
    {
        TitleLabel.Text = myReview.Title;
        SummaryLabel.Text = myReview.Summary;
        BodyLabel.Text = myReview.Body;
        Title = myReview.Title;
        MetaDescription = myReview.Summary;
    }
}
```

(6) 打开 Reviews 文件夹中的 All.aspx 页面，删除在第 14 章创建的 GridView。用一个简单的 Repeater 控件代替它，该控件包含一个指向新的详细信息页面的 Hyperlink。

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <asp:Repeater ID="Repeater1" runat="server">
        <ItemTemplate>
            <asp:HyperLink ID="HyperLink1" runat="server"
                NavigateUrl='<%# "ViewDetails.aspx?ReviewId=" + Eval("Id").ToString() %>'
                Text='<%# Eval("Title") %>'></asp:HyperLink>
            <br />
        </ItemTemplate>
    </asp:Repeater>
</asp:Content>
```

```
</ItemTemplate>
</asp:Repeater>
</asp:Content>
```

(7) 切换至页面的 Code Behind 文件, 将使用 GridView 的最后两个调用替换为运用 Repeater 控件。

VB.NET

```
Repeater1.DataSource = allReviews
Repeater1.DataBind()
```


C#

```
Repeater1.DataSource = allReviews;
Repeater1.DataBind();
```

(8) 打开 Styles 文件夹中的 Styles.css, 并在文件末尾添加下面的 CSS 声明。

```
.Summary
{
    font-style: italic;
    display: block;
}
```

(9) 保存所有更改, 然后在浏览器中请求 Reviews 文件夹中的 All.aspx 页面。单击评论的标题, 您将被带到 ViewDetails.aspx 页面, 此时查询字符串中带有被请求评论的 ID。



常见错误: 如果看到有关页面标题无效的错误消息, 要确保在 ViewDetails.aspx 页面的 Page 指令中删除了 Title=""。当把该特性设为一个空字符串时, 在 Code Behind 中设置的标题不会生效, 而 BasePage 则会因为缺失标题而引发一个异常。

现在应该可以看到页面中显示的评论的详细信息。按下 Ctrl+F5 键或者 Ctrl+R 键来刷新页面的内容。虽然看不到区别, 但是评论现在是来自缓存。如果想要确认这个事实, 可以在 ViewDetails.aspx 页面中添加一个 Label 控件, 然后根据数据项是否位于缓存中, 使用不同的文本更新控件。

VB.NET

```
Label1.Text = "In the cache"
If myReview Is Nothing Then
    Label1.Text = "NOT in the cache"
Using myEntities As New PlanetWroxEntities()
```

C#

```
Label1.Text = "In the cache";
if (myReview == null)
{
    Label1.Text = "NOT in the cache";
    using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
```

这段代码先将 Label 的 Text 设为 "In the cache"。但是, 如果没有找到数据项, 则更新 Label 控件来反映这种结果。



常见错误：如果在页面上出现错误“Sequence contains no elements”，请确保在 ReviewId 查询字符串参数中为 ViewDetails.aspx 页面传递了有效的 Review ID。当使用 Ctrl+F5 键从 VWD 中直接浏览详细信息页面，而不是借助于 All.aspx 页面时，常常会发生这种错误。

工作原理

在页面 ViewDetails.aspx 中，首先添加一些标签(label)来保存 Review 的相关属性，例如其 Title、Summary 和 Body。代码向摘要信息的 Label 控件添加了一个 CssClass，以便可以通过 CSS 文件改变其样式。为 Summary 选择器指派了斜体字体，并将 display 属性设为 block，强制后面的正文显示在单独的行中。

然后，Code Behind 文件中的代码首先查看是否可以从缓存中检索数据。

VB.NET

```
Dim myReview As Review = TryCast(Cache("Reviews" + reviewId.ToString()), Review)
```

C#

```
Review myReview = Cache["Reviews" + reviewId.ToString()] as Review;
```

代码使用单词 Reviews 和每一个缓存项的 Id 作为该项的键。这样，缓存中的每个评论就有了一个唯一的键。如果在缓存中没有找到该项(可能是因为第一次加载页面，或者 ASP.NET 删除了该项)，VB.NET 中的 TryCast 方法和 C#中的 as 关键字就会返回 Nothing/null。因此，通过对 myReview 变量检查该值，可以确定该项是否存在于缓存中。如果存在，那么工作就完成了。但是如果缓存中没有该数据，那么就需要使用与前面所示类似的 LINQ to Entities 查询从数据库中获取数据。注意，此查询使用 SingleOrDefault()操作符将查询限制为单个 Review 实例，因为只能有一个实例具有给定的 ID；或者当没有找到该数据项时，就返回 Nothing/null。至于在没有找到数据项时采取哪些操作，则完全由您决定。可以选择在 Label 控件中显示一个错误消息，通知用户该数据项已经不再可用；也可以重定向到主页或站点内的另一个页面。

从数据库中检索了数据项以后，将使用下面的代码把它插入到缓存中：

VB.NET

```
Cache.Insert("Reviews" + reviewId.ToString(), myReview, Nothing,
    DateTime.Now.AddMinutes(20), System.Web.Caching.Cache.NoSlidingExpiration)
```

C#

```
Cache.Insert("Reviews" + reviewId.ToString(), myReview, null,
    DateTime.Now.AddMinutes(20), System.Web.Caching.Cache.NoSlidingExpiration);
```

Insert 方法的第一个参数是缓存键，第二个参数是想要缓存的对象：在本例中是实际的 Review 实例。第三个参数定义了绝对过期日期(absolute expiration date)：认为数据项过期，因而必须从缓存

中删除的日期和时间。在本例中，这个日期是通过向当前日期和时间加上 20 分钟构建的，表明数据项最多缓存 20 分钟。最后一个参数可以用来在每次访问数据项时延长过期时间。这是缓存经常使用的数据项和确保不常使用的数据会更快地从缓存中清除的一种有效的方法。但是，本例使用的是绝对过期日期，这意味着必须传递常量值给 `System.Web.Caching.Cache.NoSlidingExpiration` 作为可调过期(sliding expiration)参数的值，因为这两个参数是互斥的。

在现在这个阶段，如果数据库中包含该项，那么就有一个有效的 `Review` 实例，而不管它是否来自缓存。这个实例将被用来填充页面的 `Label` 控件，以及页面的 `Title` 和 `MetaDescription` 属性。

VB.NET

```
TitleLabel.Text = myReview.Title
...
Title = myReview.Title
MetaDescription = myReview.Summary
```

C#

```
TitleLabel.Text = myReview.Title;
...
Title = myReview.Title;
MetaDescription = myReview.Summary;
```

设置 `Title` 和 `MetaDescription` 有助于用户明白发生的情况，也有助于搜索引擎对页面进行排名。在收藏页面时会使用标题，所以一个清晰的标题可以帮助用户再次找到您的页面。Google 和 Bing 等搜索引擎使用标题来评估页面的内容。它们使用在 `MetaDescription`(最终成为页面 HTML 中<head>部分的<meta name="description" />)中设置的文本将搜索结果呈现给用户。这意味着在这里输入的文本是您与使用搜索引擎的用户的的第一次接触。因此，这些文本是非常重要的信息。不用为了这个目的而重用 `Summary` 属性，而是可以向数据库中的 `Reviews` 表添加额外的一个列(例如，命名为 `SearchEngineDescription`)。然后，右击 `Entity Designer` 中的 `EDMX` 模型关系图，并选择 `Update Model from Database` 命令，将这个列添加到 `ADO.NET Entity Data Model` 中。完成添加以后，不要忘记修改 `Management` 部分的编辑页面(`AddEditReview.aspx` 或 `AddEditReviewHandCoded.aspx`)，使它们也支持这个新属性。最后，在 `ViewDetails.aspx` 页面中，可以将它的值赋给 `Page` 类的 `MetaDescription` 属性。

除了新的 `MetaDescription` 以外，在 ASP.NET 4 中，还使用了一个 `MetaKeywords` 属性扩展了 `Page` 类。这个属性的工作方式与 `MetaDescription` 多少有些相似，可以用来设置页面的关键字。虽然对于关键字在影响搜索引擎排名方面的重要性颇有争议(一些人认为搜索引擎根本不使用它们)，但是设置它们总没有坏处。可以按照为 `SearchEngineDescription` 属性列出的步骤，向数据库和模型添加关键词。关于搜索引擎优化(Search Engine Optimization, SEO)的更多提示，可以参考 Wrox 出版社出版的 `Professional Search Engine Optimization with ASP.NET: A Developer's Guide to SEO` 一书 ISBN: 978-0-470-13147-3。

`ViewDetails.aspx` 页面现在是一个高效的页面：第一次加载时，将从数据库中检索数据，然后把数据存储到缓存中。在后续对页面的访问中，将不再访问数据库，而是从更快的缓存中检索数据。

15.5 有关数据的实用提示

这里有一些有关在 ASP.NET Web 站点中使用数据的实用提示：

- 在使用数据绑定控件的众多样式属性时，考虑使用 `CssClass` 属性，而不是直接设置单独的样式属性。
- 15.2.1 一小节的“试一试”练习，介绍了如何显示各种事件和事件发生的顺序。您可以扩展该示例，为更多的事件编写代码。另外，也可以向页面添加更多的控件，并处理它们的事件，以更深入地理解这些事件。由于清楚理解这些事件和它们发生的顺序对于编写 Web 应用程序来说很关键，因此花些时间进行研究是值得的。
- 在编写访问数据库或其他缓慢或稀有资源(如文件或 Web 服务)的页面时，考虑使用缓存是否有益。尽管在后期添加缓存并不难，但是最好还是尽早添加。
- 考虑手动编写复杂的数据访问页面的代码。虽然一开始可能会很难编写，但是最终可以获得在长期内更易于维护的页面。

15.6 本章小结

本章介绍了一些使用数据控件表现数据的高级主题。前面的章节有意跳过了这些主题，以便使您能够更集中于核心的数据访问概念。

本章首先展示了大部分数据绑定控件所具有的大量样式元素，通过它们可以改变这些控件的外观。然后学习了控件可以触发的各种事件。这些事件可以用来以编程方式改变控件的外观。因此，深刻理解页面的生命周期十分重要。

本章最后讨论了 ASP.NET 支持的各种缓存功能，以帮助改进 Web 站点的性能。

最后这一部分还讨论了在 ASP.NET Web 应用程序中处理数据的一些高级主题。第 16 章将讨论如何通过实现 ASP.NET 安全机制来防止未授权用户访问数据(例如 Management 文件夹)。

15.7 练习

1. 假设有一个简单的 Web 窗体，它上面有一个 Button。如果在浏览器中单击此 Button，会导致一个回发并在服务器上触发其 Click 事件。首先会发生什么事件？是页面的 Load 事件还是 Button 控件的 Click 事件？
 2. 现在，当插入或者编辑 `AddEditReviewHandCoded.aspx` 上的项时，将在完成插入或者编辑工作后被带回 `Reviews.aspx`。如果已经在下拉列表中预选了新添加的评论或者更新后的评论的流派，那么会很方便。如何编写这种功能？
 3. 如何在 Code Behind 中避免在数据绑定控件的事件中处理的异常出现在页面中？
- 练习的答案见附录 A。

本章要点回顾

访问键	一种为 HTML 控件分配字母，从而可以通过按下包括分配的字母键的组合键使控件获得焦点的方式
ASP.NET 样式	从 Style 类继承的控件属性，可以改变控件的外观
缓存	用来在一个可以比访问原始源文件更快的位置存储数据副本的一种技术，可以改进性能
缓存失效	当数据项不再有效时，从缓存中删除数据项的一种机制
异常	.NET 中用于表示错误的术语。第 18 章将详细讨论异常
外键约束错误	当尝试删除其他记录所依赖着的记录时，在数据库级发生的错误
MetaDescription MetaKeywords	Page 类的这两个属性(都是在 ASP.NET 4 中引入的)允许在浏览器中设置页面可以被搜索引擎使用的元数据
输出缓存	一种缓存方式，缓存整个页面或者用户控件，以防止在每次访问时重新生成整个页面或用户控件
陈旧数据	一些不再精确表示原始数据的无效的数据缓存副本

本章要点

- 在处理安全性时会遇到的重要术语
- 启用 ASP.NET 的安全模型的 ASP.NET 应用程序服务
- 如何让用户注册站点的账户
- 如何让用户重置他们的密码或请求新的密码
- 如何在开发时管理数据库中的用户和角色
- 如何根据系统中用户的访问权限向不同用户展示不同内容

到目前为止，在 Web 站点中创建的页面可被所有访问者访问，但还没有阻止特定用户访问某些资源(如 ASPX 文件或整个文件夹)的方法。这意味着，当前任何人都能够访问 Management 文件夹并篡改系统中的流派和评论。

显然，您不希望这种情况在产品 Web 站点中出现。因此，需要考虑用一种安全策略来阻止不受欢迎的用户访问特定内容。还需要考虑一种机制，允许用户注册新账户，同时指定特定用户为 Web 站点的管理员并授予他们特殊的访问权限。

ASP.NET 4 包含了创建可靠而且安全的安全机制所需的所有工具。本章将介绍如何在 ASP.NET 网站中使用这些工具。

在介绍 ASP.NET Framework 中如何实现安全性之前，您需要理解一些在安全性的讨论中会遇到的一些重要术语。

16.1 关于安全性

安全性是一个非常复杂的主题，它通常涉及 3 个很直接的问题：

- 您是谁？
- 如何证明您是谁？
- 允许您在系统中做什么？

16.1.1 身份：您是谁

身份是您所具有的一些特征。具体是什么取决于其所在的上下文。作为一个国家的公民，您的身份涉及您的外貌、法定名字、生日，甚至是社会保障号码。不过，对于像 p2p.wrox.com——Wrox 社区网站这样的 Web 站点，您的身份可能就是您的电子邮件地址。

不管身份包括什么内容，它就是一种表示您的方式。但其他人如何知道您呢？例如，当您登录到 Web 站点时如何知道这就是您本人呢？这就要用到身份验证。

16.1.2 身份验证：如何证明您是谁

身份验证就是提供您是谁的证据。当需要注册借书证时，您需要出示您的证件来证明您所注册的名字。对于像 p2p.wrox.com 这样的网站，您需要提供电子邮件地址和密码。这两者就构成了证明您身份的证据。有许多其他身份验证机制，包括高科技的指纹或虹膜扫描、智能卡和令牌(其中，证据存储在有形物质上)等。不过，因为我们只讨论 ASP.NET 网站的安全性，所以本章只将用户名和密码用于身份验证。

16.1.3 授权：允许您做什么

根据您的身份，系统会授予您或多或少的访问特定区域的权限。例如，可以想像一下动作片里高度安全的国家安全局总部。即使主角允许进入该大楼，他通常也不能进入一些特定区域，因为他没有适当的权限(电影中，他们对系统采用黑客技术，2 分钟内就成功进入那些区域的情况，不在此讨论范围内)。

为了确定允许用户做什么，系统需要了解两件事：当前用户的权限和用户尝试访问的资源的授权规则。

用户的权限基于其用户名(它表示的身份)和对可以选择指派给用户的角色(或安全组)。同样，可以针对特定用户或角色打开或关闭资源。当当前用户和用户尝试访问的资源的访问规则匹配时，就允许用户访问。如果用户被特意阻止了，那么访问就被拒绝。例如，假设文件只能被用户 Tom 和 Developers 组访问。用户 Tom 可以访问该文件，而不管他是否在 Developers 角色中。同时，用户 Charlotte 要访问该文件，就必须在 Developers 角色中。

本章剩余部分将介绍如何运用这些概念。

ASP.NET 中的这些安全概念是使用下面将介绍的应用程序服务(application service)实现的。

16.1.4 ASP.NET 应用程序服务

ASP.NET 2.0 之前的版本都有某种安全性支持。不过，它们都缺乏 ASP.NET 2.0 和之后的版本中提供的高级控件和概念。在 ASP.NET 1.x 应用程序中，需要编写大量代码来实现可靠的安全策略。编写这一代码的不利之处在于它在所有 Web 站点中几乎是一样的。您却不得不重复编写相同的代码来实现安全机制。

而自从应用程序服务随着 ASP.NET 2.0 一起问世，这些问题就得到了解决。应用程序服务是可以在 Web 应用程序中用来支持用户、角色和配置文件等的管理的一组服务。ASP.NET 4 中仍然大量存在这类服务。

ASP.NET 4 中包含了大量应用程序服务，其中最重要的是如下几个：

- 成员(Membership)：可以管理和使用系统中的用户账户

- 角色(Role): 可以管理用户被指派的角色
- 配置文件(Profile): 允许将用户特定的数据存储到后台数据库

图 16-1 列出了这些服务的概览，显示了它们如何与 Web 站点和服务可能使用的底层数据存储相关。

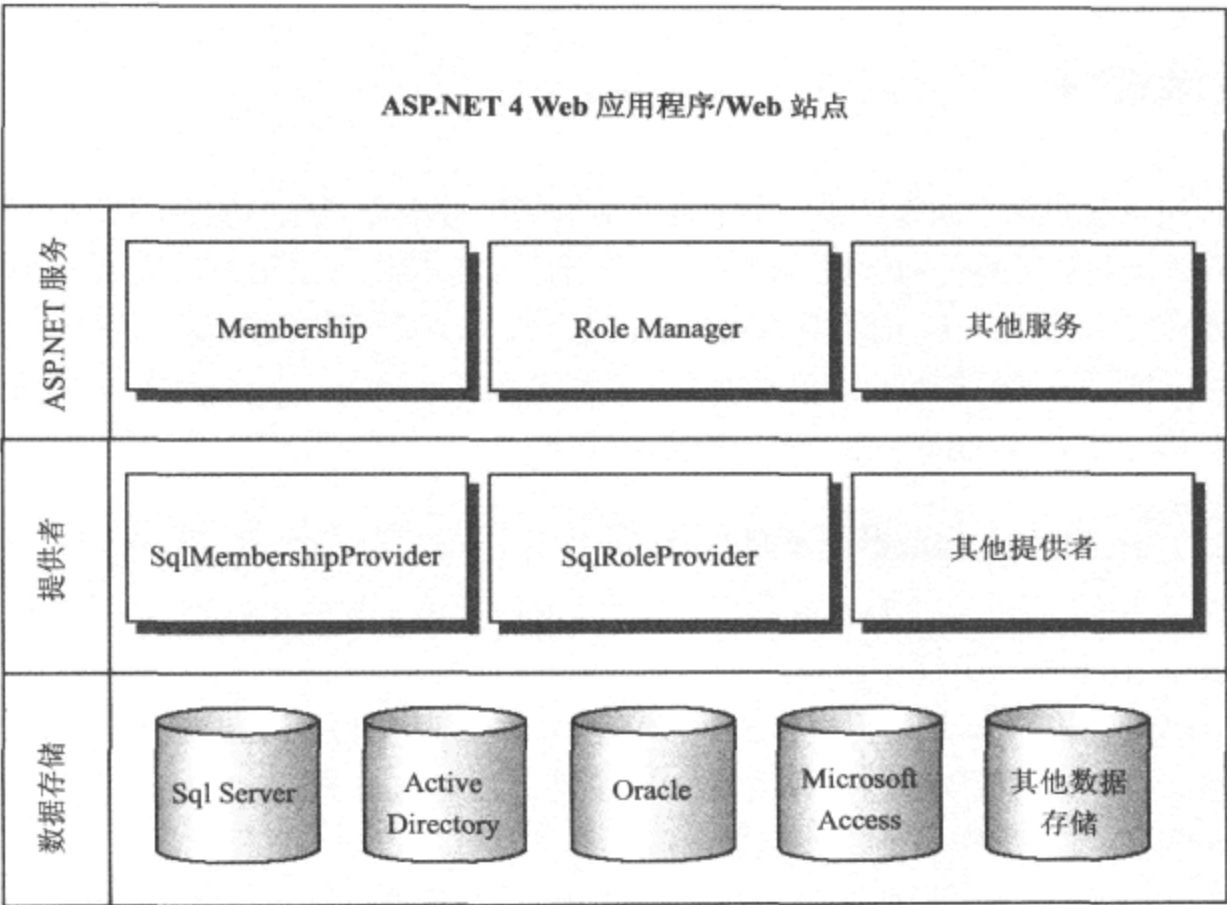


图 16-1

在图 16-1 的顶部，可看到表示您所构建的 Web 站点的 ASP.NET 4 网站和 Web 应用程序。这些网站可以包含控件，如可与 ASP.NET 应用程序服务(如成员和配置文件)通信的登录控件(后面会讨论)。为了创建一个灵活的解决方案，这些服务不直接与底层数据源通信，而是和配置好的提供者(provider)通信。提供者是软件中可用于特定任务的可交换部分。例如，在成员服务中，成员提供者设计为对底层数据存储中的用户起作用。可以根据需要为相同的应用程序服务配置不同的提供者。例如，ASP.NET 包含了一个 SQL Server 提供者，允许成员服务与 SQL Server 数据库通信。这对于与 Internet 连接的网站(如 PlanetWrox.com)来说，SQL Server 提供者是理想的提供者。它还包含了一个 Active Directory 提供者，可创建和管理 Windows 上 Active Directory 中的用户。这一提供者适合于不公开的网络，如内部网。如果用户有自己定制的数据存储，则可以编写自己的提供者并插入 Web 站点，以轻松地替换默认提供者。这些提供者及其底层模型的优势在于，可以通过配置交换它们，而不需要对程序代码进行修改。

每个提供者都需要一个数据存储(如图 16-1 的底部所示)。每个提供者都编写为与特定数据存储一起使用。例如，SQL Server 成员提供者(用于处理例如创建用户、登录和重置密码这样的成员服务)和 SQL Server 角色提供者(用于处理与角色相关的任务)设计为与 Microsoft SQL Server 数据库一起使用。

在本章剩余部分，将介绍如何使用 SQL Server 成员提供者和 SQL Server 角色提供者。在下一章中，将使用 SQL Server 配置文件提供者。这三个提供者都可配置为使用一个 SQL Server 数据库，从

而可以简单地集中所有用户数据。

理想情况下，不需要直接处理这些提供者。一般情况下，在一个集中的位置为 Web 站点配置不同的提供者。可以通过与应用程序服务通信来使用这些提供者。尽管可以直接从代码访问这些服务，但通常是使用 ASP.NET 内置的登录控件来完成这些困难的工作。下面将介绍这些控件。

16.2 登录控件

随 ASP.NET 4 提供的登录控件去除了大量通常与编写 Web 站点中的安全层相关的复杂性。可用的登录控件有效地封装了验证和管理用户所需的所有代码和逻辑。这些控件通过应用程序服务与已配置的提供者通信来进行工作，而不是直接与数据库通信。要了解其工作原理，可查看下面这个“试一试”练习，它展示了如何创建一个简单的 Login 和 Sign Up 页面，允许新用户创建账户，然后登录。后面一节将介绍 ASP.NET 4 提供的 7 种登录控件。

试一试

创建 Login 和 Sign Up 页面

在这个“试一试”练习中，将扩展前面创建的 Login 页面。另外还将创建一个新页面，允许用户在 Planet Wrox Web 站点上注册账户。

(1) 在 Markup 视图下，从站点根目录下打开第 6 章中创建的 Login.aspx 页面。(如果没有该页面，现在就基于自定义模板创建它，并将其 Title 设置为 Log in to Planet Wrox。使用相同的文本在 cpMainContent 占位符中添加一个<h1>元素。)

(2) 从 Toolbox 的 Login 类别中拖放一个 LoginStatus 控件到该页面上的 h1 元素之后。

(3) 切换到 Design 视图。然后，从 Toolbox 中拖放一个 Login 控件到 LoginStatus 控件上，并位于其之上，如图 16-2 所示(LoginStatus 显示为 Login 控件下的 Login 小链接)。

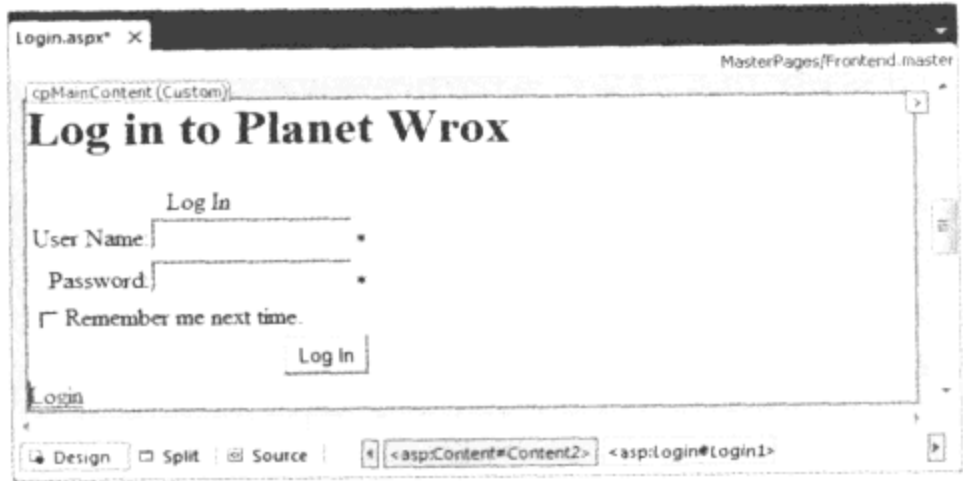


图 16-2

(4) 打开 Login 控件的 Properties 面板，设置两个如表 16-1 所示的属性。

表 16-1

属 性	值
CreateUserText	Sign Up for a New Account at Planet Wrox
CreateUserUrl	SignUp.aspx

- (5) 在该 Web 站点的根目录下创建一个名为 `SignUp.aspx` 的新 Web 窗体，使其基于您的自定义模板，设置其 Title 为 `Sign Up for a New Account at Planet Wrox`。
- (6) 切换页面至 Design 视图并从 Toolbox 中拖出一个 `CreateUserWizard` 控件到页面的主内容区。保存并关闭该页面。
- (7) 从站点根目录中打开 `web.config` 文件，并添加一个 `<authentication>` 元素作为 `<system.web>` 元素的子元素，并将其 `mode` 特性设置为 `Forms`。

```
<system.web>
  <authentication mode="Forms" />
  ...
</system.web>
```

- (8) 返回到 `Login.aspx`，保存所有修改，然后按 `Ctrl+F5` 组合键在浏览器中打开页面。将出现如图 16-3 所示的登录框。
- 注意，`Login` 控件下的登录状态当前设置为 `Login`，表明还没有登录。如果文本为 `Logout`，说明您在 `web.config` 文件中将身份验证模式设置为 `Forms`。
- (9) 输入任意用户名和密码尝试登录。显然，这会失败，因为您还没有创建该账户。可能需要一些时间才能看到结果，因为 ASP.NET 忙于建立成员数据库。
- (10) 单击 `Login` 控件下的 `Sign Up` 链接到达 `SignUp.aspx` 页面，然后通过输入您的详细个人信息创建一个账户(如图 16-4 所示)。默认情况下，密码至少需要 7 个字符，且必须包含一个非字母数字字符。要注意，数字并不是非字母数字字符，因此需要确保密码至少包含一个像 `#`、`$`、`*` 这样的字符。例如，`Pa55word` 并不是有效的密码，而 `Pass##Word` 则会被接受。另外注意，密码是区分大小写的。记下刚刚输入的用户名和密码，因为后面会用到该账户信息。

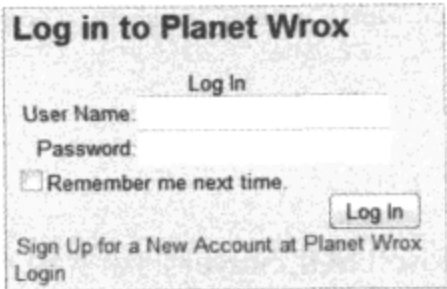


图 16-3



图 16-4

- 对于安全问题和答案，您可以自己设一个带答案的问题。如果忘记了密码，您需要提供这一安全问题的答案来找回它。
- (11) 单击 `Create User` 按钮创建账户。当页面重新加载时，您就会得到一个确认，表明已成功创建了该账户。先忽略 `Continue` 按钮(您还未给它编写任何行为)，而从 Web 站点的主 Menu 或 TreeView(取决于当前的主题)中选择 `Login` 项。再次来到 `Login.aspx` 页面，其中 `Login` 控件下面的 `LoginStatus` 控件表示您已登录(如图 16-5 所示)。在使用 `CreateUserWizard` 创建新账户时会自动登录，不过可以通过设置该控件的 `LoginCreatedUser` 属性为 `False` 来修改这一行为。

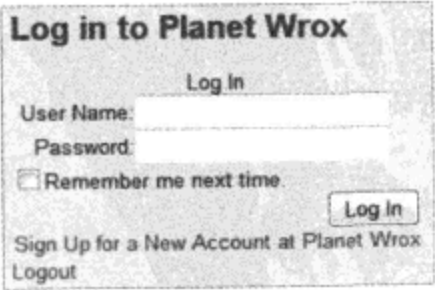


图 16-5

(12) 单击 Logout 链接将会注销登录，导致 LoginStatus 再次显示文本 Login。在 Login 控件中，输入第(10)步输入的用户名和密码，然后单击 Log In 按钮。这样您将会登录并被重定向到主页。在主 Menu 或 TreeView 上单击 Login 再次返回到 Login 页面。注意，LoginStatus 已经修改，再次显示 Logout，表示您已成功登录。

这时，登录并没有意义：您所看到的只是 LoginStatus 从 Login 修改为 Logout。不过，本章后面将介绍如何为登录用户提供不同的内容。

工作原理

在这个“试一试”练习中，除了添加和配置一些 ASP.NET 服务器控件外，您并未编写任何代码。不过，仍能实现一个功能完全的登录过程，允许用户注册一个账户，然后登录到该站点。在注册时，您提供一组后面将用于登录 Planet Wrox 站点的用户名和密码。那么这如何工作呢？正如前面所述，ASP.NET 控件与已配置的应用程序服务提供者通信；后者是位于登录控件和跟踪用户的 SQL Server 数据库之间的软件层。

在您首次尝试登录(或使用需要数据库访问的其他登录控件)时，提供者检查应用程序是否在使用带有必要数据库对象(如表)的数据库。默认情况下，它通过查找名为 LocalSqlServer 的连接字符串检查数据库。在您的 web.config 文件中无法找到这一连接字符串，因为它定义在 .NET Framework 文件夹中名为 machine.config 的文件中。后面将学习该文件的更多内容。该文件中的连接字符串类似于如下所示：

```
<connectionStrings>
  <add name="LocalSqlServer" connectionString="data source=.\SQLEXPRESS;
    Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;
    User Instance=true"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

这是一个针对 SQL Server 2008 Express Edition(Microsoft SQL Server 的免费版本，在第 1 章安装)的连接字符串。在第 19 章及附录 B 中，将介绍更多针对 SQL Server 完全版本的连接字符串示例。

第 13 章中学习了使用|DataDirectory|的连接字符串指向 Web 站点的 App_Data 文件夹中的数据库。如果首次使用某个登录控件或其他应用程序服务时，ASPNETDB.MDF 数据库并没有在特定位置出现，那么应用程序服务就会自动创建一个 ASP.NETDB.MDF 数据库(所以在第(9)步中第一次输入用户名和密码时会有一个延迟)。为了了解数据库的样子，可返回到 Visual Web Developer 中单击 App_Data 文件夹，然后单击 Solution Explorer 中工具栏上的 Refresh 按钮。将看到如图 16-6 所示的新数据库 ASPNETDB.MDF。

在 Solution Explorer 中双击该新数据库以在 Database Explorer 窗口中打开它。展开 Tables 和 Views 等项，获取已添加的数据库对象的列表，如图 16-7 所示，该图中显示了一些用于应用程序服务的表。

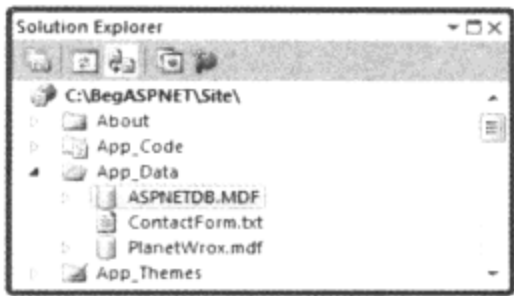


图 16-6

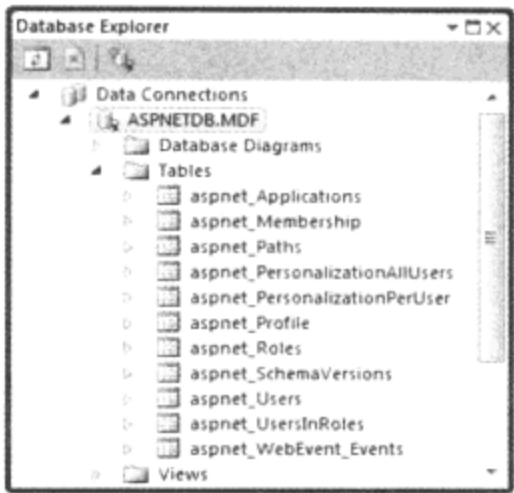


图 16-7

在成功创建数据库后，登录控件就可以使用它。例如，在使用 CreateUserWizard 控件创建一个新账户时，记录就插入到 aspnet_Membership 和 aspnet_Users 表中。同样，在您尝试登录时，您的用户名和密码会针对这些表作核对。

为了强制 ASP.NET 运行时使用基于表单的身份验证(SQL Server 成员提供者在后台使用)，需要在 web.config 文件中设置 authentication 元素的 mode 特性为 Forms：

```
<authentication mode="Forms" />
```

mode 特性的其他选项包括 Windows(其安全性由 Windows 本身来处理)、Passport(使用 Microsoft 的 Passport 服务)和 None(禁用安全性)。在本书剩余部分，将只使用 Forms 选项，因为它是面向 Internet 的 Web 应用程序的常用解决方案。

Login 控件的 Remember Me Next Time 选项的记忆时间比想象地要短。如果选择了该项，下次访问站点时，在 cookie 没有过期的情况下，您会自动登录。首次登录时，服务器会为将来的会话设置一个 cookie。不过，这一 cookie 会在 30 分钟后过期，这意味着在那个时间段后返回站点的用户需重新进行身份验证。为了延长用户持续登录的时间，需要设置<forms>元素的 timeout 特性，前者是 web.config 中的<authentication>元素的直接子元素。timeout 采用一个整型值，用分钟数表示超时时间。下列代码设置超时为 24 小时(1440 分钟)：

```
<authentication mode="Forms">
  <forms timeout="1440" />
</authentication>
```

一般较低的 timeout 值更安全，因为它们不提供无限制或持久的访问。但是较高的 timeout 值对用户更适合，因为用户不需要在每次访问站点时都重新进行身份验证。

ASP.NET 在 App_Data 文件夹中创建的数据库使得可以很容易地在开发场景中使用不同的应用程序服务。不过，如果您需要使应用程序进入您自己的网络中的产品服务器或是提供给 Internet Service Provider(ISP)，则需要对数据库和该数据库被访问的方式有更多的控制。第 19 章(讨论应用程序的部署)和附录 B(讨论 SQL Server 的配置)将对此作更详细的介绍。

现在，您已经了解登录控件如何与自动生成的 SQL Server Express 数据库一起使用，接下来将更详细地介绍 Toolbox 的 Login 类别中的控件。

16.2.1 登录控件

ASP.NET 4 中包含 7 个登录控件，每个都有不同的用途。图 16-8 显示了 Toolbox 中的 7 个登录控件。

在接下来的部分中，将会详细介绍这些控件。

1. Login

正如在前一个“试一试”练习中所看到的，Login 控件允许用户登录到站点。而在后台，它通过应用程序服务与配置好的成员提供者进行通信，查看用户名和密码是否代表系统中的有效用户。如果用户通过验证，就生成发送到用户浏览器的 cookie。对于后续的请求，浏览器重新提交该 cookie 给服务器，这样系统就知道它仍在处理有效用户。成员提供者的不同设置都在 web.config 文件的<membership />元素中进行配置。本章后面将做详细介绍。

要创建功能完全的登录页面，只需下列控件声明：

```
<asp:Login ID="Login1" runat="server" />
```

不过，在大部分情况下，可以通过设置如表 16-2 所示的一个或多个属性来增强控件的外观和行为。

表 16-2

属 性	说 明
DestinationPageUrl	该属性定义了登录请求成功后将用户发往哪个 URL
CreateUserText	该属性控制用于邀请用户注册新账户的文本
CreateUserUrl	该属性控制用户注册新账户的页面的 URL
DisplayRememberMe	该属性指定控件是否显示 Remember Me 选项。如果设置为 False 或在登录时未选择该复选框，那么每次关闭和重新打开浏览器时，用户需要重新进行身份验证
RememberMeSet	该属性指定最初是否选择 Remember Me 选项
PasswordRecoveryText	该属性控制用于告诉用户可重置或恢复他们密码的文本
PasswordRecoveryUrl	该属性指定用户可获取他们(新)密码的页面的 URL
VisibleWhenLoggedIn	该属性控制当前用户登录时控件是否可见。默认值为 True

默认情况下，ASP.NET 的身份验证机制假定在站点的根目录下有一个用于用户登录的页面 Login.aspx。这个页面要想生效，最少需要有一个 Login 控件。如果想要使用一个不同的页面，可以在<authentication />下面的<forms />元素中指定其路径，如下所示：

```
<authentication mode="Forms">
  <forms loginUrl="MyLoginPage.aspx" />
</authentication>
```

每当需要用户提供凭据的时候，这种配置就通知 ASP.NET 运行库加载页面 MyLoginPage.aspx。



图 16-8

注意，在 Login 页面(配置在 loginUrl 中)上，Login 控件的 VisibleWhenLoggedIn 属性没有作用。在配置过的 Login 页面上，Login 控件总是可见。如果想要隐藏它，可以使用 LoginView 控件，后面的“试一试”练习将展示这一点。

除了这些属性，Login 控件还有一些 Text 属性，如 LoginButtonText、RememberMeText、TitleText 和 UserNameLabelText，这些属性用于设置控件中和其各种子控件(如组成用户界面的 Button 和 Label 控件)上出现的文本。

和数据绑定控件一样，登录控件有大量可调整其外观的样式属性。可以查看控件的 Properties 面板的 Styles 类别，了解如何设置不同的样式选项。记住，和数据绑定控件一样，可以将大量样式信息移至外观和 CSS 文件。

Login 控件也提供了一些事件，这些事件通常不需要您进行处理，但时常都会派上用场。例如，LoggedIn 事件在用户刚登录后触发，如果 DestinationPageUrl 不是很灵活，这里是将用户动态发送到另一页面的理想场所。

2. LoginView

LoginView 是一个方便的控件，可用于向不同的用户显示不同的数据。它使您可以区分匿名用户和登录用户，甚至区分不同角色中的用户。LoginView 是模板驱动的，因此可允许我们定义显示给不同用户的不同模板。表 16-3 列出了两个主要的模板和特殊的 RoleGroups 元素。

表 16-3

模 板	说 明
AnonymousTemplate	该模板中的内容只显示给未进行身份验证的用户
LoggedInTemplate	该模板中的内容只显示给登录用户。该模板与 AnonymousTemplate 互斥。任何时候只有其中一个模板可见
RoleGroups	该模板可以包含一个或多个 RoleGroup 元素，这些元素包含一个定义特定角色的内容的 ContentTemplate 元素。允许查看内容的角色定义在 Roles 特性中，该特性采用一个逗号分隔的角色列表。RoleGroups 元素与 LoggedInTemplate 互斥。这意味着如果用户是 RoleGroup 的其中一个角色中的成员，那么 LoggedInTemplate 中的内容就不可见。另外，只有匹配用户角色的第一个 RoleGroup 的内容可见

除了定义在控件各种子元素中的内容，LoginView 控件本身并不输出任何标记，这意味着您可以很容易地将它嵌入一对 HTML 标记之间，如<h1>和，从而创建自定义标题或者列表项。

下列代码段显示了一个 LoginView 控件，它为 3 个不同用户定义了内容：访问站点的匿名用户、登录用户，以及已经登录并且是 Managers 角色的成员的用户：

```
<asp:LoginView ID="LoginView1" runat="server">
  <AnonymousTemplate>
    Hi there visitor. Would you be interested in signing up for an account?
  </AnonymousTemplate>
  <LoggedInTemplate>
    Hi there visitor and welcome back to PlanetWrox.com.
  </LoggedInTemplate>
```



```
<RoleGroups>
  <asp:RoleGroup Roles="Managers">
    <ContentTemplate>
      Hi there manager. You can proceed to the Management section.
    </ContentTemplate>
  </asp:RoleGroup>
</RoleGroups>
</asp:LoginView>
```

本章后面将介绍更多有关如何创建和配置角色的内容。

3. LoginStatus

正如前一个“试一试”练习所示，LoginStatus 控件提供了有关用户当前状态的信息。当用户未进行身份验证时，它提供 Login 链接，而当用户登录后，它提供 Logout 链接。通过设置 LoginText 和 LogoutText 属性，可以控制实际显示的文本。或者，可设置 LoginImageUrl 和 LogoutImageUrl 属性显示图像而非文本。最后，可设置 LogoutAction 属性来决定在用户注销时是否刷新当前页面，或是否是在用户注销后将用户带至另一页面。通过设置 LogoutPageUrl 可以确定这一目标页面。

除了这些属性，控件可以引发两个事件：LoggingOut 和 LoggedOut，它们在用户刚注销前后触发。

4. LoginName

LoginName 是一个极为简单的控件。它所做的就是显示登录用户的名称。为了将用户名嵌入到一些文本中，如 You are logged in as Imar，可以使用 FormatString 属性。如果在这个格式字符串中包含 {0}，它将被用户名所取代。

在下一个“试一试”练习中将介绍其工作原理。该练习修改站点的登录页面和母版页，使它们显示用户的相关信息。

试一试

运用登录控件

在本练习中，当用户登录后在 Login.aspx 页面上隐藏 Login 控件，然后显示一条消息。另外，添加文本到页面的页脚，以显示用户名以及注销的选项。

(1) 打开 Login.aspx 并切换至 Design 视图。从 Toolbox 的 Login 类别中拖放一个新的 LoginView 控件到 Login 控件的顶部，这样在页面中它位于该控件之上。

(2) 打开 LoginView 控件的 Smart Tasks 面板，确保在 Views 下拉列表中选择 AnonymousTemplate 选项，如图 16-9 所示。

控件中的任何内容都将放置到 AnonymousTemplate 区域，因为现在在 Design 视图中这是控件的活动模板。

(3) 单击选择 Login 控件，然后按 Ctrl+X 组合键将它剪切至剪贴板。在表示 LoginView 的白色小矩形的内部单击，将光标放到该控件中，并按 Ctrl+V 组合键将 Login 控件粘贴到 LoginView 中。

(4) 再次打开 LoginView 控件的 Smart Tasks 面板，使用 Views 下拉列表切换至 LoggedInTemplate。



图 16-9

再次在控件的白色小矩形的内部单击，然后输入文本 You are already logged in。

(5) 切换至 Mark 视图并查看代码。Login 控件将放置在 AnonymousTemplate 内部，而输入的文本将显示在 LoggedInTemplate 标记中：

```
<asp:LoginView ID="LoginView1" runat="server">
  <AnonymousTemplate>
    <asp:Login ID="Login1" runat="server" CreateUserUrl="SignUp.aspx"
      CreateUserText="Sign Up for a New Account at Planet Wrox">
    </asp:Login>
  </AnonymousTemplate>
  <LoggedInTemplate>
    You are already logged in.
  </LoggedInTemplate>
</asp:LoginView>
```

(6) 保存并关闭页面，因为现在不再需要对它执行其他操作。。

(7) 在 Markup 视图中打开母版页 Fronted.master，并把 id 为 Footer 的<div>元素定位到页面底部。删除文本 Footer Goes Here，通过从 Toolbox 中拖放一个新的 LoginName 控件到<div>元素来取代它。直接输入代码来设置其 FormatString 属性为 Logged in as {0}。

```
<asp:LoginName ID="LoginName1" runat="server" FormatString="Logged in as {0}" />
```

(8) 从 Toolbox 中拖动一个新的 LoginView 控件，将它放置在 LoginName 控件之下，但仍在 Footer <div>中。切换至 Design 视图，在 LoginView 的 Smart Tasks 面板上从 Views 下拉列表中选择 LoggedInTemplate，然后在活动 LoggedInTemplate 的白色矩形中拖放一个新的 LoginStatus 控件。

(9) 再次切换至 Markup 视图，将 LoginStatus 的代码包括到一对圆括号中。最终代码如下所示：

```
<div id="Footer">
  <asp:LoginName ID="LoginName1" runat="server"
    FormatString="Logged in as {0}" />
  <asp:LoginView ID="LoginView1" runat="server">
    <LoggedInTemplate>
      (<asp:LoginStatus ID="LoginStatus1" runat="server" />)
    </LoggedInTemplate>
  </asp:LoginView>
</div>
```

(10) 保存所有修改并在浏览器中请求 Login.aspx。使用在前一“试一试”练习中创建的账户和密码登录(您可能需要单击 Logout 链接先注销)。如果忘记了用户名和密码，可单击 Sign Up 链接创建一个新账户。注意，一旦登录，页脚会显示如图 16-10 所示的文本。



图 16-10

(11) 单击 Menu 或 TreeView 中的 Login 项转到 Login 页面。您将看到一条表明已成功登录的消息，而不是 Login 控件。

(12) 单击页面底部的页脚中的 Logout 链接。页面刷新并再次显示 Login 控件。另外，页脚的文本消失了。

工作原理

首先添加一个 LoginView 控件到 Login 页面来包装 Login 控件和一条文本消息。如果用户还没有登录，则显示 Login 控件，而文本则只为登录用户显示。

母版页页脚中的代码包含一个 LoginName 控件，它显示登录用户的名称。对于匿名用户，它不显示任何内容。为了控制显示的文本，可以使用 FormatString 属性：

```
<asp:LoginName ID="LoginName1" runat="server" FormatString="Logged in as {0}" />
```

在运行时，{0}被用户名取代。

默认情况下，添加的 LoginStatus 显示一个允许用户登录和注销的链接。由于 Menu 或 TreeView 已经包含了 Login 页面的链接，当用户已登录时，页脚又使用一个 LoginView 只显示 Logout 文本。如果也想添加一个 Login 链接，可用一个匿名模板和一个额外的 LoginStatus 扩展 LoginView 或是删除整个 LoginView，这样 LoginStatus 对所有用户都可见。

除了允许用户登录和使用当前用户的登录状态显示或隐藏相关内容的控件外，Toolbox 的 Login 类别中还包含了 3 个控件，它们允许用户在站点上注册新账户、修改现有密码或是恢复丢失密码。下面将介绍这些控件。

5. CreateUserWizard

在前面的“试一试”练习中，您已经简单了解了 CreateUserWizard 的运用。不过，该控件提供了更多的功能，而不只是前面“试一试”练习中所看到的标准行为。

首先，该控件有一个较长的 Text 属性列表，如 CancelButtonText、CompleteSuccessText、UserNameLabelText 和 CreateUserButtonText，它们会影响控件中使用的文本。所有属性都有好的默认设置(英文的)，不过您可修改它们来满足自己的需求。

该控件有许多以 ImageUrl 结尾的属性，如 CreateUserButtonImageUrl。这些属性允许定义各种用户动作的图像而非控件生成的默认按钮。如果设置任一属性为有效的 ImageUrl，则还需要设置相应的 ButtonType。例如，要将 Create User 按钮修改为图像，则需要设置 CreateUserButtonImageUrl 为有效图像并设置 CreateUserButtonType 为 Image。

ButtonType 的默认值为 Button，默认情况下它呈现为标准的按钮。也可以设置这些属性为 Link，这样它们将呈现为标准的 LinkButton 控件。

另外，该控件提供了大量可设置用于修改其行为和外观的有用属性，如表 16-4 所示。

表 16-4

属 性	说 明
ContinueDestinationPageUrl	该属性定义用户在注册后单击 Continue 按钮时被带往的页面
DisableCreatedUser	当账户创建时是否将用户标记为禁用的。如果设置为 True，那么在账户被启用以前，用户不能登录到该站点。后面将介绍如何手动激活或不激活用户账户。默认为 False
LoginCreatedUser	在账户创建后是否让用户自动登录。默认为 True
RequireEmail	决定控件是否要求用户提供电子邮件地址。默认为 True
MailDefinition	该属性包含大量子属性，允许定义在用户注册后发送给他们的电子邮件(可选)

您可能注意到，控件并没有属性来隐藏安全问题和答案，或是修改强密码策略，即要求用户输入一个至少 7 个字符的密码的策略。因为多个控件都需要访问这些设置，所以需要在底层提供者上进行配置。在本章后面 16.2.2 小节中将介绍如何配置。

CreateUserWizard 控件能够向用户发送一封确认电子邮件，通知它们新账户已经成功建立。这封电子邮件可以提醒它们的用户名和密码。在接下来的“试一试”练习中，将介绍如何配置 MailDefinition 元素，这样 CreateUserWizard 将向新用户发送电子邮件消息确认其账户并且发送用户名和密码以备后查。

试一试

用 CreateUserWizard 发送确认电子邮件

要尝试这个练习，需要使用一个有效的邮件服务器名或者本地接收文件夹配置 web.config 文件的 <system.net> 元素。如果没有这些设置，或者不知道如何配置它们，可以参考第 9 章。

- (1) 添加一个新的 Text File 到 App_Data 文件夹中，并命名为 SignUpConfirmation.txt。
- (2) 添加下列文本到该文件中，保存并关闭它。

```
Hi <% UserName %>,

Thank you for signing up for a new account at www.PlanetWrox.com.

To log in to the site, use the following details:

User name:      <% UserName %>
Your password:  <% Password %>

We look forward to your contributions.

The Planet Wrox Team
```

在输入 UserName 和 Password 占位符时要注意。它们包括在一对服务器端标记(<%和%>)中，从而被赋予特殊的意思。

(3) 打开 SignUp.aspx 并确保其位于 Design 视图中。然后在 CreateUserWizard 控件的 Properties 面板上，定位到 MailDefinition 属性并展开它。单击 BodyFileName 属性，然后单击省略号按钮来浏览文件，再选择在 App_Data 文件夹中创建的 SignUpConfirmation.txt。

(4) 设置 Subject 属性为 Your New Account at PlanetWrox.com。完成之后的 Properties 面板如图 16-11 所示。

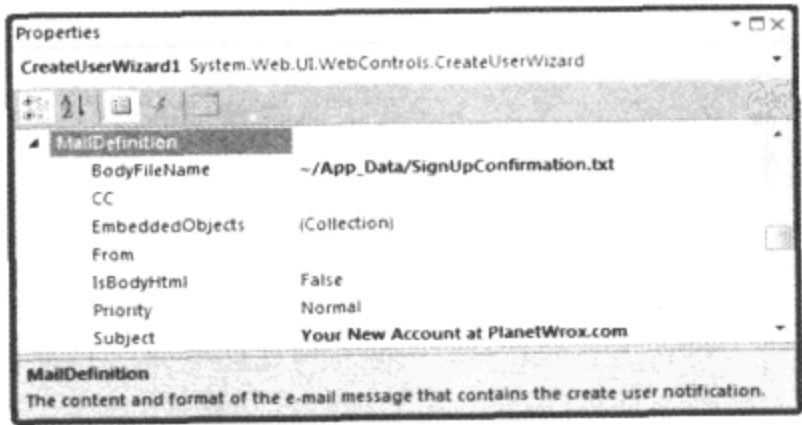


图 16-11

(5) 保存所有更改并在浏览器中请求 `SignUp.aspx`。为新账户输入必需的详细信息并单击 `Create User` 以注册新用户。如果您得到一个有关 `From` 地址错误的消息，则确保向 `web.config` 文件中的 `from` 特性赋予了一个有效的电子邮件地址：

```
<smtp deliveryMethod="SpecifiedPickupDirectory" from="info@PlanetWrox.com">
```

第 9 章介绍了如何添加这个特性。确保输入有效的电子邮件地址，否则邮件服务器仍然可能拒绝接受该地址。

如果邮件服务器要求您使用 SSL(例如，当您在使用 Gmail 的 SMTP 服务器时)，不要忘了在 `web.config` 文件中将 `<network />` 元素的 `enableSsl` 特性设置为 `true`。或者，可以在控件的 `SendingMail` 事件中编写自己的代码来发送消息，并重写 `MailMessage` 或 `SmtpClient` 的一些设置。可以访问 `SendingMail` 事件的 `e.Message` 获得对需发送的消息的访问权。为此，将 `SignUp.aspx` 切换到 `Design View`，打开 `CreateUserWizard` 的 `Properties` 面板，切换到 `Events` 选项卡，并为 `SendingMail` 事件建立处理程序。在该处理程序中，编写下面的代码(需要为 `System.Net.Mail` 名称空间添加一个 `Imports/using` 语句，以便能够使用 `SmtpClient` 类)。下面的代码假定您将 `web.config` 文件配置为使用 Gmail 的邮件服务器，并在 `<network />` 元素的 `port` 特性中配置了一个端口号，例如 587 或者 465。如果无法使代码工作，可以参考我的网站上的一篇详细讨论这些问题的文章，网址为 <http://tinyurl.com/ybx7qkh>。

VB.NET

```
Protected Sub CreateUserWizard1_SendingMail(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.MailMessageEventArgs) _
    Handles CreateUserWizard1.SendingMail
    Dim myClient As New SmtpClient()
    myClient.EnableSsl = True
    myClient.Send(e.Message)
    e.Cancel = True
End Sub
```

C#

```
protected void CreateUserWizard1_SendingMail(object sender,
    MailMessageEventArgs e)
{
    SmtpClient myClient = new SmtpClient();
    myClient.EnableSsl = true;
    myClient.Send(e.Message);
    e.Cancel = true;
}
```

这个例子显示了如何修改发送邮件的方式——例如，通过编程方式在 `SmtpClient` 上启用 SSL。如果所要做的只是使用 Gmail 的安全邮件服务器发送电子邮件，那么只使用 `web.config` 中的设置更好，如第 9 章所述。

(6) 稍后，您将接收到一封包含在第(2)步中输入的欢迎文本的电子邮件。图 16-12 显示了一条消息，其用户名和密码占位符已被第(5)步中输入的信息所替换。

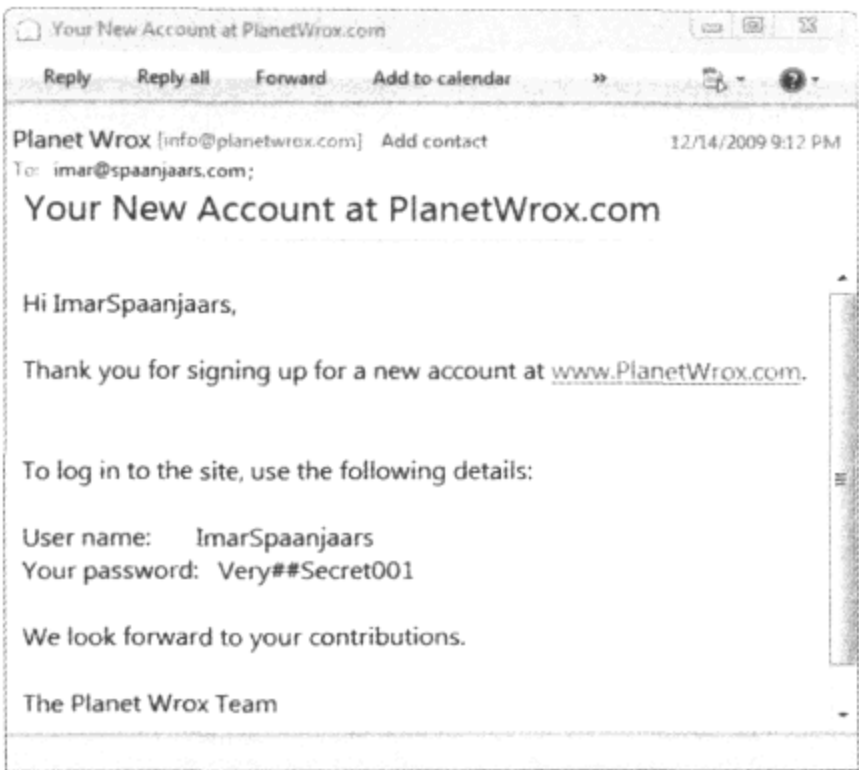



图 16-12

工作原理

CreateUserWizard 提供了将确认消息发送给用户的内置功能。在您指定了<MailDefinition>元素后，它才会发送消息。同时，使用 BodyFileName 属性指向用作电子邮件正文的文本文件或 HTML 文件。在这个正文中，可以使用特殊的占位符<% UserName %>和<% Password %>，它们将被用户在注册表单中输入的实际用户名和密码自动取代。

虽然邮件的发送通常会被自动处理，但是知道可以挂钩到这个过程并对发送的消息或者用来发送消息的 SmtpClient 进行修改仍然有帮助。在这个“试一试”练习中，看到了如何挂钩到 SendingMail 事件并修改 SmtpClient，但是也可以进行其他修改(例如，修改 e.Message 参数的属性)。



注意：如果在用户尝试提交表单以前，可以检查指定的用户名是否已经存在，那么这会是一个很方便的功能。可以使用前面章节中学到的技术轻松地实现这个目的。例如，可以在服务器上使用页面方法来检查给定的名称是否已经使用。当用户离开 User Name 字段时，可以立即使用一些 JavaScript 代码调用这个页面方法。从我的网站 <http://imar.spaanjaars.com/QuickDocId.aspx?quickdoc=494> 上可以找到这种解决方案的一个逐步指导。

下面将讨论的 PasswordRecovery 控件也支持自定义邮件正文，并允许在 SendingMail 事件的处理程序中手动发送消息。

6. PasswordRecovery

PasswordRecovery 控件允许用户获得他们已有的密码(如果系统支持)或是获得一个新的自动生成的密码。在这两种情况下，密码将发送到用户注册账户时输入的电子邮件地址中。

PasswordRecovery 控件的大部分属性都是我们所熟悉的。它有大量 Text 属性，如 GeneralFailureText

(当密码不可恢复时显示)和 `SuccessText`，该属性允许设置控件显示的文本。它还有一些以 `ButtonType`、`ButtonText` 和 `ButtonImageUrl` 结尾的属性，该属性允许修改控件的不同动作按钮的外观和行为。如果密码成功恢复且您想将用户发送到另一页面，则可以设置 `SuccessPageUrl` 为您站点中的一个页面。

和 `CreateUserWizard` 一样，`PasswordRecovery` 也有一个 `MailDefinition` 元素，该元素允许指向您想作为邮件正文发送的文件。可以对用户名和密码使用同样的占位符来自定义消息。如果不对 `MailDefinition` 进行配置，则控件会使用一个默认的邮件正文(这在下一个“试一试”练习中将看到)。

7. ChangePassword

`ChangePassword` 控件允许已有用户和登录用户更改他们的密码。类似于 `CreateUserWizard` 和 `PasswordRecovery` 控件，它有许多属性，可用于修改文本、错误消息和按钮。它还有一个 `MailDefinition` 元素，允许发送新密码的确认消息给用户的电子邮件地址。

试一试

实现密码功能

在这个“试一试”练习中，向 Web 站点添加 `PasswordRecovery` 和 `ChangePassword` 控件，允许用户修改和恢复他们的密码。由于修改密码只对登录用户有意义，所以将 `ChangePassword` 控件添加到它自己的页面。在本章后一个“试一试”练习中，您要保护这一页面，使得只有经过身份验证的用户才能访问它。

(1) 在 Markup 视图中打开 `Login.aspx`，然后定位到 `<AnonymousTemplate>` 中的 `</asp:Login>` 结束标记。在其之后，输入两个 `
` 元素(使用 `br` 代码段，然后按下 `Tab` 键来完成该元素)在 `Login` 控件下方创建一些空间。

(2) 从 `Toolbox` 中拖放一个 `PasswordRecovery` 控件到代码编辑器中，放在第(1)步添加的两个 `
` 元素之下。

(3) 在 `PasswordRecovery` 控件的起始和结束标记之间添加一个 `<MailDefinition>` 元素，然后设置电子邮件的 `Subject` 为 `Your New Password for PlanetWrox.Com`。这时代码如下所示：

```
</asp:Login>
<br />
<br />
<asp:PasswordRecovery ID="PasswordRecovery1" runat="server">
  <MailDefinition subject="Your New Password for PlanetWrox.Com"></MailDefinition>
</asp:PasswordRecovery>
```

(4) 保存更改并关闭文件。

(5) 在站点的根目录下创建一个基于自定义模板的新的 Web 窗体，并命名为 `MyProfile.aspx`。设置该页面的 `Title` 为 `My Profile`。

(6) 切换至 Markup 视图，在 `cpMainContent` 内容占位符中创建一个 `<h1>` 元素(输入 `h1`，然后按下 `Tab` 键)，设置其内容文本为 `My Profile`。在标题下输入一些文本，说明 `My Profile` 页面是用于更改密码之类的内容。将该文本包装在一对 `<p>` 标记中表示一个段落。

(7) 从 `Toolbox` 中拖动一个 `ChangePassword` 控件，将它放到 `</p>` 结束标记之后。所得代码如下所示：

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <h1>My Profile</h1>
  <p>The My Profile page allows you to make changes to your personal profile.
    For now, all you can do is change your password below.</p>
  <asp:ChangePassword ID="ChangePassword1" runat="server"></asp:ChangePassword>
```

(8) 在 Solution Explorer 中打开 Web.sitemap 并在 About 部分下添加一个新元素。使 url 指向 ~/MyProfile.aspx，然后设置 title 和 description 为 My Profile。所得代码如下所示：

```
<siteMapNode url="~/About/Default.aspx" title="About"
  description="About this site">
  <siteMapNode url="~/About/Contact.aspx" title="Contact Us"
    description="Contact Us" />
  <siteMapNode url="~/About/AboutUs.aspx" title="About Us"
    description="About Us" />
  <siteMapNode url="~/MyProfile.aspx" title="My Profile"
    description="My Profile" />
</siteMapNode>
```

(9) 保存所有更改并关闭所有打开的文件。右击 Solution Explorer 中的 Login.aspx 并选择 View in Browser 命令。在 Login 控件下，您将看到 PasswordRecovery 控件，如图 16-13 所示。



图 16-13

注意如果您已经登录，则需要先单击 Logout 链接。

(10) 在 PasswordRecovery 控件中输入您的用户名并单击 Submit 按钮。您会被要求输入安全问题的答案，这是您在注册该账户时提供的。输入答案并再次单击 Submit 按钮。稍候，您会接收到一封带有新的自动生成的密码的电子邮件消息。如果邮件服务器要求进行特殊处理，则采取与 CreateUserWizard 控件相同的步骤，在 PasswordRecovery 控件的 SendingMail 事件中手动构建 SmtpClient 实例。

(11) 使用这一新密码登录站点。在登录后，从 Menu 或 TreeView 中选择 My Profile 项。会出现如图 16-14 所示的 ChangePassword 控件。

(12) 输入通过邮件发送给您的自动生成的密码，然后输入一个易于记住的新密码，然后再次输入与之相同的密码。最后，单击 Change Password 按钮。这样，您就可以使用新密码登录到该站点了。

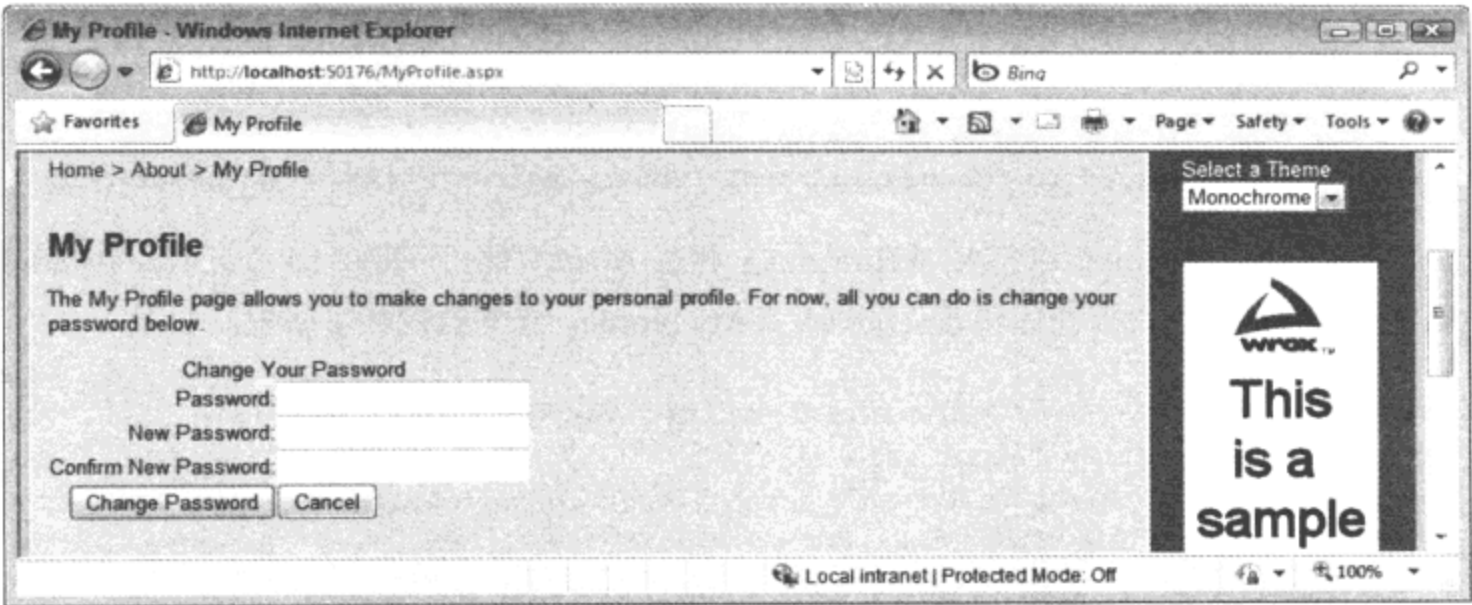


图 16-14

工作原理

默认情况下，密码在数据库中以散列形式存储，这意味着它们不可被检索。散列是一个不可逆的进程，它为数据创建唯一指纹。因为它是不可逆的，所以没有办法通过散列重新创建密码，这使得密码可以更安全地存储在数据库中。当您登录时，所输入的密码也将被散列化，然后两个散列值进行比较来判断是否允许您进入。因为原密码不可检索，所以 PasswordRecovery 控件将生成新密码。然后，它将这个密码发送到与您输入的用户名相关的电子邮件地址。作为邮件正文，它使用一个包含用户名和新密码的标准模板。而要自定义邮件正文，可以像对 CreateUserWizard 操作的那样，将 MailDefiniton 的 BodyFileName 指向一个包含用户名和密码的占位符的文本文件。

您可能也注意到了，登录控件使用了一些到目前为止都无法更改的默认属性。例如，CreateUserWizard 和 PasswordRecovery 控件总是要求用户用安全问题和答案来实现另外一层安全防护。您还需要输入带有至少 7 个字符的强密码。通过 web.config 文件可以为整个应用程序修改这些设置。

16.2.2 配置 Web 应用程序

本章前面讨论了如何在 machine.config 文件中定义名为 LocalSqlServer 的默认连接字符串，这样它就默认在所有构建的 Web 应用程序中有效。与 ASP.NET 应用程序服务(如成员、角色和配置文件)相关的其他设置与此相同。machine.config 包含了成员服务的下列设置：

```
<membership>
  <providers>
    <add name="AspNetSqlMembershipProvider"
      type="System.Web.Security.SqlMembershipProvider, System.Web,
        Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="LocalSqlServer"
      enablePasswordRetrieval="false"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="true"
      applicationName="/"
      requiresUniqueEmail="false"
      passwordFormat="Hashed"/>
```



```
maxInvalidPasswordAttempts="5"
minRequiredPasswordLength="7"
minRequiredNonalphanumericCharacters="1"
passwordAttemptWindow="10"
passwordStrengthRegularExpression=""
/>
</providers>
</membership>
```

此提供者配置具有如表 16-5 所示的大量有意思的特性。

表 16-5

特 性	说 明
connectionStringName	该特性指向应用程序的连接字符串的名称。默认为前面看到的 LocalSqlServer 连接字符串
enablePasswordRetrieval	该特性决定用户是否能检索他们的当前密码。如果 passwordFormat 为 Hashed (见 passwordFormat)，则不能设置该选项
enablePasswordReset	该特性决定用户是否能请求新密码
requiresQuestionAndAnswer	该特性决定像 CreateUserWizard 和 PasswordRecovery 这类控件是否要用户输入安全问题和答案
applicationName	该特性提供应用程序的唯一名称
requiresUniqueEmail	该特性决定系统是否允许用户账户使用重复的电子邮件地址。如果设置为 True，每个用户必须提供唯一用户名和唯一电子邮件地址
passwordFormat	该特性决定了密码存储在数据库中的方式。它支持如下格式： Clear：密码以纯文本形式存储 Encrypted：密码以一种可逆格式加密，允许系统检索密码的明文表示 Hashed：密码以一种不可逆的、单向的算法加密。如果 passwordFormat 为 Hashed，用户不能检索他们的原密码。只能请求一个自动生成的新密码
maxInvalidPasswordAttempts	该特性指定用户在账户锁定前可输入的无效密码或无效安全答案的次数
minRequiredPasswordLength	该特性决定密码的最小长度
minRequiredNonalphanumericCharacters	该特性决定密码中必须包含的非字母数字字符的最小个数
passwordAttemptWindow	该特性决定计数无效密码尝试的以分钟为单位的时间范围
passwordStrengthRegularExpression	该特性允许指定自定义正则表达式来强制使用强密码
minRequiredPasswordLength	该特性决定密码的最小长度

要修改 Planet Wrox 应用程序的这些设置，可以直接修改 machine.config。不过，强烈建议您不要这样做。由于这一文件应用于整个机器，因此可能引发大量不必要的副作用，甚至使 ASP.NET 处于瘫痪状态。

您应该重新配置 web.config 文件中的<membership>元素，只为当前应用程序修改必要的特性。在下一个“试一试”练习中，将介绍如何为 Planet Wrox 应用程序重新配置成员提供者。

试一试

配置成员

在这个简短练习中，将介绍如何为 Planet Wrox 站点中的成员重写默认行为。首先从一个集中的配置文件获取成员设置的一个副本，然后修改其设置来删除安全问题和答案选项，并修改密码的规则。

(1) 定位到 machine.config 文件，其默认位置为 C:\Windows\Microsoft.NET\Framework\v4.0.30128\Config。如果您是在另一个文件夹或驱动器中安装 Windows，则要确保更改相应的路径。如果使用的是 Windows 的 64 位版本，那么 Framework 的文件夹名则是 Framework64。如果您的系统管理员不允许您访问计算机上的这一文件，也不用担心。您可以直接在 web.config 文件中输入第(6)步的代码。另外，如果 v4.0 后面的框架版本号稍有不同，也不用担心；您可以获得一个比本书中使用的版本更新的.NET 版本。使用 Windows 资源管理器找到精确的文件夹名。

(2) 用记事本打开 machine.config 文件，并在文件结尾定位到<system.web>下的<membership>元素。

(3) 将整个<membership>元素复制到剪贴板。

(4) 返回到 VWD，打开 web.config 文件，将<membership>元素粘贴到<authentication>元素之前，但是仍然位于<system.web>元素中。除了一些不同的格式(可以将所有设置放到一行)，所得的配置设置如本章开头所示。

(5) 将 minRequiredPasswordLength 修改为 6 并设置 requiresQuestionAndAnswer 为 false。

(6) 在<add>元素前添加自结束元素<clear />。完成之后的配置设置如下所示：

```
<membership>
  <providers>
    <clear />
    <add name="AspNetSqlMembershipProvider"
      type="System.Web.Security.SqlMembershipProvider, System.Web,
        Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="LocalSqlServer"
      enablePasswordRetrieval="false"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="false"
      applicationName="/"
      requiresUniqueEmail="false"
      passwordFormat="Hashed"
      maxInvalidPasswordAttempts="5"
      minRequiredPasswordLength="6"
      minRequiredNonalphanumericCharacters="1"
      passwordAttemptWindow="10"
      passwordStrengthRegularExpression=""
    />
  </providers>
</membership>
<authentication mode="Forms"/>
```

(7) 保存所有更改并在浏览器中请求 SignUp.aspx。注意，安全问题和答案不再像本章开头的图 16-4 所示的那样在 Sign Up 表单中出现。

(8) 填写该表单，但在密码字段中输入像 pass 这样的简短字符。

(9) 单击 Create User 按钮。注意，这时控件通过在其下显示一条正确的错误消息，要求按照 web.config 中的定义输入一个最少 6 个字符的密码，如图 16-15 所示。

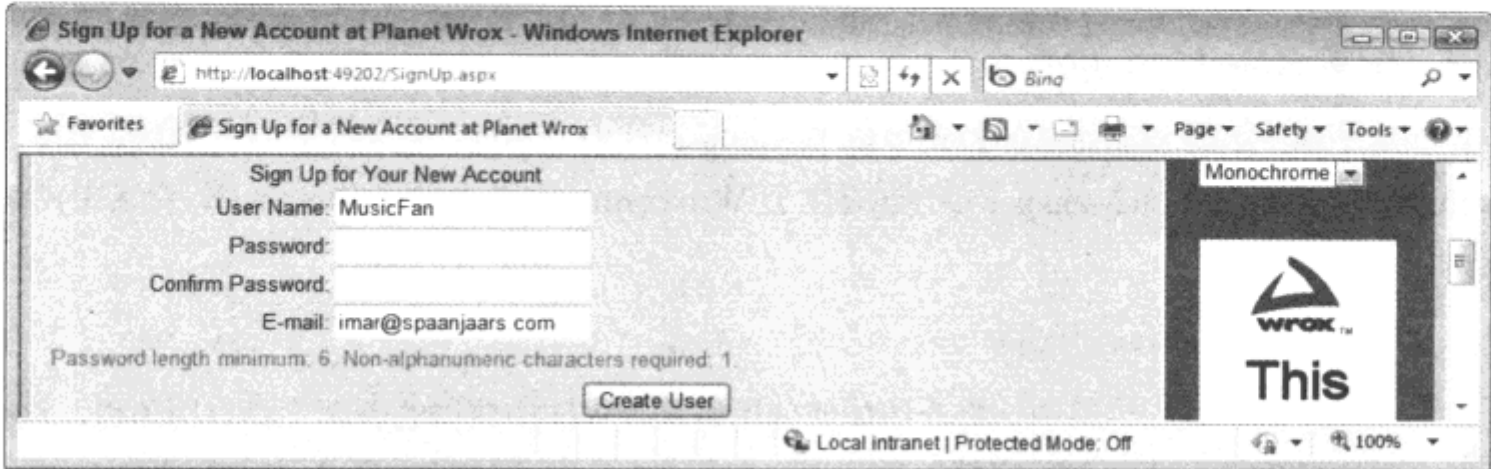


图 16-15

如果仍然看到安全问题和答案文本框，则需要重新启动浏览器，或者 VWD。

(10) 输入一个至少 6 个字符的并至少有一个非字母数字字符(如#或%)的密码，并再次单击 Create User 按钮。这次，密码就被接受了，账户也得以创建。

工作原理

ASP.NET 的配置设置以一种分层次的方式工作。这意味着在高级别(如在 machine.config 中)定义的设置可应用于计算机中的所有 Web 应用程序和 Web 站点。Planet Wrox 站点会继承 machine.config 中定义的设置，包括对成员提供者的设置。要修改单个 Web 站点中的行为，则需要创建原始设置的副本，然后修改必要的特性。

不过，为了这样做，首先要删除原始设置。方法就是使用<clear />元素：

```
<providers>
  <clear />
  <add name="AspNetSqlMembershipProvider" ...
```

在这里，<clear />的意思就是“删除配置层次结构中已添加的任何提供者设置”。因此，在<clear />元素之后，<providers>元素被看作是空的，允许再添加同样的提供者，但这次带有不同的设置。

尽管看上去这些工作只是修改一些设置，但当您需要应用程序与不同数据库或数据库服务器通信时，会发现重写您自己 Web 站点中的设置作用巨大。附录 B 介绍了如何操作该设置。

到现在为止，您已经学习了如何让用户注册账户，以便可以登录 Web 站点。但如何区分系统中的不同用户呢？如何阻止未经授权的用户访问特定文件夹(如 Management 文件夹)呢？答案就是使用角色管理器，这是随 ASP.NET 提供的另一个应用程序服务。

16.3 Role Manager

尽管现在用户可以注册并登录网站了，但如果能区分这些用户将会更好。这样，您就可以给一个或一小部分用户访问 Management 文件夹的权限，只有他们可更改评论和流派。通过随 ASP.NET 提供的 Role Manager(角色管理器)，这可以轻松实现。Role Manager 使您可为用户指派不同的角色。然后您可以使用这些角色打开或关闭站点中的特定功能。例如，可以阻止除了 Managers 角色中的用

户以外的所有用户访问 Management 文件。另外，可以像前面所看到的，使用 LoginView 根据用户所在的角色显示不同内容。

16.3.1 配置 Role Manager

和成员一样，Role Manager 的设置也位于 config 文件中。不过，它不是默认开启的，因此需要在 web.config 文件的<roleManager>元素(应位于 Web.config 文件中的<system.web>元素中)中显式设置。

```
<roleManager enabled="true" />
```

在后一个“试一试”练习中，将介绍如何使用可从 VWD 中获得的管理工具启用角色，这样就不需要直接手动修改 web.config 文件。

在启用角色后，您就可以使用多种方法将用户指派给不同角色。

- 使用 Web Site Administration Tool，通常称为 WSAT
- 使用较新的 Windows 版本上的 IIS(Windows Web 服务器)，第 19 章将讨论相关内容
- 以编程方式使用 Role Manager API(Application Programming Interface，应用程序编程接口)

使用 Role Manager API 管理角色的内容部不在本书的讨论范围内。如果您想了解这方面的内容，可以参考清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C#和 VB.NET(第 7 版)》一书(ISBN: 978-7-302-23524-8)，或者参考作者编写的 *ASP.NET 2.0 Instant Results* 一书(ISBN: 978-0-471-74951-6)中的 Bug Base 一章。虽然后一本书针对的是 ASP.NET 2.0，但是其中的很多概念仍然适用于 ASP.NET 4。

Web Site Administration Tool(WSAT)使用较多，下一节将详细讨论。WSAT 只在本地机器上可用，且作为 VWD 中的菜单快捷方式使用。因此，它适合于在开发时设置初始用户和角色，但不适合于在产品环境中管理用户。

16.3.2 使用 WSAT 管理用户

WSAT 工具随 VWD 提供，可通过 Website 菜单使用。该工具用于下列任务：

- 管理用户
- 管理角色
- 管理访问规则——例如，决定什么用户可访问什么文件和文件夹
- 配置应用程序、邮件和调试设置
- 使站点脱机，这样用户就不能请求任何页面，而是获得一个友好的错误消息

把使用 WSAT 作的一些更改保存在应用程序的 web.config 文件中。而其他一些设置，如用户和角色，存储在已配置的提供者的数据库中。

在下一个“试一试”练习中，将介绍如何启动并使用 WSAT。还将介绍如何创建新角色和新用户账户，并将该用户指派给该角色。

试一试

使用 WSAT 管理用户账户和角色

为了防止未经授权的用户访问 Management 文件夹，需要创建一个允许访问该文件夹的角色。在建立好这个角色后，可以授权该角色中的所有用户访问该文件夹，而将其他用户排除在外。在这个“试一试”练习中，您将学习如何创建 Managers 角色并给它指派一个用户。在后一个“试一试”

练习中，将介绍如何限制只允许 Managers 角色访问 Management 文件夹。

(1) 从 VWD 中，选择 Website | ASP.NET Configuration 命令。浏览器将打开并显示 Web Site Administration Tool，如图 16-16 所示。

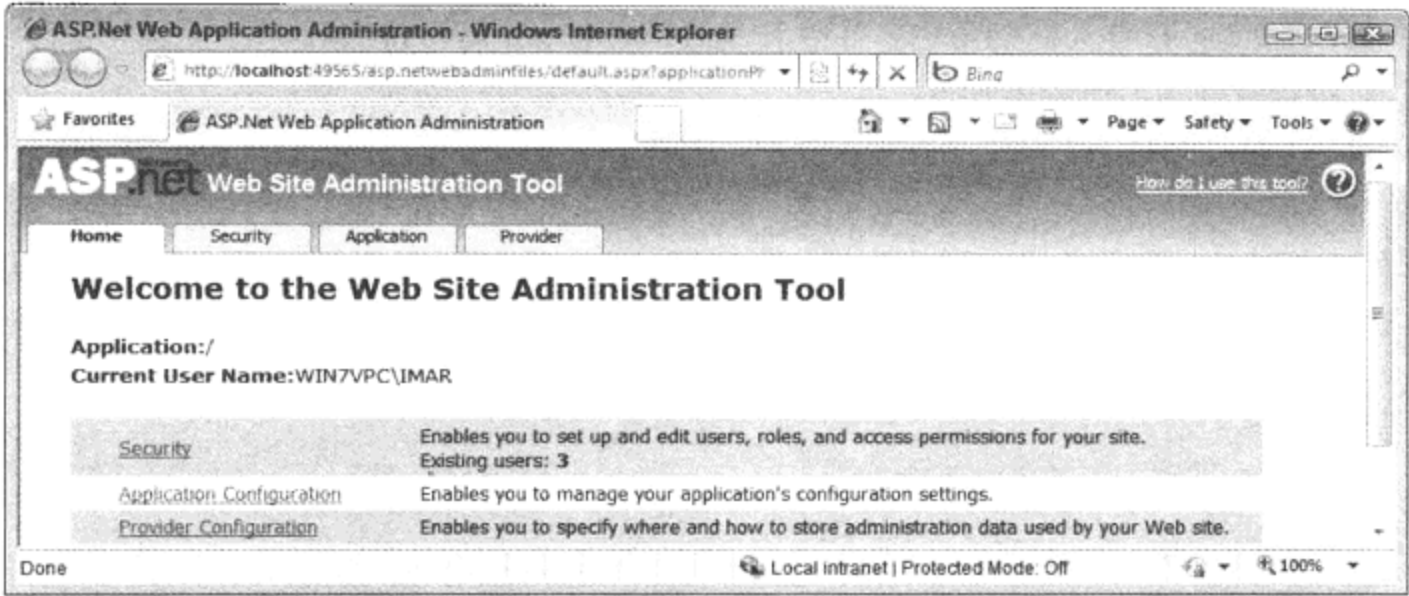


图 16-16

(2) 在其右上角可找到一个 Help 链接，它将把您带至一个描述如何使用该工具的帮助文件。在应用程序徽标之下有 4 个选项卡：Home、Security、Application 和 Provider。Home 选项卡用于将您带回图 16-16 所示的启动页面。Application 选项卡用于配置不同的应用程序设置，而 Provider 选项卡允许为应用程序重新配置选择的提供者。在这个练习中，最重要的是 Security 选项卡。切换至该选项卡，将看到如图 16-17 所示的屏幕。

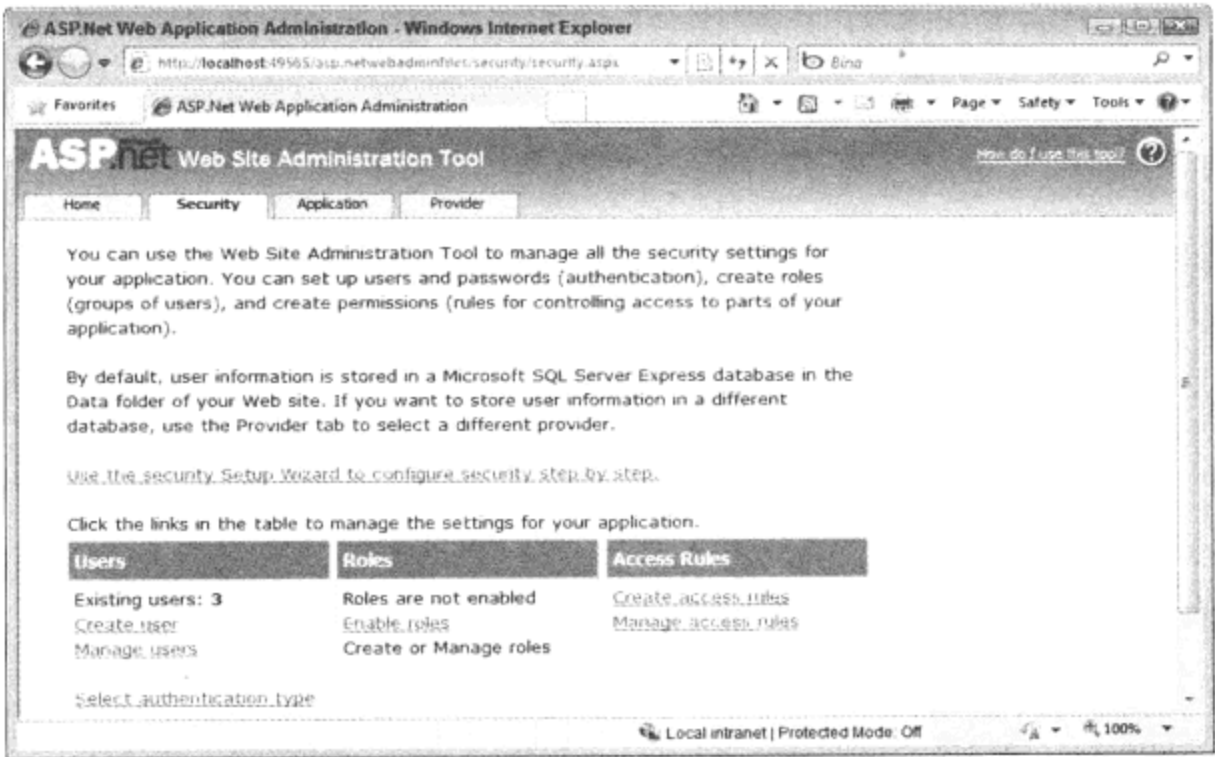


图 16-17

该屏幕的底部分为 3 个部分：Users、Roles 和 Access Rules。下面将介绍如何使用 Users 和 Roles。Access Rules 用于向用户或角色关闭或开放 Web 站点的特定部分。本章没有介绍如何使用它，不过后一个“试一试”练习中将介绍如何直接在 web.config 文件中更改其中一些设置。

(3) 确定在 Users 部分下方有 Create user 和 Manage users 链接。如果没有这些链接，而是显示一个有关 Windows 身份验证的提示，那么单击 Select authentication type 链接，然后选择 From the Internet

选项，最后单击 Done 按钮，此时屏幕将如图 16-17 所示。

(4) 在 Roles 部分，单击 Enable Roles 链接。该页面重新加载并提供一个带有 Create or Manage Roles 文本的链接。单击该链接，打开如图 16-18 所示的 Create New Role 页面。

(5) 输入 Managers 作为新的角色名并单击 Add Role 按钮。这时将出现新的角色。单击页面右下角的 Back 按钮返回主 Security 页面。

(6) 单击 Users 部分下方的 Create user 链接。您将被带至一个页面，在该页面中可以输入新用户的详细信息并将该用户指派给 Managers 角色。输入 Manager 作为用户名。对于密码，输入满足之前配置的密码规则的内容。像 Manager##123 这样的密码是可行的。输入您自己的电子邮件地址，并且不要忘了在右侧的角色列表中选中 Managers 角色名。

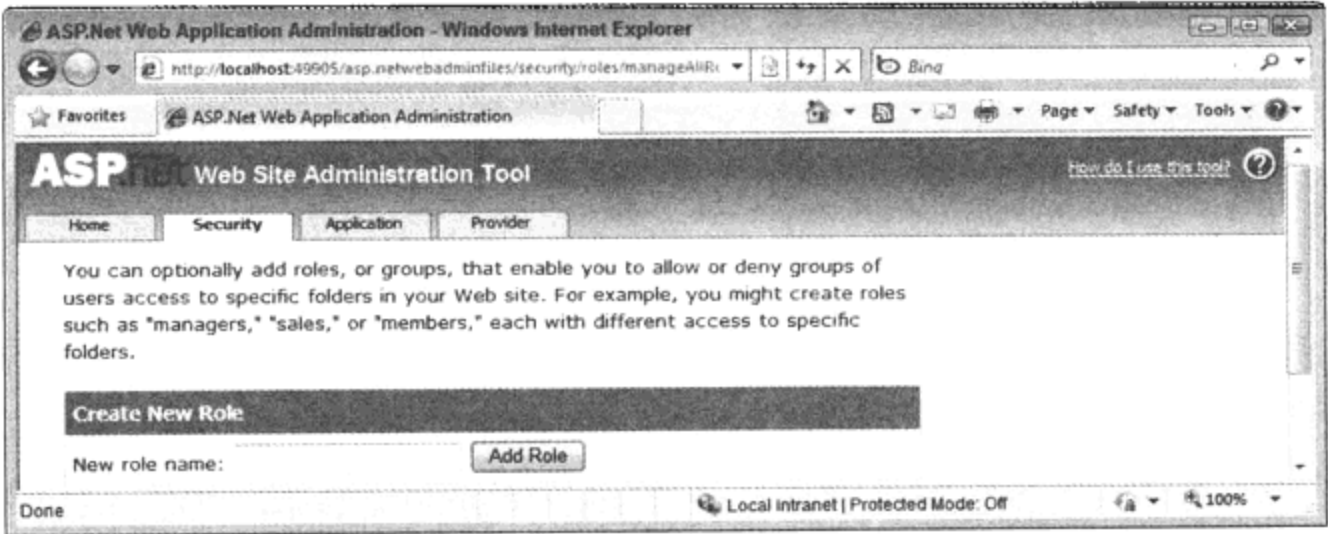


图 16-18

(7) 单击 Create User 按钮将该用户添加到系统中并单击确认页面上的 Continue 按钮。单击页面底部的 Back 按钮，直至返回到主 Security 页面。

(8) 在 Security 页面上单击 Manage Users 链接。您将被带至一个显示系统中可用用户列表的页面，如图 16-19 所示。

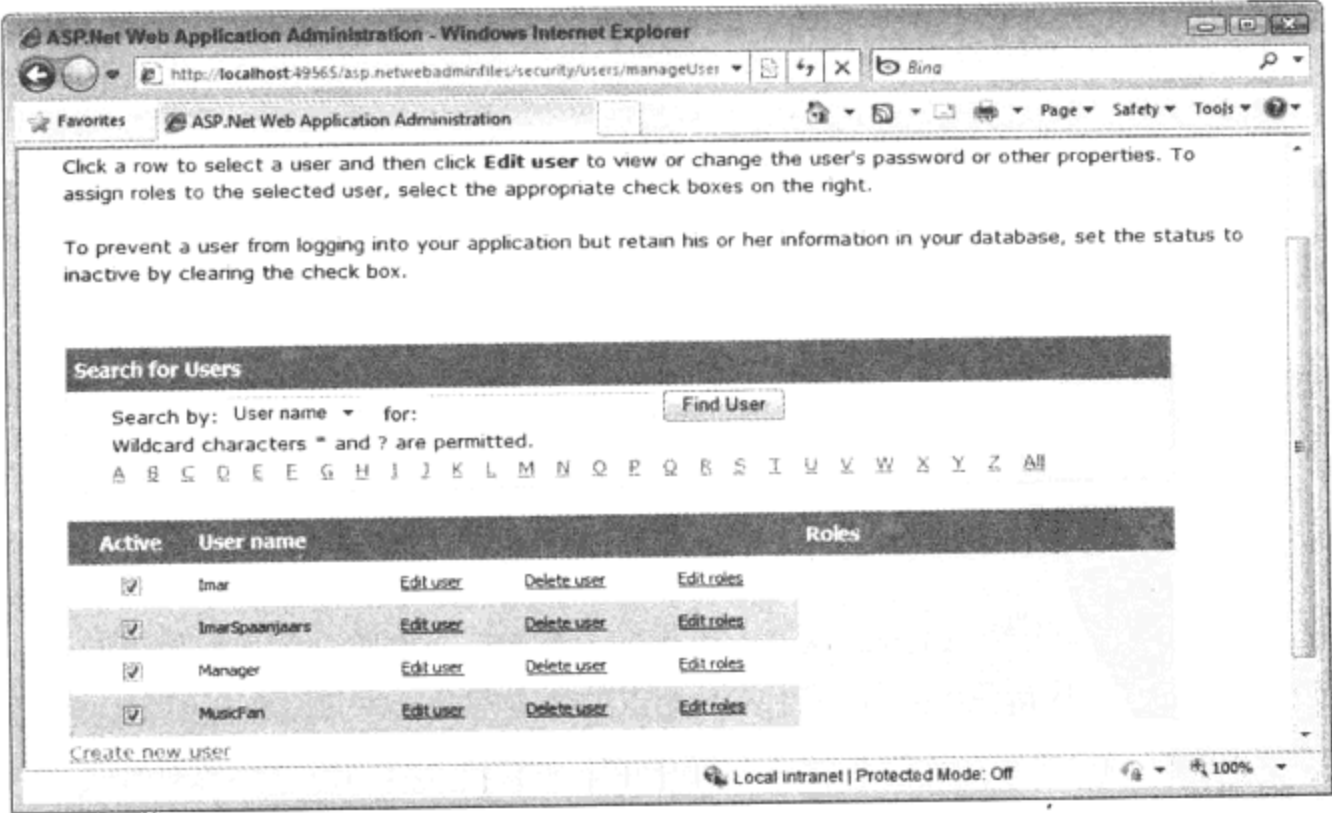


图 16-19

在这里可以编辑、启用、禁用和删除已有用户。例如，如果前面在 `CreateUserWizard` 中将 `DisableCreatedUser` 设置为 `True`，就可以在这里通过选中用户名前的复选框启用这些用户，如图 16-19 所示。通过单击 `Edit Roles` 链接，可以更改指派给该用户的角色。使用用户列表上的筛选控件和字母表，在面对大量用户账户时可以快速搜索特定用户。

(9) 要查看用户和角色的最终位置，可关闭浏览器并返回到 VWD。在 `Solution Explorer` 中，双击 `App_Data` 文件夹中的 `ASPNETDB.MDF` 数据库，然后在 `Database Explorer` 窗口中打开它。然后展开 `Tables` 节点，右击 `aspnet_Roles` 表并选择 `Show Table Data`。第(5)步创建的角色将被列出。打开 `aspnet_Membership` 和 `aspnet_UsersInRoles` 这些表，并检查它们所包含的数据。在第一个表中，将看到第(6)、(7)步创建的用户账户，而第二个表中包含新角色和用户账户的关系。

工作原理

和登录控件一样，`WSAT` 最终与 `web.config` 文件中已配置的提供者通信。在 `Planet Wrox` 应用程序的示例中，它是与 `AspNetSqlMembershipProvider` 通信，而后者又与 `LocalSqlServer` 连接字符串标识的 `SQL Server Express` 数据库进行通信。而创建的角色和用户存储在数据库的各种 `aspnet_*`表中。使用 `WSAT` 创建的用户最终和使用 `CreateUserWizard` 控件创建的用户位于相同的位置。事实上，`WSAT` 是使用 `CreateUserWizard` 创建新用户账户的。这意味着在 `WSAT` 中输入的任何用户都可以通过标准的 `Login.aspx` 页面登录。在后一个“试一试”练习中，将使用本练习中创建的 `Manager` 账户登录站点并访问 `Management` 文件夹。

为了使用本练习中创建的角色，有一些选项可选择。首先，可通过 `web.config` 文件中的设置使用角色名阻止对 Web 应用程序中特定文件夹的访问。其次，可在 `LoginView` 这些控件中使用角色向不同的用户显示不同的内容。最后，可使用 `Role API` 检查当前用户是否在特定角色中。这样您就可以很好地控制向具体有特权的用户提供的内容或功能。

接下来将介绍如何阻止对 `Management` 文件夹的访问并修改 `LoginView`，而 `Role API` 的使用将在下一个“试一试”练习中讨论。

16.3.3 配置 Web 应用程序使用角色

在 `WSAT` 的 `Security` 页面上，有一个 `Access Rules` 部分。这一部分允许关闭或开放站点中的资源。可以使用这一部分定义这类规则：“阻止除 `Managers` 角色中的用户外的任何用户访问这个文件夹”或是“除了 `Members` 角色和 `Joe` 账户，任何人都可访问这一文件”。我们可以很直观地使用这一工具，所以也很容易建立不同的规则。不过，它也有一个缺点：它将安全设置存储在不同的 `web.config` 文件中，配置的每个子文件夹都有一个。

这就给概览所有不同的安全设置带来了一些困难。但是，`ASP.NET` 允许使用 `<location>` 元素将相同的设置配置到主 `web.config` 文件中。`<location>` 元素有一个 `path` 特性，它指向您想进行不同配置的文件或文件夹。也可以对 `web.config` 文件的其他许多(但并非全部)设置使用 `<location>` 元素(例如，可以在主 `web.config` 文件中为 `Management` 文件夹设置 `<pages>` 元素的 `theme` 特性)。对于接下来的“试

一试” 练习，将只设置与安全相关的<location>的子元素。

试一试 阻止对 Management 文件夹的访问

显然，您不希望任何人都可以修改您在自己网站上发布的评论和流派。因此，有必要阻止除被指派 Managers 角色以外的任何人访问 Management 文件夹。在这个练习中，将介绍如何修改 web.config，使得只有前面指派到 Managers 角色的用户账户可访问 Management 文件夹及其包含的文件。

(1) 打开站点根文件夹下的 web.config 文件。向下滚动至</configuration>结束标记，在其前面输入一个<location>元素。添加一个 path 特性到该元素并设置其值为 Management。注意，IntelliSense 会帮助完成该元素并找到其特性。输入下面的设置完成配置：

```
</system.net>
<location path="Management">
  <system.web>
    <authorization>
      <allow roles="Managers" />
      <deny users="*" />
    </authorization>
  </system.web>
</location>
</configuration>
```

(2) 保存并关闭 web.config 文件。

(3) 在 Design 视图中打开站点的主母版页(Fronted.master)，并滚动至该文件的末尾。选择 LoginView 控件，然后打开其 Smart Tasks 面板。在该面板顶部，单击 Edit RoleGroups 链接，如图 16-20 所示。

(4) 在打开的对话框中，单击 Add 按钮插入一个新的 RoleGroup，然后设置该组的 Roles 属性为 Managers，如图 16-21 所示。

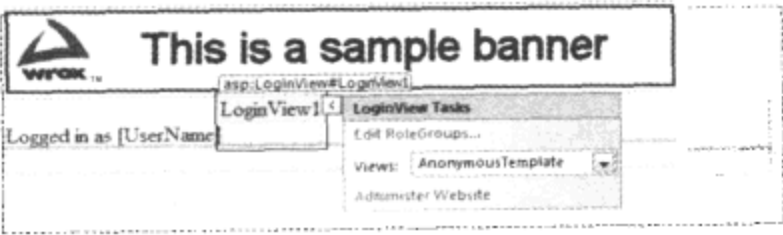


图 16-20

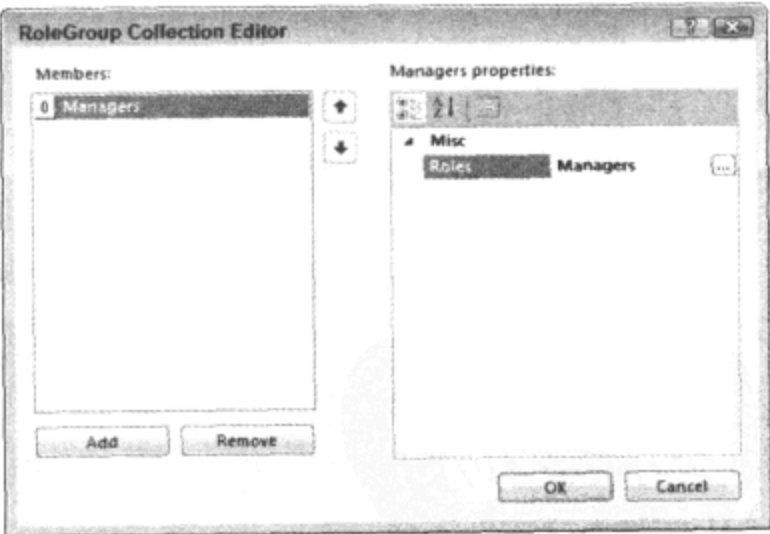


图 16-21

(5) 单击 OK 按钮以插入该 RoleGroup 并返回到 Design 视图。

(6) 仍在 LoginView 的 Smart Tasks 面板上，从 Views 下拉列表中选择 RoleGroup[0]- Managers。这将切换控件的当前模板至 Managers 的 RoleGroup，然后可添加只对 Managers 可见的内容。

(7) 从 Toolbox 的 Standard 类别中拖放一个 HyperLink 控件到 LoginView 中。使用 Properties 面板, 设置该 HyperLink 的 Text 属性为 Manage Site, 并设置 NavigateUrl 为~/Management/Default.aspx(通过单击其上有省略号的小按钮可对 HyperLink 使用 URL 选取器(picker))。切换至 Markup 视图, 在 HyperLink 控件的结束标记后输入单词 or, 后面是可从 Toolbox 中拖动的 LoginStatus 控件。或者可以复制 LoggedInTemplate 中的已有代码。

完成后, LoginView 应包含下列代码:

```
<asp:LoginView ID="LoginView1" runat="server">
...
</LoggedInTemplate>
<RoleGroups>
  <asp:RoleGroup Roles="Managers">
    <ContentTemplate>
      <asp:HyperLink ID="HyperLink1" runat="server"
        NavigateUrl="~/Management/Default.aspx">Manage Site</asp:HyperLink> or
      <asp:LoginStatus ID="LoginStatus2" runat="server" />
    </ContentTemplate>
  </asp:RoleGroup>
</RoleGroups>
...
```

(8) 保存所有更改, 然后在浏览器中请求站点的主页(Default.aspx), 确认当前未登录(检查页面的页脚, 如果有必要的话就单击 Logout 链接)。

(9) 单击 Menu 或 TreeView 上的 Login 链接, 然后用本章前面创建的 Manager 账户登录。确保未选中 Remember Me 选项。页面得到刷新并在每个页面的页脚显示 Manage Site 链接, 如图 16-22 所示。



图 16-22

如果在页脚区域没有看到 Manage Site 和 Logout 链接, 则关闭全部浏览器窗口, 返回到 WSAT(使用 Website|ASP.NET Configuration in VWD 命令)中, 确保所使用的账户指派给 Managers 角色。另外, 可能还需要使用 Windows 托盘栏停止使用内置的 Web 开发服务器, 然后再次打开 Login 页面。

(10) 单击 Manage Site 链接来打开 Web 站点的 Management 部分。将页面的当前 URL 从浏览器的地址栏复制到剪贴板(其格式应为 http://localhost:49666/Management/Default.aspx)。单击浏览器中的 Back 按钮以返回到主页, 然后单击页脚中的 Logout 按钮。关闭所有浏览器窗口并再次打开一个新实例(可以通过 Windows 开始菜单或桌面(如果上面有快捷方式)完成, 或是在 VWD 中右击一个页面, 然后选择 View in Browser 命令)。

(11) 粘贴刚才复制的地址到浏览器窗口的地址栏中并按下 Enter 键。浏览器不会打开如下页面:

```
http://localhost:49666/Management/Default.aspx
```

而是会打开如下的 Login 页面:

`http://localhost:49666/login.aspx?ReturnUrl=%2fManagement%2fDefault.aspx`

注意，最初请求的页面(`Management/Default.aspx`)已附加到查询字符串中。地址中的正斜杠(/)已自动编码为 URL 安全的对应字符：`%2f`。用 Manager 账户登录，您将看到 Management 部分再次出现。

工作原理

本“试一试”练习中有几个值得一看的有趣地方。首先，查看添加到 `web.config` 文件的设置，它限制了对 Management 文件夹的访问：

```
<location path="Management">
  <system.web>
    <authorization>
      <allow roles="Managers" />
      <deny users="*" />
    </authorization>
  </system.web>
</location>
```

当 ASP.NET 运行时处理页面请求时，它会检查各种配置文件来确定是否允许当前用户访问该资源。对于对 Management 文件夹中文件的请求，它会遇到`<location>`元素中的规则集。它首先扫描各种规则(带有 `roles` 或 `users` 特性的 `allow` 和 `deny` 元素，指定规则影响的用户或角色)，一旦找到规则，它就会停止扫描进程并应用该规则。如果规则都不满足，那么就授予访问权限。因此，有必要在规则最后用一个 `deny` 规则来阻止前面没有授予访问权限的所有其他用户。

当未通过身份验证的用户登录时，第一条规则就不匹配，因为匿名用户不是 Managers 角色的成员。然后该用户就被拒绝访问了，因为有阻止所有用户(用星号(*)表示)的拒绝规则。

在作为 Manager 登录并请求相同资源后，规则集再次被扫描。然后运行时找到授权 Managers 角色访问的 `allow` 元素并立即允许访问。最后阻止所有其他用户访问的规则甚至都未被检查。除了特定角色或用户名以及表示所有用户的星号外，您还可以使用问号(?)表示未通过身份验证的用户或匿名用户。例如，为了让所有登录用户访问 Reviews 文件夹，而不管其所在角色，并且阻止所有其他用户，可以添加下列`<location>`元素到配置文件中：

```
<location path="Reviews">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
```

这就拒绝了所有未登录的用户。由于默认规则是如果当前用户未与前面规则匹配，就授予对资源的访问权限，因此所有登录用户可成功访问 Reviews 文件夹中的文件。

通过使用逗号进行分隔，可以在 `roles` 和 `users` 特性中指定多个角色和用户名。

理解 LoginView 中的 RoleGroups 元素的工作原理很重要。尽管您可以指定多个应用于某个特定

用户的 RoleGroup 元素，但只有第一个匹配规则的 RoleGroup 会显示。考虑将一个名为 Lucas 的用户指派到 WebMasters 角色和 Managers 角色以及一个带有下列 LoginView 的 Web 页面中：

```
<asp:LoginView ID="LoginView1" runat="server">
  <RoleGroups>
    <asp:RoleGroup Roles="Managers">
      <ContentTemplate>
        <!-- Content for Managers here -->
      </ContentTemplate>
    </asp:RoleGroup>
    <asp:RoleGroup Roles="WebMasters">
      <ContentTemplate>
        <!-- Content for WebMasters here -->
      </ContentTemplate>
    </asp:RoleGroup>
  </RoleGroups>
</asp:LoginView>
```

通过这一代码，用户 Lucas 只看到第一个 RoleGroup 的内容，尽管它也被指派到 WebMasters 角色中。

16.3.4 以编程方式检查角色

尽管可以很容易地使用 LoginView 控件修改允许用户在运行时看到的内容，但它并不总是能满足要求。有时，您需要根据某人的角色成员资格以编程方式控制显示的数据。您可以以多种方式访问当前用户的角色信息。首先，可以通过当前页面或用户控件访问 User 属性的 IsInRole 方法，如下所示：

```
VB.NET
If User.IsInRole("Managers") Then
  ' This code runs for Managers only
End If
C#
if (User.IsInRole("Managers"))
{
  // This code runs for Managers only
}
```

或者，可访问包含可直接访问的大量静态方法的 Roles 类。下列代码的作用等同于前一示例：

```
VB.NET
If Roles.IsUserInRole("Managers") Then
  ' This code runs for Managers only
End If
C#
if (Roles.IsUserInRole("Managers"))
{
  // This code runs for Managers only
}
```

两者的最大差别在于，如果想让后者起作用，需要启用 Role Manager。对于 Planet Wrox 站点来

说，由于已经启用了 Role Manager，所以可选择其中任何一种方法。不过，对于自定义的解决方案来说，其中没有使用角色提供者，而是使用您自己的自定义解决方案，只能使用第一种代码结构。

除了 IsUserInRole 方法，Roles 类包含了大量其他方法，这些方法允许您以编程的方式使用角色。例如，可以创建和删除角色、将用户指派给角色或从角色中删除用户，以及获取指派给某一特定角色的用户列表。要获取有关 Roles API 的更多信息，可参阅 MSDN 文档(网址为 <http://tinyurl.com/RolesAPI>)或是 Wrox 出版社出版的：Professional ASP.NET 3.5 Security, Membership, and Role Management with C# and VB 一书(ISBN: 978-0-470-37930-1)。虽然概述针对的是 ASP.NET 4，但是书中讨论的大部分主题仍然适用于 ASP.NET 4。

在下一个“试一试”练习中，将学习如何修改相册页面，这样作为 Managers 登录的用户可从相册中删除图片。其他用户不能删除图片，因为 Delete 按钮是对他们隐藏的。

试一试

在运行时运用 IsUserInRole 检查角色

接下来的“试一试”练习是以编程的方式检查用户角色，从而隐藏或显示 Delete 按钮。尽管可以通过使用一个带有不同模板和 RoleGroups 的 LoginView 重新创建这一示例，本练习还是使用了自定义的代码，这使得后面可更容易地修改代码，使得特定相册的所有者也能删除他们自己的图片。现在，所添加的 Edit Photo Album 链接是对所有用户可见的，但是第 17 章将看到如何解决这个问题。

(1) 在 Markup 视图中打开 PhotoAlbums 文件夹中的 Default.aspx，在 ListView 控件中的结束标记下输入两个 HTML 换行标记(使用 br 代码段)，后面跟一个 HyperLink 控件。将其 ID 属性设置为 EditLink，将其 Text 属性设置为 Edit Photo Album。在第下一步将指派 NavigateUrl。

```
<br /><br />
<asp:HyperLink ID="EditLink" runat="server" Text="Edit Photo Album" />
```

(2) 切换至 Design 视图，选择 ListView 控件，打开其 Properties 面板，然后切换到 Events 选项卡。双击 DataBound 为该事件建立一个处理程序。在 VWD 创建的处理程序内，添加下面的代码：

```
VB.NET
Protected Sub ListView1_DataBound(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles ListView1.DataBound
    If Not String.IsNullOrEmpty(DropDownList1.SelectedValue) Then
        EditLink.NavigateUrl = String.Format(
            "~/ManagePhotoAlbum.aspx?PhotoAlbumId={0}", DropDownList1.SelectedValue)
        EditLink.Visible = True
    Else
        EditLink.Visible = False
    End If
End Sub

C#
protected void ListView1_DataBound(object sender, EventArgs e)
{
    if (!string.IsNullOrEmpty(DropDownList1.SelectedValue))
    {
        EditLink.NavigateUrl = string.Format(
            "~/ManagePhotoAlbum.aspx?PhotoAlbumId={0}", DropDownList1.SelectedValue);
        EditLink.Visible = true;
    }
}
```



```
else
{
    EditLink.Visible = false;
}
}
```

- (3) 打开站点根文件夹下的 ManagePhotoAlbum.aspx，然后切换到 Design 视图。选择 ListView，然后打开其 Properties 面板。切换到 Events 选项卡，双击 ItemCreated 事件为其建立一个处理程序。
- (4) 在 Code Behind 文件顶部，为 System.Web.Security 名称空间添加一个 Imports/using 语句。

VB.NET

```
Imports System.Web.Security
```

C#

```
using System.Web.Security;
```

- (5) 添加下列代码到 VWD 创建的事件处理程序中：

VB.NET

```
Protected Sub ListView1_ItemCreated(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.ListViewItemEventArgs) _
    Handles ListView1.ItemCreated
    Select Case e.Item.ItemType
    Case ListViewItemType.DataItem
        Dim deleteButton As Button =
            CType(e.Item.FindControl("DeleteButton"), Button)
        deleteButton.Visible = Roles.IsUserInRole("Managers")
    End Select
End Sub
```

C#

```
protected void ListView1_ItemCreated(object sender, ListViewItemEventArgs e)
{
    switch (e.Item.ItemType)
    {
        case ListViewItemType.DataItem:
            Button deleteButton = (Button)e.Item.FindControl("DeleteButton");
            deleteButton.Visible = Roles.IsUserInRole("Managers");
            break;
    }
}
```

- (6) 保存所有更改，然后按 Ctrl+F5 组合键在浏览器中请求 PhotoAlbums 文件夹中的 Default.aspx。如果尚未登录，单击主 Menu 或者 TreeView 中的 Login 链接，使用在本章前面创建的 Manager 账户登录，然后返回至 Gig Pics 页面。
- (7) 从下拉列表中选择一个相册，页面重新加载并显示该相册中的图片。
- (8) 单击页面底部的 Edit Photo Album 链接。图 16-23 显示了现在每张图片与一个 Delete 按钮相关联，当单击 Delete 按钮时可以删除该图片。现在单击 Delete 按钮以确认该按钮有效。



图 16-23

(9) 再次单击页脚中的 Logout 链接至 Login 页面。使用您在本章前面创建的账户登录，该账户不是 Managers 角色的成员。返回到 Gig Pics 页面，从列表中选择 一个相册，然后单击 Edit Photo Album 链接。这一次，没有显示 Delete 按钮，因为登录的账户没有指派给 Managers 角色。

工作原理

本“试一试”练习中的大部分代码都是我们所熟悉的。其中介绍了如何用 EntityDataSource 和 ListView 控件删除项。还介绍了如何处理 ItemCreated 和其他事件，以及如何使用 FindControl 搜索某项中的控件。

本示例中的新内容是检查当前用户是否为 Managers 角色的成员的方式：

```
VB.NET
Dim deleteButton As Button = CType(e.Item.FindControl("DeleteButton"), Button)
deleteButton.Visible = Roles.IsUserInRole("Managers")
C#
Button deleteButton = (Button)e.Item.FindControl("DeleteButton");
deleteButton.Visible = Roles.IsUserInRole("Managers");
```

IsUserInRole 方法返回一个 Boolean 值，表明当前用户是否为 Manager。如果该方法返回 True，就意味着 Button 的 Visible 属性设置为 True。如果该方法返回 False，按钮就被隐藏，用户不能从相册中删除图片。

16.4 有关安全性的实用提示

下面列出了一些有关安全性的实用提示：

- 尽管安全性的概念在本章中的后面部分才引入，不过您不应在事后才考虑它。为了确保创建一个可靠而安全的应用程序，您需要从开发 Web 站点的一开始就关注安全性。最好尽早决定是否希望有一些区域只允许特定用户访问，以及是否将强制用户在访问前必须先站在站点注册账户。越晚引入这些概念，集成这一功能时面对的困难就越多。
- 设法将像 ASPX 页面这样的资源聚集到提供系统中角色的文件夹下。例如 Planet Wrox Web 站点中的 Management 文件夹。所有与站点的管理相关的页面都放在单个文件夹中，这使得可以很容易地在 web.config 文件中使用一个<location>元素来阻止整个文件夹。如果您想保护的文件分散在 Web 站点中，那就需要更多的时间来配置应用程序，最终也不能清楚了解起作用的安全性设置。
- 在创建角色区分 Web 站点上的用户时，设法限制系统中不同角色的数目。将会发现，相较于只拥有一两个用户的大量角色，系统更容易管理进行了逻辑分组的少量角色。

16.5 本章小结

ASP.NET 站点中的安全性可以通过一些技术来实现，包括 Windows 身份验证(其中 Web 服务器帮助进行身份验证)或是第三方身份验证(其中，如 Microsoft Passport 这样的外部服务帮助验证用户)。还可以使用 Forms 身份验证，它是目前大多数 ASP.NET 网站的实际标准。

通常，安全性包括 3 个重要的概念：身份、身份验证和授权。它们共同决定了您是谁，以及您可以执行什么操作。

ASP.NET 成员提供者允许在中央数据库中，使用像 CreateUserWizard.PasswordRecovery 和 Login 这样可方便使用的控件，来创建和管理用户。

用户和角色是使用 WSAT 管理的，所以可像本章中那样为不同的角色指派用户账户。然后可以在 web.config 文件中使用简单的<location>元素，向特定角色中的成员开放 Web 站点中的特定资源。

各种登录控件可用于自定义用户可看到的内容。在第 17 章将更深入地讨论如何基于访问页面的用户创建动态页面。

16.6 练习

1. 身份验证和授权的区别是什么？
 2. Management 文件夹现在阻止除 Managers 角色中的用户以外的所有用户访问。如果您想向用户 John 和 Editors 角色中的所有用户开放文件夹，该如何对 web.config 文件进行更改？
 3. 假定您有一个 Web 站点，其 Login 页面只有单个 Login 控件。如何对 Login 控件进行更改，使得用户登录时能将他们导航到根目录下的 MyProfile.aspx？
 4. LoginView 和 LoginStatus 控件之间的区别是什么？何时使用它们？
- 练习的答案见附录 A。

本章要点回顾

应用程序服务	一组可以在 Web 应用程序中访问的 ASP.NET 服务，用于处理如成员、角色和配置文件管理等任务
身份验证	向系统证明身份的过程
授权	确定用户在系统中具有的权限的过程
登录控件	随 ASP.NET 一起提供的一组安全控件，用于注册、登录和恢复密码等
成员提供者	ASP.NET 应用程序服务的一种，处理与成员相关的任务(包括创建用户、登录等)
权限	决定用户在系统中可以执行哪些操作
提供者模型	一种模型，其中可以使用可互换的软件来完成特定的应用程序任务。通过配置，可以分配处理相同任务(但是使用的方式不同)的不同软件
角色管理器	ASP.NET 应用程序服务的一种，处理与角色相关的任务，包括创建角色、将用户指派给角色以及检查用户的成员角色

第 17 章

个性化 Web 站点

本章要点

- ASP.NET 中的 Profile 功能
- 如何在 Web 站点中创建和使用用户的配置文件
- 如何识别用户并向他们提供自定义的内容
- 如何访问站点中其他用户的配置文件

在 Web 上，比好的内容更胜一筹的当属好的个性化内容。在这个信息过量和大量站点竞争激烈的时代，了解您的访问者和知道提供个性化内容的必要性至关重要。通过好的个性化策略，提供人们所需的数据信息，可以创建一个满足您的用户期望的 Web 站点。个性化可用于许多不同的场景：在体育站点上，可使用它突出用户所喜爱球队的活动；在有关编程的站点上，可通过只向用户展示用他们偏爱的编程语言所编写的示例来个性化内容；而在新闻网站中，可以让用户选择一个或多个新闻类别(世界、当地、体育、商业、金融等)，使显示的内容符合用户的爱好。更个性化的做法是当某个类别中发布了新文章时，向用户发送电子邮件进行通知。

不过，个性化远远不是只存储个人偏好的内容和使内容迎合这些偏好。通过个性化，还可以跟踪其他用户信息，如姓名、生日、站点访问次数、用户在在线商店中购买的产品等。然后，可以使用这些详细信息进一步个性化 Web 页面，建立与访问者更紧密的联系。

在 Planet Wrox Web 站点中，简单而有效地实现了个性化。Reviews 页面只显示了用户感兴趣的那些流派的评论。要查看所有可用评论，用户仍可访问 All.aspx 页面，但如果访问个性化页面，将只看到他们真正喜欢的流派中的评论。

另外，用户可以输入他们自己的个人信息，如姓、名，以及个人简介。这些内容将显示在站点的 Photo Albums 详细信息页面中，由此我们就可以知道谁上传了某个特定的相册。

为支持向 Web 站点添加个性化功能，ASP.NET 4 包含了一个名为 Profile 的应用程序服务。通过该服务，可以使用非常少的代码为特定用户存储数据。

到本章结束时，您将能够掌握如何使用 Profile 带来的个性化功能创建动态的、个性化的 Web 站点。

17.1 Profile

ASP.NET Profile 是随 ASP.NET 提供的另一个应用程序服务。它使我们可存储和检索有关站点用户的信息，这些信息并不只是基本信息，如注册时输入的电子邮件地址和密码。通过 Profile，可以存储这样一些信息，如姓、名、生日等，本章后面将作介绍。通过跟踪数据所属的用户，ASP.NET 可以在用户下一次访问站点时将该数据映射到该用户，而不管该用户是在几分钟或几周后才访问站点。Profile 的优秀之处在于它允许存储注册用户以及匿名用户的数据。因此，即使访问者未进行注册，您也可以识别他们并存储他们的信息。

通过一个实际无代码的 API 可以访问用户配置文件中的信息。所需做的只是在主 web.config 文件中定义要跟踪的信息，然后由 Profile 完成其余工作。所有检索和存储数据库中的配置信息的交互操作都是自动处理的。

在 Web 应用程序中启用 Profile 是一个简单的两步骤过程：

(1) 在 web.config 文件中定义想为用户存储的信息。根据这一信息，ASP.NET 运行库即时生成和编译一个类，可用它访问自己定义的属性。然后它动态地将一个名为 Profile 的属性添加到 Web 站点中的页面，这样就可以轻松地从站点的每个页面中访问 Profile。

(2) 在应用程序中，直接用这个生成的类获取和存储当前用户的 Profile 信息。

ASP.NET Profile 默认与登录用户连接，不过，也可以为未经过身份验证的用户保存配置数据，本章后面将会介绍。

由于登录用户的数据默认存储在 cookie 中，因此用户需要使用支持 cookie 的浏览器，以便 ASP.NET Profile 服务正确运行。

在下一节中，将介绍如何在 web.config 文件中定义配置文件属性，以及如何在 Web 页面中访问它们。



注意：必须知道，内置的 Profile 功能只能用于 Web Site Project，而不能用于 Web Application Project。关于两者的区别，可以参考第 2 章。如果发现本章的示例都无法工作，请检查是否误建了一个 Web Application Project。最简单的检查方法是查看一个 Web 窗体的 Code Behind 文件。如果看到两个 Code Behind 文件(一个以页面名命名，并带有 .cs 或 .vb 扩展名，另一个带有一个额外的 Designer 扩展名)，那么就说明您创建了一个 Web Application Project。如果是这样，可以使用本书附带的源代码中的 Chapter 16 文件夹作为本章的起点。

17.1.1 配置 Profile

通过创建一个 <profile> 元素作为 <system.web> 元素的直接子元素，可在 web.config 中为 Web 站点定义一个配置文件。在 <profile> 标记之间，需要创建一个 <properties> 元素，它用于定义想通过 Profile 对象提供的属性。它有两种类型的属性：简单属性和复杂属性(也称为配置文件组)。

1. 创建简单配置文件属性

使用<add>元素可以定义简单属性为<properties>元素的直接子元素。下面的示例演示了如何创建一个可用于存储用户名字的属性和一个用于存储生日的属性。FirstName 属性可以被通过身份验证的用户和匿名用户所访问和设置，但生日属性只可被登录用户访问。

```
<system.web>
...
<profile>
  <properties>
    <add name="FirstName" allowAnonymous="True" />
    <add name="DateOfBirth" type="System.DateTime" />
  </properties>
</profile>
```

由于属性默认是 System.String 类型的，所以不需要为像名字这样的简单属性定义显式的类型。不过，对于其他类型，如 DateTime、Boolean、Integer 或自定义的类型，需要使用 type 特性及其包括名称空间的完全限定名(如 DateOfBirth 属性所示)显式定义类型。表 17-1 列出了<add>元素影响配置文件属性的最常见的特性。

表 17-1

特 性	说 明
name	该特性定义属性的名称，如 FirstName、DateOfBirth 等
type	该特性设置属性完整的.NET 类型名，如 System.String、System.Boolean 和 System.DateTime 等
allowAnonymous	该特性指定是否可为匿名用户设置该属性。默认值为 False。要设置该特性为 True，需要启用 anonymousIdentification，本章后面将讨论
defaultValue	该特性为属性定义默认值(如果还未显式设置)。如果省去该特性，配置文件属性则使用底层类型的默认值作为其默认值(例如，String 的默认值为 null，Int32 为 0 等)
readOnly	该特性指定配置文件属性是否可在运行时改变，默认为 Flase，表明可读取和写入该属性

除了简单属性外，还可以创建配置文件组，它允许将其他的简单属性组合在一起。

2. 创建配置文件组

配置文件组有两种用途：首先，它允许逻辑分组相关的属性。例如，可以创建一个名为 Address 的组，它有一些属性，如 Street、PostalCode 和 City 等。

它还允许属性具有相同名称，但要位于不同的组。例如，可以有两个分别名为 VisitAddress 和 PostalAddress 的组，它们都有 Street 和 PostalCode 这样的属性，这使得使用该 Profile 对象的开发人员可以更容易地找到相关信息。

要创建配置文件组，可向配置文件的<properties>元素中添加一个<group>元素，然后指定一个名称。<group>元素包含一个或多个简单属性。下列示例显示了 PostalAddress 配置文件组：

```
<properties>
  <add name="FirstName" />
```

```
<group name="PostalAddress">
  <add name="Street" />
  <add name="PostalCode" />
  <add name="City" />
  <add name="Country" />
</group>
</properties>
```

在<properties>标记中可以有多个组，但只可以有一个级别的组。这意味着，不能将一个<group>元素嵌套到另一个<group>或<add>元素中。

3. 使用非标准数据类型

除了前面列出的数据类型，如 String、DateTime 和 Integer，还可以使用自己(如在 App_Code 文件夹中)定义的类型。

和内置.NET 类型一样，需要使用包括名称空间和类名的完全限定名来引用类型。假定有一个名为 Preference 的类型，它包含了用户偏好的各种属性(在本例中作为自动属性实现)。要将这一类型包括到配置文件中，需要首先将它包装到一个名称空间中：

```
VB.NET
Namespace PlanetWrox
  Public Class Preference
    Public Property FavoriteColor As String
    ' Other properties go here
  End Class
End Namespace
```

```
C#
namespace PlanetWrox
{
  public class Preference
  {
    public string FavoriteColor { get; set; }
    // Other properties go here
  }
}
```

然后在一个<add />元素中引用该类型，如下所示：

```
<add name="Prefs" type="PlanetWrox.Preference" />
```

在使用泛型(generic)时，需要使用不同的语法在 Profile 配置中引用类型。第 5 章讨论了如何使用泛型字符串列表存储角色名：

```
VB.NET
Dim roles As New List(Of String)
...
roles.Add("Members")
```

```
C#
List<string> roles = new List<string>();
```

```
...
roles.Add("Members");
```

为了给配置文件指定一个泛型类型 List 的属性，需要使用一些特殊的语法。下列 web.config 中的设置创建了一个名为 FavoriteGenres 的配置文件属性，用来存储用户最喜爱的流派。在 VB.NET 中的形式是 List (Of Integer)，而在 C#中是 List<int>。

```
<add name="FavoriteGenres"
      type="System.Collections.Generic.List`1[System.Int32]" />
```

type 特性的第一部分看起来很普通。List 类位于 System.Collections.Generic 名称空间中，因此这里需要进行指定。不过，在类名(List)之后，可看到'1。这并不是印刷错误，而是.NET 在纯文本中引用泛型类型的方法。为了定义基于泛型类型的属性，需要使用反勾号，后跟一个 1。反勾号一般位于键盘上 1 键的左侧。然后'1 之后紧跟一对方括号，其中包含实际想用于列表的类型。在这个配置文件属性中指定的类型与如下 VB.NET 和 C#中的定义是等同的：

VB.NET

```
Dim FavoriteGenres As New List(Of Integer)
```

C#

```
List<int> FavoriteGenres = new List<int>();
```

在下面的“试一试”练习中，将看到如何使用这个和其他配置文件属性。首先，在下一个“试一试”练习中将学习如何在 web.config 中配置 Profile。之后的练习将学习如何使用这些属性，以及如何使用 List 类的各种方法。

试一试

创建一个配置文件

在这个“试一试”练习中，将学习如何创建一个配置文件，它可存储用户的姓和名、生日、个人简介以及用户最喜爱的流派的 ID 列表。这个列表后面将用于只显示用户感兴趣的评论。

- (1) 从站点根文件夹中打开 web.config 文件并定位到<system.web>起始标记。
- (2) 添加一个新的<profile>元素作为<system.web>的直接子元素。
- (3) 完整的<profile>元素如下所示：

```
<system.web>
  <profile>
    <properties>
      <add name="FirstName" />
      <add name="LastName" />
      <add name="DateOfBirth" type="System.DateTime" />
      <add name="Bio" />
      <add name="FavoriteGenres"
            type="System.Collections.Generic.List`1[System.Int32]" />
    </properties>
  </profile>
</system.web>
```

- (4) 按下 Ctrl+S 组合键保存该 web.config 文件。一旦保存好该文件，就会启动后台进程生成一

个用于该配置文件的类文件。在这个类文件成功创建并编译后，可以通过 Page 类的 Profile 属性以编程方式访问该 Profile。

(5) 要测试该配置文件，可以打开第 16 章中在 Design 视图中创建的 MyProfile.aspx。双击该页面，为 Load 事件建立事件处理程序并添加下列代码(包含您自己的姓和名)。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Profile.FirstName = "Your first name here"
    Profile.LastName = "Your last name here"
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    Profile.FirstName = "Your first name here";
    Profile.LastName = "Your last name here";
}
```

一旦在 Profile 后面输入点(.)，就会出现 IntelliSense 列表，显示可用的 Profile 属性，如图 17-1 所示。



图 17-1

(6) 完成代码输入后，保存并关闭该文件。

常见错误：如果在输入句点(.)时没有任何显示，可从主菜单中选择 Build | Build Web Site 选项或按 Ctrl+Shift+B 键。这样就可以强制 VWD 重新编译应用程序，包括 Profile 属性的特定类。在第 19 章将介绍更多有关编译的内容。稍过几秒，在 Page 类的 Profile 属性的 IntelliSense 列表中就会出现相应属性。如果仍没显示，则检查 Error List(从主菜单中选择 View | Error List 命令打开 Error List)确定 web.config 文件中无错误，并且确保使用的是 Web Site Project 而不是 Web Application Project。

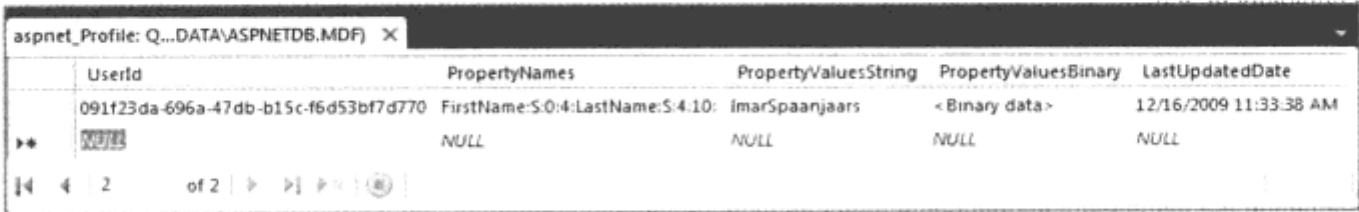
(7) 切换至 web.config 文件并滚动至最后。创建一个<location>元素的副本，用于阻止未授权用户访问 Management 文件夹，并将它粘贴到已有元素之下。然后修改该副本，它将阻止所有未经身份验证的用户访问站点根文件夹下的 MyProfile.aspx 文件。最后的设置如下所示：

```
</location>
<location path="MyProfile.aspx">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
</configuration>
```

(8) 在 Solution Explorer 中，右击文件 MyProfile.aspx 并选择 View in Browser 命令。只有登录后才可以查看该文件；如果还未登录，将被带到 Login.aspx 页面。用在前面章节中创建的用户名和密码

码登录并单击 Login 按钮，将被带至 MyProfile.aspx 页面。尽管在该页面中没有任何新内容，但 Page_Load 中的代码已经运行并在数据库中为您创建了一个配置文件。

(9) 要查看这个配置文件，可关闭浏览器并回到 Visual Web Developer。打开 Database Explorer 窗口(VS 付费版中的 Server Explorer)并展开 ASPNETDB.MDF 数据库的 Tables 元素。定位到 aspnet_Profile 表并右击它，选择 Show Table Data。结果如图 17-2 所示。



Userid	PropertyNames	PropertyValuesString	PropertyValuesBinary	LastUpdatedDate
091f23da-696a-47db-b15c-f6d53bf7d770	FirstName:5:0:4:LastName:5:4:10:	ImarSpaanjaars	< Binary data >	12/16/2009 11:33:38 AM
NULL	NULL	NULL	NULL	NULL

图 17-2

该图显示了其中一个用户的配置文件数据。在第(5)步输入的姓和名存储在 PropertyValuesString 列。由于这一数据是以特殊格式存储的，所以不能手动修改该数据，而应该使用 Profile 对象改变底层数据。

工作原理

当在 web.config 中定义 Profile 的属性时，ASP.NET 运行库在后台创建了一个类。这个名为 ProfileCommon 的类使得可访问强类型的属性，如 FirstName、LastName 和 FavoriteGenres。然后 Page 通过其 Profile 属性访问该类。ProfileCommon 继承自 ProfileBase，后者是一个在 .NET Framework 中定义的基类，包含了通过与已配置的提供者(ASP.NET Profile 提供者)通信来访问数据库中的配置文件的行为。然后该提供者负责将数据持久化到已配置的数据库中的所有复杂工作。和前面章节中看到的 Membership 和 Role 提供者一样，Profile 提供者默认使用在 LocalSqlServer 连接字符串中定义的数据库。附录 B 介绍了如何改变这一行为，使得提供者可使用一个不同的数据库。

要定义属性，可使用带有一个 name 特性和一个可选 type(如果属性是另一类型而不是 System.String) 的 <add> 元素。例如：

```
<add name="FavoriteGenres"
      type="System.Collections.Generic.List`1[System.Int32]" />
```

该属性建立了一个可存储 Integer 值的列表，用来保存用户最喜爱的音乐流派。在后面的“试一试”练习中将介绍如何使用这一属性。

在 web.config 文件中建立好该 Profile 并且后台类被编译后，就可以在代码中访问它。例如，可通过代码设置 FirstName 这样的属性：

```
VB.NET
Profile.FirstName = "Your first name here"

C#
Profile.FirstName = "Your first name here";
```

尽管在前面的练习中未采用过，但可以用相同的方法访问组中的属性。所需做的只是为属性名加一个组名和一个点作为前缀。假定有一个 PostalAddress 的示例，可以按如下所示存储街道的地址：

VB.NET

```
Profile.PostalAddress.Street = "Some Street"
```

C#

```
Profile.PostalAddress.Street = "Some Street";
```

在 EndRequest(在 ASP.NET 页面生命周期中较晚触发的事件)中，对 Profile 的更改会自动保存。这样，可以在生命周期的多个阶段更改 Profile，而不必为手动保存它而操心。

在图 17-2 中，可看到如何用单条记录存储整个配置文件。第一列包含了配置文件所属用户的唯一 ID。第二列包含为当前用户存储的属性名列表，并带有值的起始索引及长度。例如，看一下姓的表示：

```
LastName:S:4:10
```

这表明存储在 PropertyValuesString 列的 LastName 属性的值从位置 4(第 5 个字符，因为使用的是基于 0 的位置)开始并且长度为 10 个字符。LastName 和起始索引之间的字母定义了可从中找到数据的列：S 表示 PropertyValuesString 列，而 B 表示 PropertyValuesBinary 列，它用于以二进制格式存储复杂对象。这种紧凑的格式使得 Profile 提供者可以在单列中存储多种不同的属性，这样在配置文件改变时无需更改数据库模式。

在下一节中，将学习读取和写入 Profile 的更多内容。

17.1.2 使用 Profile

正如在前面一节所看到的，可以很容易地写入 Profile。要改变 FirstName 这样的属性，只需一行代码。Profile 会跟踪您所作的更改，如有必要，在 EndRequest 时会自动保存它。读取 Profile 信息也很简单，只需访问其某个属性。下列代码段显示了如何用配置文件中的名来填充 TextBox。

VB.NET

```
FirstName.Text = Profile.FirstName
```

C#

```
FirstName.Text = Profile.FirstName;
```

检索组中的属性的方法也几乎是相同的。要访问前一示例中讨论的 Street 属性，需要使用下列代码：

VB.NET

```
PostalAddressStreet.Text = Profile.PostalAddress.Street
```

C#

```
PostalAddressStreet.Text = Profile.PostalAddress.Street;
```

访问 FavoriteGenres 属性会稍有不同。由于该属性是个集合，所以不能直接访问它。而是要使用其方法和属性使数据进出。下列示例首先清除了整个列表，然后添加了两个流派的 ID。

VB.NET

```
Profile.FavoriteGenres.Clear()
```



```
Profile.FavoriteGenres.Add(7)
Profile.FavoriteGenres.Add(11)
```

C#

```
Profile.FavoriteGenres.Clear();
Profile.FavoriteGenres.Add(7);
Profile.FavoriteGenres.Add(11);
```

下面的“试一试”练习显示了如何在用户配置文件中存储基本数据。在后一“试一试”练习中，将看到使用 FavoriteGenres 列表的一个真实实现。

试一试 在 Profile 中存储基本用户数据

在这个“试一试”练习中，将修改 Profile 页面，使用户能保存其姓和名、生日以及个人简介。

- (1) 再次打开 MyProfile.aspx，切换至 Code Behind 模式。删除 Page_Load 事件处理程序中设置用户姓和名的两行代码。
- (2) 切换至 Design 视图，在 ChangePassword 控件的上方，通过选择 Table | Insert Table 命令添加一个 5 行 3 列的 HTML 表。
- (3) 在前 4 行的第 2 列中，拖放 4 个 TextBox 控件并重命名。从第 1 行到第 4 行，通过设置 ID 特性，分别命名为 FirstName、LastName、DateOfBirth 和 Bio。图 17-3 显示了 TextBox 控件应放置的位置。

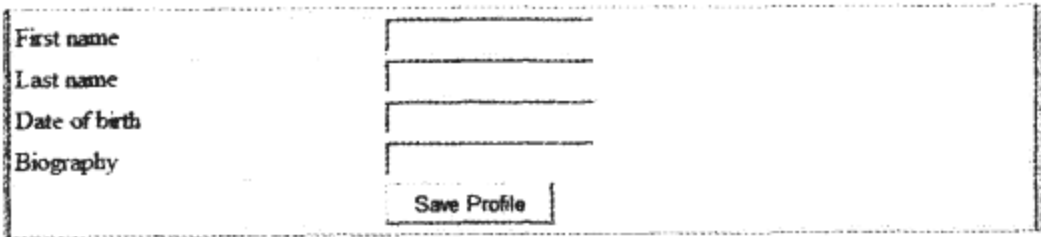


图 17-3

- (4) 在开始 4 行的第 1 列中，拖放 4 个 Label 控件并设置它们的属性如表 17-2 所示，这样每个标签都与相同行的 TextBox 相关联。

表 17-2

ID	Text	AssociatedControlID
FirstNameLabel	First name	FirstName
LastNameLabel	Last name	LastName
DateOfBirthLabel	Date of birth	DateOfBirth
BioLabel	Biography	Bio

- (5) 在第 5 行的第 2 个单元格中，拖放一个 Button 控件并设置其 ID 为 SaveButton，Text 为 Save Profile。Design 视图如图 17-3 所示。
- (6) 在前 3 行的最后一列，拖放 3 个 RequiredFieldValidator 控件。按表 17-3 所示设置它们的属性，这样每个验证器都与同行的 TextBox 联系在一起。记住：通过在选择控件时按下 Ctrl 键，可以同时设置所有控件的 Display 属性。

表 17-3

ControlToValidate	Display	Error Message
FirstName	Dynamic	First name is required.
LastName	Dynamic	Last name is required.
DateOfBirth	Dynamic	Date of birth is required.

(7) 在 DateOfBirth 框的验证器的旁边，拖放一个 CompareValidator 控件，按表 17-4 所示设置其属性。

表 17-4

属 性	值
Display	Dynamic
ErrorMessage	Please enter a valid date.
ControlToValidate	DateOfBirth
Operator	DataTypeCheck
Type	Date

(8) 设置 Bio 控件的 TextMode 为 MultiLine，并设置其 Width 和 Height 属性分别为 300px 和 75px。

(9) 修改表上方的文本，表明用户现在不只是可以改变他们的密码。这时的 Design 视图如图 17-4 所示。

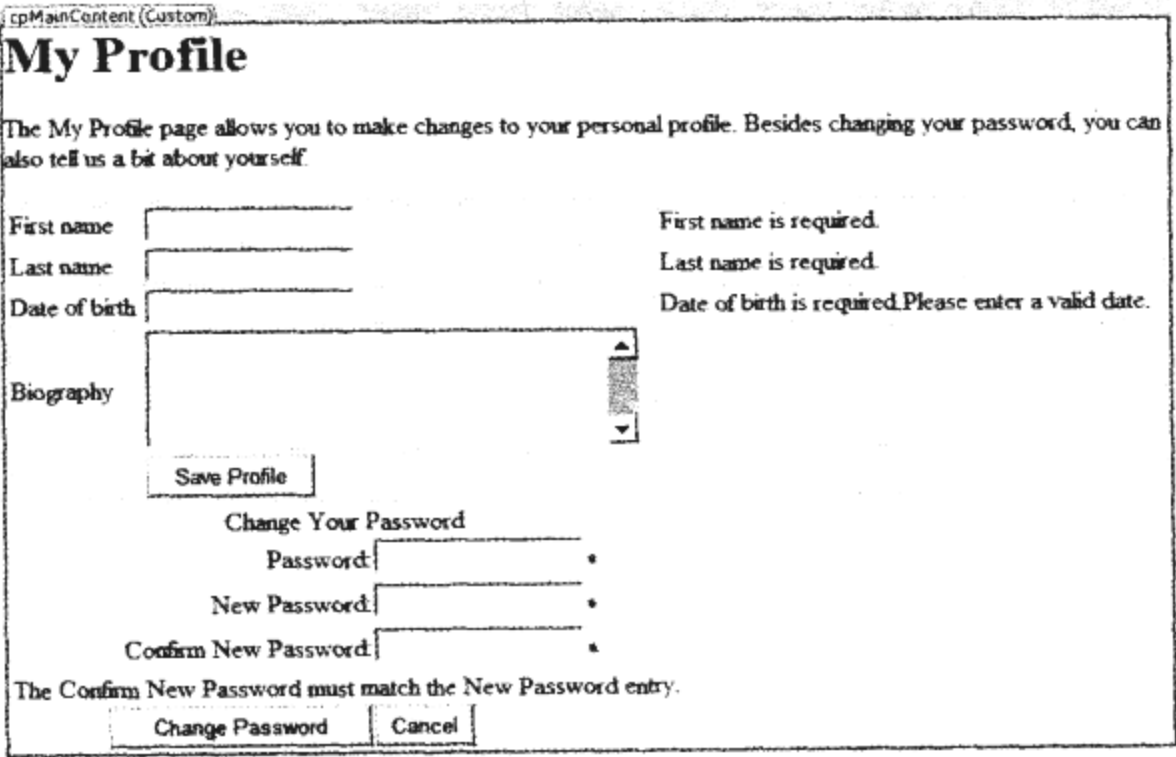


图 17-4

(10) 双击 Save Profile 按钮并在 VWD 添加的 Click 事件处理程序中，编写下面突出显示的代码：

```
VB.NET
Protected Sub SaveButton_Click(ByVal sender As Object,
```

```
        ByVal e As System.EventArgs) Handles SaveButton.Click
    If Page.IsValid Then
        Profile.FirstName = FirstName.Text
        Profile.LastName = LastName.Text
        Profile.DateOfBirth = DateTime.Parse(DateOfBirth.Text)
        Profile.Bio = Bio.Text
    End If
End Sub
```

```
C#
protected void SaveButton_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        Profile.FirstName = FirstName.Text;
        Profile.LastName = LastName.Text;
        Profile.DateOfBirth = DateTime.Parse(DateOfBirth.Text);
        Profile.Bio = Bio.Text;
    }
}
```

(11) 在同一页面的 Page_Load 事件处理程序中，添加下列代码，这样在页面加载时将使用配置文件中的数据填充文本框控件。

```
VB.NET
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    If Not Page.IsPostBack Then
        FirstName.Text = Profile.FirstName
        LastName.Text = Profile.LastName
        DateOfBirth.Text = Profile.DateOfBirth.ToShortDateString()
        Bio.Text = Profile.Bio
    End If
End Sub
```

```
C#
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        FirstName.Text = Profile.FirstName;
        LastName.Text = Profile.LastName;
        DateOfBirth.Text = Profile.DateOfBirth.ToShortDateString();
        Bio.Text = Profile.Bio;
    }
}
```

(12) 保存所有更改并在浏览器中请求该页面。如果要求必须先登录，则输入您的详细信息并单击 Login 按钮。然后就可看到 My Profile 页面重新出现，其中您在前一个“试一试”练习中输入的姓和名的数据已经得到填充。将该数据改成您的详细信息并单击 Save Profile 按钮。

(13) 关闭浏览器，然后再次请求 MyProfile.aspx。可注意到，更改在两个浏览器会话间持久保存。

工作原理

本“试一试”练习中的大部分内容都是我们所熟悉的。页面包含了多个 TextBox 控件，它们使用 RequiredFieldValidator 和 CompareValidator 控件进行验证。另外，Label 控件使用 AssociatedControlID 属性与它们各自的 TextBox 控件相关联。这样就很容易将焦点放置在浏览器中的控件上，因为单击某个 Label 会在相关的 TextBox 中放置光标。利用在第 15 章讨论的技巧，通过设置 AccessKey 属性，可以使控件变得更容易访问。

当单击 Save Profile 按钮时，就会从 4 个 TextBox 控件中检索值并存储到配置文件中。在页面首次加载时，就会发生与之相反的情况：控件使用 Profile 中的值进行预填充。为了避免重写用户已输入的数据，可通过代码在页面首次加载时而非回发时从配置文件中获取数据。

VB.NET

```
If Not Page.IsPostBack Then
    FirstName.Text = Profile.FirstName
    ....
End If
```

C#

```
if (!Page.IsPostBack)
{
    FirstName.Text = Profile.FirstName;
    ...
}
```

尽管这个示例本身很简单，但它为使用整数的 List 存储用户对特定音乐流派的偏好这样的高级场景奠定了良好的基础。然后，可以使用这个最喜爱的流派的列表限制只显示用户真正感兴趣的评论列表。在接下来的“试一试”练习中，将看到如何在 Profile 中存储用户的偏好；后一个“试一试”练习将介绍如何再次使用本练习中保存的数据。

试一试 在 Profile 中存储流派偏好

在这个“试一试”练习中，将学习如何填充用户配置文件的 FavoriteGenres 属性。为了让用户选择他们最喜爱的流派，将显示一个 CheckBoxList，它与一个检索可用流派的 EntityDataSource 控件相关联。当用户保存数据时，他所选择的项就保存在 Profile 中。

(1) 在 MyProfile.aspx 页面中，在带有 Save Profile 按钮的行上方添加一个表行。方法是在 Design 视图中右击带有该按钮的行的空白部分，然后从上下文菜单中选择 Insert | Row Above 命令。或者，可以单击该单元格，将光标放到其中，然后按下 Ctrl+Alt+上方向键组合键。

(2) 在新行的第 1 个单元格内，拖放一个 Label 控件并设置其 Text 为 Favorite genres。

(3) 在第 2 个单元格中，从 Toolbox 的 Standard 类别中拖放一个 CheckBoxList 控件，并设置其 ID 为 PreferenceList。

(4) 单击 CheckBoxList 控件的 Smart Tasks 面板中的 Choose Data Source，将该控件与一个新的 EntityDataSource 关联。从数据源下拉列表中选择<New data source>，然后选择 Entity 数据源类型，并单击 OK 按钮。在新数据源控件的 Configure Data Source 向导中，选择 PlanetWroxEntities 作为 Named Connection，单击 Next 按钮，然后选择 Genres 作为 EntitySetName。在 Select 列表中，只选

择 Id 和 Name 属性的项。这时 Configure Data Source 对话框如图 17-5 所示。

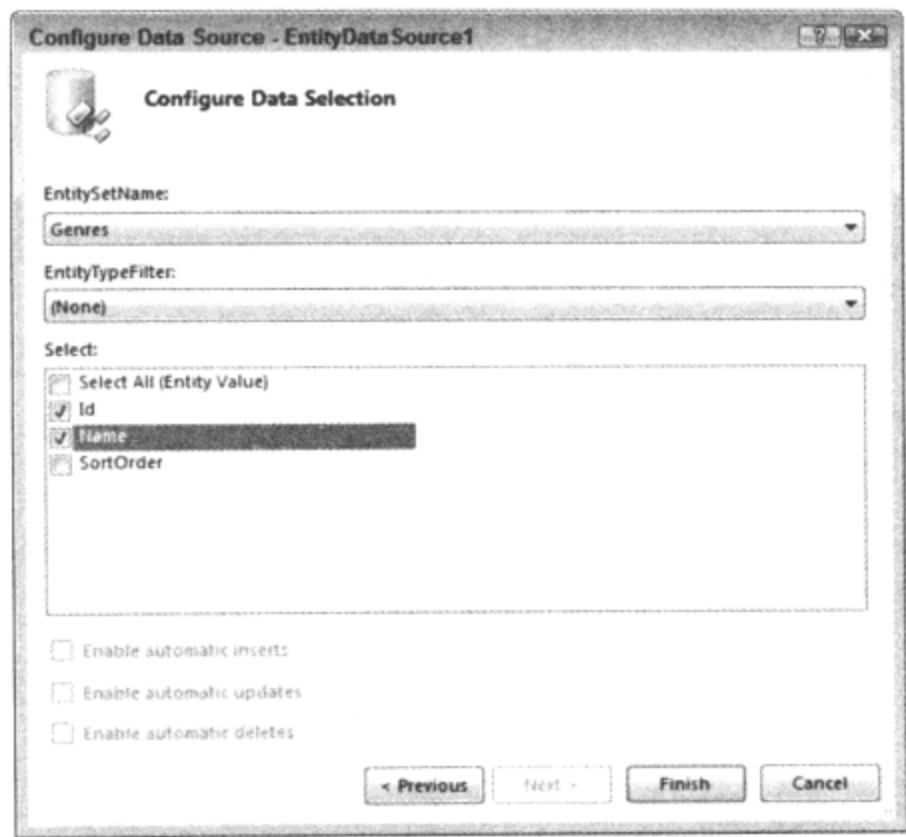


图 17-5

(5) 单击 Finish 按钮关闭该对话框。回到 CheckBoxList 控件的 Data Source Configuration Wizard，选择 Name 作为要显示的数据字段并使 Id 作为值的数据字段。如果下拉列表中没有显示这些项，那么可以首先单击对话框底部的蓝色 Refresh Schema 链接。此时的 Data Source Configuration Wizard 如图 17-6 所示。

单击 OK 按钮关闭对话框。

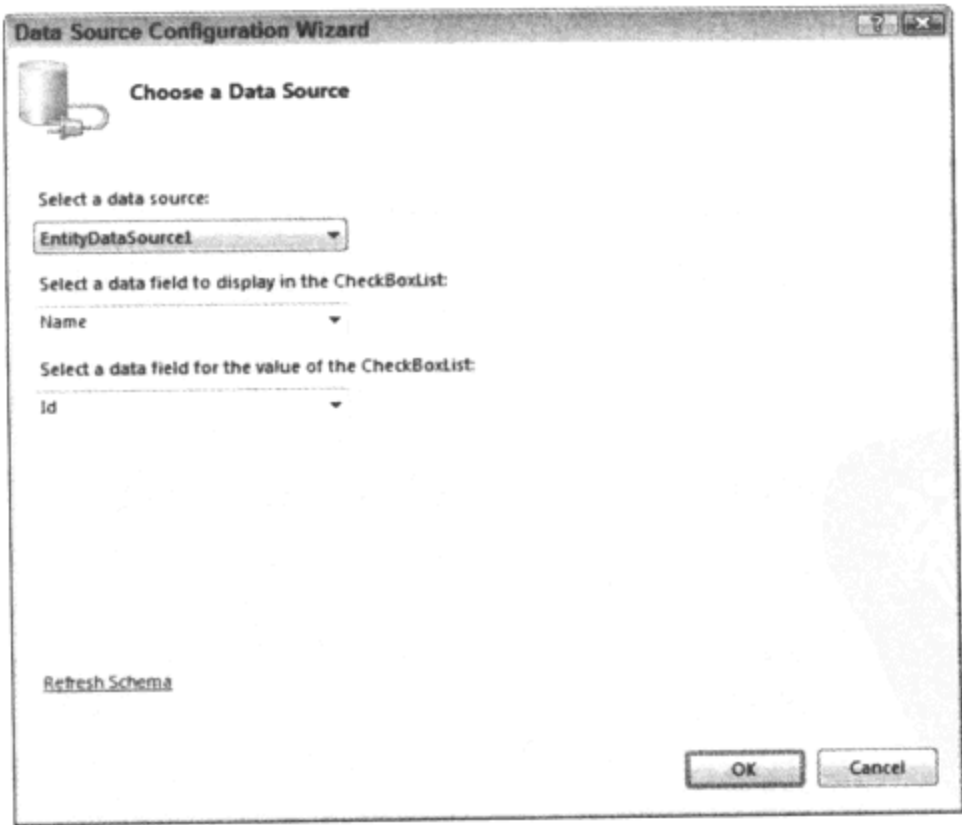


图 17-6

(6) 在 Design 视图中，单击刚才添加的 EntityDataSource 控件，然后按下 F4 键打开其 Properties 面板。定位到 OrderBy 属性，并输入 it.Name 来按字母顺序排序流派列表。切换到 Markup 视图，确认 EntityDataSource 控件和 CheckBoxList 的代码如下所示：

```
<asp:CheckBoxList ID="PreferenceList" runat="server"
    DataSourceID="EntityDataSource1" DataTextField="Name" DataValueField="Id">
</asp:CheckBoxList>
<asp:EntityDataSource ID="EntityDataSource1" runat="server"
    ConnectionString="name=PlanetWroxEntities"
    DefaultContainerName="PlanetWroxEntities" EnableFlattening="False"
    EntitySetName="Genres" OrderBy="it.Name" Select="it.[Id], it.[Name]">
</asp:EntityDataSource>
```

(7) 在 Design 视图中，单击 CheckBoxList 控件一次，打开其 Properties 面板并切换至 Events 选项卡。双击 DataBound 事件，然后在 Code Behind 中添加下列代码，根据用户的 Profile 设置预先选择列表中的项。

VB.NET

```
Protected Sub PreferenceList_DataBound(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles PreferenceList.DataBound
    For Each myItem As ListItem In PreferenceList.Items
        Dim currentValue As Integer = Convert.ToInt32(myItem.Value)
        If Profile.FavoriteGenres.Contains(currentValue) Then
            myItem.Selected = True
        End If
    Next
End Sub
```

C#

```
protected void PreferenceList_DataBound(object sender, EventArgs e)
{
    foreach (ListItem myItem in PreferenceList.Items)
    {
        int currentValue = Convert.ToInt32(myItem.Value);
        if (Profile.FavoriteGenres.Contains(currentValue))
        {
            myItem.Selected = true;
        }
    }
}
```

(8) 用下列代码扩展 SaveButton_Click 事件处理程序，使它也可以保存用户偏好的流派。

VB.NET

```
Profile.Bio = Bio.Text
' Clear the existing list
Profile.FavoriteGenres.Clear()

' Now add the selected genres
For Each myItem As ListItem In PreferenceList.Items
    If myItem.Selected Then
        Profile.FavoriteGenres.Add(Convert.ToInt32(myItem.Value))
    End If
End For
```



```
End If
Next

C#
Profile.Bio = Bio.Text;
// Clear the existing list
Profile.FavoriteGenres.Clear();

// Now add the selected genres
foreach (ListItem myItem in PreferenceList.Items)
{
    if (myItem.Selected)
    {
        Profile.FavoriteGenres.Add(Convert.ToInt32(myItem.Value));
    }
}
```

(9) 保存所有更改，然后在浏览器中请求该 Profile 页面并在指示下登录。您将在浏览器中看到流派列表，每个流派前都有一个复选框。选择几个您最喜爱的流派并单击 Save Profile 按钮。浏览至另一个页面，然后从主 Menu 或 TreeView 中再次选择 My Profile。刚才选择的流派在该页面中仍是选中状态，如图 17-7 所示。

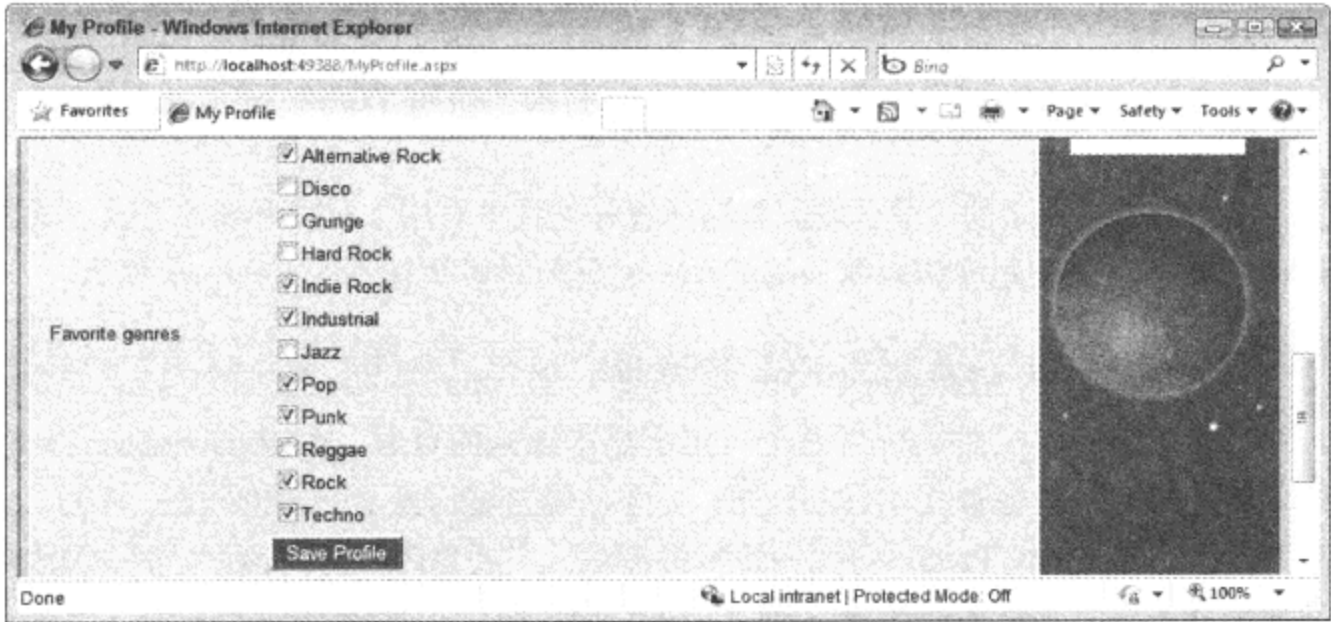


图 17-7

工作原理

首先，在 Profile 中定义了 FavoriteGenres 属性作为可存储整型值的泛型列表。由于该属性是 List，所以不能直接给它赋值；而是使用其方法(如 Add 和 Clear)来添加和删除项。由于每个流派 ID 只能在列表中存储一次，所以在用户单击 Save Profile 按钮来删除之前作出的选择时，该列表就在选择的项被再次添加之前清空了。

```
VB.NET
Profile.FavoriteGenres.Clear()

C#
Profile.FavoriteGenres.Clear();
```

然后当列表变空时，添加选择的流派的 ID。

VB.NET

```
For Each myItem As ListItem In PreferenceList.Items
    If myItem.Selected Then
        Profile.FavoriteGenres.Add(Convert.ToInt32(myItem.Value))
    End If
Next
```

C#

```
foreach (ListItem myItem in PreferenceList.Items)
{
    if (myItem.Selected)
    {
        Profile.FavoriteGenres.Add(Convert.ToInt32(myItem.Value));
    }
}
```

这段代码迭代 CheckBoxList 中的所有项。Selected 属性可以确定用户是否在 Profile 页面中选择了项。如果选择了，流派的值就会被检索并转换为 Integer(C#中为 int)，然后通过 Add 方法添加到 FavoriteGenres 列表中。

这些就是将像最喜爱的流派的列表这样的复杂数据存储到用户的 Profile 中所需做的所有工作。您所需做的只是添加一堆数字到列表中。然后.NET 运行库负责将 Profile 持久化到数据库中，从而使其在后续页面中可用。

当然，在实际用于站点之前，最喜爱流派的列表并不真正有用。在接下来的“试一试”练习中，将介绍如何使用该列表限制用户访问默认的 Reviews 页面时最初看到的 Reviews 列表。

试一试

在 Reviews 页面中使用 Profile

目前，站点的 Reviews 文件夹中有两个可用于显示评论的页面：AllByGenre.aspx 和 All.aspx。在这个“试一试”练习中，将修改 Default.aspx 页面，使它显示另一个评论列表。不过，这次的评论列表限制为属于用户在 My Profile 页面中选择的流派。当匿名用户访问该页面时，则会得到一个消息，表明他们还未设置最喜爱的流派。

- (1) 在 Markup 视图中打开 Reviews 文件夹中的 Default.aspx。
- (2) 在控件的 cpMainContent 占位符中，添加下列代码，创建一个嵌套的 Repeater，其中使用每个所选的流派作为标题，后跟属于该流派的评论的一个列表。

```
<asp:Repeater ID="GenreRepeater" runat="server">
    <HeaderTemplate>
        <p>Below you find a list with reviews for your favorite music genres.</p>
    </HeaderTemplate>
    <ItemTemplate>
        <h3><asp:Literal ID="Literal1" runat="server"
            Text='<%# Eval("Name") %>'></asp:Literal></h3>
        <asp:Repeater ID="ReviewRepeater" runat="server"
            DataSource='<%# Eval("Reviews") %>'>
            <ItemTemplate>
                <asp:HyperLink ID="HyperLink1" runat="server"
```

```
Text='<# Eval("Title") %>'
NavigateUrl='<# "ViewDetails.aspx?ReviewId=" +
Eval("Id").ToString() %>'>
</asp:HyperLink><br />
</ItemTemplate>
</asp:Repeater>
</ItemTemplate>
</asp:Repeater>
<asp:PlaceHolder ID="NoRecords" runat="Server" Visible="False">
  <p>Sorry, no reviews were found. You either didn't set your favorite genres
    or you may need to log in first. </p>
</asp:PlaceHolder>
<p>You can change your genre preferences <a href="~/MyProfile.aspx"
  runat="server">here</a>.</p>
```

可以通过编写必要的代码手动创建 Repeater 控件，或是从 Toolbox 的 Data 类别拖动它们。内部的 Repeater 包含一个 HyperLink 控件，指向第 15 章创建的 ViewDetails.aspx 页面。

(3) 在 Design 视图中双击该页面建立 Load 事件处理程序。在页面顶部为 PlanetWroxModel 名称空间创建一个 Imports/using 语句，并向 VWD 创建的处理程序添加下面的代码。

VB.NET

```
Imports PlanetWroxModel
...
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Using myEntities As PlanetWroxEntities = New PlanetWroxEntities()
        If Profile.FavoriteGenres.Count > 0 Then
            Dim favGenres = From genre In myEntities.Genres
                            Order By genre.Name
                            Where Profile.FavoriteGenres.Contains(genre.Id)
                            Select New With {genre.Name, genre.Reviews}

            GenreRepeater.DataSource = favGenres
            GenreRepeater.DataBind()
        End If

        GenreRepeater.Visible = GenreRepeater.Items.Count > 0
        NoRecords.Visible = GenreRepeater.Items.Count = 0
    End Using
End Sub
```

C#

```
using PlanetWroxModel;
...
protected void Page_Load(object sender, EventArgs e)
{
    using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
    {
        if (Profile.FavoriteGenres.Count > 0)
        {
            var favGenres = from genre in myEntities.Genres
                            orderby genre.Name
                            where Profile.FavoriteGenres.Contains(genre.Id)
```



```
        select new { genre.Name, genre.Reviews };
        GenreRepeater.DataSource = favGenres;
        GenreRepeater.DataBind();
    }
    GenreRepeater.Visible = GenreRepeater.Items.Count > 0;
    NoRecords.Visible = GenreRepeater.Items.Count == 0;
}
}
```

(4) 保存所有更改，然后在浏览器中请求该页面。如果前面在 Profile 页面中选择一个或多个流派，且这些流派也有一些可用的评论，将看到如图 17-8 所示的列表。

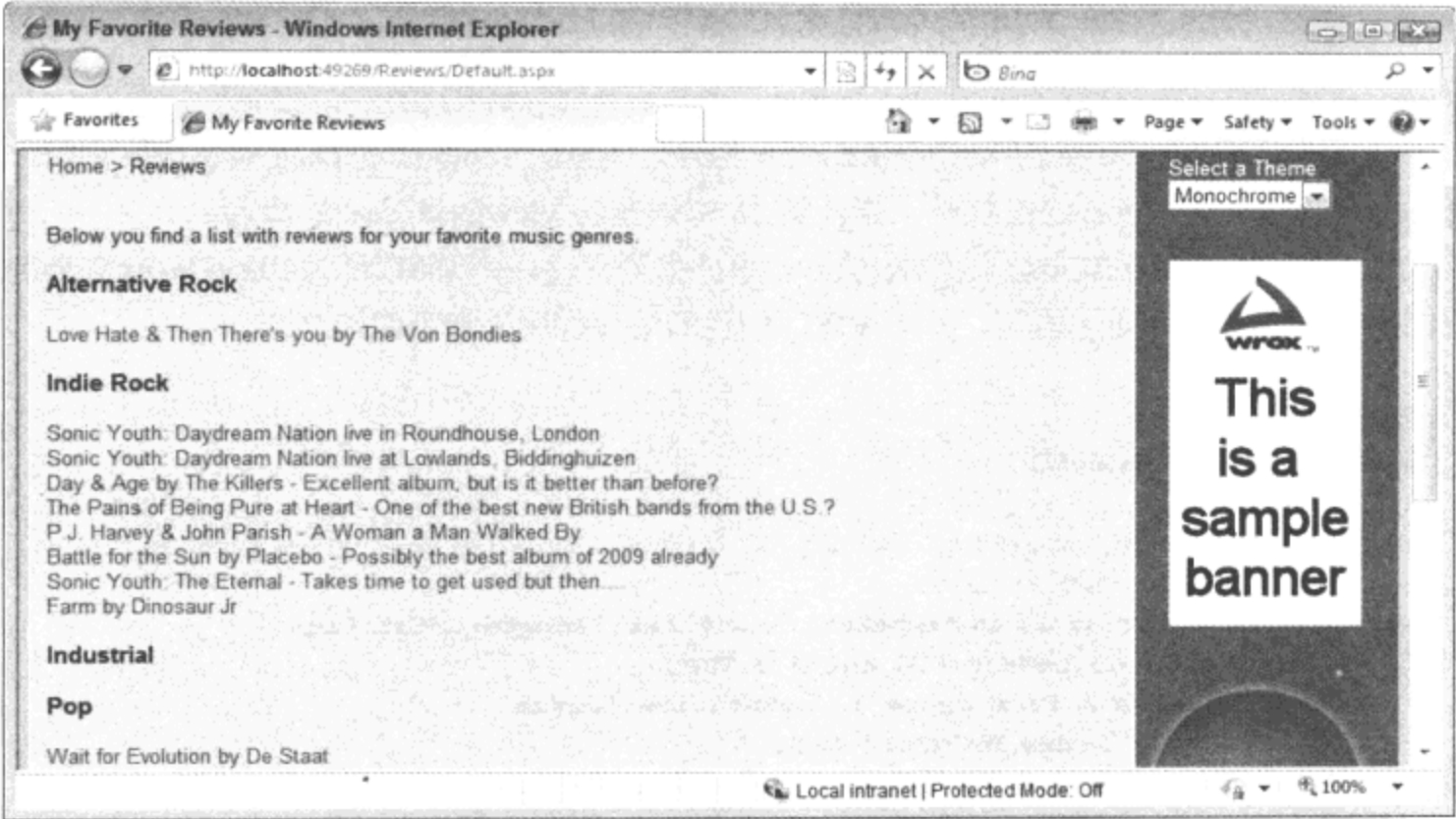


图 17-8

如果没有设置喜爱的流派或是没有登录，就会得到如图 17-9 所示的消息。

单击此消息中的链接，将被带至 My Profile 页面，从中可设置或改变最喜爱的流派。未经授权的用户将被要求登录或注册，然后才能访问 Profile 页面。

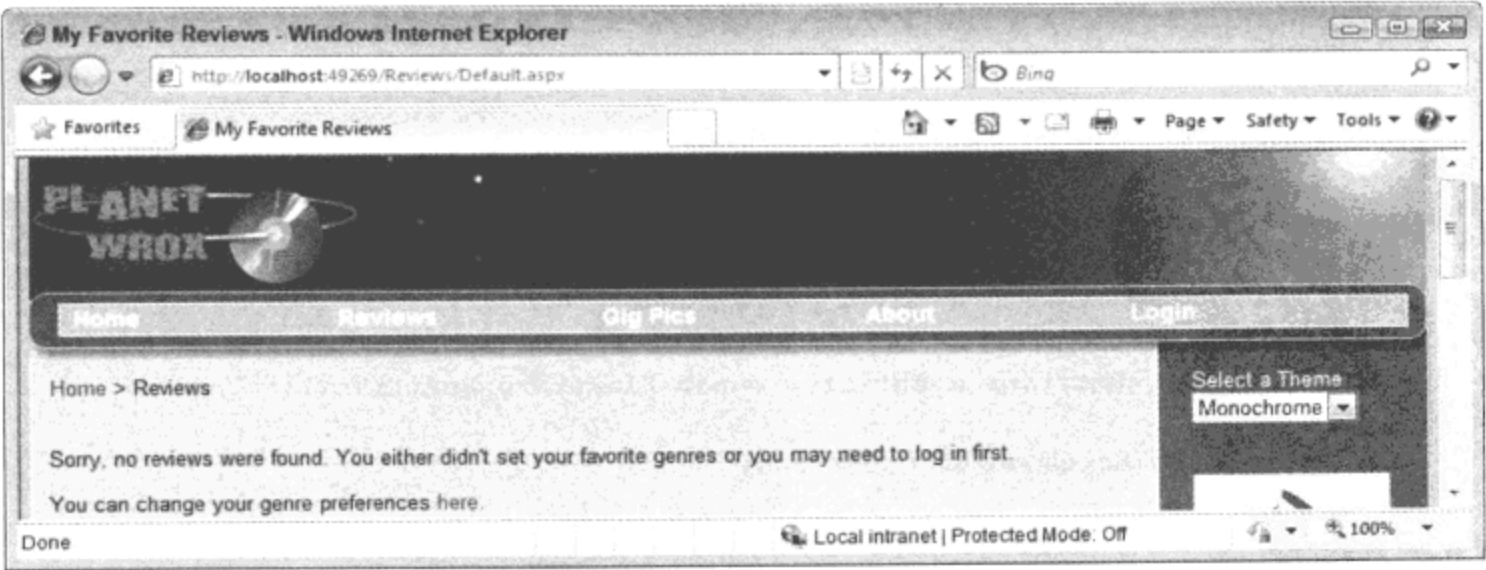


图 17-9

工作原理

Code Behind 文件中的代码执行一个 LINQ to EF 查询，它检索属于用户最喜爱的流派的所有评论。对于匿名用户来说，最喜爱的流派的列表将是空的，因而总是会看到有关在 Profile 页面设置偏好的消息。为了避免不必要地调用数据库，只有当用户选中了 FavoriteGenres 列表中的 Count 属性并选择了至少一个喜爱的流派时，查询才会执行。

添加到 Reviews 文件夹中的 Default.aspx 页面的嵌套 Repeater 的代码类似于 AllByGenre.aspx 页面(有一个包含了 BulletedList 控件的 Repeater)的代码。和那个页面一样，嵌套的 Repeater 使用 DataSource 特性从外部的 Repeater 中获取数据。

```
<asp:Repeater ID="ReviewRepeater" runat="server"
    DataSource='

```
<asp:Repeater ID="ReviewRepeater" runat="server"
 DataSource='<%# Eval("Reviews")%>'>
 ...
</asp:Repeater>
```



然后，嵌套的 Repeater 使用 Reviews 列表建立可将您带至详细信息页面的链接：



```
<asp:HyperLink ID="HyperLink1" runat="server" Text='<%# Eval("Title") %>'
 NavigateUrl='<%# "ViewDetails.aspx?ReviewId=" + Eval("Id").ToString() %>'>
</asp:HyperLink>

```



HyperLink 控件从绑定的 Review 实例中获取其 Text，并使用其 Id 构建 NavigateUrl。在 Eval("Id") 上使用 ToString 方法，将值在连接到包含 URL 的字符串之前转换为字符串。这样就避免了 Visual Basic 中的类型转换，其中 Eval("Id")的结果通常是一个不能直接与字符串相连接的数字。



为了了解这些控件如何获取数据，需要查看使用了针对 Entity Framework 的 LINQ 查询的 Code Behind 文件。



VB.NET



```
Dim favGenres = From genre In myEntities.Genres
 Order By genre.Name
 Where Profile.FavoriteGenres.Contains(genre.Id)
 Select New With {genre.Name, genre.Reviews}
```



C#



```
var favGenres = from genre in myEntities.Genres
 orderby genre.Name
 where Profile.FavoriteGenres.Contains(genre.Id)
 select new { genre.Name, genre.Reviews };
```



除了突出显示的代码行和变量名，这个 LINQ 查询与在 AllByGenre.aspx 中使用的查询完全相同。这一示例的特别之处是其 where 子句，它限制了结果用户真正感兴趣的评论。注意这里是如何使用 List 泛型类的 Contains 方法的。尽管初看起来，所有的流派和评论都是从数据库检索到 ASPX 页面，然后再与 Profile 属性 FavoriteGenres 中的值作比较，但是实际上情况正好相反。首先，Entity Framework 引擎可以智能地从 FavoriteGenres 属性中收集所有 ID，然后将它们包括到 SQL 语句中并发送到数据库以获取请求的流派和评论。这意味着请求的流派的筛选是在数据库级别发生的，而不



563



350元等于什么？答案就是等于Microsoft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！ 视频包括浪曦 本杰 中美IT 等290 GB视频教程 需要的请联系QQ：2569343569


```


是在 ASPX 页面中。而这又意味着只有少量的记录从数据库传送到 ASPX 页面(只有那些真正需要的)，从而获得了更好的性能。

对于匿名用户，Profile 属性 FavoriteGenres 返回一个空列表，而不是抛出异常。因此，即使是无 Profile 的用户也可安全地浏览此页面。但是他们看不到任何评论，而是会得到一条消息，表明他们还没有设置流派偏好或是需要首先登录。

在 Page_Load 事件处理程序的最后，有一些代码决定了是否显示或隐藏 Repeater 和 NoRecords 控件。

VB.NET

```
GenreRepeater.Visible = GenreRepeater.Items.Count > 0
NoRecords.Visible = GenreRepeater.Items.Count = 0
```

C#

```
GenreRepeater.Visible = GenreRepeater.Items.Count > 0;
NoRecords.Visible = GenreRepeater.Items.Count == 0;
```

如果在数据绑定到外部 Repeater 后，Items 集合仍为空，这就意味着没有为当前用户找到流派。如果那样的话，就隐藏整个 Repeater 并显示 Placeholder。不过，如果 Items 集合的 Count 属性大于 0，则显示 Repeater 而隐藏 Placeholder。

在第 14 章中，创建了一个名为 NewPhotoAlbum.aspx 的页面，它允许用户插入新的 Gig Pics 相册。这一页面的当前实现有一些缺点。首先，任何人都可以插入新相册。没有方法可以阻止匿名用户创建新相册和上传图片。

其次，只有管理员可以从现有相册中删除图片。如果相册所有者也可以删除自己的图片，站点功能就更加完善了。现在您已经了解了安全性和个性化 Web 页面的更多内容，因此这种功能就很容易实现了，如下面的“试一试”练习所示。

试一试 让用户管理他们自己的相册

在这个“试一试”练习中，将介绍如何阻止未经身份验证的用户访问 NewPhotoAlbum.aspx 和 ManagePhotoAlbum.aspx 页面。另外，还将介绍如何记录创建相册的用户的名字，并允许用户之后根据名字找到并修改自己的相册。

(1) 打开 Database Explorer，展开 PlanetWrox 数据库，然后定位到 PhotoAlbum 表。右击它并选择 Open Table Definition。添加一个名为 UserName 的新列，设置其数据类型为 nvarchar(256)，并且选中 Allow Nulls 选项(由于该表中已经有一些没有有效的 UserName 的相册，因此在这一阶段不能强制该列，除非先从数据库中删除这些相册以及它们相关的图片)。保存更改并关闭表设计器。

(2) 从 App_Code 文件夹中打开 ADO.NET Entity Data Model 文件 PlanetWrox.edmx，右击设计器中的空白位置，选择 Update Model from Database 命令。等待 VWD 分析数据库，然后单击 Finish 按钮。数据库中的 UserName 列现在作为 PhotoAlbum 类的一个属性显示出来，如图 17-10 所示。

保存更改并关闭文件。

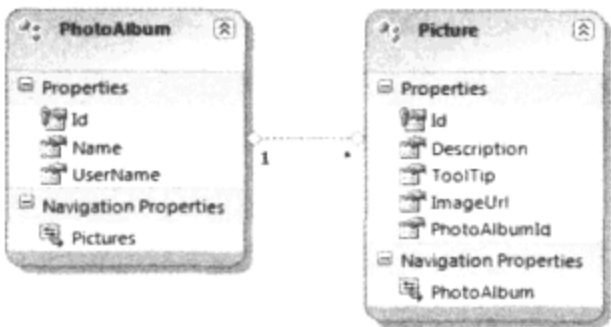


图 17-10

(3) 打开 web.config 文件并在已有的<location>元素下添加下列两个<location>元素，阻止匿名用户访问这两个引用的文件。

```
</location>
<location path="ManagePhotoAlbum.aspx">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
<location path="NewPhotoAlbum.aspx">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
</configuration>
```

保存更改，然后关闭 web.config 文件。

(4) 在 Design 视图中打开 NewPhotoAlbum.aspx 页面，定位到 EntityDataSource 控件，使用 Properties 面板中的 Events 选项卡为其 Inserting 事件建立事件处理程序。向 VWD 创建的事件处理程序添加下列代码：

VB.NET

```
Protected Sub EntityDataSource1_Inserting(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.EntityDataSourceChangingEventArgs) _
    Handles EntityDataSource1.Inserting
    Dim myPhotoAlbum As PhotoAlbum = CType(e.Entity, PhotoAlbum)
    myPhotoAlbum.UserName = User.Identity.Name
End Sub
```

C#

```
protected void EntityDataSource1_Inserting(object sender,
    EntityDataSourceChangingEventArgs e)
{
    PhotoAlbum myPhotoAlbum = (PhotoAlbum)e.Entity;
    myPhotoAlbum.UserName = User.Identity.Name;
}
```

(5) 从 PhotoAlbums 文件夹中，打开 Default.aspx，并切换到其 Code Behind。

(6) 在文件的顶部，为 PlanetWroxModel 名称空间添加一个 Imports/using 语句。然后，使用下面的代码扩展 DataBound 事件处理程序，如果当前用户是 Manager 或相册的创建者，则显示一个 Edit 链接。

VB.NET

```
Protected Sub ListView1_DataBound(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles ListView1.DataBound
    If Not String.IsNullOrEmpty(DropDownList1.SelectedValue) Then
        Dim photoAlbumId As Integer = Convert.ToInt32(DropDownList1.SelectedValue)
        Using myEntities As PlanetWroxEntities = New PlanetWroxEntities()
            Dim photoAlbumOwner As String = (From p In myEntities.PhotoAlbums
                Where p.Id = photoAlbumId
                Select p.UserName).Single()

            If User.Identity.IsAuthenticated And (User.Identity.Name = photoAlbumOwner _
                Or User.IsInRole("Managers")) Then
                EditLink.NavigateUrl = String.Format(
                    "~/ManagePhotoAlbum.aspx?PhotoAlbumId={0}", DropDownList1.SelectedValue)
                EditLink.Visible = True
            Else
                EditLink.Visible = False
            End If
        End Using
    Else
        EditLink.Visible = False
    End If
End Sub
```

C#

```
protected void ListView1_DataBound(object sender, EventArgs e)
{
    if (!string.IsNullOrEmpty(DropDownList1.SelectedValue))
    {
        int photoAlbumId = Convert.ToInt32(DropDownList1.SelectedValue);
        using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
        {
            string photoAlbumOwner = (from p in myEntities.PhotoAlbums
                where p.Id == photoAlbumId
                select p.UserName).Single();

            if (User.Identity.IsAuthenticated &&
                (User.Identity.Name == photoAlbumOwner || User.IsInRole("Managers")))
            {
                EditLink.NavigateUrl = string.Format(
                    "~/ManagePhotoAlbum.aspx?PhotoAlbumId={0}", DropDownList1.SelectedValue);
                EditLink.Visible = true;
            }
            else
            {
                EditLink.Visible = false;
            }
        }
    }
}
```

```
else
{
    EditLink.Visible = false;
}
}
```

(7) 打开根文件夹下的 ManagePhotoAlbum.aspx 的 Code Behind。向 Page_Load 处理程序添加下面的代码。如果还没有该处理程序，则在 Design 视图中双击页面，让 VWD 创建一个。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Dim photoAlbumId As Integer =
        Convert.ToInt32(Request.QueryString.Get("PhotoAlbumId"))

    Using myEntities As PlanetWroxEntities = New PlanetWroxEntities ()
        Dim photoAlbumOwner As String = (From p In myEntities.PhotoAlbums
            Where p.Id = photoAlbumId
            Select p.UserName).Single()

        If User.Identity.Name <> photoAlbumOwner And
            Not User.IsInRole("Managers") Then
            Response.Redirect("~/")
        End If
    End Using
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    int photoAlbumId = Convert.ToInt32(Request.QueryString.Get("PhotoAlbumId"));

    using (PlanetWroxEntities myEntities = new PlanetWroxEntities ())
    {
        string photoAlbumOwner = (from p in myEntities.PhotoAlbums
            where p.Id == photoAlbumId
            select p.UserName).Single();

        if (User.Identity.Name != photoAlbumOwner && !User.IsInRole("Managers"))
        {
            Response.Redirect("~/");
        }
    }
}
```

(8) 因为现在将阻止没有合适权限的用户访问整个页面，所以不再需要隐藏 ListView 控件中的按钮。这意味着可以删除 ListView1_ItemCreated 事件处理程序的代码。如果使用的是 C#，那么不要忘了也要在 Markup 视图下从 ListView 的控件中删除处理程序的定义。

(9) 保存对所有打开文件的更改(按 Ctrl+Shift+S 组合键)，然后在浏览器中请求 NewPhotoAlbum.aspx 页面。如果有必要，可以使用之前创建的账户登录。

(10) 为相册输入一个新名称并单击 Insert。这时，该相册就和用户名一起保存了。然后向这个相册加添加一些图像。

(11) 从主 Menu 或 TreeView 中单击 Gig Pics，然后从下拉列表选择刚才创建的新相册。在页面重新加载后，就会显示这个新相册，并在屏幕底部有一个 Edit Photo Album 链接。单击该链接，将被带至 ManagePhotoAlbum.aspx 页面，在该页面上可以添加或删除相册中的图片。

(12) 单击页面页脚中的 Logout 链接。然后再次到达 Gig Pics 页面，从下拉列表中选择新相册。可注意到，此时 Edit Photo Album 链接不见了。

工作原理

这个练习首先是向 PhotoAlbum 表中添加用户名的列。通过该列，可以跟踪创建相册的用户，从而可以连同相册一起显示与该用户相关的数据。当通过选择 Update Model from Database 来运行 Update Wizard 时，数据库中的改动(例如向表添加了一列)也将反映在模型中。

在 New Photo Album 页面中，通过使用下列代码将当前用户的名字赋给这个新属性来使用它：

VB.NET

```
myPhotoAlbum.UserName = User.Identity.Name
```

C#

```
myPhotoAlbum.UserName = User.Identity.Name;
```

Page 类有一个表示与当前请求相关的用户的 User 属性。这个用户有一个包含其 Name 的 Identity。然后这个 Name 被指派给从 e.Entity 检索的 PhotoAlbum 实例的 UserName 属性。

这时，名字成功地和相册的其他信息存储在数据库中。接下来就是用这个名字做一些有用的事情。使用这个名字的第一个地方是在 PhotoAlbum 文件夹的默认页面中。在那里，使用下列 LINQ to EF 查询检索相册的 UserName：

VB.NET

```
Dim photoAlbumOwner As String = (From p In myEntities.PhotoAlbums
                                   Where p.Id = photoAlbumId
                                   Select p.UserName).Single()
```

C#

```
string photoAlbumOwner = (from p in myEntities.PhotoAlbums
                           where p.Id == photoAlbumId
                           select p.UserName).Single();
```

该代码使用 Single 方法检索单个相册的 UserName；该 UserName 在 photoAlbumId 中作了指定。然后代码的其余部分根据当前用户是否登录和是否是相册的所有者或是 Managers 组的成员，决定 Edit 链接的可见性。这样，相册的所有者和所有 Manager 都可以更改已有相册。

ManagePhotoAlbum.aspx 页面的代码执行一个类似的检查，阻止未授权用户直接访问该页面。

17.2 Profile 的其他使用方法

在本章最后一节，将介绍 ASP.NET 中 Profile 的另外两种常用方法。首先是如何对匿名用户使用 Profile，然后是讨论如何访问非当前用户的配置文件。

17.2.1 匿名标识

Profile 非常易于配置且很强大。要使登录用户能够访问他们的配置文件，所需做的只是在 web.config 中创建一些元素，ASP.NET 运行库会负责其余一切工作。但是匿名用户呢？如果想为未注册账户或未登录的访问者存储数据，该怎么办？对于这些用户，就需要启用匿名标识(anonymous identification)。通过匿名标识，ASP.NET 在 aspnet_Users 表中为站点的每位新访问者创建一个匿名用户。然后，该用户获得一个链接至数据库中匿名用户账户的 cookie。每次访问时，浏览器会连同请求一起发送 cookie，使得 ASP.NET 将一个用户进而是一个 Profile 与当前请求的用户相关联。

为了启用匿名配置文件，需要做两件事：启用匿名标识以及修改部分或全部配置文件属性，将它们提供给匿名用户。

在 web.config 中的<system.web>下方使用下列元素就可以轻松启用匿名标识：

```
<anonymousIdentification enabled="true" cookieName="PlanetWroxAnonymous" />
```

enabled 特性会启用该功能，而 cookieName 特性则用于给应用程序分配一个唯一的 cookie 名，以便在客户端存储用户 ID。

在启用匿名标识后，下一步就是修改<profile>元素下的属性，设置它们的 allowAnonymous 特性为 true：

```
<add name="FavoriteGenres" type="System.Collections.Generic.List`1[System.Int32]"
      allowAnonymous="true" />
```

现在，匿名用户也可通过代码访问这个 Profile 属性。如果对于未登录的用户，您尝试设置 Profile 属性时没有将 allowAnonymous 特性设置为 true，就会得到一个错误。对于只允许登录用户访问的页面，您需要自己写入这些属性。可以读取这些属性，但是将得到空值是在 web.config 中指定的默认值。

一旦对匿名用户启用了 Profile 属性，读取和写入它们就和处理一般的 Profile 属性一样。在本章最后的“练习”部分，要用代码修改当前主题选择器，以便对匿名用户和登录用户使用 Profile。

17.2.2 清除旧的匿名配置文件

您可能想知道，当匿名用户注册了账户后，他的配置文件会发生什么情况。答案就是什么都不会发生。旧的配置文件被丢弃，用户将获取一个与注册账户相关的新配置文件。幸运的是，这很容易解决。在某个用户从匿名用户变为经过身份验证的用户时(也就是，当他们登录时)，ASP.NET 触发您可处理的 Profile_OnMigrateAnonymous 事件。您在一个名为 Global.asax 的特殊文件(第 18 章将详细讨论)中处理这一事件，该文件用于处理应用程序或会话范围的事件的代码。在这一事件的事件处理程序内部，可访问同一用户的两个配置文件：将与用户分离的旧的匿名配置文件和与当前登录用户相关联的新配置文件。然后可删除旧的用户账户和其相关的配置文件数据，因为现在可以使用新的配置文件了。尽管在 Planet Wrox Web 站点中未采用，但这个事件处理程序也是将旧配置文件中的匿名配置文件数据复制到新配置文件的绝佳之处，如下列代码所示。

VB.NET

```
Public Sub Profile_OnMigrateAnonymous(ByVal sender As Object,
                                       ByVal args As ProfileMigrateEventArgs)
```



```
Dim anonymousProfile As ProfileCommon = Profile.GetProfile(args.AnonymousID)

' Copy over anonymous properties only
Profile.AnonymousProperty = anonymousProfile.AnonymousProperty

ProfileManager.DeleteProfile(args.AnonymousID)
AnonymousIdentificationModule.ClearAnonymousIdentifier()
Membership.DeleteUser(args.AnonymousID, True)
End Sub
```

C#

```
public void Profile_OnMigrateAnonymous(object sender, ProfileMigrateEventArgs args)
{
    ProfileCommon anonymousProfile = Profile.GetProfile(args.AnonymousID);

    // Copy over anonymous properties only
    Profile.AnonymousProperty = anonymousProfile.AnonymousProperty;

    ProfileManager.DeleteProfile(args.AnonymousID);
    AnonymousIdentificationModule.ClearAnonymousIdentifier();
    Membership.DeleteUser(args.AnonymousID, true);
}
```

可注意到，该代码使用 `Profile.GetProfile(args.AnonymousID)` 获取用户之前的匿名的配置文件。这就得到了对用户登录之前的配置文件的引用。

然后，代码继续将已有的匿名配置文件属性从旧配置文件复制到新配置文件。在这个示例中，只复制了一个名为 `AnonymousProperty` 的属性。不过，可以修改该代码来复制更多属性。注意，复制匿名用户不可访问的属性是毫无意义的。这类属性之前不能被设置，因此也没有什么可以复制。

最后三行代码将删除旧的配置文件，从 `cookie` 中清除匿名用户 ID，并从数据库中删除旧的匿名用户账户。在这一代码执行完成后，旧的配置文件就成功迁移到新配置文件，且所有旧的配置文件信息也从数据库和用户的 `cookie` 中删除。

位于 `System.Web.Profile` 名称空间(要使前面的示例工作，需要导入它)中的 `ProfileManager` 类提供了使用 `Profile` 的更有用的方法。例如，可以使用 `DeleteInactiveProfiles` 删除一段时间内不活动的用户配置文件。关于 `ProfileManager` 的更详细的信息，可以参考这个 MSDN Web 页面：<http://tinyurl.com/ManageProfile>。

17.2.3 查看其他用户的配置文件

到目前为止所看到的示例都是使用 `Profile` 访问当前用户的数据。不过，如果想显示另一用户的数据，该怎么办？例如，想在一个 `Gig Pics` 相册下显示一个用户的简介。在这种情况下，不能直接使用 `Page` 类的 `Profile` 属性，因为它提供的是当前用户的信息，而不是创建相册的用户的信息。

为了解决这一问题，`ProfileCommon` 类(`Page` 类的 `Profile` 属性的基类)提供了一个 `GetProfile` 方法。如果传递给它的名称存在，`GetProfile` 方法就从数据库中检索已有的 `Profile` 文件；如果不存在，则创建一个新的配置文件。例如，要获取 `Manager` 的配置文件，可使用下列代码：

VB.NET

```
Dim theProfile As ProfileCommon = Profile.GetProfile("Manager")
```

C#

```
ProfileCommon theProfile = Profile.GetProfile("Manager");
```

创建了这个 Profile 实例后，就可以像以前一样访问其属性。下列代码将 Manager 的 Bio 属性指派给 Label 控件的 Text 属性：

VB.NET

```
BioLabel.Text = theProfile.Bio
```

C#

```
BioLabel.Text = theProfile.Bio;
```

能够读取其他人的配置文件是极其有用的。如本章最后一个“试一试”练习所示，可以使用它向其他用户显示 Profile 的一些属性，下面的“试一试”练习就展示了这一点。不过，也可以使用类似的代码更新其他用户的配置文件。例如，可以在 Management 部分创建一个页面，允许管理在站点注册的用户的配置文件。当修改其他用户的配置文件时，最后一定要调用 Save 方法。如前面所学到的，对 Profile 的更改自动在数据库中持久化。不过，这只适用于当前用户的 Profile。要改变和持久化前面检索的 Manager 的配置文件，需要使用下列代码：

VB.NET

```
theProfile.Bio = "New Bio for the Manager account here"  
theProfile.Save()
```

C#

```
theProfile.Bio = "New Bio for the Manager account here";  
theProfile.Save();
```

在接下来的练习中，当显示创建特定相册的用户的名字以及该用户的简介时，会运用其中的一些概念。

试一试 查看其他用户的配置文件

PhotoAlbums 文件夹中的 Default.aspx 页面当前显示了一个特定相册中的图片。您还无法看到是哪个用户创建了该相册，因此想进行添加。通过进一步改进该页面，使之也能够显示用户的简介。在本“试一试”练习中将看到如何实现这两个功能。

(1)在 Markup 视图中打开 PhotoAlbums 文件夹中的 Default.aspx 页面。滚动该文件并定位到前面练习中添加的两个分隔符和 HyperLink。在分隔符和 HyperLink 控件之前，从 Toolbox 中拖出一个 Placeholder 控件，并设置其 ID 为 PhotoAlbumDetails。在该 Placeholder 控件中，拖放两个 Label 控件，然后手动修改代码，结果如下所示：

```
<asp:Placeholder ID="PhotoAlbumDetails" runat="server">  
  <h2>Photo Album Details</h2>  
  Created by:
```

```
<asp:Label ID="UserNameLabel" runat="server" Text=""></asp:Label><br />
About this user:
<asp:Label ID="BioLabel" runat="server" Text=""></asp:Label>
</asp:Placeholder>
<br /><br />
<asp:HyperLink ID="EditLink" runat="server" Text="Edit Photo Album" />
```

(2) 按 F7 键切换至该页面的 Code Behind 并定位到 ListView 控件的 DataBound 事件处理程序。当用户没有合适的权限时，第一个 Else(C#中是 else)块将隐藏 HyperLink 控件。 在该块的后面添加下列代码，检索创建相册的用户的配置文件并更新相关标签：

```
VB.NET
    EditLink.Visible = False
End If

If Not String.IsNullOrEmpty(photoAlbumOwner) Then
    Dim ownerProfile As ProfileCommon = Profile.GetProfile(photoAlbumOwner)
    UserNameLabel.Text = photoAlbumOwner
    BioLabel.Text = ownerProfile.Bio
    PhotoAlbumDetails.Visible = True
Else
    PhotoAlbumDetails.Visible = False
End If

C#
    EditLink.Visible = false;
}

if (!string.IsNullOrEmpty(photoAlbumOwner))
{
    ProfileCommon ownerProfile = Profile.GetProfile(photoAlbumOwner);
    UserNameLabel.Text = photoAlbumOwner;
    BioLabel.Text = ownerProfile.Bio;
    PhotoAlbumDetails.Visible = true;
}
else
{
    PhotoAlbumDetails.Visible = false;
}
```

(3) 保存所有更改，然后在浏览器中打开该页面。

(4) 从下拉列表中选择您创建的某个相册，就可以看到相册的详细信息出现。如果未看到，要确保从列表中选择最近创建的相册。因为是在较晚的时候才向数据库添加 UserName 列的，所以有些相册没有与之相关的用户。如果 Photo Album Details 部分仍隐藏着，就创建一个新相册并为它添加一个或多个图片。这就确保了至少有一个设置了 UserName 属性的相册。这时，如果从列表中选择该相册，将看到 Photo Album Details 部分，如图 17-11 所示。

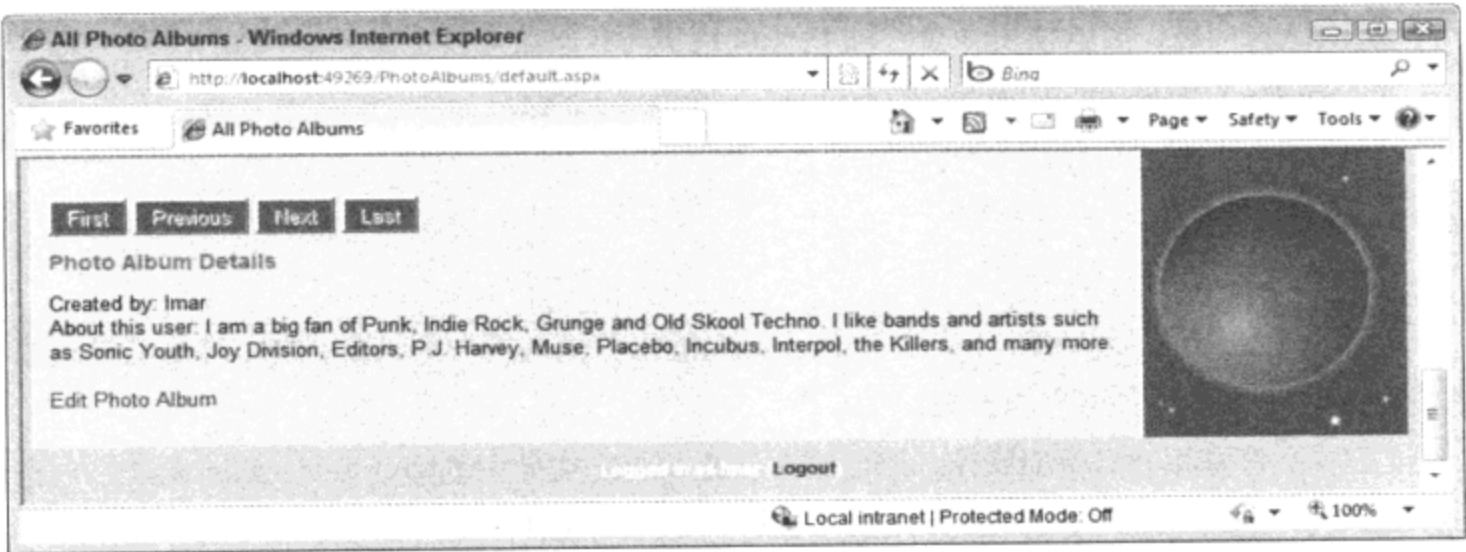


图 17-11

工作原理

本“试一试”练习中的大部分代码前面都已讨论过。在向 Photo Album 页面添加一些 Label 控件后，就使用下列代码检索相册所有者的配置文件；

VB.NET

```
Dim ownerProfile As ProfileCommon = Profile.GetProfile(photoAlbumOwner)
```

C#

```
ProfileCommon ownerProfile = Profile.GetProfile(photoAlbumOwner);
```

该代码使用 `GetProfile` 方法获得对已有配置文件的引用。返回的类是 `ProfileCommon` 类型的，是 `Profile` 属性的底层数据类型。在获取引用后，使用的方法与一般的配置文件几乎是相同的。唯一的不同是如前面所看到的：必须调用 `Save` 方法，将对 `Profile` 的更改持久化到数据库中。

17.3 关于个性化的实用提示

下面列出了一些有关个性化的实用提示：

- 不要尝试从 Login 页面访问 Profile 文件，因为它是不可用的。Profile 是在页面生命周期的早期实例化的，因此当一个 Login 控件在 Login 页面中对您进行身份验证时，已经不能将用户的配置文件与当前请求相关。应该使用 ProfileCommon 的 GetProfile 方法或重定向到另一个页面。
- 仔细考虑适合存储在 Profile 和您自己的数据库表中的内容。尽管 ASP.NET 用于存储配置文件的单条记录结构提供了一种快速和方便的解决方案，但它并不是最有效的，面对大量的数据时更是如此。不要将完整的评论或相册存储在 Profile 中，而是使用您自己的数据库表。
- Profile 的当前实现使得很难通过自己的查询从 aspnet_Profile 数据库中查询数据。例如，很难处理“给我所有喜爱 Rock 流派的用户”这样的查询，因为所有的数据存储在单列中。要

解决这些问题，可以使用一个不同 Profile 提供者，而这个提供者可以从 ASP.NET 官方网站的 Sandbox 部分(www.asp.net/downloads/sandbox/)下载。

17.4 本章小结

本章学习了使用随 ASP.NET 4 提供的 Profile 功能来存储与用户相关的数据。可使用它跟踪经过身份验证的用户以及匿名用户的数据。

建立配置文件是相当简单的操作。您需要在 web.config 文件中创建一个<profile>元素，它有一个<properties>子元素。然后使用<add>元素添加一个或多个属性。使用<group>元素可以分组相关属性。

在建立配置文件时，通过 Page 类的 Profile 属性访问其属性。这通常用来访问当前用户的配置文件。在 ASP.NET 生命周期的最后会自动持久化对这一配置文件所作的所有更改。

默认情况下，Profile 属性只可被登录用户访问。不过，可通过启用匿名标识改变这一状况。

要访问某个与当前请求无关的用户的配置文件，可以使用 GetProfile 方法。对这一配置文件的所有更改不会自动持久化，因此必须调用 Save 方法将改动保存到数据库中。

现在，页面包含了越来越多的代码，bug 和其他问题也会悄然混入您的应用程序。在第 18 章中，将学习如何使用异常处理避免这些问题出现在用户界面中。还将学习如何调试代码，在问题发生前进行修复。

17.5 练习

1. 前面创建的最喜爱的主题功能是 Profile 属性的常见用法。如果要这么做，需要向 web.config 中的配置文件添加什么代码？

2. 要在 BasePage 中设置最喜爱的主题，需要以特殊的方式访问 Profile。不是在 Page 类上访问 Profile 属性，而是按如下代码所示通过 HttpContext 访问它：

VB.NET

```
Dim myProfile As ProfileCommon = CType(HttpContext.Current.Profile, ProfileCommon)
```

C#

```
ProfileCommon myProfile = (ProfileCommon) HttpContext.Current.Profile;
```

假定有这一代码，如何重新编写 Page_PreInit，使它从 Profile 中而非 cookie 中获取最喜爱的主题？

3. 通过改变什么可以使主题存储在 Profile 中而非自定义的 cookie 中？

练习的答案见附录 A。

本章要点回顾

匿名标识	ASP.NET 的一种功能，允许跟踪站点内的用户，即使他们还没有注册账户或是还没有登录
ASP.NET Profile	允许将用户的信息存储到站点或者从站点中检索用户信息的 ASP.NET 应用程序服务
ASP.NET Profile 提供者	负责存储和检索与配置文件相关的数据的 ASP.NET 提供者
EndRequest	应用程序触发的一个事件，其中对配置文件所做的修改将被持久化到数据库中
OnMigrateAnonymous	ASP.NET Profile 功能触发的一个事件，可以在 Global.asax 中进行处理，以便将匿名属性复制到新的配置文件中
个性化	根据用户的偏好或其他信息为用户提供自定义内容的过程
配置文件组	用于分组相关配置文件属性的机制

本章要点

- 如何编写可检测和处理运行时发生的错误的代码，同时向用户屏蔽详细错误信息
- 如何检测生产机器上发生的错误，以便采取防范措施
- 调试的定义以及 Visual Web Developer 提供的调试工具
- 可使用什么工具获取开发或生产环境中运行的系统以及代码的信息

俗话说，有得必有失。同样，要编写代码，必然会带来一些 bug。不管您怎么努力，也不管您有多优秀，代码还是会包含一些影响 Web 站点行为的问题。

当然，您可以努力最小化这些 bug 的影响，争取 bug 数为零。而在这方面，ASP.NET 运行库和 Visual Web Developer 提供了大量工具。

首先，.NET 支持的语言实现了异常处理(exception handling)，这是一种标识和处理运行时发生的错误的方法。通过处理这些错误，可以向用户提供一个友好的错误消息。同时，还可以记录这些错误，以便在它们再次出现前就进行修复。本章将介绍异常处理的工作原理，以及如何记录错误。

在代码进入生产环境前，需要首先编写和调试它。为帮助调试代码，VWD 提供了一个丰富的工具集，其中包括了逐行单步执行代码、在运行时查看变量和对象，甚至在调试过程中改变它们的方法。该工具集还提供了有关代码执行路径(execution path)的有用信息：应用程序执行代码、方法、事件处理程序、If 和 Else 语句等时采取的路径。

18.1 异常处理

无论何时编写代码，都可能导致无法编译的代码、应用程序崩溃或者出现意料之外的行为。出现问题的原因有很多：代码输入错误、在运行时连接的数据库服务器突然宕机、混淆了逻辑并意外地从数据库表中删除所有记录而非一条记录、尝试从仍有相关联记录的数据库表中删除记录、尝试在没有适当权限的情况下将一个文件写入某个文件夹、用户输入了不正确的数据等。

为了理解这些问题并考虑预期、避免和处理它们的方法，首先需要了解可能在 Web 站点中可能

发生的不同类型的错误。在理解了主要区别后，本节其余部分将全面讨论防止和解决它们的方法。

18.1.1 不同类型的错误

在广义上，可将错误分为以下几类：

- **语法错误**：如因输入错误、缺少关键字或其他不正确的代码导致的错误。
- **逻辑错误**：在应用程序中看上去运行良好但会产生意料之外的、不想要的结果的错误。
- **运行时错误**：导致应用程序在运行时崩溃或出现意料之外的行为的错误。

接下来将讨论这每种错误，同时会介绍如何避免和解决这些问题。

1. 语法错误

语法错误，又称编译错误(compile error)，是最容易发现和纠正的错误，因为它们发生在开发时。IDE 会告知错误发生的时间，通常也会阻止运行带有这种错误的程序。语法错误是因为简单的输入错误、缺失或重复关键字和字符等导致的。下列示例显示了编译器在开发时捕获的错误。编译器是一种将使用 VB.NET 或 C#编写的人类可读的代码转换为可执行的机器可读的代码的程序。

VB.NET

```
mailBody = mailBody.Repalce("##Name##", Name.Text) ' Replace is misspelled

Response.Write() ' Required parameter for the
                  ' Write method is missing

If i > 10 ' Missing keyword Then
    ' Do something here
End If
```

C#

```
mailBody = mailBody.Repalce("##Name##", Name.Text); // Replace is misspelled

Response.Write(); // Required parameter for the
                  // Write method is missing

if (i > 10) // Missing opening brace or
            // extraneous closing brace

    // Do something here
}
```

编译错误通常显示在 Error List(可通过 View | Error List 菜单访问)中，如图 18-1 所示。

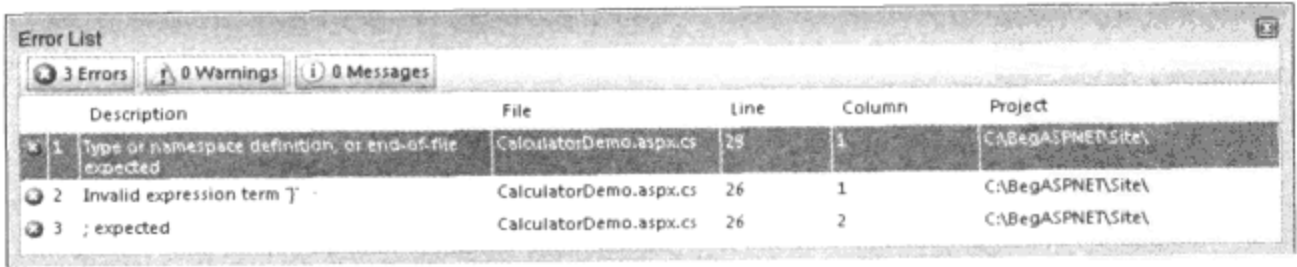


图 18-1

可以强制编译器提供站点中所有编译错误的一个最新列表。方法就是从主菜单中选择 Build | Rebuild Web Site 选项。

要定位到错误发生的位置以进行纠正，可双击 Error List 中的错误。要遍历错误和发生错误的代码，可以单击 Error List 中的错误，然后按下 F8 键找到下一个错误。

2. 逻辑错误

逻辑错误通常较难发现，因为它们在编译时没有问题，而只在代码执行过程中发生。考虑下面这个有 bug 的示例：

```
VB.NET
Dim fromAddress As String = "you@yourprovider.com"
Dim toAddress As String = EmailAddress.Text
myMessage.From = New MailAddress(toAddress)
myMessage.To.Add(New MailAddress(fromAddress))

C#
string fromAddress = "you@yourprovider.com";
string toAddress = EmailAddress.Text;
myMessage.From = new MailAddress(toAddress);
myMessage.To.Add(new MailAddress(fromAddress));
```

尽管该示例中的问题显而易见(To 和 From 地址混乱)，但要在有着 250 行代码的文件中发现却很难。另外，编译器会接受这种错误，而您只有在 Inbox 中发现想发送给访问者的消息时才会注意到这一错误。

跟踪和解决逻辑错误的最好方法是使用 VWD 的内置调试功能。本章后面将深入讨论这些工具。

3. 运行时错误

运行时错误是在运行时发生的，这使得它们非常难以跟踪。假设您有一个站点，它包含了一个隐藏在页面某处的 bug。您没有发现它，但某个访问者发现了，她获得了令人讨厌的错误消息(后面会介绍更多内容)。此时，您可以做些什么？可做的工作并不多，因为很可能您的访问者都没有向您通知有关错误的信息。

因此，有一个好的错误处理策略是很重要的，这样可避免可能出现的错误并适当地处理，另外还可以在错误发生时选择记录其相关信息。

下面一节讨论检测和处理错误(.NET 中称为异常)；本章后面，将学习如何记录错误和防止用户看到带有详细异常消息的丑陋页面。

18.1.2 捕获和处理异常

通常，发生严重异常时，用户会得到一个错误消息。例如，如果尝试向未启动运行或是不允许您连接的邮件服务器发送消息，就会得到提供了错误详细信息的 Exception 类型。默认情况下，这一异常会弹出到用户界面，从而出现所谓的 Yellow Screen of Death，这里借用了 Windows 系统崩溃时出现的“Blue Screen of Death(蓝屏死机)”的概念。下一个“试一试”练习中将介绍这一错误的真实示例。

显然，如果能预料到这一异常并编写一些代码防止异常在用户界面中出现则更好。例如，可以向用户显示一个友好的消息，通知他们现在不能发送消息。

幸运的是，对这些场景的支持已深入集成到.NET 编程语言中，如 C#和 Visual Basic .NET。在这些语言中，可使用 Try Catch Finally 块(在 C#中是 try catch finally)，将可能抛出异常的代码包装在 Try 块中。

当异常发生时，Try 块中的其余代码就被跳过，而执行 Catch 块中的代码来处理错误。可以有多个处理特定异常的 Catch 块，但只有一个被触发，这在后面一个“试一试”练习中可看到。

Try 块或 Catch 块后面可跟一个 Finally 块。Finally 块中的代码总是会触发，而不管是否有异常发生。因此，这里是编写清理代码的理想位置。

Catch 块和 Finally 块是可选的，不过还是至少需要其中一个。

下列示例中的代码尝试发送一封电子邮件，然后设置 Label 控件的 Text 属性为 userMessage 变量的值。在 Try 块(当代码成功执行时)或 Catch 块(当发生错误时)中为 userMessage 变量赋值。不管哪种情况，在 Finally 块中，这个 userMessage 总是赋值给 Label:

VB.NET

```
Dim userMessage As String = String.Empty
Try
    mySmtpClient.Send(myMessage)
    userMessage = "Message sent"
Catch ex As Exception
    userMessage = "An unknown error occurred."
Finally
    Message.Text = userMessage
End Try
```

C#

```
string userMessage = string.Empty;
try
{
    mySmtpClient.Send(myMessage);
    userMessage = "Message sent";
}
catch (Exception ex)
{
    userMessage = "An unknown error occurred.";
}
finally
{
    Message.Text = userMessage;
}
```

在这个代码示例中，Catch 块用来处理 System.Exception 类型(.NET Framework 中所有异常的基类)的异常，然后这一异常将被发送到 Catch 块的 ex 变量中(或者由该变量捕获)。由于这个示例并没有使用 ex 变量，所以可将它省去:

VB.NET

```
Catch
    userMessage = "An unknown error occurred."
End Try
```

```
C#
catch
{
    userMessage = "An unknown error occurred.";
}
```

如果预计代码会遇到多个异常，则指定 `Exception` 类型会很有用。在那种情况下，可以有多个用于不同异常类型的 `Catch` 块。下列代码可以处理在邮件发送操作中可能出现的特定的 `SmtpException` 异常，也能够使用其一般的 `Catch` 块捕获所有其他异常。

```
VB.NET
Try
    mySmtpClient.Send(myMessage)
Catch smtpException As SmtpException
    Message.Text = "Sorry, an error occurred while sending your message."
Catch ex As Exception
    ' Something else went wrong.
End Try
```

```
C#
try
{
    mySmtpClient.Send(myMessage);
}
catch (SmtpException smtpException)
{
    Message.Text = "Sorry, an error occurred while sending your message.";
}
catch (Exception ex)
{
    // Something else went wrong.
}
```

异常处理块的顺序很重要。.NET 从上到下扫描 `Catch` 块的列表并只触发与特定类型的异常匹配的第一个块中的代码。在上面的示例中，当 `SmtpException`(`Exception` 的子类)发生时，它将被处理 `SmtpException` 类型异常的 `Catch` 块捕获。尽管 `SmtpException` 也是一种 `Exception`，但是最后的 `Catch` 块中的代码也不会再触发，因为只有第一个匹配的 `Catch` 块才会被处理。因此，如果颠倒本示例中 `Catch` 块的顺序，将执行更通用的 `Exception` 块，而不会执行 `SmtpException` 块中的代码。

在接下来的“试一试”练习中，将介绍如何在代码中使用 `Try` 块和 `Catch` 块。

试一试

处理异常

在这个“试一试”练习中，将看到如何编写异常处理代码，以捕获无效电子邮件地址错误。这是一种很常见的问题，因为用户经常会输入不正确的电子邮件地址。例如，它们可能忘记输入符号 `@`，或者输入逗号而非点号。当尝试使用无效的 `From` 或 `To` 地址发送电子邮件时，如果没有采取适当的处理措施，那么代码将会崩溃。将在 `Demos` 文件夹的一个单独的页面中使用 `Try Catch` 代码，这样可以密切关注其行为。在理解了其工作原理后，可以修改 `ContactForm.ascx` 用户控件和合并异

常处理代码。不能将该代码直接应用于用户控件的原因在于它使用了一个 Ajax UpdatePanel，它默认屏蔽了向用户显示异常的详细信息。

- (1) 在 Demos 文件夹中创建一个新的文件并命名为 ExceptionHandling.aspx。将该页面基于您的自定义模板并设置其 Title 为 Exception Handling Demo。
- (2) 添加一个 Label 控件到主内容区并设置其 ID 为 Message。
- (3) 切换至 Code Behind 并在文件顶部添加 System.Net.Mail 名称空间的 Imports 或 using 语句：

VB.NET

```
Imports System.Net.Mail
```

C#

```
using System.Net.Mail;
```

(4) 切换至 Design 视图，通过双击页面的只读区域，为该页面的 Load 事件建立事件处理程序。添加下列代码到该事件处理程序中。可注意到，该代码与在 ContactForm.aspx 用户控件中添加的代码几乎相同，因此可以通过从该文件中复制部分代码来省去自行输入。注意，From 地址是无效的，因为它没有包含@符号。这里是故意这么做的，以便触发一个异常。

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load

    Dim myMessage As MailMessage = New MailMessage()
    myMessage.Subject = "Exception Handling Test"
    myMessage.Body = "Test message body"

    myMessage.From = New MailAddress("you.yourprovider.com")
    myMessage.To.Add(New MailAddress("you@yourprovider.com"))

    Dim mySmtpClient As New SmtpClient()
    mySmtpClient.Send(myMessage)
    Message.Text = "Message sent"
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    MailMessage myMessage = new MailMessage();
    myMessage.Subject = "Exception Handling Test";
    myMessage.Body = "Test message body";

    myMessage.From = new MailAddress("you.yourprovider.com");
    myMessage.To.Add(new MailAddress("you@yourprovider.com"));

    SmtpClient mySmtpClient = new SmtpClient();
    mySmtpClient.Send(myMessage);
    Message.Text = "Message sent";
}
```

不要忘了将两个电子邮件地址改为您自己的地址，也不要忘了至少在其中一个电子邮件地址中引入错误。关于为发送电子邮件设置 web.config 文件的更多内容，请参考第 9 章内容。

(5) 按 Ctrl+F5 组合键在浏览器中打开该站点，将看到带有错误消息的“Yellow Screen of Death”，如图 18-2 所示。

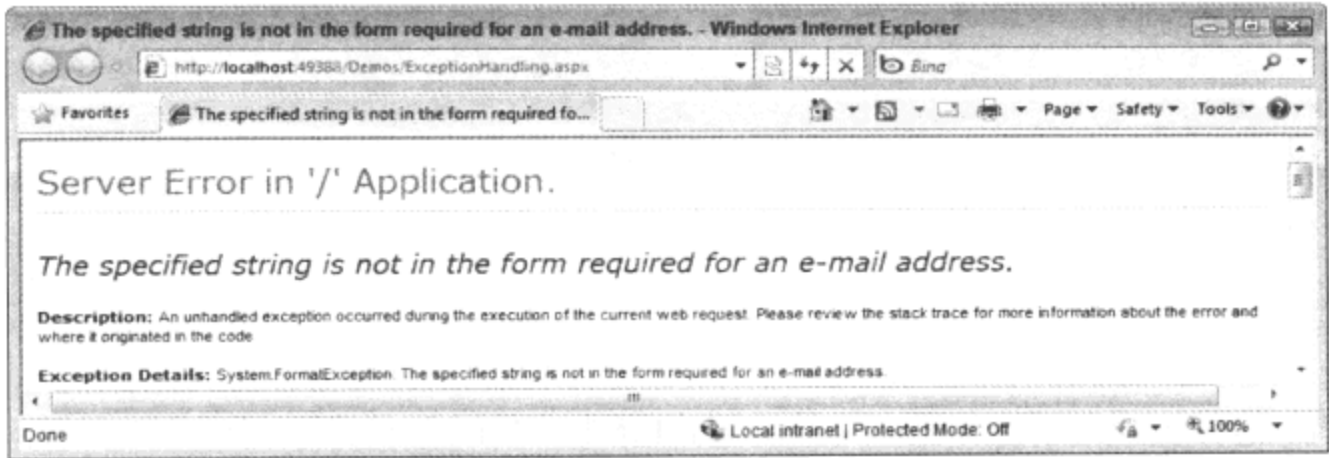


图 18-2

注意，Exception Details 部分表明发生了 System.FormatException 异常。

(6) 回到 VWD 并将分配地址以及发送该消息的代码包装到一个 Try Catch 块中。

VB.NET

```
Try
    myMessage.From = New MailAddress("you.yourprovider.com")
    myMessage.To.Add(New MailAddress("you@yourprovider.com"))
    Dim mySmtpClient As New SmtpClient()
    mySmtpClient.Send(myMessage)
    Message.Text = "Message sent"
Catch
    Message.Text = "An error occurred while sending your e-mail. Please try again."
End Try
```

C#

```
try
{
    myMessage.From = new MailAddress("you.yourprovider.com");
    myMessage.To.Add(new MailAddress("you@yourprovider.com"));
    SmtpClient mySmtpClient = new SmtpClient();
    mySmtpClient.Send(myMessage);
    Message.Text = "Message sent";
}
catch
{
    Message.Text = "An error occurred while sending your e-mail. Please try again.";
}
```

注意，代码仍然包含一个无效的电子邮件地址。

(7) 保存所有更改并在浏览器中再次请求该页面，将看到如图 18-3 所示的一个用户友好的错误消息。

抛出的异常在 Catch 块中被捕获。用户现在得到的是一个解释发生了什么错误的友好消息，而不是带有异常的所有详细技术信息的错误页面。

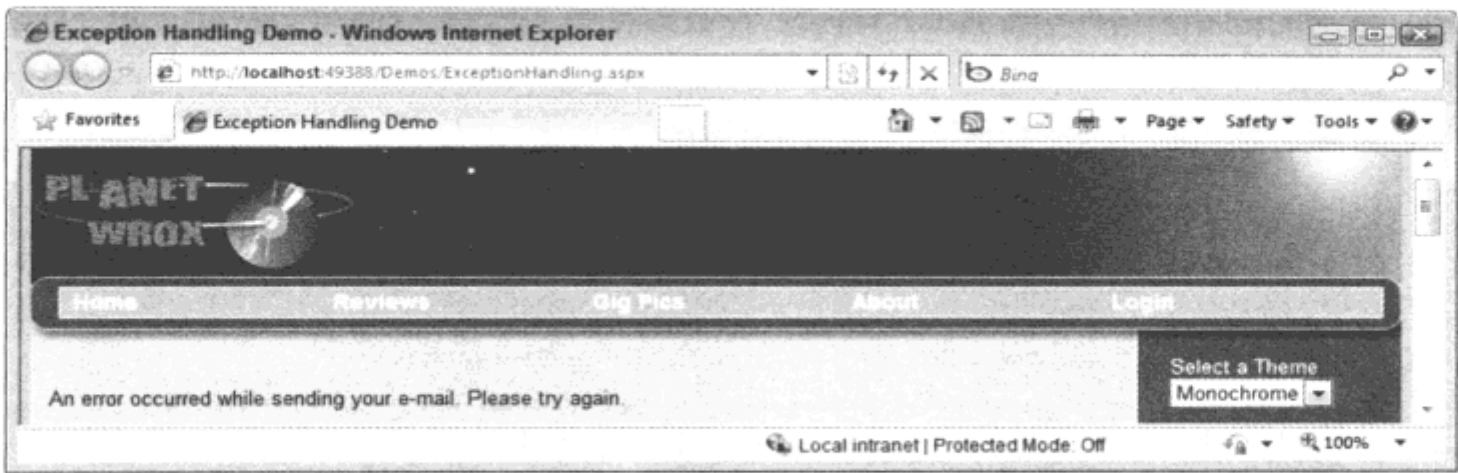


图 18-3

(8) 回到代码中并改正电子邮件地址:

VB.NET

```
myMessage.From = New MailAddress("you@yourprovider.com")
```

C#

```
myMessage.From = new MailAddress("you@yourprovider.com");
```

(9) 保存更改并再次请求该页面，将获得一条表示消息发送成功的信息。

(10) 从 Controls 文件夹中打开 ContactForm.aspx，切换至其 Code Behind 文件，将创建并发送消息的代码包装到下面的 Try Catch 块中:

VB.NET

```
Try
    Dim myMessage As New MailMessage()
    ...
    System.Threading.Thread.Sleep(5000)
Catch
    Message.Text = "An error occurred while sending your e-mail. Please try again."
Finally
    Message.Visible = True
End Try
```

C#

```
try
{
    MailMessage myMessage = new MailMessage();
    ...
    System.Threading.Thread.Sleep(5000);
}
catch
{
    Message.Text = "An error occurred while sending your e-mail. Please try again.";
}
finally
{
    Message.Visible = true;
}
```

}

注意,设置 Message 控件的 Visible 属性为 True 的代码行现在位于 Finally 块中。这样就使得 Label 可见,而不管是否发生了错误。

从现在起,在发送电子邮件的过程中无论何时发生错误,用户都将得到一个正常的错误消息,而不是.NET 默认显示的完整的详细错误页面。

工作原理

当在运行时发生异常时,.NET 会检查代码是否包装在 Try Catch 块中。如果是那样,它就扫描 Catch 块的列表,寻找与抛出的异常匹配的 Exception 类型。只有与 Exception 匹配的第一个 Catch 块将被调用;所有其他的 Catch 块都被忽略。紧随导致异常的行的 Try 块中的代码将不再执行。

在这个“试一试”练习中,代码用于处理通用的 Exception 类型。由于该类型是所有异常的基类型,它将为所有异常触发,而不管它们的类型。不过正如在练习前所学到的,可以有多个 Catch 块,每个块处理一种特定类型的异常。

通过这些 Try Catch Finally 块,可以编写代码帮助处理可能会在代码中发生的错误。将可能抛出错误的代码包装到一个 Try Catch 块中是个很好的实践,这样可以适当地处理它。可能抛出异常的代码示例包括如下几种情况:

- 发送电子邮件: 邮件服务器可能宕机或是您可能没有权限访问它。
- 访问数据库: 服务器可能宕机、您可能没有权限、您可能因为违反外键约束而出错(如第 15 章那样)等。
- 尝试向磁盘写入一个上传的文件: 磁盘可能满了、您可能没有权限写入磁盘,或者您提供了一个无效的文件名。

尽管 Try Catch 块可以极好地避免异常弹出到用户界面,但也要谨慎地使用它们,因为使用它们会有一定的开销。Try Catch 块通常要慢于没有它们的代码,因此对于可以用其他方式避免的错误,不要使用它。考虑下面这个将两数相除的示例:

VB.NET

```
Dim value1 As Integer = Convert.ToInt32(ValueBox1.Text)
Dim value2 As Integer = Convert.ToInt32(Value2Box.Text)
Try
    result = value1 / value2
    ResultLabel.Text = result.ToString()
Catch ex As DivideByZeroException
    ResultLabel.Text = "Sorry, division by zero is not possible."
End Try
```

C#

```
int value1 = Convert.ToInt32(ValueBox1.Text);
int value2 = Convert.ToInt32(ValueBox2.Text);
try
{
    result = value1 / value2;
    ResultLabel.Text = result.ToString();
}
catch (DivideByZeroException ex)
```



```
{
    ResultLabel.Text = "Sorry, division by zero is not possible.";
}
```

在这个示例中，代码用于处理 `DivideByZeroException` 异常。这一异常是在 `value2` 包含 0 值时抛出的。虽然初看起来，在这里实现异常处理是个不错的想法，但是更好的做法是在进行除法运算之前检查该值，而不是让异常发生：

VB.NET

```
If (value2 <> 0) Then
    result = value1 / value2
    ResultLabel.Text = result.ToString()
Else
    ResultLabel.Text = "Sorry, division by zero is not possible."
End If
```

C#

```
if (value2 != 0)
{
    result = value1 / value2;
    ResultLabel.Text = result.ToString();
}
else
{
    ResultLabel.Text = "Sorry, division by zero is not possible.";
}
```

当然，比这更好的做法是在页面上放置一个 `CompareValidator` 控件，以确保 `ValueBox2` 不包含 0 值。第 9 章介绍了如何使用这一控件。

尽管 `Try Catch` 块对于捕获您预期的异常很有用，但那些您不能预期的错误呢？如何处理这样的错误？由于它们是意料之外的，您无法知道它们何时发生，因此很难编写代码处理它们。

为了解决这一问题，下一节介绍了如何全局捕获和记录未处理的异常并通过电子邮件发送有关它们的信息。这样，页面开发人员就知道发生了这些异常，并有机会在它们再次发生前处理它们。

18.1.3 全局错误处理和自定义错误页面

为了向用户屏蔽异常的详细技术信息，应向他们提供一个用户友好的错误页面。幸运的是，ASP.NET 支持定义自定义错误页面(custom error page)，即，在异常发生时显示给用户的 ASPX 页面。可以将不同类型的错误(服务器错误、未找到页面错误、安全问题等)映射到不同页面。

在 `web.config` 文件的 `customErrors` 元素中定义想要显示的错误页面。该元素通常是这样的：

```
<customErrors mode="On" defaultRedirect="~/Errors/Error500.aspx"
    redirectMode="ResponseRewrite">
    <error statusCode="404" redirect="~/Errors/Error404.aspx" />
    <error statusCode="500" redirect="~/Errors/Error500.aspx" />
</customErrors>
```

`mode` 特性决定了站点的访问者是否可看到详细的错误页面。该特性支持下列 3 个值：

- **On**：在错误发生时，站点的访问者总是看到自定义错误页面。

- Off: 不显示自定义错误页面，在页面上显示完整的详细错误信息。
- RemoteOnly: 通过这一设置，完整的详细错误信息只显示给本地用户(他们使用运行站点的计算机浏览站点)，而其他用户看到的是自定义错误页面。这一设置使您可以在开发期间看到站点上的错误消息，而只向用户显示自定义错误页面。

在 customErrors 元素的起始和结束标记之间，为想支持的每个 HTTP 状态码定义单独的 < error /> 元素。前面的配置定义了两个自定义页面：一个是在请求的页面未找到(404 状态码)时显示；另一个是在发生服务器错误(500 状态码)时显示。

对于所有其他没有显式定义的 HTTP 状态码，将使用 defaultRedirect 特性指派自定义错误页面。redirectMode 特性决定了将新页面显示给用户的方式，本章后面将进行讨论。

尽管自定义错误页面向用户屏蔽了详细的异常信息，但它们并不能向您通知发生了异常。这些页面所做的就是隐藏真正的错误并显示一个带有自定义错误消息的页面。为了得到这些异常的通知，需要编写一些代码，它们能查看异常并向您发送带有详细信息的邮件。另外，可以编写在数据库中插入详细错误信息或将它们写入文本文件的代码。

ASP.NET 提供了一个便利的、集中的位置来编写异常发生时触发的代码。您是在 Global.asax 文件的名为 Application_Error 的特殊事件处理程序中编写这一代码的。而在 Global.asax 文件(注意.asax 扩展名)中可编写为响应站点范围内的事件(如应用程序启动、发出对某个资源的请求)而触发的代码。在这个事件处理程序中，可以收集异常的相关数据，将它粘贴到电子邮件消息中，然后发送到您自己的 Inbox 中。这就向您提供了有关在站点中发生的异常的详细信息，从而可以帮助尽快解决问题。下面的“试一试”练习将介绍如何编写这一代码。

试一试 处理站点范围内的异常

在这个“试一试”练习中，将学习如何在 Global.asax 文件中编写代码，通过电子邮件发送异常消息。另外，还将学习如何创建发生错误时显示给用户的全局错误页面。

(1) 右击 Solution Explorer 中的 Web 站点并选择 Add New Item 命令。在带有模板的列表中，选择 Global Application Class。将其名称设置为 Global.asax 并单击 Add 按钮。

(2) 在该文件代码的顶部，Application 指令后添加下列 Import 语句。注意，在 Markup 视图添加 Import 语句时，VB.NET 和 C#都使用关键字 Import，而不是通常在 Code Behind 文件中使用的 Imports 和 using。

```
VB.NET
<%@ Application Language="VB" %>
<%@ Import Namespace="System.Net.Mail" %>

C#
<%@ Application Language="C#" %>
<%@ Import Namespace="System.Net.Mail" %>
```

(3) 在 Global.asax 文件中应该已存在的 Application_Error 事件处理程序中，添加下列突出显示的代码，这些代码会在站点中发生未处理的异常时触发。如果没有此事件处理程序，则完整地输入下列代码段，包括未突出显示的部分。

VB.NET

```
Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    If HttpContext.Current.Server.GetLastError() IsNot Nothing Then
        Dim myException As Exception = HttpContext.Current.Server.GetLastError().
            GetBaseException()

        Dim mailSubject As String = "Error in page" & Request.Url.ToString()
        Dim message As String = String.Empty
        message &= "<strong>Message</strong><br />" & myException.Message & "<br />"
        message &= "<strong>StackTrace</strong><br />" &
            myException.StackTrace & "<br />"
        message &= "<strong>Query String</strong><br />" &
            Request.QueryString.ToString() & "<br />"
        Dim myMessage As MailMessage = New MailMessage("you@yourprovider.com",
            "you@yourprovider.com", mailSubject, message)
        myMessage.IsBodyHtml = True
        Dim mySmtpClient As SmtpClient = New SmtpClient()
        mySmtpClient.Send(myMessage)
    End If
End Sub
```

C#

```
void Application_Error(object sender, EventArgs e)
{
    if (HttpContext.Current.Server.GetLastError() != null)
    {
        Exception myException = HttpContext.Current.Server.GetLastError().
            GetBaseException();

        string mailSubject = "Error in page" + Request.Url.ToString();
        string message = string.Empty;
        message += "<strong>Message</strong><br />" + myException.Message + "<br />";
        message += "<strong>StackTrace</strong><br />" + myException.StackTrace +
            "<br />";
        message += "<strong>Query String</strong><br />" +
            Request.QueryString.ToString() + "<br />";
        MailMessage myMessage = new MailMessage("you@yourprovider.com",
            "you@yourprovider.com", mailSubject, message);
        myMessage.IsBodyHtml = true;
        SmtpClient mySmtpClient = new SmtpClient();
        mySmtpClient.Send(myMessage);
    }
}
```

不要忘了更改要传递给 MailMessage 的构造函数的两个电子邮件地址。第一个地址表示发送者地址，而第二个表示接收者地址。

- (4) 保存所有更改并关闭 Global.asax 文件。
- (5) 接着，打开 web.config 文件，添加下面的 customErrors 元素作为<system.web>的直接子元素：

```
<system.web>
  <customErrors mode="On" defaultRedirect="~/Errors/OtherErrors.aspx"
    redirectMode="ResponseRewrite">
    <error statusCode="404" redirect="~/Errors/Error404.aspx" />
  </customErrors>
```


保存并关闭配置文件。

(6) 在 Web 站点中创建一个新的文件夹并命名为 Errors。

(7) 在这个新文件夹中，创建两个新的 Web 窗体，并将它们分别命名为 Error404.aspx 和 OtherErrors.aspx。确保它们都基于您的自定义模板，以便它们使用主母版页并且继承自 BasePage。如果按照了第 17 章的练习进行操作，且现在使用 Profile 存储用户最喜爱的主题，那么可以查看本练习结尾的 Common Mistakes 部分，了解在为自定义的 404 错误页面使用母版页和 BasePage 时容易犯的错误。

(8) 设置 Error404.aspx 的 Title 为 File Not Found。在主内容的内容占位符中添加下列标记：

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <h1>File Not Found</h1>
  <p>The page you requested could not be found. Please check out the
    <a href="/" runat="server">Homepage</a>
    or choose a different page from the menu.</p>
  <p>The Planet Wrox Team</p>
</asp:Content>
```

(9) 切换到 Design 视图，双击页面以建立一个 Page_Load 处理程序，然后向该处理程序中添加下面的代码：

```
VB.NET
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Response.Status = "404 Not Found"
    Response.StatusCode = 404
End Sub
```

```
C#
protected void Page_Load(object sender, EventArgs e)
{
    Response.Status = "404 Not Found";
    Response.StatusCode = 404;
}
```

(10) 打开通用的 OtherErrors.aspx 页面，设置其 Title 为 An Error Occurred，并输入下面的内容：

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <h1>Un known error occurred</h1>
  <p>An error occurred in the page you requested. The error has been logged and
    we'll fix it ASAP.</p>
  <p>The Planet Wrox Team</p>
</asp:Content>
```

因为这个页面将用于 404 错误以外的所有可能出现的错误，所以显式地设置一个 Status 或 StatusCode 并没有意义。

(11) 通过按 Ctrl+Shift+S 组合键保存对所有打开文件的更改，然后关闭它们。右击 Solution Explorer 中的 Default.aspx 并选择 View In Browser 命令。一旦页面完成加载，通过将浏览器的地址栏改为 http://localhost:1208/DefaultTest.aspx，请求一个像 DefaultTest.aspx 这样的不存在的页面。显

然，该页面并不存在，因此会得到一个错误。但是将获取一个在本“试一试”练习中定义和创建的错误页面，如图 18-4 所示，而不是详细错误页面。

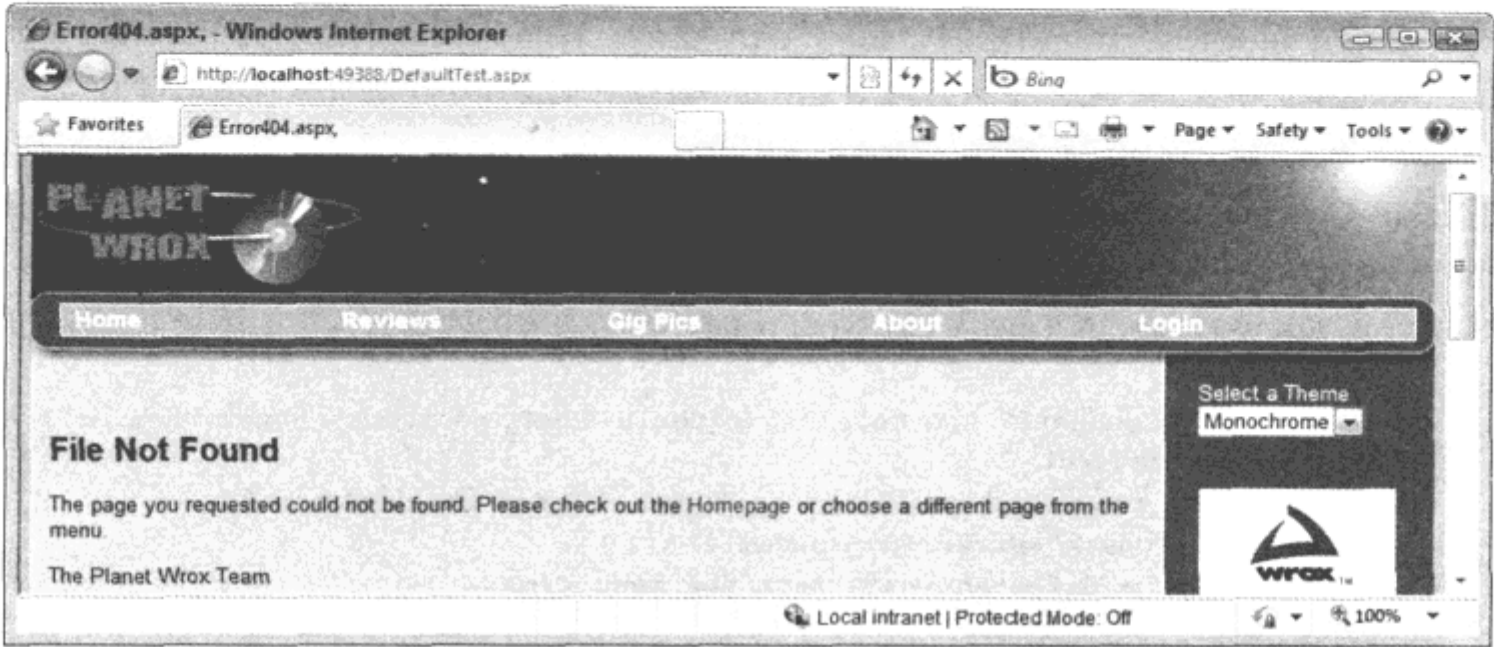


图 18-4



常见错误：如果没有显示这个错误消息，而是得到一个“File not found”异常页面，那么需要在 web.config 文件中检查 customErrors 部分的语法。另外，通过在浏览器中直接请求实际的错误页面(Error404.aspx 和 OtherErrors.aspx)，检查是否可以成功地请求它们。如果包含一个错误(例如，如果忘记设置页面标题)，那么它们就不能用作自定义错误页面。

如果按照第 17 章的练习进行操作，并重写首选主题选择器来使它使用 Profile，那么也会得到这个错误页面。由于内部处理 404 错误的方式，您不能在错误页面或者作为错误页面的基础的母版页中使用 Profile。为了解决这个问题，可以将 Error404.aspx 页面重建为一个不使用母版页和 BasePage 的标准 Web 窗体，现在一切应该正常了。也可以将 web.config 中的 redirectMode 特性设置为 ResponseRedirect，但是从搜索引擎优化的角度来说，并不推荐这么做。

除了浏览器中的错误页面以外，还会得到一个提供了有关错误的更详细信息的邮件消息。图 18-5 显示了请求不存在的页面时得到的消息。

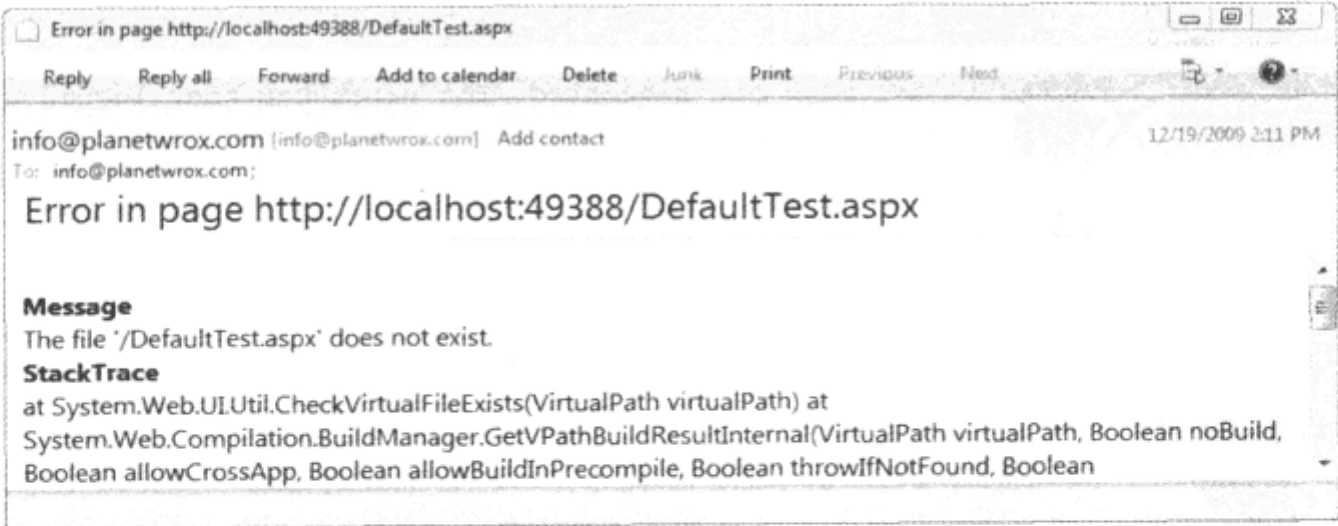


图 18-5

工作原理

本练习中有两个重要的部分值得注意。第一个部分是 ASP.NET 运行库使用自定义错误页面向用户隐藏带有详细错误信息的 Yellow Screen of Death 的方式。这样做有两个作用。首先，它防止了像密码或有关数据库连接的信息这样的隐私数据出现在错误消息中。其次，它向用户屏蔽了他们可能并不理解的错误消息，而显示了符合站点外观的、美观的错误页面。

在本练习中您需要做的唯一一件事是在 web.config 文件中启用自定义错误，提供一个或多个想为站点中可能发生的错误显示的页面。配置元素可用于为不同异常建立不同页面：

```
<customErrors mode="On" defaultRedirect="~/Errors/OtherErrors.aspx"
    redirectMode="ResponseRewrite">
    <error statusCode="404" redirect="~/Errors/Error404.aspx" />
</customErrors>
```

当.NET 遇到 404 异常(例如，请求无法找到的页面)时，您将被带至 Error404.aspx 页面。该页面的名称和内容完全取决于您，您可以向用户提供表明发生了什么情况的自己的解释。注意，这只对 ASP.NET 支持的文件类型起作用，如.aspx 文件。而当没有使用 IIS(Microsoft 的 Web 服务器)的 Integrated Pipeline 模式时，将对.html 文件或图像不起作用。在第 19 章将更详细地讨论 Integrated Pipeline 模式。

这个练习包含可以改进搜索引擎优化(SEO)效果的两个关键元素。首先，注意 redirectMode 被设为 ResponseRewrite。另一个选项是 ResponseRedirect。第 7 章曾讨论过 Server.Transfer 和 Response.Redirect 的区别。这两种设置基于完全相同的原理。如果将 redirectMode 设为 ResponseRedirect，浏览器(以及搜索引擎)将被重定向到错误页面。然后错误页面将返回 404 状态码，从而使搜索引擎认为不能找到错误页面本身。然而，如果设为 ResponseRewrite，那么原来请求的页面被会导致 404 错误代码，而 Error404.aspx 页面的内容则被引导到浏览器。这样，搜索引擎就能够正确地更新它们的索引。ResponseRewrite 设置的唯一缺点是不能在页面或者该页面所基于的母版页中使用 Profile 功能。从前面可以看到，解决这种问题的最好方法是创建一个不基于自定义模板的页面。

可以改进 SEO 的第二个部分是 Code Behind 文件中 Error404.aspx 的代码。这些代码将 HTTP 状态码设置为 404，以指出在服务器上不能找到页面。如果没有这两行代码，搜索引擎将不知道页面不存在，从而会不断地尝试索引它。

注意，只有 404 错误代码被重定向到自己的页面。其他所有异常都将导致加载通用的 OtherErrors.aspx 页面。然而，您可以向<customErrors>元素添加多个<error />元素，其中每个元素代表一个不同的状态码。要想查看 HTTP 状态码的列表，可以查看这篇知识库文章：<http://support.microsoft.com/default.aspx/kb/943891>。

这个练习的另外一个主要部分就是 Global.asax 中在发生未处理异常时触发的代码。在那种情况下，会触发 Application_Error 事件处理程序。在这个事件处理程序中，可使用下列代码检索发生的异常。

VB.NET

```
If HttpContext.Current.Server.GetLastError() IsNot Nothing Then
    Dim myException As Exception =
        HttpContext.Current.Server.GetLastError().GetBaseException()
```



```
C#
if (HttpContext.Current.Server.GetLastError() != null)
{
    Exception myException =
        HttpContext.Current.Server.GetLastError().GetBaseException();
}
```

为了首先获得导致问题的根异常，需要在 `Server.GetLastError()` 返回的 `Exception` 上调用 `GetBaseException()`。然后，这个存储在 `myException` 变量中的 `Exception` 实例提供对 `Message` 和 `StackTrace` 这样的有用属性的访问。在这一练习中，显示在有关错误的邮件中的 `StackTrace` 所包含的信息并不是您真正感兴趣的。不过，对于其他异常来说，如未正确配置邮件服务器抛出的异常或除 0 异常，`StackTrace` 会提供有关生成错误的文件、导致它错误的方法甚至代码中发生错误的行号等信息，使得很容易发现错误并纠正。

代码的其余部分创建了一个带有详细错误信息的电子邮件消息。它还用下列代码添加了有关查询字符串的信息。

```
VB.NET
message &= "<strong>Query String</strong><br />" &
    Request.QueryString.ToString() & "<br />"

C#
message += "<strong>Query String</strong><br />" +
    Request.QueryString.ToString() + "<br />";
```

如果使用了查询字符串中的值，则其信息可帮助调试问题。可以扩充 `Application_Error` 中的代码，添加其他一些有用的信息，如 `cookie` 和表单集合。要了解有关访问这类集合的更多信息，可阅读清华大学出版社引进并出版的《ASP.NET 4 高级编程——涵盖 C# 和 VB.NET(第 7 版)》一书(ISBN: 978-7-302-23524-8)。或者查看 <http://code.google.com/p/elmah/> 上的 ELMAH(Error Logging Modules and Handlers)项目。这是由 Atif Aziz 管理的一个开源项目，旨在捕获和记录异常。ELMAH 项目的优点在于它可以很容易地集成到站点中(不需要进行编程，只需要向 `web.config` 文件中添加几行配置代码)。在过去的几年中，我在自己的大部分 Web 项目中都使用了这个项目，而它也帮助我找到了使用其他方法很容易忽视的许多 bug。

尽管在运行时处理和记录异常很有用，但更好的做法是事先预防它们发生。要编写可靠的代码，使 bug 尽可能地少，需要好的工具帮助了解代码的执行，然后才能调试。VWD 带来了优秀的调试工具，可在该过程中提供帮助。在下一节中将介绍这些工具及它们的使用方法。

18.2 调试基础知识

调试是一个查找并修复代码中的 bug 的过程。尽管这听起来很简单，但通常并非如此。有些 bug 很明显，易于发现并修复；而有些 bug 则很难发现，这就要求理解程序的执行环境。随 Visual Web Developer 提供的调试工具通过提供对程序或 Web 页面的内部工作方式的直接访问，帮助您了解这一执行环境。

VWD 的调试功能简单易用。在调试时，您可以在代码中移动，检查变量、查看对象、试验

方法，甚至执行新代码。为了让 VWD 知道在何处停止，需要在代码中设置一个或多个断点 (breakpoint)。当断点下的代码准备执行时，VWD 会停止应用程序(通常是 Web 页面、用户控件或 App_Code 文件夹中的代码)的执行，然后将焦点返回到 VWD 中，这样就可以诊断代码及其环境。

按 F9 键可在想让其停止执行的代码行上设置断点。除了 F9 快捷键之外，还可以单击代码的页边空白，在那里会出现一个大的圆点，如图 18-6 所示；或者可从主菜单中选择 Debug | Toggle Breakpoint 选项。再次按 F9 键、单击页边空白中的相同圆点或者从菜单中进行选择可以切换断点。而要清除整个 Web 站点中的所有断点，可按 Ctrl+Shift+F9 组合键。

为了让您了解调试是如何工作的，以及它可以做什么，接下来的“试一试”练习显示了调试的基本操作。本章后面的部分将详细介绍大量随 VWD 提供的调试工具和窗口。

试一试 调试应用程序

在这个“试一试”练习中，将调试前面章节中创建的 Calculator 页面。如果没有此文件，可参阅第 5 章，该章介绍了如何创建该文件；或是从 <http://www.wrox.com> 网站上下载本章的代码。本章的调试练习假定您使用 IE 浏览器。如果使用的是不同的默认浏览器，如 Firefox 或 Opera，则调试过程将大致相同，不过，您会发现在通过断点进入代码时，VWD 并不总是自动获得焦点。

(1) 打开 Demos 文件夹中的 CalculatorDemo.aspx 页面并切换至 Code Behind 模式。

(2) 单击 CalculateButton_Click 事件处理程序中检查两个 TextBox 控件中的文本长度的第一行代码，然后按 F9 键设置断点。该行就会突出显示，如图 18-6 所示，并且在文档窗口的页边空白处出现一个彩色圆点。

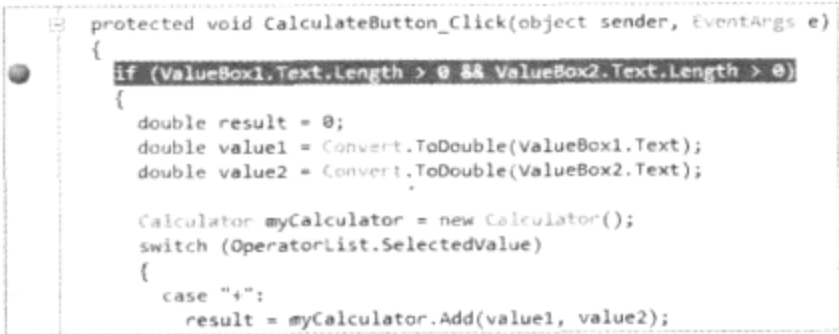


图 18-6

(3) 按 F5 键(而不是一直使用的 Ctrl+F5 组合键)在浏览器中打开该 Web 站点并启动调试过程，也可以从主菜单中选择 Debug | Start Debugging 选项。如果出现如图 18-7 所示的对话框，则单击 OK 按钮使 VWD 修改 web.config 文件。

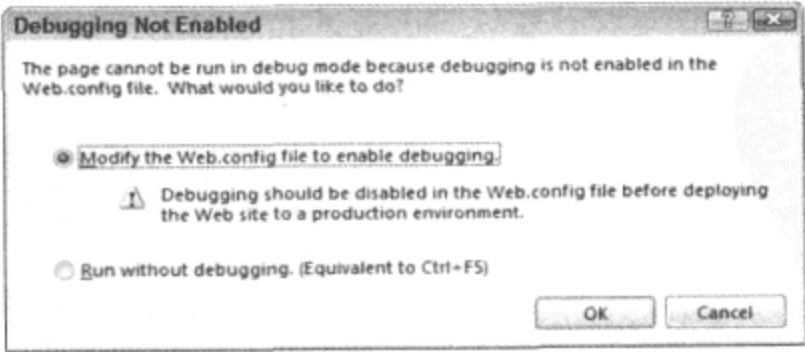


图 18-7

根据浏览器的设置，可能会出现如图 18-8 所示的对话框。

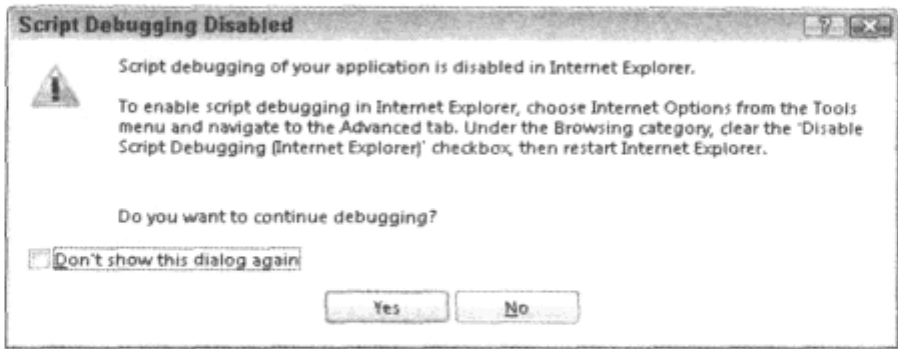



图 18-8

如果运行的是 Internet Explorer 7 或更早版本，而且没有将 IE 设置为允许调试客户端脚本(本章后面将讨论相关内容)，就会出现图 18-8 所示的对话框。如果出现该对话框，则按照其指示操作，单击 Yes 按钮返回 VWD。如果根据指示对 Internet Explorer 作了配置，就不会再出现该对话框。如果根本没有出现该对话框也不用担心；只有需要对设置作改变时，它才会出现。

(4) 页面正常加载，显示两个 TextBox 控件：DropDownList 和 Button。



常见错误：如果得到一个表明页面标题无效的错误的，则需要关闭浏览器，返回 VWD，给页面提供一个标题，然后保存改动并再次按下 F5 键。

在第一个文本框中输入 5，在第二个文本框中输入 7，然后单击 Calculate 按钮。您将被带回到 VWD，而不是在浏览器中获得答案。如果没有直接回到 VWD 中，则需要手动切换。注意，VWD 的任务栏图标通过闪烁来引起注意。

(5) 在 VWD 中，带有断点的行现在以黄色突出显示。另外，您会看到在文档页边空白中有一个黄色图标，表明该行代码即将被执行。不过在这之前，可以查看一下控件、变量和其他构成执行环境的元素。要查看在 TextBox 控件中输入的值，可以将鼠标指针悬停到突出显示行中的 Text 属性上。将会看到一个显示输入的值的小型工具提示，如图 18-9 所示。

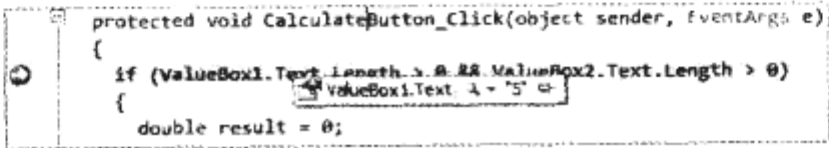


图 18-9

(6) 将鼠标指针悬停到代码中一些变量上，如 result 和 value1。注意，这时没有工具提示，因为代码还没有到达声明变量的地方。所以在调试器看来，它们并不存在。

(7) 要在代码中前进，可按 F10 键。这就会跳过选择的行并执行它。继续按 F10 键，直到声明 value2 变量的行突出显示。现在再将鼠标指针悬停到 value1 上，就会出现工具提示，表明 value1 包含值 5.0。

(8) 将鼠标指针悬停到 Select Case(在 VB.NET 中)或 switch 语句(在 C#中)中的 SelectedValue 属性上。注意，工具提示显示了在下拉列表中选择的值(加号)。即使这行代码还未执行，SelectedValue 已在前面被指派了一个值，所以在这儿可看到它。

(9) 再按 F10 键几次，直到到达指派值给 ResultLabel 控件的行。将鼠标指针悬停到 result 变量

上(如果使用的是 VB.NET，需要首先用鼠标突出显示 `result` 变量)，可注意到它显示数字 12.0(计算的结果)，如图 18-10 所示。

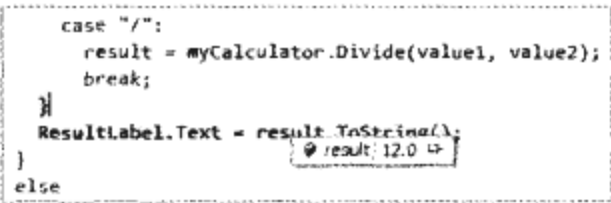


图 18-10

(10) 最后，按 F5 键。通过按该键，代码会继续前行，直到发现下一个断点。由于在代码中还没有定义另一断点，Click 事件处理程序中的其余代码就会被执行，且结果将显示在页面中。

工作原理

到本章之前，无论何时按 Ctrl+F5 组合键在浏览器中查看页面，都不会发生特殊的事情。VWD 只是打开浏览器，然后浏览器向内置的 Web 服务器请求页面。不过，如果按 F5 键，VWD 将进入调试模式，并会注意在代码中设置的断点。一旦命中断点，代码的执行就会停止，然后您就可以查看代码和其执行环境，通过此环境可访问变量、控件、方法和更多内容。注意，带有断点的行上的代码此时并不执行。需要使用 F10、F11 或 F5 键跳过它来执行。

在可以调试代码之前，需要配置应用程序支持它。方法就是设置 web.config 文件中 compilation 元素的 debug 特性为 true:

```
<compilation debug="true">
```

如果使用的是 Visual Basic，可看到这一元素的另外两个特性：strict 和 explicit。默认情况下，strict 被设置为 false，这意味着 Visual Basic 将作隐式类型转换。explicit 特性被设置为 true，这意味着在使用前需要声明所有变量。对于大部分场景，这些默认设置工作良好。

无论何时启动调试，而<compilation />中的 debug 特性设置为 false，都会得到如图 18-7 所示的对话框，通过它可启用该功能。为了避免此设置带来的开销，通常在生产服务器中将它设置为 false。

在这个练习中，还学习了如何使用数据提示(data tip)，这是当将鼠标指针悬停在选择的变量上时出现的小型工具提示窗口。对于简单的类型，如 Integer 或 String，所看到的是变量的值。对于复杂的类型，例如 LINQ 查询返回的结果，则会得到一个更丰富的数据提示，其中提供了更详细的信息。

调试时的数据提示功能很有用，但这只是一小部分的调试功能。在下一节中，将概述随 VWD 提供的所有调试工具。

18.3 调试的工具支持

通过大量快捷键和菜单项，VWD 使得您可以在调试的代码中到处移动，提供了逐行或逐块执行代码的选项。另外，IDE 提供了大量允许诊断和改变执行环境(包括运行时的变量值)的窗口。下面首先将介绍如何在代码中移动，然后讨论大量的调试窗口。

18.3.1 在调试代码中移动

当代码在某个断点处停止时，可使用大量键盘快捷键来决定接下来的工作。表 18-1 列出了最常用的快捷键。

表 18-1

快 捷 键	说 明
F5	如前面的“试一试”练习中所演示的那样，按此键启动调试。而在调试时按此键，代码将继续执行直到命中下一个断点，或是所有代码完成执行
F11	按此键执行当前行并(可能的话)单步进入被调用的方法。后面将看到其运用
F10	按此键执行当前行，而不单步进入被调用的代码，除非代码本身包含断点
Shift+F11	按此组合键执行当前代码块中的代码并返回到最初调用它的代码
Shift+F5	按此组合键停止调试。这会关闭浏览器并返回到 VWD
Ctrl+Shift+F5	按此组合键重启调试过程

除了这些键盘快捷键外，还可以使用如图 18-11 所示的 Debugging 工具栏，它提供了类似的功能。

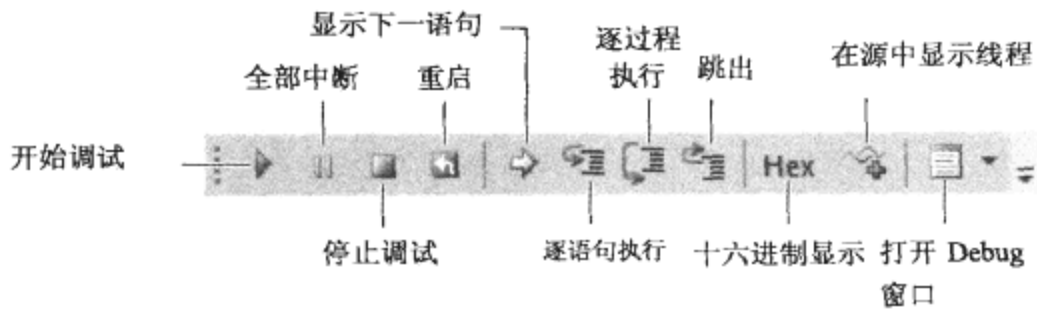


图 18-11

在开始调试时，该工具栏会自动出现，如果未出现，可右击已有工具栏并选择 Debug 选项。在 VWD 中调试代码时，还有大量的调试窗口可使用，下一节将详细介绍。

18.3.2 调试窗口

通过这些调试窗口，可查看应用程序中的变量，甚至在运行时改变它们。另外，它们提供了您在应用程序中所处位置和前面执行了什么代码的信息。所有这些信息可帮助你了解应用程序的执行流程。

可通过 Debug | Windows 菜单项访问所有的调试窗口。它们在 Visual Web Developer 的 Express Edition 中并不都是可用的。而且，要访问大部分的窗口，应用程序需首先处于调试模式。下一节将介绍可用的不同窗口。而在接着的“试一试”练习中，您将有机会使用它们，从而了解它们的使用方法和用途。

1. 查看变量

知道变量的值对于了解应用程序很关键。在这方面，VWD 提供了 3 个调试窗口，它们都提供了有用的信息。所有这些窗口都支持在运行时改变变量的值，允许使用数据提示进一步发掘对象，

并且允许复制和粘贴数据，从而可在别处重用。

Watch 窗口

这可能是需要注意的最重要的窗口。它允许查看所有变量并发掘它们。图 18-12 显示了当前正查看 Calculator 页面中使用的 value1 变量的 Watch 窗口，而 value2 变量正被添加到列表中。

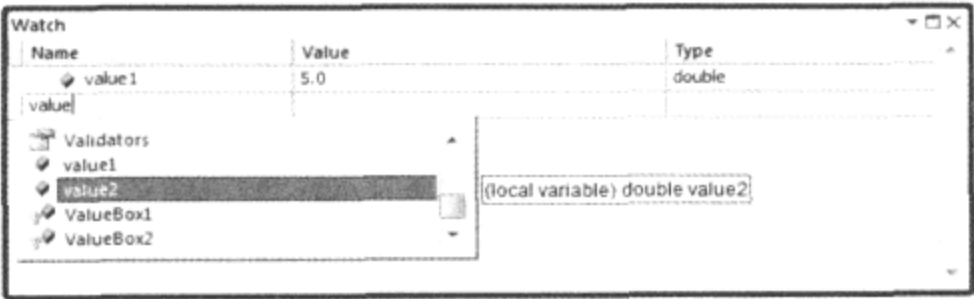


图 18-12

有一些不同的方法可以将变量添加到 Watch 窗口中。首先，可单击 Watch 窗口一次，然后开始输入一个变量名。接着可按 Ctrl+空格键打开 IntelliSense 列表，轻松完成变量名。另外，可右击文档窗口中的变量，选择 Expression: 'VariableName' | Add Watch 命令。根据所使用 Visual Web Developer 的版本，Add Watch 菜单选项可能直接出现在文档窗口的右击菜单中。最后，可选中文档窗口中的变量，并将它拖至 Watch 窗口中。

当变量在 Watch 窗口中时，可以改变它们的值来影响代码的执行。例如，可将 value1 变量的值改成一个不同的数字，来改变计算结果。要改变一个值，可在 Watch 窗口的 Value 列上双击它。或者，右击查看的变量并选择 Edit Value 选项。

除了可显示变量的值外，还可以使用 Watch 窗口执行代码。例如，可在 value1 变量上调用 ToString() 方法查看其字符串表示。方法就是双击 Watch 窗口中的变量名，使它变得可编辑，然后添加.ToString()，如图 18-13 所示。

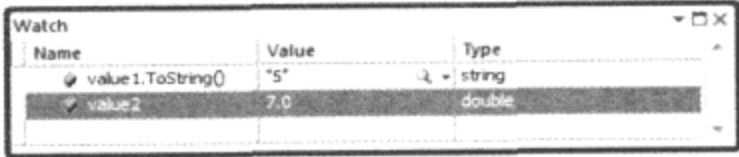


图 18-13

在 Watch 窗口中并不只限于调用 ToString 方法。任何有结果的表达式都可以在这里执行。不过，后面讨论的 Immediate 窗口更适用于即时执行代码。

除了 Watch 窗口外，Autos 和 Locals 窗口都是可用的。它们的工作方式类似于 Watch 窗口。

Autos 窗口

Autos 窗口只在 Visual Studio 的商业版中可用，而在 Express Edition 中不可用。由于它类似于 Watch 窗口，所以问题不大。Autos 窗口显示了当前和前面的代码语句中使用的变量，并在单步执行代码时自动更新。

Locals 窗口

Locals 窗口类似于 Watch 和 Autos 窗口，但它显示了当前执行的代码作用域中的(所能到达的)所有变量。这是个有用的窗口，因为它显示了所有相关的变量而不要求手动添加。

2. 其他窗口

除了查看变量的窗口，VWD 还有其他一些有用的窗口。

Breakpoints 窗口

Breakpoints 窗口提供了在整个 Web 站点的代码中设置的所有断点的一个概览。但是，这一窗口在 Express Edition 中不可用，因此需通过查看实际代码手动查找断点。

Call Stack 窗口

Call Stack 窗口提供了有关代码执行和调用的顺序的信息。从一段代码到另一段代码的每个调用都放置在一个可导航的调用堆栈中。初看起来，它有些难以理解，但在了解了其工作原理后，可轻松地在代码中导航。图 18-14 显示了 Calculator 类的 Add 方法内部的 Call Stack 窗口。

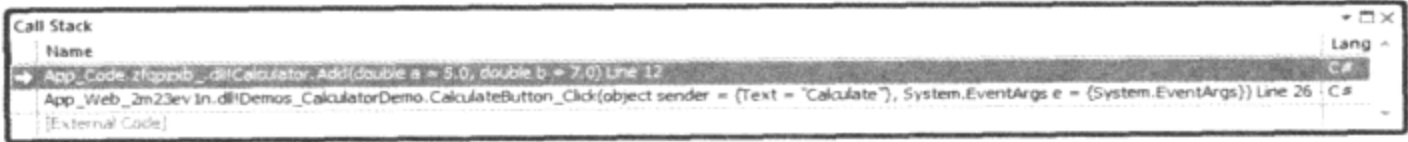


图 18-14

在突出显示的行中，可看到 Calculator.Add 是当前活动的代码。在其之下，可看到 CalculateButton_Click，这是 Calculator 页面中调用 Add 方法的事件处理程序。双击 Call Stack 窗口中的一行将带您带至适当的代码位置。

Immediate 窗口

最后一个有趣的调试窗口要属 Immediate 窗口。该窗口可用于执行代码，就好像它们已经在页面中编写了一样。可使用这个窗口测试表达式，查看函数返回什么结果等。例如，在调试模式下，可在 Immediate 窗口中输入下列命令：

VB.NET

```
? New Calculator().Add(3, 4)
```

C#

```
? new Calculator().Add(3, 4);
```

问号用于输出到 Immediate 窗口。然后代码实例化一个新的 Calculator 实例，并直接传递值 3 和 4 到其 Add 方法。代码将被执行，Add 方法返回将在 Immediate 窗口输出的值 7。

该窗口非常适用于快速测试代码。不需要在页面中编写要测试的代码，而可以直接在 Immediate 窗口中输入它，然后查看其输出。

在接下来的“试一试”练习中，将介绍这些调试窗口的运用。

试一试

广泛的调试

在本练习中，将运用前面讨论的所有调试窗口。由于有大量窗口和选项可用，所以我们不会对此过程中的每一步作详细介绍，但鼓励您去试验。试着添加更多变量到 Watch 窗口，在 Immediate 窗口中输入您自己的代码等。试验是发现 VWD 中大量调试功能的最好方法。

- (1) 再次打开 CalculatorDemo.aspx 页面的 Code Behind 文件，然后按 Ctrl+Shift+F9 组合键清除

在前面设置的所有断点。单击 Yes 按钮确认删除。

(2) 单击声明变量 value1 的行，按 F9 键设置一个断点。这时的文档窗口如图 18-15 所示，显示的是 C#项目的文档窗口。

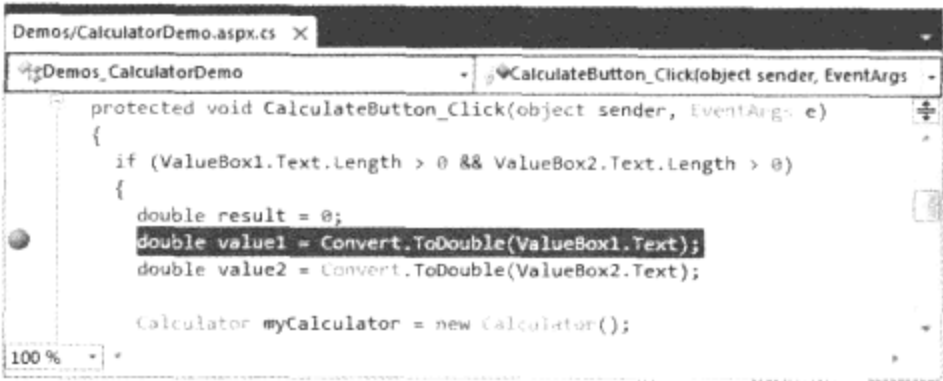


图 18-15

(3) 按 F5 键或从主菜单中选择 Debug | Start Debugging 命令开始调试应用程序。在第一个文本框中输入数字 5，确保在下拉列表中选择了加号，然后在第二个文本框中输入 7。接着单击 Calculate 按钮，VWD 就会在前一步骤设置的断点处中断。如果没有自动回到 VWD，则手动切换。

(4) 将鼠标指针悬停在当前断点下边的代码所使用的 OperatorList 变量上，注意 VWD 显示了一个前面带有加号的数据提示。这意味着可展开该项，获得有关变量的更详细的信息。图 18-16 显示了展开的数据提示。

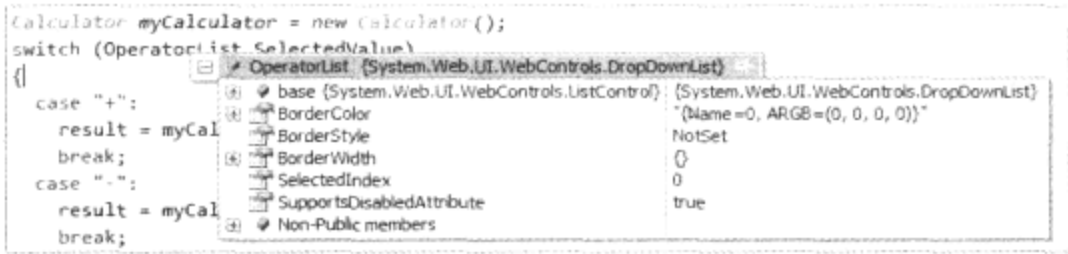


图 18-16

如果使用的是 VB.NET，需要双击文档窗口中的 OperatorList 变量来选择它，然后将鼠标指针悬停在它上面。

注意，可展开其他项(如 DropDownList 控件的基类)来获得其他相关数据。如果使用的是 C#，则将基本项展开。这时，将看到 SelectedValue 这样的属性，如果没有在第(3)步进行修改，则它们应该被设为“+”。在 VB.NET 中，可以直接在主列表中看到这个属性。

(5) 在位于顶部检查 TextBox 控件的文本长度的代码中，右击 ValueBox1，选择 Expression: ‘ValueBox1’ | Add Watch 选项。Add Watch 选项可能再次直接出现在右击菜单中。然后该变量被添加到 Watch 窗口中，在其中可像展开数据提示那样展开它。展开该项，将看到 Text 属性设置为“5”。

(6) 双击 Text 属性的值“5”，将它改为“12”(包括双引号)，然后按 Enter 键。

(7) 打开 Locals 窗口(如果该窗口未显示，可选择 Debug | Windows | Locals 命令)。按 F10 键执行断点下的行。这就获取了 ValueBox1 的值，将它转换为双精度类型，并赋给 value1。查看 Locals 窗口中的 value1 变量(如图 18-17 所示)。可注意到，它现在包含 12.0，这是在前一步中指派给文本框的 Text 属性的值，现在转换为双精度值。

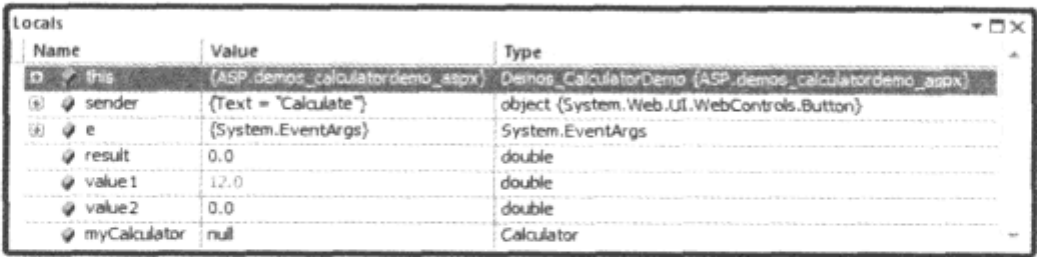


图 18-17

注意，value1 变量的值也改变了颜色。这意味着该项已发生改变。同时可注意到，由于所有这些是在回发时在服务器上发生的，浏览器并未作修改，因此文本框仍显示值 5。只有当页面完成在浏览器上的呈现，才能看到显示新值。

在图 18-17 中，还可看到现在位于作用域中的其他变量，如 result、myCalculator 和包含对当前被执行的页面的引用的 this(Visual Basic 中的 Me)。

(8) 再次按下 F10 键，value2 也得到了更新。此时，value2 变量的值的颜色变为红色，表明值已经发生改变，而 value1 的值则再次变为黑色。这使得很容易看出上一条语句修改了什么变量。

(9) 按住 F10 键，直到到达调用 Calculator 类的 Add 方法的行。不是按 F10 键执行该行，而是按 F11 键。这将逐语句执行 Add 方法，从而可看到它如何执行计算。在 Add 方法中，可以将鼠标指针悬停在该方法的参数上来查看它们的值，如图 18-18 所示。

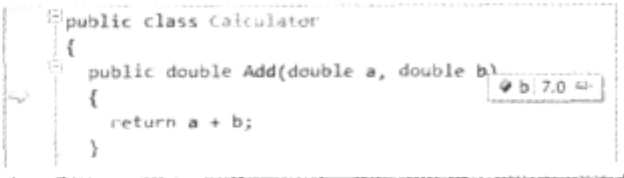


图 18-18

(10) 选择 Debug | Windows | Call Stack 命令打开 Call Stack 窗口(或者按下 Ctrl+Alt+C 组合键)，可注意到 Add 方法被 CalculateButton 的 Click 事件处理程序调用，如图 18-19 所示。

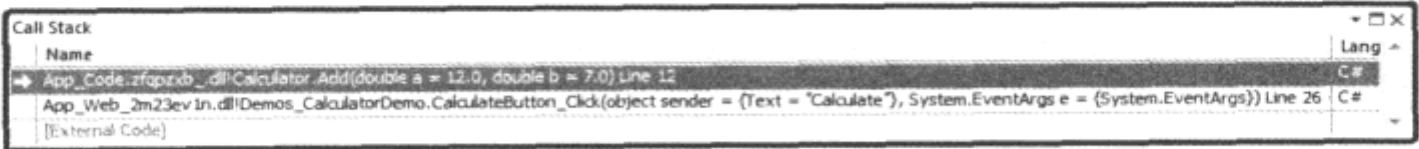


图 18-19

(11) 双击 Call Stack 窗口中的第二行，将被带回到 Calculator 页面。注意，这并不执行任何代码；它所做的只是显示相关代码。双击第一行，将再次被带到 Add 方法的代码。

(12) 按 Shift+F11 组合键跳出 Add 方法，返回到 Calculator 页面中的调用代码。如果再看一下 Call Stack 窗口，会发现 Add 方法的行已从调用堆栈中删除。

(13) 打开 Immediate 窗口(选择 Debug | Windows | Immediate 命令)测试一些代码。在这个窗口中，输入下列代码并按 Enter 键：

```
VB.NET
? New Calculator().Multiply(4, 12)

C#
? new Calculator().Multiply(4, 12);
```

Immediate 窗口显示了计算的结果，如图 18-20 所示。

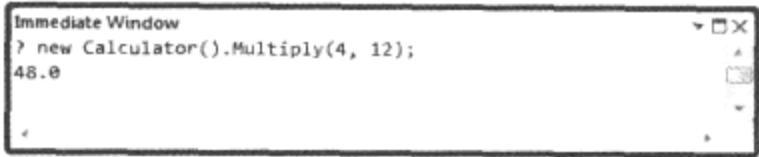


图 18-20

(14) 最后，按 F5 键。这将执行页面中其余的代码。焦点将再次来到浏览器上，在 Label 控件中显示计算结果。如果一切按计划进行，将看到数字 19：第(6)步中为 ValueBox1.Text 输入的值 12 和第(3)步中输入的值 7 的和。

工作原理

正如在前一个“试一试”练习中所演示的，如果代码中放置了断点，一旦带有断点的行要执行，整个程序的执行就会暂停。从那里开始，您可以在代码中移动、检查变量和执行语句。在这个练习中，学习了如何使用 F10 和 F11 快捷键逐语句或逐过程执行代码。如果对被调用的底层代码不感兴趣，可使用 F10 键执行代码；如果想像查看如何使用 Add 方法那样查看执行的代码，则使用 F11 键。

数据提示以及 Watch 和 Locals 窗口都是查看及改变变量和值的非常有用的工具。例如，即使在浏览器的第一个文本框中输入的是 5，也能在调试时将该值改为 12。在调试时作的任何更改都会传播给仍需执行的其余代码。

Immediate 窗口可用于测试小的代码段。这对于尝试一些想法很有用，而无须在代码窗口中编写并调试它。在这个练习中，编写了一些代码来创建一个新的 Calculator 实例，调用 Multiply 方法，并使用问号输出值。

VB.NET

```
? New Calculator().Multiply(4, 12)
```

C#

```
? new Calculator().Multiply(4, 12);
```

除了可以在服务器上调试代码以外，VWD 还支持调试客户端 JavaScript。

18.4 调试客户端脚本

到目前为止，已经使用了调试工具来调试 ASPX 页面和 Code Behind。不过，这并不是调试功能的全部。Visul Studio 2010 现在有了对调试客户端 JavaScript 的强大支持。调试客户端 JavaScript 要求使用 Internet Explorer，而并不适用于其他浏览器，如 Firefox 或 Opera。有关在 VWD 中调试客户端 JavaScript 的极好事情就是您已经知道如何去做。可以使用本章所看到的熟悉的相同工具来调试服务器端和客户端代码。

最终被 Web 浏览器中的页面使用的 JavaScript 有多种不同的来源。可以将 JavaScript 放在外部脚本文件中、嵌在页面或母版页中，甚至服务器控件也可使用它们自己的 JavaScript。这有时使得很难在正确的代码中中断，因为您并不总是知道它的来源。幸运的是，VWD 对此有好的解决方案。它允许在浏览器中显示的最终 HTML 中设置断点。为了显示在什么文件中设置断点或调试什么代码，VWD 会更新 Solution Explorer 并显示一个包含客户端脚本的所有文件的列表。一旦处于调试模

式，就可单步执行这些客户端脚本。可能时，保留调试时在这些文件中设置的断点，使调试平滑进行。

学习这个新的客户端 JavaScript 调试功能最简单的方法是尝试使用它，因此下一“试一试”练习将显示如何调试在第 10 章创建的 Web 服务测试页面。

试一试 在 Internet Explorer 中调试 JavaScript

需要使用 Microsoft Internet Explorer 执行本练习，本“试一试”练习中的大部分功能只适用于该浏览器。如果 Internet Explorer 当前不是 VWD 中的默认浏览器，则右击 Solution Explorer 中的页面，选择 Browse With 命令。从 Browsers 列表中选择 Internet Explorer，并选择 Set as Default 命令。可单击 Cancel 按钮关闭该对话框，而 VWD 仍会记住默认浏览器设置。在完成本练习时，可使用同样的步骤切换回您喜爱的浏览器。

(1) 从 App_Code 文件夹中打开 NameService.cs 或 NameService.vb 文件。定位到 HelloWorld Web 方法，并在返回个性化问候的方法的第一行代码上设置断点。关闭该文件。

(2) 从 Demos 文件夹中打开 WebServices.aspx 页面，切换至 Markup 视图。定位到 HelloWorld JavaScript 方法，单击声明 yourName 变量的行并按 F9 键设置断点，如图 18-21 所示。

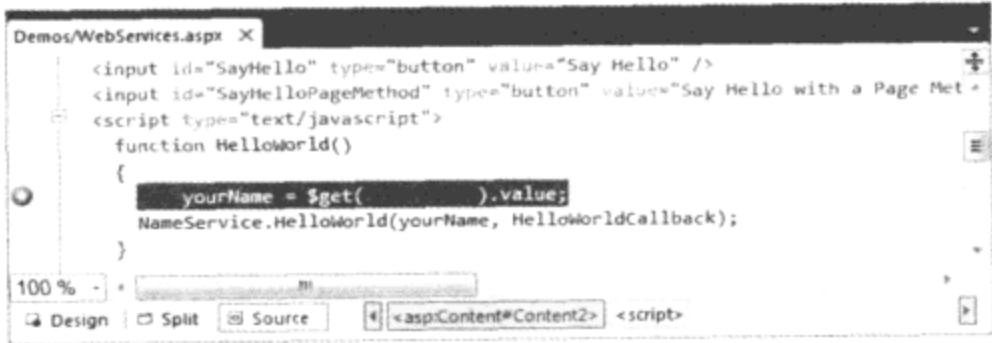



图 18-21

(3) 按 F5 键开始调试。该页面将在浏览器中加载，显示一个文本框和两个按钮。在文本框中输入名字，并单击 Say Hello 按钮。一旦单击它，焦点就回到 Visual Studio，且代码在 JavaScript 代码块中暂停。



常见错误：如果客户端 JavaScript 断点未能命中，则关闭浏览器以停止调试，在设置断点的行前输入 debugger，然后再次按下 F5 键。VWD 并不总是能调试客户端 JavaScript 断点，但是当使用 debugger 关键字的时候总可以正确工作：

```
debugger
var yourName = $get('YourName').value;
NameService.HelloWorld(yourName, HelloWorldCallback);
```

(4) 按 F10 键执行突出显示的行(如果使用了 debugger 关键字，则需要按该键两次)，将输入的名字赋给 yourName 变量。然后将鼠标指针悬停在该变量上，将看到数据提示出现。

(5) 打开其他调试窗口，可注意到它们的行为与之前看到的完全相同。可以添加 JavaScript 变量到 Watch 窗口来查看它们的值，在 Immediate 窗口中输入 JavaScript 进行计算等。还可看到，Solution Explorer 发生了改变，在 Web 项目上方显示了包含脚本的活动的客户端文件(如图 18-22 所示)。



图 18-22

(6) 双击 Solution Explorer 中出现的 Windows Internet Explorer 节点下的这些文档。文件 WebServices.aspx 应该已经在文档窗口中打开了。初看起来，该文件似与前面的相同。不过，仔细查看的话，会发现这不再是混合了 ASP.NET 控件和其他标记的原始源文件，而是在浏览器中呈现的最终 HTML。为了提醒您查看的是最终文件而非原始源文件，VWD 为文档窗口上方的文件选项卡标签添加了文本[dynamic]和一个锁形图标，如图 18-23 所示。



图 18-23

它的真正好处在于即使查看的是一个运行时文件，VWD 也仍会将这个文件与原始源文件相关联。这意味着可以在运行时文件中设置断点，它们将在原始源文件中被记下，并对下次调试会话可用。

(7) 为了了解其工作原理，在 HelloWorldCallback 处理程序中带有 alert 语句的行上设置断点。一旦代码从本练习中调用的 Web 服务中返回，就将再次回到这一处理程序，从而可检查该服务返回的值。

(8) 按 F5 键继续执行代码。在文本框中输入的名字被检索，然后发送给服务的 HelloWorld 方法。由于第(1)步在那里添加了一个断点，代码将再次停止，允许查看传递给这个 Web 方法的变量 yourName。尽管这个练习本身很简单，但在后台却有着大量不可思议的事情发生。您从运行在浏览器上的一些客户端代码逐语句执行到运行在服务器上的 Web 服务中的代码，而这是通过相同的 IDE 完成的。

(9) 再次按 F5 键，您将从服务器端 Web 服务被带回到客户端代码，在那里可看到 HelloWorldCallback 处理程序中的 Web 服务的结果。

(10) 再按 F5 键。代码将完成执行并显示一个 JavaScript 通知窗口，它带有包含您名字的问候语，

这和第 10 章中是一样的。

(11) 关闭浏览器，再次回到 VWD，通过双击 Solution Explorer 中的 WebServices.aspx 文件来打开它。这就打开了原始源文件，而不是在本练习第(6)步中看到的动态版本。定位到 WebServices.aspx 文件中的 HelloWorldCallback 处理程序。注意，第(7)步设置的断点被保存了。

工作原理

本练习中有一些需要注意的地方。首先，应该了解动态文件(或运行时文件)的概念。这些文件是 ASPX 页面的最终结果，可让您看到最终在浏览器中呈现的 HTML、CSS 和 JavaScript。这是很有帮助的，因为这样可看到所有相关内容。记住，显示在浏览器中的最终标记来自于各种源，包括母版页、内容页、外部 CSS 和 JavaScript 文件，以及页面上的各种服务器控件。从一个集中的地方看到组合结果的能力使得很容易看出所有事物是如何结合在一起的。

本练习中的另一个重点是 IDE 如何从用户界面中的客户端代码到服务器提供完全集成的调试功能。为了易于设置断点，VWD 没有限制只可以在设计时在页面中添加断点；而是还允许在动态的运行时文件中设置它们。在停止调试时，VWD 跟踪新的断点，找到它们所属的源文件，然后再次在那里添加它们，以便对下次调试会话可用。

由于一切都在同一台计算机上、同一个 IDE 中发生，您可能没有意识到这样调试时跨越了多个边界。首先，VWD 支持调试浏览器中的客户端脚本，因此您可以挂钩到(hook into)它，甚至于在任何数据发送到服务器之前。当您在第(8)步按 F5 键时，代码继续执行并发送值给服务器，在那里它用于 NameService 的 Helloworld 方法。一旦服务器端 Web 方法完成，执行就返回到客户端，允许在显示 Web 服务的消息的 alert 语句上中断。

由于某些原因，在 VWD 2010 中调试客户端 JavaScript 并不总是可行的。如果您遇到了问题，记住可以使用 debugger 关键字。把该关键字添加到想要设置断点的行前，VWD 将在遇到该关键字的时候停止执行。不要忘了在完成调试以后删除 debugger 关键字。

尽管调试在站点的开发中很有用，但在生产环境中运行时，它缺乏检查站点行为的能力。幸运的是，ASP.NET 对此也有解决方案：跟踪。

18.5 跟踪 ASP.NET Web 页面

没有跟踪，就无法在运行时最有效地找到变量的值、对象、代码采取的执行路径等。您可能会向页面添加一个 Label 控件，然后按如下所示编写一些代码。

VB.NET

```
Dim value2 As Double = Convert.ToDouble(ValueBox2.Text)
DebugLabel.Text &= "The value of value2 = " & value2.ToString() & "<br />"
```

C#

```
double value2 = Convert.ToDouble(ValueBox2.Text);
DebugLabel.Text += "The value of value2 = " + value2.ToString() + "<br />";
```

尽管这可以工作，但相当麻烦。首先，需要编写大量代码使它工作。其次，最终页面上会显示一个丑陋的 Label 控件，并且在完成调试或跟踪后要记得将它删除。最后，在一切准备就绪后，要

删除所有设置该 DebugLabel 标签的代码。通过设置 Label 控件的 Visible 属性为 False，可使问题变得简单，但仍有指派文本给 Label 控件的性能损失。

ASP.NET 中的跟踪功能解决了所有这些问题。它让页面、控件和代码将信息写入一个名为 Trace 的集中位置，然后可将其显示在浏览器中。跟踪是 ASP.NET Framework 内置的功能，这意味着可以在不手动编码的情况下使用它。另外，可以将自己的信息添加到 Trace 中。在下一节中，将介绍如何使用内置的跟踪功能，提供有关页面的大量信息。在后一“试一试”练习中，将介绍如何添加自己的信息到 Trace 中。

18.5.1 使用标准的跟踪功能

不需要做较多工作，就可以获得有关页面执行方式的许多信息。所需做的只是跟踪页面，可以在页面级或应用程序级进行。在页面级启用跟踪功能，可以选择一个或多个想跟踪的特定页面。如果想同时查看多个页面，可使用应用程序级跟踪。例如，可用它发现 Web 站点中较慢的页面。

1. 跟踪单独的页面

在一个页面中启用跟踪功能，需要设置 Page 指令中的 Trace 特性为 True。

```
<%@ Page ..... Trace="True" %>
```

当运行启用跟踪功能的页面时，会在页面底部得到一个很长的详细信息列表。图 18-24 显示了本章使用的 Calculator 演示页面的 ASP.NET Trace。

该 Trace 提供了有关当前页面的大量信息。在顶部，可看到请求详细信息的汇总，包括当前日期和时间、用于检索该页面的方法(GET 或 POST)和状态码(图 18-24 中的 200 状态码表示成功)。

在其下面是 Trace Information 部分。如果启用了它，ASP.NET Page 类就向该 Trace 进行写入。这类似于第 15 章的演示页面，它将页面生命周期中触发的许多事件写入 Label 控件。

默认情况下，数据是按时间排序的，使事件按发生的顺序进行排列。通过将 TraceMode 由 SortByTime 改为 SortByCategory，也可以将它们按类别排序(在添加您自己的信息到 Trace 的一节中将介绍有关于类别的更多内容)。

```
<%@ Page ..... Trace="true" TraceMode="SortByCategory" %>
```

再往下就是控件树(control tree)，它表示了页面中控件的层次结构视图和它们的大小。但是在图 18-24 中无法看到控件树。

在控件树之下，可看到大量重要的集合的详细信息，包括 QueryString、Cookies、Form、Headers 和 Server Variables。另外，可看到存储在 Session 或 Application 状态中的信息。查看这些集合非常有助于发现问题。例如，有一个要从 cookie 中读取数据的页面，但在该页面加载时出现崩溃，那就可能查看 Cookies 集合，以了解页面是否接收了期望的数据。这些集合是了解页面执行的非常有用的工具，可真正帮助发现和修复代码中的 bug。

页面级的跟踪意味着需要在每个想跟踪的页面上启用跟踪功能，也意味着在完成之后需要在每个页面上禁用它。由于这对于较大的站点会很麻烦，所以 ASP.NET 也允许跟踪整个应用程序。

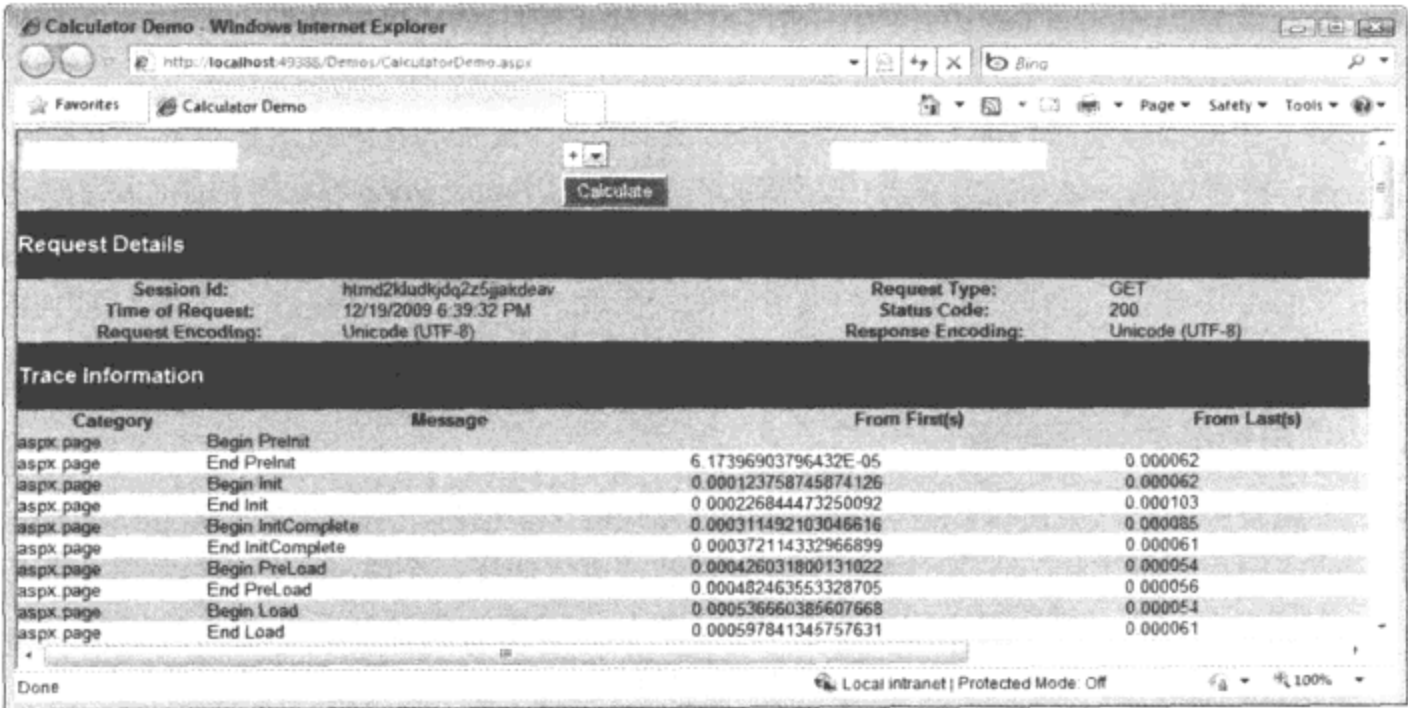


图 18-24

2. 跟踪整个 Web 站点

通过更改 web.config 文件中的跟踪设置可对整个 Web 站点启用跟踪功能。方法就是在 <system.web>下创建一个<trace />元素。表 18-2 列出了<trace />元素使用的最重要的特性。

表 18-2

特 性	说 明
enabled	该特性决定是否对应用程序启用跟踪功能。默认情况下，跟踪是禁用的，所以需要显式设置它为 True
traceMode	该特性决定项在 Trace 输出中的排序顺序。它的工作方式与 Page 指令的 TraceMode 特性相同
requestLimit	该特性决定 ASP.NET 支持的跟踪请求数
pageOutput	该特性指定是否在页面上显示跟踪信息。如果设置为 False(默认的)，那就只可以使用后面讨论的 Trace.axd 访问跟踪信息
localOnly	该特性指定是否只可从本地主机访问特殊的 Trace.axd 处理程序。从安全角度来看，最好是将它设置为 True，这意味着 Trace 对于外部用户来说是不可用的
mostRecent	该特性决定在跟踪请求数达到 requestLimit 时是否丢弃旧的跟踪记录。如果设置为 False，在到达 requestLimit 时自动禁用跟踪功能

如果启用了跟踪功能，就有两种方法读取跟踪信息。如果设置 pageOutput 为 True，则跟踪信息就被添加到每个页面，类似于页面级跟踪。

不过，为了使跟踪不是很显眼，可以禁用 pageOutput，然后使用名为 Trace.axd 的特殊文件请求跟踪信息。这是个虚拟文件，这意味着在 Web 站点中找不到它。但是，ASP.NET 运行库知道在请求这个特殊页面时，它应提供跟踪信息。虽然这个文件是虚拟的，但是仍然可以像对其他文件和文件夹那样，在主 web.config 文件中添加一个<location />元素，通过使用 ASP.NET 的 URL 安全机制保护它。

在下面的“试一试”练习中将了解其工作原理。

试一试

对整个站点启用跟踪

在这个“试一试”练习中，将介绍如何启用站点范围的跟踪。首先，对 web.config 文件作些更改。然后浏览站点，用页面请求填充跟踪日志。最后，请求特殊的 Trace.axd 页面来查看可用的 Trace 日志信息。

(1) 打开 web.config 文件并定位到<system.web>起始标记。添加下列配置信息作为该元素的直接子元素以启用跟踪：

```
<system.web>
  <trace mostRecent="true" enabled="true" requestLimit="100" pageOutput="false"
    localOnly="true" />
```

这就启用了跟踪，但没有将其输出添加到页面，而需要请求特殊的 Trace.axd 页面来查看跟踪信息。另外，通过只允许本地计算机请求跟踪信息，可以使系统更安全。保存并关闭 web.config。

- (2) 右击 Solution Explorer 中的 Default.aspx 并选择 View in Browser 命令。
- (3) 在站点上单击、打开页面、改变主题、填写联系表单等。
- (4) 在请求至少 5 个页面后，改变浏览器的地址栏为如下地址，请求特殊的 Trace.axd 页面：

```
http://localhost:49394/Trace.axd
```

您的端口号可能不同，但重要的是在本地主机上请求 Trace.axd 页面。注意，这个地址假定了您已经像第 7 章所介绍的那样，使用或作为 Virtual Path(虚拟路径)配置了站点。如果还没有这样做，可从 Solution Explorer 中选择 Web 站点，打开其 Properties 面板，设置 Virtual Path 为/。然后从第(2)步开始重新操作，将出现类似于图 18-25 所示的页面。如果显示的是空页面，则按下 Ctrl+F5 组合键刷新它。

(5) 跟踪列表按从最旧到最新的时间顺序排序。单击图 18-25 中某一 ASPX 页面的 View Details 链接，将得到类似于图 18-24 所示的页面。

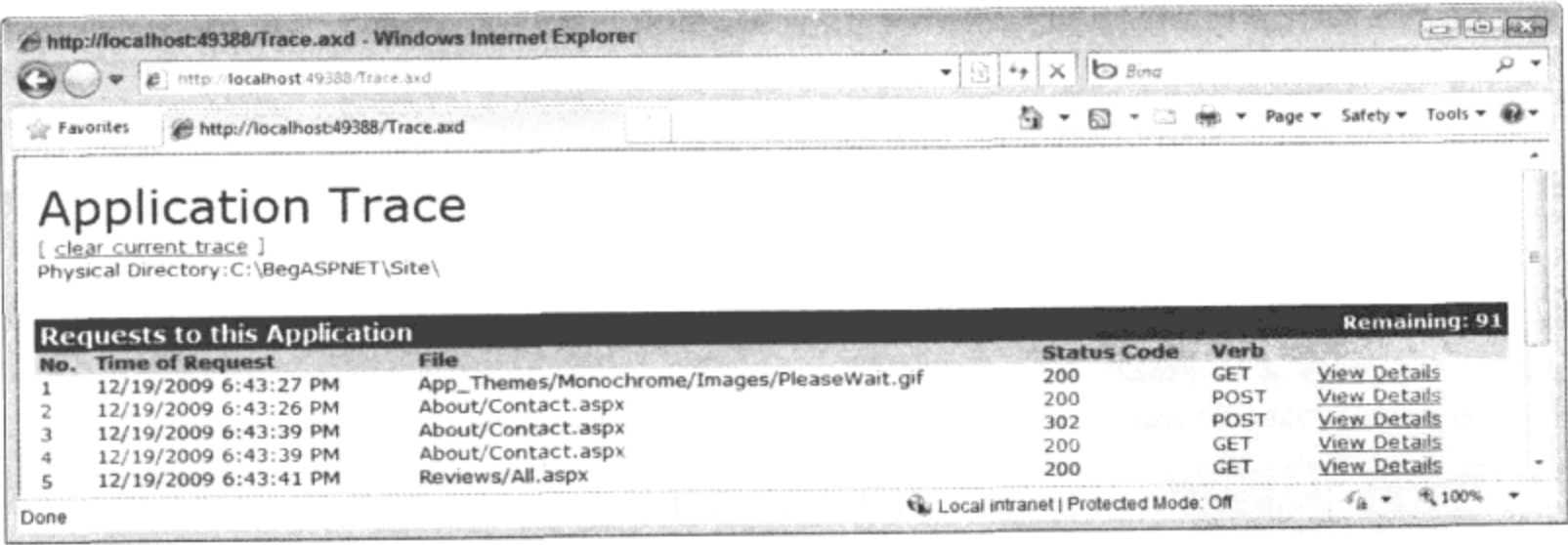


图 18-25

工作原理

能够查看被请求页面的跟踪信息是非常有用的。这一信息可帮助理解进出 Web 页面的信息流。例如，Contact.aspx 页面的跟踪信息也显示了用户在页面上的文本框控件中输入的信息。要查看该信息，可单击想查看的项的 View Details 链接。

尽管 ASP.NET 自动跟踪的信息很有用，但您也没有局限于使用这些信息，而是可以向 Trace 日志中添加您自己的信息。

18.5.2 添加自己的信息到 Trace 中

如果想查看变量的值，或是想了解特定事件是否触发，以及触发的确切时间，将自己的数据添加到跟踪中就很有用。

可以使用 Trace 类向 Trace 中添加信息。这个类提供了两个有用的方法：Write 和 Warn。这两者做几乎相同的事情：它们添加信息到 Trace，添加时也可以将信息放入您自己设置的类别中。唯一的不同点是 Warn 方法写的消息以红色显示。对于意料之外的情况可使用 Warn 方法，因为其消息将引起更多的注意。

在下面的“试一试”练习中，将显示如何使用 Warn 和 Write 方法简单地将自己的信息添加到 Trace 中。

试一试

添加页面的跟踪数据

在这个“试一试”练习中，将添加一些自定义的信息到 ASP.NET Trace 中。将使用 Write 方法写出一个普通页面执行中的跟踪信息，并将 Warn 方法用于意外的情况。

- (1) 打开 CalculatorDemo.aspx，切换至其 Code Behind 文件，定位到 Calculate 按钮的 Click 处理程序。
- (2) 在 Select Case(VB.NET)或 switch 语句(C#)前，添加下面的 Trace.Write 调用：

VB.NET

```
Trace.Write(String.Format("Performing the calculation with the {0} operator",
    OperatorList.SelectedValue))
Select Case OperatorList.SelectedValue
```

C#

```
Trace.Write(string.Format("Performing the calculation with the {0} operator",
    OperatorList.SelectedValue));
switch (OperatorList.SelectedValue)
```

- (3) 在该事件处理程序的底部，修改用于检查的 Else 语句，确保两个 TextBox 控件都包含一个值：

VB.NET

```
Else
    Result.Text = String.Empty
    Trace.Warn("Custom Category",
        "TextBox controls are empty; time to add Validation controls?")
End If
```



```
C#
else
{
    Result.Text = string.Empty;
    Trace.Warn("Custom Category",
        "TextBox controls are empty; time to add Validation controls?");
}
```

(4) 显式地对该页面启用跟踪功能。方法是在 Markup 视图中设置 Page 指令的 Trace 特性：

```
<%@ Page Title="Calculator Demo" ... Trace="true" %>
```

(5) 保存所有更改并按 Ctrl+F5 组合键在浏览器中请求 Calculator 页面。输入两个数字并单击 Calculate 按钮。注意，您的自定义信息已添加到 Trace 中，在 Begin RaisePostBackEvent 和 End RaisePostBackEvent 跟踪项之间。还要注意到，Category 不见了。

(6) 清除浏览器中两个 TextBox 控件的文本，再次单击 Calculate 按钮。这时跟踪信息将更容易被发现，因为其有着不同的颜色和自己的类别名，如图 18-26 所示。

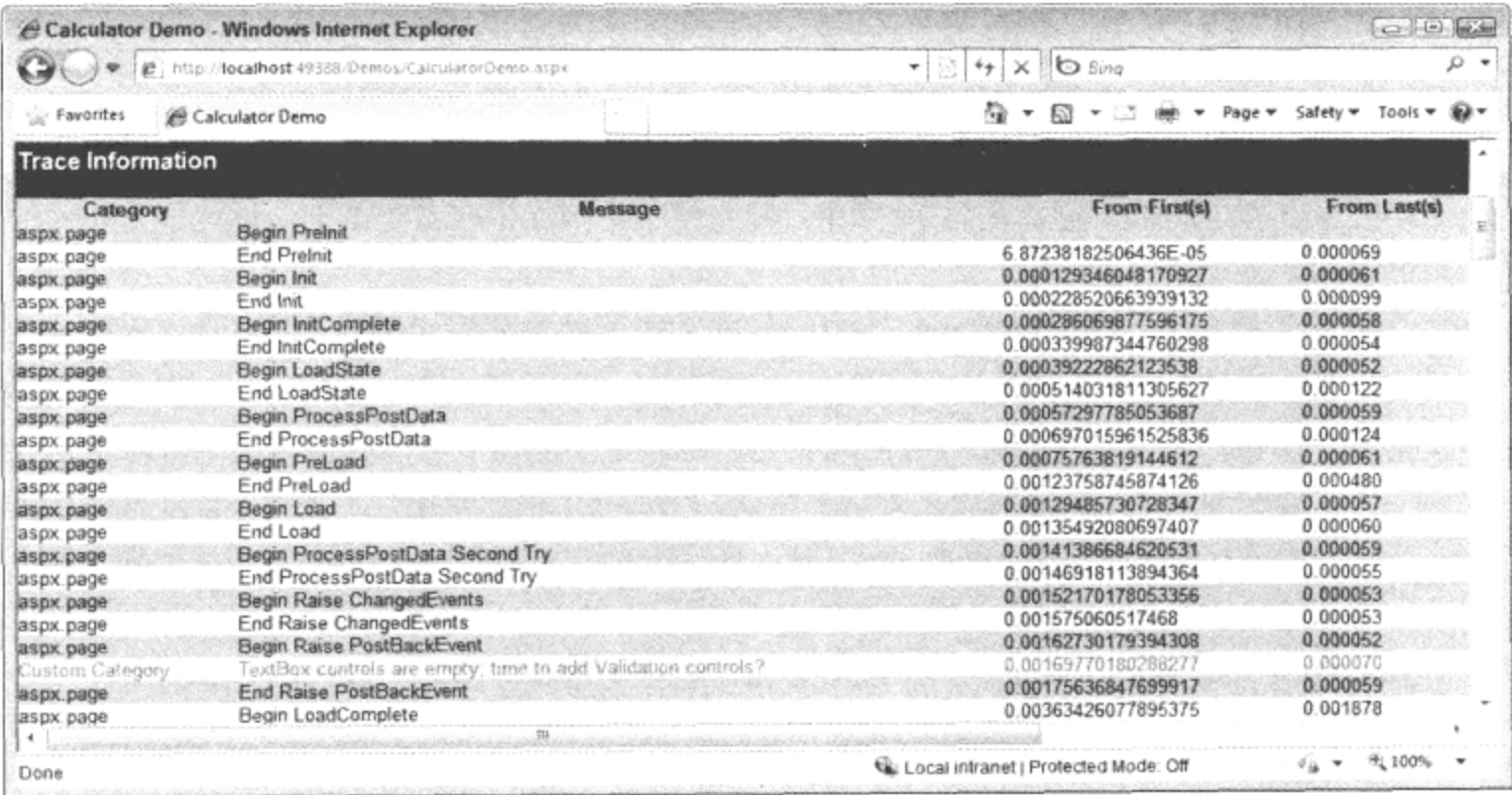


图 18-26

(7) 回到 VWD 并禁用跟踪功能。您需要在页面级(删除 Calculator 页面的 Page 指令的 Trace 特性)和站点级(通过将 web.config 中的 enabled 特性设为 false)上作修改。保存所有更改并再次请求 Calculator 页面。可注意到，该页面仍能正确工作，但不再输出跟踪信息。

工作原理

Trace 类的 Write 和 Warn 方法允许向 Trace 写入额外信息。ASP.NET 运行库跟踪该信息，直接在启用了页面级跟踪的浏览器的页面的底部或是通过特殊的 Trace.axd 页面中与其余 Trace 信息一起显示它。

Write 和 Warn 方法都有 3 个重载。第一个(在前一个示例中，只有 Write 显示)接受显示在 Message 列中的单个字符串。第二个重载也接受一个类别名，如 Warn 方法演示的那样。最后一个重载(前一

个“试一试”练习中未显示)，也接受一个 `Exception` 对象，其消息将添加到 `Trace` 输出中。这对于在 `Catch` 块中跟踪异常信息很有用。

18.5.3 跟踪和性能

尽管看上去，在生产系统的代码中使用 `Warn` 和 `Write` 语句可能会损害性能，但实际并非如此。由于在 `web.config` 文件中通过设置 `trace` 元素的 `enabled` 属性为 `false` 可禁用跟踪功能，因此可以有效地消除其性能开销。

18.5.4 安全警告

跟踪很有用，但如果在生产环境中留下跟踪信息，则会导致信息泄露。因此，应通过在 `web.config` 文件中设置其 `enabled` 特性为 `false` 来禁用它，或只设置 `localOnly` 为 `true`。在第 19 章，您将学到一个对生产服务器上所有站点作这一更改的技巧，从而使阻止对跟踪功能的访问变得很简单。

18.6 有关调试的实用提示

下面列出了一些帮助调试应用程序的实用提示：

- 在 `web.config` 文件中对于生产环境中的站点不要作 `debug="true"` 这样的设置。总是将它设置为 `false`，以便提高性能。在第 19 章中，将介绍一个确保生产服务器上该设置不为 `true` 的更好的解决方案。
- 尽量避免抑制(`swallowing`)`Catch` 块中的异常。您可能想要将代码包装到一个 `Try/Catch` 块中，然后使整个 `Catch` 块为空。尽管这样能够避免异常显示在用户界面中，但它使得调试变得特别困难。由于您不再知道有问题发生，因此也不能在一开始就编写代码来防止错误发生。一般的规则是：捕获可成功处理的错误，例如显示消息给用户。如果不能在代码中处理异常，就使其弹出，并在 `Application_Error` 事件处理程序中记录，这样就可以知道异常发生。
- 如果需要在 `Catch` 块中重新抛出异常，不要使用 `Throw ex`(C#中是 `throw ex`)，而只使用 `Throw`(C#中是 `throw`)。如果使用 `Throw ex`，就很难跟踪异常发生前代码采取的路径。如下面代码所示：

VB.NET

```
Try
    ...
Catch ex As Exception
    ' Do something with the error, such as logging it
    Throw ex      ' Bad example; you lose track of the source of the exception
    Throw         ' Good example; forwards the exception and maintains the call stack
End Try
```

C#

```
try
{
    ...
}
```

```
catch (Exception ex)
{
    // Do something with the error, such as logging it
    throw ex;    // Bad example; you lose track of the source of the exception
    throw;      // Good example; forwards the exception and maintains the call stack
}
```

- 在可能时尽量避免异常处理。正如在本章中所看到的，在一开始就避免异常会更好(速度也会更快)。例如，在执行除法前先检查零值很容易避免 DivideByZeroException 异常。
- 尽可能显式地使用在 Try/Catch 块中捕获的 Exception 类型。尽量避免捕获通用 Exception 类型，为每个预期的异常类型显式地设置多个 Catch 块。

18.7 本章小结

不管您如何仔细地编写程序，站点仍会包含一些 bug 或在运行时抛出异常。为了最小化这些异常并构建一个尽可能平滑运行的站点，需要做一些事情。

首先，可以使用异常处理技术，编写可捕获预期的异常并作适当处理的代码。

为帮助编写代码，使得 bug 尽可能的少，VWD 提供了大量调试工具。中断以进入代码以及分析并改变客户端和服务器的执行环境的能力对于调试过程很有帮助。

但即便彻底地调试了应用程序，您的站点在生产环境中仍可能存在问题，这些问题可能与性能、逻辑错误或其他意料之外的原因有关。在这些情况下，可使用 ASP.NET 的跟踪功能来跟踪运行页面的信息。分析这一跟踪信息有助于修复底层问题。

现在，您的 Web 站点已经完成了，几乎没有错误，下一步就是将它放到网络上。在第 19 章中将介绍如何部署 ASP.NET Web 站点。

18.8 练习

1. 调试和跟踪的区别是什么？

2. 假定您有一些代码可能会抛出异常。例如，您试着向邮件服务器发送一封电子邮件消息，或写文件到磁盘，而不确定自己是否有正确权限。使用什么异常处理策略可避免异常显示在浏览器中？需要什么代码？

3. 您接管了由另一位开发人员构建的 Web 站点，而该开发人员从未听说过异常处理。您的客户抱怨该站点的质量和它出现的大量“Yellow Screen of Death”问题。除了分析整个应用程序的代码外，获取有关这些错误及其发生位置的信息的快速解决方案是什么？如何向站点用户屏蔽异常消息的详细信息？

练习的答案见附录 A。

本章要点回顾

断点	可以在代码中设置的标记，调试器在运行时将在该标记位置暂停执行
数据提示	在调试时，提供变量的简单或丰富数据的工具提示
调试	找到并修复代码中的 bug 的过程
异常	.NET 术语，用来表示代码中可能出现的错误
异常处理	标识和处理运行时发生的错误的方法
Global.asax	一个集中的文件，用来处理应用程序范围内的各种事件，如 Application_Start、Application_Error 等
栈跟踪	当前代码调用栈的视觉表示
跟踪	允许 ASP.NET 控件和自己的自定义代码在运行时向一个集中的日志位置写入信息

本章要点

- 如何通过对代码和配置的简单修改来简化部署过程
- 如何通过使用 Visual Web Developer 的内置复制工具创建副本来准备站点部署
- 如何在目标计算机上安装和配置 Web 服务器以及 Web 站点
- 如何避免部署站点时遇到的常见错误
- 如何将保存在 SQL Server 2008 数据库内的数据复制到目标服务器上

恭喜您！阅读本章很可能意味着您已经拥有了功能齐全、数据库驱动的准备对外发布的 ASP.NET 网站，对于您和您的项目来说这是一个令人激动的时刻。不久以后，您的目标用户就会使用和评价您的应用程序了。

为了让世界范围内的用户能够访问您的网站，需要将它发布到与 Internet 相连的生产服务器上。使用什么类型的服务器以及哪里的服务器，这取决于您自己的需求和预算。可以将站点驻留在具有私有 Internet 连接的家用服务器上(我的站点 <http://imar.spaanjaars.com> 就属于这种情况)，或者可以使用能够直接连接到 Internet 主干的外部(通常是商业)提供商的服务器来驻留它。

不管使用哪种方法，都需要完成一些工作，以便让站点从开发位置(C:\BegASPNET\Site)移动到通过 Internet 可以访问的位置。

本章将讨论一些与成功部署 Web 站点相关的主题。还将介绍从在开发环境中准备站点到在生产服务器上实际运行和测试站点过程。

本章结尾还列出了部署站点时需要注意的一些事项。可以使用这个清单帮助自己确保以最安全和最佳的方法配置生产站点。

19.1 准备部署 Web 站点

当在开发环境中实现 Web 站点的第一个版本时，管理站点及其源代码非常简单。只有站点源代码的一个版本，因此维护非常容易。然而，一旦将站点投入到生产环境中，就拥有站点的两个版本：

一个在生产环境中运行，另一个用于开发。这就很难保持同步。例如，在生产环境中可能使用不同的数据库和连接字符串。也可能为站点发送的电子邮件使用不同的电子邮件地址。最后，在开发环境中可能要禁止发送来自 Global.asax 文件的错误的电子邮件。如果激活站点时在代码中直接进行所有修改，那么在下一次更新过程中很可能会重写某些设置，从而产生不希望看到的结果。

本节将介绍让管理相同 Web 站点的不同版本变得更简单的方法。您已经知道如何将某些硬编码的设置(例如电子邮件地址)移动到 web.config 文件中。在运行时应用程序里的代码会读取这些值。因此开发环境和生产环境之间唯一的不同就在于一个配置文件，它能使得在两种环境中进行不同设置变得很容易。

19.1.1 避免硬编码的设置

到目前为止，所构建的页面和用户控件都使用某些硬编码的设置，如电子邮件地址。例如，ContactForm.ascx(发送电子邮件的用户控件)使用下面的代码来设置接收方和发送方信息：

VB.NET

```
myMessage.From = New MailAddress("you@yourprovider.com", "Sender Name")
myMessage.To.Add(New MailAddress("you@yourprovider.com", "Receiver Name"))
```

C#

```
myMessage.From = New MailAddress("you@yourprovider.com", "Sender Name");
myMessage.To.Add(New MailAddress("you@yourprovider.com", "Receiver Name"));
```

用这种硬编码设置方法很难在不同的环境中给它们提供不同的值。每次将站点发布到生产环境中时，都要确保不会无意间重写为生产环境而修改的设置。

但是，ASP.NET 有一个好方法能够避免这类问题：web.config 文件、表达式语法和用来读取 web.config 文件的 WebConfigurationManager 类。

19.1.2 web.config 文件

在本书中已经多次使用 web.config 文件来存储与连接字符串有关的信息、成员、角色和配置文件信息等。还没有介绍的是<appSettings>元素，它允许使用<add>元素在键/值对中存储数据。<appSettings>元素允许存储简单信息(例如电子邮件地址)并通过键检索它的值。例如，要存储电子邮件地址，可以将下面的代码添加到 web.config 文件中：

```
<appSettings>
  <add key="FromAddress"value="webmaster@planetwrox.com" />
</appSettings>
```

<appSettings>元素放置在 web.config 文件中的<system.web>元素之外，但仍然在父元素<configuration>之内。

很明显，需要了解在运行时访问<appSettings>元素内的数据的方法。有许多方法可以这样做，包括使用表达式语法和使用 Web ConfigurationManager 类，这都会在后面讨论。

19.1.3 表达式语法

表达式语法允许将控件属性绑定到资源中，例如 web.config 文件中的<appSettings>元素、连接字

符串和本地化资源文件中找到的那些资源和 URL 重写时使用的各种路由设置。要显示<appSettings>元素中的数据，可以使用下面的语法，其中 AppSettingKeyName 表示在 web.config 文件中定义的键：

```
<%$ AppSettings:AppSettingKeyName %>
```

例如，要在页面上的 Literal 控件中显示版权通告，可以将下面的设置添加到 web.config 文件中：

```
<add key="Copyright" value="Copyright by Wrox" />
```

然后可以在 Literal 控件中显示文本，如下所示：

```
<asp:Literal ID="Copyright" runat="server" Text="<%$ AppSettings:Copyright %>" />
```

为了让设置像上一个示例中 Text 这样的属性变得更容易，VWD 提供了 Expression Editor。要访问这个对话框，请在 Design 或 Markup 视图窗口中选择一个控件，并打开它的 Properties 面板，然后单击(Expressions)项的省略号按钮，如图 19-1 所示。可以发现在 Markup 视图窗口中，(Expressions)项不一定总是出现。如果是这种情况，则可以首先切换到 Split 视图或 Design 视图窗口。

打开 Expressions 对话框，它允许将控件属性与表达式绑定在一起。VWD 将控件的属性列表限制为可以使用表达式绑定的属性。要将 Literal 控件的 Text 属性绑定到应用程序设置中，首先单击对话框左边的 Text 选项，从右边的 Expression Type 下拉列表中选择 AppSettings 选项，最后从 Expression Properties 部分的下拉列表中选择正确的 AppSetting 选项。图 19-2 显示 Literal 控件的完整 Expressions 对话框，该控件用来显示版权文本。

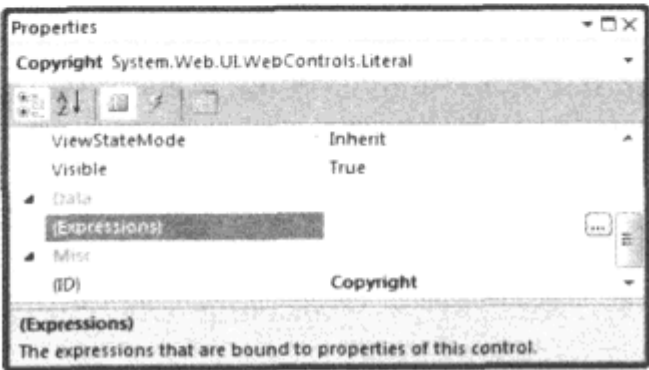


图 19-1

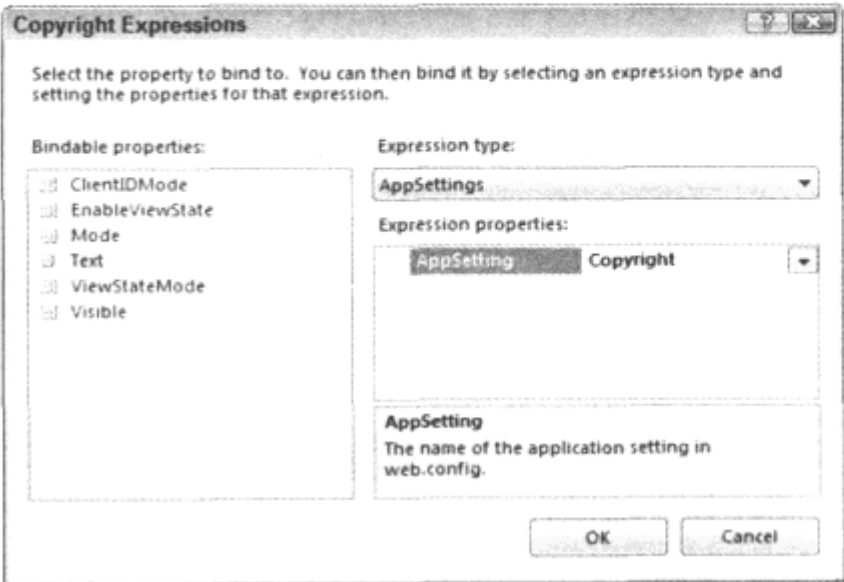


图 19-2

当单击 OK 按钮时，VWD 就会修改 Literal 控件的 Text 属性，让它包含对正确应用程序设置的引用。

使用表达式语法从 web.config 文件中获得值非常有用，但也许不能满足所有需求。因此，最好了解也能够通过编程检索这些值。要实现这个目标，可以使用 WebConfigurationManager 类。

19.1.4 WebConfigurationManager 类

System.Web.Configuration 名称空间中的 WebConfigurationManager 类提供对存储在配置文件中的数据的数据的访问权。它还特别支持 web.config 文件的 appSettings 元素和 connectionStrings 元素，从而

只使用一行代码就能从这些部分检索数据。下面的代码段显示如何从 `appSettings` 元素中检索之前看到过的 `FromAddress` 值。

VB.NET

```
Imports System.Web.Configuration
...
Dim fromAddress As String = WebConfigurationManager.AppSettings.Get("FromAddress")
```

C#

```
using System.Web.Configuration;
...
string fromAddress = WebConfigurationManager.AppSettings.Get("FromAddress");
```

`Get` 方法总是作为字符串返回数据，因此如果不需要字符串，需要将它转换为合适的类型。例如，像如下所示如果在 `web.config` 文件中保存 `Boolean` 值：

```
<add key="SendMailOnError" value="true" />
```

就要用下面的代码检索和转换值：

VB.NET

```
Dim sendMail As Boolean =
    Convert.ToBoolean(WebConfigurationManager.AppSettings.Get("SendMailOnError"))
```

C#

```
bool sendMail =
    Convert.ToBoolean(WebConfigurationManager.AppSettings.Get("SendMailOnError"));
```

虽然能够从 `Web` 窗体和用户控件中直接访问 `Code Behind` 文件中的 `WebConfigurationManager` 类（假设已经导入 `System.Web.Configuration` 名称空间），但我更喜欢在访问 `web.config` 文件的自定义配置类中创建静态的只读属性来获得值。下面的“试一试”练习将演示怎么做。

试一试

将应用程序设置移动到 web.config 文件

在本“试一试”练习中，要创建一个具有一些属性的类，这些属性从 `web.config` 文件中获得它们的值。然后要在代码中使用该类的这些属性取代前面使用的硬编码的值。

(1) 在 `App_Code` 文件夹内，创建一个新的类文件，将它命名为 `AppConfiguration.vb` 或 `AppConfiguration.cs`。在 C# 中，删除构造函数代码，如下面代码块所示：

```
public AppConfiguration()
{
    //
    // TODO: Add constructor logic here
    //
}
```

因为这个类有专门的静态属性，所以不需要这个构造函数。

(2) 在类文件的顶部，为 `System.Web.Configuration` 名称空间添加一个 `Imports/using` 语句：

VB.NET

```
Imports System.Web.Configuration
```

C#

```
using System.Web.Configuration;
```

(3) 添加一个新的 Shared (在 C#中是 static)只读属性到这个类中，它从 web.config 文件中返回 FromAddress。从第 5 章可以了解，Shared/static 成员(例如方法或属性)在类本身上起作用，而不是在类的实例上起作用。

VB.NET

```
Public Class AppConfiguration
    Public Shared ReadOnly Property FromAddress() As String
    Get
        Dim result As String =
            WebConfigurationManager.AppSettings.Get("FromAddress")
        If Not String.IsNullOrEmpty(result) Then
            Return result
        End If
        Throw New Exception("AppSetting FromAddress not found in web.config file.")
    End Get
    End Property
End Class
```

C#

```
public class AppConfiguration
{
    public static string FromAddress
    {
        get
        {
            string result = WebConfigurationManager.AppSettings.Get("FromAddress");
            if (!string.IsNullOrEmpty(result))
            {
                return result;
            }
            throw new Exception("AppSetting FromAddress not found in web.config file.");
        }
    }
}
```

(4) 重复前面的步骤，但这次通过创建 FromAddress 的副本来创建下面 3 个属性：

- FromName
- ToAddress
- ToName

不要忘记将 FromAddress 的所有 3 个属性重命名为新的属性名。

(5) 还是在 AppConfiguration 类里面，创建名为 SendMailOnError 的 Boolean 属性：

VB.NET

```
Public Shared ReadOnly Property SendMailOnError() As Boolean
```

```
Get
    Dim result As String =
        WebConfigurationManager.AppSettings.Get("SendMailOnError")
    If Not String.IsNullOrEmpty(result) Then
        Return Convert.ToBoolean(result)
    End If
    Throw New Exception(
        "AppSetting SendMailOnError not found in web.config file.")
End Get
End Property
```

C#

```
public static bool SendMailOnError
{
    get
    {
        string result = WebConfigurationManager.AppSettings.Get("SendMailOnError");
        if (!string.IsNullOrEmpty(result))
        {
            return Convert.ToBoolean(result);
        }
        throw new Exception(
            "AppSetting SendMailOnError not found in web.config file.");
    }
}
```

(6) 配置好这 5 个属性之后，保存并关闭 AppConfiguration 文件。

(7) 打开 Controls 文件夹中 ContactForm.aspx 中的 Code Behind 文件，定位设置 From 和 To 地址的代码，并用相应的 AppConfiguration 值取代硬编码的值。

VB.NET

```
myMessage.From = New MailAddress(AppConfiguration.FromAddress,
    AppConfiguration.FromName)
myMessage.To.Add(New MailAddress(AppConfiguration.ToAddress,
    AppConfiguration.ToName))
```

C#

```
myMessage.From = new MailAddress(AppConfiguration.FromAddress,
    AppConfiguration.FromName);
myMessage.To.Add(new MailAddress(AppConfiguration.ToAddress,
    AppConfiguration.ToName));
```

注意，IntelliSense 会帮助您选择 AppConfiguration 类的正确属性。

(8) 保存修改并关闭文件。

(9) 打开 Global.asax 文件，并将 Application_Error 内的所有代码包装在 If 语句中，它确保 SendMailOnError 设置为 True。另外，修改硬编码的电子邮件地址，使其使用 AppConfiguration 类的 ToAddress 属性和 ToName 属性：

VB.NET

```
Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    If AppConfiguration.SendMailOnError Then
```



```

    If HttpContext.Current.Server.GetLastError() IsNot Nothing Then
        ...
        Dim myMessage As MailMessage = New MailMessage(AppConfiguration.FromAddress,
            AppConfiguration.ToAddress, mailSubject, message)
        ...
    End If
End If
End Sub

C#
void Application_Error(object sender, EventArgs e)
{
    if (AppConfiguration.SendMailOnError)
    {
        if (HttpContext.Current.Server.GetLastError() != null)
        {
            ...
            MailMessage myMessage = new MailMessage(AppConfiguration.FromAddress,
                AppConfiguration.ToAddress, mailSubject, message);
            ...
        }
    }
}
}
```

(10) 打开 web.config 文件, 添加下面的<appSettings>元素作为<configuration>主元素的直接子元素。将 FromAddress 和 ToAddress 的电子邮件地址修改为自己的地址:

```

<configuration>
  <appSettings>
    <add key="FromAddress" value="webmaster@planetwrox.com" />
    <add key="FromName" value="Planet Wrox" />
    <add key="ToAddress" value="webmaster@planetwrox.com" />
    <add key="ToName" value="Planet Wrox" />
    <add key="SendMailOnError" value="true" />
  </appSettings>
  ...
</configuration>
```

(11) 保存所有修改。按 Ctrl+F5 组合键打开浏览器的主页。进入 Contact 页面, 然后填写联系信息表单。应该在第(10)步指定的地址中接收到电子邮件。

(12) 在浏览器中请求不存在的页面。例如, 将浏览器地址栏中的地址的页面名称修改为 DefaultTest.aspx。像第 18 章那样, 您将会收到一个“File Not Found”的消息, 以及一封带有异常的详细信息的电子邮件。

(13) 返回至 Visual Web Developer 窗口, 打开 web.config 文件, 将 SendMailOnError 的设置从 true 修改为 false:

```

    <add key="SendMailOnError" value="false" />
```

(13) 保存所有修改, 再次请求不存在的页面。因为已经修改了 SendMailOnError 设置, 因此不会再收到具有异常详细信息的电子邮件。

工作原理

AppConfiguration 类的属性在 web.config 文件中查找请求的应用程序设置。当没有定义设置或者设置没有包含值时，每个属性都会抛出一个异常。这对于在早期阶段检测遗漏的应用程序设置非常有用。现在不是默认返回一个空值，而是可以捕获一个异常，它提示必须添加所需的应用程序设置。在运行时，代码像下面这样访问这些属性：

VB.NET

```
myMessage.From = New MailAddress(AppConfiguration.FromAddress,
                                   AppConfiguration.FromName)
```

C#

```
myMessage.From = new MailAddress(AppConfiguration.FromAddress,
                                   AppConfiguration.FromName);
```

因为属性已经被定义为 Shared(在 C#中为 static)，所以可以在 AppConfiguration 类上直接访问它们，而不需要先创建一个新的 AppConfiguration 实例。

虽然可以在代码中直接访问 web.config 文件中的<appSettings>元素(例如，可以直接使用 WebConfigurationManager.AppSettings.Get("FromAddress")来获得 ContactForm.ascx 中的电子邮件地址)，但是将<appSettings>元素包装在它们自己类中的共享属性中会更好。这种解决方案在 AppConfiguration 类上提供 IntelliSense，这样就很容易知道什么配置属性可用。它也允许编写集中的代码，用于在没有找到所需的应用程序设置时抛出异常，或者用于提供合理的默认值。注意，只有当不能返回有效值的时候，属性才会抛出异常。如果直接在自己的代码中访问 web.config 文件，那么每次访问设置时都需要检查有效值。

相同的原则也可用于 SendMailOnError 设置。如果在运行时发生异常，Application_Error 中的代码会参考 SendMailOnError 属性。这个属性再检查 web.config 文件中的<appSettings>元素，以确定是否应该用电子邮件发送错误消息。因为 SendMailOnError 的属性是 Boolean 值，所以代码使用 Convert.ToBoolean 将从 web.config 文件返回的字符串转换为 Boolean 值。

通过将这些值保存在 web.config 文件中而不是硬编码它们，站点变得更容易维护和部署。如果要激活它们，所要做的就是为生产环境创建一个 web.config 副本，并修改一些设置。

将硬编码的应用程序设置移动到中心 web.config 文件之后，部署过程的下一步就是创建 Web 站点的副本。

19.2 复制 Web 站点

在开发站点的过程中，使用 Visual Web Developer 配置的内置 Web 服务器。虽然这个服务器对于本地开发非常好，但在生产环境中就不能使用它，因为它只侦听来自本地主机的请求。要将站点投入到生产环境中使用，需要将它部署到运行 IIS 的计算机中——IIS(Internet Information Services)是 Microsoft 的专业 Web 服务器。本节将介绍如何准备站点，以便可以在 IIS 下运行它。本章稍后将讨论如何安装和配置 IIS。

要将站点部署到生产服务器中，可以使用表 19-1 显示的部署目标(来自 VWD 内部)。

表 19-1

部署选项	说明
File system	该选项允许在开发计算机或网络化计算机的本地文件系统上创建站点副本。如果稍后要将这些文件手动移动到生产服务器中，那么这个选项就很有用
Local IIS	该选项允许创建将在本地 IIS 安装下运行的站点的副本
FTP Site	该选项允许使用 FTP 将组成 Web 应用程序的文件直接发送到远程服务器中
Remote Site	该选项允许将组成 Web 应用程序的文件发送到远程 IIS 服务器中。要使这个选项生效，远程服务器需要安装 Front Page Server Extensions 查看 IIS 附带的文档或者咨询远程服务器的管理员以获得使用这个选项的帮助

如果使用的是 Visual Studio 的商业版本，可以使用 VWD 提供的两种主要的部署方法(Copy Web Site 和 Publish Web Site)来访问这 4 个部署选项。如果使用的是免费的 Express 版本，就只能使用 Copy Web Site。



注意：在本书的一开始，就介绍了 Web Site Projects(WSP)和 Web Application Projects(WAP)的区别。在本书中，我选择使用 WSP 模型，因为对于初学者来说它更易于使用，并且它还支持 WAP 中不可用的一些功能(例如 Profile 和动态 App_Code 文件夹)。但是，有一个主要的功能是 WAP 支持而 WSP 不支持的：Web 包。这是一种创建整个 Web 站点的安装包的机制，从而使站点更易于部署到生产服务器上。这种机制考虑了开发计算机和生产计算机之间的配置的区别。关于 WAP 中的 Web 包的更多信息，可以查看 VWD 开发团队的博客上提供的有关开发的一系列链接。通过这个网址可以打开他们的博客：<http://tinyurl.com/WebDeploymentOverview>。

19.2.1 创建 Web 站点的简单副本

Copy Web Site 命令可以使用 4 个传输选项中的任何一项来创建站点的副本。这是将站点快速复制到其他位置的好方法，这些位置包括生产服务器，甚至是便携式媒体设备(例如可以随身携带的 USB 存储棒)。可以从 Solution Explorer 的工具栏(如图 19-3 所示)或从 Website 主菜单中访问 Copy Web Site 选项。

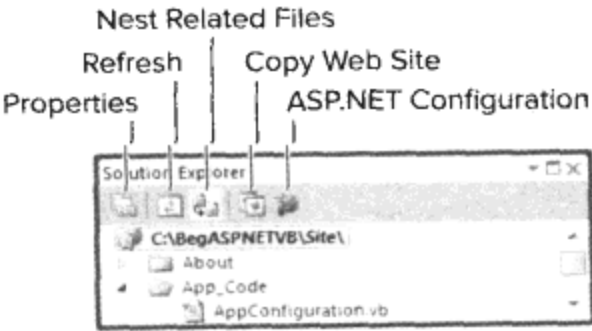


图 19-3

在创建站点的副本之前，最好检查一下 Web 站点的状态。应该使用 VWD 全部重新编译 Web

站点内的所有代码和页面。这有助于在将 Web 站点部署到生产环境之前检测它里面的问题。

部署站点也是进行管理的好时机。为了避免减缓部署进程并且让站点尽可能的保持简洁，应该从 Web 站点中删除在生产环境中不需要的一些文件。

在下面的“试一试”练习中，将介绍如何使用 Copy Web Site 命令来创建整个 Planet Wrox 项目的副本。在稍后的“试一试”练习中，当为生产 Web 站点配置 IIS 时会再次使用这个副本。

试一试 使用 Copy Web Site 选项

在这个“试一试”练习中，将使用 Copy Web Site 选项以及 Local File System 选项来创建站点的副本。其他 3 个传输选项(FTP、Local IIS 和 Remote IIS)的运行方式与此类似。这些选项最大的不同是，它们要求提供目标的详细情况，例如用户名和密码或者要使用的 IIS 的网站。使用在本练习中创建的副本，可以手动配置一个 IIS 网站，其中有些内容本章稍后讨论。

(1) 关闭 Visual Web Developer 中所有打开的文件，然后从主菜单中选择 Build | Rebuild Web Site 选项。这样做就要求 VWD 重新编译整个站点，即使它已经编译了某些部分。VWD 会在 Error List 中列出站点存在的所有问题。为了验证站点没有错误，请打开 Error List (从主菜单中选择 View | Error List 选项)，并确保没有任何编译错误。修复站点内存在的所有错误。如果使用的是 C#，则会得到一个关于 ContactForm.ascx 中不可到达的代码的警告。可以忽略该警告，或者删除对 jQuery 文件的引用，添加这些引用是为了启用 IntelliSense。

(2) Error List 变空以后，请选择 Website | Copy Web Site 选项或单击 Solution Explorer 工具栏上的 Copy Web Site 图标，如图 19-3 所示。

(3) 在对话框顶部，单击 Connect 按钮以打开对话框，它允许选择站点的目标位置。

在对话框左边，确保已选择 File System 选项。然后在对话框右边，定位文件夹 C:\BegASPNET，单击选择它，然后单击 Create New Folder 按钮；这个按钮在对话框的右上角，是一个黄色的文件夹图标。输入 Release，再按 Enter 键应用新名称。图 19-4 显示最终的对话框。

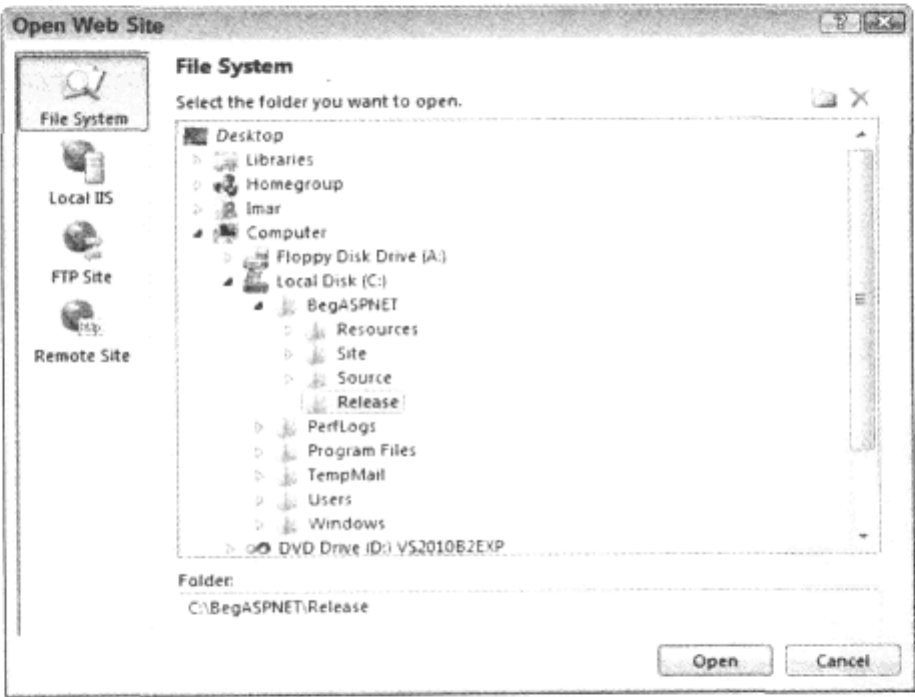


图 19-4

最后，单击 Open 按钮选择 C:\BegASPNET\Release 作为复制操作的目标位置。

(4) 在 Copy Web Site 对话框中，通过单击文件或文件夹将焦点放到左边的列表上，然后按下

Ctrl+A 组合键，选择 Source Web Site 列表中的全部文件。

(5) 单击两个列表之间的 Synchronize Selected Files 按钮(具有两个相向箭头的按钮)，如图 19-5 所示。这将启动同步进程。因为右边显示的文件夹是空的，所以左边列表中的所有文件都会复制到右边。复制完成之后，对话框如图 19-5 所示。

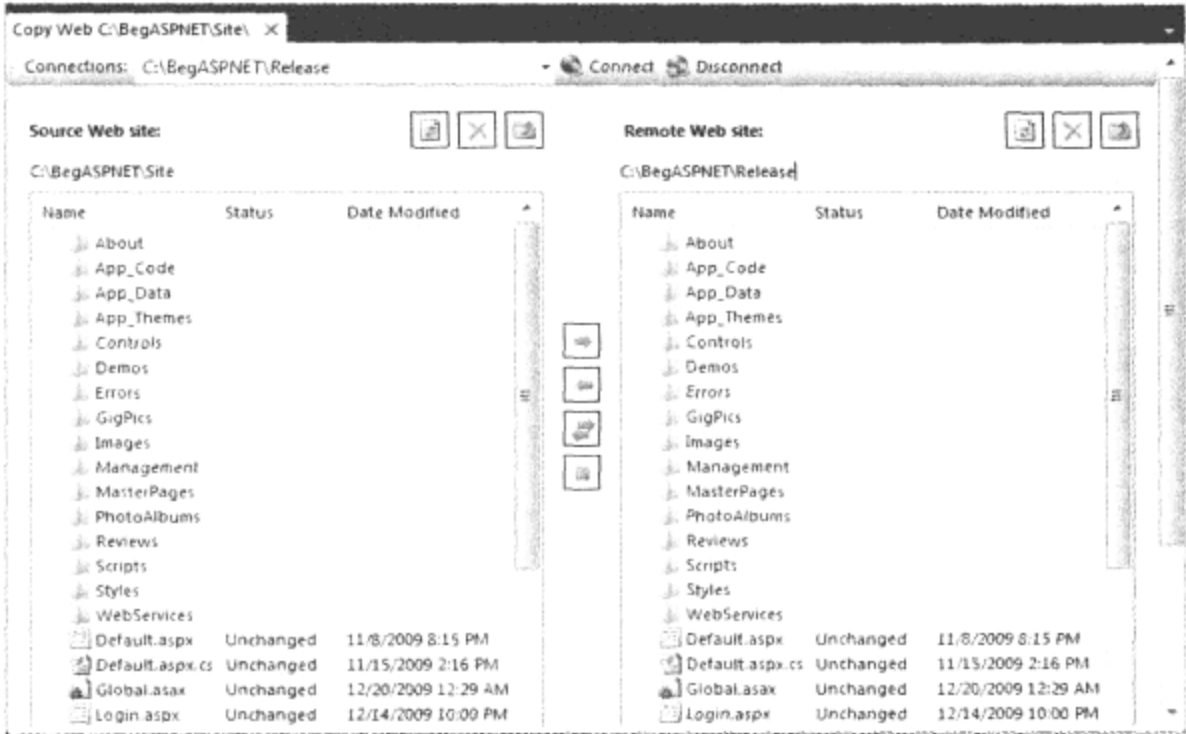


图 19-5

(6) 接着打开 Windows 资源管理器窗口，浏览到 C:\BegASPNET\Release。验证组成站点的所有相关文件都在那里。

工作原理

Copy Web Site 选项只创建组成站点的全部文件的副本。它也可以在不同位置创建站点的副本，包括本地文件系统、FTP 服务器和 IIS 服务器。在这个“试一试”练习中，在本地硬盘上创建副本。图 19-5 中两个文件列表之间的前两个按钮(蓝色箭头)允许将文件从源位置复制到远程位置，反之亦然。第 3 个按钮(有两个箭头)允许同步而不仅仅是复制文件。当在本地系统上创建副本时，这看上去并不是一个大问题，但是当通过一个缓慢的 FTP 连接来创建副本时，这个工具就很有帮助，因为它只上传新的和修改过的文件，而不上传未经修改的文件。在本练习中，选择将文件复制到本地系统中。这是创建与开发环境分离的副本的好方法，可以在本地计算机上运行这种开发环境。当然，也可以使用 FTP 程序、USB 存储棒等将相同的文件组复制到其他计算机上。另外，如果在外部驻留站点，主机提供商可能会提供基于 Web 的界面，用于将这些文件上传到它们的服务器。

这种分离的本地副本允许先修改一些文件(例如 web.config 文件)，然后将它们上传到主机。除了复制 Web 站点，Visual Studio 的商业版本也支持发布 Web 站点。

19.2.2 发布 Web 站点

Publish Web Site 命令只在 Visual Studio 的商业版本中可用，在 Express 版本中不可用，它与 Copy Web Site 选项类似，因为它创建可以用于部署的 Web 站点的副本。然而，它也有所不同，因为它允许预编译应用程序，也就是说 ASPX 页面中的 Code Behind 文件内的所有代码、控件、App_Code

中的代码文件等都编译为.NET assemblies;的扩展名是.dll 的文件，位于站点的 bin 文件夹中。预编译的主要优点是源保护(访问服务器的其他人不能查看源代码)和首次请求页面时提高性能。当首次请求没有预编译的页面时会动态编译它们，这需要占用一点时间。记住，定期访问站点的用户绝对不会看到应用程序的源代码。他们能够看到的是发送到浏览器的最终 HTML。

从 Build 菜单中可以使用 Publish Web Site 命令，并打开如图 19-6 所示的对话框。

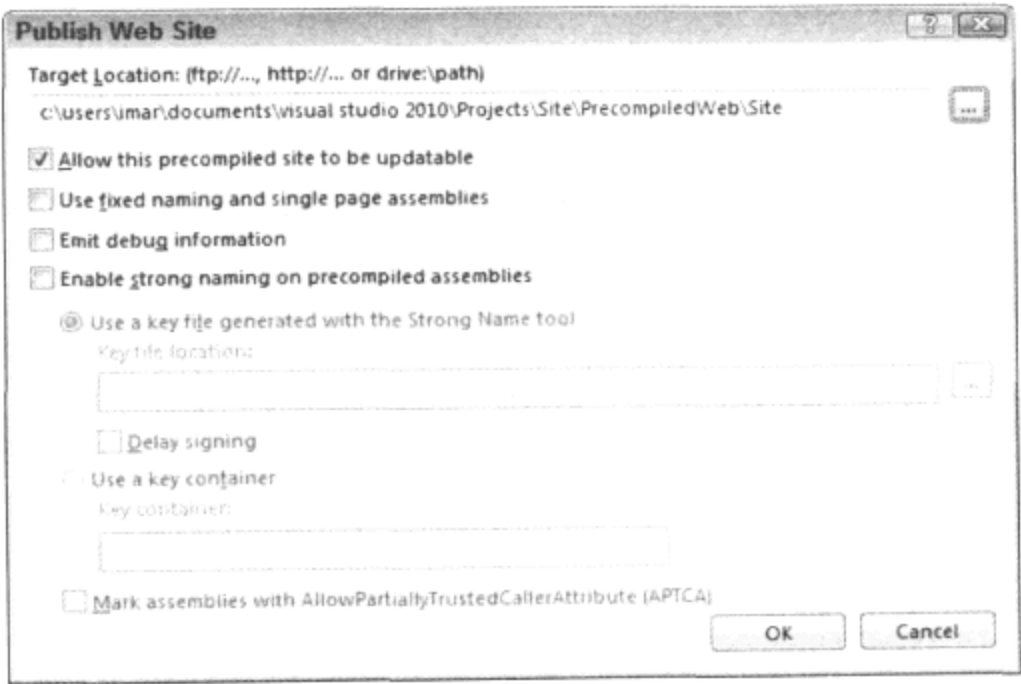


图 19-6

单击目标位置结尾的省略号按钮以打开与图 19-4 所示的相同的对话框，从中可以选择目标位置。当选中 Allow This Precompiled Site to Be Updateable 选项时，VWD 会将所有 VB.NET 和 C#代码编译为.NET 程序集，并将它们放置在它在站点根文件夹下创建的 bin 文件夹里。它不会修改 ASPX 文件和 ASCX 文件中的标记。然而，如果关闭这个选项，所有标记代码也会编译到.dll 文件中。实际文件仍然需要部署到服务器中，但它们的内容已经被占位符文本所取代：“This is a marker file generated by the precompilation tool, and should not be deleted!” (这是预编译工具生成的标记文件，不应该删除!)。当浏览器请求页面时，ASP.NET 运行库在 bin 文件夹里的程序集中找到相应内容，并显示它的内容，就像它是正常页面一样。如果要防止其他人在部署站点后修改站点以访问服务器，后面一个选项就特别有用。因为所有的源代码和标记都编译为.dll 文件，所以再也无法在服务器上修改它，除非上传一组新的已发布文件。

将 Web 站点复制或发布到本地系统上的新文件夹中，这只是部署过程的一个步骤。下一步是要配置 Web 服务器，让它知道到哪里查找文件。

19.3 在 IIS 下运行站点

到目前为止，都在使用 Visual Web Developer 配置的内置 Web 服务器来调试和测试应用程序。然而，由于对这个服务器的请求仅限于来自本地主机的请求以最小化安全隐患，因此需要使用 IIS，大多数主要的 Windows 版本里都包含它。为了在 IIS 下运行 Web 站点，需要执行下面几个步骤：

- (1) 安装和配置 IIS;
- (2) 安装和配置.NET Framework 4;

(3) 配置安全设置。
根据系统的当前状态，有些操作是可选的。下面的部分将介绍如何实现这些步骤。



注意：安装和配置 Web 服务器是一项复杂的任务。需要考虑多种因素，包括操作系统、系统配置、用于登录计算机的账户和最终使用的 SQL Server 等。如果出现问题，也不要慌张。可以访问 IIS 的网站 www.iis.net，查看详细的指导步骤，或者也可以参与本书的论坛，网址为 <http://p2p.wrox.com>，从中可以找到可以帮助您的其他程序员(包括我自己)。

19.3.1 安装和配置 Web 服务器

虽然大多数 Windows 版本都包含 IIS，但默认情况下不会安装它，因此首先就要安装它。还要确保使用的 Windows 版本支持 IIS。虽然 Windows Vista 和 Windows 7 的 Starter 版本和 Home Basic 版本提供了部分 IIS，但不能在它们上面运行 ASP.NET 页面，因此至少需要安装 Home Premium 版本。Windows 基于服务器的版本则完全支持 IIS。



注意：虽然面向消费者的 Windows 版本(如 Windows Vista 和 Windows 7)支持 IIS，但是这并不意味着这些操作系统是驻留 Web 站点的最佳选择。通常使用这些 Windows 版本进行本地开发和测试，而使用 Windows 的服务器版本(Windows Server 2003 和 Windows Server 2008)来驻留生产 Web 站点。

要在 Windows 计算机上安装和配置 IIS，需要作为 Administrator 登录。如果用来登录计算机的账户没有管理权限，就要请求管理员安装和配置 IIS。

除了安装 IIS 之外，还要了解如何在 IIS 中创建和配置 Web 站点。由于在 Windows 下安全运行的方式的原因，在配置 IIS 之后站点可能不会立即运行，除非在 Windows 下修改某些安全设置。在下一节和名为“配置文件系统”的“试一试”练习中将会介绍相关内容。

如果您已经安装了 SQL Server 2008 Express 版本，那么就能更加轻松地测试 IIS 设置。如果您按照下面的“试一试”练习在运行 VWD 的同一台计算机上进行操作，就会看到这种情况。否则，可以使用 Web Platform Installer 下载免费的 SQL Server Express 版本，其网址为 www.microsoft.com/web。如果您使用的是其他 SQL Server 版本，或者在远程计算机上运行 SQL Server，那么要特别留意本章的 18.4 节和附录 B。

1. 确保已安装 IIS

本节将介绍如何确保在较新的 Windows 版本上安装了 IIS，这些版本包括：Windows Vista、Windows 7 和 Windows Server 2008(R1 版和近期发布的 R2 版)。检查和安装 IIS 的程序步骤随着 Windows 的版本的不同而不同，因此下面的部分将通过不同选项来说明这一点。应该选择匹配您的操作系统的那一种方法。关于 IIS 的更多信息，包括如何在这里没有介绍的 Windows 版本上安装 IIS 的帮助信息，可以查看 IIS 的官方网站 www.iis.net。



注意：Microsoft 创建了一个叫做 Web Platform Installer(WPI)的工具，允许使用一个安装程序安装 IIS、SQL Server、VWD 2010 等。这是一个优秀的工具，可以快速建立新的开发或者生产服务器，它具有 ASP.NET 开发和部署所需的全部工具和服务。可以从 www.microsoft.com/web 上找到有关该工具的更多信息。如果使用 WPI，则可以跳过下面大部分介绍 IIS 和 ASP.NET 安装的部分。一定要确保至少安装了 IIS、ASP.NET 和 SQL Server 2008。为了更好地理解在安装和配置操作系统中涉及的步骤，可以按照下面部分中讨论的每个步骤逐步执行。当熟悉了这个过程以后，就可以使用 WPI 进行新的安装。

Windows Vista 和 Windows 7

应该通过 Programs and Features 部分来安装 IIS，可以通过“控制面板”或单击 Start 按钮，在搜索框中输入 appwiz.cpl，然后按 Enter 键来访问这个部分。在 Programs and Features 屏幕中，单击 Turn Windows Features On Or Off 链接来打开 Windows Features 对话框，如图 19-7 所示。

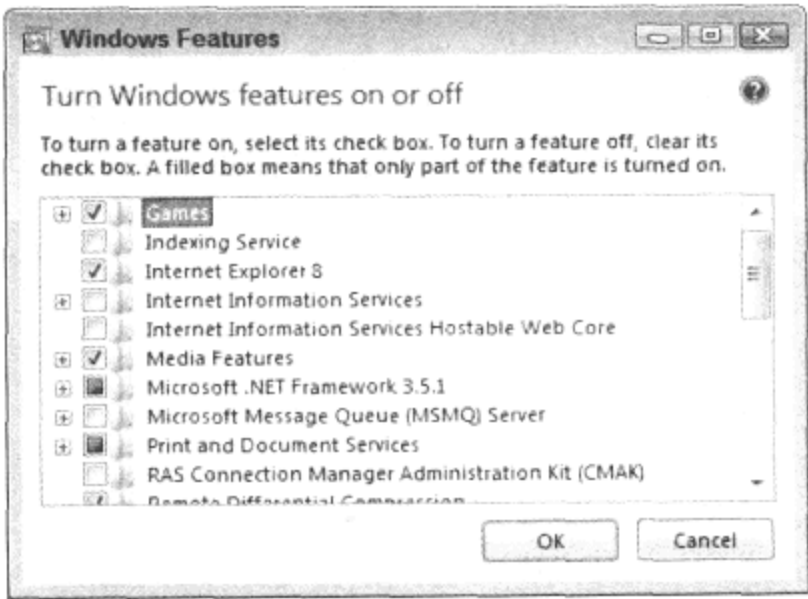


图 19-7

单击 Internet Information Services 项来选择它。这也会选择它的一些必需的子功能。然后展开 Internet Information Services | World Wide Web Services | Application Development Features 命令，并选择 ASP.NET 选项。这也会选中其他一些 Development 功能。关于其他可选组件，可以参阅 IIS 文档。最后，单击 OK 按钮，Windows 就会安装所请求的功能。在下一节会继续配置 ASP.NET。

Windows Server 2008/Windows Server 2008 R2

应该通过 Programs and Features 部分来安装 IIS，可以通过“控制面板”或单击 Start 按钮，在搜索框中输入 appwiz.cpl，然后按 Enter 键来访问这个部分。在 Programs and Features 屏幕中，单击 Turn Windows Features On Or Off 链接来打开 Server Manager 对话框，如图 19-8 所示。

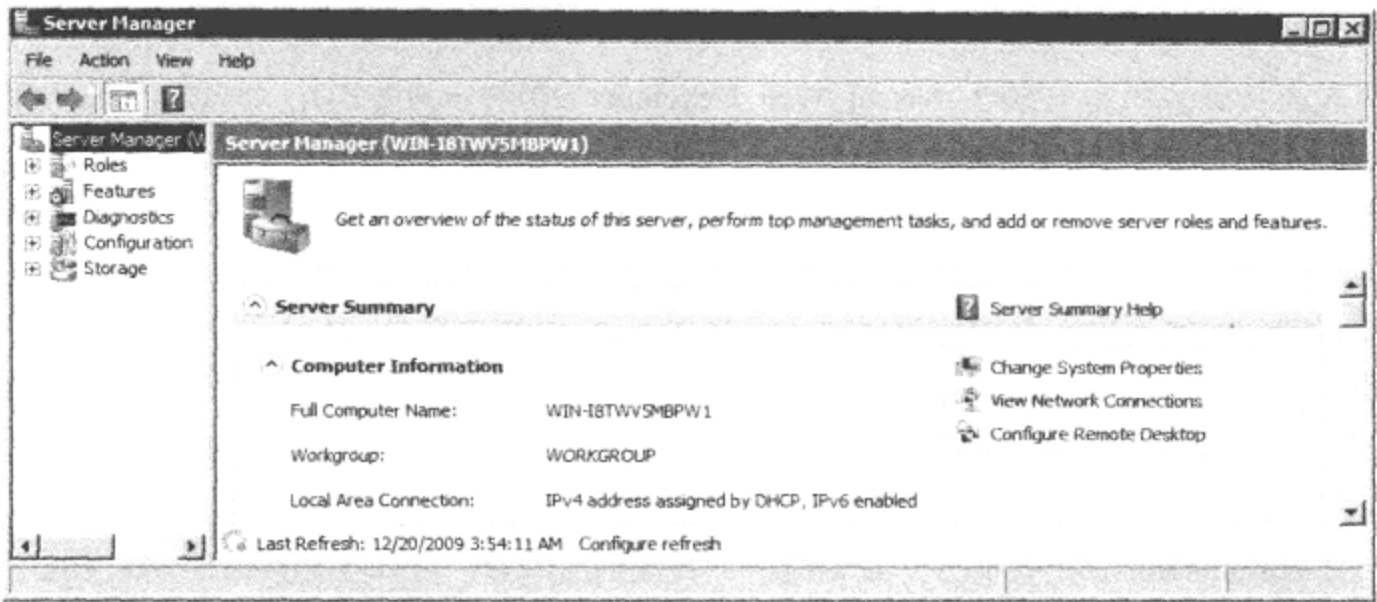


图 19-8

右击左边树状图中的 Roles 项，选择 Add Roles 命令(可能需要等待 Roles 面板完成加载)。按照屏幕指示安装角色 Web Server (IIS)。当到达能够选择不同 Role Services 的屏幕时，确保至少选择 Application Development 节点下的 ASP.NET 选项，如图 19-9 所示。这也会选择 ASP.NET 所需的大量其他功能。

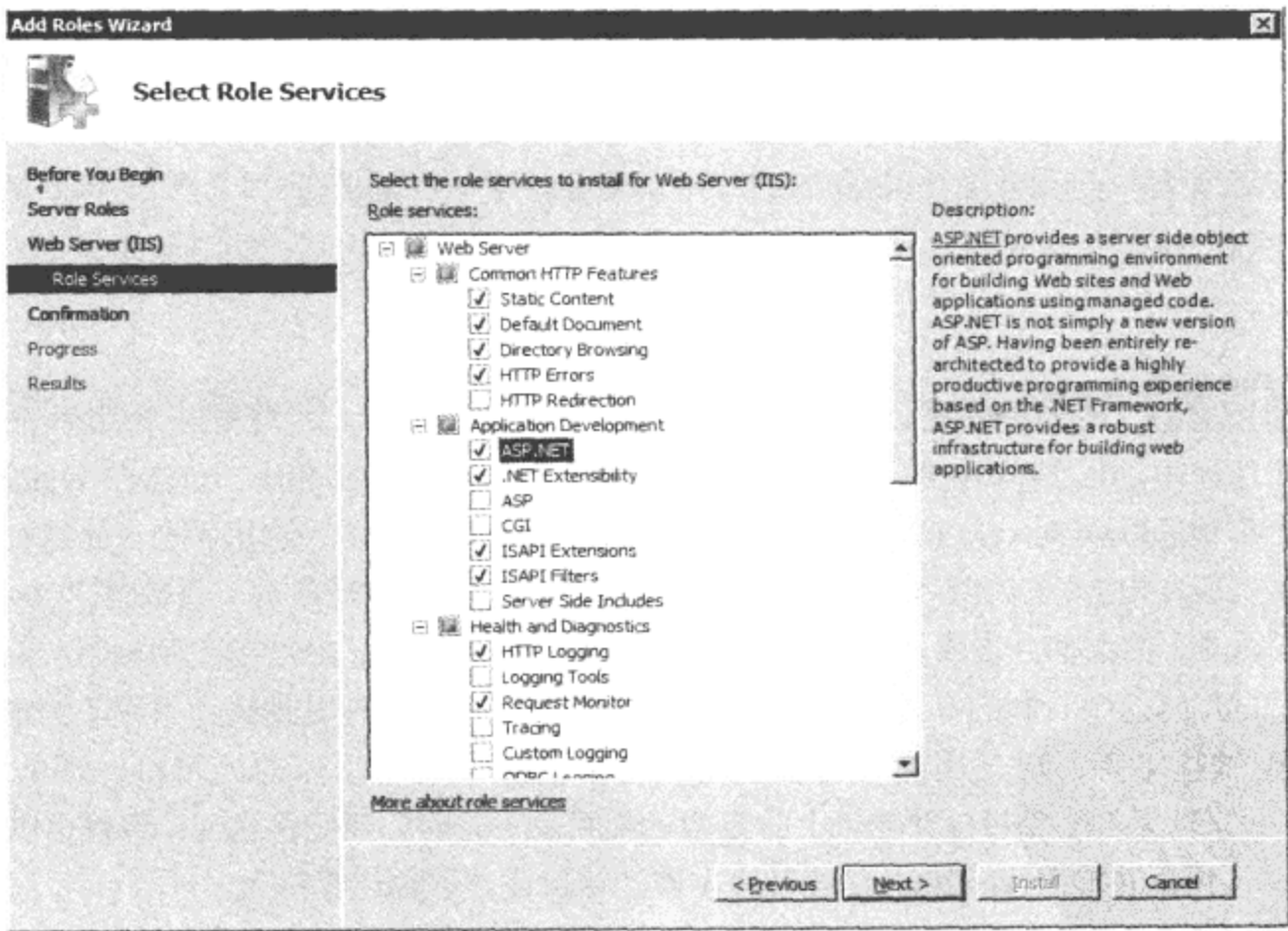


图 19-9

单击 Next 按钮继续运行 Roles Wizard，直到 Windows 开始为选中的角色安装所需的文件为止。一切准备就绪之后，还需要重启计算机。重新登录之后，就可以使用 IIS。成功安装 IIS 之后，还要确保已经安装了 Microsoft .NET Framework version 4。

2. 安装和配置 ASP.NET

如果在目标计算机上安装了 Visual Web Developer 2010(任何版本), 那么就已经安装了 .NET Framework 4。否则就要从 Microsoft 站点下载它, 其地址是 <http://msdn.microsoft.com/cn-us/netframework>。跟踪 Download 链接或 Install 链接, 或者使用搜索选项搜索“下载 .NET framework 4”。确保下载 .NET Framework 4 的完整版本, 而不是早期版本或者 Client Profile 包。下载 .NET Framework 之后, 可以运行安装程序, 并按照屏幕提示操作。

如果计算机上已经有 .NET Framework 4, 后来才安装 IIS, 那么就要告诉 IIS 已经存在 framework。通常情况下, 这在 .NET Framework 的安装过程中完成。如果后来才安装 IIS, 那么就要手动完成该操作。要在 IIS 中注册 ASP.NET, 请按如下步骤操作:

- (1) 在 Administrative 模式下打开命令提示。要做到这一点, 可以在搜索框中输入 cmd, 然后按 Ctrl+Shift+Enter 组合键启动具有提升权限的命令提示。确认操作之后, 就可以正常打开命令提示;
- (2) 通过输入下面的命令并按 enter 键后导航到 .NET Framework version 4 文件夹:

```
cd \Windows\Microsoft.NET\Framework\v4.0.30128
```

注意, 在您的计算机上, v4.0 后欧盟的实际版本号可能会有所不同。另外, 如果使用的是 64 位的 Windows 版本, 那么 Framework 文件夹命名为 Framework64。输入命令提示之前, 先使用 Windows 资源管理器找到正确的文件夹;

- (3) 输入 aspnet_regiis -i, 然后再次按下 Enter 键。

不久之后就会收到 ASP.NET 4 已经成功地在 IIS 中注册的消息。

现在已经安装并正确配置了 IIS 和 .NET Framework, 下一步就是在 IIS 下配置 Web 站点。在下面的“试一试”练习中将介绍如何完成此操作。在“试一试”练习之后, 将进一步讨论配置系统的安全权限。

试一试 在 Windows Vista、Windows 7 或 Windows Server 2008 中配置站点

在这个练习中, 将看到如何配置随 IIS 提供的标准的“Default Web Site”。虽然在 Windows Vista、Windows 7 和 Windows Server 2008 上可以在 IIS 下创建多个站点, 但是这里不会讨论这样的选项。要学习更多关于在 IIS 下创建多个 Web 站点的信息, 可以联系系统管理员, 或者阅读 IIS 的帮助文档。Windows 7 上的大部分步骤与 Windows Vista 和 Windows Server 2008 的步骤相同。但是, 在下面的“试一试”练习中看到的界面是在 Windows 7 中实现的, 并且与其他操作系统上的结果稍有不同。如果在进行本练习时使用的计算机不是构建 Planet Wrox 站点的计算机, 那么一定要将 BegASPNET 文件夹复制到目标计算机上的 C 盘的根目录下。另外, 要确保这台计算机可以访问 SQL Server 2008, 不管 SQL Server 2008 是安装在本地计算机还是远程计算机上。

- (1) 打开 Internet Information Services Manager 窗口。在控制面板的 System and Security 类别(在 Windows Vista 和 Windows Server 2008 R1 中叫做 System and Maintenance)下的 Administrative Tools 部分可以找到这个项。或者, 可以单击 Start 按钮, 在搜索框中输入 inetmgr, 然后按下 Enter 键。

- (2) 展开左边的导航树状图, 直到看到 Application Pools 和 Default Web Site 节点为止, 如图 19-10 所示。

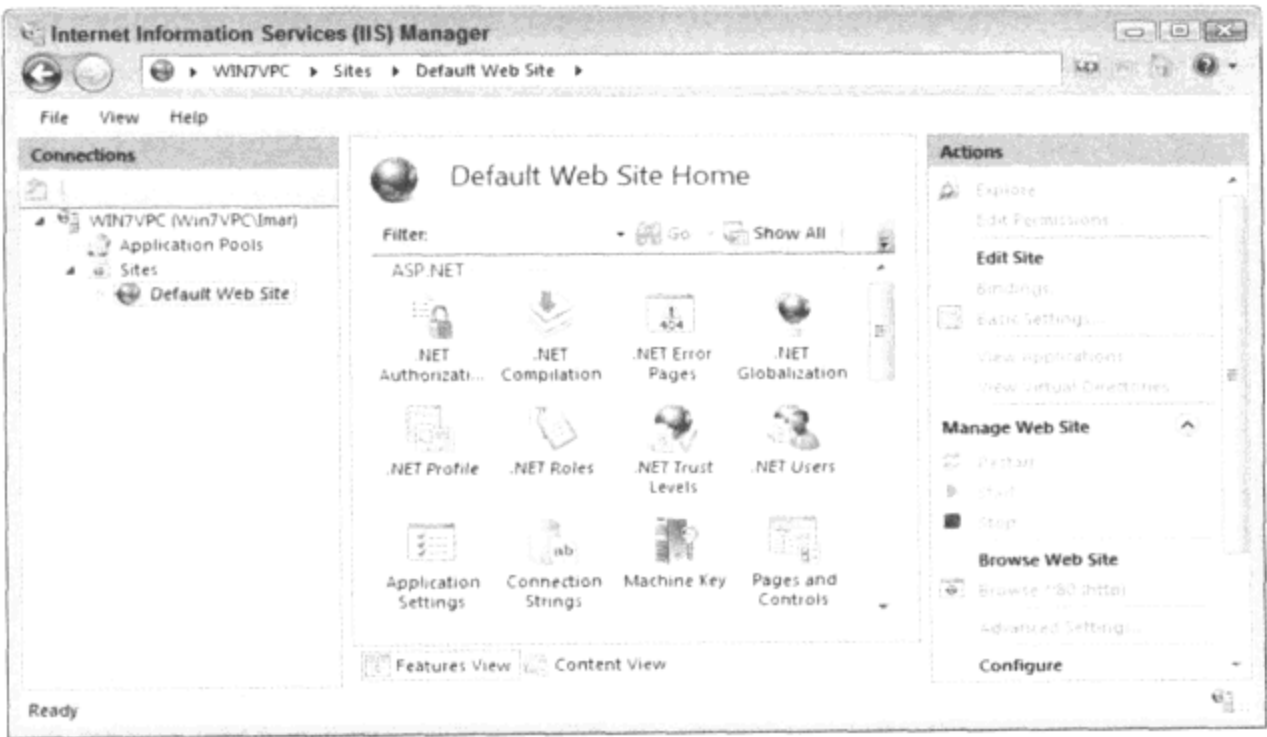


图 19-10

(3) 单击 Application Pools 项，确认有一个叫做 ASP.NET v4.0 的项，该项使用 v4.0 作为 .NET Framework Version，并且把其 Managed Pipeline Mode 设置为 Integrated。如果没有该项，则单击右侧 Actions 面板中的 Add Application Pool 选项，使用 .NET Framework version 4.0 创建一个叫做 ASP.NET v4.0 的新应用程序池，并将其 Managed Pipeline mode 设置为 Integrated。

(4) 选择 ASP.NET v4.0 应用程序池(不管是否已经在那里)并单击 Advanced Settings 选项。定位叫做 Identity 的属性，并确保它设置为 Network Service(Windows Vista 和 Windows Server 2008 R1)或 ApplicationPoolIdentity(Windows 7 和 Windows Server 2008 R2)。从选项列表中选择正确的项。后面在配置安全的时候会使用这个标识。在同一个对话框中，如果使用的是 Windows 7 或 Windows Server 2008，则将 Load User Profile 选项设置为 True。单击 OK 按钮关闭 Advanced Settings 对话框。

(5) 单击选择 Default Web Site 项，然后单击右侧的 Actions 面板中的 Advanced Settings 选项，如图 19-10 所示。

(6) 在 Advanced Settings 对话框中，单击 Physical Path 属性，然后单击省略号按钮以打开文件夹浏览器，选择文件夹 C:\BegASPNET\Release，最后单击 OK 按钮确认路径。

(7) 在同一个对话框中，单击 Application Pool，然后单击省略号按钮，选择第(3)步创建的名为 ASP.NET v4.0 的 Application Pool，然后单击 OK 按钮。再次单击 OK 按钮关闭 Advanced Settings 对话框。

(8) 接下来就是确保配置 IIS 来使用一个合理的默认文档：当只请求一个文件夹名或站点的根文件夹时会提供这个文档。Planet Wrox 站点使用 Default.aspx，它是 ASP.NET 网站最普通的默认文档名。要进行检查，则双击 IIS Features 列表中的 Default Document 选项(位于图 19-10 中显示的项的下面)。然后确保 Default.aspx 出现在列表的开头。如果没有该项，就手动添加它。对于 Windows Vista，只需在 File Name(s)列表中输入文档名，然后单击 Apply 按钮。对于 Windows 7 或 Windows Server 2008，则单击 Actions 面板中的 Add 链接来添加它，然后使用 Move Up 链接将它移动到列表的顶部。对话框如图 19-11 所示(如果列表中已经有了 Default.aspx，则它的 Entry Type 设置为 Inherited 而不是

Local)。

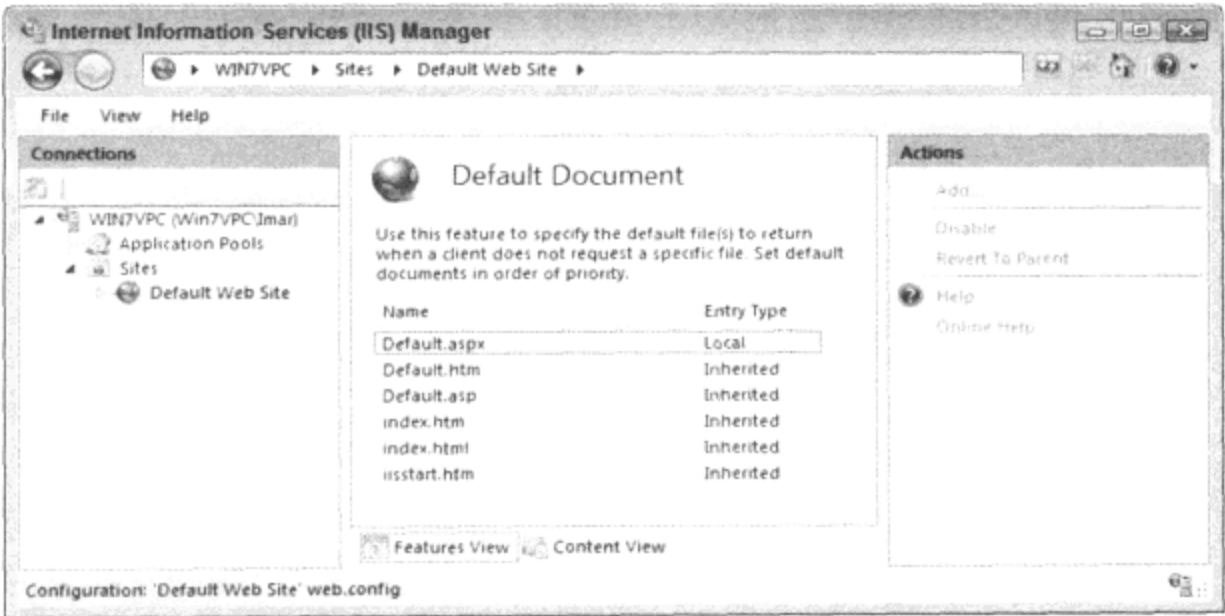


图 19-11

(9) 现在可以关闭 Internet Information Services Manager 窗口，因为就 IIS 而言，站点已经得到正确配置。然而，它仍然不能正确运行，因为还需要配置文件系统的安全权限，稍后将会讨论这一方面的内容。

工作原理

每个新的 IIS 安装都有 Default Web Site，默认情况下侦听 `http://localhost` 站点。在这个“试一试”练习中，将这个默认的 Web 站点配置为运行 Planet Wrox，但是也可以创建一个可以与其他 Web 站点一同运行的全新的站点。将站点的根文件夹指向包含 Web 站点的 Release 文件夹。建立这种映射之后，当请求 URL(例如 `http://localhost`)时 IIS 就会知道使用什么文件。这意味着像 `http://localhost/Login.aspx` 这样的 URL 映射到 `C:\BegASPNET\Release>Login.aspx` 中的物理文件中。还给 Web 站点分配了一个应用程序池，这是一种同时隔离和配置一个或多个 IIS 网站的 IIS 机制。当发生问题(例如崩溃)时，运行在不同应用程序池中的两个 Web 站点不会彼此影响。在这个“试一试”练习中，选择了一个使用 .NET 4 Framework 的应用程序池，并使用了 Integrated Pipeline mode。在这种模式中，IIS 和 ASP.NET 紧密地集成了起来，这意味着您可以在标准的 IIS 功能(例如提供静态文件)中使用 ASP.NET 功能(例如在第 16 章看到的 Forms Authentication)。关于这种模式的更多信息，可以查看 IIS 的官方网站 `http://tinyurl.com/IntegratedPipelineMode`。

在“试一试”练习结束时配置了默认文档，当请求没有显式文件名的 URL 时(例如 `http://localhost/` 或 `http://localhost/Reviews/`)会提供这个文件。通过将 Default.aspx 配置为默认文档，IIS 就会查找和提供该名称的文件。

为了确保站点在本地 IIS 安装上运行，需要做的最后一件事就是配置安全设置。这将在下面两节中讨论。

19.3.2 了解 IIS 中的安全性

由于 Visual Web Developer 2010 中内置 Web 服务器的无缝集成，您也许不知道内部发生的情况，也不知道在浏览站点中的页面时哪些安全设置有效。为了使用站点内的资源，例如 ASPX 文件、Code

Behind 文件、App_Data 文件夹中的数据库和站点内的图像，Web 服务器需要从 Windows 获得访问这些资源的权限。这就意味着您要配置 Windows，授权 Web 服务器使用的账户访问这些资源的权限。但这个账户是哪一个呢？需要权限的特定账户取决于许多因素，包括 Windows 版本，在 IIS 下运行站点还是使用内置 Web 服务器运行站点，还取决于 IIS 内的许多设置。

然而，在大多数情况下，只需要考虑两种情况：使用内置 Web 服务器还是使用 IIS 作为 Web 服务器。

在前一种情况下，内置 Web 服务器使用的账户是用来登录 Windows 计算机的账户。这个账户通常是 DomainName\UserName 或 MachineName\UserName。在 Windows 上使用这个账户登录，就启动了 VWD 2010，它再启动内置的 Web 服务器。这就意味着整个 Web 服务器都使用凭据运行。由于您可能是本地 Windows 计算机上的 Administrator 或者超级用户，有权限访问组成站点的所有文件，因此到目前为止可能一切正常，不需要修改安全设置。

在后一种情况下，它使用 IIS，所以情况完全不同。在默认情况下，IIS 下的 ASP.NET 应用程序使用在安装 IIS 时创建的特定账户运行。在 Windows Vista 和 Windows Server 2008 R1 中，这个账户命名为 Network Service。而在 Windows 7 和 Windows Server 2008 R2 中则叫做 ApplicationPoolIdentity。

在系统中不能直接找到 ApplicationPoolIdentity 用户账户，因为它取决于配置的应用程序池的名称。

因为前面介绍的 Application Pool 在 Integrated Pipeline 模式下运行，所以只需要配置一个用户账户。如果在 Classic Mode(对于 PlanetWrox 网站，并不是必须这么做)下运行，那么还需要配置另外一个叫做 IUSR 的账户。IIS 使用这个账户来提供非 ASP.NET 的内容，例如 HTML 文件和图像。关于 Classic Mode 和 IUSR 账户的更多信息，可以参考 IIS 文档。

在找到需要配置的账户之后，最后一步就是配置文件系统。

19.3.3 Planet Wrox 的 NTFS 设置

不管使用的是哪个账户，都需要修改 Windows 文件系统，从而允许 Web 服务器访问资源。这只有在使用 NTFS 而不是 FAT 或 FAT32(旧的 Microsoft 文件系统)格式化硬盘驱动器时才有必要。在本节后面的“试一试”练习中将了解如何确定驱动器类型。

要为 Planet Wrox Web 站点成功配置 NTFS 文件系统，还需要将表 19-2 所示的权限授权给在上一节确定的 Web 服务器账户。

表 19-2

文件夹名称	权 限	解 释
Release (位于 C:\BegASPNET\)	List folder contents; Read	Web 服务器账户要能够读取组成 Web 站点的所有文件和文件夹。需要将子文件夹(例如 Reviews)设置为继承这些设置
App_Data GigPics (都位于 C:\BegASPNET\)	Modify; List folder contents; Read; Write	Web 服务器账户要能够读取和写入 App_Data 文件夹中的 Microsoft SQL Server 2008 数据库。还要能够在 GigPics 文件夹中保存上传的图像

如果您从第 12 章跳到本章来学习如何为 App_Data 文件夹配置 NTFS，那么可以忽略本章前面创建的 Release 文件夹。相反，应该按照下一个“试一试”练习的说明，为站点的 App_Data 文件夹

授予自己账户的 Modify 权限。

下面的“试一试”练习将演示如何配置这些文件夹的安全设置。

试一试

配置文件系统

下面的“试一试”练习将演示如何为 Planet Wrox Web 站点配置文件系统。练习中显示的屏幕截图来自 Windows 7，但 Windows 其他版本的屏幕与此类似。如果在实现下列步骤的过程中有什么问题，请在 Windows 帮助中搜索“security NTFS”或联系管理员。

(1) 首先打开 Windows 资源管理器，然后定位到 C 盘驱动器。右击它，并选择 Properties 选项。验证在 File System 下是否已经是 NTFS 文件系统。如果看到旧的 FAT 或 FAT32 文件系统，可以跳过整个“试一试”练习，因为 FAT 不支持修改安全设置。关闭 Properties 对话框。

(2) 在相同的 Windows 资源管理器窗口中，浏览到 C:\BegASPNET\Release 下的 Release 文件夹，如图 19-12 所示。

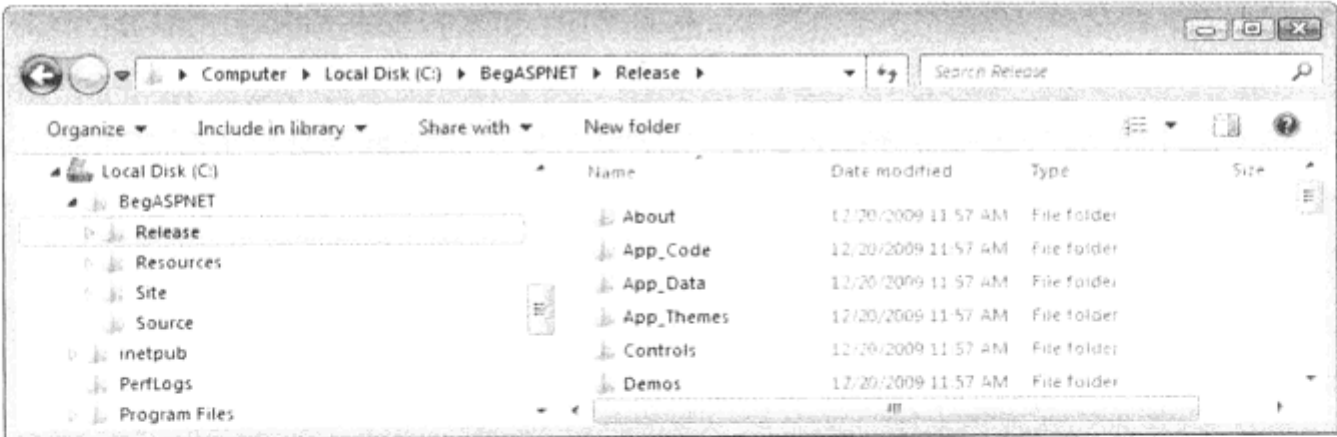


图 19-12

(3) 右击 Release 文件夹，选择 Properties 选项，然后切换到 Security 选项卡，如图 19-13 所示。

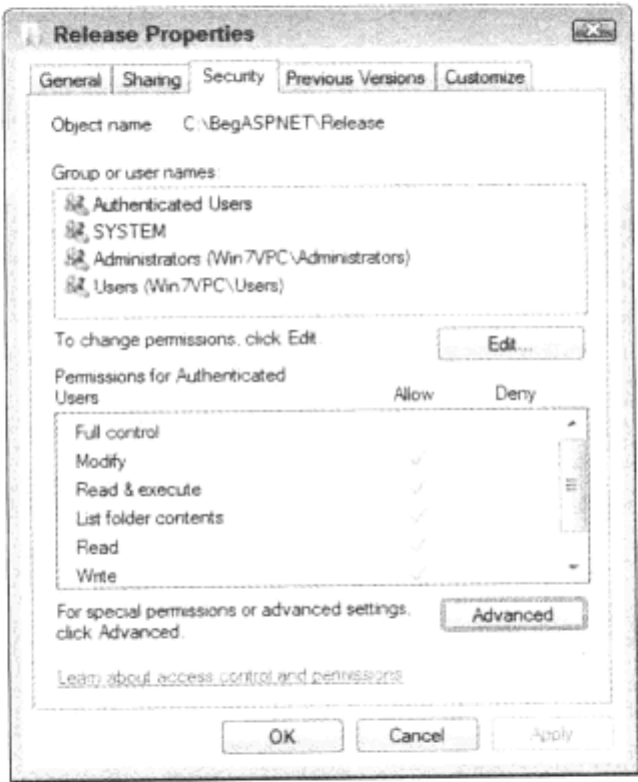


图 19-13

(4) 在默认情况下，Release 文件夹继承其父文件夹(C:\BegASPNET)的设置，父文件夹再继承 C

盘驱动器的根文件夹的设置。要打断这种继承链，可以单击 Advanced 按钮，打开 Advanced Security Settings 对话框。然后对于 Windows 7 和 Windows Server 2008 R2，单击 Change Permissions 按钮；对于 Windows Vista 和 Windows Server 2008，则单击 Advanced Security Settings 对话框上的 Edit 按钮，将屏幕切换为 Edit 模式。应该看到如图 19-14 所示的屏幕。

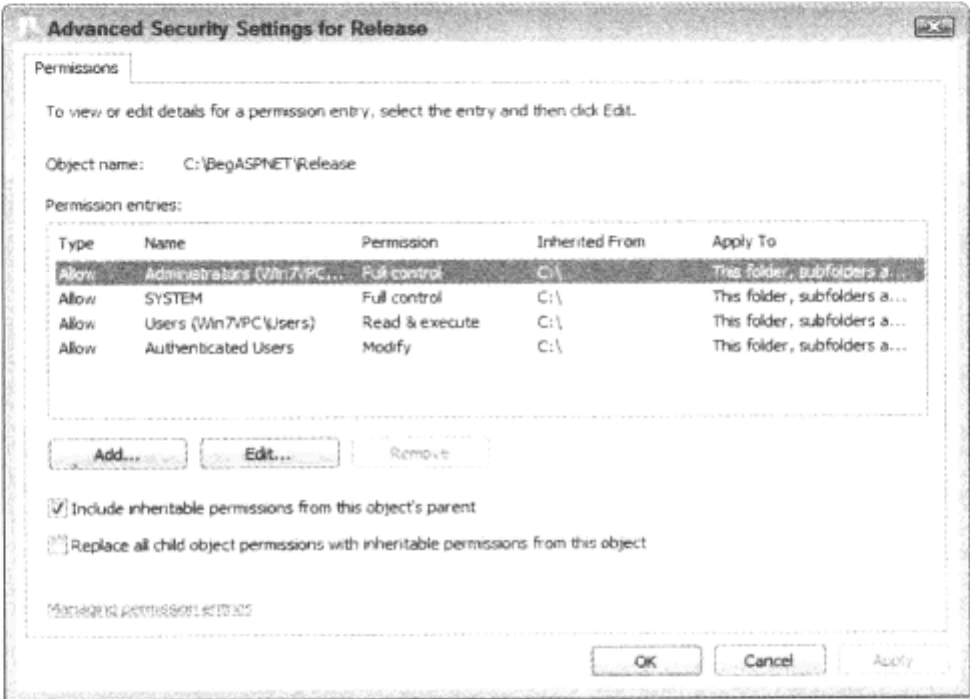


图 19-14

(5) 清除 Include Inheritable Permissions From This Object's Parent 选项，然后单击 OK 按钮。这时会出现一个对话框，它会询问是 Add(Windows 7 和 Windows Server 2008 R2)、Copy(Windows Vista 和 Windows Server 2008)还是 Remove 现有设置。单击 Add 或 Copy 按钮，然后关闭所有对话框，直到回到 Release Properties 对话框为止，如图 19-13 所示。

(6) 下一步是添加账户。单击如图 19-13 所示的 Edit 按钮，然后单击 Add 按钮。根据使用的 Windows 的版本，输入 Web 服务器账户的名称。如果账户是 ApplicationPoolIdentity，则输入 IIS AppPool\ASP.NET v4.0 作为账户名。否则，输入 Network Service。单击 OK 按钮添加账户。

在 Group 或 User Names 列表中选择了账户以后，确保只选择 List Folder Contents 和 Read。对话框最终应该如图 19-15 所示。

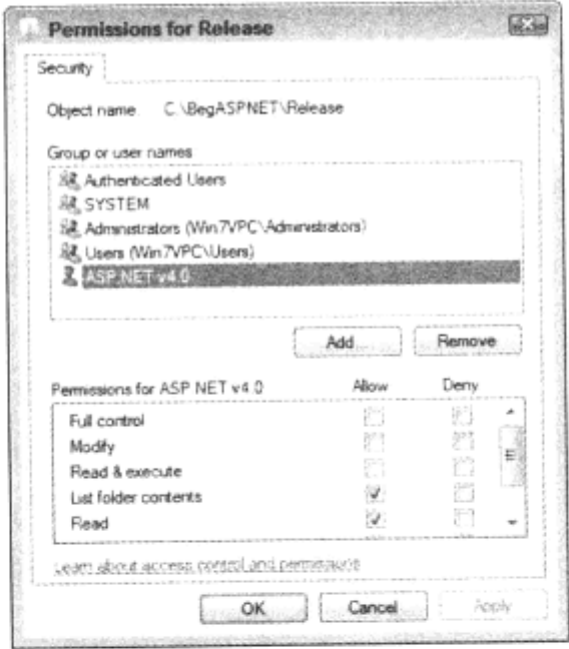


图 19-15

(7) 关闭所有打开的对话框(除 Release Properties 对话框之外)，如图 19-13 所示。

(8) 单击 Advanced 按钮，再次打开 Advanced Security Settings 对话框。对于 Windows 7 和 Windows Server 2008 R2，单击 Change Permissions 按钮；对于 Windows Vista 或 Windows Server 2008，则单击 Advanced Security Settings 对话框中的 Edit 按钮来切换到 Edit 模式。选择图 19-14 中的第 2 个复选框。这样就要求 Windows 将相同的安全设置应用于所有子文件和子文件夹中，以取代全部现有设置。单击 OK 按钮，然后确认要进行的修改。最后，关闭所有仍然打开的对话框。

(9) 返回到 Windows Explorer，右击 Release 文件夹中的 App_Data，打开其 Properties 对话框，然后打开 Security 选项卡，通过添加 Modify 权限(同时会自动选中 Write、Read & Execute)来 edit 第(6)步中添加的 Web 服务器账户的权限。需要先单击 Edit 按钮将 Properties 对话框切换为可编辑模式。对于 GigPics 文件夹而言，请重复这一步。

(10) 如果您是在安装了 SQL Server 2008 Express 的计算机上使用 IIS，现在 Web 站点应该能够运行了。如果使用的是不同的数据库服务器，可以阅读本书后面的 19.4 节，以及阅读解释如何配置一个不同的 SQL Server 数据库的附录 B。要验证这一点，请打开浏览器，并登录 <http://localhost>。您应该会看到 Planet Wrox 网站。要验证一切正常，可以通过从主菜单请求页面、填充联系信息表单、创建新的 Album、上传图片等方法浏览站点。如果出现错误，请参考 19.3.4 一节的内容。



注意：如果 Web 站点还不能运行，请尝试为 Everyone 组配置文件系统。虽然从安全的角度来看，不推荐在生产环境中使用这个组，但它有助于查明这个错误是不是因为安全问题所导致。如果在 Everyone 账户下可以运行站点，那它的确与安全相关，因此需要确保配置正确的账户。不要忘了在后面再次删除 Everyone 账户。

工作原理

在标准 Windows 系统上，都使用 Windows NTFS 文件系统保护所有文件和文件夹。为了确保 Web 站点正确运行，需要给 Web 服务器使用的账户授予必要的权限，以允许他们访问 Web 站点上的文件和文件夹。对于大多数文件和文件夹而言，Read 权限就足够了。然而，对于两个文件夹而言，需要修改这些权限。App_Data 和 GigPics 文件夹在运行时写入，因此需要给账户授予对这两个文件夹的 Modify 和 Write 权限。

19.3.4 检修 Web 服务器错误

当尝试在 Web 浏览器中访问站点时，可能会遇到一些问题。为了获得有帮助的错误消息，所要做的第一件事就是将 web.config 文件中的 <customErrors> 部分从 On 修改为 Off 或 RemoteOnly。这样更容易看到发生的情况。另外，可能还需要检查 Windows Event Viewer(从 Start Menu 中输入 eventvwr)，以获得更多有关错误的详细信息以及有关如何解决它们的提示。

本节列出了一些最常见的问题，并提供了一些解决它们的办法。应该知道产生错误有许多可能的原因，因此这里不可能包含所有原因。如果遇到无法解决的问题，可以访问本书在 Wrox 社区网站上的论坛，网址为 <http://p2p.wrox.com>。您会发现有很多人(包括我自己)理解您的问题，并可以帮助您找到一种解决方案。

- 在应用程序级别以外使用注册为 allowDefinition= 'MachineToApplication' 的节是错误的：当 Web 站点不在 Web 服务器的根文件夹中，或者没有作为单独的应用程序配置文件夹时，就会出现这种错误。假设使用 Planet Wrox 站点的当前配置，当将 IIS 中的站点映射到 C:\BegASPNET 中，然后浏览到 http://localhost/Release 时，就会出现这个错误。要修复这个错误，请确保让 IIS 网站的根文件夹指向包含 web.config 主文件的文件夹；这里是在 C:\BegASPNET\Release 中。当在 VWD 中打开不正确的文件夹时，也会出现这个错误。例如，当打开 C:\BegASPNET，然后浏览到 http://localhost/Site 时。相反，应该打开 C:\BegASPNET\Site 作为 VWD 中的 Web 站点。如果站点内的一个子文件中也包含一个 web.config 文件，而该文件尝试重写应该只在站点根文件夹下定义的设置，那么也会出现这个错误。例如，您在 Management 文件夹中的 web.config 文件中有一个 <membership /> 元素。
- HTTP Error 401.3-Unauthorized: 当 Web 服务器使用的账户没有获得读取磁盘上文件的权限时，就会出现这个错误。要解决这些问题，请参考本章前面 19.3.3 一节中的“试一试”练习，正确配置权限。
- 无法更新数据库“C:\BEGASPNET\RELEASE\APP_DATA\ASPNETDB.MDF”，因为数据库是只读的：当数据库文件标记为只读或者不允许 Web 服务器使用的账户写入数据库文件时，就会出现这种错误。如果是前一种情况，可以在 Windows 资源管理器中打开文件的 Properties 选项，确认清除了 Read Only 复选框。如果是后一种情况，请确保 ASP.NET 使用的账户在 App_Data 文件夹上至少拥有 Modify 权限。
- HTTP Error 403.14-Forbidden: 虽然这个错误似乎首先表示问题与 NTFS 权限有关，但它通常由不正确或遗漏的默认文档产生。如果出现这个错误，请确保访问的站点或文件夹包含名为 Default.aspx 的文档，并且已经将这个文档名配置为 IIS 中的默认文档。
- HTTP Error 404.0-Not Found: 当请求的文件或文件夹不存在时就会出现这个错误，例如 http://localhost/DoesNotExist 或 http://localhost/ DoesNotExist.gif。
- 建立与服务器的连接时出现错误。连接到 SQL Server 2008 时，出现这种失败可能是因为在默认设置下，SQL Server 不允许远程连接。不能找到或者无法访问该服务器。验证实例名称是正确的，并且 SQL Server 配置为允许远程连接。(提供者: Named Pipes Provider, 错误: 40-Could not open a connection to SQL Server)。或者，可能得到这个错误：在建立与 SQL Server 的连接时出现了网络错误或者与特定实例有关的错误。不能找到或者无法访问该服务器。验证实例名称是正确的，并且 SQL Server 配置为允许远程连接。(提供者: SQL Network Interfaces, 错误: 26-Error Locating Server/Instance Specified)：出现这种错误有许多原因。虽然错误消息明确提及 SQL Server 2008，但 SQL Server 2005 也会出现相同的错误。通常，出现这个错误是因为到达配置的数据库服务器的问题。在连接字符串中拼错服务器名称、服务器关闭或者只能从本地计算机到达服务器而不能通过网络连接它，如果出现这些情况也会产生这个错误。为了确保 SQL Server 正确运行，请打开 Administrative Tools 中(在 Control Panel 中可以找到)的 Services 部分。然后查看 SQL Server，验证已经启动 SQL Server。附录 B 详细解释了 SQL Server 的安全性，并提供了这些问题的解决方案。
- 由于无法检索用户的本地应用程序数据路径，所以不能生成 SQL Server 的用户实例。请确保用户在计算机上具有本地用户账户。连接将关闭：当使用 Windows 7 或 Windows Server

2008 R2 时，如果忘记启用在 19.3.1 一节的“试一试”练习中讨论的“Load User Profile”选项，就会出现这种错误。

- HTTP Error 500.21-内部服务器错误处理程序“PageHandlerFactory-Integrated”的模块列表 Detailed Error Information 中有一个出错模块“ManagedPipelineHandler”：在 Windows 7 或 Windows Server 2008 中，当 ASP.NET 没有在 IIS 中注册时，就会出现这个错误。参考 19.3.1.2 一节来了解如何使用 aspnet_regiis 来解决这个问题。
- 运行时错误描述：服务器上产生应用程序错误。这个应用程序的当前自定义错误设置可以阻止查看应用程序错误的详细信息。详细信息：为了使得特定错误消息的详细信息可以在本地服务器计算机上查看，请在当前 Web 应用程序的根目录下的“web.config”配置文件中创建一个<customErrors>标记。然后应该将<customErrors>标记的“mode”特性设置为“RemoteOnly”。为了能够在远程计算机上查看错误的详细信息，请将“mode”设置为“Off”：当发生运行时错误，并且 web.config 文件不包含<customErrors>元素时，就可能会发生这种错误。但是，当 web.config 文件本身包含一个错误时，例如您忘记结束一个元素，也可能会发生这个错误。为了解决后面这种错误，则在 VWD 中打开文件，然后 VWD 会提供关于错误的更多详细信息。
- 不能从程序集‘System.ServiceModel, Version=3.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089’中加载类型‘System.ServiceModel.Activation.HttpModule’：当 IIS 配置为运行.NET 一个早期版本时，将会发生这种错误。为了解决这个问题，按照前面的说明，在命令提示中的 ASP.NET 4 Framework 文件夹中运行 aspnet_regiis -i。

如果要部署到安装了 SQL Server 2008 Express 版本的计算机，现在就可以完成部署过程。然而，如果面对不同的 SQL Server，剩下的唯一一件事就是确保新的 Release 站点有所需的数据。下面将介绍如何实现。

19.4 将数据移动到远程服务器中

将站点发布到本地计算机上的 IIS 中非常简单。只要将数据复制到新位置，配置 IIS，然后修改一些安全设置就行了。因为站点继续使用 SQL Server 2008 Express 版本，所以它会正常运行。

如果需要将站点移动到外部服务器或主机，事情就不那么简单。虽然使用 FTP 程序复制组成站点的文件通常非常简单，但将数据从 SQL Server 2008 数据库复制到主机通常有些复杂。这是因为大多数 Web 主机不支持 SQL Server 2008 Express 版本，因此不能只将.mdf 文件复制到远程主机上的 App_Data 文件夹中。相反，这些主机通常提供 SQL Server 的完全版本，可以使用基于 Web 的管理工具或使用像 SQL Server 2005 和 SQL Server 2008 的 SQL Server Management Studio 这样的工具来访问它们。

为了方便地将数据从本地 SQL Server 2008 数据库传送到 Web 主机的 SQL Server 数据库中，Microsoft 创建了 Database Publishing Wizard。

19.4.1 使用 Database Publishing Wizard

Database Publishing Wizard 允许创建.sql 脚本，它包含在远程服务器上重建数据库及其数据所需

的全部信息。重建数据需要两步：

- (1) 从本地 SQL Server 数据库上创建.sql 脚本；
- (2) 将这个脚本发送到主机，并在那里执行它。

下面的“试一试”练习将演示第一步。这里没有介绍如何在主机上运行脚本，因为它随着主机不同而不同。这里只提供了一些通用的指示，指出在主机上要注意什么。

试一试

导出 Planet Wrox 数据库

可以从 Database Explorer(Visual Studio 付费版中的 Server Explorer)窗口访问 Database Publishing Wizard。完成向导之后，就得到了一个.sql 文件，它包含在不同服务器上重建数据库所需的全部 SQL 代码。

(1) 在 VWD 内加载 Planet Wrox 项目后，选择 View | Database Explorer(或 View | Server Explorer) 选项。

(2) 右击 PlanetWrox.mdf 数据库，选择 Publish to Provider 选项激活 Database Publishing Wizard。如果遇到欢迎屏幕，就单击 Next 按钮，确保已选择数据库，并且已选择数据库中的 Script All Objects，然后单击 Next 按钮。出现的对话框如图 19-16 所示。

(3) 在这个屏幕中，可以在两个选项之间进行选择。第一个选项允许使用所需的 SQL 语句创建文本文件，第二个选项允许通过 Internet 直接与共享的主机提供商会话。如果主机支持这一点，它们可以提供所需的信息在这里建立一个 Provider。但现在，请选择 Script to File 选项，并单击 Next 按钮。

(4) 在下一步中，接受所有默认值(如图 19-17 所示)，然后再单击 Next 按钮。

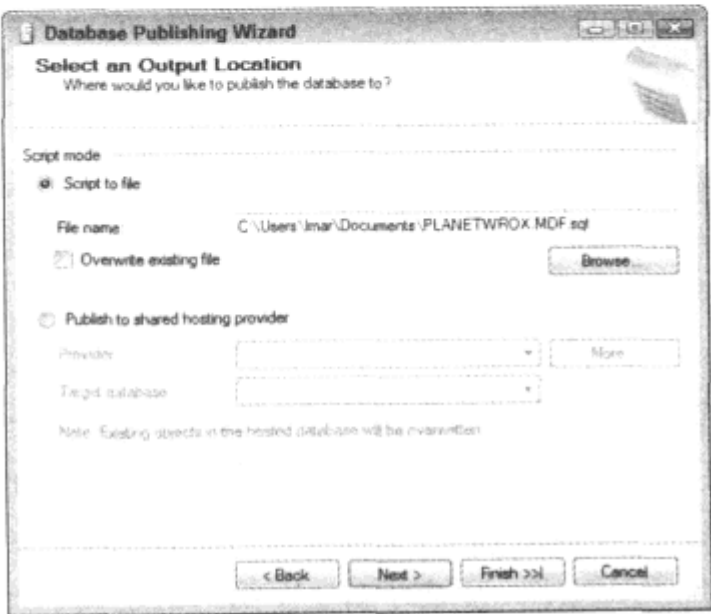


图 19-16

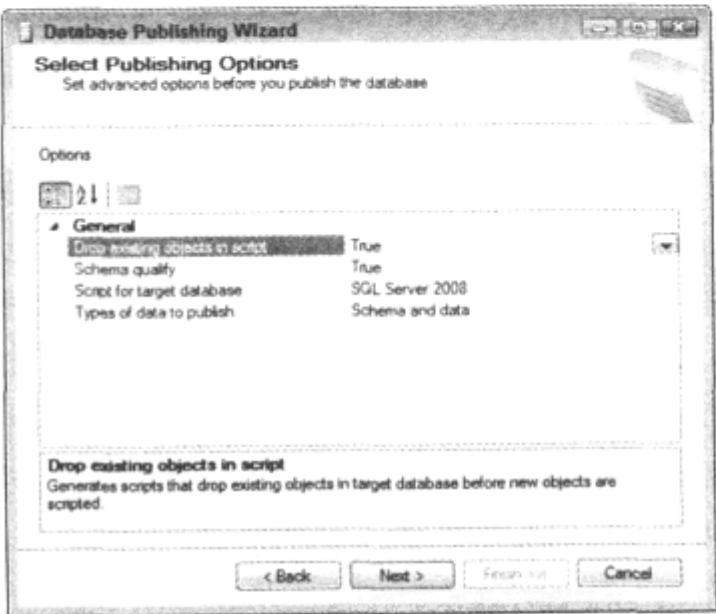


图 19-17

(5) 单击 Finish 按钮，向导会在 Document 文件夹中生成 SQL 脚本。在 Notepad 中打开文件，查看它包含的 SQL 语句。虽然它看起来像乱码，但可以使用它在兼容的 SQL Server 2008 数据库上重建数据库。

工作原理

数据库的内容可以分为两类：数据库的结构和实际数据。Database Publishing Wizard 运行时，它

首先调查数据库的结构，并为它在数据库内找到的所有项(例如前面几章创建的表)创建 SQL CREATE 语句。然后它创建 INSERT 语句，这些语句在目标数据库内重建所有记录，例如 Reviews、Genres 甚至是用户。在向导开始时清除 Script All Objects 复选框，可以有选择性地选择部分数据库，例如，只编写一些表脚本。

最后，向导汇编所有 SQL 语句，并将它们保存到一个.sql 文件。现在可以在主机上运行这个文件来重建数据库。

19.4.2 重建数据库

虽然每个主机在提供对 SQL Server 的访问权时都有自己的规则和程序，但它们可以分为三类：

第一类，有些主机不允许远程访问数据库，它要求提交.sql 文件以便它们执行它。在这种情况下，除了发送文件然后等待主机创建数据库之外，不需要做任何事情。



注意：关于使用外部主机提供商的服务驻留站点的详细信息，可以查看 ASP.NET 官方网站的主机部分提供的教程，网址为 www.asp.net/learn/hosting。

第二类包含的主机允许通过 Web 界面执行 SQL 语句。通常需要登录联机控制面板，然后通过上传文件或者将其内容粘贴到 Web 页面中的文本区，来执行 Database Publishing Wizard 创建的 SQL 语句。不管使用哪种方法，最终都要使用从应用程序可以访问的数据库。其工作原理随着主机的不同而不同，因此更多信息请参考主机服务的帮助或支持系统。有些提供商提供的基于 Web 的数据库管理工具存在一些已知的问题，当运行生成的 SQL 文件时，它们会出现错误。虽然从技术上说文件本身是有效的，但工具使用它时可能会遇到问题。如果出现这种情况，请联系 ISP 帮助解决这个问题。

最后一类包含的主机允许通过 Internet 连接到 SQL Server。这允许使用像 SQL Server Management Studio 这样的工具从桌面连接到主机上的数据库，并远程执行 SQL 脚本。

SQL Server Management Studio 也有免费的 Express 版本，可以从 Microsoft 网站(<http://www.microsoft.com/express/database/>)下载。这个工具的运行原理与它的商业版(SQL Server 2008 付费版本附带)几乎完全相同，对于使用 SQL Server 数据库的人而言，它非常有价值。

在目标服务器上重建数据库之后，需要修改 Web 站点内的连接字符串，以便重新配置 ASP.NET 应用程序，让它使用新的数据库。需要修改三个连接字符串：在前面一章创建的 PlanetWroxConnectionString1 和 PlanetWroxEntities，以及 ASP.NET Application Services 默认使用的 LocalSqlServer 连接字符串。完成这项任务最简单的方法是使用<clear />清除原来的连接字符串，然后添加一个新的连接字符串。连接字符串的形式取决于使用的数据库及其配置。要想了解更多合适的连接字符串的示例，请登录 www.connectionstrings.com 查阅。下面的代码段提供了一个简单示例，它重新配置应用程序，以便使用一个名为 DatabaseServer 的数据库服务器。该示例显示了要求使用用户名和口令登录的 SQL Server 的一个连接字符串 (在配置文件中，每个连接字符串应该放在一行上)。

```
<connectionStrings>
  <clear />
  <add name="PlanetWroxConnectionString1" connectionString="Data
    Source=DatabaseServer;Initial Catalog=PlanetWrox;User Id=YourUserName;
    Password=YourPassword;" providerName="System.Data.SqlClient"/>
  <add name="PlanetWroxEntities" connectionString="metadata=res://*
```



```

/App_Code.PlanetWrox.csdl|res:/**/App_Code.PlanetWrox.ssdl|res:/**
/App_Code.PlanetWrox.msl;provider=System.Data.SqlClient;
provider connection string='Data Source=DatabaseServer;
Initial Catalog=PlanetWrox;User Id=YourUserName;Password=YourPassword;
MultipleActiveResultSets=True';"
providerName="System.Data.EntityClient"/>
<add name="LocalSqlServer" connectionString="Data
Source=DatabaseServer;Initial Catalog=ASPNETDB;User Id=YourUserName;
Password=YourPassword;"providerName="System.Data.SqlClient"/>
</connectionStrings>
```

这将使包含 Reviews 和 Albums 表的 PlanetWrox 数据库以及包含 Membership、Roles 和 Profile 信息的 ASPNETDB 数据库的三个连接字符串都指向不同的 SQL Server。<clear />元素用于删除此前声明的连接字符串，从而允许使用一个不同的连接字符串再次添加 LocalSqlServer 项。有关配置 ASP.NET 应用程序和 SQL Server 来相互运行的更多信息，请参阅附录 B。附录 B 还介绍了将 ASPNETDB 数据库的数据合并到自己的数据库中的方法，如果主机为每个单独的数据库收费，这就特别有用。

现在已经完成了新建的 Web 站点的配置。恭喜您！然而，在开始放松和享受新的 Web 站点之前，请阅读下面的清单，它有助于保护自己的站点，提高其性能。

19.5 部署清单

本章结尾不是关于部署的通用实践技巧，而是提供了一个实用列表，当准备在生产环境中发布 Web 站点时应该检查该列表里面的内容。

- 确保没有在 web.config 文件中激活调试。这会导致不必要的系统开销，并且降低 Web 站点的性能，因为代码执行更缓慢，而且浏览器无法缓存重要文件。要确保不激活调试，请打开要在生产环境中使用的 web.config 文件，验证 debug 是否设置为 false:

```
<compilation debug="false">
```

- 将 web.config 文件中 customErrors 元素的 mode 特性设置为 On 或 RemoteOnly，以确保打开自定义错误。在第一种情况下，每个人都可以看到自定义错误页面；而在第二种情况下，只有 Web 服务器中的本地用户可以看到错误的详细情况。绝不要将 mode 设置为 Off，因为这样会导致信息泄漏。下面的代码段显示了 customErrors 元素的安全配置:

```
<customErrors mode="On" defaultRedirect="~/Errors/OtherErrors.aspx">
Optional <error /> elements go here
</customErrors>
```

- 禁止跟踪，或者至少将跟踪信息限制到本地计算机上的用户。下面来自 web.config 文件的 <trace />元素为来自其他计算机(除 Web 服务器本身之外)的用户阻止了跟踪。另外，它防止跟踪信息在页面中出现:

```
<trace mostRecent="true" enabled="true" requestLimit="1000"
pageOutput="false" localOnly="true"/>
```


- 考虑将 machine.config 文件中部署元素的 retail 特性设置为 true:

```
<configuration>
  <system.web>
    <deployment retail="true"/>
  </system.web>
</configuration>
```

这一部分用来表示服务器驻留的是可以发布的 Web 站点版本，并将前面 3 个项全部修改为安全设置：禁止调试和跟踪，并且只允许本地用户访问错误消息。

要进行这种修改，需要作为 Administrator 登录系统。同时，确保首先备份文件。因为它担当所有 ASP.NET 网站的根配置文件，所以不想这个文件出错。

- 扫描站点，查找可能包含敏感信息的重要文件(例如 Word 或文本文档)，从发布版中删除它们或者考虑将它们移动到 App_Data 文件夹。那个文件夹中的文件不能直接访问。然而，自己的代码仍然可以访问这些文件，如第 9 章所述。
- 确保打开错误日志。使用在第 18 章创建的错误日志代码，发生错误时就会通知您，这样允许主动地监视服务器，并在这些错误再次发生之前修复它们。
- 如果在站点中使用主题，请确保从 web.config 文件的<pages>元素中删除 theme 特性或 styleSheetTheme 特性。Planet Wrox 网站使用主题时，添加 styleSheetTheme 特性是为了在 Visual Web Developer 中激活设计时支持。在生产服务器上所需要的是：

```
<pages theme="Monochrome">
  ...
</pages>
```

这样页面就不会两次都包含相同的样式表。

19.6 补充资源

在完成第一个 ASP.NET 网站之后，确信您还希望创建下一个站点。 Planet Wrox 网站可以作为要构建的新站点的基础。您可能不会在自己的站点中直接使用它的页面，但这本书和 Planet Wrox 网站会激励您自己构建新的 Web 站点。

因为本书面向初学者，所以本书无法提供有关重要主题的大量详细信息。本书中自成一章的一些主题很容易扩充为专门介绍的一本书。例如，像 CSS、AJAX 和 LINQ 这样的主题，内容非常广泛，因此 Wrox 已经出版了许多相关书籍。在掌握这些技术的基本知识之后，还可以阅读下面的 Wrox Professional 丛书进一步研究它们：

- *Professional CSS: Cascading Style Sheets for Web Design, 2nd Edition*(ISBN: 978-0-470-17708-2)
- *Professional ASP.NET 2.0 Design: CSS, Themes, and Master Pages*(ISBN: 978-0-470-12448-2)
- *Professional AJAX 3.5*(ISBN: 978-0-470-39217-1)
- *Professional ASP.NET 4 in C# and VB*(ISBN: 978-0-470-50220-4)

- *Professional LINQ*(ISBN: 978-0-470-04181-9)
- *Professional IIS 7*(ISBN: 978-0-470-09782-3)

当然，网络也是获得更多信息的好地方。下面的 URL 可以帮助你搜索有关 ASP.NET 及其相关技术的更多信息：

- <http://p2p.wrox.com>：Wrox 的公共论坛，可以在里面查找所有与编程有关的问题。在这个站点上本书有自己的类别，允许提出有针对性的问题。我是这个论坛的常客，会尽量回答您对本书提出的每个问题。
- <http://imar.spaanjaars.com>：我自己的网站，在这里我会更新与 Web 编程相关的各种主题。
- <http://www.asp.net>：ASP.NET 技术的 Microsoft 社区站点。那里有 ASP.NET 新闻、其他下载和指南。
- <http://msdn.microsoft.com/asp.net>：Microsoft 开发人员网站上 ASP.NET 的官方主页，提供大量有关 ASP.NET 的信息。

19.7 本章小结

很明显，部署是新 Web 站点开发周期后期的重要操作。然而，不可能只部署站点一次。只要发布了站点的第一个版本，就要考虑添加其他的新功能，让站点开发永不停止。为了实现这一目标，要让站点容易部署。

一种方法是将硬编码的配置设置移动到 web.config 文件中，在开发和生产环境中提供一个修改站点参数的集中位置。

当准备发布站点时，最好给站点创建一个副本，然后在将文件发送到目标服务器之前清理它。使用 Copy Web Site 和 Publish Web Site 命令很容易实现站点的复制和发布。

由于根据 IIS 部署站点，因此需要理解这个 Web 服务器的一些重要设置。本章介绍了配置默认 Web 站点和修改某些配置的方法。由于 Windows 和 IIS 中安全的运行方式问题，还需要配置硬盘驱动器，以便 Web 服务器使用的账户能够读取站点内的文件，以及写入特定文件夹(例如 App_Data 和 GigPics 文件夹)。

本章结尾简单讨论了 Database Publishing Wizard，它是一种工具，允许创建数据库的脚本化副本，可以使用这个副本在远程服务器上恢复数据库。附录 B 将进一步讨论配置和连接到远程数据库服务器这个问题。

19.8 练习

本章没有练习，因为 Planet Wrox 网站现在已经彻底完成。然而，最大的挑战也开始了：使用从本书获得的知识构建 Web 站点。如果使用本书的信息构建了站点并且想与我共享，请通过我的网站联系我，网址为 <http://imar.spaanjaars.com>。祝您愉快！

ASP.NET 4 入门经典——涵盖 C# 和 VB.NET(第 6 版)

本章要点回顾

.NET 程序集	具有.dll 扩展名的文件，包含可执行和可调用的.NET 代码
应用程序池	隔离 IIS 中的(一个或多个)Web 站点以便使它们具有自己的资源集的一种机制
Database Publishing Wizard	VWD 中集成的一种工具，用于从 SQL Server 数据库中导出数据到平面文件或远程数据库中
部署	将 Web 站点从开发环境发布到生产环境的过程
表达式语法	允许将控件属性绑定到不同资源(例如 web.config 文件中定义的应用程序设置)的一种技术
IIS	Internet Information Services——Microsoft 在 Windows 平台上的 Web 服务器
Integrated Pipeline 模式	当在 IIS 中为应用程序池启用了 Integrated Pipeline 模式后，ASP.NET 和 IIS 紧密集成在一起，从而允许为非.NET 资源(如静态文件)使用 ASP.NET 技术
预编译	编译 Web 应用程序到部署到生产服务器的.dll 文件集中的过程。 没有预编译程序，ASP.NET 文件会在第一次被请求时即进行编译
WebConfigurationManager 类	允许访问存储在配置文件中的数据

附录 A

练习答案

A.1 第 1 章

练习 1 答案

在 VWD 中页面标记包含页面原始和未处理的源代码，包括 HTML、ASP.NET 服务器控件和程序代码。然后 Web 服务器处理页面，并把最终的 HTML 发送到浏览器。在浏览器中，这个 HTML 用来呈现用户界面。

练习 2 答案

XHTML 与 HTML 相关，因为 XHTML 是使用 XML 规则重写的 HTML。HTML 更随意，而 XHTML 要求编程人员所写的代码必须符合 XML 规则，例如正确的大小写、结束标记以及引号内的特性。

练习 3 答案

保存常用的 HTML 片段最简单的方法是在文档窗口中选中它们，然后将它们拖动到 Toolbox 上的空白区域。添加项后，可以给它重命名，给它起一个更有意义的名称。现在可以双击一个项，或者当需要它时，将它从 Toolbox 拖动到页面中。

练习 4 答案

有许多方法可以重置所作的自定义修改部分，包括：

- 通过选择 Window | Reset Window Layout 来重新设置窗口布局。
- 通过右击 Toolbox 并选择 Reset Toolbox 选项来重新设置它。

- 使用 Tools | Settings | Import and Export Settings 或者 Tools | Import and Export Settings(取决于使用的 VWD 版本)重新设置 VWD 的所有设置。

练习 5 答案

要修改页面上控件的属性，可以在 Design 视图或 Markup 视图窗口中单击控件，然后使用 Properties 面板(按 F4 键打开)来改变属性的值，也可以直接在 Markup 视图中改变属性。

A.2 第 2 章

练习 1 答案

许多文件都属于 Web Files 类别，包括.aspx 文件(在 Web 浏览器中最终作为页面的 Web 窗体)、.html 文件(包含站点的静态 HTML)、.css 文件(包含层叠样式表信息)和.config 文件(包含 Web 站点的配置信息)。文件的完整列表请参阅第 2 章包含不同文件类型的表。

练习 2 答案

当要让一段文本变成粗斜体时，可以选择文本，然后单击 Formatting 工具栏中的 Bold 按钮，接着单击 Italic 按钮。页面中最终的 HTML 代码如下所示：

```
<strong><em>Welcome to Planet Wrox</em></strong>
```

练习 3 答案

第 1 种方法，使用 Solution Explorer。右击项目，选择 Add Existing Item 选项，然后浏览要添加的项。

第 2 种方法，可以从 Windows 资源管理器或直接从桌面将文件拖放到 VWD 项目中。

第 3 种方法，可以使用 Windows 资源管理器直接将文件放入项目的文件夹中。例如，添加到文件夹 C:\BegASPNET\Site 中的文件会自动变成 Web 站点的一部分。如果在 VWD 中没有直接看到这些新文件，请单击 Solution Explorer 工具栏上的 Refresh 图标。

练习 4 答案

在 VWD 中，代码有 3 种不同视图：Design 视图、Markup 视图(也称为 Source 视图或 Code 视图)和 Split 视图。第 1 个视图提供 Web 页面在浏览器中的呈现方式，而第 2 个视图显示原始标记，Split 视图可以用来同时看到这两个视图。

对于其他文件而言，VWD 也有不同的视图。例如，ASPX 页面的程序代码通常称为 Code Behind 视图或者 Code Behind。

A.3 第 3 章

练习 1 答案

外部样式表最大的好处是它能够应用于整个站点。只要改变这个文件中的一条规则，就可以影

响使用这条规则的站点中的所有页面。如果使用嵌入或内联样式，就要手动改变站点中的所有文件。外部样式表也让站点内的许多不同元素重用某种样式变得更容易。只要创建一个类或 ID 选择器，然后在认为合适的位置重用它们就行了。

练习 2 答案

规则如下所示：

```
h1
{
    font-family: Arial;
    color: Blue;
    font-size: 18px;
    border-top: 1px solid blue;
    border-left: 1px solid blue;
}
```

注意边框属性的另一个缩略版本。这个属性与在本章前面看到的边框属性类似，它允许一次设置边框的大小、样式和颜色。在这条规则里，border-top 和 border-left 只用于改变上边和左边边框；其他两个方向(底部和右边边框)不受此规则影响。

练习 3 答案

第 2 个声明在站点中的可重用性更好，因为它表示 Class 选择器而不是 ID 选择器。在单个页面中，ID 选择器只能被一个页面中具有匹配 id 特性的元素使用一次：这里是 MainContent。另一方面，Class 选择器 BoxWithBorders 可以被单个页面中的多个元素使用，因为允许将相同的 class 特性应用于单个页面中的多个元素。

练习 4 答案

VWD 允许按下面的方法附加样式表：

- 直接在 Markup 视图中页面的<head>里输入所需的<link>元素。
- 将 CSS 文件从 Solution Explorer 拖动到 Markup 视图窗口中页面的<head>部分。
- 从 Solution Explorer 中拖动 CSS 文件，然后在 Design 视图窗口中的页面上释放它。
- 使用主菜单 Format | Attach Style Sheet，然后浏览到 CSS 文件。
- 使用 Manage Styles 或 Apply Styles 窗口，并单击 Attach Style Sheet 链接。

A.4 第 4 章

练习 1 答案

让控件能够保持其状态的机制称为 View State。

练习 2 答案

ASP.NET 运行库在名为_VIEWSTATE 的隐藏字段上保存控件的值。这个隐藏字段与每个回发一起被发送到服务器，在那里它解开包装，然后用前面的值重新填充页面中的控件。

练习 3 答案

DropDownList 控件只允许用户进行单项选择，而 ListBox 允许进行多个选择。另外，当没有展开时，DropDownList 在列表中只显示一个项，而 ListBox 能够同时显示多个项。

练习 4 答案

为了让 CheckBox 控件提交回服务器(当在浏览器中选择或清除它时)，需要将 AutoPostBack 属性设置为 True:

```
<asp:CheckBox ID="CheckBox1" runat="server" AutoPostBack="True" />
```

练习 5 答案

许多 ASP.NET 服务器控件都允许使用属性(例如 BackColor 和 ForeColor)来改变颜色。另外，可以使用 BorderColor、BorderStyle 和 BorderWidth 改变浏览器中控件周围的边框。最后，如果要改变控件的大小，还要设置它的 Height 和 Width 属性。

练习 6 答案

除了设置单个样式属性之外，最好也设置一下指向规则集的 CssClass 属性。这样页面就变得更易于维护，因为与样式相关的信息保存在一个位置：样式表中。同时，页面加载会变得更快速，因为不需要为每个请求中的每个控件发送全部的样式信息；而是浏览器读入样式表一次，然后保存一个本地缓存的副本。

A.5 第 5 章

练习 1 答案

Byte 和 SByte 数据类型都能保存小的数字值。这两者占用的计算机内存也完全相同，因此最好使用 Byte 数据类型。因为它不允许保存负数，所以一开始就很清楚，它只能包含 0 到 255 之间的数字。然而，最好不要使用它保存某人的年龄，但可以保存出生日期。这样就总是可以用出生日期计算年龄(将出生日期与今天的日期进行比较)。因为出生日期在时间上是固定的，所以它总是可以正确地反映人的年龄，而不需要每年更新。

练习 2 答案

这段代码可切换 DeleteButton 控件的可见性。它使用了赋值运算符和取反运算符。首先，将逻辑非运算符应用到 Visible 的当前值上。当这个值现在为 True 时，Not(C#中是!)运算符将它转换为 False；而值为 False 时，则转换为 True。然后，将这个结果再赋值给 Visible 属性。所以，如果按钮之前是隐藏的，那么现在就可见。如果之前是可见的，那么现在就隐藏。

练习 3 答案

要创建 Person 的专用版本，需要创建一个继承自 Person 类的辅助类，并通过添加 StudentId 属性来扩展它的行为。

VB.NET

```
Public Class Student
    Inherits Person
    Public Property StudentId As String
End Class
```

C#

```
public class Student : Person
{
    public string StudentId { get; set; }
}
```

A.6 第 6 章

练习 1 答案

ContentPlaceHolder 元素应该放在母版页中，它定义了内容页可以填充的区域。Content 控件应该放在基于母版页的内容页中，它用来为连接它的 ContentPlaceHolder 元素提供内容。

练习 2 答案

要将 Content 控件链接到母版页中的 ContentPlaceHolder，需要设置 ContentPlaceHolderID:

```
<asp:Content ID="Content1" ContentPlaceHolderID="IdOfContentPlaceHolder"
    Runat="Server">
</asp:Content>
```

练习 3 答案

有一些方法可以实现。首先，可以使用一个不同的 CSS 类在相同的 skin 文件中创建一个已命名外观。

```
<asp:Button runat="server" SkinID="RedButton" CssClass="RedButton" />
```

然后使用 SkinID 特性将这个已命名外观与要改变的控件关联起来。

```
<asp:Button ID="Button1" runat="server" Text="Button" SkinID="RedButton" />
```

同样，可以在 Button 控件上禁止主题，并直接在 ASPX 页面中给它提供一个不同的 CSS 类。

```
<asp:Button ID="Button1" runat="server" EnableTheming="False"
    CssClass="RedButton" Text="Button" />
```

在这两种答案中，需要一个 CSS 类来设置背景颜色:

```
.RedButton
{
    background-color: Red;
}
```

练习 4 答案

在页面生命周期的早期应用 `StyleSheetTheme`。这让 ASPX 页面中的控件有机会重写它们最初从 `StyleSheetTheme` 中获得的设置。这意味着 `StyleSheetTheme` 只是建议控件的外观，给单个控件提供重写该外观的能力。而 `Theme` 则重写控件应用的所有设置。这允许页面开发人员使用这个主题的设置实现站点内页面的外观。如果还要改变单个控件，可以通过将 `EnableTheming` 设置为 `False` 来禁用主题。

练习 5 答案

有 3 种方法可以在 ASP.NET 4 Web 站点内设置 `Theme` 属性。第一种选择是直接在 `@ Page` 指令中设置这个属性，让它只应用于该页面：

```
<%@ Page Language="C#" ..... Theme="Monochrome" %>
```

要将该主题应用于站点内的所有页面，可以设置 `web.config` 文件中 `<pages>` 元素的 `theme` 特性：

```
<pages theme="Monochrome">
```

最后一种选择是通过编程来设置这个主题。必须在 `Page` 类的 `PreInit` 事件中完成，可以使用 `BasePage` 类在站点内或在中心位置的单个页面中操作，如本章所述。

练习 6 答案

基页允许集中站点内所有页面的行为。不是在每个页面中重复地重新编码相同的功能，而是可以将这个代码移动到基页中，这样所有的 ASPX 页面都可以使用它。当实现主题转换器时，需要做的就是中心 `BasePage` 类中写一些代码。站点内从该 `BasePage` 类继承的所有页面都会正确设置选中的主题，而不再需要其他代码。

A.7 第 7 章

练习 1 答案

要改变 `TreeView` 控件中的项的背景色，可以像下面这样使用 `NodeStyle` 元素：

```
<asp:TreeView ID="TreeView1" runat="server"
    DataSourceID="SiteMapDataSource1" ShowExpandCollapse="False">
    <NodeStyle BackColor="White" />
    ...
</asp:TreeView>
```

不要设置 `BackColor` 属性(这样会产生内联样式)，而是最好设置 `CssClass` 属性：

```
<NodeStyle CssClass="TreeViewNodeStyle" />
```

然后需要在 CSS 文件中创建一个单独的类：

```
.TreeViewNodeStyle
{
```



```
background-color: White;
}
```

这样就更容易从一个集中位置管理样式。

练习 2 答案

要通过编程将用户重定向到另一个页面，可以使用 `Response.Redirect`、`Response.RedirectPermanent` 和 `Server.Transfer`。前两个选项将重定向指令发送到浏览器，因此被认为是客户端重定向。另一方面，`Server.Transfer` 在服务器上发生，允许提供不同的页面，同时又不影响用户的地址栏。

练习 3 答案

要显示不能展开或折叠节点的 `TreeView` 控件，需要将 `TreeView` 的 `ShowExpandCollapse` 属性设置为 `False`。

A.8 第 8 章

练习 1 答案

标准属性使用正常的后台变量来保存它的值，而 `View State` 属性使用 `ViewState` 集合来保存值。正常属性在每个回发上重置，而 `View State` 属性能够维护它的值。然而，`View State` 属性的这种优点是有代价的。在 `View State` 中保存值增加了页面的大小，不管是在请求过程中还是在响应过程中。正常属性没有这种不足。应该仔细考虑在 `View State` 中保存的内容，以最小化页面大小。

练习 2 答案

要让属性在各回发之间维持其状态，需要将它转换为 `View State` 属性。所需的代码几乎与 `NavigateUrl` 的代码相同，但它使用的是 `Direction` 数据类型而不是字符串类型。需要删除自动属性，再用下面的代码取代它：

VB.NET

```
Public Property DisplayDirection() As Direction
    Get
        Dim _displayDirection As Object = ViewState("DisplayDirection")
        If _displayDirection IsNot Nothing Then
            Return CType(_displayDirection, Direction)
        Else
            Return Direction.Vertical ' Not found in View State; return a default value
        End If
    End Get
    Set(ByVal Value As Direction)
        ViewState("DisplayDirection") = Value
    End Set
End Property
```

C#

```
public Direction DisplayDirection
{
```

```
get
{
    object _displayDirection = ViewState["DisplayDirection"];
    if (_displayDirection != null)
    {
        return (Direction)_displayDirection;
    }
    else
    {
        return Direction.Vertical; // Not found in View State; return a default value
    }
}
set
{
    ViewState["DisplayDirection"] = value;
}
}
```

练习 3 答案

与使用数字或字符串数据类型相比，使用自定义数据类型(例如 Direction 枚举)有两个好处。鉴于 IntelliSense 帮助选择正确项的方法，不必记住像 0 或 1 这样的数字或字符串。另外，编译器也会帮助检查拼写，因此如果输入 Direction.Vrtical 而不是 Direction.Vertical，在开发时就会出现错误。

A.9 第 9 章

练习 1 答案

首先，需要在用户控件的 Code Behind 中编写属性，与第 8 章为 Banner 控件创建的 DisplayDirection 属性类似。该属性如下所示：

```
VB.NET
Public Property PageDescription As String

C#
public string PageDescription { get; set; }
```

然后需要修改控件声明。例如在 Contact.aspx 中，可以像下面这样修改控件：

```
<uc1:ContactForm ID="ContactForm1" runat="server"
    PageDescription="Contact Page"/>
```

注意，PageDescription 属性包含对所含页面的简短描述。很明显，可以将任何认为适合描述页面的文本放到属性中。

最后需要将 PageDescription 添加到电子邮件消息的主题或正文中。下面的代码段显示如何使用新属性的值扩展主题：

VB.NET

```
myMessage.Subject = "Response from web site. Page: " & PageDescription
myMessage.From = New MailAddress("you@yourprovider.com", "Sender Name")
```

C#

```
myMessage.Subject = "Response from web site. Page: " + PageDescription;
myMessage.From = new MailAddress("you@yourprovider.com", "Sender Name");
```

从现在开始，自定义页面描述已经被添加到邮件消息的主题之中。

练习 2 答案

如果没有检查 IsValid 属性，系统就很容易受到无效数据的攻击。用户在他们的浏览器中可以禁用 JavaScript，并直接将无效数据提交给页面。通过检查 IsValid 属性，就可以知道继续使用提交的数据是否安全。

练习 3 答案

MailMessage 类的 From 属性是 MailAddress 类型，也就是说可以直接将这个类的单个实例分配给它。由于可能有多个接收方，所以 To 属性是 MailAddress 对象的集合，因此需要使用 Add 方法将 MailAddress 的实例添加到 To 属性中。

练习 4 答案

要调用客户端验证函数，需要像下面这样设置 CustomValidator 控件的 ClientValidationFunction 属性：

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
    ClientValidationFunction="FunctionName" >*</asp:CustomValidator>
```

要添加到页面标记中的客户端函数必须有如下签名：

```
function FunctionName(source, args)
{ }
```

source 参数包含对客户端 HTML 代码中实际 CustomValidator 控件的引用。args 参数提供与数据有关的上下文信息，并允许指出数据是否有效。参数名称不一定必须是 source 和 args；然而，当使用这些名称时，客户端函数与它的服务器端对应函数看起来十分接近。另一种通用命名方案(几乎可用于 ASP.NET 中其他所有事件处理程序)是分别使用 sender 和 e。

练习 5 答案

要告诉验证机制检查的数据是否有效，需要在自定义验证方法中设置 args 参数的 IsValid 属性。这既适用于客户端代码，也适用于服务器端代码。下面的代码段显示了在客户端验证方法中为 ContactForm.ascx 控件实现这些的方法：

```
if (phoneHome.value != '' || phoneBusiness.value != '')
{
    args.IsValid = true;
}
```



```
    }
    else
    {
        args.IsValid = false;
    }
}
```

A.10 第 10 章

练习 1 答案

在几乎所有与 Ajax 相关的操作中，ScriptManager 控件都是必不可少的组件。它负责注册客户端 JavaScript 文件，处理与 Web 站点中定义的 Web 服务的交互，还负责部分页面的更新。如果认为只在小部分页面上需要使用 Ajax，通常可以将 ScriptManager 控件直接放置到内容页中。然而，也可以将 ScriptManager 放置在母版页中，这样它在整个站点中都可用。

当母版页中有 ScriptManager 控件时，就可以使用 ScriptManagerProxy 在内容页上注册单个 Web 服务或脚本文件。由于在一个页面中只能有一个 ScriptManager 控件，因此不能在使用具有 ScriptManager 的母版页的内容页中添加另一个 ScriptManager。ScriptManagerProxy 是内容页和 ScriptManager 之间的桥梁，它为在哪里注册服务提供了极大的灵活性。

练习 2 答案

将 UpdateProgress 控件添加到页面，就可以让用户知道部分页面更新正在进行。也可以使用该控件的 AssociatedUpdatePanelID 属性将它连接到 UpdatePanel。在<ProgressTemplate>元素内部，可以定义认为合适的标记来告诉用户更新正在进行。典型的<ProgressTemplate>包含一个动画图标，一些文本，或者这两者都有。

练习 3 答案

要创建脚本可以调用的 Web 服务，首先要使用 Add New Item 对话框将 Web 服务文件添加到站点中。创建的 Web 服务继承自 System.Web.Services.WebService，该基类定义所有 Web 服务的默认行为。

当创建 Web 服务时，需要给它添加 ScriptService 特性：

VB.NET

```
<System.Web.Script.Services.ScriptService()> _
<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class NameService
    Inherits System.Web.Services.WebService
```

C#

```
[System.Web.Script.Services.ScriptService]
public class NameService : System.Web.Services.WebService
```

最后，还要使用 WebMethod 特性修饰这个类中想要作为 Web 方法来提供的所有方法：

VB.NET

```
<WebMethod()>
Public Function HelloWorld(ByVal yourName As String) As String
```

C#

```
[WebMethod]
public string HelloWorld(string yourName)
```

练习 4 答案

- 要想提供和使用一个页面方法，需要执行下面的步骤：
- (1) 将 ScriptManager 控件的 EnablePageMethods 设为 True。
 - (2) 在页面中定义一个 static(VB.NET 中是 Shared)方法，并像下面这样应用 WebMethod 特性：

VB.NET

```
<WebMethod()>
Public Shared Function HelloWorld(ByVal yourName As String) As String
    ...
End Function
```

C#

```
[WebMethod]
public static string HelloWorld(string yourName) { ... }
```

- (3) 通过 PageMethods 对象调用 JavaScript 中的方法，并定义一个回调方法来处理这个方法的结果：

```
PageMethods.HelloWorld('Some Value', HelloWorldCallback);
```

A.11 第 11 章

练习 1 答案

为了完成这个任务，首先需要将 VerticalPanel 的 ClientIDMode 设为 Static。这样就更容易在面板的内部使用控件的客户端 ID：

```
<asp:Panel ID="VerticalPanel" runat="server" ClientIDMode="Static">
```

然后需要在 VerticalPanel 的下面添加一个带有 Hide Banner 文本的 span 元素。当鼠标悬停在文本上时，style 特性将光标改为手形，这样用户就知道这些文本是可以单击的。需要添加一个 runat 特性和一个服务器端 ID，这样就可以在 Code Behind 中隐藏链接。因为使链接变为服务器端链接会改变客户 ID，所以还需要将 ClientIDMode 设为 Static。得到的 span 元素如下所示：

```
<span id="HideBanner" style="cursor: pointer;" runat="server" ClientIDMode="Static">
Hide Banner</span>
```

在最后一个面板下的脚本块中，需要添加下面的代码：

```
<script type="text/javascript">
$(function()
{
    $('#HideBanner').bind('click', function()
    {
        $('#VerticalPanel').slideToggle();
        if ($(this).css('display') == 'block')
        {
            $('#HideBanner')[0].innerText = 'Hide Banner';
        }
        else
        {
            $('#HideBanner')[0].innerText = 'Show Banner';
        }
    });
});
</script>
```

这段代码将一些代码动态地绑定到使用\$('#HideBanner')找到的 span 元素的 click 事件上。然后，在单击事件处理程序的内部，代码找到整个面板(#VerticalPanel)并调用 slideToggle，这会在匹配集中切换项的显示状态(当项是可见时，隐藏该项；不可见时，显示项)。然后 if 使用 HideBanner span(现在使用 this 关键字引用)的 css 方法并请求其 display 属性。如果为 block，则横幅可见，所以文本一定是 Hide Banner。否则，文本被设为 Show Banner。

最后，当横幅处于水平模式时，需要隐藏链接：

VB

```
Case Direction.Horizontal
    HorizontalPanel.Visible = True
    HorizontalLink.HRef = NavigateUrl
    HideBanner.Visible = False
```

C#

```
case Direction.Horizontal:
    HorizontalPanel.Visible = true;
    HorizontalLink.HRef = NavigateUrl;
    HideBanner.Visible = false;
    break;
```

练习 2 答案

slideUp 方法通过逐渐降低元素的高度来隐藏它们。slideDown 通过逐渐增加元素的高度，直至其完全可见来显示元素。除了其他的许多参数以外，两种方法都接受一个速度参数，这个速度参数可以是一个定值(慢、中或快)，或者是一个指定了动画的播放速度(单位为毫秒)的数字。

练习 3 答案

只有在第一次请求页面时，页面在浏览器中完成加载后，jQuery 的文档就绪函数才会触发。而 pageLoad 方法会在页面第一次加载时或者异步回发(例如，使用 UpdatePanel1)后触发。这种区别可

以用来选择期望的行为。需要在第一次加载时和回发后触发代码则选择 `pageLoad`。否则，选择 jQuery 的文档就绪函数。

A.12 第 12 章

练习 1 答案

DELETE 语句失败，是因为 Genre 表的 Id 和 Review 表的 GenreId 之间有关系。只要这种关系有效，就不能删除仍有评论与之关联的流派。为了能够删除请求的流派，需要首先删除相关的评论，或者使用 UPDATE 语句将它们指派给不同的流派。具体方法参见练习 4。

练习 2 答案

Genre 和 Review 表之间的关系是单向关系。这种关系强调指派给评论的 GenreId 必须在 Genre 表中作为 ID 存在。同时，它阻止删除仍有评论与之关联的流派。然而，这种关系不会阻止删除 Review 表中的记录。

练习 3 答案

要删除 Id 为 100 或更小的评论，需要下面的 SQL 语句：

```
DELETE FROM Review WHERE Id <= 100
```

练习 4 答案

在能够删除流派之前，首先要将已有评论重新指派给新的流派。可以使用下面的 UPDATE 语句完成：

```
UPDATE Review SET GenreId = 11 WHERE GenreId = 4
```

该代码将 GenreId 为 11 的流派分配给前面将 GenreId 设置为 4 的所有评论。这也意味着 ID 为 4 的流派不再有任何评论与之关联，因此可以使用下面的 SQL 语句删除这个流派：

```
DELETE FROM Genre WHERE Id = 4
```

练习 5 答案

要更新名字，需要执行一个 UPDATE 语句。要将受影响的记录限制为 Rock 流派，需要使用 WHERE 子句。可以基于流派的 Id 或 Name，使用 WHERE 子句筛选记录。下面的 SQL 语句在功能上是等效的：

```
UPDATE Genre SET Name = 'Punk Rock' WHERE Id = 7
UPDATE Genre SET Name = 'Punk Rock' WHERE Name = 'Rock'
```

这段代码假定当前的 Rock 流派的 Id 为 7。

A.13 第 13 章

练习 1 答案

这种情况下最好的控件是 GridView。它很容易设置，并且内置支持分页、更新和数据删除。与 DetailsView 控件一起使用，就可以给用户提供全部 4 种 CRUD 操作。要连接数据库，需要使用 SqlDataSource 控件。第 14 章提供了 GridView 和 SqlDataSource 控件的可替代方法。

练习 2 答案

对于无序的简单列表，最好使用与 SqlDataSource 控件关联的 Repeater 控件。Repeater 控件最大的优点是它自己不产生 HTML 代码，从而允许控制最后的标记。这个控件的缺点是它不支持数据编辑和删除操作，但如果只想以列表形式呈现数据，那么这就不是问题。第 14 章介绍了如何使用 Repeater 控件。

练习 3 答案

BoundField 直接与数据源中的列相关联，并且只提供了有限的方法来自定义它的外观。另一方面，TemplateField 可以用来完全控制呈现这个字段的方法。因此，对于更复杂的场景而言这是一个理想字段，例如，当要将验证控件添加到页面时；或者想要让用户使用不同控件时，如 DropDownList 而不是默认的 TextBox。

练习 4 答案

应该总是将连接字符串存储在 web.config 文件中。这个文件有一个元素为 <connectionStrings>，它专门用于存储连接字符串。将它们存储在 web.config 文件中，可以非常轻松地找到连接字符串并修改它们。如果在页面级别存储它们，就必须仔细搜索整个项目查找相关的连接字符串。

可以使用表达式绑定语法访问连接字符串。例如，要在 SqlDataSource 中设置连接字符串，就可以使用下面的代码：

```
ConnectionString="<%= ConnectionStrings:PlanetWroxConnectionString1 %>"
```

要想使这段代码生效，需要在 web.config 文件中使用与下面类似的连接字符串：

```
<connectionStrings>
  <add name="PlanetWroxConnectionString1" connectionString="Data
    Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\PlanetWrox.mdf;
    Integrated Security=True;Connect Timeout=30;User Instance=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

A.14 第 14 章

练习 1 答案

要从系统中获得最近的 10 个评论，查询需要两个重要的 LINQ 结构：首先它需要使用 Order By

(在 C#是 orderby)子句来按降序排列列表。然后使用 Take 方法从结果集中获得前 10 个评论：

VB.NET

```
Using myEntities As New PlanetWroxEntities()  
    Dim recentReviews = (  
        From myReview In myEntities.Reviews  
        Order By myReview.CreateDateTime Descending  
        Select New With {myReview.Title, myReview.Genre.Name}  
    ).Take(10)  
  
    GridView1.DataSource = recentReviews  
    GridView1.DataBind()  
End Using
```

C#

```
using (PlanetWroxEntities myEntities = new PlanetWroxEntities())  
{  
    var recentReviews = (from myReview in myEntities.Reviews  
        orderby myReview.CreateDateTime descending  
        select new { myReview.Title, myReview.Genre.Name }).Take(10);  
  
    GridView1.DataSource = recentReviews;  
    GridView1.DataBind();  
}
```

这段代码还使用 New 关键字(在 C#中是 new)来创建一个新的匿名类型,它只包含评论的标题和流派名称。

练习 2 答案

ListView 控件最大的优点是它结合了其他这些数据控件的性能。就像 GridView 控件一样,ListView 控件使得以网格形式(用户可以在网格内编辑)显示数据变得更容易。另外,ListView 控件允许插入新记录,像 DetailsView 和 FormView 这样的控件也有这种行为,但是 GridView 则没有。

最后,ListView 控件允许完全控制发送到浏览器的标记,这是一个重要的功能,只有 Repeater 控件能够立即使用。

练习 3 答案

首先需要改变 PhotoAlbums 文件夹中的 Default.aspx 页面,让它将每个缩略图链接到详细信息页面,并将图片的 ID 传递给这个新页面。在 Default.aspx 文件中 ListView 控件的<ItemTemplate>元素中,在已经存在的 Image 控件周围添加这个 HyperLink 控件:

```
<asp:HyperLink ID="HyperLink1" runat="server"  
    NavigateUrl='<%# "PictureDetails.aspx?Id=" + Eval("Id").ToString() %>'>  
<asp:Image ID="Image1" runat="server" ImageUrl='<%# Eval("ImageUrl") %>'>  
    ToolTip='<%# Eval("ToolTip") %>' />  
</asp:HyperLink>
```

注意,使用静态文本 PictureDetail.aspx?Id=和数据库中图片的 ID 构建 NavigateUrl。然后创建一个名为 PictureDetails.aspx 的新页面,并在标记中添加一个 Image 控件:


```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
  <asp:Image ID="Image1" runat="server" />
</asp:Content>
```

最后，需要在 Code Behind 里页面的 Load 事件中执行下面的 LINQ 查询来设置 ImageUrl:

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    Dim pictureId As Integer = Convert.ToInt32(Request.QueryString.Get("Id"))
    Using myEntities As New PlanetWroxEntities()
        Dim imageUrl As String = (From picture In myEntities.Pictures
                                   Where picture.Id = pictureId
                                   Select picture).Single().ImageUrl
        Image1.ImageUrl = imageUrl
    End Using
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    int pictureId = Convert.ToInt32(Request.QueryString.Get("Id"));
    using (PlanetWroxEntities myEntities = new PlanetWroxEntities())
    {
        string imageUrl = (from picture in myEntities.Pictures
                           where picture.Id == pictureId
                           select picture).Single().ImageUrl;
        Image1.ImageUrl = imageUrl;
    }
}
```

这段代码首先从查询字符串中获得图片的 ID，然后将它填充到 LINQ 查询中。Single 方法用来从 Picture 表中检索单个图片，其 ImageUrl 用来在浏览器中显示图像。

练习 4 答案

EntityDataSource 控件的 Deleted 事件是从磁盘删除图像的最佳位置，该事件在从数据库中删除某个数据项后触发。在该事件的处理程序内，获取对被删除的 Picture 的一个引用，找到其 ImageUrl，将该 URL 转换为一个物理位置，然后删除图像。为了实现这个目的，执行下面的步骤：

- (1) 在 Design 视图中打开站点根文件夹中的 ManagePhotoAlbum.aspx，选择 EntityDataSource 控件，打开其 Properties 面板，并切换到 Events 选项卡。
- (2) 双击 Deleted 事件，在 Code Behind 中，将下面的代码添加到 VWD 在 ListView 控件的 Properties 面板的 Events 选项卡中添加的处理程序中。

VB.NET

```
Dim myPicture As Picture = CType(e.Entity, Picture)
Dim fileName As String = Server.MapPath(myPicture.ImageUrl)
System.IO.File.Delete(fileName)
```

C#

```
Picture myPicture = e.Entity as Picture;
string fileName = Server.MapPath(myPicture.ImageUrl);
System.IO.File.Delete(fileName);
```

(3) 将 ImageUrl 添加到 ListView 控件的 DataKeyNames 列表中:

```
<asp:ListView ID="ListView1" ... DataKeyNames="Id,ImageUrl" ...>
```

这段代码首先使用 e.Entity 找到对 Picture 的引用。然后获取其虚拟的 ImageUrl，将其转换为一个物理路径，然后使用 File 类的 Delete 方法从磁盘上删除该图片。因为整张图片都是从 View State 中构建的，并且没有在 EF 中请求，所以需要将其 ImageUrl 存储到 DataKeyNames 属性中。这样会将它添加到 View State 中，使其传递给浏览器，并且可以在 Deleted 事件中使用。

练习 5 答案

为了只显示至少有一个评论的流派，所要做的就是 Where 中像下面这样使用 Count 方法来筛选掉空流派:

VB.NET

```
Dim allGenres = From genre In myEntities.Genres
                  Order By genre.Name
                  Where genre.Reviews.Count() > 0
                  Select New With {genre.Name, genre.Reviews}
```

C#

```
var allGenres = from genre in myEntities.Genres
                  orderby genre.Name
                  where genre.Reviews.Count() > 0
                  select new { genre.Name, genre.Reviews };
```

A.15 第 15 章

练习 1 答案

Page 的 Load 事件总是在用户触发的事件(例如 Button 控件的 Click 事件)之前触发。

练习 2 答案

为了在用户插入或者编辑评论之后在下拉列表中预先选择正确的项，需要做两个修改。首先，需要在页面 AddEditReviewHandCoded.aspx 中修改 Redirect 语句，使其包含流派的 ID:

VB.NET

```
Response.Redirect(String.Format("Reviews.aspx?GenreId={0}",
                                GenreList.SelectedValue))
```

C#

```
Response.Redirect(string.Format("Reviews.aspx?GenreId={0}",
```

```
GenreList.SelectedValue));
```

与使用普通的字符串连接(在 VB.NET 中使用&, 在 C#中使用+)相比, 使用 String.Format 使得代码更容易阅读。

如果现在插入或者编辑新评论, 可以看到流派的 ID 被传递回 Reviews.aspx 页面。在该页面上, 可以使用该 ID 在 DropDownList 控件中预先选择正确的项, 这可以通过在 Page_Load 方法中使用下列代码实现:

VB.NET

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.Load
    If Not Page.IsPostBack Then
        Dim genreId As String = Request.QueryString.Get("GenreId")
        If Not String.IsNullOrEmpty(genreId) Then
            DropDownList1.DataBind()
            Dim myItem As ListItem = DropDownList1.Items.FindByValue(genreId)
            If myItem IsNot Nothing Then
                myItem.Selected = True
            End If
        End If
    End If
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        string genreId = Request.QueryString.Get("GenreId");
        if (!string.IsNullOrEmpty(genreId))
        {
            DropDownList1.DataBind();
            ListItem myItem = DropDownList1.Items.FindByValue(genreId);
            if (myItem != null)
            {
                myItem.Selected = true;
            }
        }
    }
}
```

只有当新请求(而不是回发)的页面加载时, 这段代码才会触发。然后代码会尝试从查询字符串中找到 GenreId。如果能找到, 它就会尝试在 DropDownList 中找到具有请求值的项。因为 DropDownList 控件还不是数据绑定的, 所以它不包含任何数据项。因此, 需要首先调用 DataBind() 方法。这会使用 EF 从数据库中获取流派, 并把它们放到 DropDownList 中。一旦操作完成, 并且可以从 Items 集合中找到该项, 就通过将其 Selected 属性设为 true 来使其成为活动的项。SqlDataSource 控件查看这个 DropDownList, 所以当数据源获取其评论后, 它就会为正确的流派执行这些操作。

练习 3 答案

各种数据绑定控件都可能产生异常，可以在它们的事件处理程序中处理这些异常。在正确处理了这些异常之后，还需要将 e 参数的 ExceptionHandled 属性设置为 True。下面的代码段显示了如何使用错误消息更新 Label 控件。然后将 ExceptionHandled 设置为阻止 Exception 传递到用户界面，否则它可能在那里导致“Yellow Screen of Death”错误。

VB.NET

```
Protected Sub SqlDataSource1_Deleted(ByVal sender As Object,
    ByVal e As System.Web.UI.WebControls.SqlDataSourceStatusEventArgs) _
    Handles SqlDataSource1.Deleted
    If e.Exception IsNot Nothing Then
        ErrorMessage.Text = "We're sorry, but something went terribly wrong while " &
            "deleting your genre."
        e.ExceptionHandled = True
    End If
End Sub
```

C#

```
protected void SqlDataSource1_Deleted(object sender, SqlDataSourceStatusEventArgs e)
{
    if (e.Exception != null)
    {
        ErrorMessage.Text = @"We're sorry, but something went terribly wrong while
            deleting your genre.";
        e.ExceptionHandled = true;
    }
}
```

A.16 第 16 章

练习 1 答案

身份验证就是向系统(例如 Web 站点)证明您的身份。验证之后，授权机制就能确定您在系统中能做什么、不能做什么。

练习 2 答案

要将对 Management 文件夹的访问扩展到 John 和 Editors 角色中的所有用户，需要将当前的 roles 特性扩展为包含 Editors，并添加另外一个 allow 元素，将该元素的 users 特性设置为 John:

```
<system.web>
  <authorization>
    <allow roles="Managers, Editors" />
    <allow users="John" />
    <deny users="*" />
  </authorization>
</system.web>
```

roles 特性允许指定多个用逗号分隔的角色。为了授权给 John 账户，需要添加另一个 allow 元素，

然后在 users 特性中填充 John 的名称。

从可维护性的角度来看，如果可能，最好将 John 添加到 Managers 或 Editors 角色中。然而，最后可能会给 John 提供更多的权限(他可以访问 Manager 或 Editor 可以访问的所有内容)。通常，最好尽可能通过角色管理用户，但也要知道也可以给单个账户授予必需的权限(或者使用 deny 元素显式地剥夺这些权限)。

练习 3 答案

如果要将所有用户重定向到相同的页面，所需要设置的是 DestinationPageUrl:

```
<asp:Login ID="Login1" runat="server" DestinationPageUrl="~/MyProfile.aspx">
```

当用户成功登录时，会自动被带到 MyProfile.aspx 页面。

练习 4 答案

LoginStatus 只显示表示用户是否登录的简单文本。在默认情况下，当用户当前没有登录时，显示的文本是 Login；当用户已经登录时，显示 Logout。单击链接要么将用户发送到默认 Login 页面，要么让用户注销。

LoginView 在某种程度上与之类似，因为它也依据用户当前是否登录来显示不同内容。然而，因为控件完全是模板驱动的，所以可以完全控制显示的内容。为了能够区分不同用户角色之间的差异，可以使用 RoleGroups 元素创建模板，使它们只显示给特定角色中的用户。

A.17 第 17 章

练习 1 答案

应该将最喜欢的主题作为 String 属性实现，并命名为 FavoriteTheme。为了确保总是拥有有效主题，还要设置默认值。最后，应该让匿名用户也能访问这个属性。最终的 Profile 属性应该如下所示：

```
<add name="FavoriteTheme" defaultValue="Monochrome" allowAnonymous="true" />
```

为了支持匿名 Profile，还需要通过在 web.config 文件中添加一个<anonymousIdentification>元素作为<system.web>的直接子元素来显式启用它们：

```
<system.web>
  <anonymousIdentification enabled="true" cookieName="PlanetWroxAnonymous" />
```

练习 2 答案

根据在问题中看到的语法，现在可以访问新属性，并用它来改变 BasePage 中的当前主题：

```
VB.NET
Private Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles Me.PreInit
    Dim myProfile As ProfileCommon =
        CType(HttpContext.Current.Profile, ProfileCommon)
```

```

    If Not String.IsNullOrEmpty(myProfile.FavoriteTheme) Then
        Page.Theme = myProfile.FavoriteTheme
    End If
End Sub
```

```

C#
private void BasePage_PreInit(object sender, EventArgs e)
{
    ProfileCommon myProfile = (ProfileCommon) HttpContext.Current.Profile;
    if (!string.IsNullOrEmpty(myProfile.FavoriteTheme))
    {
        Page.Theme = myProfile.FavoriteTheme;
    }
}
```

练习 3 答案

为了使用 Profile 确定主题选择器，还要改变母版页 Frontend.master 中的代码。不是在 cookie 中保存用户选中的主题，现在应该将它保存在 Profile 文件中。修改 Page_Load 中的代码，如下所示：

```

VB.NET
If Not Page.IsPostBack Then
    Dim selectedTheme As String = Page.Theme
    If Not String.IsNullOrEmpty(Profile.FavoriteTheme) Then
        selectedTheme = Profile.FavoriteTheme
    End If
    If ThemeList.Items.FindByValue(selectedTheme) IsNot Nothing Then
        ThemeList.Items.FindByValue(selectedTheme).Selected = True
    End If
End If
Select Case Page.Theme.ToLower()
```

```

C#
if (!Page.IsPostBack)
{
    string selectedTheme = Page.Theme;
    if (!string.IsNullOrEmpty(Profile.FavoriteTheme))
    {
        selectedTheme = Profile.FavoriteTheme;
    }
    if (ThemeList.Items.FindByValue(selectedTheme) != null)
    {
        ThemeList.Items.FindByValue(selectedTheme).Selected = true;
    }
}
switch (Page.Theme.ToLower())
```

然后，可以在母版页中将 lstPreferredTheme_SelectedIndexChanged 中的代码简化为：

```

VB.NET
Protected Sub ThemeList_SelectedIndexChanged(ByVal sender As Object,
    ByVal e As System.EventArgs) Handles ThemeList.SelectedIndexChanged
```



```
Profile.FavoriteTheme = ThemeList.SelectedValue
Response.Redirect(Request.Url.ToString())
End Sub
```

```
C#
protected void ThemeList_SelectedIndexChanged(object sender, EventArgs e)
{
    Profile.FavoriteTheme = ThemeList.SelectedValue;
    Response.Redirect(Request.Url.ToString());
}
```

为了对匿名用户启用 Profile 功能，需要在 web.config 文件中，<system.web>正下方，使用下面的元素启用匿名标识：

```
<anonymousIdentification enabled="true" cookieName="PlanetWroxAnonymous" />
```

A.18 第 18 章

练习 1 答案

调试是在开发环境中观察代码执行，为了理解代码的执行路径而研究变量、考察对象、查找漏洞并修复它们的过程。调试通常在开发时在 Visual Web Developer IDE 中进行。

另一方面，跟踪提供与代码运行时执行相关的信息。如本章所述，可以使用跟踪来获得有关触发事件以及触发顺序的信息。另外，可以给 Trace 添加自己的信息。由于通过配置禁用跟踪能够极大地减少与之相关的系统性能开销，因此可以在代码中保留跟踪调用，以便在需要时能够方便地禁用或启用跟踪。

练习 2 答案

防止可能的异常在用户界面中出现的最好方法是将代码包装在 Try/Catch 块中。这样，在出现错误的情况下，就可以给用户显示错误消息。代码应该如下所示：

```
VB.NET
Try
    ' Execute code here to send an e-mail or write to a file.
Catch ex As Exception
    ErrorMessage.Text =
        "Something went wrong while executing the requested operation"
End Try

C#
try
{
    // Execute code here to send an e-mail or write to a file.
}
catch (Exception ex)
{
    ErrorMessage.Text =
        "Something went wrong while executing the requested operation";
}
```

}

练习 3 答案

要理解站点中出现了哪些异常并找出它们出现的位置(也就是说, 哪些页面或代码文件产生了异常), 可以使用 `Application_Error` 事件处理程序中的某些代码记录全部异常。在这种方法中截获的异常详细信息可以帮助理解异常产生的原因, 从而有助于修复它们。

要防止用户看见“Yellow Screen of Death”错误消息, 需要使用自定义错误页面。应该创建一个简单的 Web 窗体, 告诉用户出现了错误。为了告知 ASP.NET 运行库显示文件的内容而不是错误消息, 在 `web.config` 文件中要有下面的元素:

```
<customErrors mode="On" defaultRedirect="~/Errors/AllOtherErrors.aspx"
    redirectMode="ResponseRewrite">
  <error statusCode="500" redirect="~/Errors/Error500.aspx" />
</customErrors>
```

该元素为错误代码 500(当代码意外崩溃时会出现这个错误代码)创建专门的页面。当出现其他异常时, 例如“Page not found”错误, 用户就会被发送到更通用的 `AllOtherErrors.aspx` 页面。

附录 B

配置 SQL Server 2008

本书假定使用 Microsoft SQL Server 2008 Express Edition 作为 Planet Wrox 项目的数据库。SQL Server 2008 Express Edition 对于开发而言非常好，因为它是免费的并且相对是轻量级的，同时易于使用。然而，由于在处理器和内存空间的利用方面，以及数据库大小的限制，在生产环境中它就不一定很适用。在 Express Edition 不足以胜任的情况下，就需要使用它的姊妹版：SQL Server 2008 商业版，例如 Standard 或 Enterprise Editions。本附录将介绍 SQL Server 2008 的安全性，如何让 SQL Server 2008 数据库和 ASP.NET 4 应用程序一起运行，如何获得和使用 SQL Server Management Studio Express(Microsoft 提供的一种用于管理数据库的免费工具)。

虽然本附录没有讨论 SQL Server 2000 和 2005，但其中大多数概念同样适用于 SQL Server 的较早版本。实际上，甚至可以使用 SQL Server 2008 Management Studio Express 来管理 SQL Server 2000 和 2005 数据库。附录中显示的截图来自 Express Edition，但是同样的原理适用于 SQL Server 商业版。

除了配置 SQL Server 和 ASP.NET 4 应用程序让它们一起运行之外，本附录还介绍了如何配置 SQL Server 数据库以支持 ASP.NET 4 应用程序服务。如果想让应用程序使用自己的自定义数据库而不是自动生成的 ASPNETDB.MDF 数据库，这就很有必要。

B.1 配置 SQL Server 2008

在配置数据库之前，需要了解与数据库和 Web 应用程序必然相关的不同安全概念。在第 19 章中，已经知道当配置文件系统的安全设置时，Web 服务器使用的不同计算机账户是如何发挥重要作用的，而且在连接到 SQL Server 时也没有什么不同。在下一节中，将初步了解连接到 SQL Server 的不同方法。随后几节介绍将.mdf 数据库文件附加到 SQL Server 的方法，接下来讨论如何配置应用程序和数据库，以便它们能够相互会话。

B.1.1 术语和概念

当连接到 SQL Server 数据库时，SQL Server 需要知道您是谁，因此它才能在对象(例如数据库中的表)上执行正确的访问策略。SQL Server 2008 支持两种不同的身份验证机制：SQL Server Authentication 和 Windows Authentication(通常称为 Integrated Security)。这两种验证机制各有优点，要求写不同的连接字符串来连接 SQL Server。在下一节中将介绍两个最常见的连接字符串，建议访问 <http://www.connectionstrings.com> 以获得可用连接字符串的扩展列表。

1. SQL Server Authentication

使用 SQL Server Authentication 时，SQL Server 负责用户管理。可以使用 Microsoft SQL Server Management Studio(要么是 Express Edition(本附录稍后将讨论如何获得和使用它)或者 SQL Server 2008 商业版附带的完整版本)来管理 SQL Server 数据库的用户。

要将 Web 应用程序连接到使用 SQL Server 身份验证的 SQL Server 2008，需要在应用程序的连接字符串中传递用户名和密码。典型的连接字符串如下所示：

```
Data Source=ServerName;Initial Catalog=DatabaseName;
User Id=UserName;Password=Password;
```

在这里，Data Source 给 SQL Server 的未命名实例提供地址：服务器由它的名称提供地址。从第 13 章可以知道，也可以作为命名实例安装 SQL Server。使用命名实例时，服务器的名称之后是一个反斜杠和特定 SQL Server 实例的名称。例如：

```
Data Source=ServerName\InstanceName;Initial Catalog=DatabaseName;
User Id=UserName;Password=Password;
```

当需要通过 Internet 连接到远程 SQL Server 数据库时，经常使用 SQL Server 身份验证，因为下面将讨论的 Windows 身份验证并不支持这种场景。

2. Windows Authentication

使用 Windows Authentication 时，Windows OS 负责用户管理。与数据库的所有交互都在调用用户的上下文中完成，因此数据库知道谁在访问系统。还需要将 Windows 账户映射到 SQL Server 账户，以便 SQL Server 可以确定这个账户是否有足够的权限。稍后将介绍这些如何实现。

使用 Windows Authentication 的典型连接字符串如下所示：

```
data source=ServerName;Initial Catalog=DatabaseName;Trusted_Connection=True
```

这里没有指定用户名和密码，而是将 Trusted_Connection=True 添加到连接字符串中，以表示要使用调用用户的用户上下文连接到服务器。Integrated Security=True 这种设置具有同样的效果。

由于这两种身份验证方法的最终结果相同(它们都允许连接到 SQL Server)，因此可能想知道应该使用哪种方法。

3. 在 Windows Authentication 和 SQL Server Authentication 之间选择

通常，推荐使用 Windows Authentication。不需要在连接字符串中使用密码，这就意味着应用程

序更安全。不需要通过网络发送密码，它也不保存在应用程序的配置文件中。

然而，SQL Server Authentication 更容易建立。由于用户名和密码是显式指定的，因此不需要知道最终的用户账户(应用程序在该账户下运行)。

本附录稍后将讨论如何使用这两种身份验证机制连接到数据库。然而，首先要考虑其他一些东西：用来管理 SQL Server 的工具。

B.1.2 使用 SQL Server Management Studio

可以使用 SQL Server Management Studio 来管理数据库。它可以用来将数据库附加到 SQL Server 或者让它与 SQL Server 分离，在现有数据库中创建新的数据库对象(例如表)，选择、创建、编辑和删除数据等。如果使用的是 SQL Server 2008 商业版，就可以直接访问到 SQL Server Management Studio，因为它是与数据库引擎绑定在一起的。如果使用的是 SQL Server 的免费 Express Edition，就要先下载 SQL Server Management Studio Express。这种数据库管理工具的 Express 版本与它的商业版类似，允许执行大多数(如果不是全部)数据库管理任务。

获得和安装 SQL Server Management Studio Express

可以从Microsoft站点的 www.microsoft.com/express/database/ 页面上下载 SQL Server Management Studio Express。同样，也可以登录主下载页面 www.microsoft.com/downloads，搜索“SQL Server 2008 Management Studio Express”。确保下载和安装的是 Management Studio 的 2008 版本，而不是较早的 2005 版本。最后一种方法是按照第 19 章的介绍使用 Web Platform Installer。

下载完 Management Studio 安装文件之后，双击它启动安装程序，并按屏幕指令操作。当要求选择安装类型时，选择 Perform a New Installation of SQL Server 2008。这个选项允许选择 Management Tools 作为安装向导后期安装的组件。安装完成之后，就可以从“开始”菜单中启动 SQL Server Management Studio。看到的屏幕如图 B-1 所示，在这里选择要登录到的 SQL Server 实例。



图 B-1

如果还没有在 Server name 下拉列表中预选某个项，则输入想要连接到的 SQL Server 实例的名称。如果要连接到本地 Express 安装，则输入.\SqlExpress。否则，输入数据库服务器和 Instance(如果使用了)的名称。根据连接的数据库服务器，可以用当前的 Windows 账户使用 Windows Authentication 登录，或者从 Authentication 下拉列表中选择 SQL Server Authentication，然后输入用户名和密码。登录之后看到的屏幕如图 B-2 所示，在这里可以管理 SQL Server 实例。

如果在连接远程 SQL Server(例如，SQL Server 的某个实例不在运行 Management Studio 的物理服务器上)时遇到麻烦，可能要首先在 SQL Server 中激活远程连接。这将在稍后讨论。

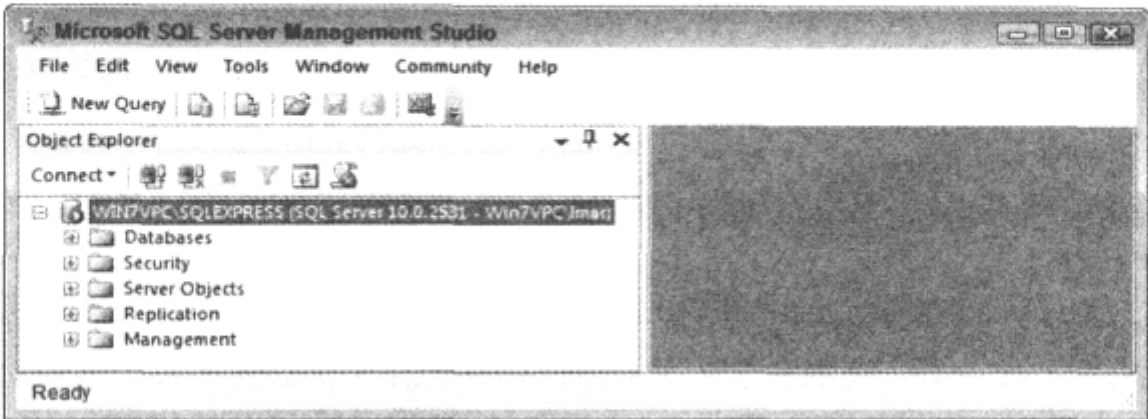


图 B-2

B.1.3 激活 SQL Server 中的远程连接

在使用 SQL Server 2008 时，可能会接收到下列错误：建立到服务器的连接时出现错误。在连接 SQL Server 2008 时，出现这种错误可能是因为在默认设置下 SQL Server 不允许远程连接。或者，可能会收到一个更一般的错误，指出没有找到服务器或者服务器不可访问。

虽然在数据库服务器关闭或没有响应时也会出现这个错误，但没有为远程连接配置 SQL Server 时也会出现这个错误。在默认安装中，SQL Server 只允许本地应用程序连接到它，并自动中断远程连接。要解决这个问题，就要授权远程系统访问数据库，请按下列步骤操作：

- (1) 从 Microsoft SQL Server 2008 的 Start 菜单选项中打开 SQL Server Configuration Manager。根据使用的 SQL Server 的版本，这个选项可能在 Configuration Tools 子菜单下。
- (2) 在出现的窗口中，展开 SQL Server Network Configuration，找到 SQL Server 实例，单击它显示可用协议列表。图 B-3 显示名为 SQLEXPRESS 的 SQL Server 实例的列表。
- (3) 在右边的协议列表中，右击 Named Pipes，选择 Enable(如果其当前状态设置为 Disabled)。
- (4) 重复第(3)步，启用 TCP/IP。

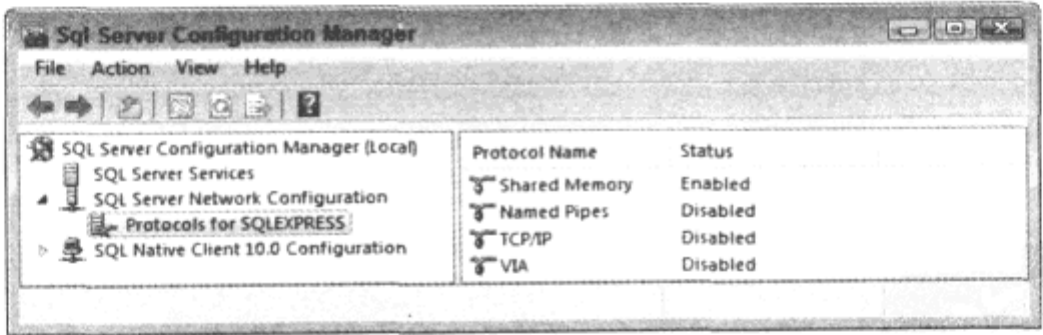


图 B-3

(5) 重启 SQL Server。为此，可以右击 SQL Server Management Studio 的 Object Explorer 中的服务器(如图 B-2 所示)，并选择 Restart 选项。如果得到有关安全权限的错误，可能需要重启计算机。

SQL Server 现在允许来自远程计算机的传入连接。然而，在实际使用数据库之前，要先将它们附加给 SQL Server。这将在下一节讨论。



提示：如果在连接 SQL Server 时有问题，请确保 SQL Server 已安装并正在运行。要验证这一点，请打开控制面板，然后打开 System and Security(在 Windows Vista 上叫做 System and Maintenance)类别中的管理工具部分。再打开 Services 选项，然后验证连接的 SQL Server 实例是否在运行。如果在本地计算机上安装了 SQL Server Express，这个服务就称为 SQL Server (Express)。

给 SQL Server 附加数据库

SQL Server Express 允许通过两种方法运用数据库文件：可以使用连接字符串中的特定属性在运行时附加它们，或者在开始使用数据库之前，使用像 Management Studio 这样的工具附加它们。到目前为止，已经在所有与数据库相关的章节中使用过第一种选项。使用的 Planet Wrox 连接字符串如下所示：

```
<add name="PlanetWroxConnectionString1" connectionString="Data Source=.\SQLEXPRESS;
AttachDbFilename=|DataDirectory|\PlanetWrox.mdf;Trusted_Connection=True;
User Instance=True" providerName="System.Data.SqlClient" />
```

这个连接字符串指向 Web 站点的 App_Data 文件夹(由|DataDirectory|确定)中名为 PlanetWrox.mdf 的数据库。当使用这个数据库文件时该连接字符串指示 SQL Server 将它动态分配给 SQL Server。当不再需要这个数据库时，再分离它。它还指示服务器使用 User Instance(一种 SQL Server 功能)，允许低权限的账户也可以运行 SQL Server，并动态地将数据库附加给该账户。

对于本地开发而言，这是一种好的解决方案，因为它允许方便地使用 SQL Server 数据库，创建和使用新的数据库，还可以将它们在项目之间移动。然而，对于使用生产数据库的情况，这个选项就不够好，首先要将数据库附加给 SQL Server。下面的步骤解释了如何将 PlanetWrox.mdf 和 aspnetdb.mdf 数据库附加给 SQL Server 的实例。如果要使用 SQL Server Management Studio 在 SQL Server 数据库中执行维护任务(例如，管理用户和角色)，而在 VWD 中不能完成这些任务时，就可以采用相同的步骤。

- (1) 创建一个文件夹来保存新数据库，例如 C:\Data\SqlServer。
- (2) 将文件 PlanetWrox.mdf 和 aspnetdb.mdf 及其相关的.ldf 文件从 Web 站点的 App_Data 文件夹(位于 C:\BegASPNET\Release)移动到该新文件夹中。
- (3) 为 SQL Server 使用的账户(在默认情况下是 Network Service 账户)以及自己的账户激活存放数据库的文件夹(C:\Data\SqlServer)上的 Modify 权限。第 19 章讨论过设置这些权限的方法。
- (4) 打开 SQL Server 2008 Management Studio(任何版本都一样)，登录要附加数据库的 SQL Server 实例。取决于安全设置，可能需要作为管理员运行这个程序。为此，请右击 Windows 的开始菜单中的 Management Studio 菜单项，并选择 Run as Administrator 选项。不管如何启动 SQL Server Management Studio，得到的屏幕都如图 B-2 所示。
- (5) 右击图 B-2 中所示的 Databases 节点，并选择 Attach 选项。
- (6) 在出现的对话框中，单击 Add 按钮，然后选择在第(2)步中移动到 C:\Data\SqlServer 文件夹的 PlanetWrox.mdf 文件。
- (7) 单击 Attach As 栏中的值使之可编辑，然后输入 PlanetWrox 作为数据库的新名称。完成之后，

对话框应该如图 B-4 所示。

(8) 单击 OK 按钮将数据库附加给 SQL Server。如果出现错误，请确保自己的账户(或者您所属的 Users 组)和 Network Service 账户对 C:\Data\SqlServer 文件夹和该文件夹包含的.mdf 文件都有 Modify 权限。

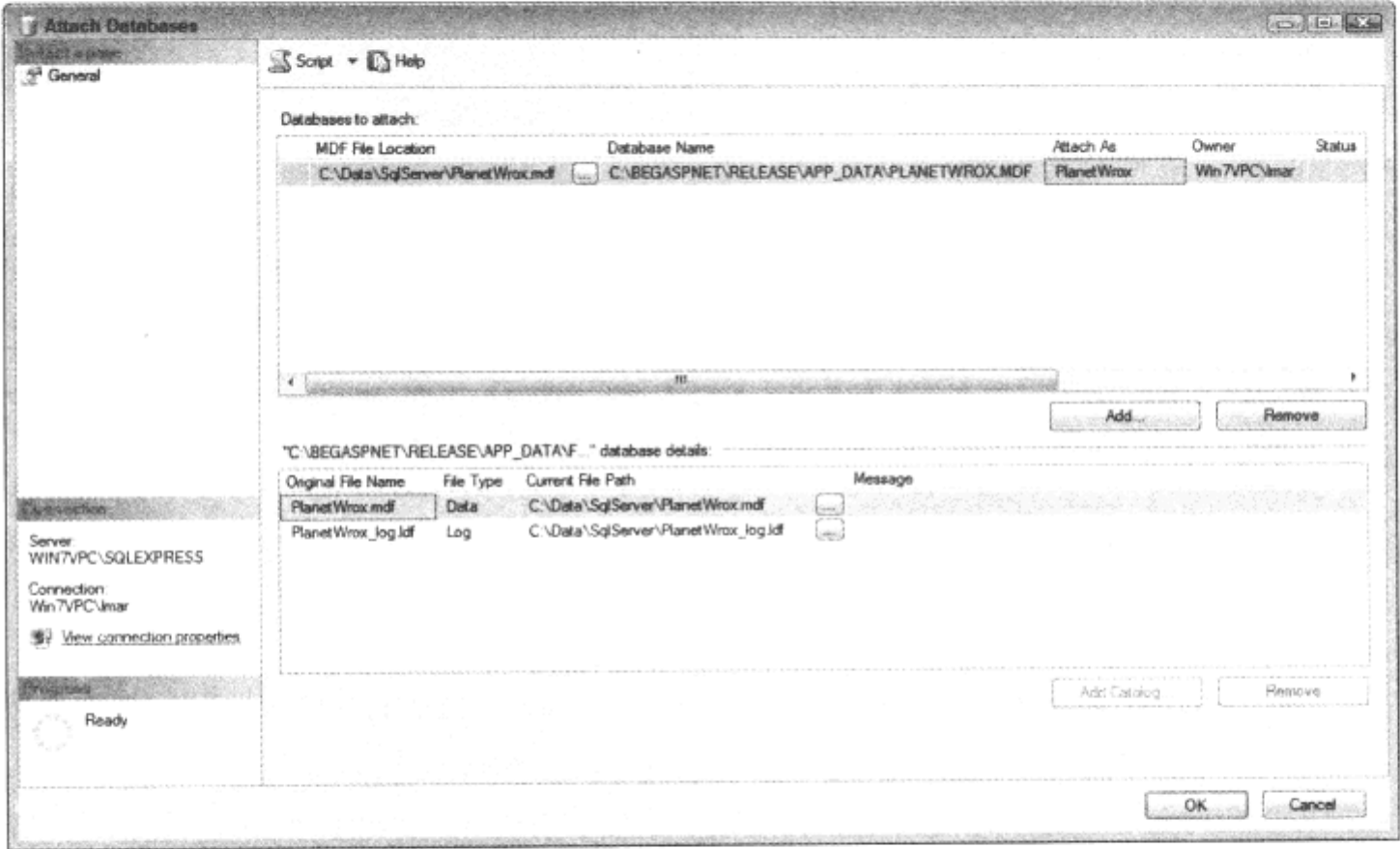


图 B-4

(9) 现在可以在 Management Studio 中 Object Explorer 的 Databases 节点下访问数据库。如果展开 Databases 元素然后查看数据库，会看到一些非常熟悉的项(例如 Tables)，本书前面在 Visual Web Developer 中的 Database Explorer 中也出现过这些选项。图 B-5 显示了附加的数据库和它的表。

(10) 重复步骤(5)~(8)，但此次附加 aspnetdb.mdf 数据库。在步骤(7)中输入 aspnetdb 作为数据库的新名称。

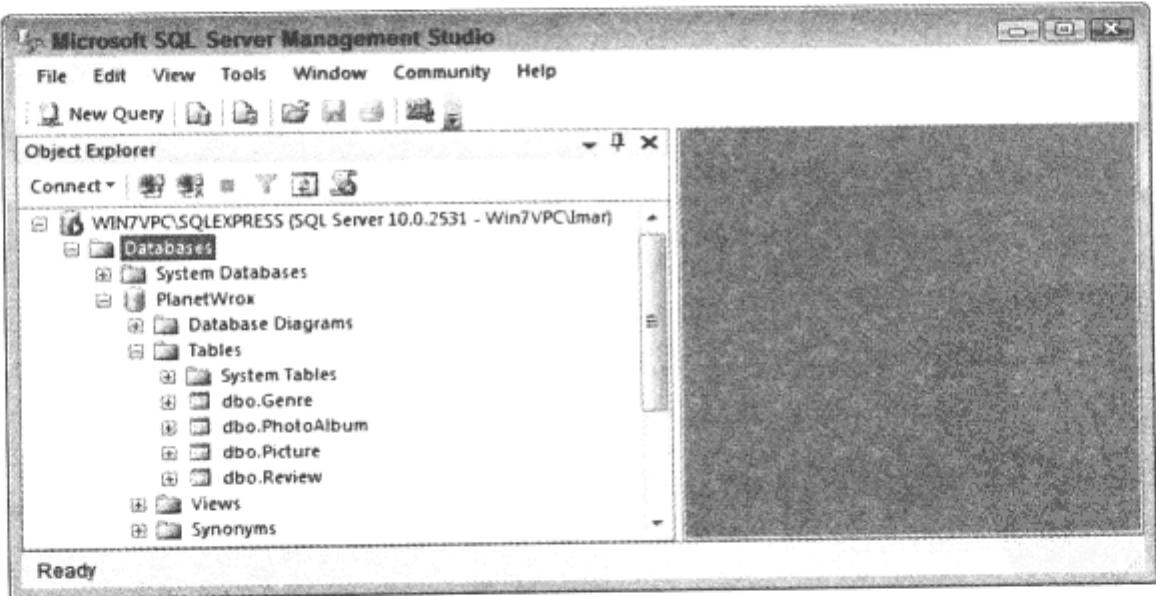


图 B-5

此时，只有管理账户(Windows 管理员或名为 SA 的内置 SQL Server 管理账户)能够访问该数据库。为了让 Planet Wrox Web 站点使用这两个数据库，还要配置 SQL Server 安全以及 Web 站点。下

面将介绍如何做。

B.1.4 将应用程序连接到 SQL Server 2008

在下一节将通过两个不同但又常见的场景来介绍连接到 SQL Server 的方法：使用 SQL Server Authentication；当 IIS 和 SQL Server 在同一个服务器上时，使用 Windows Authentication。对于这两个场景而言，将了解如何配置 SQL Server、Planet Wrox Web 站点，如果有必要，还将配置 Windows 账户。

当处理驻留站点的外部宿主公司时，可能使用第一个场景。当 Web 主机提供 SQL Server 时，它们通常使用 SQL Server Authentication，因此要求将用户名和密码传递给数据库服务器。

如果自己驻留站点，并且在同一台计算机上既有 SQL Server 2008 也有 IIS 时，通常使用第二个场景。

更高级的场景(例如当 IIS 和 SQL Server 在两台不同计算机上时使用 Windows Authentication)超出了本附录讨论的范围。有关配置和保护 SQL Server 的更多信息，请参考《SQL Server 2008 DBA 入门经典》或 *Professional SQL Server 2008 Administration*(ISBN: 978-0-470-24796-9)这两本书。

场景 1——使用 SQL Server Authentication

从配置的角度来看，这可能是最容易配置的场景：所要做的就是确保 SQL Server 安装支持 SQL Server 身份验证，在 SQL Server 中创建一个用户，然后在 Planet Wrox Web 站点的连接字符串中使用这个账户。要实现这一点，请按如下步骤操作：

(1) 在 SQL Server Management Studio 中，在图 B-2 中显示的 Object Explorer 中右击服务器名称，选择 Properties 选项，再切换到 Security 类别。出现的对话框如图 B-6 所示。

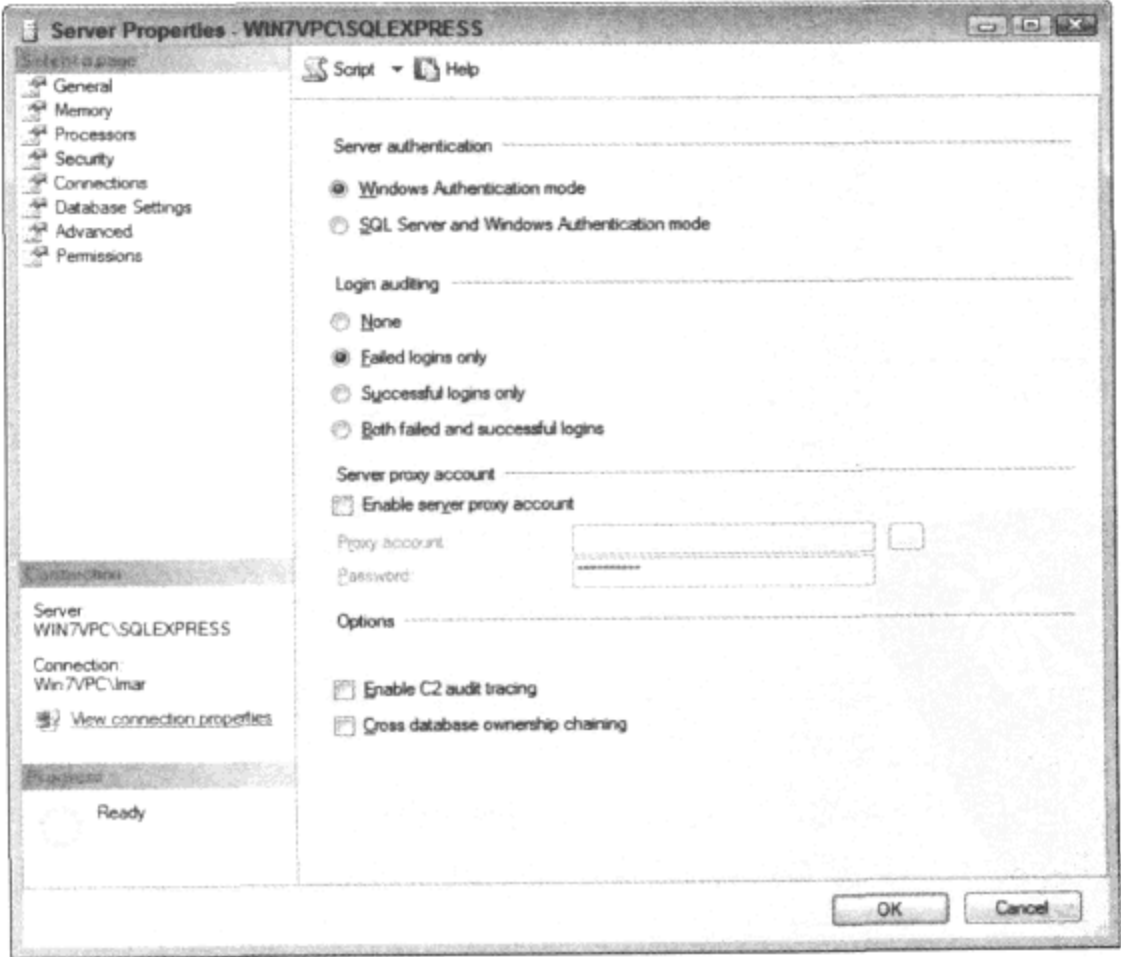


图 B-6

(2) 如果还没有选中，请选择屏幕顶部的 SQL Server and Windows Authentication mode 选项。在单击 OK 按钮之前，请单击屏幕顶部的 Help 按钮，阅读更多有关 SQL 和 Windows Authentication 的信息，确定是否真的需要 SQL Server Authentication。Windows Authentication 比 SQL Server Authentication 更安全，因此建议使用它。

(3) 重启 SQL Server。要完成这个步骤，可以右击 Object Explorer 中的服务器，然后选择 Restart 选项。如果得到与安全权限有关的错误，可能还需要重启计算机。

(4) 回到 SQL Server Management Studio 的 Object Explorer 窗口中，展开服务器的 Security 节点，如图 B-2 所示。确保选择直接在服务器名称下面的那个项，而不是属于特定数据库的那一个。右击 Logins，选择 New Login 命令。

(5) 输入 Login 名称，然后选择 SQL Server Authentication 选项，输入密码。在本示例以及后面的示例中，都使用 PlanetWroxUser 作为用户名，使用 Pa\$\$w0rD (使用 0 而不是字母 o)作为密码。

(6) 清除 Enforce password expiration 复选框。这也将禁用 User must change password at next login。看到的对话框应该如图 B-7 所示。

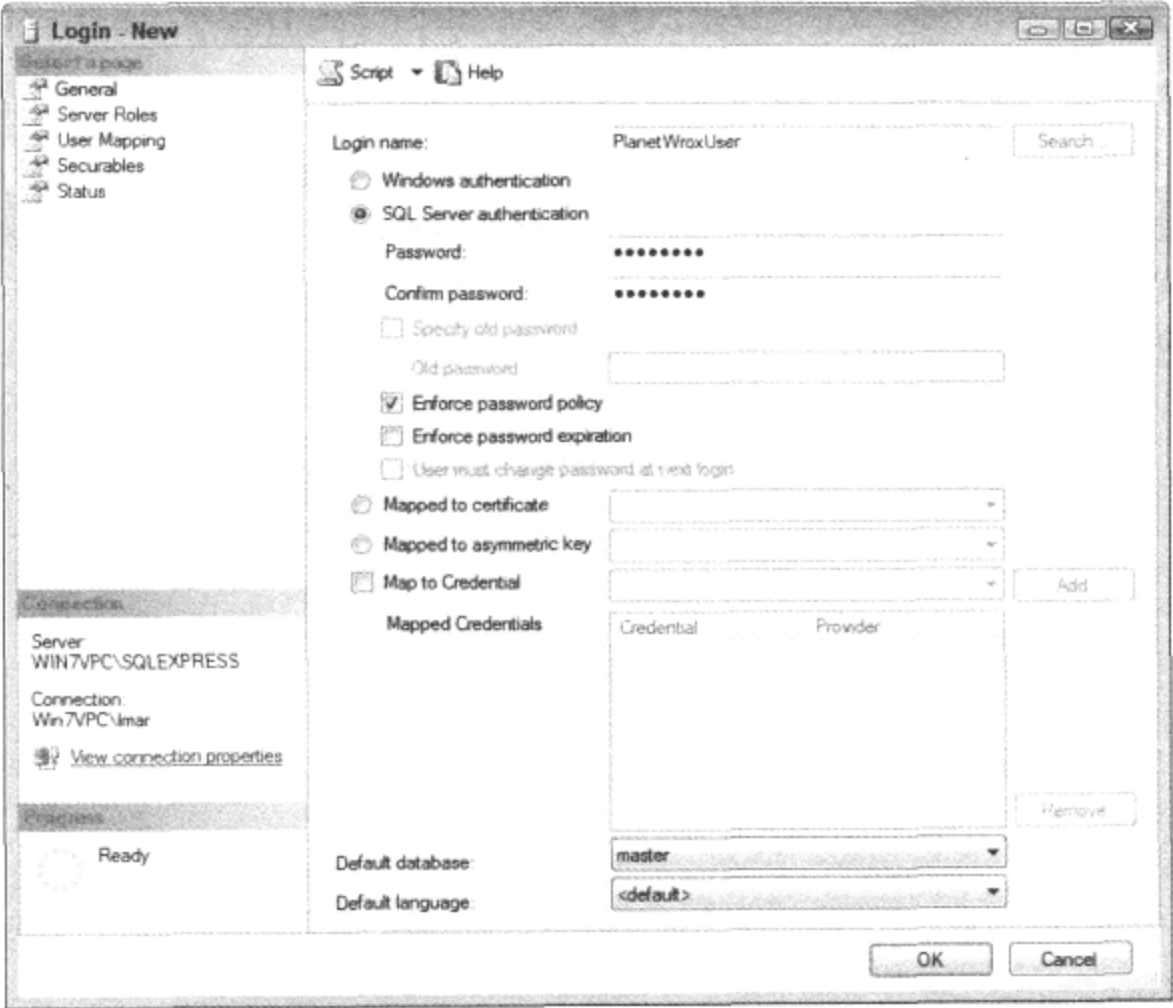


图 B-7

如果要了解屏幕上各个设置的更多信息，请单击屏幕顶部的 Help 按钮。

(7) 单击 OK 按钮，创建该新账户。

创建账户之后，下一步就是给这个新账户提供访问数据库的正确权限。

(1) 在 Object Explorer 上，展开 Databases 节点，然后展开 PlanetWrox 数据库，接着是展开 Security 节点。最后，右击 Users 节点，选择 New User 命令。

(2) 在 User name 文本框中，输入 PlanetWroxUser。

(3) 在 Login name 文本框中，再次输入 PlanetWroxUser。或者，单击省略号按钮，再单击 Browse 按钮，从出现的列表中选择用户。

(4) 在屏幕底部，可以看到一个标记为 Database role membership 的框。在这个框中，可以选择许多可授权给新用户角色。这里的规则是：尽可能给用户提供最少的权限。最好选择 db_datareader 和 db_datawriter，它们允许账户读取和写入数据库中的表，因此选择这两项，如图 B-8 所示。

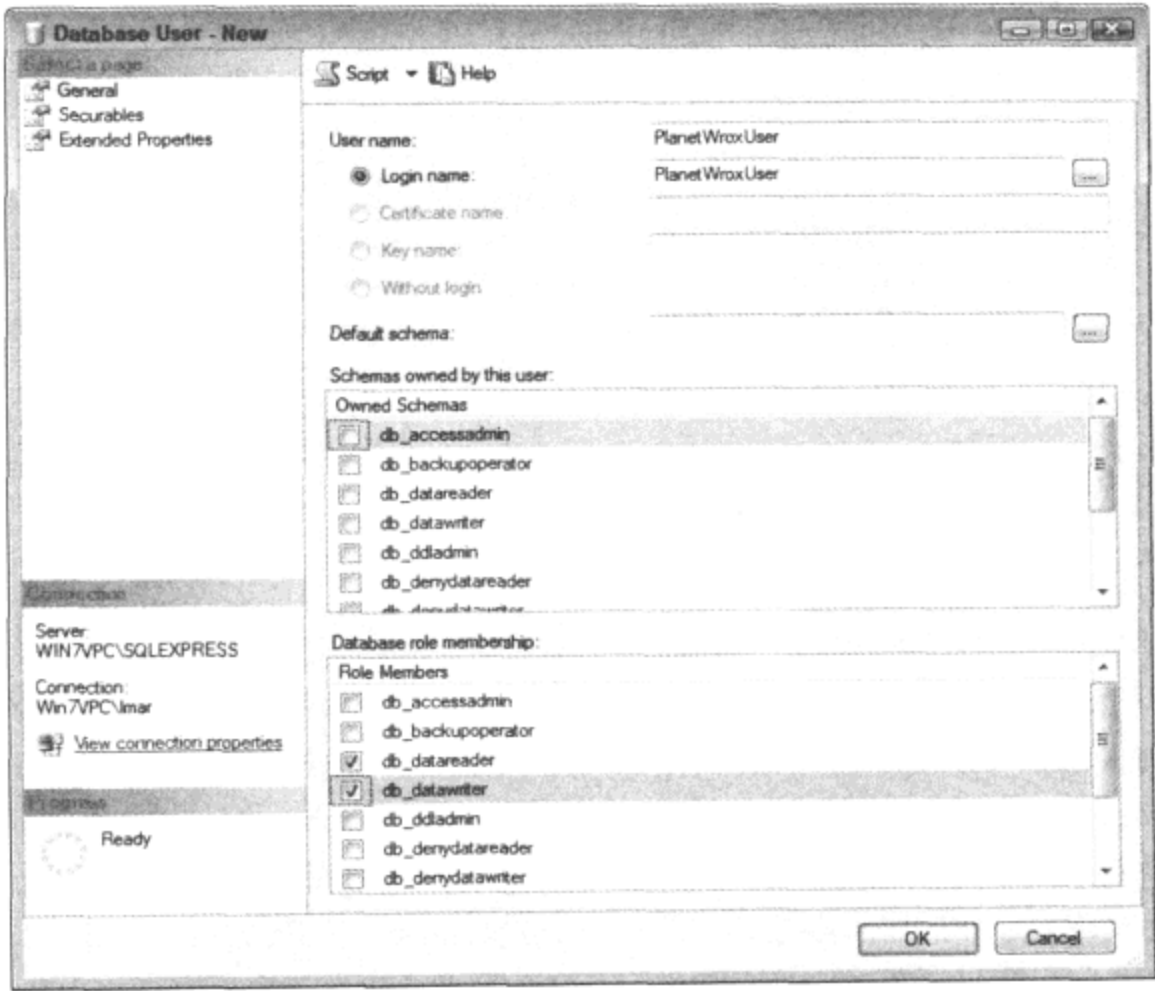


图 B-8



提示：有关各种角色的更多信息，请参考 SQL Server 的联机丛书。

(5) 如果要为数据库对象设置精细的安全选项，请单击图 B-8 中左边的 Securables 选项。这个对话框允许确定数据库中对象上用户账户的权限，这些对象包括表、视图和存储过程等。对于 Planet Wrox Web 站点而言，不需要在这个对话框中进行任何修改。

(6) 最后，单击 OK 按钮，将 PlanetWroxUser 账户分配给 db_datareader 和 db_datawriter 角色。

(7) 重复前面所有的步骤，配置在前面附加的 aspnetdb 数据库。到第(4)步时，选择以 aspnet_ 开头的角色。这些都是 ASP.NET 应用程序服务使用的角色，它们给新用户账户授予了足够的权限来支持各种应用程序服务，例如 Membership 和 Roles。不需要选择其他任何角色。

(8) 现在可以关闭 SQL Server Management Studio，因为已经完成了对它的操作。

正确配置了 SQL Server 和用户账户之后，最后一个阶段就是配置 Web 站点来使用这个新用户账户。

- (1) 从 C:\BegASPNET\Release 文件夹中打开已部署的 Planet Wrox 应用程序的 web.config 文件。
- (2) 修改<connectionStrings>元素，如下所示：

```
<connectionStrings>
  <clear />
  <add name="PlanetWroxConnectionString1" connectionString="Data Source=ServerName;
    Initial Catalog=PlanetWrox;User ID=PlanetWroxUser;password=Pa$$w0rD"
    providerName="System.Data.SqlClient"
  />
  <add name="PlanetWroxEntities" connectionString="
    metadata=res://*/App_Code.PlanetWrox.csdl|res://*/App_Code.PlanetWrox.ssdl|res
    ://*/App_Code.PlanetWrox.msl;provider=System.Data.SqlClient;provider connection
    string=&quot;Data Source=ServerName;Initial Catalog=PlanetWrox;
    User ID=PlanetWroxUser;password=Pa$$w0rD;MultipleActiveResultSets=True&quot;;"
    providerName="System.Data.EntityClient"
  />
  <add name="LocalSqlServer" connectionString="Data Source=ServerName;
    Initial Catalog=aspnetdb;User ID=PlanetWroxUser;password=Pa$$w0rD"
    providerName="System.Data.SqlClient"
  />
</connectionStrings>
```

在配置文件中，每个连接字符串都应该放在单行上。不要忘记用有效的服务器名称替换 Data Source 特性中的值 ServerName。根据服务器和配置，这可能非常简单，就像指向本地计算机上的 SQL Server 的(local)或.\SqlExpress，指向网络上名为 DatabaseServer 的服务器的 DatabaseServer，或者像指向计算机 DatabaseServer 上名为 Sql2008 的命名实例的 DatabaseServer\Sql2008 这样的对象。如果在第 19 章中没有添加 LocalSqlServer 连接字符串，那么现在就需要添加。

- (3) 保存这些变更，然后启动浏览器，登录 http://localhost，打开站点。一切仍运行正常，但此时站点不再使用 App_Data 文件夹中的数据库，而是使用在连接字符串中定义的 SQL Server。

如果在浏览到本地主机上的这个站点时发生错误，就需要关闭 web.config 文件中的自定义错误(出于安全考虑，将它设置为 RemoteOnly 而不是 Off)，以查看实际的错误消息。产生这种错误可能的原因是连接字符串中使用的用户名、密码或服务器名不正确，或者错误地配置了 PlanetWroxUser 账户的数据库角色成员。

场景 2——当 IIS 和数据库位于同一台计算机上时使用 Windows Authentication

这是一个常见场景，特别是当在自己控制的本地计算机上运行站点时。Web 服务器(IIS 或内置的 Web 开发服务器)和 SQL Server 都在相同的物理机器上运行。这个场景使得使用 Windows Authentication 变得更容易，因为 Web 服务器和 SQL Server 都可以使用相同的 Windows 账户。要为此场景配置服务器，请按下列步骤操作：

- (1) 确定 Web 服务器使用的账户。有关细节请参考第 19 章，但是您很可能需要使用 Network Service 或 ApplicationPoolIdentity 账户(默认情况下叫做 IIS AppPool\ASP.NET v4.0)。在本节剩余部分使用 IIS AppPool\ASP.NET v4.0 账户。
- (2) 接下来需要将这个 Windows 账户映射到一个 SQL Server 账户。为此，请打开 SQL Server Management Studio，然后登录 SQL Server 实例。展开服务器(而不是单个数据库)的 Security 节点，如图 B-2 所示。然后右击 Logins，选择 New Login 命令。

(3) 根据配置的账户，在 Login name 文本框中输入 NT Authority\Network Service 或 IIS AppPool\ASP.NET v4.0，然后单击 OK 按钮添加新登录账户。

创建账户之后，下一步就是给这个新账户提供访问数据库的相关权限。

(1) 打开 PlanetWrox 数据库的 Security 节点，展开 Users 节点，右击它，选择 New User 命令。

(2) 根据在前面配置的账户名，在 User name 文本框中输入 Network Service 或 ASP.NET v4.0。

(3) 对于 Login name 文本框，单击省略号按钮，然后单击 Browse 按钮，这时可以选择用户名称。选择在前面配置的账户并单击 OK 按钮两次。

(4) 在屏幕底部，可以看到一个标记为 Database role membership 的框。在这个框中，可以选择许多可授权给新用户的角色。这里的规则是：尽可能给用户提供最少的权限。最好选择 db_datareader 和 db_datawriter，它们允许用户读取和写入数据库中的表，因此选择这两项，如图 B-9 所示。

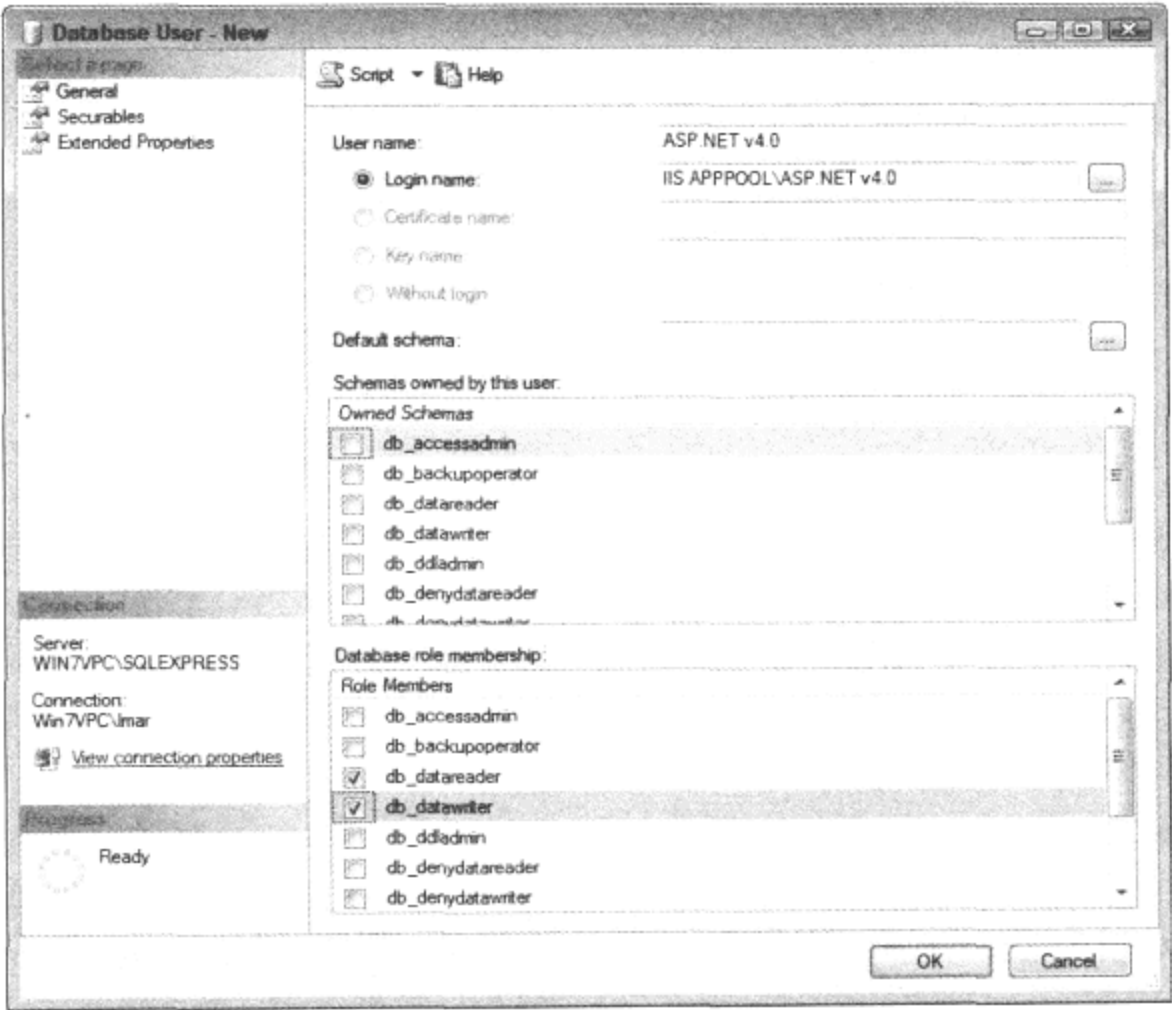


图 B-9



提示：有关各种角色的更多信息，请参考 SQL Server 的联机丛书。

(5) 如果要为数据库对象设置精细的安全选项，请单击图 B-9 中的 Securables 选项。这个对话框允许确定数据库中对象上用户账户的权限，这些对象包括表、视图和存储过程等。对于 Planet Wrox Web 站点而言，不需要在这个对话框中进行任何修改。

(6) 最后，单击 OK 按钮，将账户分配给 db_datareader 和 db_datawriter 角色。

(7) 重复步骤(1)~(6)，配置在前面附加的 aspnetdb 数据库。到第(4)步时，请选择以 aspnet_开头

的所有角色。这些都是 ASP.NET 应用程序服务使用的角色，它们给新用户账户授予了足够的权限来支持各种应用程序服务，例如 Membership 和 Roles。

正确配置了 SQL Server 和用户账户之后，最后一步就是配置 Web 站点来使用这个新用户账户。

- (1) 从 C:\BegASPNET\Release 文件夹中打开 PlanetWrox 应用程序的 web.config 文件。
- (2) 修改<connectionStrings>元素，如下所示：

```
<connectionStrings>
  <clear />
  <add name="PlanetWroxConnectionString1" connectionString="Data Source=ServerName;
    Initial Catalog=PlanetWrox;Trusted_Connection=True"
    providerName="System.Data.SqlClient"
  />
  <add name="PlanetWroxEntities" connectionString="
    metadata=res://*/App_Code.PlanetWrox.csdl|res://*/App_Code.PlanetWrox.ssdl|res
    ://*/App_Code.PlanetWrox.msl;provider=System.Data.SqlClient;provider connection
    string="Data Source=ServerName;Initial Catalog=PlanetWrox;
    Trusted_Connection=True;MultipleActiveResultSets=True";"
    providerName="System.Data.EntityClient"
  />
  <add name="LocalSqlServer" connectionString="Data Source=ServerName;
    Initial Catalog=aspnetdb;Trusted_Connection=True"
    providerName="System.Data.SqlClient"
  />
</connectionStrings>
```

在配置文件中，每个连接字符串都应该放在单行上。不要忘记用有效的服务器名称替换 Data Source 特性中的值 ServerName。根据服务器和配置，这可能非常简单，就像指向本地计算机上的 SQL Server 的(local)或\SqLExpress，指向网络上名为 DatabaseServer 的服务器的 DatabaseServer，或者像指向计算机 DatabaseServer 上名为 Sql2008 的命名实例的 DatabaseServer\SqL2008 这样的对象。如果在第 19 章中没有添加 LocalSqlServer 连接字符串，那么现在就需要添加。

(3) 保存这些变更，启动浏览器，登录 http://localhost (或是在第 19 章配置的地址)，打开站点。一切仍如期运行，但站点现在不再使用 App_Data 文件夹中的数据库，而是通过 Windows Authentication 使用在连接字符串中定义的 SQL Server，由连接字符串中的 Trusted_Connection=True 特性确定。

如果在浏览到这个站点时发生错误，可能需要关闭 web.config 文件中的自定义错误(或将它设置为 RemoteOnly)，以查看实际的错误消息。产生这种错误可能的原因是连接字符串中使用的服务器名不正确，或者错误地配置了账户的数据库角色成员。

找出正确的账户并正确配置 SQL Server 之后，使用 Windows Authentication 就很容易。实际上，连接字符串现在变得更简单、更安全，因为不需要在它里面保存用户名和密码。

B.2 配置应用程序服务

在本书前面已经知道 ASP.NET 应用程序服务使用的是 SQL Server 2008 数据库，通过使用名为 LocalSqlServer 的连接字符串可以访问这个数据库。当第一次使用应用程序服务之一(例如 Membership

或 Profile)时，应用程序服务会在 Web 站点的 App_Data 文件夹中创建一个名为 aspnetdb.mdf 的新数据库。

如果不想使用默认的 aspnetdb.mdf 数据库而是要使用自己的数据库，情况又会怎样呢？或者，如果不想依赖于创建的本地 SQL Server Express 数据库，而是要从自己专门的服务器中获得成员、角色和配置文件数据，情况又将如何？

在这些情况下，需要重新配置应用程序(通过 web.config 文件)，并告诉它数据库在哪里。有两种方法可以实现此目的：重写 LocalSqlServer 连接字符串和重写各种应用程序服务的设置。然而，在考察这两种选项之前，需要知道如何为 ASP.NET 应用程序服务准备自定义数据库。

B.2.1 为应用程序服务配置数据库

当在 App_Data 文件夹中创建 aspnetdb 数据库时，它已经包含许多对象，例如表和存储过程(可以在服务器中执行的 SQL 代码段)。如果要使用自己的数据库而不是默认数据库，首先要为应用程序服务准备数据库。Microsoft 的 .NET Framework 有一个专门用于完成这项任务的便利工具。可以在下面的位置找到这个工具，它名叫 aspnet_regsql.exe:

```
%windir%\Microsoft.NET\Framework\v4.0.30128
```

如果有 .NET Framework 的更新版本，那么 v4.0 后面的数字会稍有不同。而且，如果使用的是 64 位的 Windows 版本，那么父文件夹为 Framework64。应该在 Windows 资源管理器的地址栏中像上面显示的那样输入 %windir%，它会自动扩展到 Windows 文件夹。要执行这个工具(aspnet_regsql.exe)，请双击文件夹中的 aspnet_regsql.exe 文件。它会启动 ASP.NET SQL Server Setup Wizard。单击 Next 按钮两次，就会进入一个屏幕，它允许选择一个 SQL Server 和一个数据库，如图 B-10 所示。

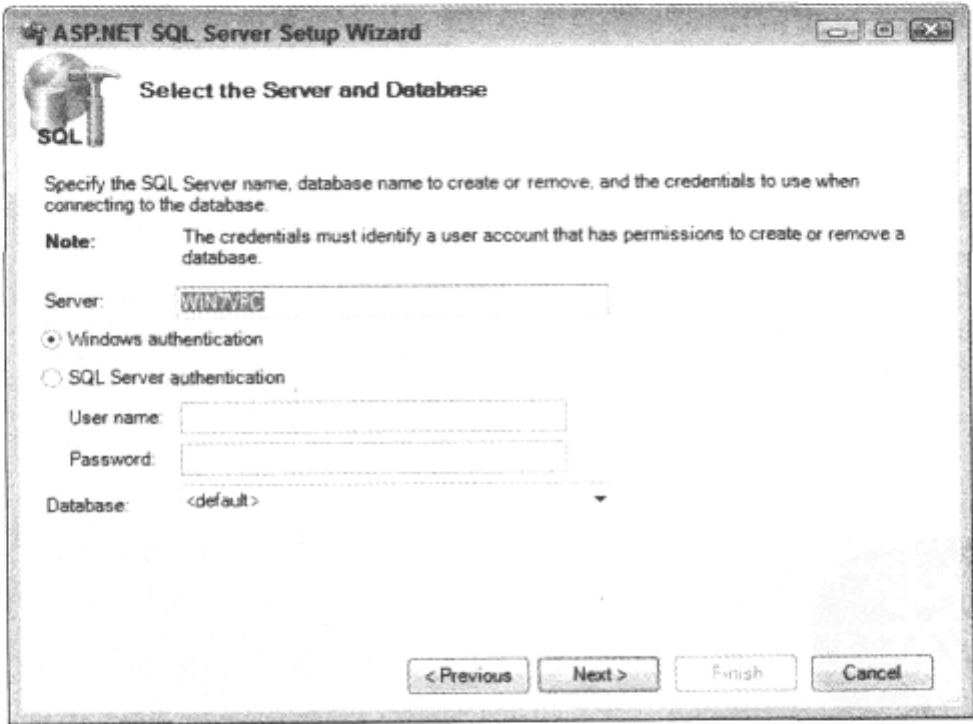


图 B-10

输入数据库服务器的名称(例如，为本地 SQL Server Express Edition 输入.\SqlExpress)，然后从屏幕下半部的下拉列表中选择数据库(例如 PlanetWrox 数据库)。如果没有看到所要的数据库，就要首先在 SQL Server 上附加它。本附录前面讨论过附加数据库的方法。

选择数据库之后，单击 Next 按钮两次，该工具(aspnet_regsql.exe)就会在数据库中创建所需的对

象。完成之后，就可以单击 Finish 按钮，关闭 Wizard。

刚才看到的该工具的 GUI 版本一次性添加对所有应用程序服务的支持。如果想控制创建的对象，请打开命令窗口，导航到%windir%\Microsoft.NET\Framework\v4.0.30128，输入 aspnet_regsql /?，然后按下 Enter 键，就会获得(长长的)选项列表。同样，您的框架文件夹的名称可能会有所不同。在选项列表中滚动查看所有可用的选项。要添加对应用程序服务的支持，至少需要定义下列选项：

- **-S:** 允许设置 SQL Server 名称。
- **-E:** 表示要使用 Windows Authentication 连接到 SQL Server。
- **-A:** 这个选项之后一定要有一个或多个字母来表示对不同应用程序服务的支持，例如 m 表示 Membership，r 表示 Role manager，p 表示 Profile。
- **-d:** 定义必须配置的数据库的名称。如果不改变这个名称，工具就会使用默认的 aspnetdb 数据库。

在数据库 PlanetWrox 中，要依据名为 SqlExpress 的本地 SQL Server 实例来执行创建支持 Membership、Role manager 和 Profile 服务的命令，需要执行下面的命令：

```
aspnet_regsql -S .\SqlExpress -E -A mrp -d PlanetWrox
```

不久就会得到一条表明已经成功配置所请求的服务的消息。

不管是使用向导还是命令行界面，从现在开始，您所配置的数据库可以用于 ASP.NET 应用程序服务。

B.2.2 重写 LocalSqlServer 连接字符串

重写 LocalSqlServer 连接字符串是使用不同数据库最方便最快捷的路径。所要做的就是重写 LocalSqlServer 连接字符串，并让它指向新的数据库。在本附录前面和第 19 章已经介绍过该连接字符串的应用，但在这里将指出它的运行方式：

```
<connectionStrings>
  <clear />
  ...
  <add name="LocalSqlServer" connectionString="Data Source=.\SQLEXPRESS;
    Initial Catalog=aspnetdb;Trusted_Connection=True"
    providerName="System.Data.SqlClient"
  />
</connectionStrings>
```

注意，<connectionStrings>元素包含一个<clear />元素。这个元素用来清除所有以前定义的连接字符串，包括在 machine.config 文件中定义的 LocalSqlServer 连接字符串。然后配置文件使用相同的名称和不同的连接字符串来定义一个新的连接字符串。本示例中连接字符串指向本地 SQLExpress 命名实例上名为 aspnetdb 的附加数据库。当应用程序服务需要连接到 SQL Server 时，它们会找到连接字符串 LocalSqlServer，LocalSqlServer 再允许它们连接到已经定义的 SQL Server。当向数据库中添加对应用程序服务的支持时，应该使用自己的数据库替换 aspnetdb，如前一节所述。

重写 LocalSqlServer 连接字符串是让 ASP.NET Web 应用程序使用不同服务器的快捷方法。所要做的就是指派一个有效的 Data Source 和 Initial Catalog。然而，这种解决方案有两个不足：第一，让名为 LocalSqlServer 的连接字符串指向远程服务器让人感觉有点笨拙。第二，该选项不允许完全重

写已配置提供者的设置。在 web.config 文件中重新配置应用程序服务可以避免这两个问题，如下所述。

B.2.3 重写应用程序服务的设置

在 web.config 文件中重写不同提供者的设置具有极大的灵活性。它不仅允许将各种应用程序服务指向新的数据库连接，而且允许修改各提供者支持的其他设置。要重写 Planet Wrox 应用程序的设置，需要给每个已配置的提供者(Membership、Role Manager 和 Profile)提供它自己的<provider>设置。位于%windir%\Microsoft.NET\Framework\v4.0.30128\Config 的文件 machine.config 提供了不同提供者的示例。注意，您的 machine.config 文件的路径功能稍有不同。

如果要重写 Planet Wrox 应用程序的全部 3 个提供者，让它们使用标准的 PlanetWroxConnection-String1 而不是默认的 LocalSqlServer，需要将下面的配置信息添加到 web.config 文件中。当添加这些设置时，请确保重写了文件中已经存在的所有当前设置。为了让它能够运行，需要使用工具 aspnet_regsql.exe 为应用程序服务配置 PlanetWrox 数据库，如前所述：

```
<membership>
  <providers>
    <clear />
    <add name="AspNetSqlMembershipProvider"
      type="System.Web.Security.SqlMembershipProvider, System.Web, Version=4.0.0.0,
        Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="PlanetWroxConnectionString1"
      enablePasswordRetrieval="false"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="false"
      applicationName="/"
      requiresUniqueEmail="false"
      passwordFormat="Hashed"
      maxInvalidPasswordAttempts="5"
      minRequiredPasswordLength="6"
      minRequiredNonalphanumericCharacters="1"
      passwordAttemptWindow="10"
      passwordStrengthRegularExpression=""
    />
  </providers>
</membership>

<profile>
  <providers>
    <clear />
    <add name="AspNetSqlProfileProvider"
      connectionStringName="PlanetWroxConnectionString1"
      applicationName="/"
      type="System.Web.Profile.SqlProfileProvider, System.Web, Version=4.0.0.0,
        Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
    />
  </providers>
  <properties>
    <add name="FirstName"/>
    <add name="LastName"/>
    <add name="DateOfBirth" type="System.DateTime"/>
  </properties>
</profile>
```



```
<add name="Bio"/>
<add name="FavoriteGenres"
      type="System.Collections.Generic.List`1[System.Int32]"/>
</properties>
</profile>

<roleManager enabled="true">
  <providers>
    <clear />
    <add name="AspNetSqlRoleProvider"
          connectionStringName="PlanetWroxConnectionString1"
          applicationName="/"
          type="System.Web.Security.SqlRoleProvider, System.Web, Version=4.0.0.0,
              Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          />
  </providers>
</roleManager>
```

每个应用程序服务都有<providers>部分，它允许为服务定义一个或多个提供者。再次使用<clear/>元素从配置层次结构中删除所有以前定义的提供者。然后使用<add />元素添加每个提供者。注意，每个配置部分都使用 `connectionStringName="PlanetWroxConnectionString1"`来指向在相同的 web.config 文件中定义的自定义连接字符串。同时还要注意，可以修改提供者支持的所有其他设置。

所有提供者配置设置都有一个名为 `applicationName` 的特性，它设置为`/`。每个应用程序都使用这个名称在数据库中查找请求的用户、角色等。如果喜欢，也可以使用其他名称。但重要的是，所有已配置的提供者要使用相同的 `applicationName`，否则，它们就会查找错误的信息。如果没有匹配的 `applicationName`，服务就不能将记录从一个服务关联到另一个服务。从而导致数据库中用户、角色及其配置文件错配。

使用这些重新配置的应用程序服务，就不再需要在本附录前面或第 19 章添加到 web.config 文件中的 LocalSqlServer 连接字符串了。

保存经过修改的 web.config 文件之后，站点就可以继续正常运行。但是应用程序服务现在使用的是 PlanetWrox 数据库，而不是单独的 aspnetdb 数据库。

[General Information]
name=ASP.NET 4入门经典 涵盖C#和VB.NET 第6版
author=(美)史潘加斯著
bookcontentsStr=http://ndfe.5read.com/300-0
8/diskQJS/QJS1548/06/BookContents.dat;
dxid=000008016738
pages=682
pagesatt=0
pagesbok=1
pagescov=2
pagesfow=9
pagesleg=1
press=清华大学出版社
publishDate=2010.12
ssid=12730649
url=http://book1.duxiu.com/readDetail.jsp?d
xNumber=000008016738&d=1D7417394912CA3127B2
00A1F2084E93