

还在为学习 .NET技术发愁吗？350元等于什么？

还在为学习 .NET技术发愁吗？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！！ 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！！

需要的请联系QQ：2569343569  在线咨询

还在为学习.NET技术发愁吗？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！！！！需要的请联系QQ：2569342569

Microsoft

微软技术丛书

ASP.NET 4 从入门到精通

(美) George Shepherd 著
张大威 译

Step by Step

清华大学出版社

ASP.NET 4 从入门到精通

本书围绕ASP.NET 4, 引领读者进入Web开发世界

通过本书, 读者可以根据自身实际情况循序渐进地学习ASP.NET 基础知识。不论是Web开发方面的初学者, 还是刚接触ASP.NET 4的资深Web开发人员, 都可以通过本书掌握网站开发实用技能。

本书重要主题:

- 管理网站的视觉风格
- 构建导航结构
- 添加服务器端控件, 并了解每种控件的应用场景
- 开发Web窗体和登录页面
- 实现身份验证与授权
- 在数据驱动的应用程序中使用“动态数据”控件
- 管理会话状态与缓存
- 了解Microsoft Silverlight、ASP.NET MVC和AJAX
- 调试和部署Web应用

→ 练习文件下载地址:
<http://www.tup.com.cn>
<http://www.wenyuan.com.cn>

《微软技术丛书》包括以下几个子系列:

从入门到精通

- 适合新手程序员的实用教程
- 侧重于基础技术和特征
- 提供范例文件

技术内幕

- 权威、必备的参考大全
- 包含丰富、实用的范例代码
- 帮助读者熟练掌握微软技术

高级编程

- 侧重于高级特性、技术和解决问题
- 包含丰富、适用性强的范例代码
- 帮助读者精通微软技术

精通&宝典

- 着重剖析应用技巧, 以帮助提高工作效率
- 主题包括办公应用和开发工具

认证考试教材

- 完全根据考试要求来阐述每一个知识点
- 提供可供搜索的Ebook (英文版) 和训练题
- 提供实际场景、案例分析和故障诊断实验

Microsoft

ISBN 978-7-302-25284-9



9 787302 252849 >

定价: 69.00元

微软技术丛书

ASP.NET 4 从入门到精通

(美) George Shepherd 著
张大威 译

清华大学出版社
北 京

内 容 简 介

本书以 ASP.NET 应用程序开发为主题，全面介绍了 ASP.NET 4 的所有功能和特性。书中采用深受读者欢迎的 Step by Step 风格，指导读者通过具体的示例来熟悉和掌握 ASP.NET 4 的重要特性，并通过练习的方式来进一步呈现和演示，有助于读者采用 AJAX、WCF 服务、自定义控件和母版页等流行技术来创建出色的网站。

本书适合刚接触 ASP.NET 的初学者、希望了解 ASP.NET 4 新特性的读者和渴望了解 ASP.NET 工作原理的开发人员。

Microsoft ASP.NET 4 Step by Step(978-0-7356-2701-7)
Copyright © 2010 by George Shepherd
Original English Language Edition Copyright © 2010 by George Shepherd
Published by arrangement with the original publisher, Microsoft Press, a division of Microsoft Corporation, Redmond, Washington, U.S.A.

本书中文简体版由 Microsoft Press 授权清华大学出版社出版发行，未经出版者书面许可，不得以任何方式复制或抄袭本书的任何部分。

北京市版权局著作权合同登记号 图字：01-2010-7120

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，翻印必究。举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

ASP.NET 4 从入门到精通/(美)谢菲尔德(Shepherd, G.)著；张大威译. —北京：清华大学出版社，2011.6
(微软技术丛书)
书名原文：Microsoft ASP.NET 4 Step by Step
ISBN 978-7-302-25284-9

I. ①A… II. ①谢… ②张… III. ①网页制作工具—程序设计 IV. ①TP393.092
中国版本图书馆 CIP 数据核字(2011)第 057006 号

责任编辑：文开琪
装帧设计：杨玉兰
责任印制：王秀菊
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 装 者：北京鑫海金澳胶印有限公司
经 销：全国新华书店
开 本：185×260 印 张：33.75 字 数：818 千字
版 次：2011 年 6 月第 1 版 印 次：2011 年 6 月第 1 次印刷
印 数：1~4000
定 价：69.00 元

《微软技术丛书》出版前言

在黄昏里希冀皓月与繁星
在深夜希冀着黎明
在炎夏希冀凉秋
在严冬又希冀新春
这不断的希冀啊，
使我感触到世界的存在，
带给我多量的生命的力。
这样，
我才能跨过——

这黎明黄昏，黄昏黎明，春夏秋冬，秋冬春夏的茫茫的时间的大海啊。

——艾青

时间在流逝，技术也在迅猛发展。在希冀中，微软的.NET 战略早已经变成现实，带来全新、快速而敏捷的企业计算能力，也给软件开发商和软件开发人员提供了支持未来计算的高效 Web 服务开发工具。在希冀中，我们欣喜地看到，微软的每一个技术创新，都对中国开发人员产生巨大的推动作用，使得越来越多的人加入微软开发阵营。

微软出版社为了配合 Visual Studio 的推广和普及，邀请项目开发组的核心开发人员和计算机图书专业作家精心编写了微软 IT Pro 系列图书。该丛书自上市以来，在美国图书销量排行榜上一直高居前列，颇受读者好评，成为程序开发人员和网络开发人员了解微软技术的权威工具书。随着新的开发平台的发布，该系列得以大幅度扩充，在美国及欧洲图书市场广受好评。

从 2002 年开始，清华大学出版社为了满足中国广大程序开发人员、网络开发人员以及计算机用户学习最新技术的渴望，在微软出版社的配合下，先后推出了《微软.NET 程序员系列》和《微软.NET 程序设计系列》。这两套书阵容庞大，几乎涵盖.NET 技术及其应用的各个方面；也正因为如此，翻译和编辑加工的工作量也大得惊人。但为了保持国外优秀技术图书的魅力，同时使读者领会新技术的真谛，本丛书的翻译和编辑都是经过严格筛选的、具有很高的翻译水平或丰富编辑经验的技术人员。同时，我们还聘请微软公司相关产品组的技术专家审读每一本书，确保在技术上准确无误。

2005 年，随着微软新的开发平台的推出，我们将原有的两套丛书整合为《微软技术丛书》。这套丛书针对不同层次的读者，分为 5 个子系列：从入门到精通、技术内幕、高级编程、精通&宝典和认证考试教材。各系列特色如下：

★ 从入门到精通

- 适合新手程序员的实用教程
- 侧重于基础技术和特征
- 提供范例文件

★ 技术内幕

- 权威、必备的参考大全
- 包含丰富、实用的范例代码
- 帮助读者熟练掌握微软技术

★ 高级编程

- 侧重于高级特性、技术和解决问题
- 包含丰富、适用性强的范例代码
- 帮助读者精通微软技术

★ 精通&宝典

- 着重剖析应用技巧, 以帮助提高工作效率
- 主题包括办公应用和开发工具

★ 认证考试教材

- 提供完整的 Ebook(英文版)
- 提供实际场景、案例分析和故障诊断实验
- 完全根据考试要求来阐述每一个知识点

这套丛书延续以前严谨的编校风格, 一切以保证图书内容和技术质量为核心, 付出了大量心血。相信整合后的这套丛书必然会帮助程序开发人员、网络开发人员以及具有一定编程基础的中高级读者, 快速、全面地掌握微软技术, 为将来的技术生涯奠定扎实的基础, 使之成为中国软件产业的栋梁!

为增强本书的可读性, 便于读者迅速定位关键术语的原文和快速根据索引来定位知识点(概念、函数等)的详细介绍, 有些经典图书中在相应位置标注了原书页码(在当前行末尾用粗体方括号【】或椭圆形底纹表示), 并在书后附上原书索引, 以期能对大家提供更多的帮助。已经采用这一体系设计的图书有《Windows 核心编程(第5版)》、《Visual C# 2008 从入门到精通》、《ASP.NET 3.5 核心编程》、《Visual C# 2008 核心编程》、《精通 Windows 3D 图形编程》和《CLR via C#(第3版)》。

在此, 感谢参与本丛书的翻译和审校人员, 感谢他们付出的心血和时间。他们来自培训和实践前沿, 具有深厚的技术底蕴和文化素养, 善于用浅显易懂的语言阐述晦涩难懂的技术细节。同时也要感谢这一年来时刻关注这套书的读者朋友们。他们热心地提出自己的意见和建议, 感谢他们的宽容和善意关爱。我们将和大家一样, 时刻关注微软技术发展的最新动态, 时刻保持自己的技术动力!

亲爱的读者朋友, 期待着您把每一次看书的机会, 都当成增进知识的时候。这个过程, 绝对不是浅尝辄止, 更非自认把书看过一两遍就可以了。深度的阅读是尽可能地把书本的知识转换为自己熟悉的, 甚至读到自己内心的深处。同时, 也请把您对这套书的感受告诉我们, 我们期待着和您分享, 联系信箱 coo@netease.com。

尽管我们注入大量心血, 但疏忽纰漏之处在所难免, 恳请读者朋友提出建议和批评。本丛书在创作、翻译和编辑过程中得到了微软(中国)公司的大力支持。本丛书能够顺利出版, 更是倾注了无数幕后人员的汗水和心力。在此, 对他们的辛勤劳动一并表示衷心感谢!

清华大学出版社

译者序

近些年网络正逐步充斥人们生活的方方面面，各类 Web 应用和依赖于这些应用的应用层出不穷，使 Web 世界更加丰富多彩，并承载着无数商业传奇。如国外的 Amazon、Facebook、Google、Twitter、Yahoo 等，国内的百度、淘宝、腾讯等，都是 Web 的受益者。因而也就不难想象，Web 开发是软件行业最为活跃的领域之一。

近些年“云计算”的概念猛然兴起，更为 Web 世界注入了新的血液。IBM 副总裁 Irving Wladawsky Berger 甚至认为“云计算就是将以以前那些需要大量软硬件投资以及专业技术能力的应用，以基于 Web 服务的方式提供给用户”。可见 Web 开发在云计算中也同样具有重要地位。

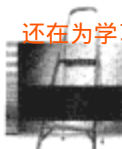
从服务器到客户端，Web 应用程序所涉及的技术十分广泛，而相关技术也层出不穷。在微软平台下，从 Internet Explorer 9、HTML 5 和 Silverlight 这样能够为客户端带来丰富体验的客户端技术，到 Windows Azure 和 SQL Server Azure 这样的云平台技术，再到 SharePoint 这样的企业级应用，都能见到 ASP.NET 的身影。

从最初的 ASP 发布到现在的 ASP.NET 4，微软 Web 开发技术已经经历了十余年的演变和改进，如今成为当今主流的 Web 开发技术之一。ASP.NET 门槛低、学习曲线平滑等诸多优点，使其非常适合作为 Web 开发的入门工具。

有人或许觉得 ASP.NET 更新太快，担心紧跟潮流力不从心。其实大可不必担心，因为新的版本不会完全推翻之前的版本，大多数变化在于为提供更有力的工具而对原有部分进行扩充(至少到目前为止仍然是这样)。不论哪个版本，都会有相同的东西需要经历。如果选择了 ASP.NET，不妨现在就有一个开端。

关于本书

本书是 ASP.NET 的基础教程，由浅入深、内容丰富，从不同角度介绍了基于 ASP.NET 的 Web 应用程序开发。书中包含大量示例，手把手地帮助读者理解 ASP.NET 中的各种功能。除了常见的功能外，本书还介绍了一些高级技术和 ASP.NET 4 的新特性。主要内容包括：ASP.NET 原理、服务器控件、配置、登录、数据绑定、网站导航、个性化、Web 部件、会话状态、缓存、调试技术、HTTP 模块与处理程序、动态数据、WPF、Silverlight、ASP.NET MVC 框架、AJAX、WCF 和部署。



本书的使用

作为译者，为了使读者通过本书能够真正受益，本人在这里给出几点建议。为充分理解本书内容，应至少掌握 C#编程语言，了解.NET Framework 的常见数据类型，并对 HTML 有初步认识。本书的第 I 部分介绍 ASP.NET 的原理和基础知识，这对理解后续内容大有裨益，建议重点阅读。对于书中的示例和练习，建议读者朋友能够分析其真正意图，灵活变通，在操作中遇到问题时能够积极思考并不断尝试。

虽然我们极力避免翻译过程中的失误，并在征得作者同意的前提下更正了原书中存在的一些问题，但仍很难尽善尽美。加之本人英文和技术水平有限，错误在所难免，敬请各位读者不吝赐教。如果读者有关于本书内容的问题需要与本人沟通，请发邮件至 david@david-zhang.net。

致谢

本书与读者见面离不开许多人的努力与心血。在此，我要感谢我的家人对我的支持，感谢我的父亲和奶奶从千里之外到北京照顾我的生活，使我能够更专注于本书的翻译。感谢清华大学出版社的工作人员，本书的出版与这些人的汗水密不可分。感谢 O'Reilly 出版社的 Chris Olson 为了解答和确认我的问题使我与原书的作者和编辑取得联系。感谢北京中软资源信息科技有限公司的 Windows Phone 7 Intl 组的同事们对我翻译工作的理解与关心。

最后，感谢读者朋友对我们一如既往的支持。希望您能通过本书走进丰富多彩的 Web 世界，并在激烈的竞争中储备新的力量！

张大威

于北京



致 谢

上次我为本书的前一版写致谢的时候，我提到我的儿子 Ted 如何使用 HTML 为我制作父亲节卡片。如今，Ted 已经上了大学，我还记得他在高中时查找和申请学校的经历。他完全都是在网上完成的。这与我过去申请学校的方式完全不同！

Web 已经渗透到社会基础设施的方方面面。不论是希望促进业务影响力的商务人士、希望购买纸质图书的读者、要从学校网站下载作业任务的学生，还是其他信息的生产者或消费者，都会接触到 Internet。

本书的出版颇具艰辛。虽然我的名字出现在封面上，但我只是团队中的一员。本书的出版得到了许多人的帮助和支持。

感谢 Claudette Moore 再次让我与微软出版社(Microsoft Press)合作。Claudette 一直代表我与微软出版社沟通，处理各种业务问题，使我能够专心写作。感谢 Maria Gargiulo 在项目管理方面的工作，非常高兴与你合作。感谢 Charlotte Twiss 将示例代码制作成光盘。感谢 Steve Sagman 对本书精妙的组织。感谢 Christina Yeager 对本书的审校，梳理句子逻辑，并为本书编写索引。他们在编辑、制作和排版上的工作非常出色。感谢 Kenn Scribner 用他独到的技术视角给我以提示。感谢 Ben Ryan 接受我的申请并聘用我撰写本书。

感谢 Jeff Duntemann 录用并让我在 *PC Tech Journal* 杂志上发表我的处女作。感谢 JD Hildebrand 录用并发表我的第二篇文章，并且在 Oakley Publishing 与你的合作非常愉快。感谢 Sandy Daston 在我写作生涯早期对我的支持和指导。感谢 DevelopMentor 的同事们，这是一个出色的团队，在这里我学到了很多新技术。感谢 Schwab Performance Technologies 的朋友们。

感谢我的挚友 Pat Shepherd 和他的家人：Michelle、Belfie 和 Bronson。感谢我的儿子 Ted Shepherd，你是我永远的骄傲。感谢我的父母 George Robbins Shepherd 和 Betsy Shepherd，你们总是鼓励我，让我做得最好。我非常想念你们！

最后，感谢读者朋友对本书的支持，并花费精力学习 ASP.NET。希望您能够继续探索 ASP.NET，并找到更好、更有趣的方式来处理 HTTP 请求。

前言

本书将帮助您学习使用现今 Microsoft 最新的 HTTP 请求处理框架 ASP.NET 4 来编写 Web 应用程序。20 世纪 90 年代第一个网站的出现掀起了 Web 开发的浪潮。虽然 Web 开发目前有多种开发工具可供选择，但在过去几年里，ASP.NET 逐渐成为最一致、稳定、功能齐全的 HTTP 请求管理框架之一。

ASP.NET 和 Microsoft Visual Studio 中包含许多强大的功能，能够大大简化 Web 开发者的工作。例如，Visual Studio 提供了多种用于网站开发的项目模板。Visual Studio 支持许多开发模式，包括通过 Microsoft Internet 信息服务(IIS)在开发过程中直接测试我们的网站、使用内建的 Web 服务器，以及通过 FTP 连接来进行开发。使用 Visual Studio 中的调试器，我们可以运行网站，并通过单步调试找到代码的症结。使用 Visual Studio 中的“设计器”，我们可以将控件元素拖放至画布上，以可视方式来设计用户界面。在需要部署应用程序时，还可以通过 Visual Studio 方便地创建部署包。而这些特性只是 ASP.NET 和 Visual Studio 的一小部分。

本书旨在介绍 ASP.NET 开发。每一部分针对 ASP.NET 的一个特定功能，并配有易于理解的示例。本书将引领读者一步一步地完成每个练习。对于大多数 ASP.NET 的主要功能，这里都将通过简明的示例反复加以演示。示例可能会演示多种功能，而并不针对某一种。除了通过示例来展示 ASP.NET 的强大功能外，本书还为每个功能提供了实际的应用程序，以便读者可以通过练习来实际应用这些技术。阅读本书并进行相应的练习可以帮助您奠定扎实的基础，以此来实际构建包含 AJAX、WCF 服务、自定义控件和母版页(master page)这些流行技术的网站。

读者可以按需要单独阅读某一章。除第 1 章和介绍服务器控件的 3 个章节(第 3 章到第 5 章，这 3 章一起阅读效果更佳)之外，每章都只讲述 ASP.NET 的单个特定功能。此外，出于完整性的考虑，第 1 章还包含 IIS 与 ASP.NET 之间协作的内容。

本书读者对象

本书适用于以下 3 类读者。

- **对 ASP.NET 较为陌生的初学者**


本书包含足够多的背景信息来讲述 Web 开发，即便是那些只开发过桌面应用程序的开发者也同样适用。



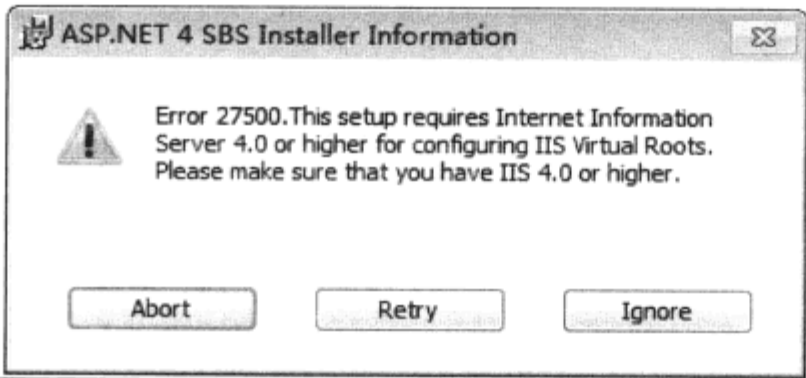
- **希望从 ASP.NET 1.x、2.0、3.x 或传统 ASP 迁移过来的开发者**
本书介绍了 ASP.NET 4^①与 ASP.NET 之前版本间的差异,以及 ASP.NET 与传统 ASP 的不同之处。
- **希望通过分门别类的方式获得 ASP.NET 过程性知识(how-to knowledge)的开发者**
这类读者无需为获得所需要的内容而进行通读。本书中的章节多数是独立的(第 1 章除外,它介绍了 Web 应用程序的基础知识,如果以前只接触过桌面应用程序开发,不妨先阅读一下这一章)。最好能够一并阅读有关服务器控件的 3 章内容(第 3 章到第 5 章),但不是绝对必要。

准备工作

至此,读者您可能已经做好编写代码的准备了。

 **注意** 在开始之前,请确认以下事项:

- 计算机上已安装有 Visual Studio 2010
只要安装了开发环境,便可以确保 .NET 运行库也已安装
- 具有 Administrator(管理员)权限
详见本前言稍后的“C#示例代码的安装”一节
- IIS 已安装在您的计算机上,并处于运行状态
第 1、2、9 和 26 章的示例代码要用到 IIS。为在 Windows 7^②上安装 IIS,可以依次单击“开始”|“控制面板”。在“控制面板”中,单击“程序和功能”|“打开或关闭 Windows 功能”。在“Windows 功能”对话框中,展开“Internet 信息服务”节点,选择“Web 管理工具”和“万维网服务”旁边的复选框^③,然后单击“确定”按钮。
如果试图在未运行 IIS 的情况下安装代码,则会看到如下提示。如果要忽略该错误信息,可以单击 Ignore 按钮使安装继续进行。



① 译者注: 请注意, ASP.NET 4 的数字后面没有任何额外的字符。
② 译者注: 如果使用 Windows Server 2008, 请参考 MSDN 文档——“在装有 IIS 7.0 和 Visual Studio 的 Windows Server 2008 上运行 Web 应用程序”, 网址为 [http://msdn.microsoft.com/zh-cn/library/bb763178\(VS.90\).aspx](http://msdn.microsoft.com/zh-cn/library/bb763178(VS.90).aspx)。
③ 译者注: 还要特别选中“Internet 信息服务\万维网服务\应用程序开发功能\ASP.NET”复选框。

前几个示例项目除文本编辑器和已运行的 IIS 外，不需要其他工具。首先，通过一些基本的示例，您将了解到 ASP.NET 面向对象的特性和编译模型。除了清楚地了解 ASP.NET 如何处理请求外，也可以从一个较高层面鸟瞰 ASP.NET 架构。随后，本书将带您掌握 Web 窗体的编程，并开始使用 Visual Studio 来编写代码——这个工具使得开发变得更加容易！

在学习 Web 窗体开发后，您还将通过示例了解到 ASP.NET 的其他特性，如服务器控件、内容缓存、自定义处理程序、输出与数据缓存、调试与诊断，以及 ASP.NET 对 Web 服务的支持。

本书阅读建议

本书旨在帮助读者掌握多方面技能。不论您是 Web 编程的初学者，还是从其他 Web 开发平台迁移过来，本书都适用。可以从下表给出的建议入手。

读者背景	具体建议
不熟悉 Web 开发	<div>1. 安装示例代码</div> <div>2. 按顺序学习第 1 章和第 2 章的示例。这些示例将帮助您奠定 Web 开发的基础，也将使您熟悉 ASP.NET 和 Visual Studio</div> <div>3. 按需要完成本书剩余部分的学习</div>
不熟悉 ASP.NET 和 Visual Studio	<div>1. 安装示例代码</div> <div>2. 学习第 2 章的各示例。这些示例将为您使用 ASP.NET 和 Visual Studio 奠定基础</div> <div>3. 按需要完成本书剩余部分的学习</div>
从 ASP.NET 的其他版本迁移过来	<div>1. 安装示例代码</div> <div>2. 略读前两章，了解在 Microsoft 环境中和使用 Visual Studio 2010 进行 Web 开发的基础知识</div> <div>3. 将重点放在第 3 章到第 26 章中的必要部分。您可能已经熟悉其中的某些主题，这样则只需要了解某一功能在 ASP.NET 的当前版本中与在之前版本中有何差异。有时可能需要探究 ASP.NET 4 特有的新功能</div>
在做过所有练习后，将本书作为参考	<div>1. 通过目录找到相应主题</div> <div>2. 通过阅读每章末尾的“快速参考”部分来找到该章节中重点语法或技术的简要回顾</div>

本书的约定与特性

本书的约定旨在使内容更易于理解。在您开始阅读本书之前，请先参考下面的列表。这个列表给出了书中的所有约定，以及您可能会用到的特性。

约定

- 每章的开始部分都有学习目标。
- 每个练习都包含一系列任务。每项任务又会被表示为一系列步骤，这些步骤需要按顺序进行。
- “技巧”部分会提供额外的内容或完成某一步骤的备选方法。
- “注意”部分会给出一些安装和使用配套资源中示例代码的注意事项。
- 需要输入的文本会像下面这样加粗显示：

```
static void Main(string[] args)
{
    System.Console.WriteLine("HelloWorld");
}
```

- 除给定的步骤外，还可能有其他方法，殊途同归。例如，为在某个 Visual Studio 项目中添加新项，可以使用主菜单，也可以在“解决方案资源管理器”中右击。
- 本书中的示例都使用 C# 语言编写。

其他特性

- 有些主题包含“补充内容”和“提示”部分，以对该主题进行更深入地说明。补充内容可能包含背景信息、设计建议，或者与所讨论的功能相关的特性。其中还可能给出某一功能在 ASP.NET 的当前版本中与在之前版本中的差异。
- 每章末尾的“快速参考”部分将列出该章节主要任务的简要提示。


预发布的软件

本书在定稿前一周根据 Visual Studio 2010 发布候选版(release candidate, 简称 RC)做了复审和测试。我们基于 Visual Studio 2010 发布候选版对示例代码进行了回顾和测试。读者可能会发现本书中的示例、文字和截图与 Visual Studio 2010 正式版存在细微差异。然而，我们希望这种差异能够降至最低。^①

① 译者注：在本书中，本人使用 Visual Studio 2010(版本号为 10.0.30319.1 RTMRel)和 .NET Framework 4(版本号 4.0.30319 RTMRel)的正式版逐一核对了原书中的文字、插图和示例代码。如果发现与原书存在某些不一致，本人会相应给出脚注加以说明，并以最新版本为准适当修改原书内容。

硬件与软件要求

为完成本书的练习，需要具备以下硬件和软件。

 **注意** 本书并不包含 Visual Studio 2010 软件！配套资源只包含完成练习所需的示例代码。Visual Studio 2010 软件必须单独购买。

- Windows 7、Windows Server 2003、Windows Server 2008 或 Windows Vista。
- Internet 信息服务(包含于 Windows 中)。要求 IIS 5.1 或更新的版本。在本书英文版截稿前，最新版本是 IIS 7.5^①。
- Microsoft Visual Studio 2010 Ultimate、Visual Studio 2010 Premium 或 Visual Studio 2010 Professional。
- Microsoft SQL Server 2008 Express(Visual Studio 2010 中包含该软件)或 SQL Server 2008(在本书英文版截稿前，SQL Server 2008 R2 是最新版本)。
- 最低 1.6 GHz 的 Pentium 或兼容的处理器。
- x86 架构至少需要 1 GB 的内存。
- x64 架构至少需要 2 GB 的内存。
- 如果在虚拟机中运行，则额外需要至少 512 MB 的内存。
- 支持 DirectX 9 的显卡，分辨率至少为 1024 × 768。
- 最低 5400 转/分的硬盘，并具有至少 3 GB 的可用磁盘空间。
- DVD-ROM 驱动器。
- Microsoft 鼠标或兼容的指点设备。
- 为安装示例代码，需要 5 MB 的可用磁盘空间。

为配置 Microsoft SQL Server 2008 Express，还需要具有计算机的 Administrator 权限。

示例代码

本书的配套资源可从 www.wenyuan.com.cn 下载，其中包含书中练习所要使用的示例代码

^① 译者注：截止到本书截稿时，只有 Windows 7 和 Windows Server 2008 R2 支持 IIS 7.5，但后者尚无 32 位版本。除非特殊说明，本书的操作步骤说明将以 Windows 7 搭载的 IIS 7.5 为准。

(使用 C# 编写)。使用这些示例代码，读者可以不必花时间创建与练习无关的文件。每个示例当中的文件和步骤说明将帮助读者完成练习，这也是掌握新技术简单而有效的方法。

C# 示例代码的安装

在计算机上安装 C# 示例代码后，便可以结合书中练习来使用它。为此，请遵循以下步骤。

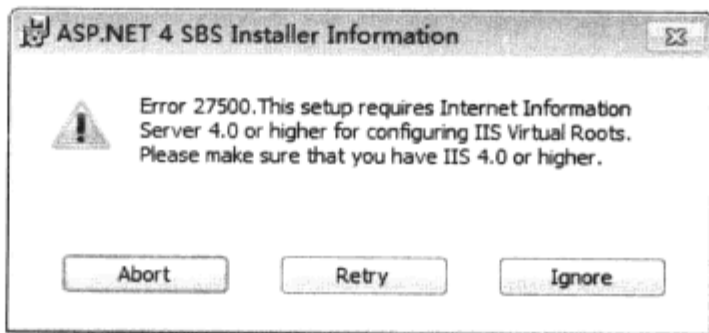
注意 在开始之前，请确认以下事项：

- 拥有计算机的 Administrator 权限
- 已在计算机上安装 IIS，并使其处于运行状态

第 1、2、9 和 26 章的内容要用到 IIS，并且配套的示例代码也要求使用 IIS。示例代码的安装程序需要修改 IIS 设置。对 IIS 进行操作需要用户具有 Administrator 权限。如果使用家用的计算机，则可能已具有 Administrator 权限。如果使用某个组织的计算机，但没有 Administrator 权限，则需要咨询计算机支持人员或 IT 人员。

为在 Windows 7 上安装 IIS，可以依次单击“开始”|“控制面板”。在“控制面板”中，单击“程序和功能”|“打开或关闭 Windows 功能”。在“Windows 功能”对话框中，展开“Internet 信息服务”，选中“Web 管理工具”和“万维网服务”复选框，单击“确定”按钮。

如果试图在未运行 IIS 的情况下安装代码，则会看到以下提示。如果要忽略该错误信息，可以单击 Ignore 按钮来使安装继续进行。



1. 访问 <http://www.wenyuan.com.cn>，找到本书。下载本书配套资源。
2. 选择 Code。InstallShield 向导将帮助您完成整个安装过程。
3. 阅读最终用户许可协议。如果同意各项条款，则选择接受选项，然后单击 Next。
4. 接受默认设置便可安装示例代码。

示例代码会安装在计算机的以下位置：

C:\Microsoft Press\ASP.NET 4 Step by Step\

另外，如果 IIS 处于运行状态，打开“Internet 信息服务管理器”会发现，安装程序在 Default Web Site 创建了一个名为 asp.net4sbs 的虚拟目录。这个虚拟目录包含许多 Web 应用程序。

示例代码的使用说明

本书中的每一章都说明了使用配套示例代码的时机和方法。在需要使用示例代码时，本书会给出如何打开相关文件的步骤。有些章节需要创建全新的项目，以便读者能够理解整个开发过程。有些示例会依赖于之前示例的代码。

下面是示例代码中各项目的完整列表。

项 目	说 明
第 1 章 HelloWorld.asp, Selectnoform.asp, Selectfeature.htm, Selectfeature2.htm, Selectfeature.asp	这些 Web 资源演示了不同类型原始 HTTP 请求的形式
WebRequestor	这是一个能够发送原始 HTTP 请求的应用程序，非常简单
第 2 章 HelloWorld, HelloWorld2, HelloWorld3, HelloWorld4, HelloWorld5, partial1.cs, partial2.cs	这些 Web 资源演示了 ASP.NET 中的编译模型和分部类
第 3 章 BunchOfControls.htm, BunchOfControls.asp, BunchOfControls.aspx	这些 Web 资源演示了控件标记的呈现过程
ControlsORama	这个基于 Visual Studio 的项目演示了 Visual Studio 和服务器控件的使用
第 4 章 ControlsORama	该项目是对第 3 章项目的扩展，演示了服务器控件的创建和使用
第 5 章 ControlsORama	该项目是对第 4 章项目的扩展，演示了复合服务器控件和用户控件的创建和使用
第 6 章 ControlPotpourri	该项目演示了控件的验证、TreeView、Image、ImageButton、ImageMap 和 MultiView/View 控件
第 7 章 MasterPageSite	该项目演示了在单个 Web 应用程序中如何通过母版页、主题和皮肤来实现一致的外观和用户体验
第 8 章 ConfigORama	该项目演示了 ASP.NET 的配置方法，其中包括如何管理 web.config 文件、如何添加新的配置元素，以及如何获取那些配置元素

续表

项 目	说 明
第 9 章 SecureSite Login.aspx, OptionalLogin.aspx, Web.Config, Web.ConfigForceAuthentication, Web.ConfigForOptionalLogin	该项目演示了网站的“Forms 身份验证”和授权 这些 Web 资源进一步演示了“Forms 身份验证”
第 10 章 DataBindORama	该项目演示了几种控件的数据绑定(其中包括 GridView)、几种 DataSource 控件，以及以 XML 和 XML 架构的形式来加载和保 存数据集
第 11 章 NavigateMeSite	该项目演示了 ASP.NET 导航方面的不同特性
第 12 章 MakeItPersonal	该项目演示了 ASP.NET 个性化方面的不同特性
第 13 章 UseWebParts	该项目演示了 Web 应用程序中不同“Web 部件”的使用
第 14 章 SessionState	该项目演示了 Web 应用程序中会话状态的使用
第 15 章 UseDataCaching	该项目演示了如何通过缓存数据来增强性能
第 16 章 OutputCache	该项目演示了如何通过缓存输出来增强性能
第 17 章 DebugORama	该项目演示了 Web 应用程序的调试与跟踪
第 18 章 UseApplication	该项目演示了如何将全局应用程序对象和 HTTP 模块作为应用 程序的集合点(rendezvous point)，还演示了如何存储全局范围的 数据和处理应用程序范围的事件
第 19 章 CustomHandlers	该项目分别演示了通过单独的程序集和 ASHX 文件来自定义 HTTP 标头的方法
第 20 章 DynamicDataLinqToSQLSite	演示了如何使用“ASP.NET 动态数据”来创建数据驱动的网站
第 21 章 XAMLORama	该项目演示了如何在网站中使用松散 XAML
XBAPORama	该项目演示了如何创建“XAML 浏览器应用程序”(XBAP)
第 22 章 MVCORama	该项目演示了如何创建和管理基于 MVC 的网站(包含数据库)

续表

项 目	说 明
第 23 章 AJAXORama	该项目演示了如何通过 AJAX 来增进最终用户的体验
第 24 章 SilverlightSite SilverlightLayout SilverlightAnimations SilverlightAndWCF	该项目演示了如何在 ASP.NET 网站中包含 Silverlight 内容 该项目演示了 Silverlight 布局面板的使用 该项目演示了 Silverlight 中动画的使用 该项目演示了 Silverlight 组件如何通过 WCF 与网站进行通信
第 25 章 WCFQuotesService	该项目演示了如何创建和使用 ASP.NET WCF 服务
第 26 章 DeployThisApplication	该项目演示了新的“ASP.NET 打包”系统

所有这些项目都以完整解决方案的形式存在，方便实际练习和事后参考之用。

示例代码的卸载

为从计算机上卸载示例代码，请遵循以下步骤。

1. 在“控制面板”中打开“程序和功能”。
2. 在列表中选择 ASP.NET 4 SBS。
3. 单击“卸载”。
4. 根据提示操作便可以卸载示例代码。

本书的支持

我们尽了最大努力来确保本书及配套内容的准确性，但疏漏在所难免。收集的勘误和变更会被添加到“微软知识库”(Microsoft Knowledge Base)中。微软出版社在以下网站提供其图书和配套内容的支持：

<http://www.microsoft.com/learning/support/books/>


如果您对本书或配套内容有意愿、疑问或看法，或您的问题在上面提到的网站中未得到解决，请将其通过电子邮件发送至微软出版社，邮件地址是 mspinput@microsoft.com。

需要注意的是，上述地址不提供对微软软件产品的支持。

反馈

我们希望收到来自您对本书的反馈。请通过简短的在线调查来分享您的意见和看法，网址为 <http://www.microsoft.com/learning/booksurvey>。

您的参与将帮助微软出版社出版更符合您的需要和意愿的图书。

 **注意** 我们希望您在我们的在线调查中提供详细的反馈内容。如果你有关于出版计划、新书或针对微软出版社的问题，希望您通过 Twitter 与我们进行互动，网址是 <http://twitter.com/MicrosoftPress>。关于支持方面的事宜，请使用上述邮件地址。



目 录

第 I 部分 基础知识

第 1 章 Web 应用程序基础.....3	2.4 ASP.NET HTTP 管线.....39
1.1 HTTP 请求.....4	2.4.1 IIS 5.x 和 IIS 6.x 的管线.....39
1.1.1 从浏览器发出的 HTTP 请求.....4	2.4.2 IIS 7.x 集成的管线.....40
1.1.2 在不使用浏览器的情况下生成请求.....5	2.4.3 管线内部的组件.....40
1.2 超文本标记语言.....7	2.5 Visual Studio 与 ASP.NET.....42
1.3 动态内容.....9	2.5.1 本地 IIS 网站.....42
1.3.1 HTML 表单.....9	2.5.2 基于文件系统的网站.....43
1.3.2 公共网关接口——非常陈旧的技术.....11	2.5.3 FTP 网站.....43
1.3.3 Microsoft 环境下的 Web 服务器.....11	2.5.4 远程网站.....44
1.3.4 Internet 信息服务.....11	2.5.5 Hello World 与 Visual Studio.....44
1.3.5 Internet 服务器应用程序编程接口 DLL.....12	2.6 快速参考.....49
1.3.6 “Internet 信息服务”的运行.....13	第 3 章 页面的呈现模型.....51
1.4 传统的 ASP: ASP.NET 的前身.....16	3.1 将控件呈现为标签.....51
1.5 Web 开发思想.....18	3.2 将界面元素包装成组件.....53
1.6 ASP.NET.....19	3.2.1 ASP.NET 页面.....54
1.7 快速参考.....19	3.2.2 页面的呈现模型.....55
第 2 章 ASP.NET 应用程序基础.....21	3.2.3 页面的控件树.....57
2.1 经典的 Hello World 程序.....22	3.3 使用 Visual Studio 添加控件.....58
2.1.1 可执行代码与 HTML 的混合.....26	3.4 快速参考.....66
2.1.2 服务器端的可执行块.....28	第 4 章 自定义控件.....69
2.2 ASP.NET 编译模型.....33	4.1 Control 类.....69
2.3 编码风格.....35	4.2 Visual Studio 与自定义控件.....70
2.3.1 ASP.NET 1.x 风格.....35	4.3 回文验证器.....77
2.3.2 现代 ASP.NET 风格.....37	4.4 控件与事件.....80
	4.5 HtmlTextWriter 与控件.....82
	4.6 控件与视图状态.....84
	4.7 快速参考.....87

第 5 章 复合控件.....	89	6.2 页面验证的工作方式.....	110
5.1 复合控件与自定义控件.....	89	6.2.1 客户端验证.....	110
5.2 自定义的复合控件.....	90	6.2.2 服务器端验证.....	111
5.3 用户控件.....	97	6.3 其他验证控件.....	113
5.4 这两种控件的适用范围.....	103	6.4 验证控件的属性.....	113
5.5 快速参考.....	104	6.5 基于图片的控件.....	114
第 6 章 常用控件介绍.....	105	6.6 TreeView.....	117
6.1 验证控件.....	105	6.7 MultiView.....	119
		6.8 快速参考.....	122

第 II 部分 高级特征

第 7 章 一致的界面.....	125	9.3 用户的管理.....	168
7.1 用户界面一致性的管理.....	125	9.4 ASP.NET 登录控件.....	173
7.2 ASP.NET 母版页.....	126	9.5 用户的授权.....	175
7.3 主题.....	135	9.6 快速参考.....	177
7.4 皮肤.....	138	第 10 章 数据绑定.....	179
7.5 快速参考.....	140	10.1 在不使用数据绑定的情况下显示 集合的内容.....	179
第 8 章 配置.....	141	10.2 通过数据绑定来显示集合.....	180
8.1 Windows 的配置机制.....	142	10.2.1 基于 ListControl 的控件.....	180
8.2 .NET 的配置机制.....	142	10.2.2 TreeView 控件.....	181
8.2.1 Machine.Config.....	143	10.2.3 Menu 控件.....	181
8.2.2 配置节处理程序.....	143	10.2.4 FormView 控件.....	181
8.2.3 web.config.....	144	10.2.5 GridView 控件.....	181
8.2.4 ASP.NET 1.x 的配置管理.....	146	10.2.6 DetailsView 控件.....	181
8.2.5 ASP.NET 后续版本的 配置管理.....	146	10.2.7 DataList 控件.....	182
8.3 在 IIS 中配置 ASP.NET.....	150	10.2.8 Repeater 控件.....	182
8.4 快速参考.....	155	10.3 简单数据绑定.....	182
第 9 章 登录.....	157	10.4 数据库的访问.....	186
9.1 基于 Web 的安全性.....	157	10.5 .NET 对数据库的支持.....	187
9.1.1 IIS 的保护.....	158	10.5.1 连接.....	187
9.1.2 基本的“Forms 身份验证”.....	159	10.5.2 命令.....	189
9.2 ASP.NET 身份验证服务.....	164	10.5.3 结果的管理.....	190
9.2.1 FormsAuthentication 类.....	164	10.6 ASP.NET 数据源.....	192
9.2.2 可选的登录页面.....	165	10.7 其他数据绑定控件.....	196
		10.8 LINQ.....	202
		10.9 快速参考.....	204

第 11 章 网站的导航.....	205	12.2.1 用户配置文件.....	224
11.1 ASP.NET 对导航的支持.....	205	12.2.2 个性化提供程序.....	224
11.1.1 导航控件.....	205	12.3 个性化功能的使用.....	224
11.1.2 XML 站点地图.....	206	12.3.1 在 web.config 中定义配置	
11.1.3 SiteMapProvider.....	207	文件.....	225
11.1.4 SiteMap 类.....	207	12.3.2 配置文件信息的使用.....	225
11.1.5 SiteMapNode.....	208	12.3.3 配置文件变更的保存.....	226
11.2 导航控件的使用.....	208	12.3.4 配置文件与用户.....	226
11.2.1 Menu 控件与 TreeView		12.4 快速参考.....	231
控件.....	208	第 13 章 Web 部件.....	233
11.2.2 SiteMapPath 控件.....	209	13.1 “Web 部件” 简史.....	234
11.2.3 站点地图的配置.....	210	13.2 “Web 部件” 的优点.....	234
11.3 实现可导航的网站.....	211	13.3 “Web 部件” 控件的开发.....	234
11.4 SiteMapResolve 事件的捕获.....	214	13.3.1 “Web 部件” 页面的开发	235
11.5 为节点添加自定义特性.....	215	13.3.2 “Web 部件” 应用程序的	
11.6 安全性调整.....	217	开发.....	235
11.7 URL 映射.....	217	13.4 “Web 部件” 的架构.....	235
11.8 URL 重写.....	221	13.4.1 WebPartManager	
11.9 快速参考.....	221	与 WebPartZone.....	236
第 12 章 个性化.....	223	13.4.2 内建的区域.....	236
12.1 为访客提供个性化服务.....	223	13.4.3 内建的“Web 部件”.....	236
12.2 ASP.NET 中的个性化.....	224	13.5 “Web 部件” 的开发.....	243
		13.6 快速参考.....	250

第 III 部分 状态管理与缓存

第 14 章 会话状态.....	253	14.6.1 通过 Cookie 跟踪会话状态.....	268
14.1 何为会话状态.....	253	14.6.2 通过 URL 跟踪会话状态.....	269
14.2 ASP.NET 与会话状态.....	254	14.6.3 自动检测.....	269
14.3 会话状态简介.....	255	14.6.4 使用设备配置文件.....	269
14.4 会话状态与复杂的数据类型.....	259	14.6.5 会话状态超时.....	269
14.5 会话状态的配置.....	265	14.7 会话的其他设置.....	269
14.5.1 禁用会话状态.....	266	14.8 Wizard 控件——会话状态的一种	
14.5.2 在进程中存储会话状态.....	266	替代方案.....	270
14.5.3 在状态服务器中存储会话		14.9 快速参考.....	276
状态.....	267	第 15 章 应用程序数据的缓存.....	279
14.5.4 在数据库中存储会话状态.....	267	15.1 前期准备.....	279
14.6 会话状态的跟踪.....	268	15.2 数据缓存的使用.....	281



15.3	缓存的影响	283	16.2	缓存内容的管理	302
15.4	缓存的管理	284	16.2.1	OutputCache 指令的使用	302
15.4.1	内存中的 DataSet	285	16.2.2	HttpCachePolicy	306
15.4.2	缓存过期	288	16.2.3	缓存的位置设置	307
15.4.3	缓存依赖项	290	16.2.4	输出缓存依赖项	308
15.4.4	SQL Server 依赖项	293	16.2.5	缓存配置文件	308
15.4.5	缓存项的清除	294	16.3	用户控件的缓存	309
15.5	快速参考	297	16.4	适合应用输出缓存的场景	311
第 16 章	输出缓存	299	16.5	其他缓存提供程序	312
16.1	页面内容的缓存	299	16.6	快速参考	312

第 IV 部分 诊断与插件

第 17 章	诊断与调试	317	18.4.4	Application_BeginRequest	339
17.1	页面跟踪	317	18.4.5	Application_Authenticate-Request	340
17.1.1	跟踪	317	18.4.6	Session_Start	340
17.1.2	跟踪语句	320	18.4.7	Session_End	340
17.2	应用程序跟踪	323	18.5	HttpApplication 的事件	340
17.2.1	以编程方式启用跟踪	325	18.6	HttpModule	342
17.2.2	TraceFinished 事件	326	18.6.1	现有的模块	343
17.2.3	融合其他跟踪消息	327	18.6.2	模块的实现	344
17.3	使用 Visual Studio 进行调试	327	18.6.3	查看活动的模块	347
17.4	错误页面	329	18.6.4	在模块中存储状态	348
17.5	未处理的异常	332	18.7	Global.asax 与 HttpModule	351
17.6	快速参考	334	18.8	快速参考	352

第 18 章	HttpApplication 类与 HTTP 模块	335	第 19 章	HTTP 处理程序	353
18.1	Application 对象——全局访问点	335	19.1	ASP.NET 请求处理程序	353
18.2	HttpApplication 的重写	336	19.2	内建的处理程序	355
18.3	使用应用程序状态的注意事项	338	19.3	处理程序与 IHttpHandler	357
18.4	事件的处理	339	19.4	处理程序与会话状态	361
18.4.1	Application_Start	339	19.5	一般处理程序(ASHX 文件)	362
18.4.2	Application_End	339	19.6	快速参考	365
18.4.3	Application_Error	339			

第 V 部分 动态数据、XBAP、MVC、AJAX 和 Silverlight

第 20 章 动态数据	369	23.4.2 ScriptManagerProxy 控件	420
20.1 动态数据控件	369	23.4.3 UpdatePanel 控件	420
20.2 动态数据详解	373	23.4.4 UpdateProgress 控件	420
20.3 快速参考	377	23.4.5 Timer 控件	420
第 21 章 ASP.NET 与 WPF 内容	379	23.5 AJAX 客户端支持	420
21.1 通过降低往返次数来改进界面性能	379	23.5.1 ASP.NET AJAX Control Toolkit 简介	421
21.2 WPF 是什么	380	23.5.2 AJAX Control Toolkit 中的组件	422
21.2.1 WPF 与 Web 的关系	381	23.6 AJAX 入门	424
21.2.2 松散 XAML 文件	382	23.7 定时器	428
21.2.3 XBAP 应用程序	382	23.8 进度的更新	435
21.3 WPF 内容与 Web 应用程序	386	23.9 扩展程序控件	438
21.4 关于 Silverlight	391	23.9.1 AutoComplete 扩展程序	439
21.5 快速参考	392	23.9.2 一种类似模式对话框的组件	444
第 22 章 ASP.NET MVC 框架	393	23.10 快速参考	448
22.1 “模型-视图-控制器” (MVC) 架构	393	第 24 章 Silverlight 与 ASP.NET	451
22.2 ASP.NET 与 MVC	395	24.1 Web 应用程序的发展	452
22.3 ASP.NET MVC 与 Web 窗体	396	24.2 何为 Silverlight	453
22.4 MVC 与测试	397	24.3 创建 Silverlight 应用程序	454
22.5 MVC 与 ASP.NET 的结合	398	24.4 架构	458
22.6 快速参考	413	24.5 XAML	458
第 23 章 AJAX	415	24.5.1 可视树的构造	459
23.1 富 Internet 应用程序(RIA)	415	24.5.2 XAML 与命名空间	459
23.2 何为 AJAX	416	24.6 Silverlight 应用程序的编译	460
23.3 ASP.NET 与 AJAX	417	24.7 在网页中添加 Silverlight 内容	460
23.3.1 使用 AJAX 的原因	417	24.7.1 使用<object>标签	461
23.3.2 现实中的 AJAX	418	24.7.2 使用 JavaScript 函数	461
23.3.3 AJAX 展望	419	24.8 控件与事件	462
23.4 ASP.NET 对 AJAX 的服务器端支持	419	24.8.1 路由事件	462
23.4.1 ScriptManager 控件	419	24.8.2 Silverlight 控件与类成员	462
		24.9 Silverlight 的布局方式	463
		24.10 Silverlight 与 HTML 的结合	468
		24.11 动画	469

24.12	WCF 服务与 Silverlight.....	476	24.13	快速参考	483
-------	--------------------------	-----	-------	------------	-----

第 VI 部分 服务与部署

第 25 章	Windows Communication Foundation	487	25.7	WCF 客户端的构建	498
25.1	分布式计算的复兴	487	25.8	快速参考	503
25.2	种类繁多的通信 API.....	488	第 26 章	部署	505
25.3	针对连接型系统的 WCF.....	488	26.1	Visual Studio 网站	505
25.4	WCF 的组成元素.....	489	26.1.1	HTTP 网站	506
25.4.1	端点	489	26.1.2	FTP 网站	506
25.4.2	信道	490	26.1.3	文件系统网站	506
25.4.3	行为	490	26.2	预编译	507
25.4.4	消息	491	26.2.1	针对性能的预编译	507
25.5	WCF 与 ASP.NET	491	26.2.2	针对部署的预编译	507
25.5.1	并行模式	491	26.3	Visual Studio 2010 的部署支持	508
25.5.2	ASP.NET 兼容模式	492	26.4	快速参考	513
25.6	编写 WCF 服务.....	492			

还在为学习 .NET技术发愁吗？350元等于什么？

还在为学习 .NET技术发愁吗？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！！ 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！！

需要的请联系QQ：2569343569  在线咨询



第 I 部分

基础知识

- ▶ 第 1 章 Web 应用程序基础
- ▶ 第 2 章 ASP.NET 应用程序基础
- ▶ 第 3 章 页面的呈现模型
- ▶ 第 4 章 自定义控件
- ▶ 第 5 章 复合控件
- ▶ 第 6 章 常用控件介绍

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 1 章

Web 应用程序基础

学习目标

- 理解 HTTP 请求
- 在不使用浏览器的情况下通过 Microsoft .NET Framework 发起 HTTP 请求
- 理解 HTML
- 使用“Internet 信息服务”(IIS)
- 在不使用 Microsoft ASP.NET 的情况下生成动态 Web 内容


本章将介绍构建 Web 应用程序所需的一些基础知识。对于大多数桌面应用程序，其组成部分都在本地(以组件的形式存储在用户的硬盘驱动器上)，而 Web 应用程序需使用一种无连接的协议(disconnected protocol)从分布甚广的网络获取软件的各组成部分。虽然 ASP.NET 的底层技术已经存在许久了，但各自相对独立。ASP.NET 将这些技术有机地结合在一起，从而极大地简化了 Web 开发。


本章包含 3 个主题，将帮助您理解 ASP.NET 的使用：

- HTTP 请求的工作方式
- HTML 的特点
- HTTP 请求在 IIS(Microsoft 的生产型 Web 服务器)上的处理方式

相比过去，虽然 ASP.NET 简化了 Web 应用程序的开发，但对相关组件实际工作原理的正确认识仍很重要，这能够帮助我们理解 Web 应用程序开发的各个环节。例如，在跟踪某个异常的 HTTP 请求，或者尝试查找页面中字体在客户端浏览器中显示错误的原因时，如果理解 HTTP 和 HTML 是如何配合来将页面投递给客户端的，很多问题就变得迎刃而解了。此外，在为 Web 页面编写自定义控件时，由于自定义控件往往要求开发者手动编写控件呈现代码，并确保控件按正确的顺序生成 HTTP 标签，此时就需要掌握 HTML。

理解 ASP.NET 的这 3 个底层技术将为该系统的其他部分奠定基础。在学习 ASP.NET 时，您便可以理出各部分的头绪来。

 **注意** 安装本书的示例代码要求用户具有计算机的 Administrator 权限。如果使用家用计算机，则可能已具有 Administrator 权限。如果使用某个组织的计算机，并且没有 Administrator 权限，则需要咨询计算机支持人员或 IT 人员。相关内容，请参考本书“前言”部分“示例代码”一节。

 **注意** 执行配套资源上针对本章的示例代码要求 IIS 的支持。有关运行本章示例代码的注意事项，请阅读本书“前言”部分“示例代码”一节。

1.1 HTTP 请求

Web 浏览器和网站之间的通信机制称为“超文本传输协议”(Hypertext Transfer Protocol, HTTP)。如今我们所熟知的“万维网”(World Wide Web)起源于“欧洲核子研究中心”(CERN)瑞士分部的一个研究项目。在当时，超文本(彼此之间可以任意链接的文档)的符号表示曾一度持续受到关注。苹果计算机公司的 Hypercard 产品将超文本应用程序推广开来。如果文档能够在网络间链接起来，这将为出版业带来一场革命。这就是 HTTP 被开发出来的原因，它处于 TCP/IP 协议的最顶层——应用层(application layer)。

HTTP 起初旨在传输超文本文档。也就是说，它最初只是将文档链接在一起，而没有考虑现代网站的重要功能(如基于 Web 的用户界面)。HTTP 协议的早期版本只支持用于获取指定资源的 GET 请求。在收到请求后，服务器会以文本流的形式发送文件。在响应到达客户端的浏览器后，连接便会被断开。HTTP 协议的几个早期版本只支持文本流的传输，而不支持其他数据类型。

最早的 HTTP 正式规范是 1.0 版，发布于 20 世纪 90 年代中期。HTTP 1.0 增加了更多对复杂通信的支持，已不再是传输简单文本的协议了。随着发展，HTTP 开始支持不同类型的媒体(由 Multipurpose Internet Mail Extensions(MIME)标准进行规约)。HTTP 协议目前的版本是 1.1。

HTTP 是连接型协议，围绕几个基本命令构建。在开发 ASP.NET 应用程序时，最重要的是 GET 和 POST 命令。HEAD 和 PUT 在 ASP.NET 中不常用，但也是重要的 HTTP 命令^①。

GET 命令用于通过请求获取信息，具体信息由“统一资源标识符”(Uniform Resource Identifier, URI)指定。HEAD 命令用于通过请求获取标头(header)信息(即不返回消息主体)，具体信息也由 URI 指定。可以使用 POST 命令向服务器发送可能导致连带作用的请求，如发送需要服务器处理的信息。PUT 命令也用于向服务器发送信息，但内容面向的是文档和记录，而 POST 一般用于发送与 HTML 页面请求有关的参数。在最初请求页面时，一般要先使用 GET 命令，随后通过 POST 命令来实现与服务器后续的一系列交互。

1.1.1 从浏览器发出的 HTTP 请求

举例来说，假设浏览器要向服务器发送请求来获取位于本地主机虚拟目录 asp.net4sbs 中的 helloworld.htm 资源。(稍后会介绍虚拟目录的概念，而现在可以将虚拟目录看作 Web 应用

^① 译者注：这些命令有时也被称作“方法”(method)。

程序中一个可以公开访问的地址。)下面是一个(虚构的)HTTP 服务器请求示例：

```
GET / asp.net4sbs /helloworld.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, ... , */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; ... .NET CLR 3.0.04506.30)
Host: localhost:80
Connection: Keep-Alive
```

如果希望查看交互过程中的实际数据，可以使用 TCP 监视程序。这类软件有很多，TcpTrace 是一个不错的选择，可以到 <http://www.pocketsoap.com/tcptrace/> 下载。该网站同时也提供了使用说明。

为向 Web 服务器发送请求，浏览器需要通过 URI 和其他信息(如标头信息和被请求文件的文件名)创建 HTTP 请求。请求中的标头信息包括浏览器所处操作系统环境的详细信息，以及服务器可能要使用的其他信息。浏览器会向 HTTP 标头中指定的服务器发送请求。当服务器收到请求后，它会以文本流的形式返回被请求的资源。浏览器对其进行解析并呈现内容部分。下面的代码是请求 HelloWorld.htm 文件后服务器提供的响应。通常，在通过浏览器查看资源时，它不会显示所有标头信息，但可以通过 TcpTrace 这样的 TCP 监视程序进行检查。此外，我们还可以通过后面介绍的 ASP.NET 跟踪功能来获得标头信息。

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET
Date: Thu, 01 Nov 2007 23:44:04 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Mon, 22 Oct 2007 21:54:20 GMT
ETag: "04e9ace185fc51:bb6"
Content-Length: 130
<html>
  <body>
    <h1> Hello World </h1>
    Nothing really showing here yet, except some HTML...
  </body>
</html>
```

第一行提示了协议的类型(HTTP 协议)、版本(1.1 版)以及返回代码(200，表示“正常”)。响应中标头的其他部分(<html>之前)提示了请求的时间、文件上次修改的时间、内容的类型等信息。在追查页面缓存问题和检测浏览器功能时，可能会用到这些信息。紧跟标头的是服务器以文本形式发送的 HTML 文件。

1.1.2 在不使用浏览器的情况下生成请求

.NET 开发环境除了作为构建 Web 应用程序的框架外，其中还包含能够以编码方式生成 HTTP 请求的类。WebRequest 类包含一个名为 GetResponse 的成员，可以向由“统一资源

定位符” (Uniform Resource Locator, URL) 指定的地址发送请求。为了理解如何在不使用浏览器的情况下直接请求 Web 服务器，请编译和运行下面这段程序，它会获取 Microsoft.com 主页的内容。

➤ 构建一个简单的 HTTP 请求程序

- 1. 打开 Visual Studio .NET，在主菜单中依次选择“文件”|“新建”|“项目”命令。在“新建项目”对话框中，单击“控制台应用程序”命令，将其命名为 WebRequestorApp，如下图所示。



创建项目后，Visual Studio 会生成一个空的控制台应用程序。

- 2. 在程序中添加用于生成 Web 请求的必要代码。Visual Studio 会将该控制台应用程序的入口点放在名为 Program.cs 的文件中。(该文件包含的代码默认会显示在代码窗口中。) 下面是用于生成 Web 请求的代码，需要输入的部分加粗显示：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;
namespace WebRequestorApp
{
    class Program
    {
        static void Main(string[] args)
        {
            WebRequest req =
                WebRequest.Create
                    ("http://www.microsoft.com");
            WebResponse resp = req.GetResponse();
        }
    }
}
```

```
        StreamReader reader =  
            new StreamReader(resp.GetResponseStream(),  
                             Encoding.ASCII);  
        Console.WriteLine(reader.ReadToEnd());  
    }  
}
```

3. 为运行这个应用程序，在主菜单上单击“调试”|“开始执行(不调试)”。Visual Studio 会打开一个控制台，并运行该程序。在一段时间过后，我们可以在屏幕上看到一些 HTML 文本。

当然，这些 HTML 并不是供人们阅读的，而需要浏览器对其进行解析。不过，本示例的确演示了建立 Web 请求的基本方法(并且可以看到响应中返回的内容)。

在这种情况下，发送至服务器的请求要比一个 POST 请求小得多。WebRequest.GetResponse 不会像 POST 请求那样包含较多信息，它只生成一条 GET，后面跟有 URI、主机信息和连接类型：

```
GET /aspnet2sbs/helloworld.htm HTTP/1.1  
Host: localhost:80  
Connection: Keep-Alive
```

大多数浏览器的基本职责是：(1)对请求进行打包，并将这个包发送至通过 URI 指定的服务器；(2)从服务器获得响应，并将其呈现为供人们阅读的表单(form)^①。响应通常以文本流的形式返回，通过 HTML 标签进行标记。下一节将讨论 HTML。

1.2 超文本标记语言

在学习 ASP.NET 的过程中，我们会看到许多 HTML 文本。其中的大多数是通过 ASP.NET 服务器端控件生成的，而有的是为创建所期望的简单页面而自己动手编写的。不论怎样，理解 HTML 都很重要，因为有时需要从头编写服务器端控件，而有时需要优化或调试 ASP.NET 应用程序的输出。

大多数 HTTP 请求的结果是，发送请求的程序会收到文本流。而使用 HTML 语言来格式化文档现已成为普遍共识，所有浏览器都能够解释 HTML。

真正可用的第一版 HTML 是 2.0。3.2 版引入了一些新功能，包括表格、文本流、小程序、上角标和下角标，并向后兼容 HTML 2.0 标准。

从根本上讲，Web 用户界面开发技术是以权威的浏览器和良构的 HTML 表单为基础产生的。由于 HTML 要能够被运行在不同平台的浏览器解释，所以需要实现一种能够在全局范围内

^① 译者注：form 这个词在传统的 Web 开发中称为“表单”，但 ASP.NET 的某些思想来源于桌面应用程序开发，因此有时会看到“窗体”的说法。本书约定，在 HTML 中称其为“表单”，在谈论 ASP.NET 控件时称其为“窗体”(如 Web 窗体控件)。

进行交互的计算平台。为此，除了需要有一个成熟的 HTML 版本外，还要求服务器能够在运行时满足特定用户的请求。

例如，下面的 HTML 流会被呈现为一个网页，其中包含一个按钮和一个已填充的选项列表。(这段代码位于本章配套示例代码的 SelectNoForm.htm 文件中。)

```
<html>
<body>
  <h2>Hello there. What's your favorite .NET feature?</h2>
  <select name='Feature'>
    <option>Type-Safety</option>
    <option>Garbage collection</option>
    <option>Multiple syntaxes</option>
    <option>Code Access Security</option>
    <option>Simpler threading</option>
    <option>Versioning purgatory</option>
  </select>
  <br/>
  <input type=submit name='Lookup' value='Lookup'></input>
  <br/>
</body>
</html>
```


 **提示** 该 HTML 页面所呈现的内容是在后续章节中所要编写的。现在深入细节还为时尚早，但可以确定的是 HTML 就是以这种形式呈现的。

图 1.1 展示了上述代码在浏览器中显示的效果。



图 1.1 一个简单的 HTML 页面，包含一个选择标签(其样式类似 Windows 中的选项列表)和一个提交按钮

这是一个静态页面。虽然页面中包含选项列表和按钮，但这没有任何实际作用。我们可以单击打开这个选项列表，并在浏览器中使用它。还可以单击那个按钮，但所有操作都只能在本地进行。如果要获得其他特性，则要求另一端的服务器支持动态内容。

1.3 动态内容

最早期的网站主要使用静态HTML页面构建。也就是说,用户只能浏览网站上有限的HTML文档。虽然这在当时已经令人惊叹不已,但随着时代的变革,如今的HTML已远远不只具有格式化文本的功能了。

例如,浏览器会将HTML中的<select>和</select>标记解释为选项列表(selection list)控件,该控件在ASP.NET中被称为“下拉列表”(drop-down list)。第一个标签(<select>)称作“开标签”(opening tag),第二个标签(</select>)称作“闭标签”(closing tag)。标签可以包含子标签,如在之前的代码中,<option></option>标签用于提供下拉列表每个选项的内容。标签还可以包含“特性”(attribute),用于修改或定制标签的行为。对<input></input>标签应用不同属性后,浏览器会将其显示为不同的控件(如文本框或按钮)。HTML提供了一个特殊的标签——<form>(表单),它能够包容其他标签,将信息返回给服务器进行处理。

1.3.1 HTML 表单

HTML中的<form>开标签和</form>闭标签用于提示浏览器:用户会将HTML文档中两者之间的内容返回给服务器。我们可以使用<form>标签来指定Web文档如何处理最终用户的输入(它不仅作为输出)。表单的内容(包含在输入控件中的数据)会被“投递回”服务器进行处理。这个操作通常被称为“回发”(postback)。这就是HTTP通常先进行GET(获取),再进行POST(投递,或者说是GET的另一种形式)的原因——前者用于获取最初的文档,后者用于在必要时将数据返回给服务器。

<form>标签通常包含一组特定的用户输入控件。下面的标记代码会被呈现为一个与之前示例功能相同的选项页面,但这里添加了<form>标签(这段代码位于本章配套示例代码的SelectFeature2.htm文件中):

```
<html>
<body>
  <form action="http://localhost/HttpHandlers/selectfeature2.htm"
    method="get">
    <h2>Hello there. What's your favorite .NET feature?</h2>
    <select name='Feature'>
      <option>Type-Safety</option>
      <option>Garbage collection</option>
      <option>Multiple syntaxes</option>
      <option>Code Access Security</option>
      <option>Simpler threading</option>
      <option>Versioning purgatory</option>
    </select>
    <br/>
    <input type=submit name='Lookup' value='Lookup'></input>
```



```
<br/>
</form>
</body>
</html>
```

为使这段代码正常工作，可以将其保存为 `SelectFeature2.htm`，并置于 `c:\inetpub\wwwroot` 目录下。在浏览器的地址栏中输入 `http://localhost/selectfeature2.htm`。

我们可以通过 `<form>` 标签的几个属性来控制页面的行为。不难发现，在上述示例中，`<form>` 标签多了一个 `action` 属性，它用于指定接收该表单内容的服务器。如果没有 `action` 属性，则默认会使用当前文档的 URL。

上述 HTML 中 `<form>` 的另一个属性是 `method`。`method` 属性用于指定提交表单时执行的 HTTP 方法，进而控制表单数据提交到服务器的方式。本示例采用的是 GET 方法，因为要向服务器请求一个新页面。如果选择最后一个选项——Versioning Purgatory，然后单击 Lookup 按钮，那么表单的 GET 方法会使表单中输入控件的内容被追加到 URL 中，如

`http://localhost/SelectFeature2.htm?Feature=Versioning+purgatory&Lookup=Lookup`。

这个修改过的 URL(通常被称为“查询字符串”)会被发送至服务器。

表单的 POST 方法会使表单的内容包含在 HTTP 包的主体中发送至服务器，如下所示：

```
POST /SelectFeature2.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, ... , */*
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; ... .NET CLR 3.0.04506.30)
Host: localhost:80
Content-Length: 42
Connection: Keep-Alive
Cache-Control: no-cache

Feature=Versioning+purgatory&Lookup=Lookup
```

向文档的主体中添加 `<form>` 标签，是使 HTTP 应用程序能够与用户进行交互的前提之一。此时，只需要对服务器端稍事修改就可以获得动态内容。在单击 Lookup 按钮后，浏览器实际会强制进行一次与服务器的往返交互(在本示例中，由于在表单的 `method` 属性中指定了 GET 操作，因此只会执行一次获取文档的 HTTP GET 命令)。

至此，一个常规的 HTTP GET 命令只返回指定的文档。在实际的交互环境中，服务器端往往要更改对浏览器与服务器之间传输的内容。

举例来说，假设某个用户为获得某个资源而发起 GET 请求。该用户从选项列表中选择一项，然后单击 Lookup 按钮。在可交互的应用程序中，为对用户的输入进行处理，浏览器必须通过另一次请求与服务器进行第二次往返交互。服务器要对来自浏览器的请求进行分析，以确定如何处理。服务器此时才开始进行更为复杂的计算。对于不同平台，服务器处理


回发的方式也不同, 这类服务器程序包括“公共网关接口”和 IIS。

1.3.2 公共网关接口——非常陈旧的技术

最早支持动态 Web 内容的 Web 服务器是通过“公共网关接口”(Common Gateway Interface, CGI)实现的。CGI 是构建 Web 服务器的早期标准。CGI 程序以实时方式运行, 根据应用程序的状态和传入的请求更改其输出。对于运行 CGI 的 Web 服务器, 为进行响应, 每个传入的请求都会通过单独的进程进行处理。这种应用程序可以执行任何操作, 包括在数据库中查询数据、获取信用卡号以及发送具有某种格式的信息。

1.3.3 Microsoft 环境下的 Web 服务器

如果使用 Microsoft 的操作系统来处理 Web 内容, 那么为每个请求创建新进程(就像 CGI 那样)的代价是十分高昂的。Microsoft 的解决方案是通过单个守护进程(daemon process, 这种进程在 Windows 操作系统中被称为“服务”)来监视 80 端口传入的网络数据包, 并在需要修改内容时加载处理每个请求所需的动态链接库(dynamic-link library, DLL)。Microsoft 的标准 Web 平台是以“Internet 信息服务”(IIS)为基础构建的。

 **提示** 如果在 Microsoft Visual Studio 2010 中创建和编辑 Web 应用程序, 则可以从文件系统或 IIS 加载相关文件(也有其他几种方式)。如果通过 IIS 加载 Web 应用程序, IIS 便很自然地充当 Web 服务器。如果从文件系统加载 Web 应用程序, 那么由哪个程序为 HTML(或 ASP.NET)文档服务呢? 我们可以使用一种特殊的 Web 开发服务器来简化调试与管理, 该功能起始于 Visual Studio 2005。这种 Web 开发服务器是基于一种名为 Cassini 的 Web 服务器构建的, 可以为 HTML 和 ASP.NET 页面服务, 其效果与 IIS 相同, 出于开发目的可以使用。然而, 请记住, IIS 是微软专业级别的服务器, 具有良好的健壮性和安全性, 是承载 ASP.NET Web 应用程序的首选。虽然 Web 开发服务器与 IIS 十分类似, 但并不完全一致。如果在两种服务器之间迁移 Web 应用程序, 差异便可能显现出来。这些不同往往与安全性和权限有关。

1.3.4 Internet 信息服务

从本质上讲, 所有 Web 应用程序环境的工作方式都相同。不论使用何种软硬件平台, 服务器上都要监视 80 端口(通常)传入的 HTTP 请求。当某个请求到达时, 服务器要以某种有意义的方式来响应请求。在 Microsoft 操作系统中, IIS 是最常见的监视程序, 它能够处理 80 端口(HTTP 请求的常规入站端口)。Internet 服务器也可能使用其他端口。例如, HTTPS(Secure HTTP)会使用 443 端口。然而, 我们目前最关心的是 80 端口上的常规 Internet 流量。

当浏览器调用 Microsoft 平台的服务程序时，IIS 会侦听到该请求，并通过 URL 查找资源。IIS 能够将其目录空间分割成可管理的单元，这些单元被称为“虚拟目录”(virtual directory)。举例来说，假设某人试图通过下面的 URL 来访问服务器上的资源：

`http://www.northwind.com/products/showfeatures.htm`

在这里，northwind 的域名是虚构的，仅用于演示。然而，如果真有某个已注册的服务器使用了该域名，那么这个 URL 便能够定位其指定的资源。在这条 URL 中，`http://www.northwind.com` 用于指定服务器，请求会通过一系列路由器被定向。一旦请求到达该服务器，服务器便会在名为 products 的某种目录下查找 showfeatures.htm。如果服务器运行的是 IIS，那么 products 就可能是一个虚拟目录。

IIS 将其工作空间划分为多个虚拟目录。每个虚拟目录一般针对单个应用程序，用于将服务器硬盘上的物理目录映射到 Internet URL 上。为每个应用程序建立一个虚拟目录可以使 IIS 为多个应用程序服务。不同虚拟目录可以包含不同的配置属性，其中包括安全选项、错误处理重定向，以及应用程序隔离等选项。配置参数还包括文件扩展名与 IIS 扩展 DLL(可选)，这些 DLL 通常被称为 ISAPI DLL(ISAPI 代表“Internet 服务器应用程序编程接口”，Internet Server Application Programming Interface^①)。(在 IIS 7.0 之前的版本中，ASP.NET 本身也由其中的一个 ISAPI DLL 处理。)随着发展，ASP.NET 已变成 IIS 中的“一等公民”。

虽然了解 IIS 的工作原理对编写 ASP.NET 应用程序并非至关重要，但在调试、测试和最终部署 Web 应用程序时就显得非常必要了。Visual Studio 内建的 Web 服务器(Cassini)可以满足大多数任务的需要，但缺乏许多 IIS 中的支持。真正的 ASP.NET 开发者懂得这一点，并非常善于管理 IIS。如果希望立即开始着手编写应用程序，则可以略过下面有关 IIS 的一节，但 IIS 的操作和管理将会贯穿全书。下面，让我们简单了解一下 ISAPI，以及如何通过它来扩展 IIS。

1.3.5 Internet 服务器应用程序编程接口 DLL

在 Microsoft 的操作系统中，创建进程空间在系统资源和时间周期上的一系列开销较高。如果所编写的服务器要为处理每个请求而单独创建一个进程，那么这个糟糕的服务器将很快崩溃，其上的电子商务网站也将随之停止运营。

Microsoft 环境会通过 DLL 响应请求。加载 DLL 的成本较低，而且 DLL 中代码的运行速度也较快。由于某些历史原因，处理 Web 请求的 DLL 被称为 ISAPI。在 IIS 7 之前，IIS 会侦听请求，然后将其交给特定的 ISAPI DLL 来处理该请求。ASP(ASP.NET 的前身)由一个名为 ASP.DLL 的 ISAPI DLL 进行实际的处理。事实上，ASP.NET 的早期版本依赖于名为 ASPNET_ISAPI.DLL 的 ISAPI DLL，但如今的 ASP.NET 管线(pipeline)已集成进了 IIS。

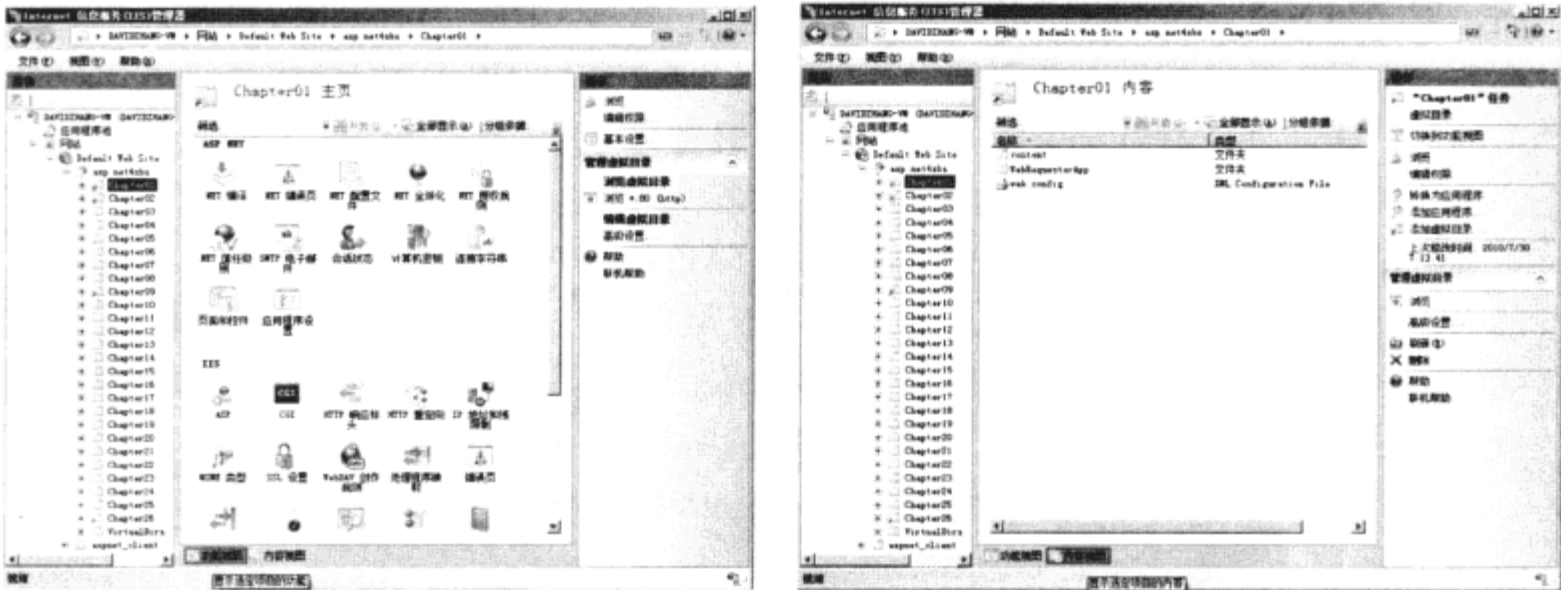
^① 译者注：根据 MSDN 上的资料，ISAPI 是 Internet Server Application Programming Interface 的缩写，而不是 Internet Services Application Programming Interface。

1.3.6 “Internet 信息服务” 的运行

IIS 可以在“控制面板”中打开。通过下面的练习，您将对 IIS 的管理有所了解。了解 IIS 至关重要，因为 ASP.NET 依赖它来处理实际 Web 应用程序的请求。IIS 7.0 和之前的版本都是将服务器的应用程序空间划分为不同虚拟目录。IIS 6.0 和 IIS 7.0 还包含许多其他的功能，如应用程序隔离、为控制流失的请求而进行回收，以及在请求处理发生异常的情况下限制内存的使用。

➤ 运行 IIS

- 1. 为打开 IIS，首先要找到“管理工具”。在 Windows 7 操作系统中，在“控制面板”中，依次单击“系统和安全”|“管理工具”选项，在打开的窗口中双击“Internet 信息服务 (IIS) 管理器”选项。IIS 的用户界面会随即显示在屏幕上。下面的两张截图展示了“功能视图” (左图) 和“内容视图” (右图) 在 Windows 7 下的效果。



界面左侧是一个可展开的树状视图，显示了计算机上通过 IIS 管理的所有网站和虚拟目录。IIS 5.x 和 IIS 6.0 在左侧显示虚拟目录，在右侧显示目录内容。IIS 7.0 的管理控制台 (如图 所示) 包含两个视图——“功能视图”和“内容视图”。“功能视图”的图标用于管理左侧选定项的不同 IIS 功能。“内容视图”能够显示选定项中包含的文件。

- 2. 查看特定虚拟目录的配置。在“功能视图”中，可以查看特定虚拟目录的配置情况，双击相应图标即可。例如，如果要查看在未指定文件名的情况下所选择的默认文件，则可以双击“默认文档”图标。下图展示了 IIS 会尝试加载的默认文件的列表。

IIS 中的许多功能都可以这样配置，它们都以图标的形式显示在“功能视图”中。这个功能集合涉及的内容非常多，涵盖了对外公开虚拟目录的所有选项。我们不必花过多的时间在这儿，因为 ASP.NET 会负责大部分事宜 (并非都交给 IIS 处理)。

ASP.NET 4 从入门到精通




3. 查看虚拟目录的模块映射。像.htm 文件这样的静态文件通常会直接返回给客户端。然而，对于内容在两次投递之间发生变化的动态页面，则需要做进一步处理，因而这些页面会被分配给特定的处理程序。正如稍后要介绍的，IIS 7.0 会通过托管代码来处理大部分请求，而这种代码会在“.NET 公共语言运行库”(Common Language Runtime, CLR)的环境下运行。对于那些希望编写本地代码的开发者，IIS 7.0 还包含一套 C++/本地(native)^①核心服务器应用程序编程接口(API)。该 API 通过 IsapiModule 来暴露传统的 ISAPI 扩展 DLL，以这种方式来与 IIS 7.0 协作。有个叫做 IsapiFilterModule 的模块，取代了之前 IIS 中的 ISAPI 筛选 API。为查看 IIS 7.0 的模块映射，在“功能视图”中双击“模块”图标。此时会看到 IIS 7.0 中侦听请求的各模块(参见下图)。



① 译者注：native 在不同资料中的翻译不同，大多数资料会将其翻译为“本地”。如果在 IIS 中看到“本机”，可以认为这两者的意思是相同的。

名称	路径	状态	操作
已启用			
ASPClassic	* .asp	已启用	[X] [Y]
aspq-Integrated-4_0	* .aspq	已启用	[X] [Y]
aspq-ISAPI-4_0_32bit	* .aspq	已启用	[X] [Y]
aspq-ISAPI-4_0_64bit	* .aspq	已启用	[X] [Y]
AssemblyResourceLoader-In...	VehResource.axd	已启用	[X] [Y]
AssemblyResourceLoader-In...	VehResource.axd	已启用	[X] [Y]
AED-ISAPI-2_0	* .axd	已启用	[X] [Y]
AED-ISAPI-4_0_32bit	* .axd	已启用	[X] [Y]
AED-ISAPI-4_0_64bit	* .axd	已启用	[X] [Y]
CGI-exe	* .exe	已启用	[X] [Y]
cshite-Integrated-4_0	* .cshite	已启用	[X] [Y]
cshite-ISAPI-4_0_32bit	* .cshite	已启用	[X] [Y]
cshite-ISAPI-4_0_64bit	* .cshite	已启用	[X] [Y]
cshitel-Integrated-4_0	* .cshitel	已启用	[X] [Y]
cshitel-ISAPI-4_0_32bit	* .cshitel	已启用	[X] [Y]
cshitel-ISAPI-4_0_64bit	* .cshitel	已启用	[X] [Y]
ExtensionlessUrl-Integrat	*	已启用	[X] [Y]
ExtensionlessUrl-ISAPI-4	*	已启用	[X] [Y]
ExtensionlessUrl-ISAPI-4	*	已启用	[X] [Y]
HttpPlatformHandlerFactor	* .ran	已启用	[X] [Y]
HttpPlatformHandlerFactor	* .ran	已启用	[X] [Y]
HttpPlatformHandlerFactor	* .ran	已启用	[X] [Y]
HttpPlatformHandlerFactor	* .ran	已启用	[X] [Y]
4			[X] [Y]

 **提示** 如果出于某种原因需要运行传统的 ASP，则应注意，IIS 7.0 默认不会安装 ASPClassic，你必须有意识地添加该功能。

- (1) 在“控制面板”中，单击“程序”|“程序和功能”选项。
- (2) 选择“打开或关闭 Windows 功能”命令。
- (3) 在打开的对话框中展开“Internet 信息服务”节点。
- (4) 依次展开“万维网服务”节点和“应用程序开发功能”节点。
- (5) 选中 ASP 复选框便可安装 ASP 的处理程序，如下图所示。



1.4 传统的 ASP：ASP.NET 的前身

虽然本书旨在介绍 ASP.NET，但理解传统的 ASP 也会对主题有所帮助。比较 ASP 和 ASP.NET，有助于理解 ASP.NET 中各机制形成的缘由及 ASP.NET 存在的价值。

Microsoft 最初开发的“活动服务器页面”(Active Server Pages, ASP)旨在吸引除 C++以外的庞大开发者队伍从事 Web 开发。在 IIS 出现之后，相比其他操作系统，它很自然地成为 Microsoft 操作系统中开发网站的环境。事实上，如今也能看到某些网站以纯粹的 ISAPI DLL 网站的模式进行开发(eBay 就是其中之一)，只要查看往返于浏览器与服务器之间的查询字符串便可以找到蛛丝马迹。例如，在查询字符串中有时会嵌有 ACMEISAPI.DLL 这样的文件名。

然而，整个网站都使用 ISAPI DLL 开发会令人望而却步。使用 C 或 C++编写 ISAPI DLL 可以完全控制网站的执行并可使其正常工作。然而，获得那些自由度是要付出同等的代价的，因为使用 C 或 C++进行软件开发并不轻松。

正因如此，Microsoft 开发了 ASP，提供了单个名为 ASP.DLL 的 ISAPI DLL。ASP Web 开发者将其编写的代码保存在扩展名为 .asp 的文件中(如 somefile.asp)。这些文件是静态 HTML 和可执行代码(通常使用脚本语言编写)的混合，这些可执行代码会在运行时编译。例如，清单 1.1 给出了一段 ASP 程序，它会呈现一个 HelloWorld 页面，其中包含静态 HTML 和运行时生成的文本。(这个示例对应的文件位于本书配套的示例代码中，文件名为 HelloWorld.asp。)

清单 1.1 一个传统的 ASP 文件

```
<%@language="javascript"%>
<html>
  <body>
    <form>
      <h3>Hello world!!! This is an ASP page.</h3>

      <%Response.Write("This content was generated ");%>
      <%Response.Write("as part of an execution block");%>
    </form>
  </body>
</html>
```

清单 1.1 中的代码会呈现下图所示页面。IIS 会监视 80 端口的请求。当针对 HelloWorld.asp 文件的请求到达时，IIS 会识别出 .asp 扩展名，并令 ASP.DLL 来处理该请求(这就是设置文件映射的原因)。ASP.DLL 会直接呈现“Hello world!!! This is an ASP page.”这样的静态 HTML 字符串。当 ASP.DLL 遇到这种看似怪异的<%和%>执行标签时，它会通过 JavaScript 解析器来运行它们(请注意代码清单第一行的 language 属性)。图 1.2 展示了该页面在 Windows Internet Explorer 中的显示效果。

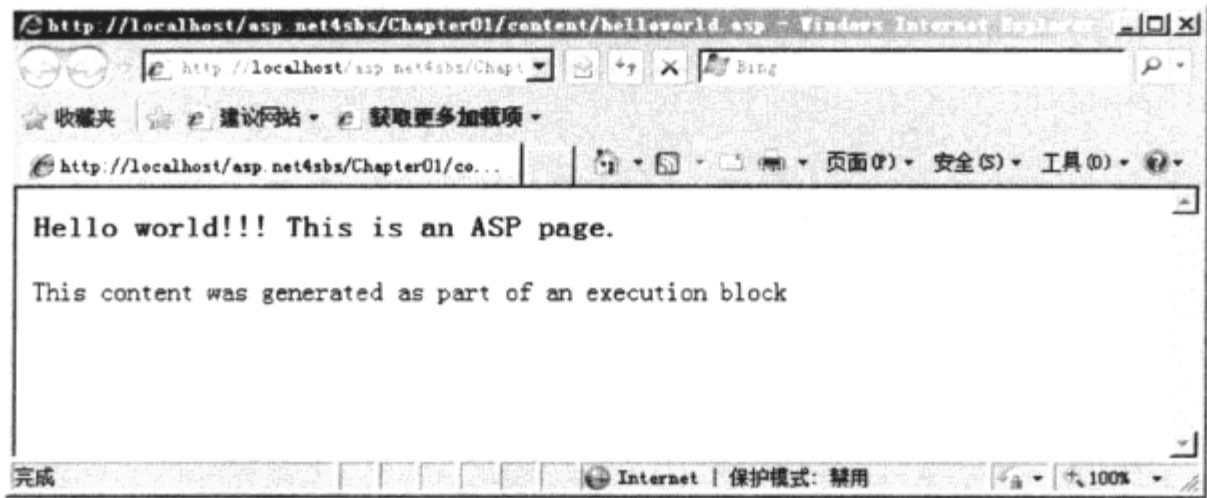


图 1.2 请求清单 1.1 的中 ASP 程序得到的结果

本书是关于 ASP.NET 软件开发的，因此会将重点放在这部分。然而，在结束传统 ASP 的话题之前，不妨再看一个示例。下面的代码将 SelectFeature.htm 以传统 ASP 页面的形式进行了重写。这个简单的 ASP 应用程序蕴含了 Web 开发中的一些核心问题，揭示了 Microsoft 重写其 Web 服务器技术，从而开发出 ASP.NET 的原因。(这段代码位于本章配套示例代码的 SelectFeature.asp 文件中。)

清单 1.2 以传统 ASP 页面形式重新编写的 SelectFeature.htm 页面

```
<%@language="javascript"%>
<html>
  <body>
    <form>
      <h2>HelloWorld</h2>

      <h3>What's your favorite .NET feature?</h3>
      <select name='Feature'>
        <option>Type-Safety</option>
        <option>Garbage collection</option>
        <option>Multiple syntaxes</option>
        <option>Code Access Security</option>
        <option>Simpler threading</option>
        <option>Versioning purgatory</option>
      </select>
      <br>
      <input type=submit name="Submit" value="Submit"></input>
    <p>
      Hi, you selected <%=Request("Feature") %>
    </p>
  </form>
</body>
</html>
```

SelectFeature.asp 的文本与 SelectFeature.htm 中的十分类似，不是吗？两者的差别主要在第一行(它现在指定了可执行部分的语法)和由<%和%>标记的可执行部分。静态 HTML 的剩余部分在表单中呈现为选项控件。

请注意可执行部分的代码以及它是如何通过 Response 对象(受 ASP 基础架构管理)将文本输

出至浏览器的。这段可执行代码会检查 Feature 控件(通过<select>标记定义)的内容，并将用户选定的值输出。

图 1.3 展示了 SelectFeature.asp 在 Internet Explorer 中呈现的效果。



图 1.3 清单 1.2 的代码在 Internet Explorer 中呈现的效果

图 1.3 给出的界面可能显得有些古怪，因为下拉列表显示的是 Type-Safety，而呈现的文本却是 Simpler threading。下拉列表框总会将第一个元素呈现为选定的元素，而没有额外的行为。在后面讨论服务器端控件时，我们会看到 ASP.NET 是如何解决这个问题的。下面一节介绍一些 Web 开发思想，将为您日后的学习奠定基础。

1.5 Web 开发思想

归根结底，开发 Web 应用程序无非是要处理两大问题：(1)在无连接型协议上通过 HTML 来管理用户界面(UI)；(2)管理应用程序的状态。这两种基本任务使得 Web 开发区别于其他类型应用程序的开发。

这种编程模型的形成在许多方面要追溯到 20 世纪 70 年代中期。当时大型机直接为与之连接的终端提供输出。用户向大型机提交任务，并从终端获取输出。那么如今有何变化呢？首先，终端计算机变得强大许多——它如今是功能强大的计算机，能够运行解释 HTML 的浏览器。其次，浏览器连接的另一端变成了 Web 服务器(而且有可能是服务器场)。最后，客户端和服务端使用的连接协议在物理上不直接关联(在用户得到结果之前，请求可以准确无误地在网络间穿梭)。

在 Web 应用程序开发中，程序的主要工作是接收来自远程客户端的请求，并为请求者提供有意义的响应。这一般意味着生成复杂的 HTML，以便结果能够呈现在浏览器中，供人们阅读。这其中的逻辑在现代商业网站这类应用中往往十分复杂。客户可能要通过网站查询当前报价、库存水平，甚至预订货品或服务。此时，为客户端生成有意义的 HTML 突然涉及另外一些任务，如访问数据库，对客户端进行身份验证，或者对客户的产品订单进行跟踪。如果这些功能都要从头开发，那会怎样？

虽然像传统 ASP 这样的框架一直致力于使 Web 开发更为简易，但开发者仍要自行开发许多功能(大部分与本节开始提到的两大问题有关)。例如，如果使用传统的 ASP 来构建既安全又易于管理的网站，则通常需要自行编写(或购买)安全子系统。管理网站的 UI 状态也是一项枯燥乏味的工作。

1.6 ASP.NET

所有这一切问题促使了 ASP.NET 的产生。在阅读本书后您会发现，ASP.NET 实现了许多原来要由开发者(重复)实现的功能，并将这些功能集成进了 ASP.NET 框架。

ASP.NET 自最早的版本开始就不断发展。ASP.NET 1.0 引入了一条定义明确的管线、一个切实有效的可扩展模型、一个服务器端控件呈现模型，以及诸多简化网站开发的功能。ASP.NET 2.0 把 ASP.NET 带到了另一层次，在框架中添加了更多需要经常实现的功能。例如，ASP.NET 2.0 在身份验证与授权服务方面对 ASP.NET 之前的版本做了极大的改进。ASP.NET 1.x 包含一个合理且易于管理的身份验证模型。然而，开发者仍然要创建自己的身份验证系统，并需要自行将其集成到所开发的网站中。ASP.NET 2.0 添加了一套身份验证子系统，有效地解决了这一问题。(第 9 章深入介绍了 ASP.NET 中的“Forms 身份验证”和其他安全特性)

ASP.NET 2.0 在市场中的应用已两年有余。但即便 2.0 版提供了如此多的升级，也仍然还有诸多改进空间。ASP.NET 3.5 也引入了两大新功能。其一是支持异步 Java 和 XML 风格的编程(这项技术一般被称为 AJAX)。其二是支持通过 IIS/ASP.NET 来承载 Windows Communication Foundation(WCF)应用程序。在如今的第 4 版中，AJAX 的支持得到了改进，支持动态数据，提供了一种“模型-视图-控制器”(Model-View-Controller, MVC)模式的实现，并支持 Microsoft Silverlight。

本书后面的章节将涵盖 ASP.NET 中最重要的功能。在读完本书之后，您将具备充足的知识来开发基于 ASP.NET 的网站。

1.7 快速参考

目 的	操 作
打开“Internet 信息服务(IIS)管理器”	打开“控制面板”，找到“管理工具”，选择“Internet 信息服务(IIS)管理器”
新建虚拟目录	打开“Internet 信息服务(IIS)管理器”，依次展开“网站” Default Web Site。在 Default Web Site 节点上右击，选择“添加虚拟目录”。然后根据向导的提示进行操作
在 IIS 中浏览某个资源	在所要浏览的资源上右击，然后选择“浏览”图标按钮
查看特定 IIS 虚拟目录中支持的文件类型	选择该虚拟目录，在“功能视图”中浏览“处理程序映射”和“模块”页面

1.6 ASP.NET

附錄四 臺灣省與世界各國之貿易關係
 附錄五 臺灣省與世界各國之貿易關係

参 考 文 献 5.1

<p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p>	<p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p>
<p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p>	<p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p>
<p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p>	<p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p> <p>“远端 Web 站”：在 Default Web Site 站点上，选择“站”。</p>

第 2 章

ASP.NET 应用程序基础


学习目标

- 创建“Internet 信息服务”虚拟目录
- 将 HTML 页面转换为 ASP.NET 程序
- 将 HTML 与可执行代码相结合，并使用服务器端脚本块
- 找到并查看 ASP.NET 根据.aspx 文件编译的程序集
- 使用代码隐藏和代码旁置执行模型
- 使用 Microsoft Visual Studio 2010 创建 Web 项目

本章将介绍构建 ASP.NET 应用程序的基础知识。从语法角度看，编写.NET 代码与编写“达康时代”^①(dot-com era)晚期的传统 ASP 代码非常类似。许多符号是相同的，而且有些语法甚至沿用至今。然而，相比传统的 ASP，ASP.NET 的底层执行模型发生了翻天覆地的变化。执行传统的 ASP 页面主要的任务是呈现 HTML，解释脚本代码，调用“组件对象模型”(Component Object Model, COM)代码，而 ASP.NET 引入了全新的、面向对象的执行模型。ASP.NET 的执行围绕着“公共语言运行库”(Common Language Runtime, CLR)中实现 IHttpHandler 接口的类。ASP.NET 包含许多已实现 IHttpHandler 的类，但也可以完全自行编写。但通常来讲，开发者编写的 ASP.NET 页面会在底层通过 ASP.NET 本身提供的 IHttpHandler 生成。

本章将重点介绍 ASP.NET 的执行模型，以便理解 ASP.NET 中各功能的内部机制。本章将会采用自底向上的方式解释简单 ASP.NET 页面的执行方式。在这个过程中，您将学习到多种 ASP.NET 编程技术，其中包括代码隐藏。您还将学习到 ASP.NET 编译模型的原理。最后，本章会讲解 ASP.NET 的“Web 窗体”架构的工作方式，以及 Microsoft Visual Studio 2010 是如何对其提供支持的。

您将从一个简单的页面开始来学习 ASP.NET 的各项编程技术。

 **注意** 执行配套资源中针对本章的示例代码要求 IIS 的支持。有关运行本章示例代码的注意事项，请阅读本书“前言”部分“示例代码”一节。

^① 译者注：也称“稻糠时代”，指 1995 年至 2001 年出现“互联网泡沫”(dot-com bubble)这段时间。

2.1 经典的 Hello World 程序

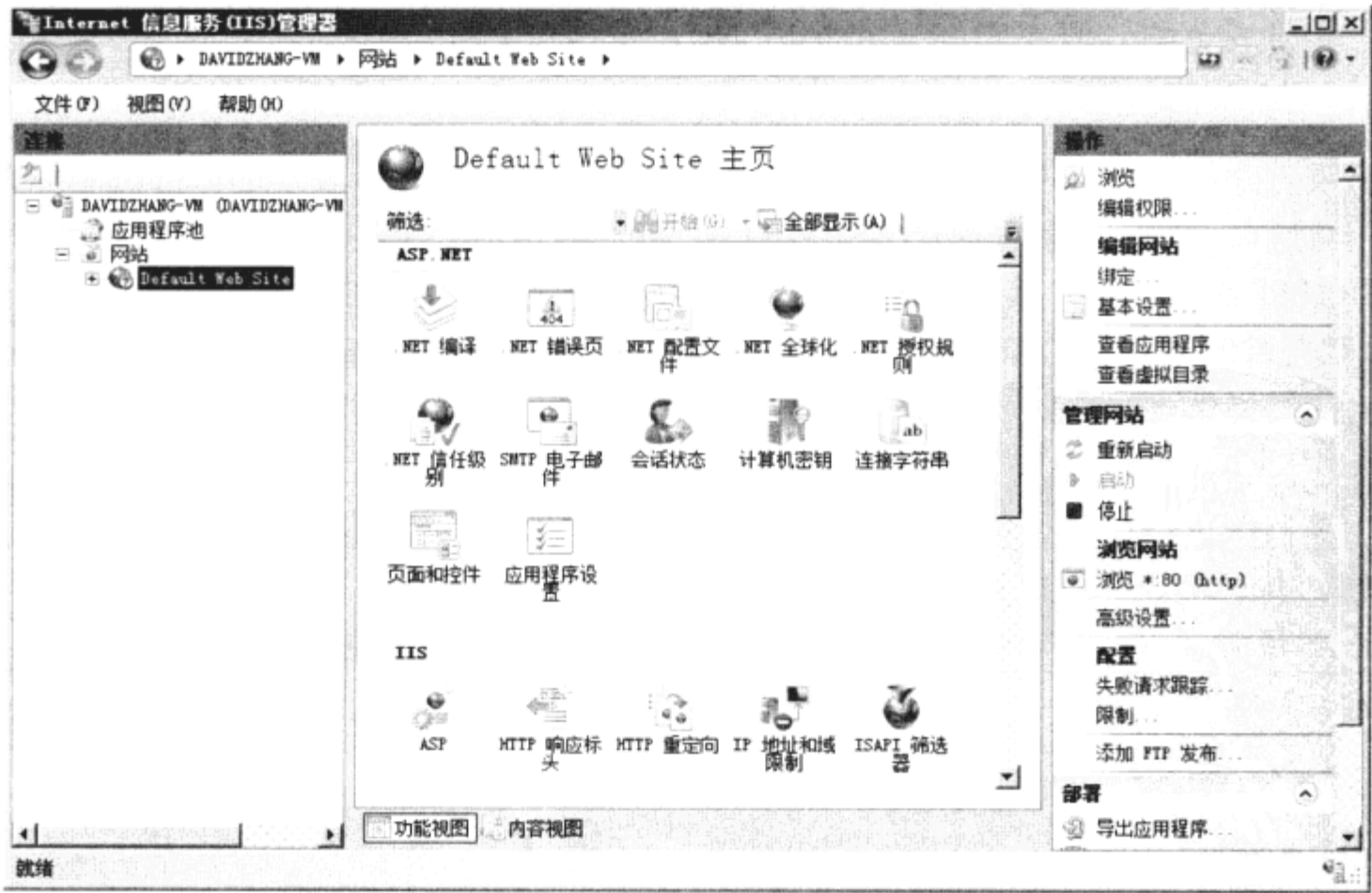
几乎所有介绍编程的书都从这个程序开始来介绍某种技术——该程序会向最终用户显示 Hello World 字符串。在本书中，我们的目的是向浏览器发送这个字符串。

为理解 ASP.NET 的工作原理，我们将制作这个最简单的网页，并将其开发成为一个 ASP.NET Web 应用程序。Visual Studio 是一个功能丰富的开发环境，它能够方便地开发和运行 Web 应用程序，但这里并不急于使用它。在完整地介绍 Visual Studio 的功能之前，我们将从头构建这个简单的示例，在此过程中您会了解 ASP.NET 的工作原理。

➤ 构建 HelloWorld Web 应用程序

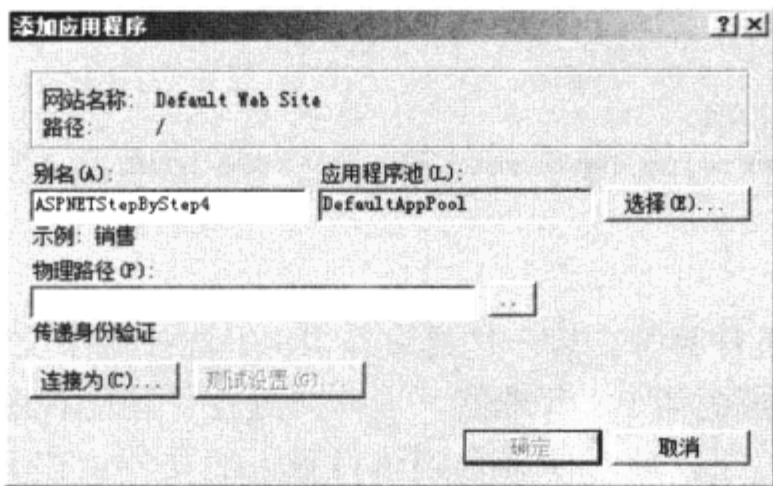
1. 创建将要包含 Web 应用程序文件的目录。可以使用命令行或 Windows 资源管理器来创建这个目录。虽然“Internet 信息服务”(IIS)并不在乎该目录的名称是什么，但最好为其起一个有意义的名字。本示例将使用 c:\aspnetstepbystepexamples。
2. 创建一个放置文件的应用程序/虚拟目录。首先，我们需要一个放置源代码的虚拟目录。正如之前介绍 Microsoft Windows 环境下 Web 应用程序的架构时提到的，IIS 通过虚拟目录来分隔不同的应用程序。除了提供对应用程序的管理功能之外，IIS 还会通过虚拟目录在 80 端口传入的请求与计算机上实际的目录之间建立映射。虚拟目录也决定了应用程序代码的执行位置。

打开“控制面板”，找到“管理工具”，启动“Internet 信息服务(IIS)管理器”。展开左侧节点，直至看到 Default Web Site 节点，如下图所示。



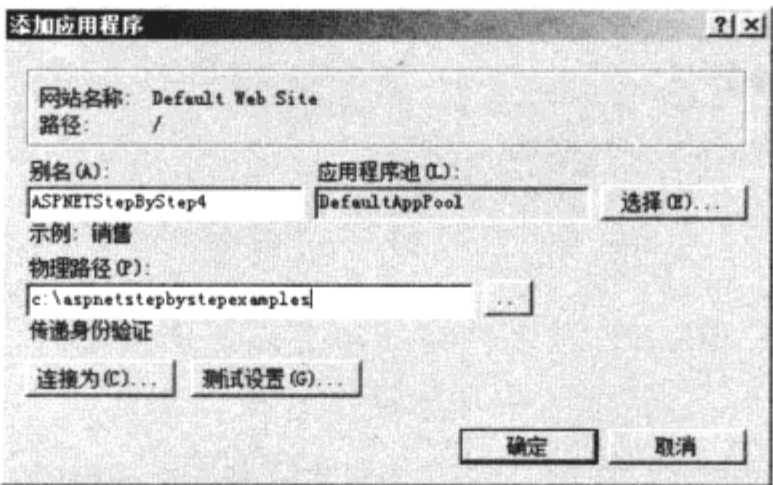
然后，在 Default Web Site 节点上右击，在快捷菜单中选择“添加应用程序”命令。(这

个示例给出的是 IIS 7.5 的操作方法。如果使用之前版本的 IIS，界面会稍有不同，但添加虚拟目录的方法是相同的。)IIS 会询问应用程序/虚拟目录的名称。如下图所示。



这里将网站命名为 ASPNETStepByStep4(别名)。这个就是发布 Web 应用程序时所使用的名称。例如，如果 Internet 用户要访问这个网站，则可以使用这样的 URL：
http://www.contoso.com/ASPNETStepByStep4。
contoso.com 是一个虚构的域名，这里仅用于演示。如果是在宿主计算机上访问该网站，
则服务器名称应改为 localhost。^①

如下图所示，这个对话框还会询问虚拟目录的物理路径。可以通过旁边的“浏览”按钮选择刚刚创建的物理目录，也可以直接输入该目录的路径。先不要关闭 IIS 管理工具，后面还会用到它。



单击“确定”按钮，创建虚拟目录。

- 3. 制作一个简单的 HTML 页面。最简单的方法是将文本存储在 HTML 文件中，然后浏览它。创建这样的文件，“记事本”程序就可以胜任，但也可以使用 Visual Studio。我们选择 Visual Studio，依次单击“文件”|“新建”|“文件”命令。文件类型选择“HTML 页面”，然后单击“打开”。这个新建的空文件会在 Visual Studio 编辑器中打开。输入下面<body>开标签和闭标签之间的 HTML 标记。将该文件保存到之前创建的物理目录下(之前映射到虚拟目录的那个物理目录)，并将文件命名为 HelloWorld.htm。

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

① 译者注：这个主机名称指向本地计算机，代表“环回地址”(loopback)。IPv4 的环回地址为“127.0.0.1”，IPv6 的环回地址为“0:0:0:0:0:0:0:1”(也可以表示为“::1”)。


```
<head>
  <title>Untitled Page</title>
</head>

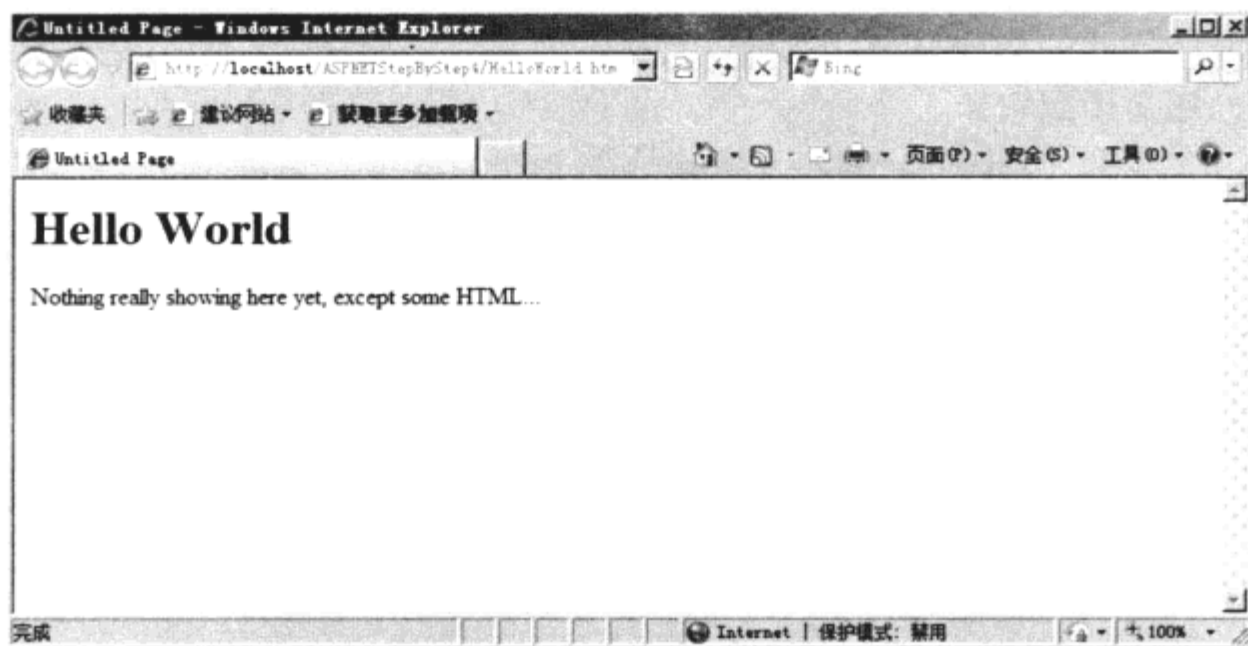
<body>
  <h1>Hello World</h1>
  Nothing really showing here yet, except some HTML...
</body>
</html>
```

4. 浏览这个页面。方法有两种。其一是通过在 IIS 中选择这个文件来浏览它。在 IIS 中导航至相应目录(如果未关闭 IIS 管理器, 当前应处在“功能视图”上)。单击详细视图底部的“内容视图”按钮。此时便会看到该目录下的文件。在 HelloWorld.htm 文件上右击, 然后选择“浏览”。其二是直接在浏览器的导航栏中输入完整的 URL:

http://localhost/ASPNETStepByStep4/helloworld.htm

此时, 浏览器会向服务器发送一个 HTTP 请求。在 Microsoft 平台上, IIS 会监视针对 HTM 扩展名的请求, 并将指定文件的内容直接返回给浏览器。由于我们使用的是标准的 HTML 标记, 因此浏览器会正确解析并显示它。

下图是在浏览器中看到的效果。



5. 将这个 HTML 文件转换为 ASP.NET 应用程序。下面我们将刚刚编写的 HelloWorld.htm 文件转换为 ASP.NET 运行库能够调用的文件类型。将这个文件转换成 ASP.NET 应用程序只需要简单的两步: 在该文件的顶部添加一行代码(Page 指令), 然后将 HelloWorld.htm 重命名为 HelloWorld.aspx。下面这段代码就是一个针对 ASP.NET 框架的 HelloWorld 实现(请确认在“文件”菜单中选择了“HelloWorld.htm 另存为”, 以 HelloWorld.aspx 文件名保存了该文件):

```
<%@Page Language="C#"%>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Untitled Page</title>
  </head>
```

```
<body>
  <h1>Hello World</h1>
  Nothing really showing here yet, except some HTML...
</body>
</html>
```

启动浏览器并导航至服务器虚拟目录中的该文件后，浏览器会显示下图所示的页面。



不可否认，在浏览器中显示这样一些文本显得有些微不足道。然而，这个示例反映了一个简单的 ASP.NET 应用程序是如何与 IIS 的协作方式。

6. 查看浏览器所解析的 HTML 源代码。为在浏览器中查看上一步的内容，可以在菜单中依次单击“查看”|“源文件”命令。此时会显示被浏览器处理的 HTML 源文件，如下所示：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Untitled Page</title>
</head>
<body>
  <h1>Hello World</h1>
  Nothing really showing here yet, except some HTML...
</body>
</html>
```

不难发现，这段文本几乎与 HelloWorld.aspx 中的代码一致(只是没有了<%@ Page Language="C#">这条 Page 指令)。本示例中页面的处理逻辑非常简单，ASP.NET 运行库只是将 HelloWorld.aspx 中的文本直接输出。

代码顶端的 Page 指令(directive)供 ASP.NET 运行库在编译时使用。本示例中的 Page 指令比较简单——它指示运行库基于 Page 类来编译代码，并将所遇到代码语法按 C#代码处理。ASP.NET 支持将.aspx 文件与程序集结合(稍后会看到)。

在后面的示例中，我们将看到 ASP.NET 会在运行时编译代码，并将编译后的程序集存储在临时目录中。HelloWorld.aspx 中还没有 C#代码，所以下面我们就来添加一些。

关于应用程序池

除了将传入的请求与实际的物理目录映射起来，IIS 6.x 和 IIS 7.x 还支持被称为“应用程序池” (application pool) 的功能。应用程序池旨在实现应用程序隔离。例如，假设要将 Web 应用程序与运行在同一台计算机上的由 IIS 管理的其他软件隔离。通过为每个 Web 应用程序创建独立的应用程序池，可以使 IIS 在单独的工作进程中运行每个应用程序。如果某个应用程序池中发生错误，其他应用程序仍能够继续运行而不受影响。

还可以通过应用程序池管理 Web 应用程序的安全性。不同应用程序可能需要不同的安全级别。

IIS 5.x 以 LocalSystem 帐户运行 ASP.NET 工作进程。LocalSystem 拥有系统管理员(system administrator)的权限。这存在许多安全隐患，因为该帐户几乎可以访问服务器上的所有资源。在 IIS 6.x 和 IIS 7.x 中，我们可以设置工作进程的身份，而应用程序池也处于这个级别。应用程序池默认工作在 NetworkService 帐户下(该帐户不具有 LocalSystem 所拥有的那么多权限)。

2.1.1 可执行代码与 HTML 的混合

传统的 ASP 有一种有趣的方式在页面嵌入代码段。ASP 一直都支持传统的脚本标签(<script></script>)，<script>标签之间的内容会被看作是可执行代码。然而，在传统的 ASP 中，脚本块会被发送到浏览器，运行其中的脚本是浏览器的工作。除了客户端脚本块，也可以在传统 ASP 页面中定义在服务器端解释的脚本块。其中的方法通常用于完成数据库查询之类的任务。为使代码在服务器上执行，需要用尖括号和百分号标记可执行代码段，如：

```
<%ExecuteMe() %>
```

ASP.NET 也支持服务器端代码的执行。以内联方式执行的代码同样使用<% %>标签进行标记。在 ASP.NET 解析代码文件来生成表示页面的运行时类(稍后介绍)的时候，它会将执行标签之间发现的代码按可执行代码处理。唯一的要求是执行标签之间的代码是有效的.NET 语言(如 C# 或 Visual Basic，具体选择哪种要在 Page 指令中指定)。^①

➤ 以内联方式添加可执行代码

1. 向 Web 应用程序中添加可执行代码。在 Visual Studio 中创建一个空的文本文件。输入以下代码，并将其保存为 HelloWorld2.aspx。

```
<%@Page Language="C#" Debug="true"%>
<html>
  <body>
```

^① 译者注：原文只提到了 C#，但这会造成不必要的误解，因为.aspx 文件中以内联方式嵌入的代码可以是任何 .NET Framework 支持的语言，其中包括 Visual Basic、C# 和 Jscript。这里稍作修改。

```
<h1>Hello World!!!</h1>
<%
    // This block will execute in the Render_Control method
    Response.Write("Check out the family tree: <br/><br/>");
    Response.Write(this.GetType().ToString());
    Response.Write(" which derives from: <br/> ");
    Response.Write(this.GetType().BaseType.ToString());
    Response.Write(" which derives from: <br/> ");
    Response.Write(this.GetType().BaseType.BaseType.ToString());
    Response.Write(" which derives from: <br/> ");
    Response.Write(
        this.GetType().BaseType.BaseType.BaseType.ToString());
    Response.Write(" which derives from: <br/> ");
    Response.Write(
        this.GetType().BaseType.BaseType.BaseType.BaseType.ToString());
%>
</body>
</html>
```

上述代码几乎与在之前传统 ASP 应用程序中见到的一样，都包含对 `Response` 对象的引用。在传统的 ASP 中，`Response` 对象是几个重要的内部对象之一，对应用程序的执行块总是可用。从根本上说，传统 ASP 中的 `Response` 是 COM 对象，在底层组件 (ISAPI DLL) 管理的线程上运行。ASP.NET 也有 `Response` 对象。然而，这个 `Response` 对象是 `HttpContext` 的一部分，受 ASP.NET 管线管理，除了名称外，它与传统 ASP 中的同名对象无任何联系。

2. 使用 Windows 的 Internet Explorer 浏览这个 ASP.NET 页面，其效果如下图所示。



`HelloWorld2.aspx` 生成的输出反映了 ASP.NET 执行模型的重要一面。在深入讲解该模型之前，让我们先比较一下该练习中内联的代码与该页面对浏览器输出的结果。我们会发现上述代码中包含这样的语句：

```
Response.Write(this.GetType().BaseType.ToString());
```


C# 中的 `this` 关键字代表当前类的实例。显然，这说明执行的代码是类实例的某个成员函数。浏览器中显示的输出表明，生成所呈现的 HTML 的类叫做 `ASP.helloworld2_aspx`，该类派生自 `System.Web.UI.Page` 类。本章稍后会更详细地讲解这方面内容。

2.1.2 服务器端的可执行块

ASP.NET 还支持服务器端的代码块(不仅是内联的执行标签)。ASP.NET 为脚本标签新引入了 `runat` 特性(attribute)，可以告知 ASP.NET 运行库在服务器端执行其中的代码块。

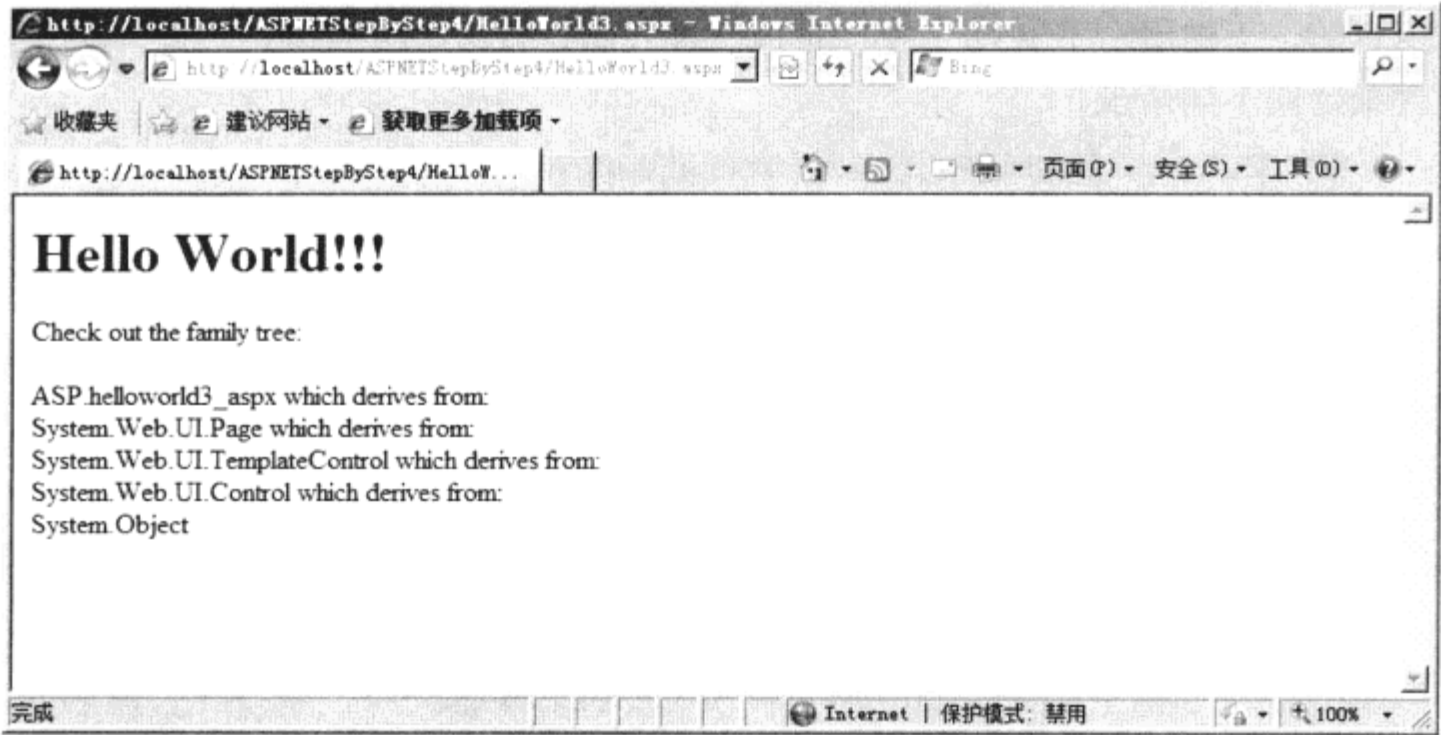
➤ 使用脚本块来添加可执行代码

1. 在页面中添加可执行脚本块。在 Visual Studio 中新建一个文本文件。在 Visual Studio 编辑器中输入以下代码。注意，这段代码隔离了要呈现的 HTML 与在服务器端运行的脚本块。将该文件命名为 `HelloWorld3.aspx`，并保存于虚拟目录中。

```
<%@Page Language="C#" Debug="true"%>
<script runat="server">
    void ShowLineage()
    {
        Response.Write("Check out the family tree: <br/><br/>");
        Response.Write(this.GetType().ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(this.GetType().BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(this.GetType().BaseType.BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(
            this.GetType().BaseType.BaseType.BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(
            this.GetType().BaseType.BaseType.BaseType.BaseType.ToString());
    }
</script>
<html>
    <body>
        <h1>Hello World!!!</h1>
        <%
            ShowLineage();
        %>
    </body>
</html>
```

与内联执行块相同，对于脚本块的内容，最重要的条件是其所使用的语法要与 `Page` 指令中指定的语言一致。这个示例定义了一个名为 `ShowLineage()` 的方法，它会在页面中被调用。

2. 浏览该页面。可以注意到，`HelloWorld3.aspx` 与 `HelloWorld2.aspx` 的输出是相同的。



在包含 ShowLineage 方法的<script>标签中添加 runat="server"属性，会使 ASP.NET 在服务器端执行该标签中的代码。传统 ASP 根据指定的脚本语言来解释脚本块，但 ASP.NET 的执行模型则完全不同——整个页面会被编译成一个类，由“公共语言运行库”(CLR)运行。下面将介绍 ASP.NET 编译模型的工作原理。

ASP.NET 架构概述

当 HTTP 请求到达 Web 服务器时，它会在许多服务器端对象之间进行传递，同时被处理。请求在被服务器处理之前，会穿越 IIS/ASP.NET 管线。理解 HTTP 请求在 ASP.NET 中所走路径的最佳方法是通过浏览器发起请求，并跟随它，直到它被“Internet 信息服务”和 Web 应用程序处理。

当某个最终用户在输入 URL 并按 Enter 键后，浏览器会向目标网站发送一条 HTTP GET 请求。在到达 Web 服务器之前，该请求会穿越一系列路由器。如果 Web 服务器的操作系统中有监听 80 端口的软件，那么该软件便会对这个请求进行处理。在 Microsoft 平台上，监听 80 端口的软件通常是 IIS。ASP.NET 目前支持 3 个版本的 IIS：IIS 5.x(Windows XP Professional 操作系统搭载的)、IIS 6.x(Windows Server 2003 操作系统搭载的)、IIS 7.x(Windows Vista、Windows 7 和 Windows Server 2008 操作系统搭载的)。

不论选择哪个版本的 IIS，处理请求的基本流程都是一样的。IIS 维护着文件扩展名与解释请求的二进制组件之间的映射(稍后会看到所谓的二进制组件)。当请求传入后，IIS 会读取请求中的文件名，并将该请求交给相应的组件。

早期的 IIS(7.0 之前的版本)实现了与 ASP.NET 无关的客户端身份验证和输出缓存。也就是说，IIS 与 ASP.NET 各自对这些功能的实现是独立的。IIS 7.x 则集成了部分 ASP.NET 的功能(其中的一些将会在后续章节介绍)。从 ASP.NET 开发者的角度看，IIS 7.x 引入的变化在于，.NET 功能以“集成”(Integrated)模式运行使得其成为核心管线的一部分。“Forms 身份验证”这样的功能如今可用于多种内容——而不仅仅是 ASP.NET 窗体。例如，可以使用一种统一的身份验证方法来保护整个网站。

ASP.NET 4 从入门到精通

出于演示目的，下图展示了 IIS 7.x 在“集成”模式下运行所采用的模块映射，IIS 7.x 会根据该映射来定向不同请求。



下图也是出于演示目的，展示了 IIS 7.x 在“集成”模式下运行所采用的处理程序映射。



除了“集成”模式外，IIS 7.x 还支持“经典” (Classic) 模式，以实现向后兼容。在“经典”模式下运行时，IIS 7.x 会采用另一套模块和处理程序架构，它们会将处理工作交给传统的二进制组件(即 ISAPI DLL)。

下图展示了 IIS 7.x 在“经典”模式下运行时所采用的模块映射。



下图展示了 IIS 7.x 在“经典”模式下运行时所采用的处理程序映射。



一旦 IIS 监听到请求并将其映射到工作进程，该请求便会穿越一个特定的管线。后文会更深入地阐述几种管线的每一部分。下面先简单了解一下在 IIS 7.0 之前的版本中，请求所走的路径。

1. 请求到达 IIS。

- 2. IIS 将请求交给 aspnet_isapi.dll。
 - a. 如果运行的是 IIS 5.x,那么 IIS 会通过 asp_isapi.dll 将请求交给 aspnet_wp.exe 处理。
 - b. 如果运行的是 IIS 6.x, 那么请求已处于工作进程中。
- 3. ASP.NET 通过 HttpContext 的实例对请求进行包装。
- 4. ASP.NET 将请求交给 HttpApplication 对象(或 HttpApplication 的派生对象)的实例。
- 5. 如果该应用程序对象订阅了请求处理事件, 那么 HttpApplication 会引发这些事件。订阅这些事件的 HttpModule 对象也会收到通知。
- 6. 运行库实例化一个处理程序, 并对请求进行实际的处理。

图 2.1 展示了 IIS 5.x 版与 ASP.NET 处理 HTTP 请求的协作方式。图 2.2 展示了 IIS 6.x 版与 ASP.NET 处理请求的协作方式。

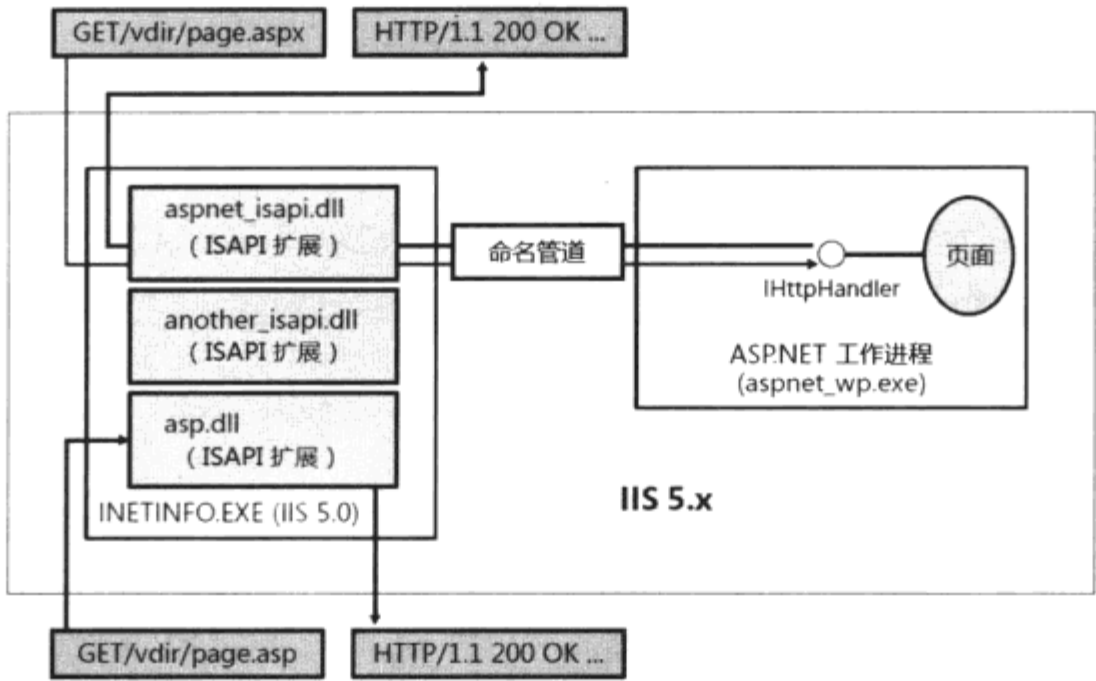


图 2.1 IIS 5.x 与 ASP.NET 的协作方式

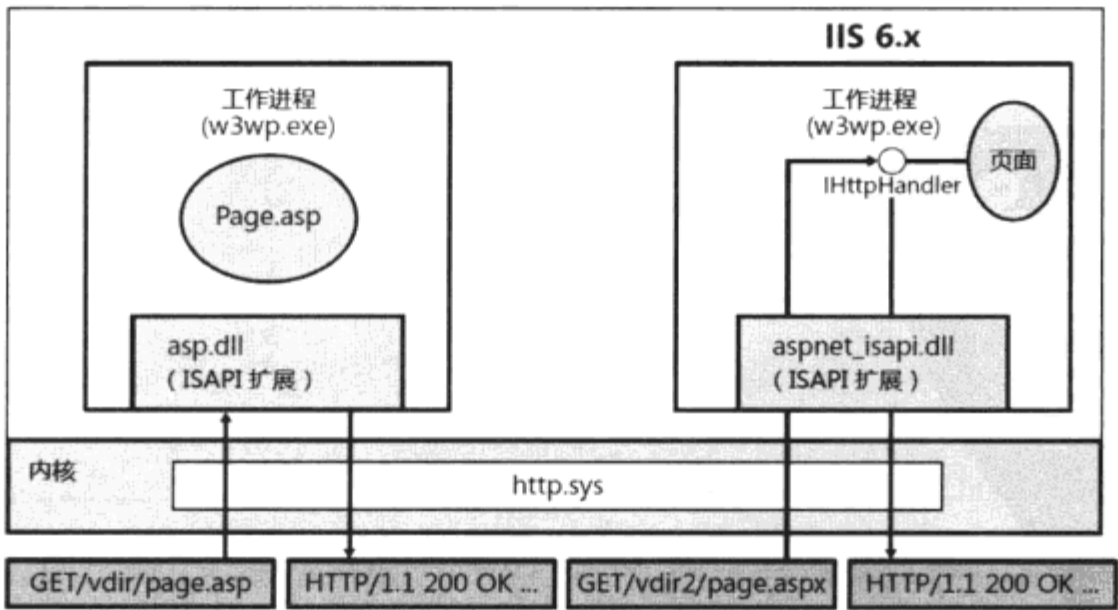


图 2.2 IIS 6.x 与 ASP.NET 的协作方式

比较而言, IIS 7.x 请求处理的路径稍有不同。具体如下所述。

- 1. 浏览器向 Web 服务器请求某个资源。
- 2. 服务器上的 HTTP.SYS 接收该请求。
- 3. HTTP.SYS 使用 WAS^① 来查找配置信息，以便将请求传给“WWW 服务”。
- 4. WAS 将配置信息返回给“WWW 服务”，以便配置 HTTP.SYS。
- 5. WAS 启动请求所针对的应用程序池中的工作进程。
- 6. 该工作进程对请求进行处理，并将响应返回给 HTTP.SYS。
- 7. HTTP.SYS 将这个响应发送给客户端。

图 2.3 展示了 IIS 7.x 与 ASP.NET 之间的关系。

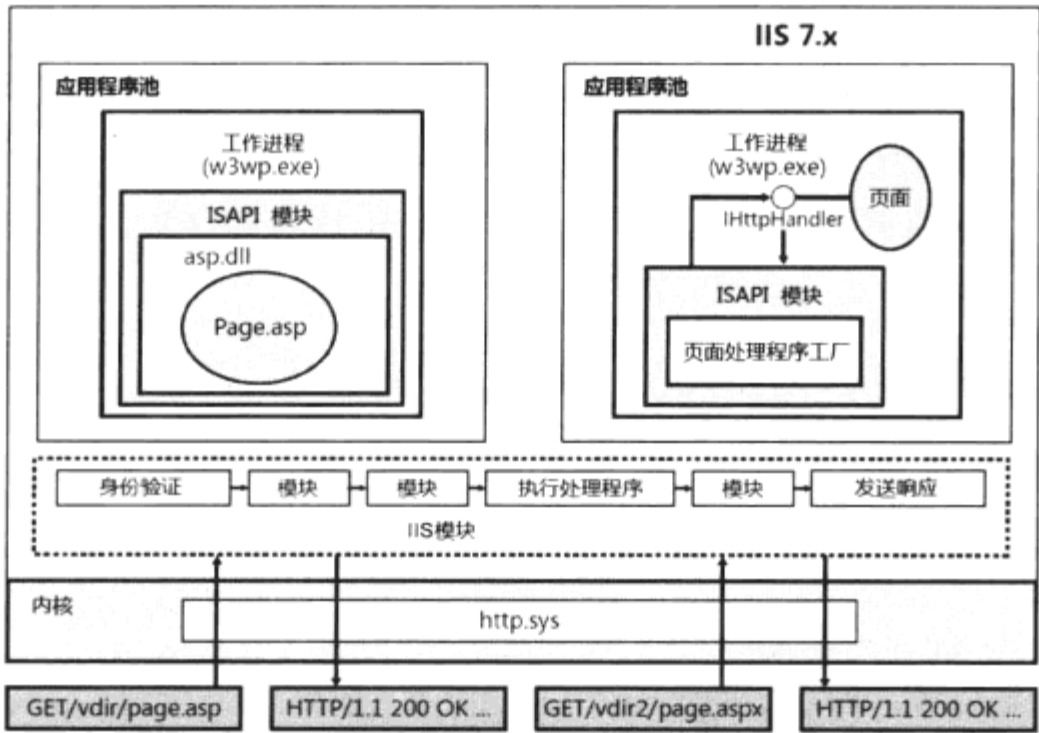


图 2.3 IIS 7.x 与 ASP.NET

后面几章会详细介绍请求穿越 ASP.NET 管线所涉及的内容。开发者可以在 ASP.NET 管线的很多位置上添加插件来处理请求的不同方面。例如，如果要进行某些预处理，则可以重写 `HttpApplication` 类中的处理程序，也可以编写 HTTP 模块，然后将其插到管线上。在构建 Web 用户界面方面，`System.Web.UI.Page` 类提供的特性非常之多。而且管线也十分灵活，允许开发者方便地编写自定义的处理程序。

2.2 ASP.NET 编译模型

Microsoft 对传统 ASP 开发环境最大的改进之一是通过类构建了一套 Web 请求处理框架。将请求处理构建到一个基于类的框架中，就实现了编译型的 Web 处理框架。当 ASP.NET

① 译者注：这里的 WAS 代表“Web 应用程序服务”(Web Application Service)。

页面首次被访问时，它会被编译为程序集。

这是一项重大改进，因为后续的访问可以直接通过加载这些程序集来进行处理。传统的 ASP 需要重复解释相同的脚本，而 ASP.NET 应用程序被编译为 .NET 程序集会最终获得更好的性能和更高的安全性。编译过的代码执行效率更高，因为它不需要解释。此外，托管的运行库是类型安全的环境，不会遇到脚本环境(如传统的 ASP)中特有的错误和异常。

对 Web 请求框架进行编译还能够在调试方面得到更好的健壮性和一致性。只要使用 Visual Studio 运行 ASP.NET 应用程序，便可以像调试普通桌面应用程序那样来调试它。

ASP.NET 会自动编译 .aspx 文件。为使 .aspx 页面被编译，用浏览器浏览该文件即可。此时，ASP.NET 会将页面编译成一个类。不过，包含该类的程序集并不在虚拟目录中。ASP.NET 会将结果程序集复制到一个临时目录中。

Visual Studio 的 .NET 版本总会包含一个名为“中间语言反汇编程序”(Intermediate Language Disassembler, ILDASM)的工具，能够通过反射对程序集进行反编译，这样便可以看到其中的内容。通过该工具得到的结果是一个易于使用的树状视图，可以通过它来查看程序集的内容。稍后您会发现这很重要。(如果希望更深入地了解这种程序集，请阅读中间语言代码，也可以使用 ILDASM。^①)

为查看 ASP.NET 生成的程序集，可以使用下文给出的方法。

➤ 查看 ASP.NET 程序集

1. 为运行 ILDASM，请打开“Visual Studio 命令提示(2010)”，然后输入 **ILDASM**。
2. 选择“文件”|“打开”命令。
3. 查找 ASP.NET 运行库编译的程序集。进入 C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET Files\aspnetstepbystep4\ (在 64 位计算机上，需要选择“Framework64”子目录)。在本书截稿前，其中的“版本”文件夹为“v2.0.50727”。这个子目录在未来可能有所变化。^②此时，我们会看到其中包含一些名称怪异的目录。例如，ASP.NET 会生成 4076b23e 和 ac422a67 这样的目录名称。对于不同的计算机，这种目录的名称也会有所不同。我们很难判断哪些目录包含刚刚执行的代码(但可以按文件创建的日期和时间查找)，所以需要连续进入某个路径，直到看到满足条件的 .dll 文件。根据应用程序运行次数的不同，可能会看到多个文件。每次打开一个文件，直到在 ILDASM 中看到类似图 2.4 展示的内容。

① 译者注：有一个叫 .NET Reflector[®] 的免费工具，能够将程序集反编译为 IL、C#、Visual Basic .NET、F# 等语言。该工具可以到这个地址下载：<http://reflector.red-gate.com/>。

② 译者注：“版本号”文件夹的选择与应用程序池所使用的 .NET 版本有关。对于要查看的 Web 应用程序来说，如果其应用程序池使用的是 .NET Framework v4.0.30319，则需要在 Framework 目录下相应地选择 v4.0.30319 文件夹。应用程序池使用的 .NET Framework 版本可以在 IIS 管理器的“应用程序池”节点下查看(应用程序池可能有多)。虚拟目录指定的应用程序池的名称，可以在其属性中查看(默认使用 DefaultAppPool)。

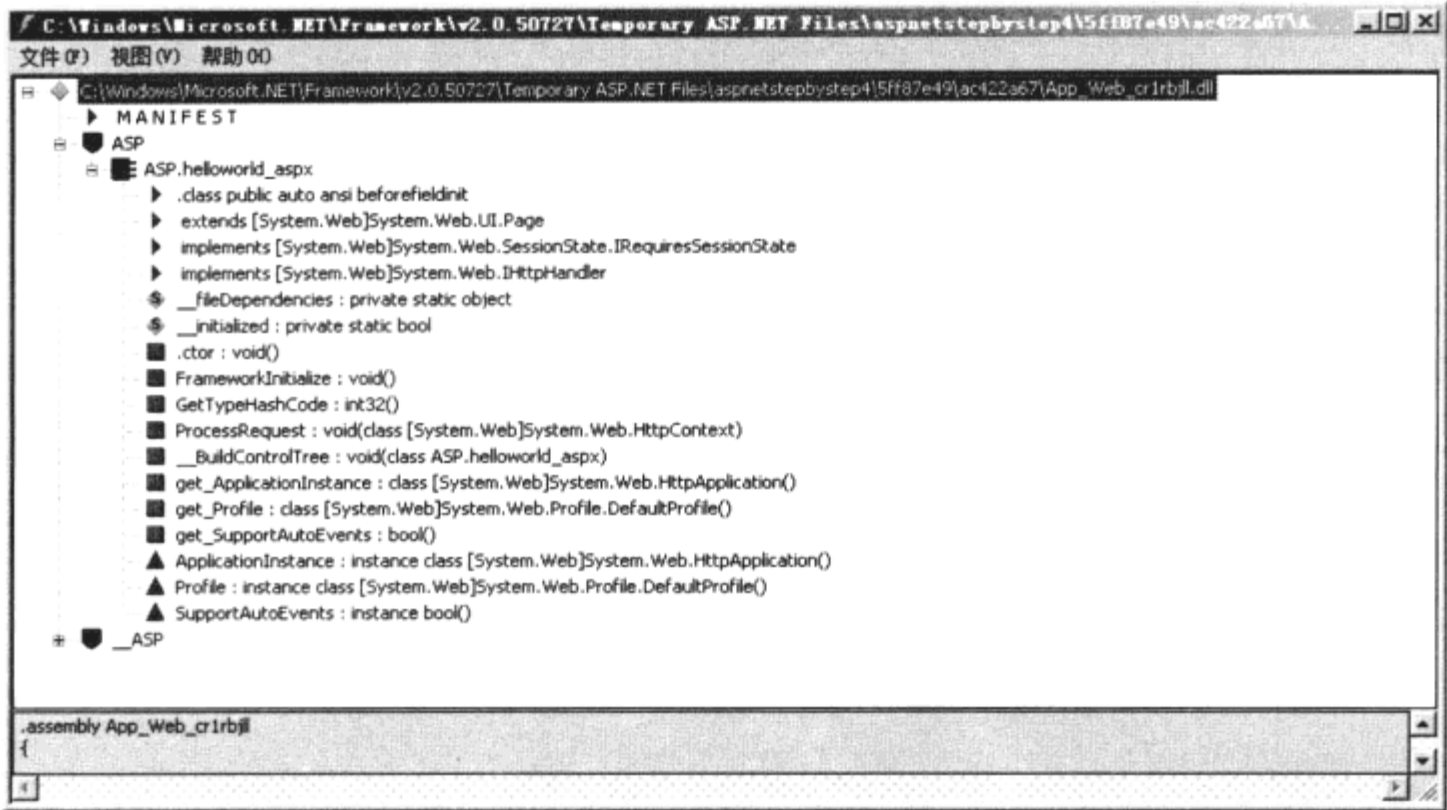


图 2.4 在 ILDASM 中查看运行 HelloWorld.aspx 后 ASP.NET 生成的程序集

ASP.NET 自 1.0 版本开始就采用这种临时目录策略了。ASP.NET 将这些文件复制到临时目录的原因是为了解决传统 ASP 中错误长时间驻留的问题。传统 ASP 网站往往依赖于 COM 对象来执行某些复杂的操作(如数据库查询和事务)。在部署传统 ASP 网站，并且用户开始访问后，代码文件便会被锁定。当然，也有例外(如在升级或修改网站的情况下代码文件就不会被锁定)。

传统的 ASP 在执行期间锁定代码文件意味着，如果不停止网站运行，则无法向虚拟目录中复制新文件。在许多 Web 部署场景中，这都会带来麻烦。由于 ASP.NET 将文件和组件复制到临时目录下，并“在这个临时目录中运行它们”，因此这些文件不会被锁定。如果要更新某个组件，将新的程序集复制到虚拟目录下即可。之所以能够这样做就是因为 ASP.NET 不锁定文件。

2.3 编码风格

当今的 ASP.NET 除了支持内联代码(即直接在服务器端脚本块中添加可执行代码)，还提供了两种管理代码的方式：ASP.NET 1.x 代码隐藏(code behind)和现代 ASP.NET 代码旁置(code beside)。ASP.NET 支持代码隐藏是为了向后兼容。代码旁置是 Visual Studio 2010 引入的风格。下面让我们看看两者的差异。

2.3.1 ASP.NET 1.x 风格

现在的 ASP.NET 仍支持 ASP.NET 1.x 风格的代码隐藏。如果要运行当时遗留下来的代码，这种兼容性就显得尤为重要了。通过在.aspx 文件中使用代码隐藏指令，可以将幕后执行的

ASP.NET 4从入门到精通

代码置于一个单独的类中，并使用 **Page** 指令来告知 ASP.NET 对页面应用哪个类。然后，可以为 ASP.NET 指定包含该类源代码的文件的名称。例如，假设将代码放在一个名为 **HelloWorld4Code.cs** 的文件中：

```
using System.Web;
public class HelloWorld4Code : System.Web.UI.Page
{
    public void ShowLineage()
    {
        Response.Write("Check out the family tree: <br/><br/>");
        Response.Write(this.GetType().ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(this.GetType().BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(this.GetType().BaseType.BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(
            this.GetType().BaseType.BaseType.BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(
            this.GetType().BaseType.BaseType.BaseType.BaseType.ToString());
    }
}
```

下面是一个使用 **HelloWorld4Code** 类来驱动自身的页面：

```
<%@Page Language="C#" Inherits="HelloWorld4Code"
    Src="HelloWorld4Code.cs" Debug="true"%>
<html>
    <body>
        <h1>Hello World!!!</h1>
        <%
            this.ShowLineage();
        %>
    </body>
</html>
```

如果采用 ASP.NET 1.x 风格的代码隐藏，ASP.NET 会读取 **Page** 指令的 **Src** 属性，并编译其指定的文件。ASP.NET 通过 **Inherits** 属性来确定用于运行页面的基类。在上面的示例中，ASP.NET 会使用 **HelloWorld4Code** 类来派生当前页面。

使用 **Src** 属性，可以告知 ASP.NET 运行库去编译该属性的值所指定的文件。ASP.NET 运行库会将其编译并复制到临时目录下。另外，我们还可以预先将包含 **HelloWorld4Code** 类的文件编译为程序集。如果这样，预编译的程序集必须位于虚拟目录的 **bin** 目录中。如果预先编译页面类，并将相应程序集置于了 **bin** 目录中，我们甚至可以不用关心源代码文件。在缺少 **Src** 属性的情况下，ASP.NET 运行库会在 **bin** 目录下查找 **Inherits** 属性所指定的类。

2.3.2 现代 ASP.NET 风格

ASP.NET 的另一种编码风格起始于 2.0 版本。这个模型有时被称为“代码旁置”。让我们先看看下面这个 ASP.NET 页面：

```
<%@Page Language="C#" CodeFile="HelloWorld5Code.cs"
    Inherits="HelloWorld5Code"%>
<html>
  <body>
    <h1>Hello World!!!</h1>
    <%
      // This block will execute in the RenderControl method
      ShowLineage();
    %>
  </body>
</html>
```

该页面引用的代码位于 HelloWorld5Code.cs 文件中：

```
using System.Web;
public partial class HelloWorld5Code : System.Web.UI.Page
{
    public void ShowLineage()
    {
        Response.Write("Check out the family tree: <br/><br/>");
        Response.Write(this.GetType().ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(this.GetType().BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(this.GetType().BaseType.BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(
            this.GetType().BaseType.BaseType.BaseType.ToString());
        Response.Write(" which derives from: <br/> ");
        Response.Write(
            this.GetType().BaseType.BaseType.BaseType.BaseType.ToString());
    }
}
```

在这个示例中，ASP.NET 会通过 CodeFile 属性的值来确定要编译的代码。ASP.NET 会尝试查找实现该页面逻辑的分部类(partial class)。分部类使我们可以将类型(类、结构或接口)的定义拆分到多个源代码文件中^①，即每个文件中包含类型定义的一部分。将这些源代码文件编译后便会形成完整的类。在使用自动生成的代码时(如由 Visual Studio 生成的)，这种特性便显得非常有用。我们可以对分部类进行扩充，同时不影响原有代码。Visual Studio.NET

① 译者注：这些文件要求位于同一个程序集中，不论这个程序集是.exe 还是.dll。

2010 倾向于使用代码旁置/分部类。^①

下面的两段代码(清单 2.1 和清单 2.2)共同实现了一个名为 SplitMe 的类。

清单 2.1 Partial1.cs

```
// Partial1.cs
using System;

public partial class SplitMe
{
    public void Method1()
    {
        Console.WriteLine("SplitMe Method1");
    }
}
```

清单 2.2 Partial2.cs

```
// Partial2.CS
using System;

public partial class SplitMe
{
    public static void Main()
    {
        SplitMe splitMe = new SplitMe();
        splitMe.Method1();
        splitMe.Method2();
    }

    public void Method2()
    {
        Console.WriteLine("SplitMe Method2");
    }
}
```

为编译清单 2.1 和清单 2.2 中的代码,可以使用 Visual Studio 生成一个项目,也可以在“Visual Studio 命令提示(2010)”中使用以下命令行(如果这些文件是零散的):

```
csc /t:exe Partial1.cs Partial2.cs
```

该命令会生成一个名为 Partial2.exe 的可执行文件。

在了解了如何手动编写 ASP.NET 源代码后,下面让我们看看如何将 Visual Studio 和 ASP.NET 结合在一起。Visual Studio .NET 2010 在创建和开发 Web 应用程序方面引入了许多新特性,正如后续的示例将要展示的。

① 译者注: ASP.NET 会根据.aspx 文件生成一个类。在 ASP.NET 1.x 中, Src 属性指定的文件中的类是这个类的基类;而在 ASP.NET 2.0 和更高版本中, CodeFile 属性指定的文件中的类是一个同名的类(分部类),即是同一个类的另一部分。这是两种风格的主要区别之一。

2.4 ASP.NET HTTP 管线

ASP.NET 1.0 对传统 ASP 的改进是巨大的。ASP.NET 1.0 精心定义了代码处理模块, 而这些模块合在一起构成了 ASP.NET HTTP 管线(pipeline)。传统的 ASP 由几种不同的组件构成(IIS、“Web 应用程序管理器”和 ASP ISAPI DLL)。Request 和 Response 对象是 COM 对象, 处在 IIS 的线程上。如果要在 ASP 上下文以外进行某些处理, 则需要编写 ISAPI 筛选器。如果要编写在处理期间执行的代码, 则必须在实现 IDispatch 接口的 COM 对象中进行(该接口对可以使用的数据类型有很大限制, 并且性能欠佳)。如果要编写请求处理代码(在 ASP 上下文之外), 则必须单独编写 ISAPI DLL。ASP.NET HTTP 管线也能够实现这些任务, 但是以一种更易管理的方式。

在 ASP.NET 中, 应用程序可以在 HttpModule 中进行预处理(preprocess)和后处理(postprocess)。如果将 IIS 5.x 或 IIS 6.x 作为 Web 服务器, 那么 ASP.NET 管线是独立的。在 aspnet_isapi.dll 将控制权交给 ASP.NET 工作进程后, 请求便完全由 ASP.NET 负责处理。IIS 7.x Web 服务器本身集成了 ASP.NET 管线, 因而可以将大部分 ASP.NET 服务交给非 ASP.NET 内容负责。不论使用哪个版本的 IIS, 应用程序都有机会通过 HttpApplication 对象来处理应用程序范围内的事件。由于有了 ASP.NET 对象模型, 因而不需要单独的、基于 COM 的脚本对象。所有请求的端点(endpoint)都是一个 IHttpHandler 的实现。ASP.NET 已提供了一些 IHttpHandler 的实现(如 System.Web.UI.Page 和 System.Web.Services.WebService)。不过, 开发人员也可以编写自己的实现(稍后有演示)。

2.4.1 IIS 5.x 和 IIS 6.x 的管线

一旦请求到达由 ASP.NET 运行库管理的 AppDomain(应用程序域), ASP.NET 便会使用 HttpRequest 类来存储请求信息。随后, 运行库会使用 HttpContext 类对请求信息进行包装。HttpContext 类包含所有有关请求的信息, 其中含有对当前请求的 HttpRequest 和 HttpResponse 对象的引用。运行库会生成一个 HttpApplication 的实例(如果其尚不可用), 然后引发一系列应用程序范围的事件(如 BeginRequest 和 AuthenticateRequest)。依附在管线上的各 HttpModule 也会收到这些事件的通知。最终, ASP.NET 会找到处理请求的处理程序, 创建其实例, 并通过它来对请求进行实际的处理。在处理程序完成其任务后, ASP.NET 还会向 HttpApplication 和各 HttpModule 引发一系列后处理事件。

图 2.5 展示了在 IIS 6.x 中 ASP.NET 工作进程内部的管线的结构(IIS 5.x 与之唯一的差别是工作进程的名称)。

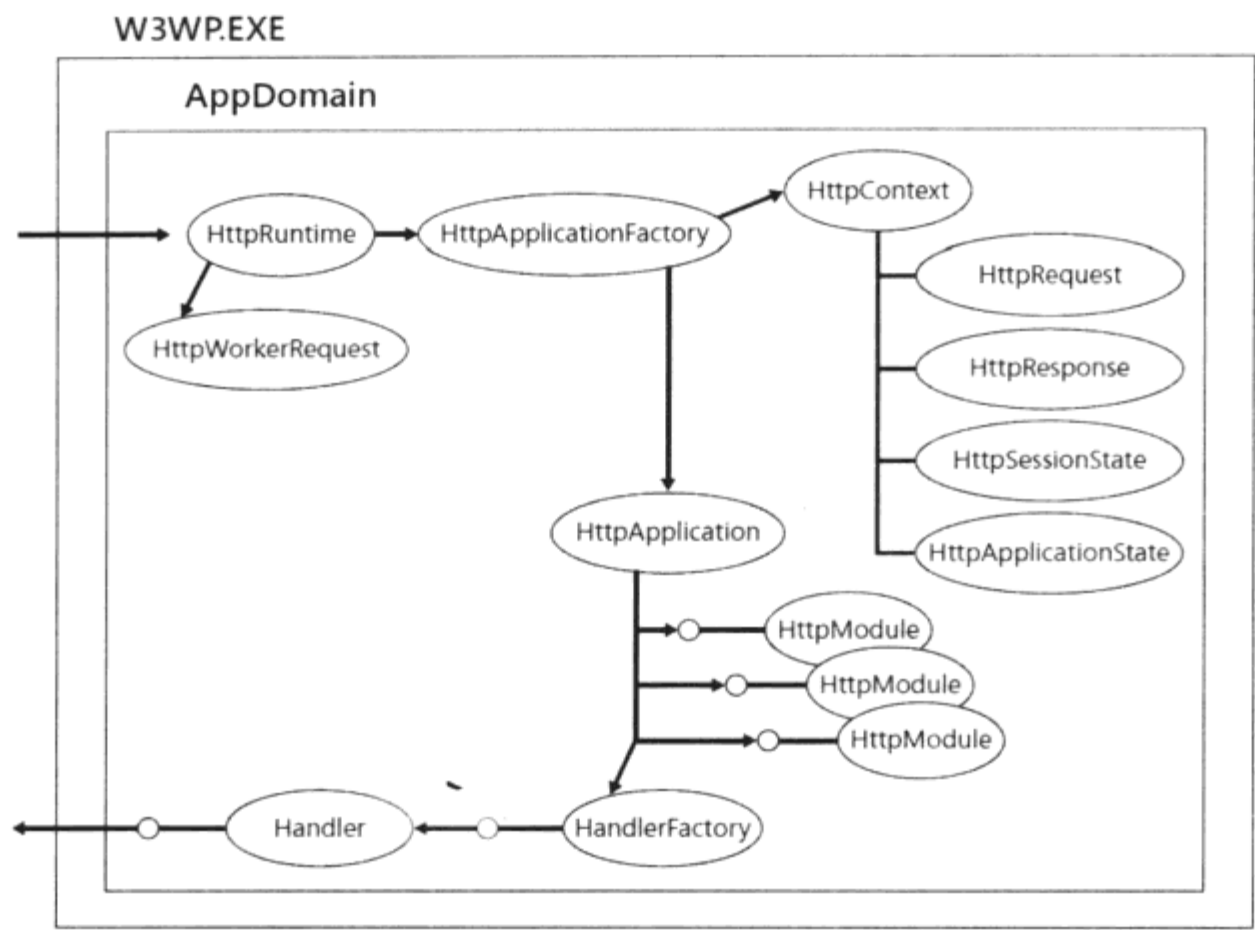


图 2.5 ASP.NET 中 HTTP 管线的主要组件

2.4.2 IIS 7.x 集成的管线

IIS 7.x 集成的管线与自首次发布后就一直沿用的 ASP.NET HTTP 管线非常接近(如图 2.5 所示)。正如之前使用 IIS 7.x 管理控制台所看到的，IIS 7.x 集成的管线会像 ASP.NET 之前版本中的 HTTP 管线那样使用模块和处理程序。不过，ASP.NET HTTP 管线完全在 ASP.NET 工作进程中运行，而 IIS 7.x 的管线直接由 IIS 自身运行。两种管线的执行方式基本是相同的，因而 **HttpApplication** 暴露的应用程序范围的事件会像之前那样工作(稍后会详细介绍这种事件)。在 IIS 7.x 的“集成”模式下运行应用程序，请求不再由 **aspnet_isapi.dll** 经手。IIS 7.x 会使请求直接穿越模块和处理程序。

2.4.3 管线内部的组件

虽然开发者接触不到管线的部分内容，但其中有一部分是可以直接利用的，而且由此提供了一种很有用的方法来管理穿越管线的请求。对于开发者来说，管线中能接触到的最重要的部分是 **HttpApplication**、**HttpContext**、**HttpModule** 和 **HttpHandler**。

下面几小节将详细介绍 HTTP 请求必经之路中的这些重要部分。

2.4.3.1 HttpApplication

至此，我们已经了解了 Web 应用程序与普通桌面应用程序的本质区别。我们所编写的代码要负责向客户端返回某种 HTML 响应。这个模型在许多方面要追溯到 20 世纪 70 年代中期

的“终端-主机”模型。在 ASP.NET 中，请求的端点是 `IHttpHandler` 的实现(如根据 ASP.NET Web 窗体代码而生成网页的处理程序)。

HTTP 处理程序的生命周期非常短暂。这些处理程序的实例在处理完请求后便会消失。对于相对简单的应用程序，这并没有什么问题。然而，对于较大规模的商业应用程序来说，这一点就需要考虑了。因为这种生命周期短暂的处理程序无法实现应用程序范围的功能。例如，为了避免与数据库的频繁交互，要对数据进行缓存，这就需要将数据存储在一个所有 HTTP 处理程序都可以访问的位置。

`HttpApplication` 类就是为了满足这一需求而存在的——充当处理请求的汇集点。在 Web 应用程序的生命周期中，可以通过 `HttpApplication` 对象存储应用程序范围的数据和处理针对应用程序的事件。

2.4.3.2 HttpContext

`HttpContext` 类也充当一种中心机构的角色，我们可以在请求穿越管线期间通过它访问当前请求的信息。事实上，有关当前请求的每种信息都可以通过 `HttpContext` 获得。虽然 `HttpContext` 组件包含的只是对管线其他部分的引用，但将这些引用集中在一起可以更方便地对请求进行管理。

下面罗列了 `HttpContext` 中开发 Web 应用程序时最常用到的成员，它们都以属性的形式暴露出来。

```
Class HttpContext
{
    public static HttpContext Current {...};
    public HttpRequest Request {...};
    public HttpResponse Response {...};
    public HttpSessionState Session {...};
    public HttpServerUtility Server {...};
    public HttpApplicationState Application {...};
    public HttpApplication ApplicationInstance {...};
    public IDictionary Items {...};
    public IPrincipal User {...};
    public IHttpHandler CurrentHandler {...};
    public Cache Cache {...};
    ...
}
```

我们可以随时通过 `Current` 属性获取当前请求。在许多情况下，`HttpContext` 都会以方法参数的形式传入(如 `IHttpHandler.RequestProcess(HttpContext ctx)` 方法)，但也有可能在需要使用上下文时，它并未通过参数被传入。而使用 `Current` 属性，则可以随时获取对请求处理的控制权。下面举一个 `HttpContext.Current` 的例子：

```
public void DealWithRequest()
{
    HttpContext thisRequest = HttpContext.Current;
    thisRequest.Response.Write("<h3> Hello World</h3>");
}
```


正如我们从之前展示的 `HttpContext` 对象的成员所看到的，这些成员为开发者提供了以下内容：

- `Response` 对象的引用(可用于向客户端发送输出)
- `Request` 对象的引用(可用于获取有关请求本身的信息)
- 中心应用程序本身的引用(可用于获取应用程序的状态)
- 针对每个请求的字典的引用(可用于在请求的生命周期中存储数据)
- 应用程序范围缓存的引用(可用于存储数据，从而避免与数据库的频繁交互)

后面，您还会更深入地了解这个上下文对象(尤其是在介绍编写自定义的 `HttpHandler` 时)。

2.4.3.3 `HttpModule`

虽然 `Application` 对象适合小规模地处理应用程序范围的事件和数据，但应用程序范围的任务有时需要更复杂的处理。`HttpModule` 就是为此而存在的。

ASP.NET 包含许多预定义的 `HttpModule`。例如，会话状态、身份验证和授权都是由 `HttpModule` 来处理的。编写 `HttpModule` 并不复杂，非常适合完成复杂的、应用程序范围的操作。例如，如果要编写某种自定义的逻辑，并在处理每个请求之前执行，则使用 `HttpModule` 是一个不错的选择。稍后会详细介绍 `HttpModule`。

2.4.3.4 `HttpHandler`

请求在管线中的最后一站是 `HttpHandler`。任何实现 `IHttpHandler` 接口的对象都可以作为处理程序。当请求到达管线尾部，ASP.NET 会根据配置文件来将特定的文件扩展名映射到某个 `HttpHandler` 上。然后，ASP.NET 会加载这个处理程序，并调用处理程序的 `IHttpHandler.ProcessRequest` 方法来执行请求处理任务。

2.5 Visual Studio 与 ASP.NET

Visual Studio 2010 提供了多种开发期间定位网站的方法。Visual Studio 2010 的向导定义了 4 种网站项目部署位置：本地 IIS 网站、基于文件系统的网站、FTP 网站和远程网站。

下面几小节将分别介绍使用项目向导会遇到的这几种网站类型。不同类型的网站针对不同场景。相比之前的版本，Visual Studio 2010 提供的这些选项使得开发和部署 ASP.NET 应用程序更为容易。

2.5.1 本地 IIS 网站

与之前版本的 Visual Studio 一样，创建本地 IIS 网站需要指定一个本地虚拟目录。这种方

式创建的网站会使用安装在本地的 IIS。本地 IIS 网站会将页面和文件夹存储在 IIS 的默认目录结构中(即\inetpub\wwwroot)。在默认情况下，Visual Studio 会在 IIS 中创建一个虚拟目录。不过，也可以提前将虚拟目录创建好，并将网站的代码存储在任意物理文件夹中。然后，只需要将该虚拟目录引向这个物理文件夹即可。

在本地 IIS 中创建网站来测试应用程序，其目的之一可能是测试应用程序池、ISAPI 筛选器或 HTTP 身份验证这样的功能。网站可以从其他计算机访问，因此相较于在本地通过 IIS 来测试应用程序，各个方面容易得多。为创建本地网站，需要有管理员权限。对于大多数开发者来说，这不成问题。

2.5.2 基于文件系统的网站

基于文件系统的网站可位于任意指定的文件夹。该文件夹可以在本地计算机上，也可以位于其他计算机的共享文件夹中。基于文件系统的网站不要求在计算机上安装 IIS。它的页面的运行会使用 Visual Studio 的 Web 开发服务器。

Visual Studio 的 Web 开发服务器

在 Visual Studio 2005 出现以前，开发环境要直接使用 IIS 来运行页面。也就是说，为能够进行开发，开发者需要在其计算机上完整地启用 IIS。这会带来潜在的安全问题。Visual Studio 2010 包含内建的 Web 服务器。这样，即便未在计算机上安装 IIS，也可以顺利开发 Web 应用程序。

文件系统网站可用于在本地测试网站，而与 IIS 无关。最常见的方法是创建、开发并测试文件系统网站。然后，在要部署网站时，在部署服务器上创建 IIS 虚拟目录，最后将网站的页面复制到该目录下。

由于文件系统网站使用 Visual Studio 的 Web 开发服务器，而不是 IIS，因而可以在没有管理员权限的情况下开发网站。

在开发和测试应用程序特定的功能时，这种方式十分奏效。然而，由于没有 IIS 的参与，开发者无法使用(或调试)ISAPI 筛选器、应用程序池和身份验证这样的 IIS 功能(尽管在开发期间一般不需要关心此类问题)。

2.5.3 FTP 网站

除了创建基于 HTTP 的网站外，我们还可以在 Visual Studio 中通过 FTP 服务器来管理网站。例如，如果使用远程计算机来承载网站，则可以使用 FTP 服务器方便地在开发位置和宿主服务器之间传输文件。

Visual Studio 可以连接任何具有读写访问权限的 FTP 服务器。一旦连接，Visual Studio 便可以管理远程 FTP 服务器上的内容。

2.5.4 远程网站

如果要将网站放置在部署服务器上，则适合使用该选项。此外，整个开发团队可以同时开发一个网站。这种方式的弊端在于不易远程调试和配置网站，因为进程的速度较慢，而且难以从整体上管理网站。

2.5.5 Hello World 与 Visual Studio

➤ 创建 HelloWorld Web 应用程序

-
- The screenshot shows the Visual Studio .NET Framework 4.0 Web Site Wizard. The left pane, titled '最近的项目' (Recent Projects), contains 'Visual Basic' and 'Visual C#'. The right pane, titled '已安装的项目' (Installed Projects), displays a list of projects with icons, names, and types. The projects listed are:
- ASP.NET 网站 (ASP.NET Website) - Visual C#
 - ASP.NET 空网站 (ASP.NET Empty Website) - Visual C#
 - ASP.NET Dynamic Data 实体网站 (ASP.NET Dynamic Data Entity Website) - Visual C#
 - ASP.NET Dynamic Data Link to SQL 网站 (ASP.NET Dynamic Data Link to SQL Website) - Visual C#
 - WCF 服务 (WCF Service) - Visual C#
 - ASP.NET Reports 网站 (ASP.NET Reports Website) - Visual C#
 - ASP.NET Crystal Reports 网站 (ASP.NET Crystal Reports Website) - Visual C#
- The 'Web 位置 (L)' (Web Location) field at the bottom is set to 'http://localhost/ASPNETStepByStepExamples'. The '浏览 (B)...' (Browse...) button is visible next to the field.

- 为突出本示例中 HTTP 网站的不同，可以将其与本章前面基于 IIS 的示例做一下比较。从“Web 位置”下拉列表中选择 HTTP。为网站起一个有意义的名称，如 ASPNETStepByStepExamples。尽管这个目录名称已在之前的 IIS 示例中用过了，但这里还能使用，因为 Visual Studio 会在 IIS 的默认子目录 `\inetpub\wwwroot` 下新建该子目录。

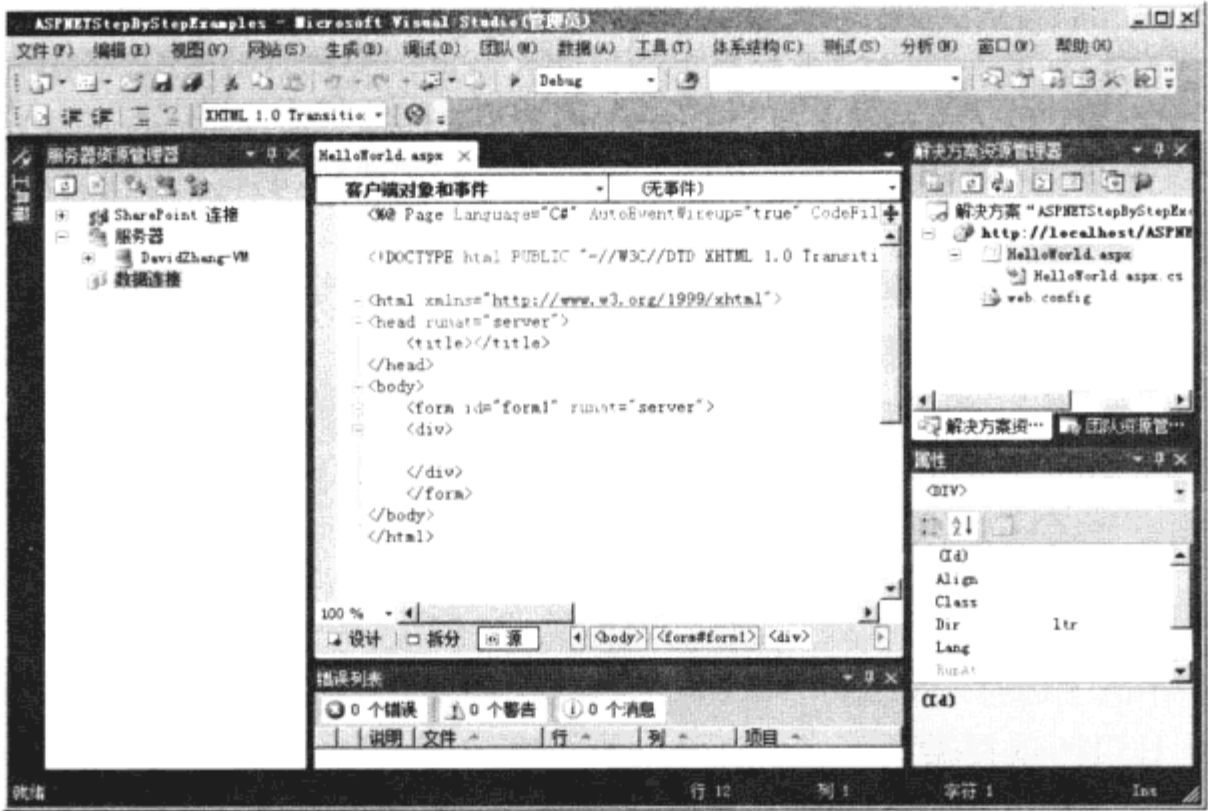
- 3. 选择项目模板^①。这个对话框包含许多网站类型。这里选择“ASP.NET 空网站”。Visual Studio 会在“我的文档”的 Visual Studio 2010\Projects 目录下生成一个 ASP.NET 解决方案文件。Visual Studio 还会在 inetpub\wwwroot 目录新建一个文件夹，并将其与 IIS 虚拟目录映射起来。然而，这个虚拟目录没有任何文件。
有一个叫做“ASP.NET 网站”的项目模板。比较而言，选择“ASP.NET 网站”会使 Visual Studio 生成与“空网站”类似的目录结构。然而，Visual Studio 会包含一个默认的 Web 窗体和源代码(default.aspx 和 default.aspx.cs)。我们还可能会获得一个用于存放网站数据的 App_Data 目录(例如，包含 ASP.NET 安全信息的数据库文件就可以存放在该目录下)。
- 4. 创建本地网站。在这个示例中，从“Web 位置”组合框中选择 HTTP。这样，网站会在本地计算机上运行。Visual Studio 的默认选项是在本地计算机的文件系统上创建网站。使用 HTTP 项目类型后，尝试访问网站客户端的请求会由 IIS 进行定向。这是学习 ASP.NET 如何与 IIS 协同工作的最佳选项，因为这样可以将网站按一个完整的系统进行对待，并在本地计算机上进行跟踪和调试。后面的示例会将重点放在 ASP.NET 的特定功能上，此时选择基于文件系统的项目更为方便。
- 5. 添加 HelloWorld 页面。依次选择“网站”|“添加新项”选项。此时会打开下图所示的“添加新项”对话框。



该对话框列出了可添加到网站中的各种文件类型。选择列表最顶端的“Web 窗体”项，然后在“名称”文本框中输入 **HelloWorld**。其他选项保持默认值。
Visual Studio 会在 HelloWorld.aspx 文件中生成一些简单的 ASP.NET 代码。
注意，Visual Studio 生成的这段代码的顶部包含一条指令，它的目的在于将 HelloWorld.aspx 与源文件 HelloWorld.aspx.cs 关联在一起(通过 Page 指令的 CodeFile 和 Inherits 属性)。该指令后面是 Visual Studio 最初生成的 HTML 文本，如下图所示。

① 译者注：这里对段落有所调整，并添加一个步骤，以适应 Visual Studio 2010 RTM 版本。

ASP.NET 4从入门到精通

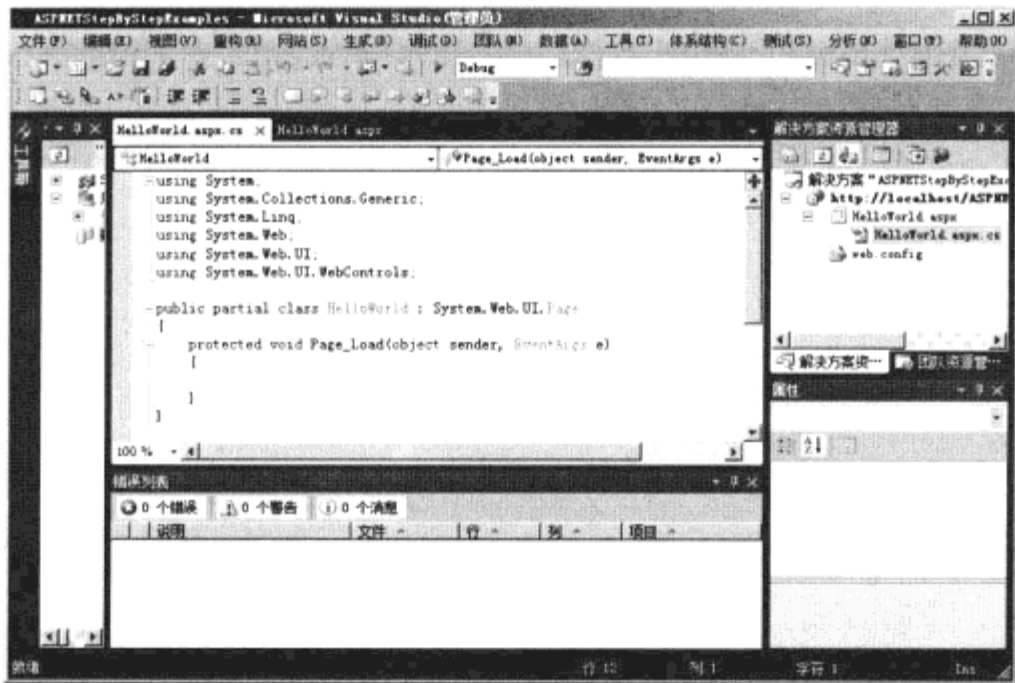


现在，让我们看一下 Visual Studio 的布局。在窗口的顶部，可以看到许多工具栏按钮和菜单选项。在本书的练习中，您会用到其中的大部分。其下方是代码窗口，该窗口包含 3 个选项卡——“设计”、“拆分”和“源”（默认会选择“源”选项卡）。如果切换到“设计”选项卡，则可以看到页面在浏览器中的样子。目前，该页面中未包含任何可见的 HTML 标签或 ASP.NET “Web 窗体”控件，因此设计视图是空的。

“源”窗口右侧是“解决方案资源管理器”，其中列出了应用程序的内容。Visual Studio 会将这些文件编译成可执行的代码库。“解决方案资源管理器”上端有一些按钮。将鼠标指针悬停在这些按钮上可以看到相应的功能说明。下图展示了在选择.aspx 文件的情况下各按钮的功能。属性、刷新、嵌套相关文件、查看代码、视图设计器、复制网站和 ASP.NET 设置。



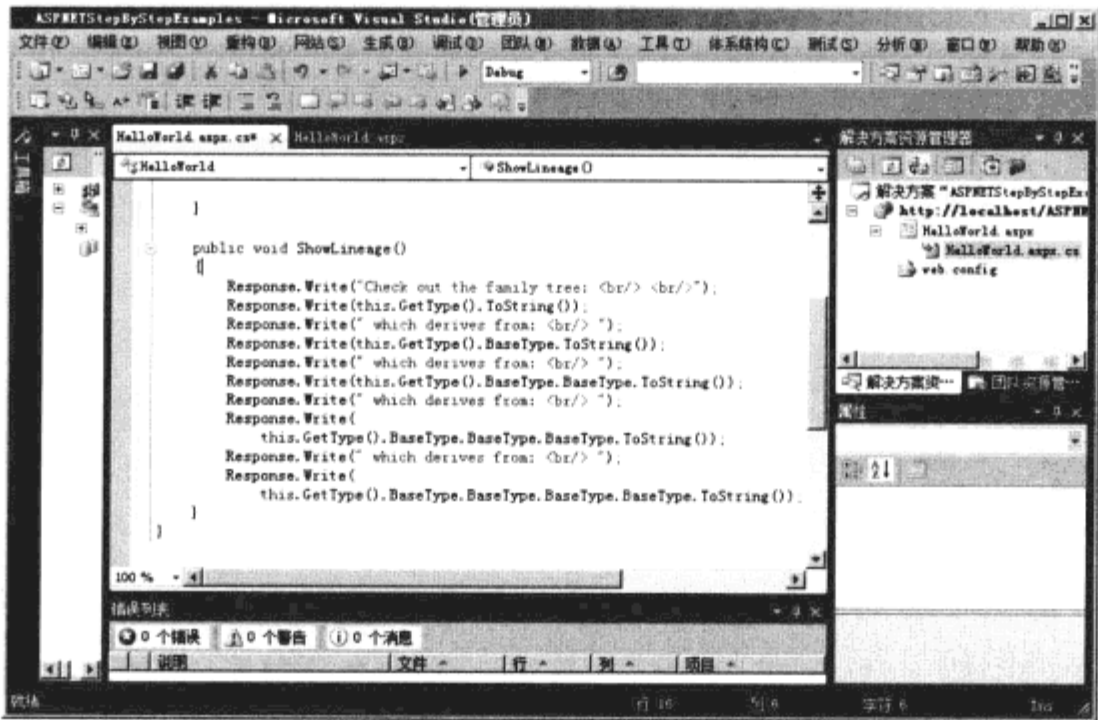
- 6. 在页面中编写代码。在“解决方案资源管理器”中选择 HelloWorld.aspx 文件，然后单击“查看代码”按钮。此时会在“源”代码窗口中显示相应的 C# 代码，如下所示。



添加以下显示页面特征的代码(与之前 HelloWorld5Code.cs 示例中的代码相同)。将其添加到 Page_Load 方法之后：

```
public void ShowLineage()  
{  
    Response.Write("Check out the family tree: <br/><br/>");  
    Response.Write(this.GetType().ToString());  
    Response.Write(" which derives from: <br/> ");  
    Response.Write(this.GetType().BaseType.ToString());  
    Response.Write(" which derives from: <br/> ");  
    Response.Write(this.GetType().BaseType.BaseType.ToString());  
    Response.Write(" which derives from: <br/> ");  
    Response.Write(  
        this.GetType().BaseType.BaseType.BaseType.ToString());  
    Response.Write(" which derives from: <br/> ");  
    Response.Write(  
        this.GetType().BaseType.BaseType.BaseType.BaseType.ToString());  
}
```

此时，HelloWorld.aspx.cs 文件的格式应如下图所示。

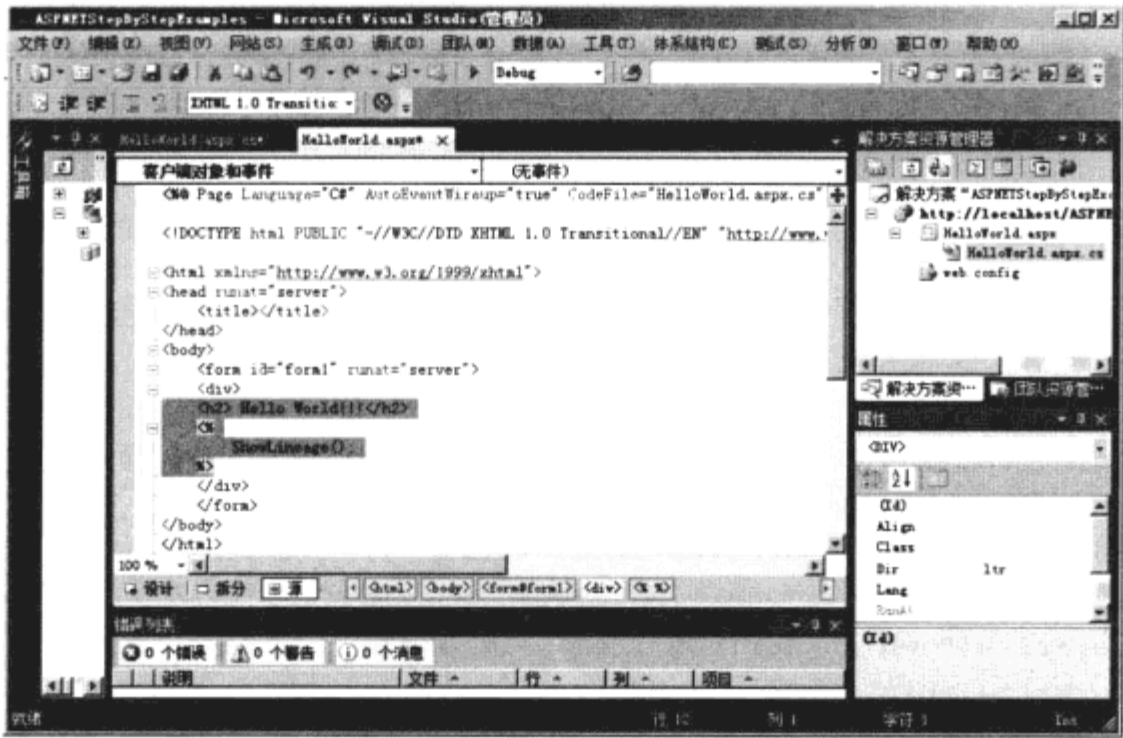


ASP.NET 4从入门到精通

7. 在.aspx 文件中调用 ShowLineage 方法。单击编辑器窗口的 HelloWorld.aspx*选项卡，返回可视编辑器。单击窗口底部的“源”选项卡。此时会显示 HTML 内容。在页面中 <div>的开标签和闭标签之间插入以下标记：

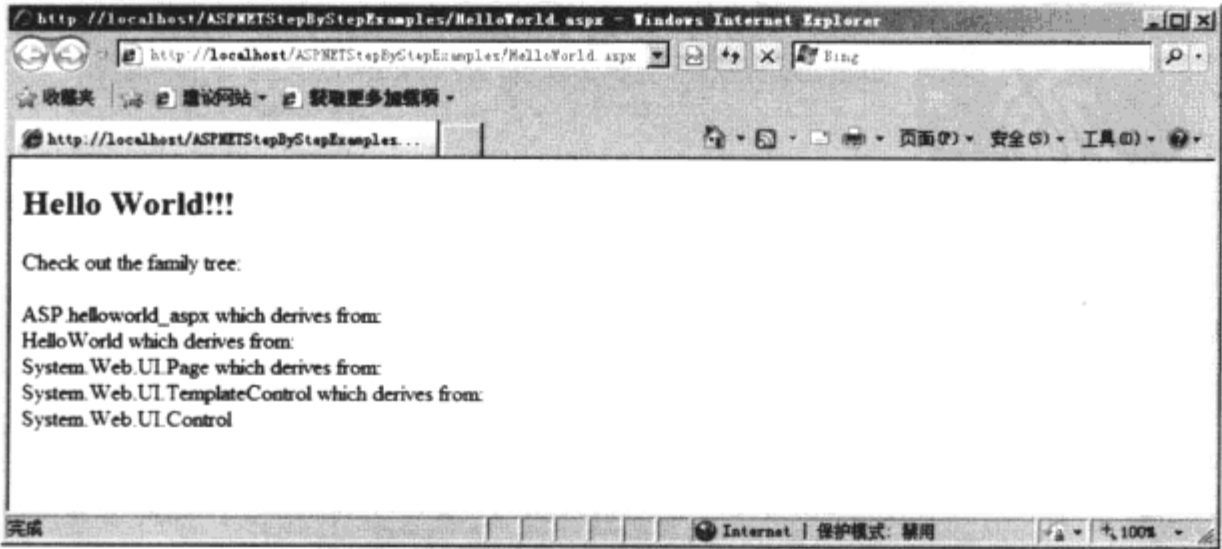
```
<h2> Hello World!!!</h2>
<%
ShowLineage();
%>
```

此时，HelloWorld.aspx 的标记应如下图所示。



8. 在 Visual Studio 中生成项目并运行网站。为生成该应用程序，在主菜单中依次选择“生成”|“生成解决方案”命令。如果代码有错误，则会在底部的“错误列表”窗口中显示出来。

为运行该应用程序，在主菜单中依次选择“调试”|“开始执行(不调试)”命令(或直接按 Ctrl+F5)。Visual Studio 会启动 Internet 浏览器(默认使用 Internet Explorer)的一个实例，并转到该页面。此时应看到下图所示的页面(确保在运行之前已通过“解决方案资源管理器”打开了 HelloWorld.aspx 页面)。



在运行该程序前，Visual Studio 会对 HelloWorld.aspx 及其代码隐藏文件

(HelloWorld.aspx.cs)进行编译，并将编译后的结果放置于临时的 ASP.NET 目录下。然后，IIS 会激活 ASP.NET HTTP 管线，通过该管线加载已编译的文件(DLL)，并呈现刚刚创建的页面。

2.6 快速参考

目 的	步 骤
在 Visual Studio 2010 中创建 FTP 网站	在主菜单中依次选择“文件” “新建” “网站”命令。在“Web 位置”组合框中选择“FTP”。该选项适合创建最终通过 FTP 协议部署的网站
在 Visual Studio 2010 中创建 HTTP 网站	在主菜单中依次选择“文件” “新建” “网站”命令。在“Web 位置”组合框中选择“HTTP”。该选项适合创建在整个开发期间将 IIS 作为 Web 服务器的网站
在 Visual Studio 2010 中创建基于文件系统的网站	在主菜单中依次选择“文件” “新建” “网站”命令。在“Web 位置”组合框中选择“文件系统”。该选项会使网站使用 Visual Studio 内建的 Web 服务器。这样可以在计算机未安装 IIS 的情况下开发网站

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 3 章

页面的呈现模型

学习目标

- 直接使用服务器端控件标签
- 在 Visual Studio 中使用“Web 窗体”和服务器端控件
- 在 Visual Studio 中使用回发事件
- 理解 ASP.NET 页面的呈现模型

本章将介绍 ASP.NET Web 窗体呈现模型的核心部分——控件。正如您将要在本章学习到的，System.Web.UI.Page 会将呈现过程拆分到小组件中去，这些组件被称为“服务器端控件”。

为理解 ASP.NET 控件模型，需要了解基础控件架构。本章将从在浏览器中呈现控件所必需的 HTML 开始讲起。然后，介绍一下传统 ASP 控件的呈现方式。虽然在您的职业生涯中可能不会使用 ASP，但通过本书了解它有助于明确 ASP.NET 的价值所在。在后续章节中您会学习到控件的自定义呈现、用户控件和标准用户界面(UI)控件，以及一些现代的、更为复杂的控件，而本章将为此奠定基础。下面让我们先了解一下 ASP.NET 的呈现模型。

3.1 将控件呈现为标签

通过之前简单的 HTML Web 窗体示例可知，开发基于 Web 的用户界面无非就是为浏览器提供正确的标签。例如，假设要使应用程序在客户端的浏览器中显示图 3.1 所示的界面。



图 3.1 有些 HTML 标签会以控件的形式呈现在 Internet Explorer 中

在客户端浏览器中显示这样的页面，意味着要使用正确的标签来填充 HTML 流，以便使浏览器通过客户端控件来呈现相应的界面。清单 3.1 给出了能够完成此项任务的 HTML。如果要自己运行这个页面，可以在本章的示例代码中找到名为 BunchOfControls.htm 的文件，将其复制到一个虚拟目录中，并浏览它。

ASP.NET 4从入门到精通

清单 3.1 BunchOfControls.htm 页面最初的 HTML 标记

```
<html>
<body>
<h2> Page in HTML </h2>
<form method="post" action="BunchOfControls.htm" id="Form1">
    <label>Type in me</label>
    <input name="textinfo" type="text" id="textinfo"/>
    <br/>
    <select name="selectitems" id="ddl">
    <option value="Item 1">Item 1</option>
    <option value="Item 2">Item 2</option>
    <option value="Item 3">Item 3</option>
    <option value="Item 4">Item 4</option>

    </select>
    <br/>
    <input type="submit" name="clickme" value="Click Me!" id="clickme"/>
</form>
</body>
</html>
```

当然,在页面上使用控件通常暗含在页面中实现动态内容,因此应该动态地(以编程方式)将这些 HTML 标记输出到浏览器。传统 ASP 也具有生成动态内容的特性。然而,传统 ASP 一般要通过原始的 HTML 来呈现其内容。例如,清单 3.1 中给出的 BunchOfControls.htm 在传统 ASP 中可能会像清单 3.2 那样写。图 3.2 展示了这个 ASP 页面在 Internet Explorer 中显示的效果。

清单 3.2 采用传统 ASP 技术的 BunchOfControls 页面

```
<%@Language="javascript"%>
<h2> Page in Classic ASP </h2>
<form>

    <label>Type in me</label>
    <input name="textinfo" type="text" id="textinfo"/>
    <br/>
    <select name="selectitems" id="ddl">
    <option value="Item 1">Item 1</option>
    <option value="Item 2">Item 2</option>
    <option value="Item 3">Item 3</option>
    <option value="Item 4">Item 4</option>

</select>
    <br/>
    <input type="submit" name="clickme" value="Click Me!" id="clickme"/>
<p>
    <% if (Request("textinfo") != "") { %>
        This was in the text box: <%=Request("textinfo") %><br/>
        And this was in the selection control: <%=Request("selectitems") %>
    <% } %>
```

```
</p>

</form>
```



图 3.2 清单 3.2 的 ASP 页面在 Internet Explorer 中的显示效果

在从选择控件中选定一项并提交后，会发现页面做出了响应——提示用户选择了哪一项。这说明 ASP 支持动态内容。

虽然传统的 ASP 提供了一种在运行时确定页面内容的方法，但仍需要通过原始的 HTML 创建许多内容。此外，控件的状态在回发后总会被重置。(稍后在介绍 ASP.NET 的 ViewState 时会讨论这个问题。)ASP.NET 在原始的 HTML 和待呈现的页面之间添加了一个中间层——该层是由 ASP.NET 服务器端控件集合提供的。相比传统 ASP 开发 Web 用户界面，ASP.NET 服务器端控件极大地降低了开发者的工作量。

3.2 将界面元素包装成组件

能够通过“组装”组件来完成用户界面是组件的最大优势。在 Windows 操作系统中，最早构建组件的方法是编写自定义的 Windows 过程，或使用控件(如列表框或按钮)的绘制功能，以及从现有窗口派生。在 20 世纪 90 年代早期，Windows 就采用了 Microsoft Visual Basic 控件(VBX)来作为当时的 UI 构建技术。当然，那已经是十多年前的事了。在 20 世纪 90 年代中期到 21 世纪的最初几年，ActiveX 控件代表了当时的图形用户界面组件化技术。而如今的富客户端应用程序 GUI 模块化标准是“Windows 窗体”。

在 20 世纪 90 年代晚期，ActiveX 还引入了一套方法，能够通过组件来呈现基于 Web 的 GUI。其思想是在页面中使用 ActiveX，当用户浏览网页时便会下载所需控件。在 20 世纪 90 年代中期也流行过使用 Java 小程序来包装通过 Web 分发的 GUI 组件。然而，这两种技术要求客户端计算机具有非常复杂的基础设施(为支持 ActiveX，需要有“组件对象模型”基础设施；为支持 Java 小程序，需要有“Java 虚拟机”)。在开发网站时，开发者不能指望客户端计算机是否有支持自己所开发 GUI 的特定架构。为支持更广泛的客户端，应只使用 HTML 来构建 GUI。这也就是说，GUI 的组件化必须在服务器端实现。

如今，现代的客户端平台变得更为统一，Web UI 开始越来越倾向于采用“异步 JavaScript 和 XML”(AJAX)编程模型。(本书的后面会介绍 AJAX 的工作原理。)AJAX 一般会使任务在浏览器的幕后执行。然而，AJAX 应用程序仍有许多呈现工作要做。ASP.NET 的 UI 组件化模型使 AJAX 应用程序的开发变得非常容易。AJAX 编程模型包含许多底层的适配代码，能够完美地适应 ASP.NET 的服务器端控件架构。

正如您在之前看到的，ASP.NET 引入了一套全新的网页管理模型。ASP.NET 的基础设施包含一个精心设计的请求处理管线。当请求到达服务器后，ASP.NET 便会实例化处理程序(IHttpHandler 的实现)来处理请求。在读到第 19 章后您会发现，这套处理架构非常灵活。开发者可以自由编写处理请求的代码。System.Web.UI.Page 类实现了 IHttpHandler，这是一种面向对象的呈现方式。也就是说，ASP.NET 生成的页面中的所有元素，都是由“服务器端控件”生成的。下面让我们看看它是如何工作的。

3.2.1 ASP.NET 页面

下面我们尝试将之前的网页更改为 ASP.NET 应用程序。这样做会用到 ASP.NET 中的一些常见功能，其中包括服务器端控件和服务器端脚本块。

1. 创建一个名为 BunchOfControls.aspx 的文件。根据第 2 章中所述方法来创建一个空白的文本文件。由于这里的所有代码都位于单个文件中，因而在这一步没有必要使用向导来创建完整的 ASP.NET 文件。
2. 在该文件中插入清单 3.3 所示代码。

清单 3.3 采用 ASP.NET 技术的 BunchOfControls 页面的源代码

```
<%@Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs ea)
    {
        ddl.Items.Add("Item 1");
        ddl.Items.Add("Item 2");
        ddl.Items.Add("Item 3");
        ddl.Items.Add("Item 4");
    }
</script>

<h2> Page in ASP.NET </h2>

<form id="Form1" runat="server">
    <asp:label text="Type in me" runat="server" />
    <asp:textbox id="textinfo" runat="server" />
    <br/>
    <asp:DropDownlist id="ddl" runat="server" />
    <br/>
```

```
<asp:Button id="clickme" Text="Click Me!" runat="server" />
</form>
```

3. 将该文件保存在某个虚拟目录中(可以创建一个新的,也可以使用在第 2 章创建的虚拟目录)。

这里仍能看到许多传统 ASP 页面元素的身影(如顶部的 Page 指令)。Language 属性是 ASP.NET 引入的。这里设置的值会告知 ASP.NET 运行库,将其遇到的代码按 C# 代码进行编译。这段代码包含一个处理 Page_Load 事件的脚本块。在该脚本块下方是一个<form>标签(HTML 标签)。应注意的是,<form>标签带有一个名为 runat 的属性,其值被设置为 server。runat=server 属性会指示 ASP.NET 运行库,在服务器上通过生成服务器端控件来处理 UI 元素。本章后面会逐步对此进行介绍。

如果为页面的控件标签添加 runat=server 属性,ASP.NET 运行库便会隐式地在内存中创建该控件的实例。结果程序集中也会包含与页面中控件的类型和名称相同的成员变量。这个示例中的 ASP.NET 代码会在服务器端运行名为 ddl 的 DropDownList(下拉列表)。如果要以编程方式访问这个 DropDownList 控件,代码块(示例中以内联方式出现)只需要使用成员变量 ddl 来引用它即可。上面的示例通过该成员变量向下拉列表插入了几个项目。

如果要以代码旁置方式访问该控件,则可以在关联的代码文件中将 DropDownList 显式地声明为 ddl。这是必要的,因为 ASP.NET 会从 System.Web.UI.Page 派生这个代码旁置类。Microsoft Visual Studio 能够自动为开发者完成这一工作(您稍后将会看到)。

在 ASP.NET 代码下面,可以看到其他元素(如标签、文本框、选择控件和按钮)也以服务器端控件的形式出现。这些控件都会在响应中添加一些 HTML。对于页面中添加的每个服务器端控件,ASP.NET 都会在内存中页面的控件树上添加一个实例。这个控件树是一个容器,包含每个服务器端控件所封装的元素。这其中也包含显示在页面顶部的标题文本,虽然没有为<h2>标签显式地添加 runat=server 属性。^①

3.2.2 页面的呈现模型

为理解 ASP.NET 页面模型的工作原理,可以在启用跟踪功能的情况下运行上一示例。(后面介绍 ASP.NET 调试功能时会详细讲解该特性。)现在只需要知道,在页面的 Trace 属性被设为 true 时,ASP.NET 会记录请求和响应的整个上下文。为此,应这样修改 Page 指令:

```
<%@Page Language="C#" Trace="true"%>
```

图 3.3 展示了启用跟踪功能后该页面的效果。

^① 译者注:ASP.NET 会将每个整段的文本看作一个服务器端控件,而不论其中包含多少 HTML 标签和客户端脚本。HTML 标签也是文本,ASP.NET 不会对其进行任何处理。



图 3.3 清单 3.3 的.aspx 文件开启跟踪后在 Internet Explorer 中呈现的效果^①

如果查看响应的原始文本(在 Internet Explorer 中依次单击“页面”|“查看源文件”), 可以看到 ASP.NET 的响应是非常简单且普通的 HTML。在顶部有一个额外的标记——__VIEWSTATE 隐藏字段, 第 4 章会详细介绍它。在该标记后面是我们熟悉的 HTML, 它描述了一个表单。

清单 3.4 给出了清单 3.3 中 ASP.NET 代码生成的原始 HTML。在浏览器页面前, 记得先将跟踪功能禁用。

清单 3.4 BunchOfControls.aspx 文件生成的原始 HTML

```
<h2> Page in ASP.NET </h2>
<form method="post" action="BunchOfControls.aspx" id="Form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJODQ1ODEz ... " />
</div>

    <span>Type in me</span>
    <input name="textinfo" type="text" id="textinfo"/>
    <br/>
    <select name="ddl" id="ddl">
    <option value="Item 1">Item 1</option>
    <option value="Item 2">Item 2</option>
    <option value="Item 3">Item 3</option>
    <option value="Item 4">Item 4</option>

</select>
    <br/>
    <input type="submit" name="clickme" value="Click Me!" id="clickme"/>
</form>
```

① 译者注：这里对原书插图作了修正，增加了页面跟踪。

在所呈现的页面中，我们看不到任何 `runat=server` 属性。这是因为 `runat=server` 属性是针对 ASP.NET 的(旨在指示其如何构造页面的控件树)。

3.2.3 页面的控件树

在将页面的 `Trace` 属性设置为 `true` 后，ASP.NET 会在页面的跟踪表单中生成大量信息。浏览跟踪信息，我们会看到 ASP.NET 页面跟踪中包含的控件树部分。图 3.4 展示了 `BunchOfControls.aspx` 页面跟踪信息中有关控件树的部分。

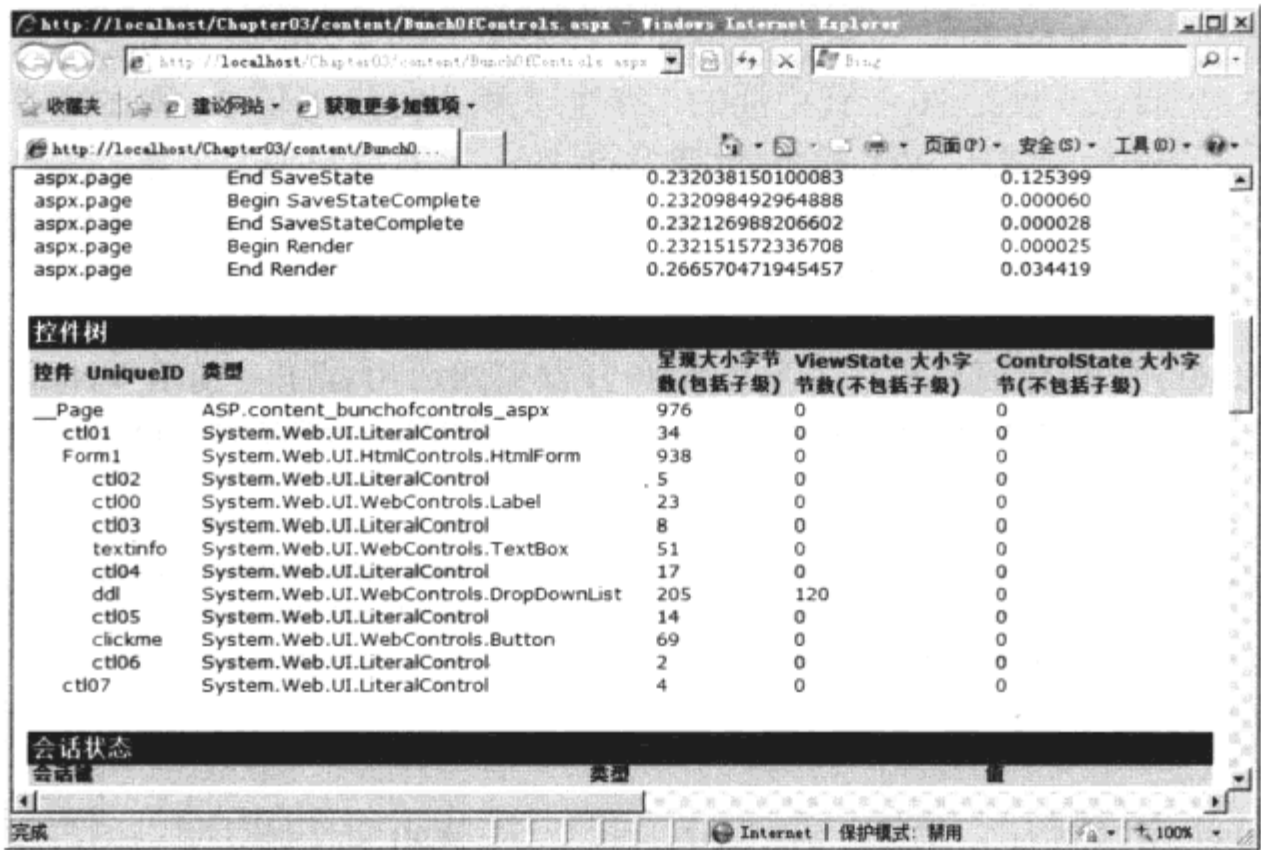


图 3.4 BunchOfControls.aspx 页面跟踪中的控件树信息

跟踪信息中页面控件树的第一行是一个名为 `__Page` 的项。这实际上代表着在内存中运行的 `System.Web.UI.Page` 对象。在它下面是其他控件。我们可以辨别出其中的一些名称，因为那是我们在 ASP.NET 源代码中指定的，其中包括 `Form1`、`textinfo` 和 `clickme`。这些名称都来自于 .aspx 源文件中的标记。

ASP.NET 将页面呈现架构划分为更小、更易于管理的组件。图 3.4 的控件树中的每一项都派生自 `System.Web.UI.Control` 类。每当 `System.Web.UI.Page` 需要呈现页面时，它会遍历控件树，并让每个控件呈现自身。例如，当 ASP.NET 运行库让服务器端控件 `TextBox` 呈现自身时，该控件会在为浏览器提供的输出流中生成以下 HTML：

```
<input name="textinfo" type="text" id="textinfo"/>
```

其他控件也是一样。例如，`DropDownList` 会生成 `<select>` 和 `<option>` 标签(`<option>` 标签代表 `DropDownList` 控件的项目集合)。

```
<select name="ddl" id="ddl">
  <option value="Item 1">Item 1</option>
```



```
<option value="Item 2">Item 2</option>
<option value="Item 3">Item 3</option>
<option value="Item 4">Item 4</option>
</select>
```

现在，您应该已经清楚这些标签的工作方式了。在下面一节中，让我们看看如何在 Visual Studio 中管理这些标签。

3.3 使用 Visual Studio 添加控件

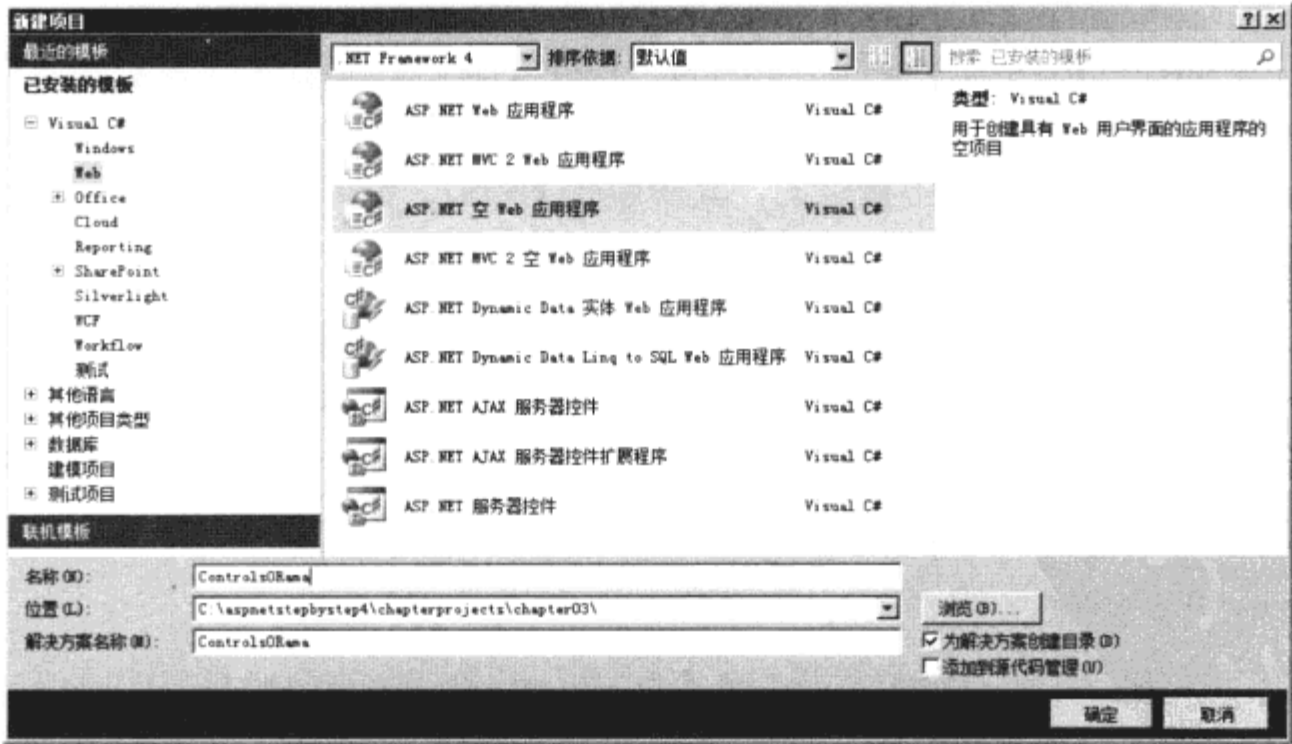
Visual Studio(与 ASP.NET)对 Web 开发的底层技术做了包装。正如您在之前章节看到的，Web 开发要追溯到 20 世纪 70 年代中期的“终端-主机”时代。不过，现在终端变成了复杂的浏览器，计算平台变成了 Web 服务器(或 Web 场)，并且用户遍及全球。客户端浏览器与服务器进行一次交互，客户端收到的只是服务器状态的一个快照。这是因为构建 Web 用户界面的标记语言是以非连接协议为基础的。

在 Visual Studio 中构建应用程序非常像是在开发桌面应用程序。使用 Visual Studio，开发者不必花大量时间录入 ASP 风格的代码。“设计器”是以可视方式设计 Web UI 的理想工具。

为学习如何使用 Visual Studio，下面我们使用服务器端控件来开发一个简单的页面。该页面看起来与前面给出的示例页面差不多。

➤ 使用 Visual Studio 构建一个页面

- 1. 在 Visual Studio 中，新建一个“ASP.NET 空 Web 应用程序”。将该项目命名为 ControlsORama，如下图所示。



- 2. 添加默认页面。在项目节点上右击，依次选择“添加”|“新建项”命令。在“添加新项”对话框列出的模板中，选择“Web 窗体”，将其命名为 Default.aspx，并单击“确

定”按钮。Visual Studio 会打开 Default.aspx 标记的编辑窗口。单击编辑框口底部的“设计”选项卡，将当前视图切换至“设计”视图(见下图)。

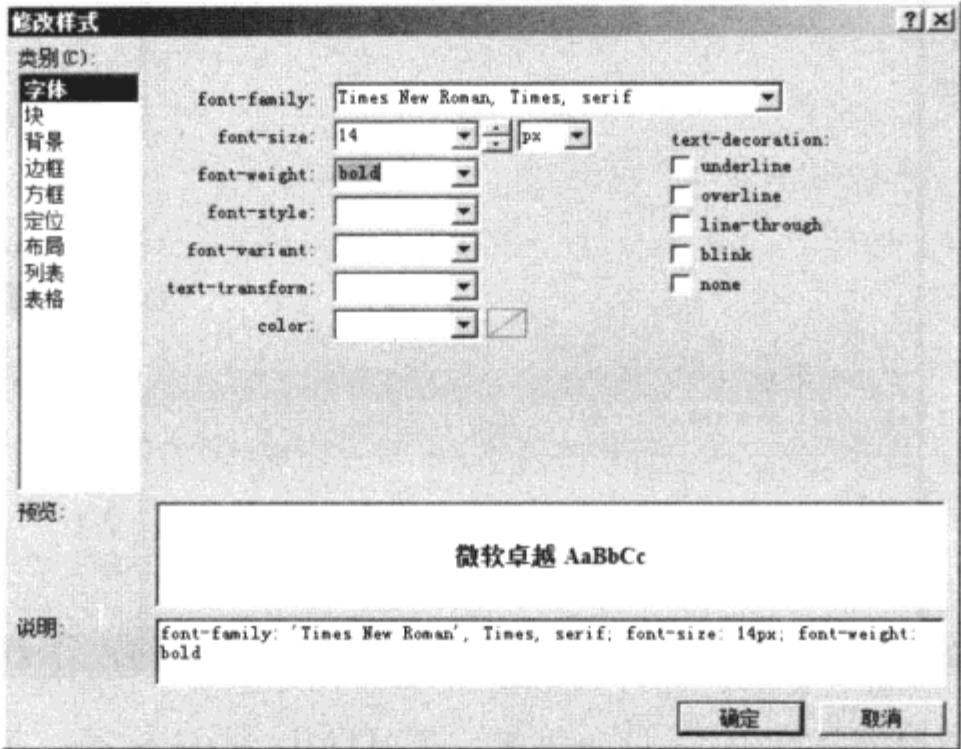


Visual Studio 生成的 ASP.NET 页面包含一个 HTML <div> 标签。在“设计器”中修改页面元素后，为查看 Visual Studio 生成的代码，可以单击设计窗口底部的“源”选项卡。Visual Studio 还提供了一个贴心的“拆分”视图，可以同时看到“设计”和“源”视图的内容。

在“设计”视图输入的文本会显示在页面顶部。单击虚线框的内部，然后输入 **Page in Visual Studio**。下图展示了插入文本后的“设计”视图。



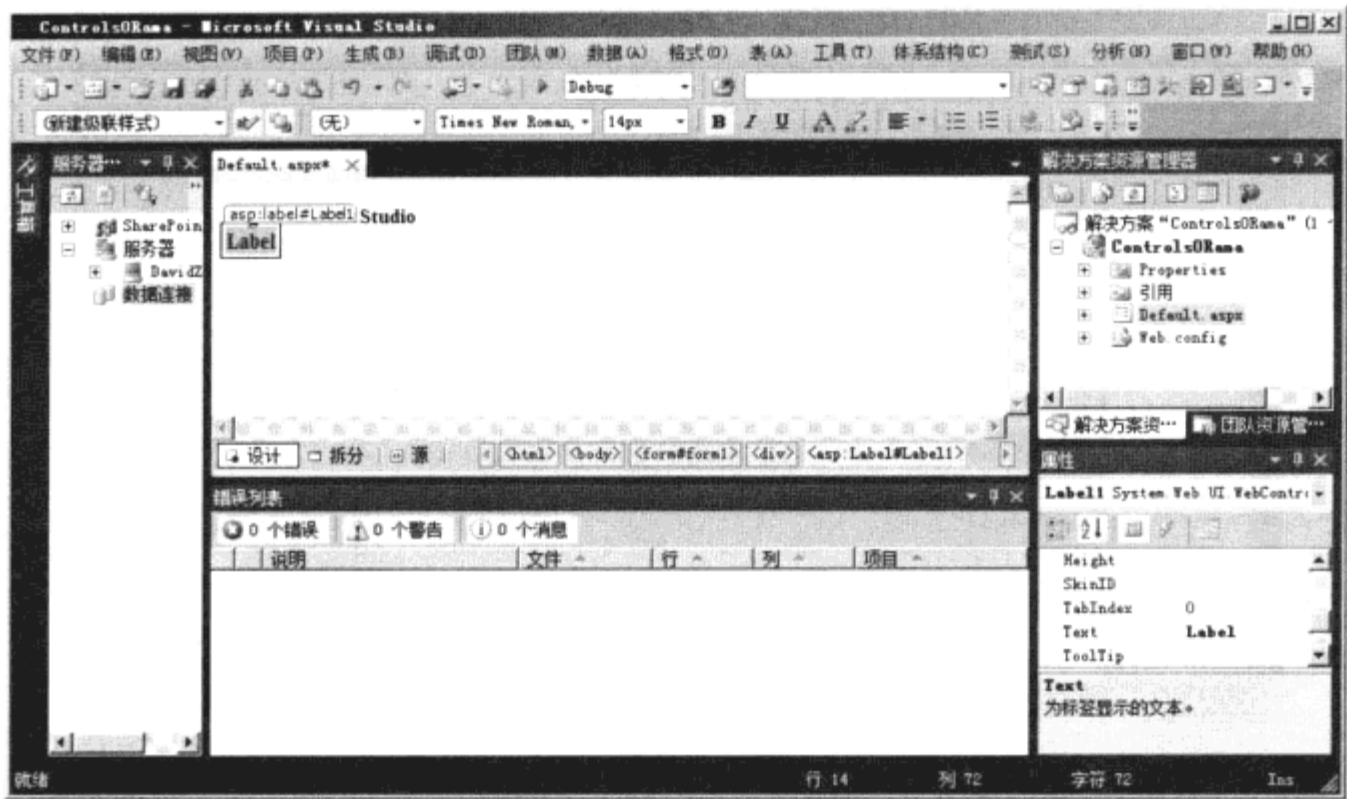
3. 编辑页面上文本的格式。为此，我们需要查看页面的属性。将文本选中，并在之上右击，选择“属性”命令。在“属性”窗口中选择 Style 属性。此时，对应的值字段会显示一个省略号按钮(⋮)。单击该按钮会打开“修改样式”对话框。在这个对话框中，我们可以修改<div>的属性，其中包括字体样式和大小。按下图所示设置 font-family、font-size 和 font-weight 的值，并单击“确定”按钮。



4. 单击 Visual Studio 左侧的“工具箱”选项卡来打开“工具箱”窗口，如下图所示。



5. 插入一个换行标记(
), 从“工具箱”中拖放一个 Label 控件到页面上，然后按下图所示选择它。(Visual Studio 的“设计器”会在该标签的上端添加一个小标记，这有助于在开发者选择它时在“设计器”中辨别它。)



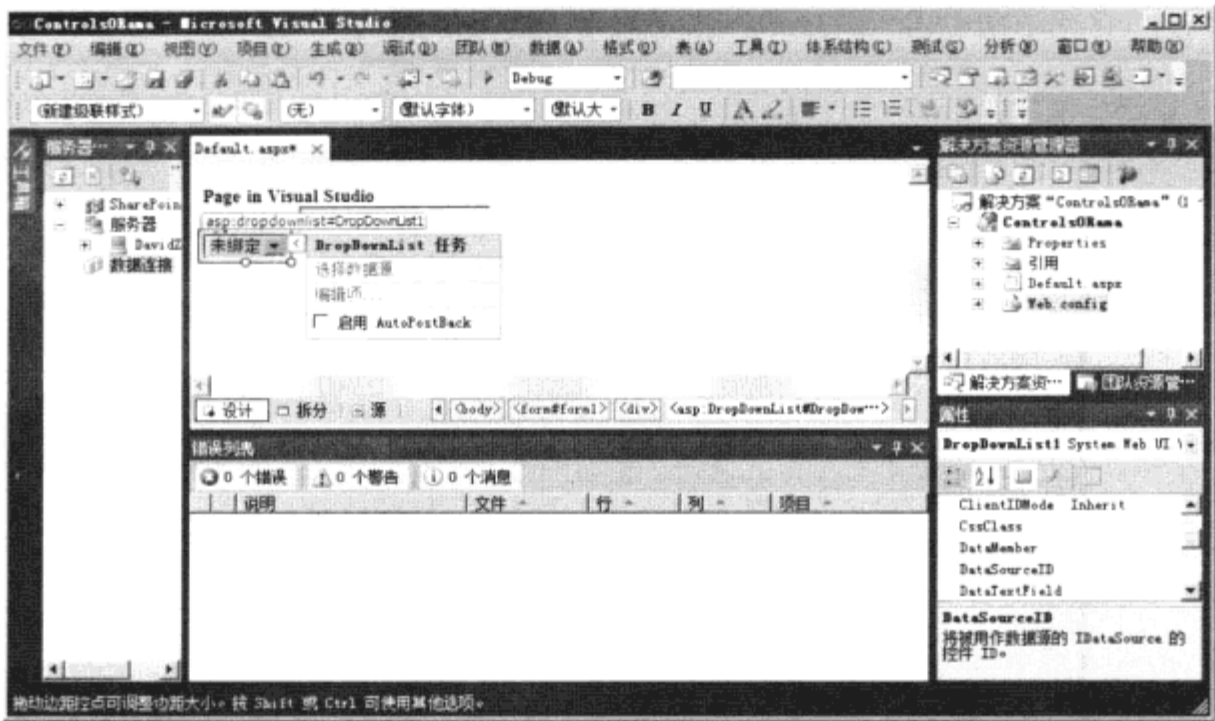
- 6. 查看该标签控件的属性并编辑其内容。如果“属性”窗口未被显示，可以在该标签上右击，然后从快捷菜单中选择“属性”命令。该控件的“属性”窗口如下图所示。



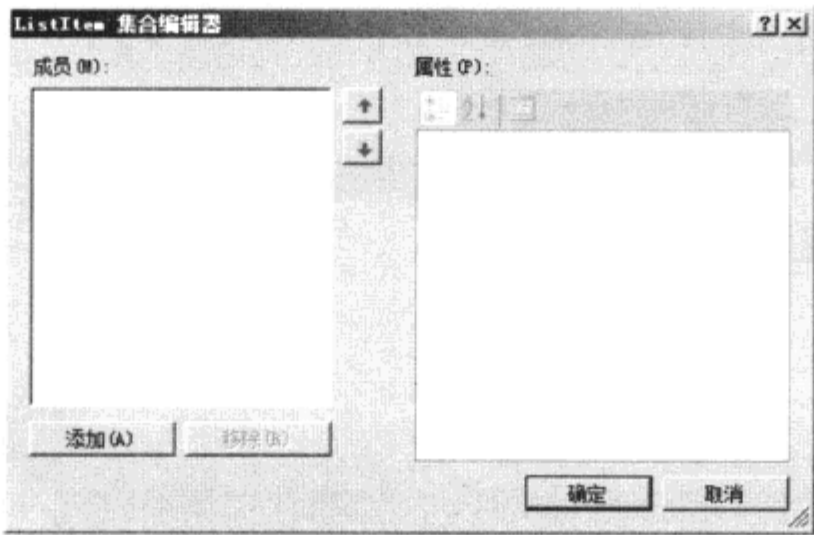
读者可以随意修改该标签的外观。本示例的标签控件使用的是小号的 Times New Roman 字体，其文本内容为 **Type in me:**。

- 7. 从“工具箱”拖放一个 TextBox 控件到页面上，将其置于 Label 控件旁边。在这个 TextBox 后面，插入一个换行标签(
)。
- 8. 接下来，从“工具箱”拖放一个 DropDownList 到页面上。下图展示了这个下拉列表控件在“设计器”中的样式。

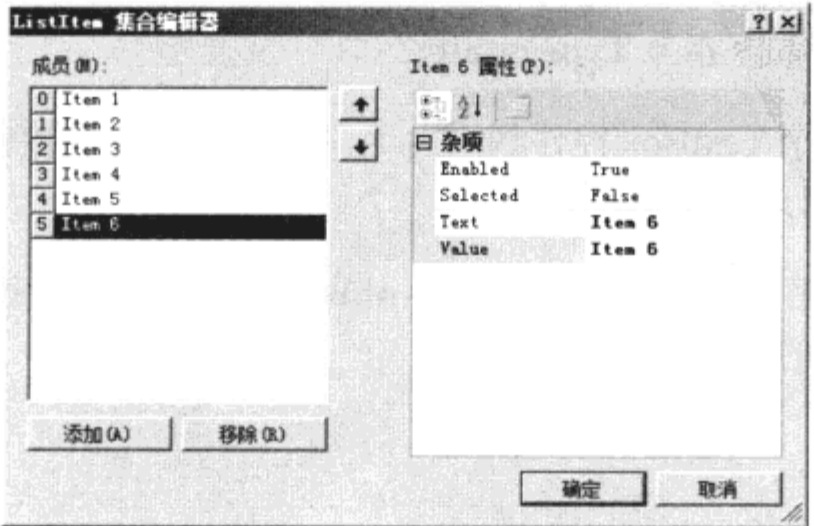
ASP.NET 4从入门到精通



在将该控件拖放到页面上后，Visual Studio 便允许我们通过如图所示的快捷菜单来为 DropDownList 添加项。单击“DropDownList 任务”菜单上的“编辑项”选项。此时会打开“ListItem 集合编辑器”，如下图所示。



每次单击“添加”按钮，“ListItem 集合编辑器”都会向 DropDownList 的集合中添加一个新项。在添加并选择一项后，可以对其显示的名称(Text 属性)进行编辑。也可以添加一个与文本对应的值。例如，在库存跟踪应用程序中，可以将 Text 属性设置为产品名，而将 Value 属性设置为企业内部的产品编码。这两个属性都可以在运行时获取。按下图所示在这个 DropDownList 中添加一些项目。添加完毕后单击“确定”按钮。



9. 在页面中添加一个按钮。首先，在 DropDownList 控件后面添加一个换行标签(
)。然后，从“工具箱”拖放一个 Button 到页面上。此时的页面如下图所示。



修改按钮的 Text 属性，为其添加一个有意义的描述。

在继续构建这个页面之前，让我们先来看一下 Visual Studio 生成的源代码。Visual Studio 会在代码中为 Label、TextBox、DropDownList 和 Button 控件分别添加一个成员变量(通过控件标记中的 runat=server 便可以看出)。此时，.aspx 文件的内容(从<form> 标签开始)应与清单 3.5 所列代码类似。

清单 3.5 Default.aspx 文件最终的标记

```
<form id="form1" runat="server">
<div style=
    "font-family: 'Times New Roman', Times, serif; font-size: 14pt; font-weight: bold">
    Page in Visual Studio<br/>
    <asp:Label ID="Label1" runat="server" Text="Type in me:"></asp:Label>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <br/>
    <asp:DropDownList ID="DropDownList1" runat="server">
        <asp:ListItem>Item 1</asp:ListItem>
        <asp:ListItem>Item 2</asp:ListItem>
        <asp:ListItem>Item 3</asp:ListItem>
        <asp:ListItem>Item 4</asp:ListItem>
        <asp:ListItem>Item 5</asp:ListItem>
        <asp:ListItem>Item 6</asp:ListItem>
    </asp:DropDownList>
    <br/>
    <asp:Button ID="Button1" runat="server" Text="Click Me"/>
    </div>
</form>
```

需要注意的是，所有在服务器端运行的 ASP.NET 标签都有 ID 属性。这是在运行时区分不同控件的标识符。我们稍后会用到它。

10. 最后，为使按钮真正发挥作用，需要在页面中为其添加事件处理程序，以便在该按钮被单击后作出响应。添加事件处理程序最简单的方法是在“设计”视图双击该按钮。Visual Studio 会为按钮的单击事件生成一个处理程序，并显示在“源”视图中。此时便可以添加响应按钮单击事件的代码了。
11. 在按钮的事件处理程序中添加清单 3.6 所示代码。

清单 3.6 按钮的事件处理程序

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write("Hello. Here's what you typed into the text box: <br/>");
    Response.Write(this.TextBox1.Text);

    Response.Write("<br/>");
    Response.Write("And the selected item is: <br/>");
    Response.Write(this.DropDownList1.SelectedItem.Text);
}
```

这段响应按钮单击事件的代码使用 `Response` 对象向输出流发送了一些文本。`Response.Write` 的输出是客户端浏览器最先读取的文本，因此这些文本会显示在页面的顶端。

需要注意的是，这段处理程序使用了页面类中的 `TextBox1` 成员变量。这说明我们能够以编程方式在运行时调用控件。下图展示了该页面在浏览器中的效果。注意，`Response.Write` 输出的文本会被插入到所有控件之前。



为测试页面上的控件，可以在“调试”菜单中选择“开始执行(不调试)”。为查看所有服务器端控件生成的 HTML，可以查看发送给浏览器的源文件。(如果使用 Internet Explorer，则依次在命令栏中单击“页面”|“查看源文件”命令)打开源文件后，应看到清单 3.7 所示的代码。注意，`Response.Write` 输出的文本位于输出流的最顶端。

清单 3.7 运行 Default.aspx 后得到的 HTML

```
Hello. Here's what you typed into the text box: <br/>Hello world<br/>
And the selected item is: <br/>Item 4
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" id="form1">
<div>
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMTc0ODY5OTM3N2RkvlciIj/EgelyYjZ7ti5liSLFZ6g="/>
</div>
<div>
    <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
value="/wEWCQLi6ca/AgLs0bLrBgKTmtqTCAKTmsaTCAGBhPVvUl6OSn5X9GuTKtorQFlaF3r"/>
</div>
<div style="font-family: 'Times New Roman', Times, serif;
font-size: 14pt; font-weight: bold">
Page in Visual Studio<br/>
<span id="Label1">Type in me:</span>
<input name="TextBox1" type="text" value="Hello world" id="TextBox1"/>
    <br/>
    <select name="DropDownList1" id="DropDownList1">
        <option value="Item 1">Item 1</option>
        <option value="Item 2">Item 2</option>
        <option value="Item 3">Item 3</option>
        <option selected="selected" value="Item 4">Item 4</option>
        <option value="Item 5">Item 5</option>
        <option value="Item 6">Item 6</option>
    </select>
    <br/>
    <input type="submit" name="Button1" value="Click Me" id="Button1"/>
</div>
</form>
</body>
</html>

```

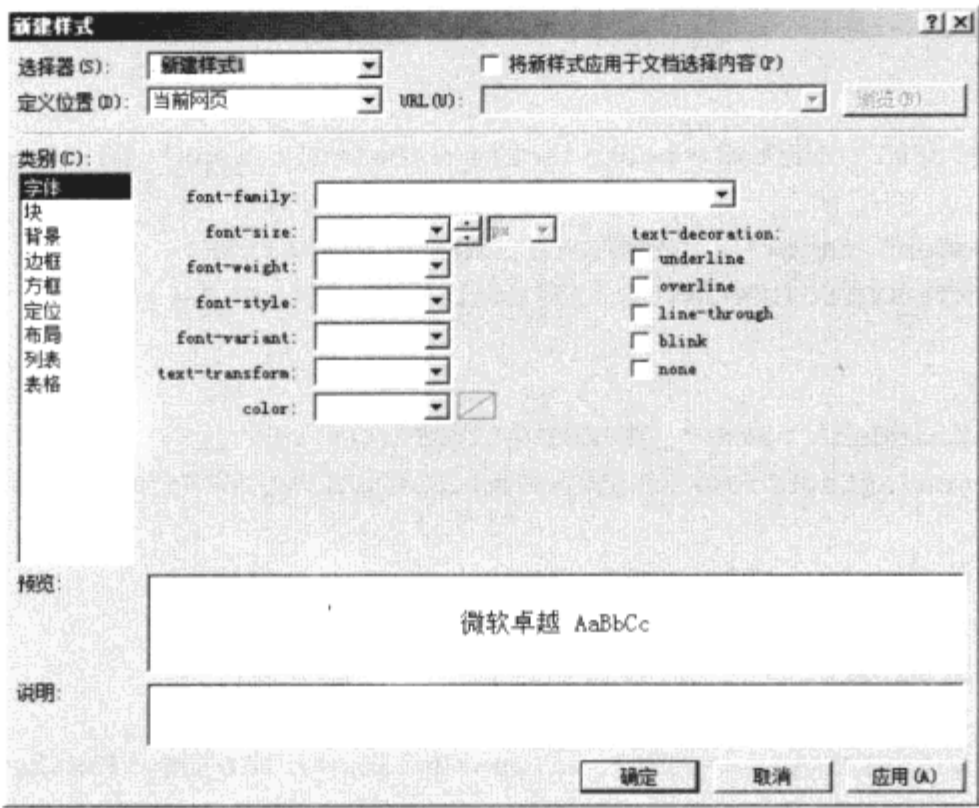
注意，浏览器所显示的是纯 HTML——是 ASP.NET 通过页面的呈现模型生成的，但浏览器不会对此有察觉。

布局技巧

在构建上一页面时，您可能会发现它采用的是流布局。即每次将控件拖放到页面上，“设计器”总会将其置于其他控件的后面。某些版本的 Visual Studio 采用的默认行为与之不同。Visual Studio 2003 对页面上的元素采用的是绝对定位(就像在开发富客户端或标准的 Windows 应用程序一样)。

虽然 Visual Studio 2010 没有在“设计器”直接提供设置定位方式的选项，但可以通过应用

于整个页面或页面中某个元素的样式来进行设置。为对页面添加新的样式，请确保当前窗口显示的是“设计器”，然后在 Visual Studio 的主菜单中依次单击“格式”|“新建样式”命令。此时会打开“新建样式”对话框(见下图)。



为通过样式来设置布局，在对话框左侧的“类别”列表中选择“定位”。请注意设置定位方式的组合框。在这里设置的样式可以应用于页面上的多个元素，只要这些元素通过 `CssClass` 属性引用样式类别名称即可。

不同的定位设置会使页面产生不同的布局方式。为理解不同定位方式，可以在此尝试一下。第 7 章中讨论母版页时会更深入地探讨有关样式的话题。

3.4 快速参考

目 标	操 作
在 ASPX 的“源”视图和“设计”视图之间切换	切换到“设计”或“源”选项卡，它们通常显示在编辑器窗口的左下角。也可以使用“拆分”选项卡来同时显示“源”和“设计”视图
在页面中添加服务器端控件	打开“工具箱”。如果该窗口没有被显示，可以在主菜单中依次选择“视图” “工具箱”选项。(也可以使用组合键 <code>Ctrl+W, X</code> 。①) 在“工具箱”中单击该控件，并将其拖放到页面上
修改页面上控件的属性	确保页面设计器处于“设计”模式。选中要编辑属性的控件。在“属性”窗口中修改目标的属性

① 译者注：Ctrl+W, X 表示按住 Ctrl 键不放，依次按 W 和 X 键。

续表

目 标	操 作
启用跟踪	在“源”代码编辑模式下，为 Page 指令添加 Trace= “true” 属性。 或者 在“属性”窗口顶部选择 DOCUMENT 元素，将 Trace 属性设置为 true
修改服务器端控件的大小	选中该控件，用鼠标拖动控件边框上的手柄，直到将控件调整到合适的大小
为控件的默认事件添加处理程序	双击要添加事件处理程序的控件。此时会显示代码编辑器，可以在此为该事件的处理程序添加代码
修改页面的布局方式	在主菜单上依次单击“格式” “新建样式”命令。在“类别”中选择“布局”，在右侧定义样式(除布局外，还可以定义字体样式和页边空白等属性)。该样式可以应用到整个页面，也可以应用到单个元素

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 4 章

自定义控件

学习目标

- 在现有的 Microsoft Visual Studio 解决方案中添加新项目
- 创建能够呈现自定义 HTML 的服务器端控件
- 在 Visual Studio 的“工具箱”中添加服务器控件
- 为“Web 窗体”添加服务器端控件
- 管理控件的事件
- 通过 ASP.NET 来检测不同客户端浏览器的差异，并利用这种信息

第 3 章介绍了 ASP.NET 呈现模型幕后的基础架构。System.Web.UI.Page 管理着一系列服务器端控件；每个控件的任务是呈现页面的特定区域。ASP.NET 将服务器端控件分为两大类：

- 自定义控件(可以完全控制这种控件的呈现过程)
- 复合控件(由多个服务器端控件组合而成的一个单元)

本章将重点介绍第一类控件——自定义控件。在本章中，您将会学习到这种控件在网页中的工作方式。所涉及的话题还包括控件事件的管理和客户端浏览器差异的检测。

首先，让我们了解一下 ASP.NET 服务器端控件架构的核心部分——System.Web.UI.Control 类。

4.1 Control 类

ASP.NET 服务器端控件都派生自 System.Web.UI.Control 类。事实上，Control 类是 ASP.NET 用户界面中最核心的部分，甚至连 System.Web.UI.Page 本身也派生自 Control 类。表 4.1 列出了 System.Web.UI.Page 类的主要成员。

表 4.1 Page 类主要的属性、方法和事件

成 员	说 明
Application	当前请求的 HttpApplicationState 对象的引用
Cache	应用程序缓存的引用，缓存即内存中应用程序范围的状态字典(通常用于优化)
Controls	Page 维护的控件集合
CreateChildControls	页面创建其控件树时调用的虚方法
Init	指示页面已被初始化的事件

续表

成 员	说 明
IsPostBack	用于判断当前请求是新的请求或回发的属性
Load	指示页面已加载的事件
RenderControl	用于呈现页面内容的虚方法
Request	引用的是一个有状态对象，代表传入的请求
Response	引用的是一个有状态对象，代表传出的响应
Session	引用的是一个有状态对象，其中包含针对当前请求的信息
Unload	指示页面已被卸载的事件

表 4.1 所列成员有一小部分来自于 System.Web.UI.Control。后面探究 ASP.NET “Web 窗体” 时会用到所有这些成员。上一章介绍过，ASP.NET Web 窗体在内部结构中维护着控件的集合。当页面要将其内容呈现到客户端时，System.Web.UI.Page 会对这个控件集合进行迭代，并让每个控件呈现自身。如果遇到包含子控件的控件(就像页面包含控件一样)，ASP.NET 也会继续对深入这个子控件集合进行迭代。^①表 4.1 中有一个叫 RenderControl^②的方法。该方法只接受一个 HtmlTextWriter 类型的参数。稍后会介绍这个类，现在只需将其看作将页面的响应发送回客户端的管道。

Control 类还包含以下成员：

- 用于管理控件视图状态的属性
- 用于管理皮肤(skin)的属性(用于使同一网站的不同页面获得一致的外观)
- 用于获取父控件(如果是复合控件的话)或父页面的属性
- Init、Load、PreRender 和 Unload 事件
- Init、Load、PreRender 和 Unload 事件的方法
- 管理子控件的方法

本章将论述开发自定义控件和复合控件所涉及的重要内容。Page 类是可扩展的，其逻辑简单，非常易用。下面让我们先从如何构建自定义控件开始。

4.2 Visual Studio 与自定义控件

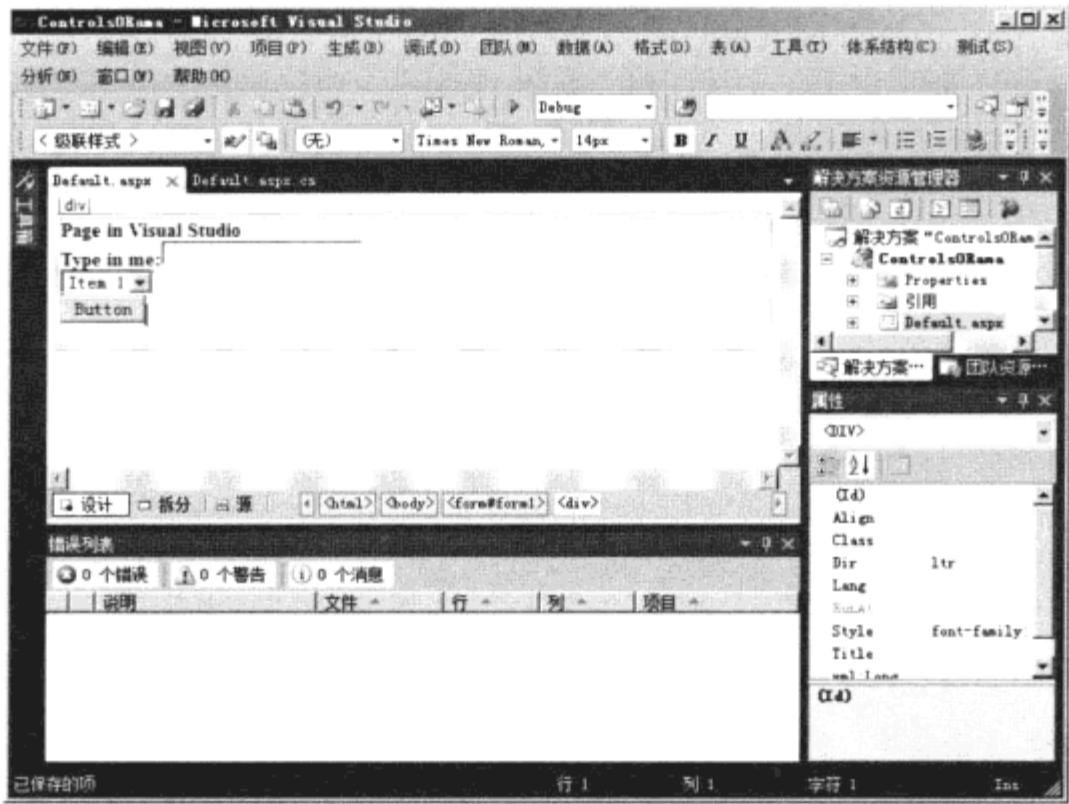
在本节中，我们将构建一个简单的控件(Visual Studio 自动生成的默认控件)，看看它是如何呈现在 Web 窗体中的。Visual Studio 会创建一个只包含 Text 属性的简单控件，该控件会将

^① 译者注：迭代过程是递归的，ASP.NET 会在运行时对控件树中的所有控件进行遍历，直到所有控件都被呈现。
^② 译者注：这里将原书的“RenderContents”方法更正成“RenderControl”方法。

其 Text 属性的文本呈现给浏览器。为探究服务器端控件工作方式，这是一个不错的方法。

➤ 创建自定义控件

- 1. 打开第 3 章创建的 ControlsORama 项目，如下图所示。注意，如果希望保存该项目当前的状态，则可复制整个项目目录，然后对副本进行操作。本章的其他示例也将在该项目的基础之上构建，下一章也如此，只不过使用另一种控件开发方法。



- 2. 在“解决方案资源管理器”窗口的“解决方案‘ControlsORama’”节点上单击右键，在快捷菜单中依次选择“添加”|“新建项目”。^①将新项目命名为 CustomControlLib。选择“ASP.NET 服务器控件”模板，如下图所示。



Visual Studio 会自动添加一个名为 ServerControl1 的简单控件。清单 4.1 展示了 Visual Studio 为 Web 控件库生成的默认代码。

^① 译者注：如果要在现有的控件库中添加其他服务器端控件，可以使用“添加新项”窗口，而不必添加一个完整的项目。

清单 4.1 自定义控件的默认实现

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CustomControlLib
{
    [DefaultProperty("Text")]
    [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1>")]

    public class ServerControl1 : WebControl
    {
        [Bindable(true)]
        [Category("Appearance")]
        [DefaultValue("")]
        [Localizable(true)]
        public string Text
        {
            get
            {
                String s = (String)ViewState["Text"];
                return ((s == null) ? "[" + this.ID + "]" : s);
            }
            set
            {
                ViewState["Text"] = value;
            }
        }

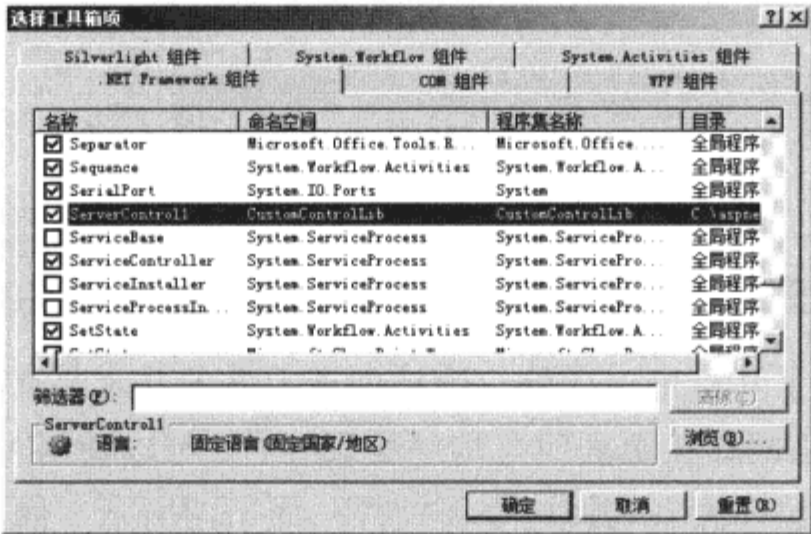
        protected override void RenderContents(HtmlTextWriter output)
        {
            output.Write(Text);
        }
    }
}
```

Visual Studio 生成的这段代码包含一个派生自 `System.Web.UI.WebControl.WebControl` (派生自标准的 `Control` 类) 的类，并添加了一些标准属性。注意，该控件包含一个名为 `Text` 的属性，并重写了 `WebControl` 类的 `RenderContents` 方法。这是一个能够实际工作的控件(虽然其功能只相当于 `Label`)。

3. 在主菜单中依次选择“生成”|“生成解决方案”，生成该项目。
4. 在编译过该控件后，Visual Studio 便自动将其添加到“工具箱”中，供主项目使用。另外，也可以通过另一种方法手动将该控件添加到“工具箱”中：在“工具”菜单中，

单击“选择工具箱项”。在“选择工具箱项”对话框中，单击“浏览”按钮，导航到 ControlsORama 项目的目录，然后进入 CustomControlLib 目录。随后，打开 Bin\Debug 目录。(Visual Studio 默认会生成 Debug 版本。)选择 CustomControlLib.DLL 程序集，然后单击“打开”按钮。

此时，SeverControl1 应显示在“选择工具箱项”对话框中，并且相应的复选框已被选中(见下图)。



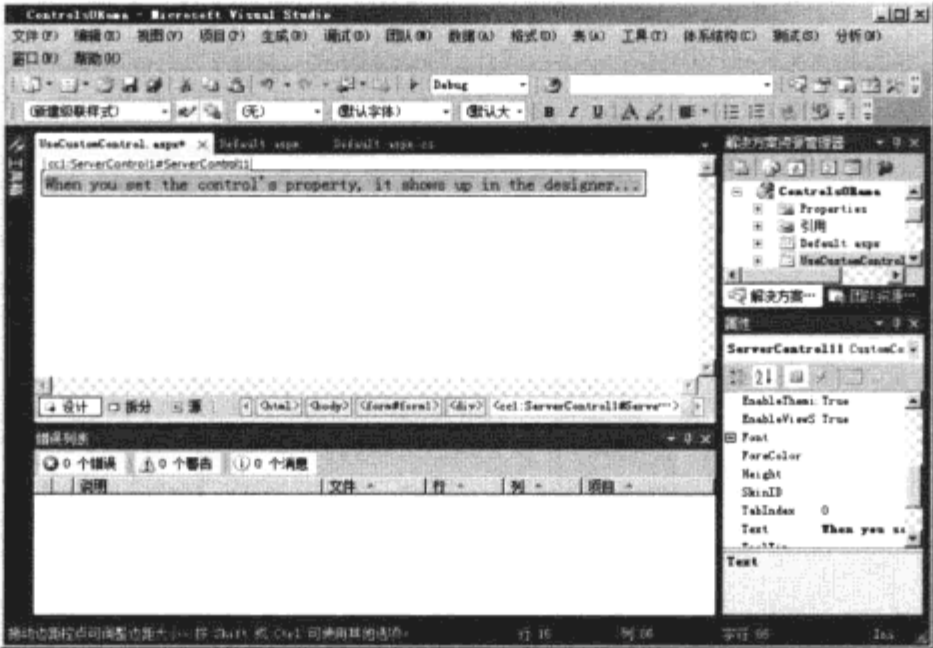
在“选择工具箱项”对话框中单击“确定”按钮后，ServerControl1 控件便会显示在“工具箱”中。为更容易查找该控件，可在“工具箱”上单击右键，单击“按字母属性排序”。此时，ServerControl1 会像下图这样显示在“工具箱”中。



5. 为了解该控件的工作方式，需要将其添加到页面上。下面我们为当前网站添加一个新页面。在“解决方案资源管理器”中选择 ControlsORama 项目。在主菜单中依次选择“项目”|“添加新项”，然后添加一个“Web 窗体”。将其命名为 UseCustomControl.aspx。为将该控件置于页面上，先切换至“设计”视图。从“工具箱”拖放一个 ServerControl1 到 UseCustomControl 页面的“设计”视图。此时，我们会发现设计界面的左上角出现了一个矩形框(见下图)。



控件中的文本表明了其身份。此外，该控件正上方的文本是其类名。虽然该控件已经被选中，但稍后在来回切换页面时可能还要选择它。可以使用 Visual Studio 右下角“属性”窗口顶端的下拉列表来选择这个新控件。如下图所示，修改该控件的 Text 属性，并观察其在“设计器”中的变化。Text 属性会被该控件呈现出来。可以在此处修改其值。



再读一下该控件的源代码。特别注意一下 RenderContents 方法。该方法简单地通过一个参数(HtmlTextWriter)来将 Text 属性的内容发送给浏览器。这就是在“设计器”中修改该属性后其值会被显示出来的原因。

下面是在.aspx 文件中 Visual Studio 为包含该控件而添加的代码。为实际查看此代码，可在 Visual Studio 的代码窗口底部切换到“源”选项卡。这条 Register 指令用于告知 ASP.NET 运行库该自定义控件所在的位置(在哪个程序集中)，并将其与指定的标签前缀关联起来。

```
<%@ Register
Assembly="CustomControlLib, Version=1.0.0.0,
Culture=neutral,
```

```
PublicKeyToken=null"
Namespace="CustomControlLib" TagPrefix="ccl" %>
```

清单 4.2 展示了该控件在页面上的声明方式，其 Text 属性被设置为 “When you set the control's property, it shows up in the designer ...”。

清单 4.2 包含自定义控件的 UseCustomControl.aspx

```
<form id=form1 runat=server>
<div>
<ccl:ServerControl1 ID="ServerControl11"
    runat="server"
    Text="When you set the control's property,
        it shows up in the designer..." />

</div>
</form>
```

现在可以修改该控件的其他几个属性，并观察其在“设计器”中的变化(例如，修改字体的效果会比较明显)。我们在“属性”窗口中所能看到的属性都是标准属性，都是从 System.Web.UI.WebControl 继承下来的。

6. 向页面添加一个标签、一个文本框和一个按钮，用换行符来分隔文本框与按钮。在完成之后，Visual Studio 会生成清单 4.3 所示代码。

清单 4.3 修改后的 UseCustomControl.aspx

```
<form id=form1 runat=server>
<div>
<ccl:ServerControl1 ID="ServerControl11"
    runat="server"
    Text="When you set the controls property,
        it shows up in the designer..." />
    <br />
    <br />
    <asp:Label ID="Label1"
        runat="server"
    />
    <asp:TextBox ID="TextBox1" runat="server">
    </asp:TextBox>

    <br />
    <asp:Button ID="Button1"
        runat="server" Text="Button" />
</div>
</form>
```

请注意，这些标准的 ASP.NET 控件(标签、文本框和按钮)的标签都带有“asp:”前缀，而新的自定义控件标签的前缀是“ccl:”。这个前缀是在“设计器”中添加控件时由 Visual Studio 生成的，但开发者可以通过修改 Register 指令的 TagPrefix 属性。^①

① 译者注：修改此属性后，对于已添加的自定义控件，Visual Studio 2010 不会自动更新其标签前缀(需要逐一替换)，但后续添加的控件会使用新的设置。

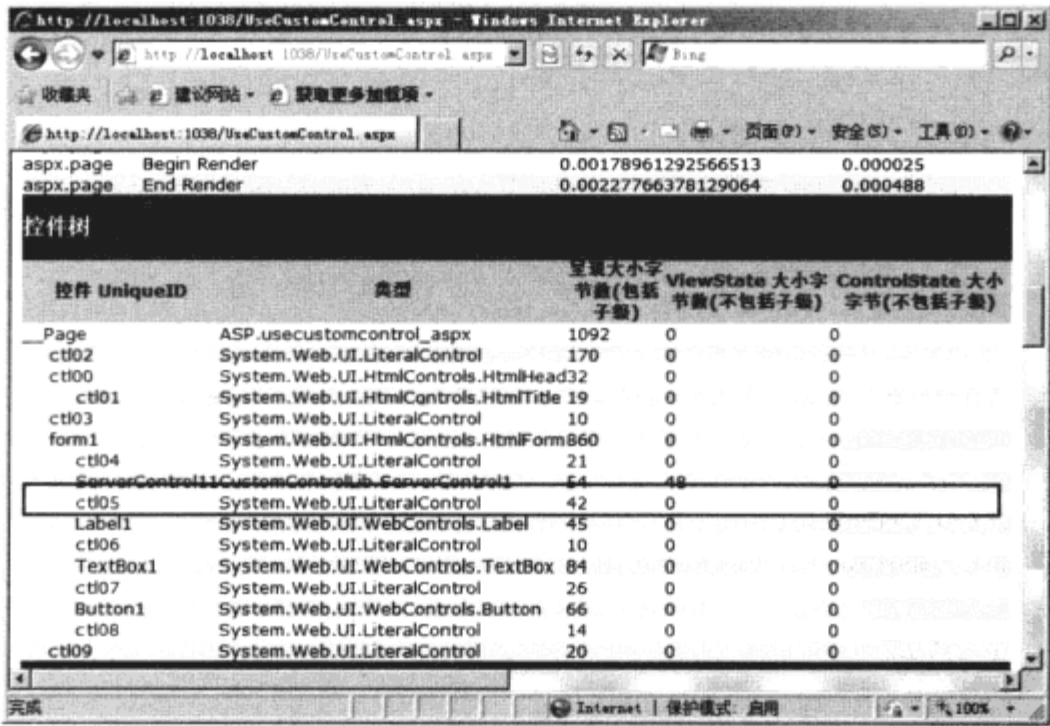
7. 在设计器中双击刚才创建的按钮，为其添加一个事件处理程序。在 Visual Studio 创建好事件处理程序的框架后，在其中将 TextBox 中的文本赋给自定义控件的 Text 属性。为此，需要键入以下加粗显示的代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    this.ServerControl11.Text = this.TextBox1.Text;
}
```

按 Ctrl+F5 浏览这个带自定义控件的页面。当我们在文本框中输入一些文本并按下按钮后，浏览器便会将请求发送至服务器。服务器会读出文本框的内容，并将其赋给 ServerControl1 的 Text 属性(见下图)。



可以观察一下在控件树中新控件的跟踪信息(见下图)。(为启用页面跟踪，需要将 Page 指令 Trace 属性设置为 true，详见第 3 章。)



至此我们已经构建了一个简单的控件。这种控件框架十分灵活，我们可以使用 RenderContents 方法来生成任何文本形式的内容。接下来，我们将开发一个稍微复杂一些的控件，以演示更高级的控件呈现技术。

4.3 回文验证器

针对编写为客户端呈现标记的简单服务器端控件，上一个示例已经介绍了相关的基础知识。不过，ASP.NET 已经能够呈现非常完美的 Label 控件，我们没有必要再去制作了。下面将要制作一个能够检查用户在客户端输入的字符串是否为回文的服务器端控件。在这个练习中，您将学习到更高级的呈现技术，并从中了解到控件事件的工作方式。

➤ 构建一个回文验证器

- 1. 为创建这个回文验证控件，在“解决方案资源管理器”中，右键单击“CustomControlLib”节点，在快捷菜单中依次选择“添加”|“新建项”。在下图所示的“已安装的模板”选择“Web”类别，然后选择“ASP.NET 服务器控件”。在“名称”文本框中输入 **PalindromeCheckerRenderedControl**，然后单击“添加”按钮生成相关代码。



- 2. 为 PalindromeCheckerRenderedControl 类添加一个验证回文的方法。在英语里，所谓“回文” (palindrome)是指对称的单词、句子或短语(如“radar”)。下面我们为该控件添加一个检查内部文本是否为回文的方法。这是一个简单的检验回文的方法，它将文本中的字母都转换为大写，倒序，然后将结果与倒序前的文本进行比较。为此，我们应剔除非字母字符。清单 4.4 给出了实现该功能的代码。(这个示例使用了一种过时的字体标签，但稍后就会将其替换为当今流行的风格。)

清单 4.4 字符提取

```
protected string StripNonAlphanumerics(string str)
{
    string strStripped = (String)str.Clone();
    if (str != null)
    {
        char[] rgc = strStripped.ToCharArray();
        int i = 0;
        foreach (char c in rgc)
        {
```



```

        if (char.IsLetterOrDigit(c))
        {
            i++;
        }
        else
        {
            strStripped = strStripped.Remove(i, 1);
        }
    }
}
return strStripped;
}

protected bool CheckForPalindrome()
{
    if (this.Text != null)
    {
        String strControlText = this.Text;
        String strTextToUpper = null;
        strTextToUpper = Text.ToUpper();
        strControlText =
            this.StripNonAlphanumerics(strTextToUpper);
        char[] rgcReverse = strControlText.ToCharArray();
        Array.Reverse(rgcReverse);
        String strReverse = new string(rgcReverse);
        if (strControlText == strReverse)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}
}

```

3. 修改呈现方法，将回文以蓝色文本输出，非回文以红色文本输出。RenderContents 方法只接受一个 HtmlTextWriter 类型的参数。HtmlTextWriter 除了允许我们将文本流发送给浏览器外，还具有一些非常有用的特性，稍后会用到它们。现在，可以将其当作 Response.Write 看待。通过 Write 方法发送的内容，都将传给客户端浏览器。

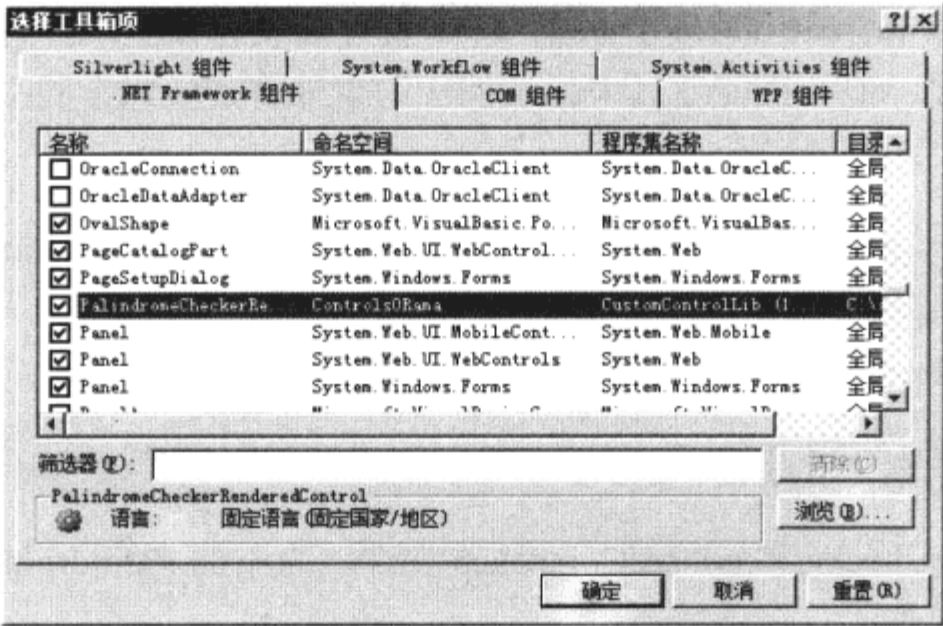
```

Protected override void RenderContents(HtmlTextWriter output)
{
    if (this.CheckForPalindrome())
    {
        output.Write("This is a palindrome: <BR/>");
    }
}

```

```
        output.Write("<FONT size=5 color=blue>");
        output.Write("<B>");
        output.Write(this.Text);
        output.Write("</B>");
        output.Write("</FONT>");
    } else {
        output.Write("This is NOT a palindrome <BR/>");
        output.Write("<FONT size=5 color=red>");
        output.Write("<B>");
        output.Write(this.Text);
        output.Write("</B>");
        output.Write("</FONT>");
    }
}
```

- 4. 在主菜单中依次选择“生成”|“生成解决方案”，生成项目。
- 5. Visual Studio 会将 PalindromeCheckerRenderedControl 添加到“工具箱”中。如果没有，则可以手动添加：在“工具箱”上单击右键，单击“选择项”。使用“浏览”按钮来查找 CustomControlLib.DLL 程序集，然后选择它。这样，Visual Studio 便会在“工具箱”中加载这个新控件。



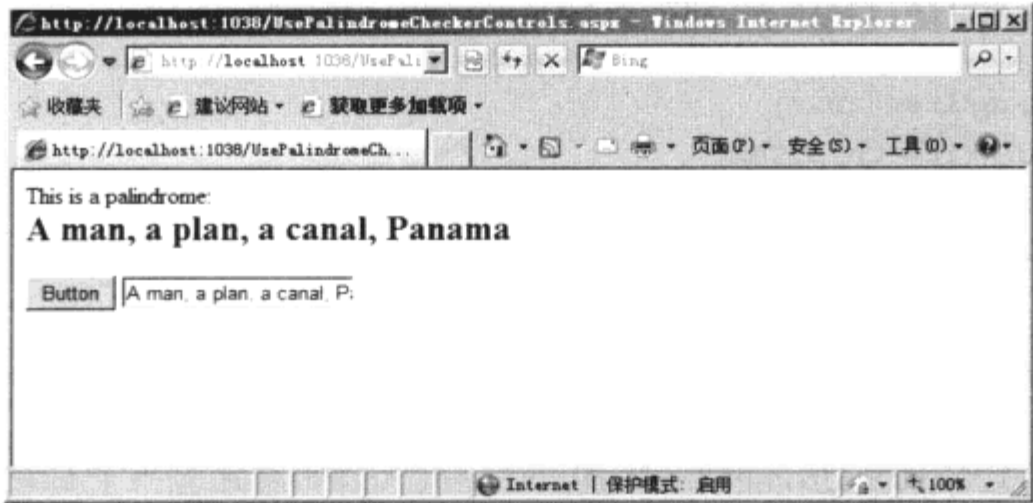
- 6. 在 ControlsORama 项目中添加一个“Web 窗体”来容纳刚刚创建的回文验证控件，将该页面命名为 UsePalindromeCheckerControls.aspx。拖放一个 PalindromeCheckerRenderedControl 到该页面上。再添加一个文本框和按钮，以便我们输入一段文本，然后判断它是否为回文。
- 7. 双击按钮，为其添加一个事件处理程序。Visual Studio 会自动生成这个处理程序的框架。在这个处理程序中，将 PalindromeCheckerRenderedControl 的 Text 属性设置为 TextBox.Text 属性。

```
Public partial class UsePalindromeCheckerControls : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
```



```
        this.PalindromeCheckerRenderedControl1.Text = this.TextBox1.Text;
    }
}
```

- 8. 运行这个页面，并测试回文验证功能。见下图，回文和非回文会有不同的显示在你的屏幕上，显示为蓝色，而非回文显示为红色。



4.4 控件与事件

PalindromeCheckerRenderedControl 会根据 Text 属性的状态来呈现控件的内容。虽然它自身已经实现了该功能，但如果能通过来自该控件的事件将其中包含回文这一事实告知宿主页面，这样会更好。

ASP.NET 中大多数标准服务器端控件都支持事件。我们已经了解了 Button 控件在被单击后是如何向宿主页面发送消息的。这一机制适用于任何控件。下面，我们将为 PalindromeCheckerRenderedControl 控件添加一个名为 PalindromeFound 的事件。

➤ 添加 PalindromeFound 事件

- 1. 打开 PalindromeCheckerRenderedControl.cs 文件。为添加 PalindromeFound 事件，需要键入下面这行代码：

```
public class PalindromeCheckerRenderedControl : WebControl
{
    public event EventHandler PalindromeFound;
    // Other palindrome control code goes here
}
```

- 2. 在宿主页面订阅该事件后，接下来需要确定其被引发的时机。应该在发现回文时引发该事件，比较理想的地方是在 Text 属性的 set 方法中。在该控件的 Text 属性中添加下面一行加粗显示的代码，并重新生成项目：

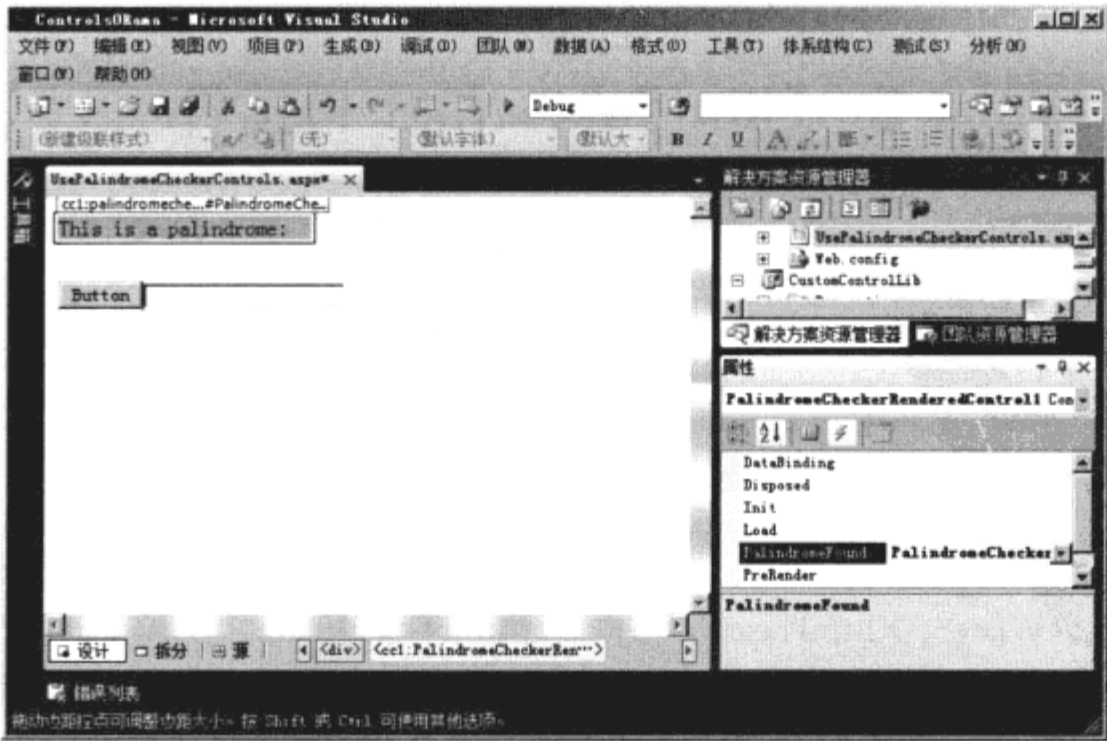
```
[Bindable(true)]
[Category(Appearance)]
[DefaultValue("")]
[Localizable(true)]
```

```
public string Text
{
    get
    {
        string s = (String)ViewState[Text];
        return ((s == null) ? String.Empty : s);
    }

    set
    {
        ViewState[Text] = value;
        if (this.CheckForPalindrome())
        {
            if (PalindromeFound != null)
            {
                PalindromeFound(this, EventArgs.Empty);
            }
        }
    }
}
```

不难注意到，Visual Studio 2010 生成的代码会将 Text 属性的值存储在该控件的 ViewState(视图状态)中。这样，在两次回发间，该属性能够保留其值。稍后会更深入地介绍这一机制。

- 3. 为了能使该事件被引发，可以将页面中的 PalindromeCheckerRenderedControl 实例删除，并重新创建一个。这样会使 Visual Studio 刷新 CustomControlLib.DLL 程序集，从而使更改(新的事件)反映在界面上。在页面上选择 PalindromeCheckerRenderedControl 控件，单击 Visual Studio “属性”窗口中的“事件”按钮。双击 PalindromeFound 事件旁边的文本框。Visual Studio 会自动为我们创建一个事件处理程序。



- 4. 响应 PalindromeFound 事件。这里只是通过 Response.Write 向浏览器发送了一些文本。


```
public partial class UsePalindromeCheckerControls : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        this.PalindromeCheckerRenderedControl1.Text =
            this.TextBox1.Text;
    }
    protected void PalindromeCheckerControl1_PalindromeFound(
        object sender, EventArgs e)
    {
        Response.Write("The page detected a PalindromeFound event");
    }
}
```

运行该页面。下图展示了输入回文后的效果。



至此，这个自定义控件已能够正确呈现回文并引发了一个事件。下面让我们深入了解一下调用 `RenderContents` 方法时传入的参数——`HtmlTextWriter`。

4.5 HtmlTextWriter 与控件

让我们简单地回顾一下上一示例中的 `RenderContents` 方法。该方法通过文本字体标签更改了回文文本的颜色。虽然这么做可行，但这种技术有几个缺陷。例如，HTML 具有多个标准。浏览器一般同时支持 HTML 的 3.2 版本和 4.0 版本，而对某些 HTML 元素的支持在这两个版本之间是不同的。如果所呈现的 HTML 只针对某种浏览器，那么使用其他浏览器的用户可能会对 HTML 解释的差异而感到困惑。

提示 .NET Framework 包含多个版本的 `HtmlTextWriter` 类：`Html32TextWriter`、`HtmlTextWriter`、`XhtmlTextWriter` 和 `ChtmlTextWriter`。当服务器收到某个请求时，它会读取用于甄别浏览器类型的标头信息。大多数浏览器都能够解释当前流行的 HTML 版本。在这种情况下，ASP.NET 会将常规的 `HtmlTextWriter` 对象传入 `RenderContents` 方法。然而，如果遇到只能解释 HTML 3.2 的浏览器，ASP.NET 便会传入 `Html32TextWriter`。这些类各具所长且可以相互替换。`Html32TextWriter`

能够生成 HTML 3.2 格式的标签(如表格所涉及的标签)而 `HtmlTextWriter` 能够生成 HTML 4.0 格式对应的标签。`machine.config` 中的信息和浏览器功能配置，能够帮助 ASP.NET 确定应该使用哪种 `HtmlTextWriter`。ASP.NET 运行库使用的这种浏览器功能配置不仅仅能够用来确定 `HtmlTextWriter`。`Request` 属性(`HttpContext` 和 `Page` 中的)包含一个 `Browser` 对象。这个对象具有许多标志，指示着不同信息(如是哪种浏览器发出的请求、浏览器是否支持脚本，以及浏览器所处平台的名称等)。这种信息会包含于每个请求的标头中。ASP.NET 运行库会通过配置文件中的常见正则表达式来获取不同的信息。下面这段代码演示了如何判断发出请求的浏览器是否支持框架(frame)：

```
public class TestForFramesControl : Control
{
    protected override void RenderContents(HtmlTextWriter output)
    {
        if (Page.Request.Browser.Frames)
        {
            output.Write(
                "This browser supports Frames");
        }
        else
        {
            output.Write("No Frames here");
        }
    }
}
```

为了掌握 `HtmlTextWriter` 的高级功能，下面我们将修改 `PalindromeCheckerRenderedControl`。`RenderContents` 方法，把使用硬编码字体标签的代码替换为利用 `HtmlTextWriter` 的代码。

➤ 使用 `HtmlTextWriter`

1. 打开 `PalindromeCheckerRenderedControl.cs` 文件。
2. 更新 `RenderContents` 方法，使它利用 `HtmlTextWriter.RenderBeginTag` 来生成字体和加粗的开始标签。通过 `HtmlTextWriter.AddStyleAttribute` 改变字体颜色。

```
protected override void RenderContents(HtmlTextWriter output)
{
    if (this.CheckForPalindrome())
    {
        output.Write("This is a palindrome: <br/>");
        output.RenderBeginTag(HtmlTextWriterTag.Font);
        output.AddStyleAttribute(HtmlTextWriterStyle.Color, "blue");
        output.RenderBeginTag(HtmlTextWriterTag.B);
        output.Write(Text);
    }
}
```



```
        output.RenderEndTag(); // bold
        output.RenderEndTag(); // font
    } else {
        output.Write("This is not a palindrome: <br/>");
        output.RenderBeginTag(HtmlTextWriterTag.Font);
        output.AddStyleAttribute(HtmlTextWriterStyle.Color, "red");
        output.RenderBeginTag(HtmlTextWriterTag.B);
        output.Write(Text);
        output.RenderEndTag(); // bold
        output.RenderEndTag(); // font
    }
}
```

HtmlTextWriter 类和相应的枚举掩盖了 HTML 3.2 和 4.0 之间的差异。清单 4.5 和清单 4.6 分别给出了该控件针对 HTML 4.0 和 HTML 3.2 标准呈现的内容。

清单 4.5 采用 HTML 4.0 标准呈现的控件

```
<span id="PalindromeCheckerRenderedControl1">
This is a palindrome:
<br/>
<font><b style="color:blue;">Radar</b></font>
</span>
```

清单 4.6 采用 HTML 3.2 标准呈现的控件

```
<span id="PalindromeCheckerRenderedControl1">
This is a palindrome:
<br/>
<font color=blue><b>Radar</b></font>
</span>
```

4.6 控件与视图状态

在结束自定义控件的话题之前，让我们了解一下由控件维护的视图状态(view state)。如果回顾之前章节中的传统 ASP 页面示例，可能会注意到在回发后呈现的控件行为比较怪异。如果在组合框中选择某一项并将表单提交给服务器，那么在响应返回后，控件(如示例中选择控件)的状态会丢失。曾经介绍过，Web 编程模型无非就是获取服务器状态的快照，并通过浏览器将其显示出来。这从根本上说就是试图在非连接协议上实现有状态的用户界面(UI)开发。

ASP.NET 服务器端控件包含一种保存页面可视状态的方法——使用 Page 类中的 ViewState 属性，我们可以在需要时轻松地访问它。ViewState 是一种字典(键/值集合)，能够存储任何可序列化的对象。

大多数 ASP.NET 服务器端控件都会通过存储和获取 ViewState 中的数据项来管理其可视状态。例如，选择控件会在两次回发期间保存选定项的索引，以便确定为哪一项添加 selected 特性，从而呈现相应的标签来使该项在用户的浏览器中显示为选中。

在两次回发期间，页面的所有状态信息都会编码到一个隐藏字段中。如果浏览.aspx 页面，并查看来自服务器的源代码，则会发现 ViewState 会被转换为以 BASE 64 编码的字节流。

为了解 ViewState 的工作方式，下面我们将添加一些代码，跟踪通过该控件显示过的回文。

➤ 使用 ViewState

1. 打开 PalindromeCheckerRenderedControl.cs 文件。

2. 在 using 列表中添加 System.Collections 的引用。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Collections
```

3. 为控件添加一个 ArrayList 来保存查看过的回文。更新 Text 属性的 set 方法，如果当前文本是回文，则将其存储在视图状态中：

```
public class PalindromeCheckerRenderedControl : WebControl
{
    public event EventHandler PalindromeFound; // public event
    ArrayList alPalindromes = new ArrayList();

    [Bindable(true)]
    [Category(Appearance)]
    [DefaultValue("")]
    [Localizable(true)]
    public string Text
    {
        get
        {
            string s = (String)ViewState[Text];
            return ((s == null) ? String.Empty : s);
        }
        set
        {
            ViewState[Text] = value;
            string text = value;
            this.alPalindromes =
                (ArrayList)this.ViewState["palindromes"];
            if (this.alPalindromes == null)
            {
                this.alPalindromes = new ArrayList();
            }
            if (this.CheckForPalindrome())
            {

```



```

        if (PalindromeFound != null)
        {
            PalindromeFound(this, EventArgs.Empty);
        }
        alPalindromes.Add(text);
    }
    ViewState.Add("palindromes", alPalindromes);
}
}
}

```

4. 添加一个方法，以表格形式呈现回文集合，并更新 `RenderContents` 方法使其调用该方法：

```

protected void RenderPalindromesInTable(HtmlTextWriter output)
{
    output.AddAttribute(HtmlTextWriterAttribute.Width, "50%");
    output.AddAttribute(HtmlTextWriterAttribute.Border, "1");
    output.RenderBeginTag(HtmlTextWriterTag.Table); // <table>

    foreach (string s in this.alPalindromes)
    {
        output.RenderBeginTag(HtmlTextWriterTag.Tr); // <tr>
        output.AddAttribute(HtmlTextWriterAttribute.Align, "left");
        output.AddStyleAttribute(HtmlTextWriterStyle.FontSize, "medium");
        output.AddStyleAttribute(HtmlTextWriterStyle.Color, "blue");
        output.RenderBeginTag(HtmlTextWriterTag.Td); // <td>
        output.Write(s);
        output.RenderEndTag(); // </td>
        output.RenderEndTag(); // </tr>
    }
    output.RenderEndTag(); // </table>
}

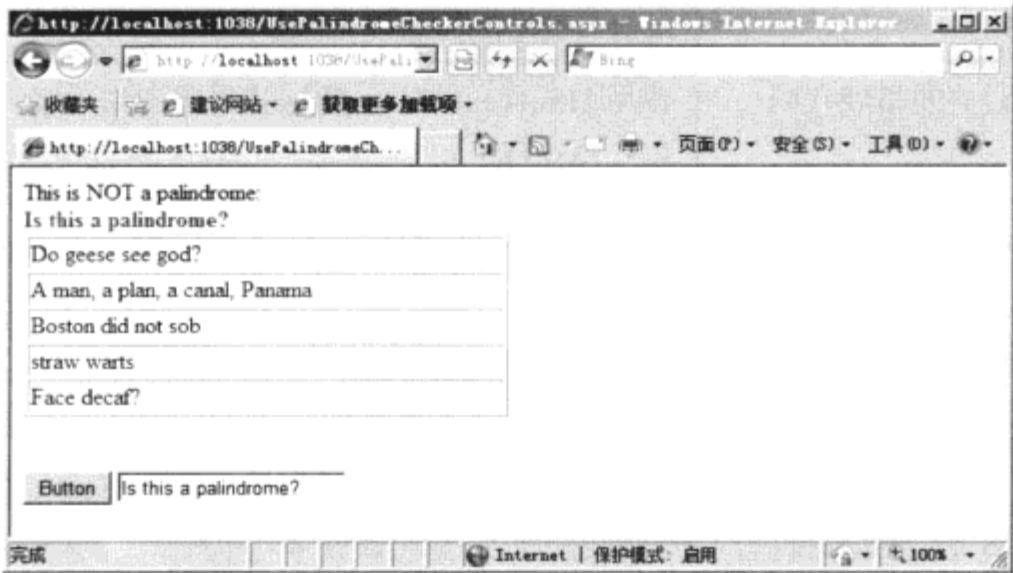
protected override void RenderContents(HtmlTextWriter output)
{
    if (this.CheckForPalindrome())
    {
        output.Write(This is a palindrome: <br>);
        output.RenderBeginTag(HtmlTextWriterTag.Font);
        output.AddStyleAttribute(HtmlTextWriterStyle.Color, blue);
        output.RenderBeginTag(HtmlTextWriterTag.B);
        output.Write(Text);
        output.RenderEndTag(); // bold
        output.RenderEndTag(); // font
    } else {
        output.Write(This is NOT a palindrome: <br>);
        output.RenderBeginTag(HtmlTextWriterTag.Font);
        output.AddStyleAttribute(HtmlTextWriterStyle.Color, red);
        output.RenderBeginTag(HtmlTextWriterTag.B);
        output.Write(Text);
    }
}

```

```
        output.RenderEndTag(); // bold
        output.RenderEndTag(); // font
    }

    output.Write("<br>");
    RenderPalindromesInTable(output);
}
```

- 5. 生成并运行该应用程序。在浏览这个包含回文验证控件的页面时，应在表格中看到之前输入过的回文(见下图)。



由于该控件在 ViewState 中存储了更多信息，所以回发后 HTML 响应中的 _VIEWSTATE 字段也随之变大。向页面多添加几段回文，每次查看一下发送给浏览器的源码。我们会发现每次回发，隐藏字段 VIEWSTATE 的大小都有增加。应注意的是，使用了视图状态的控件会增加返回浏览器的 HTTP 有效载荷的体积。过度使用视图状态会导致网站性能的下降，应谨慎为之。

4.7 快速参考

目 标	操 作
新建呈现过程可控的自定义控件	从 System.Web.UI.Control 派生一个类。重写 RenderControl 方法。也可以使用 Visual Studio 中的一种名为“ASP.NET 服务器控件”的项目类型
将自定义控件添加到“工具箱”	打开“工具箱”(如果其尚未被显示，可以在主菜单中依次选择“视图” “工具箱”)。在“工具箱”的任意位置单击右键，然后在快捷菜单中单击“选择项”。从列表中选择该控件，或通过浏览包含该控件的程序集来添加
修改页面上控件的属性	确保页面的编辑器处于“设计”视图。选择目标控件，并在“属性”窗口中编辑所要修改的属性

续表

目 标	操 作
管理页面上控件所引发的事件	确保页面的编辑器处于“设计”视图。选择目标控件，在事件窗口中选择所要处理的事件(单击“属性”窗口上端的闪电图标)。可以通过双击它来使 Visual Studio 插入相应的处理程序，也可以直接输入处理程序的名称
存储特殊的视图状态信息	确保所要存储的数据类型是可序列化的，并使用控件的 ViewState 属性(一种键/值集合)。在获取之前存储的信息时，应确保使用相同的索引值/键
编写与浏览器版本无关的呈现代码	使用 HtmlTextWriter 的标签呈现方法来插入标签，而不要硬编码。ASP.NET 会根据浏览器的标头信息来决定将哪种 HtmlTextWriter 传入 RenderControl

第 5 章 复合控件

学习目标

- 创建自定义复合控件
- 创建复合用户控件
- 在应用程序中引用这两种控件
- 了解这两种控件各自的适用范围

第 4 章详细介绍了呈现过程可自定义的控件，而本章将介绍另一种控件——复合控件。ASP.NET 定义了两类复合控件——自定义复合控件和用户控件。每种复合控件都各有所长，本章会介绍它们各自的优缺点。下面先让我们看看复合控件与自定义控件有何区别。


5.1 复合控件与自定义控件

自定义控件需要使用 `System.Web.UI.Control.RenderControl` 方法来生成返回给客户端的所有 HTML。这种控件能够控制整个呈现过程。使用自定义控件，我们可以获得非常优异的灵活性，完全掌控网站发送的 HTML(具体到每个 HTML 标签)。

然而，这种可控性和灵活性需要考虑的细节较多。例如，假设要为自定义控件添加一个输入按钮，则需要在返回给客户端的响应流中插入能够正确描述这种按钮的 HTML 标记。如果要添加更复杂的控件，如选择控件，那么情况就更复杂了，可能需要对项目集合进行跟踪。虽然可以使用 HTML 轻松地描述输入按钮和选择控件，但 ASP.NET 已经有能够呈现所需标签的服务器端控件了。这些标准的 ASP.NET 控件极大地简化了基于“Web 窗体”的用户界面(UI)编程。

复合控件由其他控件复合而成，能够利用现有的服务器端控件。为了说明复合控件的效用，不妨设想这样一个场景：有多个项目要求具有外观一致的登录界面。虽然在 Microsoft Visual Studio 中构建“Web 窗体”非常简单，但如果需要创建一并出现的某组控件的多个实例，那么重复地创建页面就显得枯燥乏味了。ASP.NET 通过复合控件解决了这一问题。

如果多个网站要使用相同的登录功能，则可以在单个控件中把用户名/密码标签和文本框控件组织在一起。然后，如果某个网站需要登录页面，则只需要将这个独立的控件放置到新窗体上。该控件中的子控件(和执行逻辑)捆绑在一起，因而不需要重复编写相同的 HTML。

 **提示** 自 2.0 版本开始，ASP.NET 便包含了一组与登录相关的复合控件，因此不必从头编写。在这里提到这种控件只是为了说明复合控件的强大功能。

下面先从自定义的复合控件开始说起。

5.2 自定义的复合控件

第 4 章介绍了为浏览器呈现 HTML 的自定义控件。这种控件与其他控件的主要区别在于，前者需要重写 `RenderContents` 方法。请不要忘记，`System.Web.UI.Page` 类管理着一系列服务器端控件。当 ASP.NET 令页面进行呈现时，页面会对其上的所有控件进行遍历，并使每个控件呈现自身。自定义控件只是将文本写入准备发送给浏览器的流中。如果页面的呈现过程遇到复合控件，那么该复合控件也会遍历自己的子控件，并让每个控件呈现自身(就像 `Page` 遍历自己的子控件一样)。

复合控件可以包含任意数量的控件(只要内存可以容纳)，控件的嵌套深度也可以是任意的。然而，实践当中应限制子控件的数量和深度。添加过多的控件或嵌套层次过深，都会增加页面的复杂程度，降低其可读性。此外，添加过多嵌套的控件还会严重影响应用程序的性能——遍历控件集合并呈现每个控件是耗费时间的。

在第 4 章中，我们创建了一个能够验证回文的控件。当该控件的 `Text` 属性被设置成回文后，该控件会将其呈现为蓝色，将这段回文添加到一个 `ArrayList` 中，并以表格的形式呈现这个回文集合的内容。下面，我们将构建一个功能类似的控件，但使用复合控件来实现。

► 以自定义复合控件的形式来构建回文验证器

1. 打开 `ControlsORama` 项目。在“解决方案资源管理器”中选择“`CustomControlLib`”项目。右键单击该项目节点，选择“添加”|“新建项”。创建一个“ASP.NET 服务器控件”(所使用的模板与第 4 章 `PalindromeCheckerRenderedControl` 控件的模板相同)，并将其命名为 `PalindromeCheckerCompositeControl.cs`。
2. 在 Visual Studio 创建默认代码后，进行以下操作。
 - a. 编辑代码，将基类由 `WebControl` 改为 `CompositeControl`。从 `CompositeControl` 控件派生会隐式地添加 `INamingContainer` 接口。(ASP.NET 会通过该接口为每个控件的子控件分配唯一的 ID。)
 - b. 添加 `PalindromeFound` 事件，以便在发现回文后通知宿主页面(与上一章中对应的步骤相同)。
 - c. 移除 `RenderContents` 方法。
 - d. 为该控件添加 4 个字段，类型分别为 `TextBox`、`Button`、`Label` 和 `LiteralControl`。

完成后的代码应如下所示：

```
public class PalindromeCheckerCompositeControl : CompositeControl
{
    protected TextBox textboxPalindrome;
    protected Button buttonCheckForPalindrome;
    protected Label labelForTextBox;
    protected LiteralControl literalControlPalindromeStatus;
```

```
public event EventHandler PalindromeFound;
...
// RenderContents method removed.
}
```

保留完整的 `Text` 属性。这个控件仍需要它。

该控件与第 4 章中创建的非常类似。然而，这个版本包含用于输入回文的 `TextBox` 和用于触发验证过程的 `Button`，还包含显示当前输入的文本是否为回文的文本(literal)控件。

3. 从 `PalindromeCheckerRenderedControl` 类将 `StripNonAlphanumerics` 和 `CheckForPalindrome` 方法复制过来：

```
protected string StripNonAlphanumerics(string str)
{
    string strStripped = (String)str.Clone();

    if (str != null)
    {
        char[] rgc = strStripped.ToCharArray();
        int i = 0;

        foreach (char c in rgc)
        {
            if (char.IsLetterOrDigit(c))
            {
                i++;
            }

            else
            {
                strStripped = strStripped.Remove(i, 1);
            }
        }
    }

    return strStripped;
}

protected bool CheckForPalindrome()
{
    if (this.Text != null)
    {
        String strControlText = this.Text;
        String strTextToUpper = null;

        strTextToUpper = Text.ToUpper();

        strControlText =
            this.StripNonAlphanumerics(strTextToUpper);
    }
}
```



```

        char[] rgcReverse = strControlText.ToCharArray();
        Array.Reverse(rgcReverse);
        String strReverse = new string(rgcReverse);
        if (strControlText == strReverse)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }
}

```

4. 为 **Button** 添加一个事件处理程序(稍后会在页面中添加这个按钮控件)。由于我们现在所编写的是一种没有“设计器”支持的二进制控件，因而需要使用“文本向导”来添加这个事件处理程序(也就是说，需要手动添加)。

```

Public void OnCheckPalindrome(Object o, System.EventArgs ea)
{
    this.Text = this.textboxPalindrome.Text;
    this.CheckForPalindrome();
}

```

5. 重写 **CreateChildControls** 方法。**CreateChildControls** 是复合控件与自定义控件的主要区别。在这个方法中，我们需要手动创建每个 UI 元素，设置希望在该控件中显示的属性，并将每个子控件逐一添加到复合控件的控件列表中。

```

Protected override void CreateChildControls()
{
    labelForTextBox = new Label();
    labelForTextBox.Text = "Enter a palindrome: ";
    this.Controls.Add(labelForTextBox);

    textboxPalindrome = new TextBox();
    this.Controls.Add(textboxPalindrome);

    Controls.Add(new LiteralControl("<br/>"));

    buttonCheckForPalindrome = new Button();
    buttonCheckForPalindrome.Text = "Check for Palindrome";
    buttonCheckForPalindrome.Click += new EventHandler(OnCheckPalindrome);
    this.Controls.Add(buttonCheckForPalindrome);

    Controls.Add(new LiteralControl("<br/>"));
}

```

```
literalcontrolPalindromeStatus = new LiteralControl();
Controls.Add(literalcontrolPalindromeStatus);

this.tablePalindromes = new Table();①
this.Controls.Add(tablePalindromes);

Controls.Add(new LiteralControl("<br/>"));

this.ChildControlsCreated = true;
}
```

虽然这段代码非常简单，但有几点需要注意。首先，这段代码使用 `LiteralControl` 控件来呈现换行。请记住，页面上的每个元素(在本示例中为控件)都要通过服务器端控件呈现。如果字符文本也是控件的一部分(如`
`元素)，则需要使用服务器端控件对其进行包装。`LiteralControl` 旨在将自身的内容(`Text` 属性)呈现到传出的流中。

第二点需要注意的是，事件处理程序与 `Button` 是通过委托关联起来的。如果有 `Visual Studio` 的“设计器”支持，则可以通过在“设计器”中单击 UI 元素来关联事件处理程序——`Visual Studio` 会自动添加必要的代码。然而，这里没有“设计器”的支持，因而只能手动添加。

6. 为在 `Text` 属性被设置时显示回文验证状态，可按以下代码修改 `Text` 属性。`Text` 属性的 `set` 方法会判断值是否为回文，并将结果呈现在第 2 步添加的 `LiteralControl` 中。如果视图状态中没有回文列表，则应在 `set` 方法中实例化一个。这段代码还应引发 `PalindromeFound` 事件。

```
private string text;

public string Text
{
    get
    {
        return text;
    }
    set
    {
        text = value;
        if (this.alPalindromes == null)
        {
            this.alPalindromes = new ArrayList();
        }
        if (this.CheckForPalindrome())
        {
            if (PalindromeFound != null)
            {

```

^① 译者注：本书在争得作者同意的前提下，更正了原书此处对应的及本示例后面的几段代码。


```

        PalindromeFound(this, EventArgs.Empty);
    }
    literalcontrolPalindromeStatus.Text =
        String.Format(
            "This is a palindrome <br/>
            <FONT size=\"5\" color=\"blue\"><B>{0}</B></FONT>",
            text);
    }
    else
    {
        literalcontrolPalindromeStatus.Text =
            String.Format(
                "This is NOT a palindrome <br/>
                <FONT size=\"5\" color=\"red\"><B>{0}</B></FONT>",
                text);
    }
}
}

```

7. 为了像该控件的上一版本一样在表格中显示回文，需要给 **PalindromeCheckerComposite Control** 添加 **ArrayList** 和 **Table** 控件：

```

using System.Collections;

public class PalindromeCheckerCompositeControl :
    CompositeControl
{
    protected Table tablePalindromes;
    protected ArrayList alPalindromes;
}

```

8. 添加一个根据 **ArrayList** 的内容生成回文表格的方法。检查 **ViewState** 中是否保存有该数组列表(之前在 **Text** 属性的 **set** 方法中创建的)。如果没有，则跳过。如果有，则对回文集合进行迭代，针对每个回文创建 **TableRow** 和 **TableCell**：

```

protected void BuildPalindromesTable()
{
    this.alPalindromes = (ArrayList)this.ViewState["palindromes"];
    if (this.alPalindromes != null)
    {
        foreach (string s in this.alPalindromes)
        {
            TableCell tableCell = new TableCell();
            tableCell.BorderStyle = BorderStyle.Double;
            tableCell.BorderWidth = 3;
            tableCell.Text = s;
            TableRow tableRow = new TableRow();
            tableRow.Cells.Add(tableCell);
            this.tablePalindromes.Rows.Add(tableRow);
        }
    }
}

```

```

    }
}
}

```

9. 更新 Text 属性的 set 方法，以便对表格进行管理。一旦找到回文，就将其添加到 ArrayList，而且每次文本有变化，都要重新构建回文表。

```

public string Text
{
    get
    {
        return text;
    }
    set
    {
        text = value;
        this.alPalindromes =
            (ArrayList)this.ViewState["palindromes"];
        if (this.alPalindromes == null)
        {
            this.alPalindromes = new ArrayList();
        }

        if (this.CheckForPalindrome())
        {
            if (PalindromeFound != null)
            {
                PalindromeFound(this, EventArgs.Empty);
            }

            alPalindromes.Add(text);

            literalcontrolPalindromeStatus.Text =
                String.Format(
                    "This is a palindrome <br/>
                    <FONT size=\"5\" color=\"blue\"><B>{0}</B></FONT>",
                    text);
        }
        else
        {
            literalcontrolPalindromeStatus.Text =
                String.Format(
                    "This is NOT a palindrome <br/>
                    <FONT size=\"5\" color=\"red\"><B>{0}</B></FONT>",
                    text);
        }

        this.ViewState.Add("palindromes", alPalindromes);
        this.BuildPalindromesTable();
    }
}

```

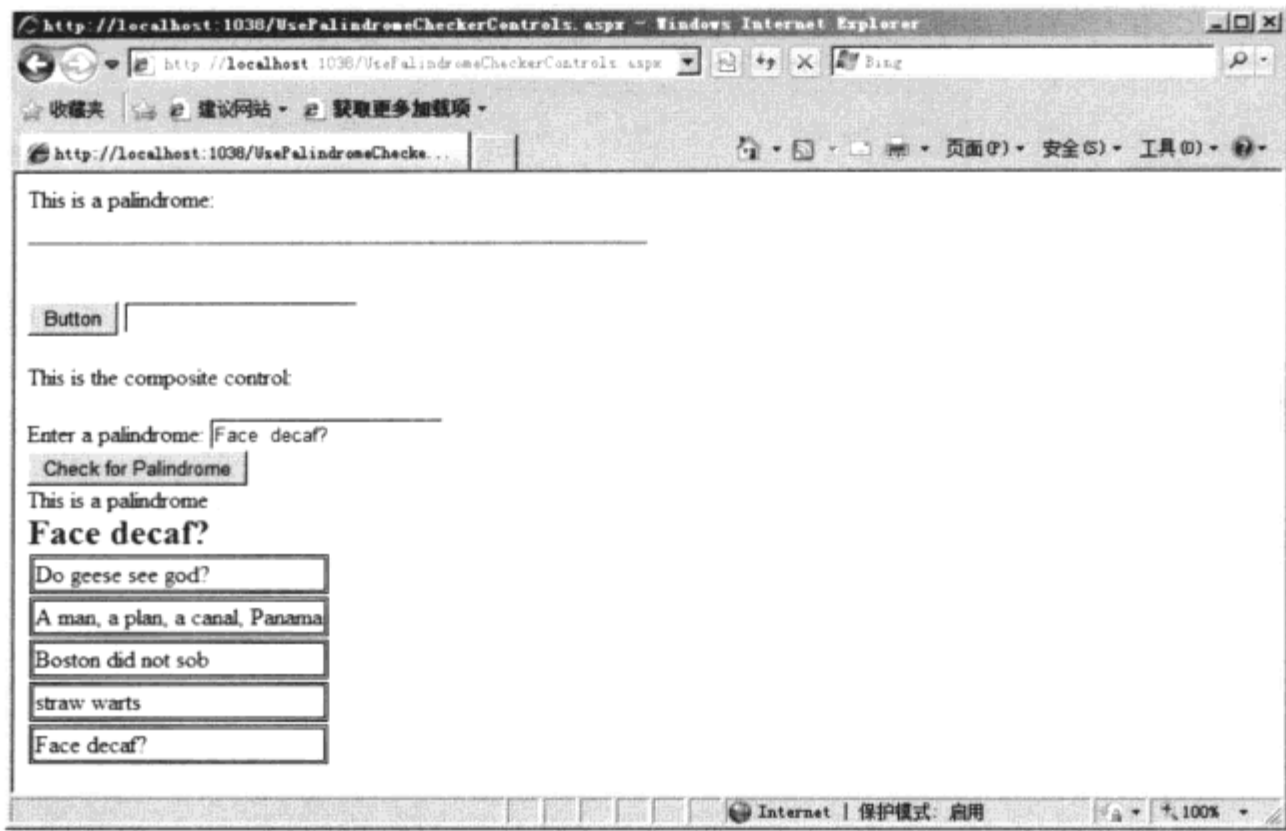
10. 生成当前项目，并将 `PalindromeCheckerCompositeControl` 控件添加到 `ControlsORama` 项目的 `UsePalindromeCheckerControls.aspx` 页面中。如果您是在上一章示例的基础之上继续进行的，则在上一章的自定义控件后面添加一个换行标签(`
`)。添加一个 `Label` 或一段文本以便提示下面的控件是一个复合控件，并插入一些换行标签。然后，从“工具箱”直接拖放一个 `PalindromeCheckerCompositeControl` 控件到页面上。当运行该页面后，新控件应能够验证回文，并跟踪所有回文记录(如下图所示)。(可以启用页面跟踪来控件树。)请注意，这个示例是对之前章节示例的扩展，页面中包含上一章创建的控件。



启用跟踪后便可以通过控件树看到控件的内部。由下图可知，`PalindromeCheckerCompositeControl` 充当了复合控件的主节点，其子控件位于这个节点以下。



在输入回文并单击按钮后，该控件会检测它们。如果不是回文，则会将当前 `Text` 属性的值显示为红色；如果是，则显示为蓝色。我们还会看到呈现的表格，其中包含输入过的所有回文(见下图)。



回文验证器非常适合演示自定义复合控件。这个复合控件完全包含在 CustomControlLib 程序集中，没有任何可视的“设计器”支持。(可以自行设计高质量的设计时支持代码，但这不在本书的讨论范围之内。)下一节将介绍另一种自行构建复合控件的方法——使用“用户控件”。

5.3 用户控件

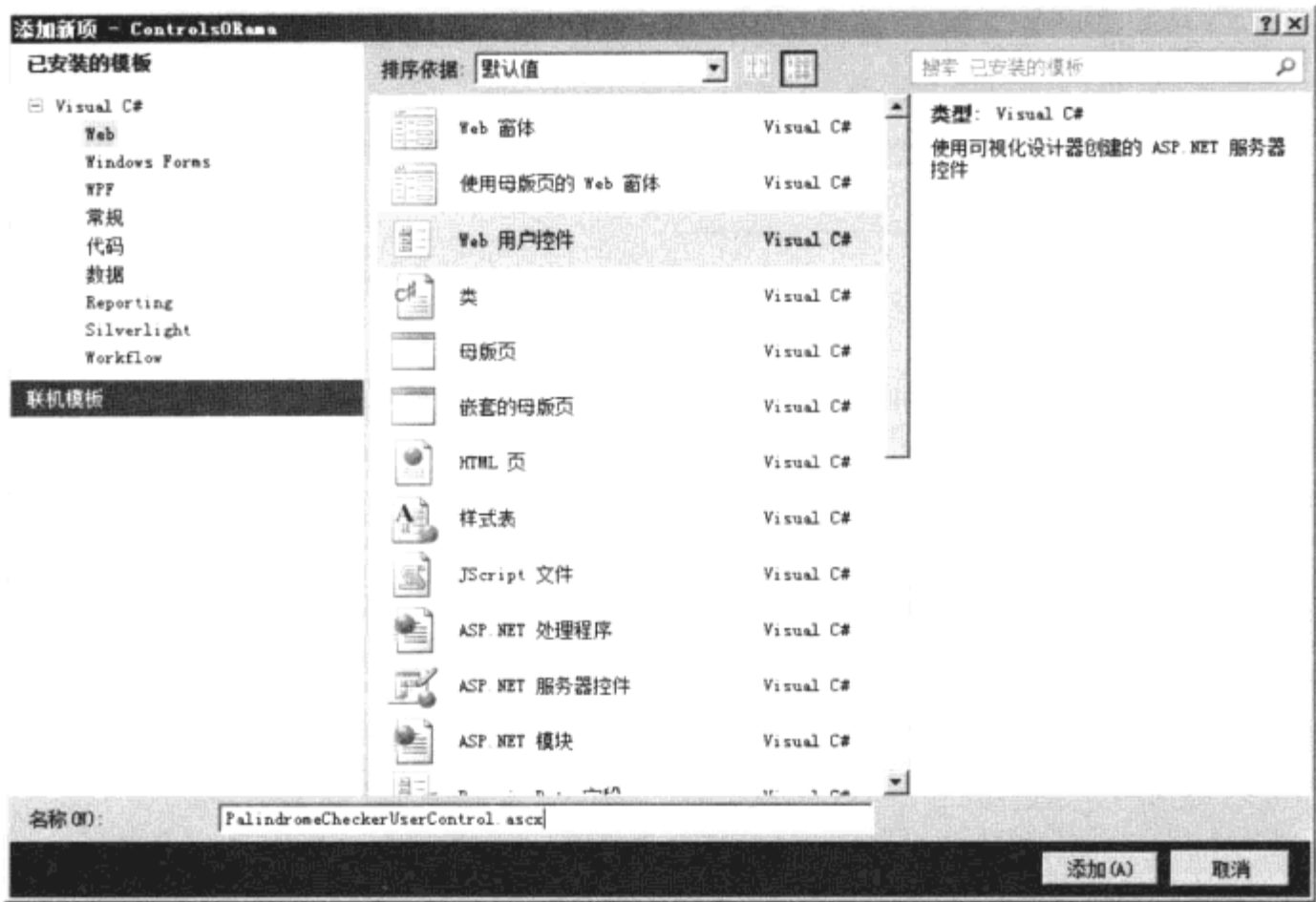
用户控件也是一种包含子控件的复合控件，非常类似自定义复合控件。然而，用户控件派生自 System.Web.UI.UserControl，而不是 System.Web.UI.CompositeControl。可以这么说，用户控件就是一个小型的“Web 窗体”。这种控件具有受 Visual Studio “设计器”支持的 UI 组件(.ascx 文件)，通过相应的类来管理执行过程。然而，与“Web 窗体”不同的是，可以将这种控件从“解决方案资源管理器”直接拖放到“Web 窗体”中。^①

为说明“用户控件”的工作方式，下面让我们以“用户控件”的形式来构建一个回文验证器。

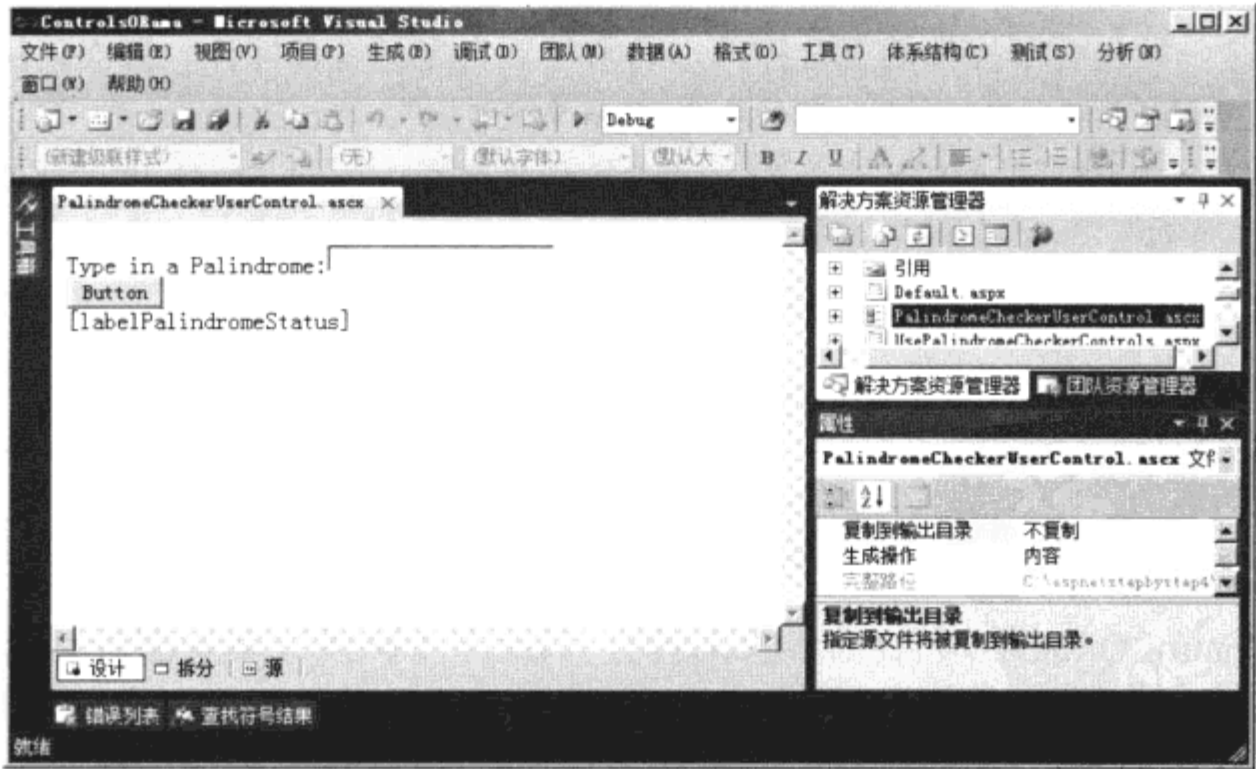
➤ 以“用户控件”的形式来构建回文验证器

1. 打开 ControlsORama 项目(如果它未被打开)。在“解决方案资源管理器”中单击“ControlsORama”网站(不是之前创建的 CustomControlLib 项目)。右键单击该网站，依次选择“添加”|“新建项”。选择“Web 用户控件”模板，将该控件命名为 **PalindromeCheckerUserControl.ascx**，如下图所示。

^① 译者注：这里根据实际情况对原文进行了修正。



2. 下面我们添加一些新控件。在添加新项后，Visual Studio 可能会直接打开“设计器”。(如果处于代码视图，则单击“设计”选项卡，切换到“设计”视图。)用户控件非常适合用“设计器”编辑。依次从“工具箱”拖放一个 Label、一个 TextBox、一个 Button 和另一个 Label 到当前用户控件上。删除第二个标签控件的 Text 属性值，以使其显示自己的标识符。按下图所示摆放各控件。



将第二个标签控件命名为 labelPalindromeStatus，以方便在代码旁置类中使用它。

3. 打开 PalindromeCheckerCompositeControl.cs 文件，将 PalindromeCheckerCompositeControl 类的 StripNonAlphanumerics 和 CheckForPalindrome 方法复制到 PalindromeCheckerUserControl.ascx.cs 文件的 PalindromeCheckerUserControl 类中。请注意，这段代码依

赖于 **Text** 属性，但该属性目前尚未被添加。这个项目在第 5 步完成之后就可以编译了。

```
protected string StripNonAlphanumerics(string str)
{
    string strStripped = (String)str.Clone();
    if (str != null)
    {
        char[] rgc = strStripped.ToCharArray();
        int i = 0;
        foreach (char c in rgc)
        {
            if (char.IsLetterOrDigit(c))
            {
                i++;
            }
            else
            {
                strStripped = strStripped.Remove(i, 1);
            }
        }
    }

    return strStripped;
}

protected bool CheckForPalindrome()
{
    if (this.Text != null)
    {
        String strControlText = this.Text;
        String strTextToUpper = null;

        strTextToUpper = Text.ToUpper();

        strControlText = this.StripNonAlphanumerics(strTextToUpper);

        char[] rgcReverse = strControlText.ToCharArray();
        Array.Reverse(rgcReverse);
        String strReverse = new string(rgcReverse);
        if (strControlText == strReverse)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    else
    {

```



```

        return false;
    }
}

```

4. 为该控件类添加 PalindromeFound 事件：

```
public event EventHandler PalindromeFound; // public event
```

5. 打开代码文件，像之前在自定义复合控件中那样添加 **text** 成员变量和 **Text** 属性。与自定义复合控件不同的是，Visual Studio 不会为用户控件生成默认属性。(相比上一个示例，这里有些细微的变动，如使用 **Label** 控件替代 **Literal** 控件来反映回文状态。如果从上一控件复制代码，则应对其做必要的修改。)

```

private String text;

public string Text
{
    get
    {
        return text;
    }
    set
    {
        text = value;

        if (this.CheckForPalindrome())
        {
            if (PalindromeFound != null)
            {
                PalindromeFound(this, EventArgs.Empty);
            }

            this.labelPalindromeStatus.Text =
                String.Format(
                    "This is a palindrome <br/>
                    <FONT size=\"5\" color=\"blue\"><B>{0}</B></FONT>",
                    text);
        }
        else
        {
            this.labelPalindromeStatus.Text =
                String.Format(
                    "This is NOT a palindrome <br/>
                    <FONT size=\"5\" color=\"red\"><B>{0}</B></FONT>",
                    text);
        }
    }
}

```

6. 为对回文进行跟踪，为该控件类添加一个 `ArrayList`(为使用 `ArrayList`，应在 `PalindromeCheckerUserControl.ascx.cs` 文件的顶部通过 `using` 语句添加 `System.Collections` 命名空间的引用)：

```
ArrayList alPalindromes;
```

7. 切换到 `PalindromeCheckerUserControl` 的“设计”视图，拖放一个 `Table` 到设计界面上，这样会添加一个 `Table` 控件。
8. 添加一个构建回文表格的方法。与 `PalindromeCheckerCompositeControl` 中的那个方法类似，只是在本示例中 `Visual Studio` 会自动将表格命名为 `Table1`。

```
protected void BuildPalindromesTable()
{
    this.alPalindromes = (ArrayList)this.ViewState["palindromes"];
    if (this.alPalindromes != null)
    {
        foreach (string s in this.alPalindromes)
        {
            TableCell tableCell = new TableCell();
            tableCell.BorderStyle = BorderStyle.Double;
            tableCell.BorderWidth = 3;
            tableCell.Text = s;
            TableRow tableRow = new TableRow();
            tableRow.Cells.Add(tableCell);
            this.Table1.Rows.Add(tableRow);
        }
    }
}
```

9. 在 `Text` 属性的 `set` 方法中增加对回文跟踪的支持，并调用 `BuildPalindromesTable` 方法：

```
public string Text
{
    get
    {
        return text;
    }

    set
    {
        text = value;

        this.alPalindromes =
            (ArrayList)this.ViewState["palindromes"];
        if (this.alPalindromes == null)
        {
            this.alPalindromes = new ArrayList();
        }
    }
}
```

```

        if (this.CheckForPalindrome())
        {
            if (PalindromeFound != null)
            {
                PalindromeFound(this, EventArgs.Empty);
            }

            alPalindromes.Add(text);

            this.labelPalindromeStatus.Text =
                String.Format(
                    "This is a palindrome <br/>
                    <FONT size=\"5\" color=\"blue\"><B>{0}</B></FONT>",
                    text);
        }
        else
        {
            this.labelPalindromeStatus.Text =
                String.Format(
                    "This is NOT a palindrome <br/>
                    <FONT size=\"5\" color=\"red\"><B>{0}</B></FONT>",
                    text);
        }

        this.ViewState.Add("palindromes", alPalindromes);
        this.BuildPalindromesTable();
    }
}

```

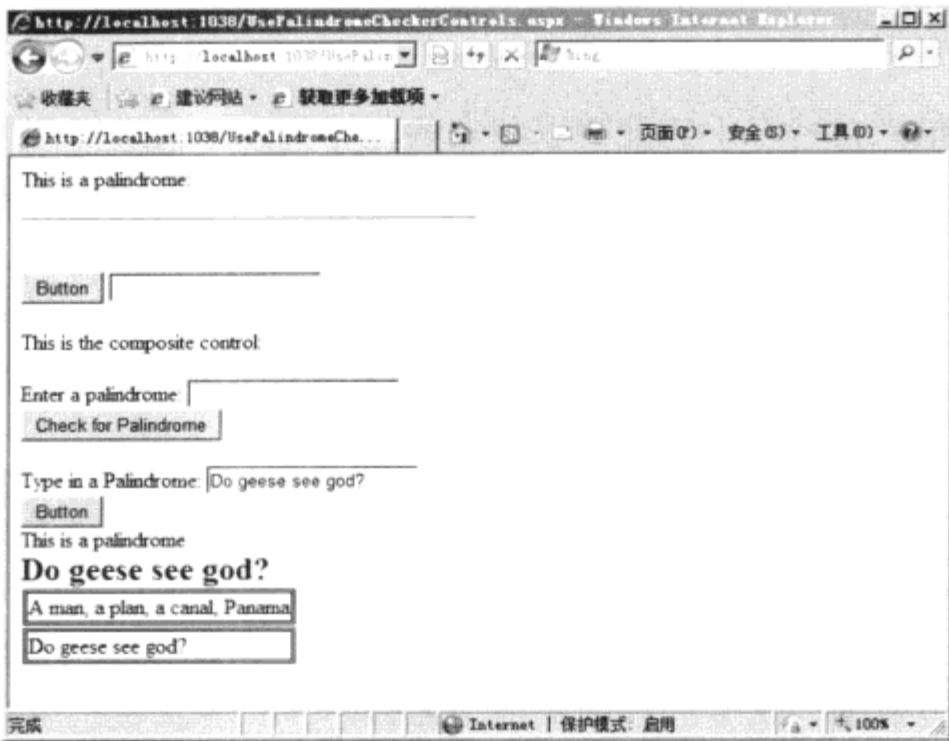
10. 在设计器中双击按钮，为其添加 Click 事件的处理程序。该操作会在关联的代码文件中生成处理程序。在该处理程序中，将 TextBox.Text 属性赋给宿主控件的 Text 属性。这样会设置控件的 Text 属性，并生成回文表格：

```

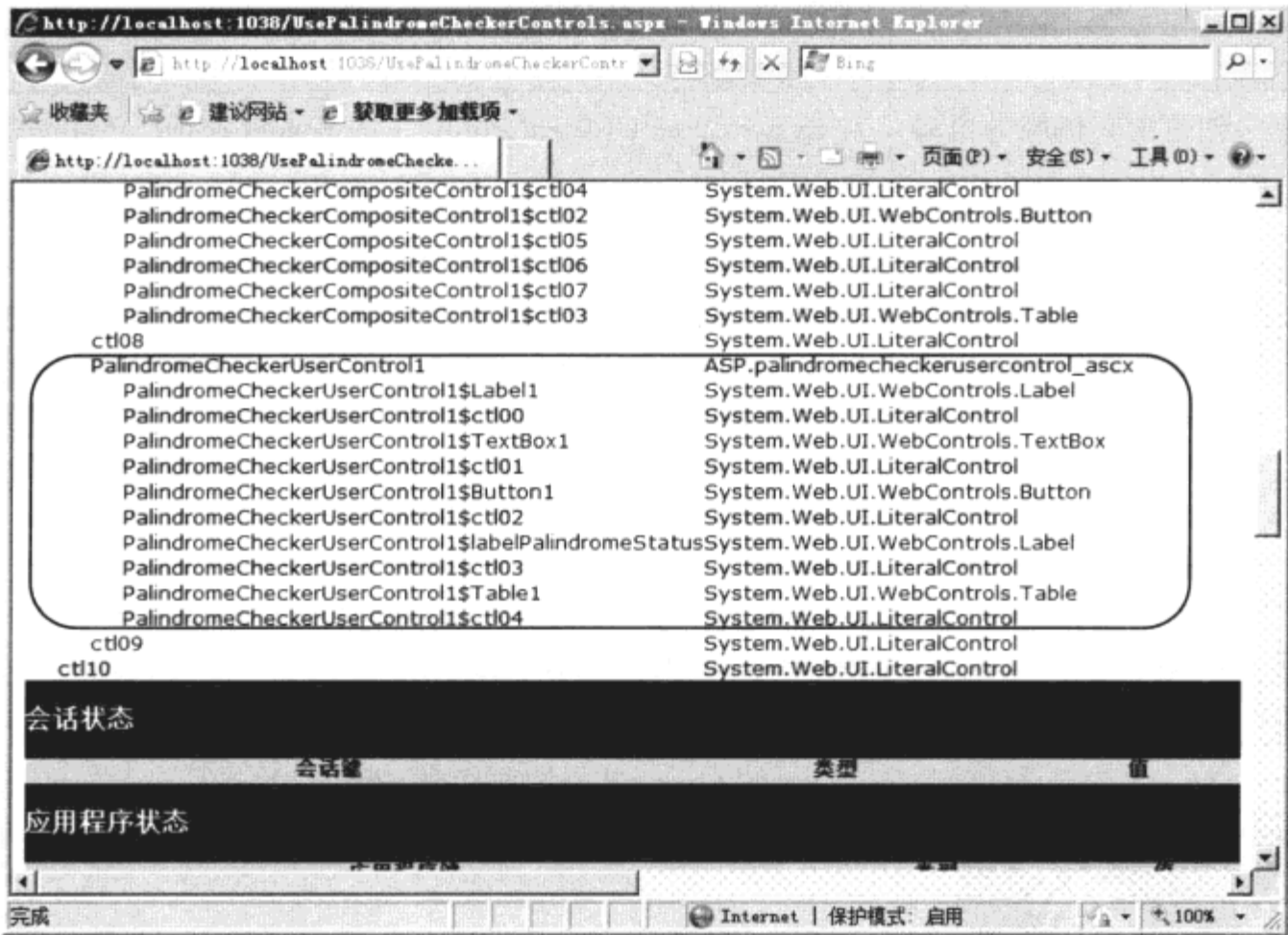
protected void Button1_Click(object sender, EventArgs e)
{
    this.Text = this.TextBox1.Text;
}

```

11. 从“解决方案资源管理器”中将控件的 .ascx 文件拖放到 UsePalindromeCheckerControls.aspx 中，以便将该控件添加到页面上。另外，也可以先将该控件拖放到“工具箱”中，然后再拖放到页面上。最后，可以在上一个控件与新控件之间插入一个换行，以使布局更美观。
12. 生成并运行当前项目。下图展示了向 PalindromeCheckerUserControl 输入几条回文后的页面。



在结束这个示例之前，在 Visual Studio 中打开页面跟踪并按 F5 键，看一下内存中页面/控件的层次结构(见下图)。



不难发现，用户控件非常类似自定义复合控件。两者都是内嵌多个子控件的复合控件，能够非常方便地将丰富的用户界面特性包装成一个单元。

5.4 这两种控件的适用范围

自定义复合控件和用户控件相似，因此二者似乎没必要同时出现在框架中。由于用户控件具有良好的“设计器”支持，所以似乎根本不需要使用自定义复合控件。然而，这两种控

件各有所长，各有所短。

自定义复合控件的最大优势在于它能够以独立的程序集为单位进行部署。由于自定义复合控件包装于单独的程序集中，因此可以对其进行签名，并在企业范围内进行部署。我们还可以将其安装在“全局程序集缓存”(global assembly cache, GAC)中。签名和部署全局程序集是高级话题——在这里提及是为了说明这是自定义复合控件优于用户控件的主要原因之一。自定义复合控件的劣势在于它要求开发者更加注重编码过程的细节(这种控件没有“设计器”支持，需要完全通过代码创建)。

用户控件的主要优势在于它具有“设计器”支持，可以非常方便地通过可视方式设计。然而，用户控件也有缺点：这种控件必须伴随所在项目，实际也要这样被部署。我们可以在其他项目中复用现有的用户控件，但需要将相应的.ascx 和.cs 文件复制到新项目中。此外，这种控件不能通过签名的、安全的程序集进行部署。

5.5 快速参考

目 标	操 作
在程序集中创建由其他服务器端控件复合而成的自定义复合控件	从 System.Web.UI.WebControls.CompositeControl ^① 派生一个类。重写 CreateChildControls 方法。Visual Studio 包含一种能够满足这一需求的项目类型——“ASP.NET 服务器控件”
为自定义复合控件添加子控件	初始化要添加的子控件，并将其添加到宿主控件的 Controls 集合中
将自定义控件添加到“工具箱”中	打开“工具箱”(如果尚未显示，可以在主菜单中依次选择“视图” “工具箱”)。在“工具箱”的任意位置右击，然后在快捷菜单中单击“选择项”。从列表中选择该控件，或通过浏览包含该控件的程序集来添加
使 ASP.NET 为复合控件中的子控件分配唯一的 ID	从 ASP.NET 框架中的 CompositeControl 类派生自定义复合控件。用户控件本身具有该特性
为复合控件添加自定义的事件	通过 event 关键字暴露(使用 public 修饰符)该事件
通过 Visual Studio 的“设计器”来创建复合控件(用户控件)	在 Visual Studio 的网站项目中，根据项目类型在主菜单中选择“项目”或“网站” “添加新项”，然后选择“Web 用户控件”模板

① 译者注：这里对命名空间和类名进行了修正。

第 6 章 常用控件介绍

学习目标

- 使用 ASP.NET 验证控件
- 使用 Image、ImageButton 和 ImageMap 控件
- 使用 TreeView 控件
- 使用 MultiView 控件

ASP.NET 的目标之一是降低开发者构建网站的工作量。纵览 ASP.NET 后便不难发现，Microsoft 为开发者构建了许多可能需要使用的工具，并将其放在框架中。前面 3 章介绍了 ASP.NET “Web 窗体”和控件幕后的架构。理解这种架构有助于扩展这个框架以满足我们的要求。

ASP.NET 的 1.0 版和 1.1 版为开发者提供了许多原来要在传统 ASP 中自行构建的功能。例如，在显示网站用户界面方面，ASP.NET 提供的服务器端控件消除了大量枯燥的编码工作（如下拉列表控件能够保存上次选择的状态）。

ASP.NET 后续的版本得到了进一步扩展，在框架中引入了新的服务器端控件，提供了诸多常用功能。本章将讨论 ASP.NET 对控件中数据的验证支持，并探究其他几种常用控件：几种图片控件、MultiView 和 TreeView 控件。

下面我们先来认识一下什么是验证控件。

6.1 验证控件

ASP.NET 的主要目标之一是提供常见场景所需的功能。例如，一般网站需要的身份验证和授权（本书后面会介绍）。大多数网站在验证访客身份之前不会为其提供有价值的内容。ASP.NET 包含一整套安全基础设施和配套的登录控件^①，使身份验证和授权更为方便。

另一个常见的使用场景是验证网站中用户输入的各种信息。例如，在用户访问某个网站时，可能需要输入用户名和密码。如果该用户订阅某些通过电子邮件分发的内容，那么可能还需要输入其电子邮件地址。

① 译者注：这里所谓的“登录控件”指的不单是登录时输入用户名和密码的控件，而是一个统称，包含用户注册、修改密码、忘记密码、登录信息显示功能的控件。

如果网站的所有者要求访客输入信息，那么网站的程序逻辑就需要确保所获得的信息是有效的。虽然不能确保用户的输入是真实的，但至少可以通过验证输入的字段来获得某些有价值的信息。例如，某些字段是必填的，网站则应确保这些字段填有数据。网站可能要求用户以某种特定的格式输入电话号码，则可以通过正则表达式(regular expression)来验证所输入的信息在格式上是正确的。如果用户要修改密码，则应要求其输入两次新密码，以防止出现输入错误而使用户无法登录。

ASP.NET 包含一些与“Web 窗体”上的标准控件配套的验证控件。这些验证控件能够与标准控件配合使用，在用户输入的信息可能出现错误时发送错误消息(经过配置，还可以弹出提示窗口)。

ASP.NET 包含 6 种验证控件：

- RequiredFieldValidator 能够确保字段必须填入数据
- RangeValidator 能够确保控件的数据在某一特定范围内
- RegularExpressionValidator 能够验证控件中的数据是否匹配某一正则表达式
- CompareValidator 能够通过将控件中的数据与某个值或另一个控件的数据比较来验证数据的有效性
- CustomValidator 允许自定义服务器端和客户端的验证过程
- ValidationSummary 能够显示页面上所有验证错误的摘要

所有验证控件的使用方法基本相同。首先，在页面上定义常规控件。然后，添加相应的验证控件，将其置于希望显示错误消息的位置。验证控件包含一个名为 **ControlToValidate** 的属性，要使其指向被验证的控件，而其他工作会自动被完成。当然，对于来自验证控件的错误消息，还有许多用于定义其显示方式的属性。

ASP.NET 验证控件支持以下服务器端控件：

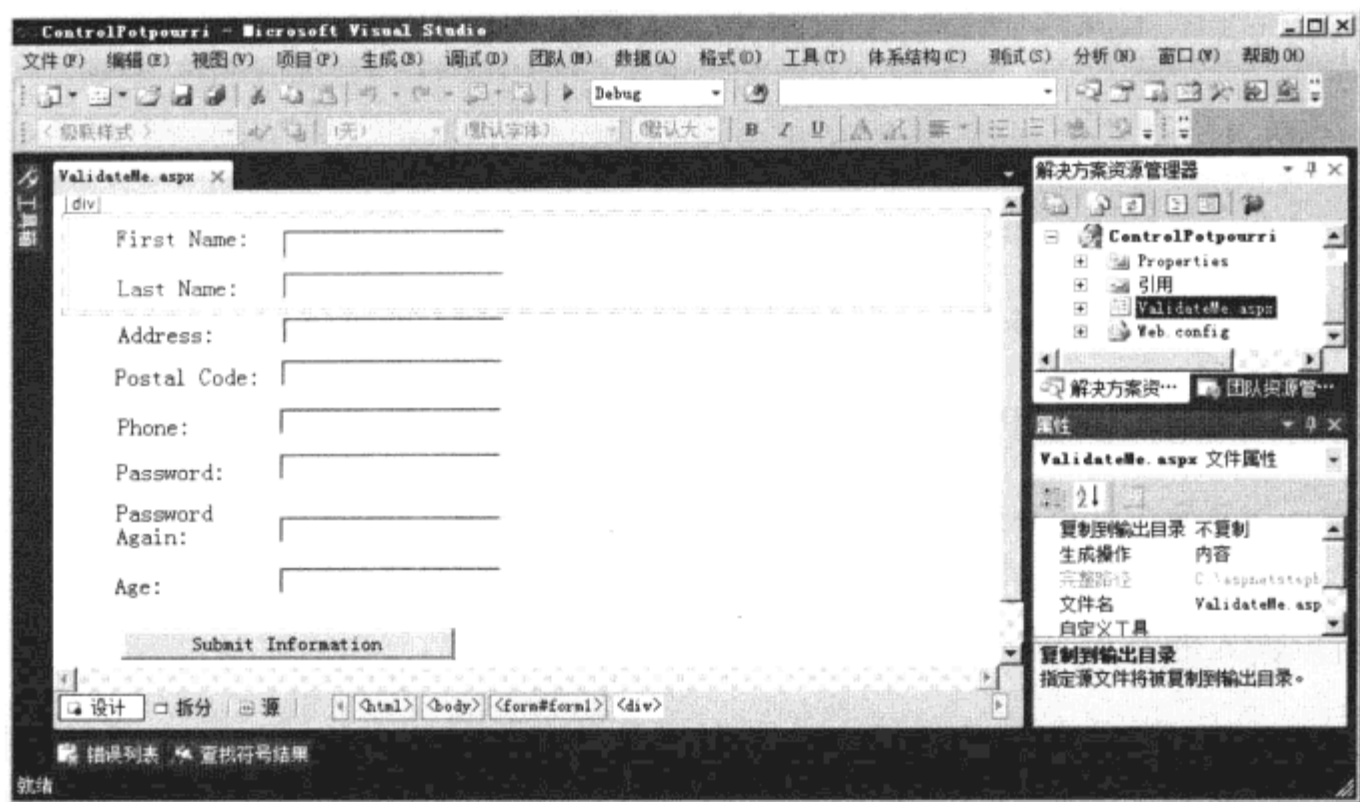
- TextBox
- ListBox
- DropDownList
- RadioButtonList
- HtmlInputText
- HtmlInputFile
- HtmlSelect

- HtmlTextArea
- FileUpload

下面让我们通过示例来了解一下验证控件在“Web 窗体”上的使用方法。

➤ 创建利用验证控件的页面

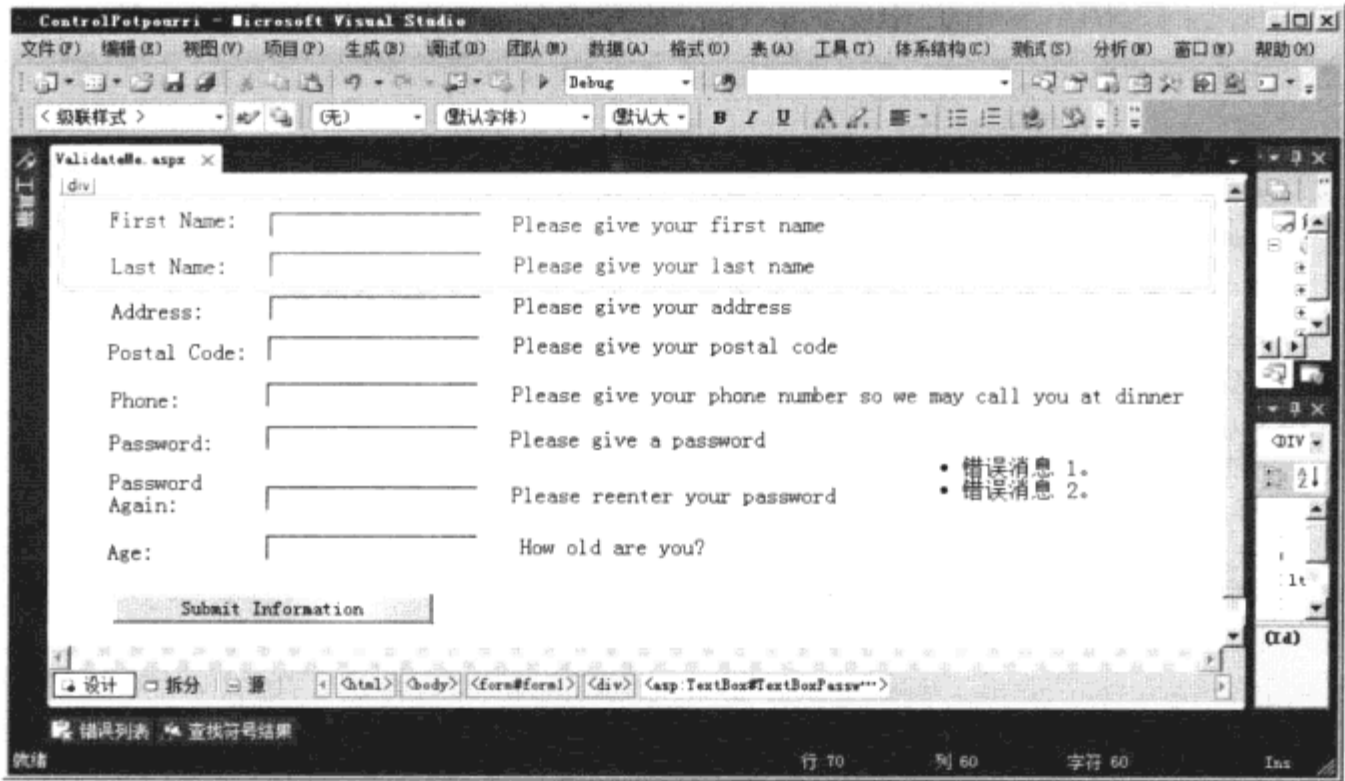
1. 创建一个名为 ControlPotpourri 的“ASP.NET 空 Web 应用程序”。
2. 添加一个名为 ValidateMe.aspx 的“Web 窗体”。下面会在这个窗体上添加常规的服务端控件及相应的验证控件。我们使该窗体提供与一般网站类似的注册功能。这是验证用户输入的常见场景。
3. 添加一个用于输入名字的 TextBox。将其命名为 TextBoxFirstName。最好为控件定义有意义的名称，因为验证控件会通过该名称与被验证的控件关联。如果使用 Microsoft Visual Studio 自动生成的名称(如 TextBox1、TextBox2、TextBox3 等)，则很难记住每种控件的含义，在将其与验证控件关联时便会遇到麻烦。为了窗体的美观，每添加一个文本框意味着要再添加一个与之关联的标签和
元素，这在以下各步骤中不再赘述。在此步骤中，TextBoxFirstName 文本框之前的标签的内容为“First Name:”。其他标签的内容可自己构思。注意，应使标签的 ControlToAssociate 属性指向紧随其后的文本框。这样便能够将标签与文本框关联起来(事实上，标签会通过<label>元素呈现，而不是纯文本)。
4. 添加一个用于输入姓的 TextBox，并将其命名为 TextBoxLastName。
5. 添加一个用于输入地址的 TextBox，并将其命名为 TextBoxAddress。
6. 添加一个用于输入邮政编码的 TextBox，并将其命名为 TextBoxPostalCode。
7. 添加一个用于输入电话号码的 TextBox，并将其命名为 TextBoxPhone。
8. 添加两个分别用于输入密码和确认密码的 TextBox，并分别命名为 TextBoxPassword 和 TextBoxPasswordAgain。将两者的 TextMode 属性设置为 Password 以避免它们向用户显示输入的文本。使用第二个(确认)TextBox 可以保证用户正确地输入了两次密码。(将 TextBox 的 TextMode 属性设置为 Password，用户便无法看到输入的内容。)
9. 添加一个用于输入用户年龄的 TextBox，并将其命名为 TextBoxAge。
10. 向表单添加一个 Button，将其 Text 属性设置为 **Submit Information**。
此时的页面如下图所示。



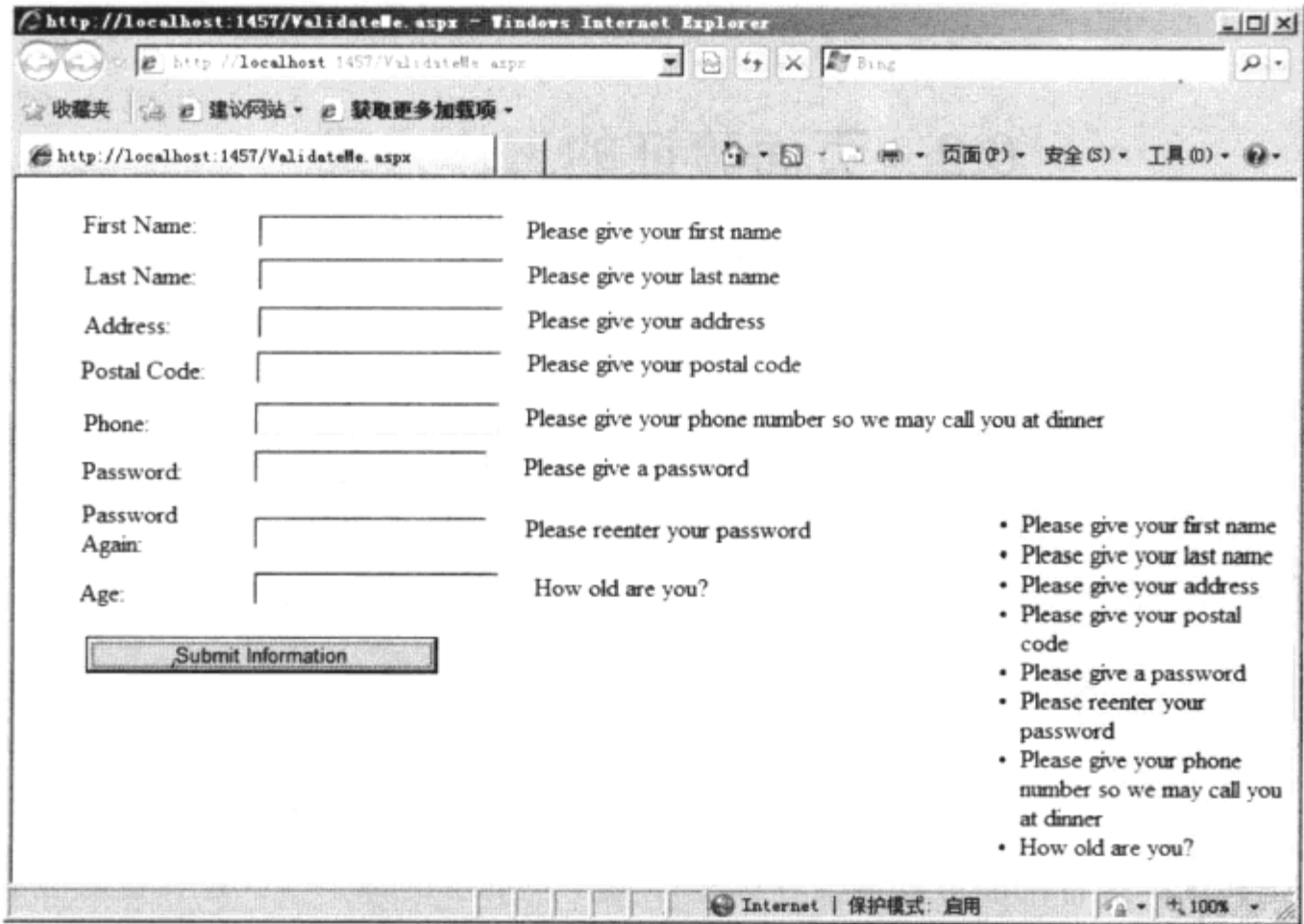
11. 下面开始添加验证控件。在用于输入名字的 `TextBoxFirstName` 控件的右侧添加 `RequiredFieldValidator` 控件的实例。找到这个验证控件的 `ControlToValidate` 属性，将其设置为 `TextBoxFirstName`。将 `ErrorMessage` 属性设置为 “Please give your first name” 或其他有意义的错误消息。
12. 与名字文本框类似，为姓文本框添加 `RequiredFieldValidator` 控件。找到这个验证控件的 `ControlToValidate` 属性，将其设置为 `TextBoxLastName`。将 `ErrorMessage` 属性设置为 “Please give your last name” 或其他有意义的错误消息。
13. 依次为地址、邮政编码、电话号码、密码和年龄文本控件添加 `RequiredFieldValidator` 控件。
14. 与之前步骤相同，对于验证邮政编码的控件，找到其 `ControlToValidate` 属性，并为其选择 `TextBoxPostalCode`。将 `ErrorMessage` 属性设置为 “Please give your postal code” 或其他有意义的错误消息。
15. 对于验证电话号码的控件，将 `ControlToValidate` 属性设置为 `TextBoxPhone`。将 `ErrorMessage` 属性设置为 “Please give your phone number so that we can call you at dinner time” 或其他有意义的错误消息。
16. 在第一个验证密码的控件的属性中，单击 `ControlToValidate` 属性的组合框，选择 `TextBoxPassword`。将 `ErrorMessage` 属性设置为 “Please make up a password” 或其他有意义的错误消息。
17. 在第二个验证密码的控件的属性中，单击 `ControlToValidate` 属性的组合框，选择 `TextBoxPasswordAgain`。将 `ErrorMessage` 属性设置为 “Please confirm your password” 或其他有意义的错误消息。
18. 对于验证年龄的控件，单击 `ControlToValidate` 属性的组合框，选择 `TextBoxAge`。将

ErrorMessage 属性设置为 “Please give your age” 或其他有意义的错误消息。

19. 为表单添加一个 ValidationSummary 控件。该控件会在出现验证错误时显示摘要。如果希望弹出提示框，那么将 ValidationSummary.ShowMessageBox 属性设置为 true 即可。添加并设置完所有验证控件后的页面如下图所示。这个示例的控件采用了绝对定位方式，因而所有控件可以自由移动。

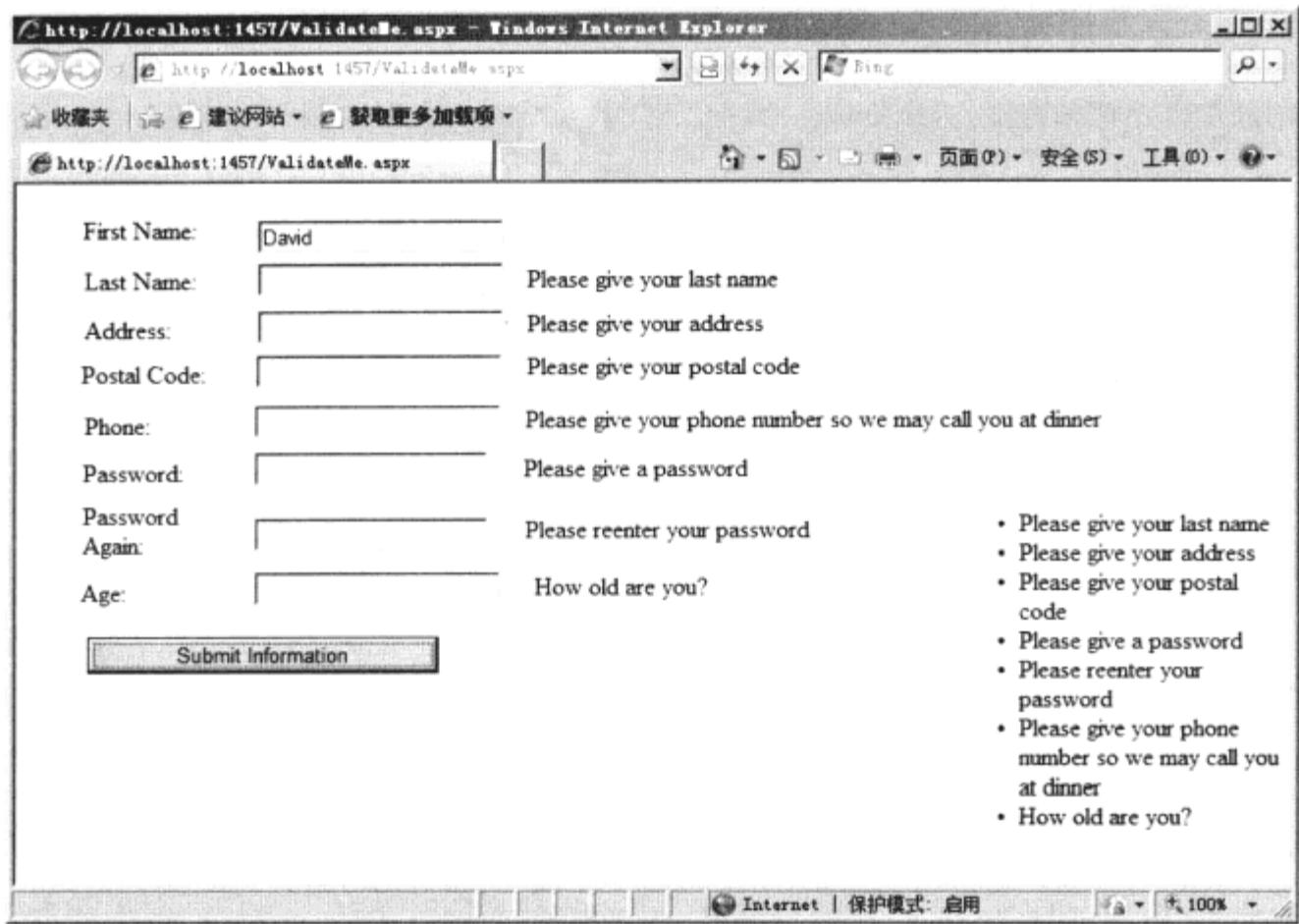


20. 编译项目，并查看该页面。首先，我们会看到所有输入框。在输入任何字段之前，先按一下 Submit Information 按钮。此时页面会显示下图所示的错误消息。



21. 输入名字，然后按回车键。该操作会触发客户端 JavaScript 验证脚本的执行。看看此

时发生了什么。ASP.NET 验证控件会在 HTML 中插入一些 JavaScript，并将其发送到浏览器(这要求浏览器能够解释 JavaScript，当今大多数浏览器都支持)。通过客户端脚本的执行，这些必填字段验证控件(RequiredFieldValidator)便可以显示错误消息，而不用与服务器交互，如下图所示。



在进一步添加其他验证控件之前，让我们先探究一下 ASP.NET 的用户输入验证是如何进行的。

6.2 页面验证的工作方式

ASP.NET 页面验证机制非常智能——所有验证控件都基于服务器端控件架构实现。与 ASP.NET 中的其他功能类似，验证机制针对的是 Web 开发中常见的使用场景。大多数网站都支持客户端验证和服务器端验证。使用客户端验证，则无需在验证用户输入之前与服务器交互。除了客户端验证，大多数网站还支持服务器端验证，原因有二：(1)防止数据在传输过程中被恶意篡改；(2)支持无法执行客户端脚本的客户端(客户端浏览器可能不支持 JavaScript)。下面让我们先看看什么是客户端验证。

6.2.1 客户端验证

如果查看 Visual Studio 根据页面上控件生成的 ASPX 源代码，则会发现页面中满是标签(如支持文本框和选择控件的服务器端控件标签)。置于页面上的每个验证控件也对应着单独的标签。验证控件也是服务器端控件，能够呈现标准的、与浏览器无关的代码——这与一般的服务器端控件一样。

为支持客户端验证，ASP.NET 验证控件会在发送给浏览器的 HTML 源文件中添加 JavaScript 函数。这些客户端验证函数是实现客户端验证所必不可少的。

在验证控件向浏览器呈现自身时，它会在 HTML 的 `span` 元素中添加自定义的属性，并在 HTML 文档加载到浏览器时与对应的控件关联。

客户端验证要求客户端支持 JavaScript，不支持 JavaScript 的客户端只能依赖于服务器端验证。如果愿意，也可以禁用客户端脚本的生成，只需将验证控件的 `EnableClientScript` 属性设置为 `false` 即可。

6.2.2 服务器端验证

在通过客户端验证后，请求会被回发到服务器，服务器端验证便会开始。服务器端验证受 `Page` 类中的基础设施管理。如果在页面中添加了验证控件，那么这些控件会被添加到由页面管理的验证控件集合中。所有验证控件都实现了 `IValidator` 接口。`IValidator` 接口定义了 `Validate` 方法、`ErrorMessage` 属性和 `IsValid` 属性。当然，在判断对应控件中数据的有效性方面，每种验证控件都有自己的逻辑。例如，`RequiredFieldValidator` 会检查所关联的控件是否被填入了数据。`RegularExpressionValidator` 会根据指定的正则表达式对控件中的数据进行校验。

在页面的生命周期中，验证工作会在 `Page_Load` 事件被引发后执行。页面会逐一检查每个验证控件所关联的控件。如果验证失败，那么失败的那些服务器端验证控件则会将自身呈现为可见的 `span` 元素。

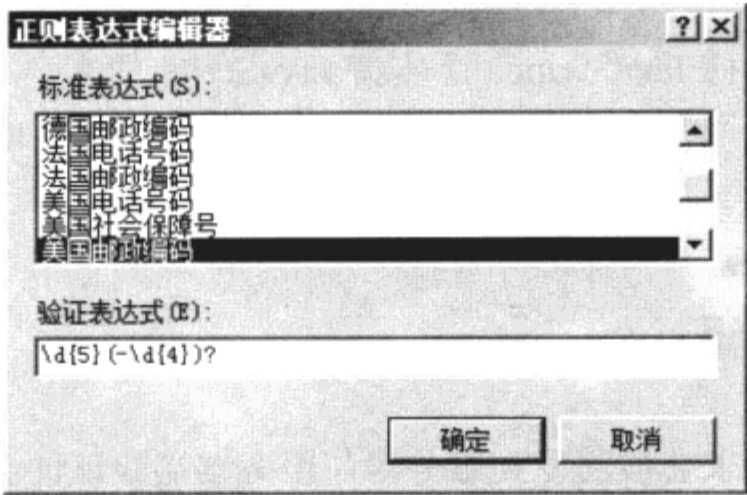
页面自身有一个名为 `IsValid` 的属性，开发者可以在实际使用控件中的数据前，自行检查其有效性。此外，`Page` 类实现了一个名为 `Validate()` 的方法。该方法会遍历所有验证控件，逐一调用每个验证控件的 `Validate` 方法。

在确保用户填写了必填字段后，还有必要确保其输入的数据是正确的。例如，虽然无法确保用户输入真实的电话号码，但可以保证其输入的格式正确，不包含电话号码不应包含的字符。在下面的练习中，我们将添加一个能够根据正则表达式来验证字符串模式的验证控件，并再为页面添加几个其他类型的验证控件。

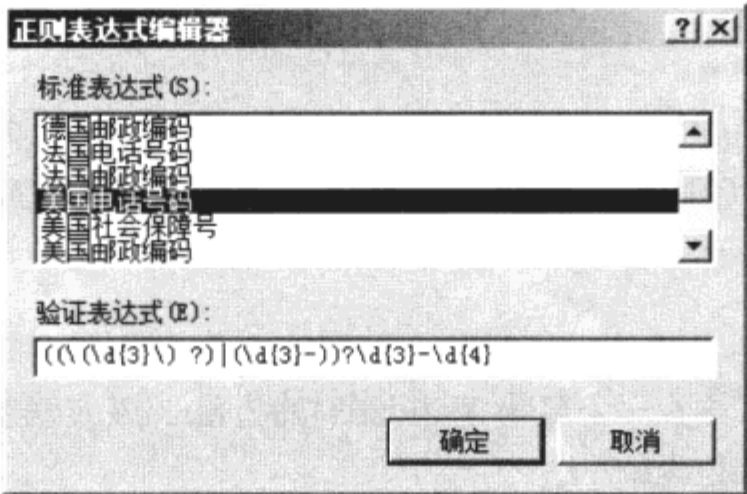
► 对数据做进一步验证

1. 关闭浏览器，回到 Visual Studio 的“设计”窗口。这个页面已包含能够在用户遗漏某些字段时显示错误消息的控件，但用户有可能向窗体中输入无效的数据。下面我们来实现进一步的验证。
2. 我们很难在名字、姓和地址字段上继续“做文章”，只能期望用户填写真实有效的数据。然而，可以通过为 `TextBoxPostalCode` 文本框添加 `RegularExpressionValidator`，以确保用户只在“Postal Code”（邮政编码）字段输入数字。添加这个验证控件，并使

ControlToValidate 指向用于输入邮政编码的文本框。为了显示错误消息，将 ErrorMessage 属性设置为 “The postal code you provided is invalid”。在 “属性” 窗口中，单击与 ValidationExpression 属性关联的按钮。在 “正则表达式编辑器” 对话框中，选择 “美国邮政编码” 作为验证表达式，如下图所示。



- 3. 为 TextBoxPhone 控件添加正则表达式验证控件。将验证控件的 ControlToValidate 属性设置为 TextBoxPhone。将 ErrorMessage 属性设置为 “The phone number you typed is invalid”。打开 “正则表达式编辑器”，选择 “美国电话号码” 作为验证表达式，如下图所示。



- 4. 为 TextBoxPasswordAgain 控件添加 CompareValidator 验证控件。这个验证控件能够验证两次输入的密码是否相同。将其 ControlToValidate 属性设置为 TextBoxPasswordAgain，将 ControlToCompare 属性设置为 TextBoxPassword。将 ErrorMessage 属性设置为 “The passwords provided do not match” 或其他有意义的错误消息。
- 5. 为 TextBoxAge 控件添加一个 CompareValidator 验证控件。将 ValueToCompare 属性设置为 30，将 Type 属性(数据类型)设置为 Integer(整数)。可以将错误消息设置为 “You must be younger than 30 to submit data”。将 Operator 属性设置为 LessThanEqual。
- 6. 生成并运行这个程序。输入一些错误的的数据看看会发生什么。在这种情况下，验证控件应给出相应的错误消息。例如，如果年龄输入 33，TextBoxAge 控件对应的 CompareValidator 验证控件则会显示错误消息，因为正确值应小于或等于 30。

6.3 其他验证控件

除了之前提到的验证控件，ASP.NET 中还包含其他 3 种验证控件：RangeValidator、CustomValidator 和 DynamicValidator。本节将简单介绍一下这三种控件。

RangeValidator 与 CompareValidator 类似，也能够将控件中的数据与数值进行比较。不过，RangeValidator 会在数据超出某一范围时报告错误。我们可以为这种验证控件指定一个最小值和最大值。如果控件中的数据超出其中的某个阈值后，该验证控件则会报告错误。

如果来实现其他类型的验证逻辑，则可以使用 CustomValidator。在页面中，我们可以像其他验证控件那样来使用 CustomValidator。然而，该控件没有预先定义验证逻辑(服务器和客户端都没有)，而是留给开发者定义。在页面上添加 CustomValidator 后，可以将其与目标控件关联，然后引用验证函数(需要编写服务器端代码)。也可以指定发送给客户端的验证脚本块，使其与其他客户端验证脚本一起执行。

最后，为支持“动态数据”(Dynamic Data)模型，ASP.NET 引入了一个新的验证控件——DynamicValidator。第 20 章会介绍“动态数据”。ASP.NET 的“动态数据”模型支持数据驱动的应用程序开发。DynamicValidator 控件能够捕获数据绑定和验证过程中出现的所有异常，并将这些异常转发给页面的验证事件。

6.4 验证控件的属性

验证控件包含其他 ASP.NET 标准控件具有的标准属性。例如，Text、Font 和各种用于设置外观的属性。此外，对于向浏览器发送的错误消息，验证控件也有对其进行管理的属性。

Display 属性的值可以是 Static(静态)或 Dynamic(动态)。该属性能够决定错误消息在客户端的显示方式。Static(默认值)会使验证控件为客户端生成的 span 元素事先预留显示错误消息的布局空间，即便这些消息未被显示。如果将 Display 属性设置为 Dynamic，验证控件生成的 span 元素则不会预留空间，而是在消息显示时展开。

ASP.NET 还能够对验证控件进行分组。也就是说，每个验证控件可以从属于某个指定的组。为此，要将 ValidationGroup 属性设置为组别的名称。如果某个控件属于一个组，那么该组中任何一个控件引发验证过程，整个组的所有控件都会被验证。^①这使得页面具有“多表单”效果。

下面几节将介绍另外几种有趣的控件：3 种基于图片的控件、TreeView 和 MultiView 控件。

^① 译者注：分组还可以隔离不相关的控件，使验证的触发只在组内进行。

6.5 基于图片的控件

网页中经常使用图片。HTML 包含一个图片标签, 可以使浏览器获取图片文件(如.gif、.jpg 或.png 文件)并将其显示出来。如果要在页面上添加图片, 可以使用 HTML标签。不难想到, ASP.NET 会通过服务器端控件来包装标签——这个服务器控件就是 Image。

Image 控件的使用非常简单。与其他控件类似, 在“工具箱”中选择 Image 控件, 然后将其拖放到页面上即可。ASP.NET 中的 Image 控件会生成带有 src 属性的标签。

除了一般的 Image 控件, ASP.NET 中还有 ImageButton 控件和 ImageMap 控件。ImageButton 控件是对<input type=image />标签的包装, 能够将指定的图片作为按钮的背景。ImageMap 控件能够显示带有“作用点区域”(hot spot region)的图片。

下面我们看看如何使用 ASP.NET 中各种基于图片的控件。

➤ 在页面中使用图片控件

1. 在项目中添加一个“Web 窗体”来放置图片控件, 将该页面命名为 UseImageControls.aspx。
2. 从“工具箱”拖放一个 Image 控件到页面上。
3. 在“属性”窗口中, 将 ImageUrl 属性设置为一个有效的图片路径。该图片文件可以位于本地计算机, 也可以使 ImageUrl 属性指向网络上某个有效的 URL。为使用网络上的图片, 可以在浏览器中右键单击图片, 然后选择“属性”, 再将“属性”窗口的 URL 复制到控件的 ImageUrl 属性中。如果使用本地计算机上的文件, 则需要事先添加到项目中——将文件从本地磁盘拖放至“解决方案资源管理器”的 ControlPotpourri 解决方案中。如果希望将图片组织到单独的文件夹中, 可以创建一个新文件夹, 将图片拖放进去即可。如果使用来自网络的图片, 则可以在“源”视图中手动编辑 ImageUrl 属性。显然, 不论使用哪个 URL, 如果浏览器无法找到图片(导致标签出现错误), 它便会提示标准的“无法找到图片”错误消息。在 Microsoft Internet Explorer 中, 该提示表现为中间带有一个红色 X 的方框。
4. 运行这个网站, 查看 ASP.NET 中的 Image 控件生成的内容。(实际的 URL 可能与本示例中的不同。)

```

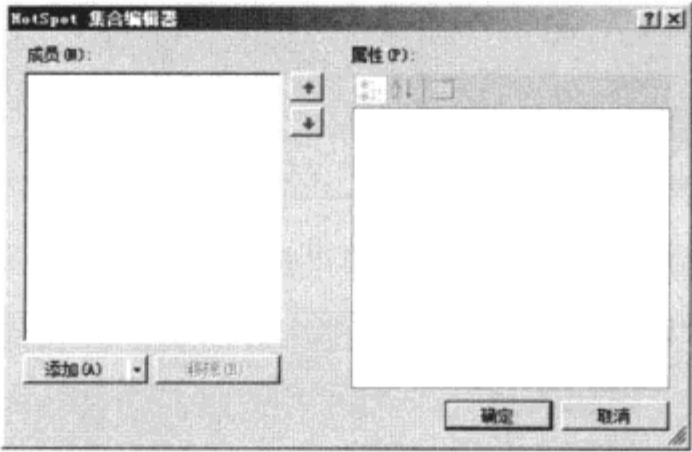
```

5. 向页面添加一个 ImageButton 控件。该控件可以将普通的输入按钮显示为图像。应用程序响应 ImageButton 的方式有 3 种: (1)可以在服务器端为 ImageButton 添加 Click 事件的处理程序, 使它像一般的按钮那样工作。(2)可以定义客户端脚本块, 将 ImageButton 的 OnClientClick 属性指向该脚本。当用户单击按钮后, 单击操作会在浏览器中触发脚本的执行。(3)可以通过 ImageButton 的 PostBackUrl 属性将请求重定向到指定的页面。

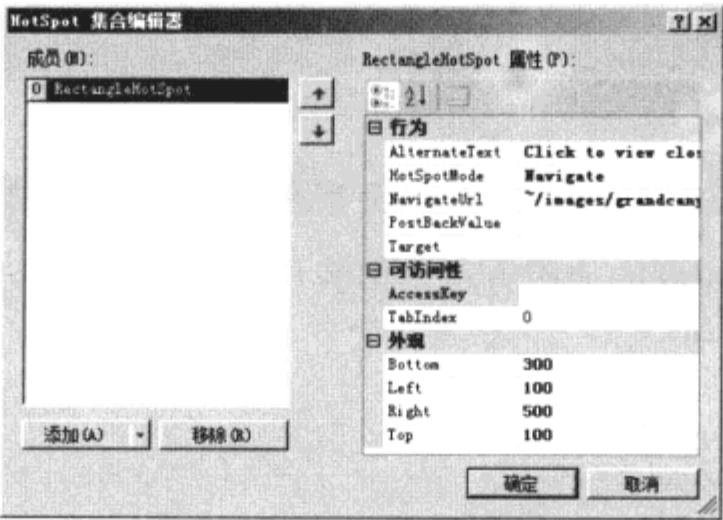
6. 运行当前页面，查看 ImageButton 控件生成的 HTML。该控件应生成类似这样的标记(实际的 URL 可能会不同)：

```
<input type="image" name="ImageButton1" id="ImageButton1"
src="Images/goldengatebridge.jpg" style="border-width:0px;" />
```

7. 向页面添加最后一个控件——ImageMap。该控件能够在图像上定义可点击的区域。找一张图片(可以从网上下载，也可以在本地硬盘上查找)。使 ImageMap 的 ImageUrl 属性指向该图片文件。
8. 使用图片编辑器(如 Windows 中的“画图”或 Visual Studio 中的“位图编辑器”)打开为 ImageMap 设置的图片。在本示例中，我们将为 ImageMap 定义作用点区域，使它能够像地图一样放大图片的局部。在图片中选择一个矩形区域，记录下这个矩形区域四角的坐标。用选中的这一区域创建新的图片，将其放大，并保存到新的文件中。
9. 下面我们为 ImageMap 定义作用点区域。ImageMap 中有一个叫做 HotSpots 的属性。单击该属性字段对应的按钮。此时会打开“HotSpot 集合编辑器”，如下图所示。



10. 单击“添加”按钮便可以向集合中添加作用点区域。注意，我们可以通过“添加”按钮的下拉列表来定义圆形、矩形和多边形的作用点区域。根据刚刚记录的图片坐标创建一个矩形作用点区域。为 AlternateText 属性设置一段文本——这段文本会在工具提示(tooltip)中显示。将 HotSpotMode 属性设置为 Navigate，通过“选择 URL”窗口使 NavigateUrl 属性指向新建的图片文件(为手动添加新图片文件，可以在“解决方案资源管理器”中右键单击当前项目，然后选择“添加”|“现有项”)。作用点区域的定义如下图所示。



ASP.NET 4从入门到精通

11. 在添加作用点区域后，运行页面。此时应可以看到如下图所示的图片——本示例选择了一张“大峡谷” (Grand Canyon)^① 的图片，其中的作用点区域已用红色的矩形框标出。(这个矩形框是手动画上去的，作用点区域不会自动添加这种图形。)注意显示的工具提示。



12. 单击作用点区域，应用程序会被重定向到“放大”的图片，如下图所示。



本节只是简单介绍了图片控件的使用方法，但我们会发现定义图片的外观和行为是非常灵活的。

^① 译者注：位于美国亚利桑那州北部的科罗拉多。

6.6 TreeView

在现代软件中，最常见的用户界面元素之一是通过可展开的节点呈现的层次型结构。例如，在使用“Windows 资源管理器”浏览某个文件时，则可能需要展开或折叠文件夹(子文件夹)来查看其内容。这种控件一般被称为“树控件”(tree control)。

使用树控件，用户可以通过可展开的和可折叠的节点来对层次型结构进行导航。例如，使用“Windows 资源管理器”浏览 C 盘内容，如果使用“Windows 经典”主题，则会看到有许多文件夹的左侧带有小加号。当单击加号后，“Windows 资源管理器”会打开这个文件夹，然后列出该文件夹下的子文件夹。如果还有嵌套的子文件夹，则能够以相同的方式打开它们。使用其他主题的话，展开和折叠节点的方式会稍有不同。

ASP.NET 通过 TreeView(树视图)控件提供了这种功能。在需要显示嵌套的数据结构时可以使用该控件，它提供了一种向下钻取(drill down)的方法。

下面的练习通过层次型的、可展开的列表演示了 TreeView 控件的使用方法，其中的数据是 20 世纪 70 年代成立且至今仍存在的 3 个团队。这个示例能够展示各团队的层次结构——每个团队的名称下面是成员姓名，而每个成员下面是其所承担的角色。

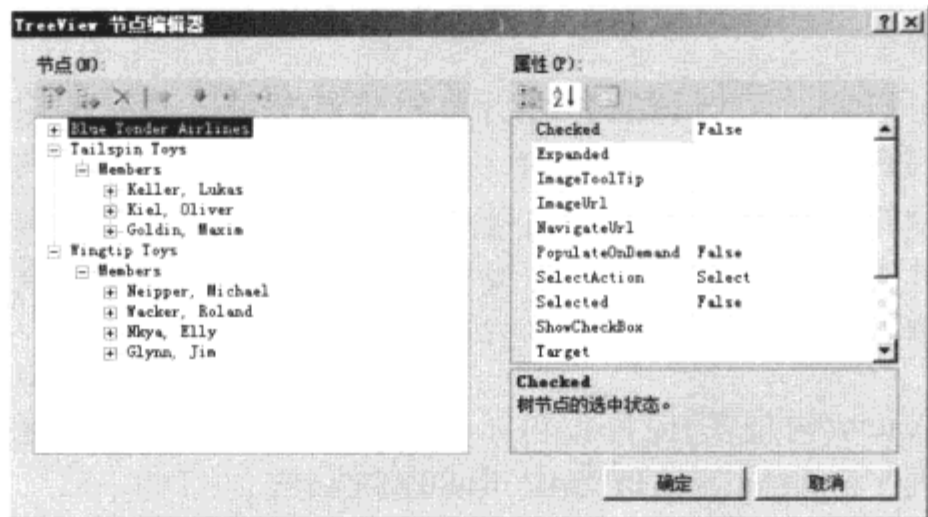
➤ TreeView 控件的使用

1. 为 ControlPotpourri 网站添加一个“Web 窗体”，将其命名为 UseTreeView。
2. 从“工具箱”拖放一个 TreeView 到默认打开的页面上。可以在“导航”选项卡中找到该控件。
3. 下面我们通过 Visual Studio 提供的选项来设置树视图控件的格式。在“设计”视图中单击 TreeView 控件的智能标签，在弹出的“TreeView 任务”中选择“自动套用格式”。此时会显示一个包含许多样式的对话框。单击几个其中的样式，浏览其效果。下图展示了带有“自动套用格式”链接的“TreeView 任务”菜单。

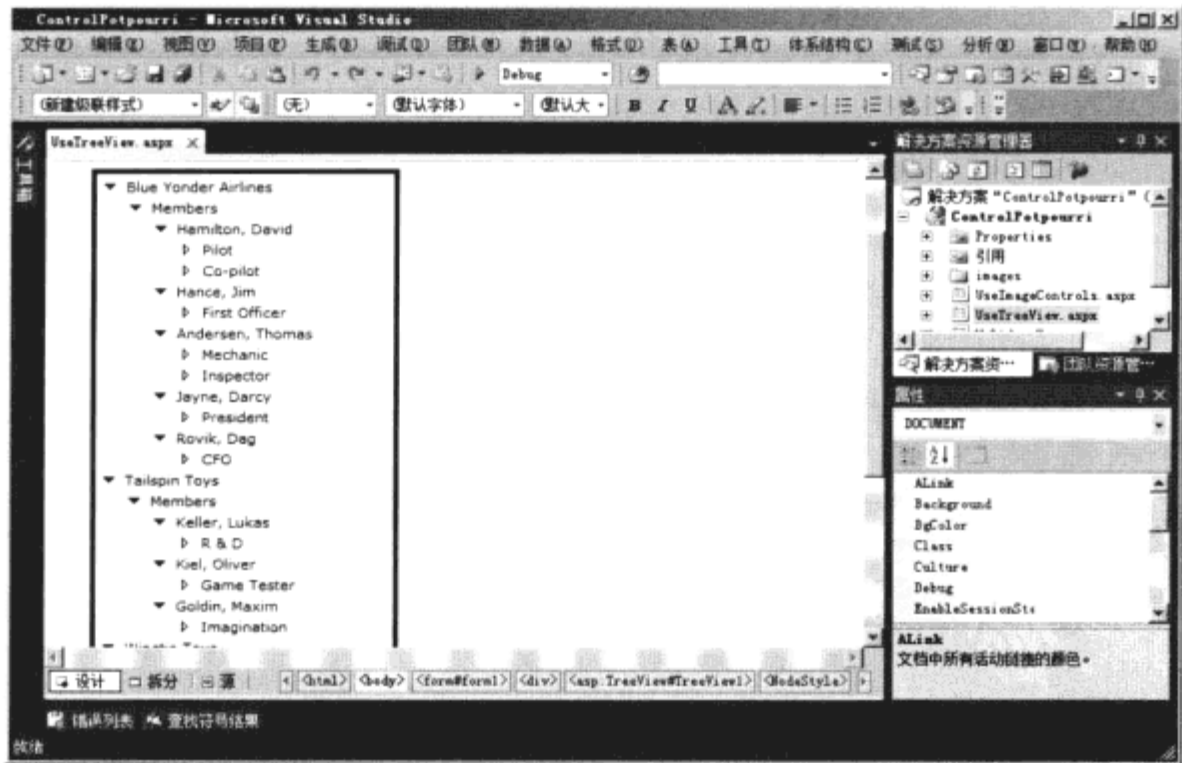


① 译者注：这里经作者确认对原图作了修正。

4. 右键单击 TreeView 控件，选择“编辑节点”。此时会打开“TreeView 节点编辑器”。我们可以通过这个编辑器来编辑每个节点。最左端的按钮用于添加根节点，第二个按钮用于添加子节点。节点可以任意嵌套。在这个示例中，我们用根节点来表示团队，用第二层的子节点表示团队的成员，用第三层子节点表示其角色。下图展示了“TreeView 节点编辑器”。



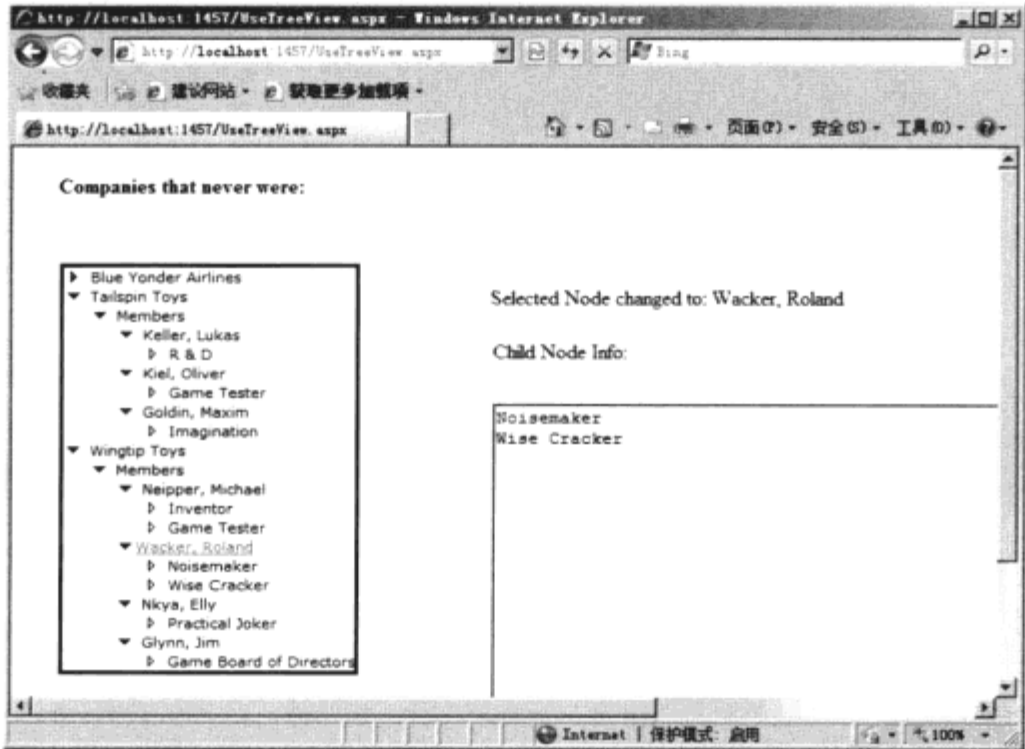
5. 使用 BorderStyle 和 BorderColor 属性为 TreeView 添加边框。将边框样式设置为 Solid，将颜色设置为 Black。当然，这只是为了美观。
6. 生成项目并浏览该页面。如果选用了“箭头”这样的格式，则应能够展开或折叠节点。在页面运行起来后，浏览一下 ASPX 源代码，看看 TreeView 是如何组织其节点的。下图展示了 TreeView 在浏览器中显示的效果。



7. 为使其更丰富，我们在树视图的事件处理程序中添加一些功能。首先，添加一个显示选定节点名称的标签，将其命名为 LabelSelectedNode，以便我们能够以编程方式访问它。添加一个显示选定节点详细信息的 TextBox，将其命名为 TextBoxInfo。将这个 TextBox 的 TextMode 属性设置为 MultiLine。为 TreeView 的 SelectedNodeChanged 事件添加处理程序。添加以下代码，列出选定节点的子节点的信息。不要忘记使用 using 语句添加对 System.Text 的引用(因为这段代码使用了 StringBuilder 类)：

```
protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e)
{
    this.LabelSelectedNode.Text =
        String.Format("Selected Node changed to: {0}",
            this.TreeView1.SelectedNode.Text);
    TreeNodeCollection childNodes =
        this.TreeView1.SelectedNode.ChildNodes;
    if (childNodes != null)
    {
        this.TextBoxInfo.Text = String.Empty;
        StringBuilder sb = new StringBuilder();
        foreach (TreeNode childNode in childNodes)
        {
            sb.AppendFormat("{0}\n", childNode.Value);
        }
        this.TextBoxInfo.Text = sb.ToString();
    }
}
```

选定节点的详细内容会被显示在 TextBox 中，如下图所示。



这里只演示了 TreeView 的一小部分功能。除了使用“设计器”添加节点外，还能够以编程方式添加。此外，也可以在程序中展开和折叠节点。TreeView 还支持数据绑定，因而能够正确呈现层次型的数据结构。

最后，让我们看看 ASP.NET 中的 MultiView 和 View 控件。

6.7 MultiView

有时我们可能需要将一些控件组织到几个窗格中，让用户翻查。当 ASP.NET 1.0 还流行时，Microsoft 发布了几个特性丰富的动态控件(但不受官方支持)，能够生成动态 HTML(DHTML)

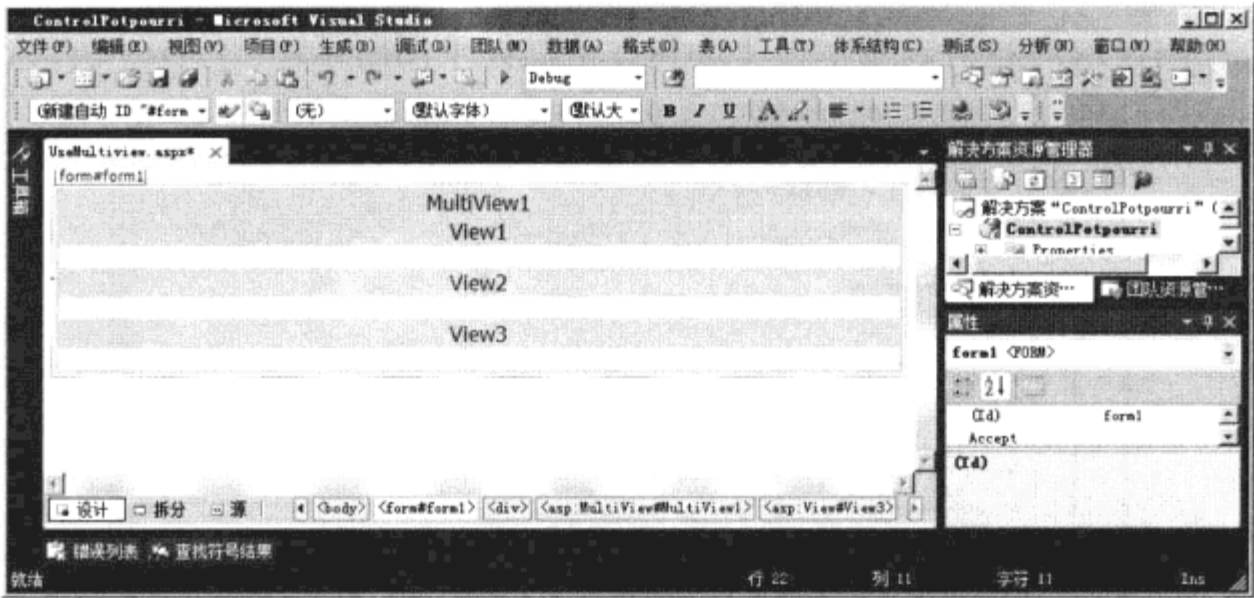
来取代一般的 HTML。其中有一组控件——TabStrip、MultiView(旧版本)和 PageView——能够实现选项卡式的窗格。

这组控件未在 ASP.NET 的后续版本中出现，但出现了两个控件(MultiView 和 View)，可以提供类似的功能。MultiView 用作窗格式控件(View 控件)的容器，用户可以在其中的不同 View 之间进行切换。MultiView 每次只显示一个 View。

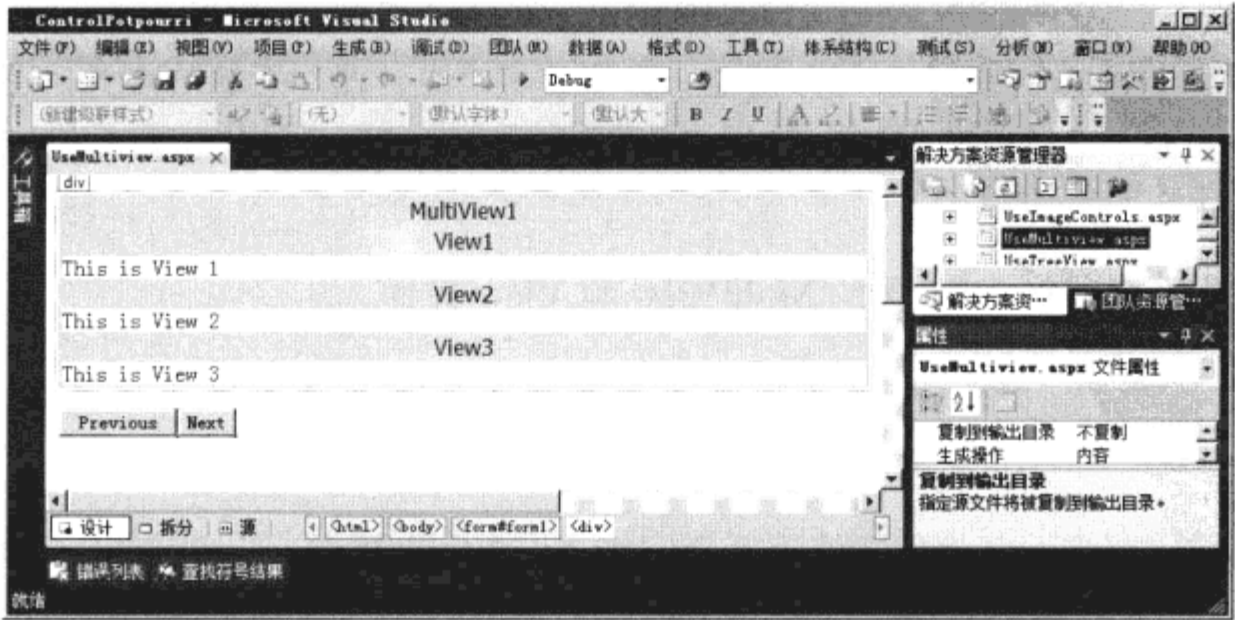
下面通过练习给出一个 MultiView 和 View 结合使用的示例。

➤ MultiView 和 View 控件的使用

1. 为 ControlPotpourri 网站添加一个“Web 窗体”，将其命名为 UseMultiview.aspx。
2. 向“Web 窗体”添加一个 MultiView 控件。
3. MultiView 的主要目的是用于管理内部所有的 View。为向 MultiView 添加 View，可以从“工具箱”拖放 View 的实例到 MultiView 的“内部”。按下图所示添加 3 个 View 到当前“Web 窗体”中。为容纳多个视图，可能需要手动调整容器的大小。



4. 为每个 View 控件添加一些内容。可以认为 View 与窗格非常类似。在这个示例中，每个视图所包含的是用于标识自身的标签。下图展示了此时每个 View 在“设计器”中的效果。

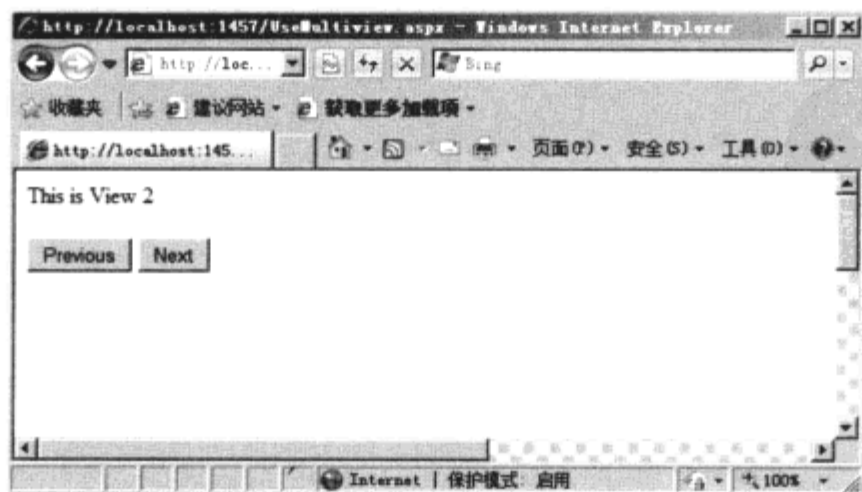


5. 激活第一个窗格。为使 MultiView 中的第一个 View 显示出来，需要将 MultiView 的 ActiveViewIndex 设置为 0。
6. 为在 MultiView 的不同 View 之间进行导航，在表单底部添加两个按钮，分别命名为 ButtonPrev 和 ButtonNext。
7. 分别双击这两个按钮，为它们添加事件处理程序。
8. 添加以下切换 View 的代码。这段代码通过改变当前 View 的索引来响应按钮的单击：

```
protected void ButtonPrev_Click(object sender, EventArgs e)
{
    if (MultiView1.ActiveViewIndex == 0)
    {
        MultiView1.ActiveViewIndex = 2;
    }
    else
    {
        MultiView1.ActiveViewIndex -= 1;
    }
}

protected void ButtonNext_Click(object sender, EventArgs e)
{
    if (MultiView1.ActiveViewIndex == 2)
    {
        MultiView1.ActiveViewIndex = 0;
    }
    else
    {
        MultiView1.ActiveViewIndex += 1;
    }
}
```

9. 编译项目并浏览当前页面。单击导航按钮会触发向服务器的回发，使服务器呈现每个视图。下图展示了 MultiView 和第二个 View 在浏览器中的效果。



正如您所看到的，MultiView 和 View 能够像窗格一样来回切换。它们提供了一种方法来管理包含大量数据的界面区域。这类控件的另一个例子是配合会话状态实现的 Wizard(向导) 控件。

6.8 快速参考

目 标	操 作
验证表单的输入	<p>对于服务器端控件，ASP.NET 包含许多能够校验其中数据的验证控件，如下所示：</p> <ul style="list-style-type: none">• CompareValidator• RangeValidator• RequiredFieldValidator• RegularExpressionValidator• ValidationSummary• CustomValidator <p>为验证服务器端控件的输入，将适当的验证控件拖放到页面上，将 ControlToValidate 属性设置为目标控件的 ID，并合理设置其他属性</p>
直观地显示层次型数据	<ul style="list-style-type: none">• 使用 TreeView 控件• 可以手动添加项或将 TreeView 绑定到层次型数据源。第 12 章会介绍如何用该控件来进行导航
在同一网页上切换不同信息页	<ul style="list-style-type: none">• 使用 MultiView 和 View 控件• 可以将 View 看作小型页面的管理控件• MultiView 用于管理 View 集合，并支持在不同 View 之间切换
在网页中添加图片	<ul style="list-style-type: none">• 在页面上添加 Image 控件• 将 Image 控件的 ImageUrl 属性设置为待显示图片的 URL
在网页中添加带有可点击区域的图片	<ul style="list-style-type: none">• 在页面上添加 ImageMap 控件• 使用“HotSpot 集合编辑器”来定义可点击的区域



第 II 部分

高级特征

- ▶ 第 7 章 一致的界面
- ▶ 第 8 章 配置
- ▶ 第 9 章 登录
- ▶ 第 10 章 数据绑定
- ▶ 第 11 章 网站的导航
- ▶ 第 12 章 个性化
- ▶ 第 13 章 Web 部件

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569



第 7 章


一致的界面

学习目标

- 通过母版页使整个网站具有一致的界面
- 使用主题来设置网站中控件和页面的样式
- 使用皮肤来设置自定义控件的样式

当今，设计精良的网站与其他网站最大的区别是拥有一致的观感(look and feel)。例如，许多网站都有自己的颜色主题和字体组合。此外，设计精良的网站在信息的组织方式和导航工具方面，也能够不同页面间保持一致。网站中每个页面的外观完全不同，用户会怎么想？轻则使人困惑，重则让人反感。

本章将介绍如何为网站提供一致的观感，这是 ASP.NET 中最具特色的功能之一——母版页。母版页可以帮助开发者为网站提供一致的外观。ASP.NET 还提供了一种为控件提供特定样式的方法。本章将一并介绍这些特性的使用。

 **注意** 安装本书的示例代码要求用户具有计算机的 Administrator 权限。如果使用家用计算机，则可能已具有 Administrator 权限。如果使用某个组织的计算机，并且没有 Administrator 权限，则需要咨询计算机支持人员或 IT 人员。相关内容，请参考本书“前言”部分“示例代码”一节。

7.1 用户界面一致性的管理

Web 开发工具从最初出现到为一致观感提供有力支持，这个过程历经了很长时间。传统的 ASP 为实现一致的观感需要使用一种文件包含机制，将 .asp 文件拆分成大量碎片。这种方式十分原始，至少可以说这很费事。虽然这能产生一定效果，但既要文件组织好，又要控制好网站的细节，这是很困难的。

ASP.NET 1.0 得到了改进，它通过粒度更小的服务器端控件和用户控件来支撑整个页面呈现机制。这方面内容已经在第 2 章和第 3 章介绍过。不论怎样，虽然能够将 Web 应用程序的不同部分封装到各自的模块中，但仍需要为应用程序中的页面实现一致的观感做一定的工作。例如，可以通过用户控件来实现一致的观感。在用户控件中添加导航控件和链接，然后在网站中所有页面的同一位置放置该控件。这个控件本身实现了一致的观感，但是否在网站中使用它是开发者的事，控件的开发者也没有办法保证该控件在 Web 应用程序中的每个页面都以相同的方式被使用。

虽然使用自定义控件或用户控件将用户界面拆分成块有助于实现一致的 UI，但这种方式存在几个缺陷。首先，应用程序中的所有页面都需要包含外围代码。也就是说，需要以相同的方式来为每个页面添加这种控件(正如刚刚提到的)。如果要修改控件的位置(或其他不受这些控件本身掌控的特性)，则必须修改所有页面。其次，使用自定义控件的页面都需要添加 `Register` 指令，为此需要更多重复的代码。相比之前的方法(即传统 ASP 所采用的)，这种可复用的模型有了很大进步。但我们真正需要的是在网站中的某一处进行修改就能一次性地更改网站中所有页面的观感。

实现此目标并避免逐一构建页面的方法之一是实现一个基类，然后使应用程序中的所有页面都从此派生。既然 ASP.NET 构建于基于 `Page` 类的对象模型上，那么为什么不能再为应用程序添加一个层次呢？图 7.1 展示了这种类层次关系。

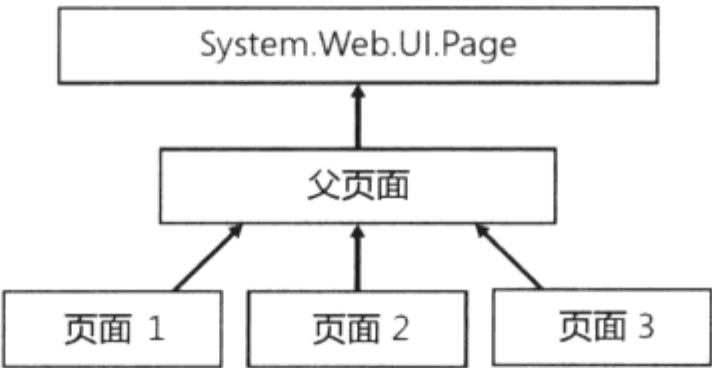


图 7.1 为多个页面实现公共功能的基类

所有 `.aspx` 页面都继承于相同的代码隐藏类，而这个代码隐藏类则派生自父页面(父页面派生自 `System.Web.UI.Page`)。父页面负责加载网站外观所必需的控件，然后不同的页面负责剩余的工作。

这种方法切实可行，只要您不介意编写大量代码。此外，ASP.NET 1.x 没有针对这种特性的设计支持，而且在 Microsoft Visual Studio 中修改 `Page` 类的层次结构，有时会造成项目编译失败。

为更方便地使网站具有一致的观感，ASP.NET 2.0 引入了母版页。而这在使网站中的不同页面具有一致性方面，已成为许多开发者的主要工具。

7.2 ASP.NET 母版页

母版页是一种配置页面，其结构与一般的页面非常类似。不过，母版页文件的扩展名为 `.master`。母版页是一种模板，能够为依赖于它的所有页面呈现相同的外观。母版页可以包含 XHTML 文档标签(如 `<html>`、`<head>` 和 `<body>`)，这些标签只应用于母版页。当访问者浏览某个依赖于母版页的页面时，请求和响应也会流经该母版页。母版页本身是不能工作的，它只是用来保证每个页面具有一致的观感，它从逻辑上对应于图 7.1 中的“父页面”。从这个角度上讲，此类便像从 `System.Web.UI.Page` 派生的其他所有类一样处理请求和呈现输出。

母版页与一般的.aspx 页面一样，可以包含一般页面能够包含的所有内容和功能。换言之，母版页可以包含服务器端控件、用户控件和文本标记。除了一般的控件和标记，母版页还可以包含 `System.Web.UI.WebControls.ContentPlaceHolder`(内容占位符)控件的实例。顾名思义，内容占位符所在的位置最终会显示基于母版页的页面所提供的实际内容。母版页会呈现它所包含的所有元素——所有未包含在 `System.Web.UI.WebControls.ContentPlaceHolder` 控件中的元素。

由于母版页要参与最终页面处理程序的融合过程，因此母版页的工作方式不同于之前提到的直接继承技术(即通过基类来实现公共的功能)。在页面执行时，母版页会将自身内容注入到.aspx 页面中。具体来说，母版页中的控件最终会被添加到.aspx 页面的 `Controls` 集合中，这些控件的呈现方式与其他控件的呈现方式完全相同。

与一般的页面类似，母版页可以在 `MasterPage` 指令中包含以下属性：

- `AutoEventWireup`
- `ClassName`
- `CompilerOptions`
- `Debug Description`
- `EnableViewState Explicit`
- `Inherits`
- `Language`
- `Strict`
- `Src`
- `WarningLevel`
- `Master`

下面的练习演示了如何围绕母版页来开发网站。

➤ 使用母版页

1. 创建一个名为 `MasterPageSite` 的“ASP.NET 空 Web 应用程序”。通常而言，创建一个新的网站项目即可，但本例中应确保使用“空”的网站模板，以便之后手动添加母版页和内容页。不要使用 Visual Studio 生成的母版页，我们从头创建，这有利于熟悉母版页的使用。
2. 为项目添加一个新项。在现在模板中选择“母版页”，并将该页面命名为 `MasterPage.master`。下图展示了如何添加母版页：

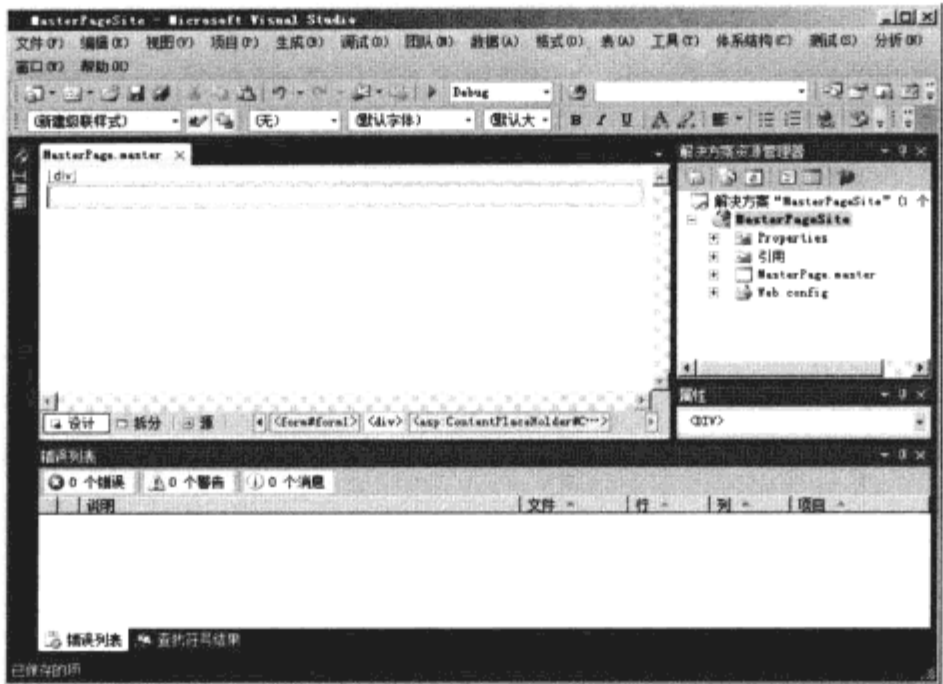


Visual Studio 会为 MasterPage.master 文件生成以下代码。请注意，其中的 ContentPlaceholder 控件也是 Visual Studio 生成的。

```
<%@Master Language="C#"
AutoEventWireup="true"
CodeBehind="MasterPage.master.cs"
Inherits="MasterPage"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title></title>
    <asp:ContentPlaceholder id="head" runat="server">
    </asp:ContentPlaceholder>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:ContentPlaceholder id="ContentPlaceholder1" runat="server">
        </asp:ContentPlaceholder>
      </div>
    </form>
  </body>
</html>
```

下图是该母版页在“设计”视图中的显示效果：

不难发现，母版页与一般的.aspx 页面非常相似。事实上，我们可以像使用一般.aspx 页面一样来使用母版页。



3. 找到 MasterPage.master 文件中的<body>标签。修改主体(body)的背景颜色：

```
<body style="background-color: #bbbbbb;">
  <form id="form1" runat="server">
    <div>
      <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
      </asp:ContentPlaceHolder>
    </div>
  </form>
</body>
```

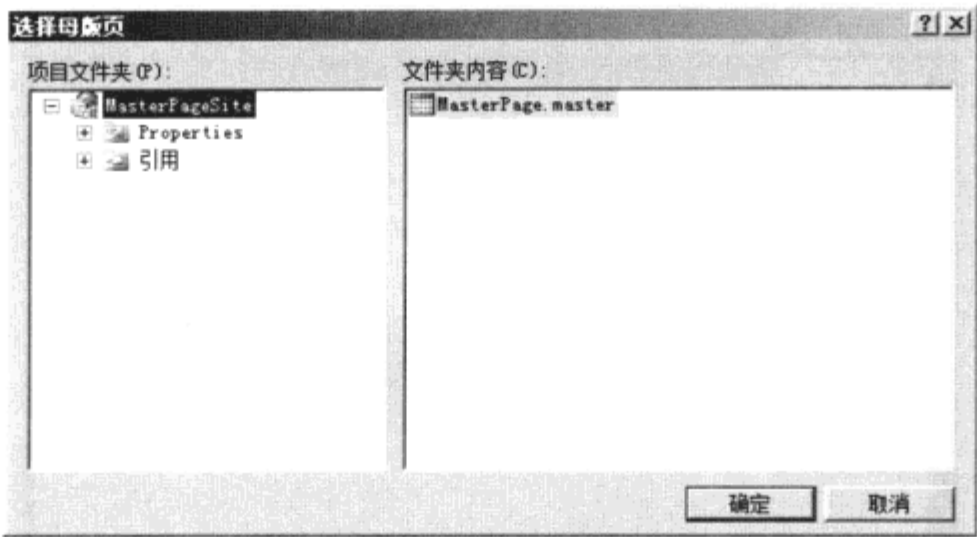
本示例采用了浅灰色。通过修改背景颜色，我们便可以轻松观察到该母版页对于.aspx 文件的作用。

4. 右键单击项目节点，选择“添加”|“新建项”。创建一个“使用母版页的 Web 窗体”，并将其命名为 UseMasterPage.aspx(如下图所示)。^①

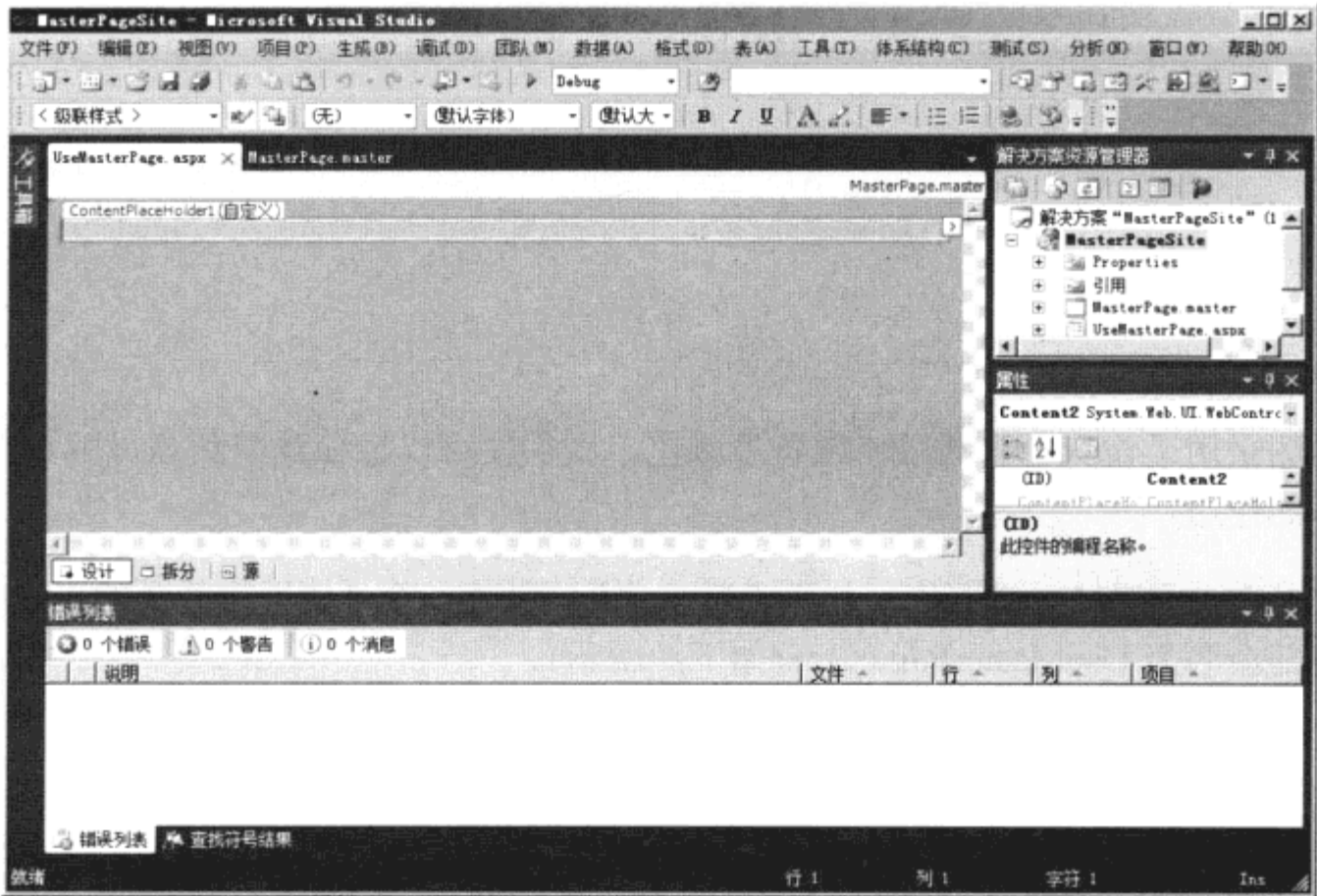


① 译者注：这里删去对“选择母版页”选项的操作，因为 Visual Studio 2010 通过“项目模板”的形式取代了之前版本中的这个选项。

Visual Studio 会提示我们选择一个母版页，如下图所示。



UseMasterPage.aspx 在“设计器”中的效果与刚刚创建的母版页非常相近。该页面也具有了浅灰色背景(如下图所示)，这说明母版页已作用于该页面。



Visual Studio 会为该页面生成以下代码来使其支持母版页：^①

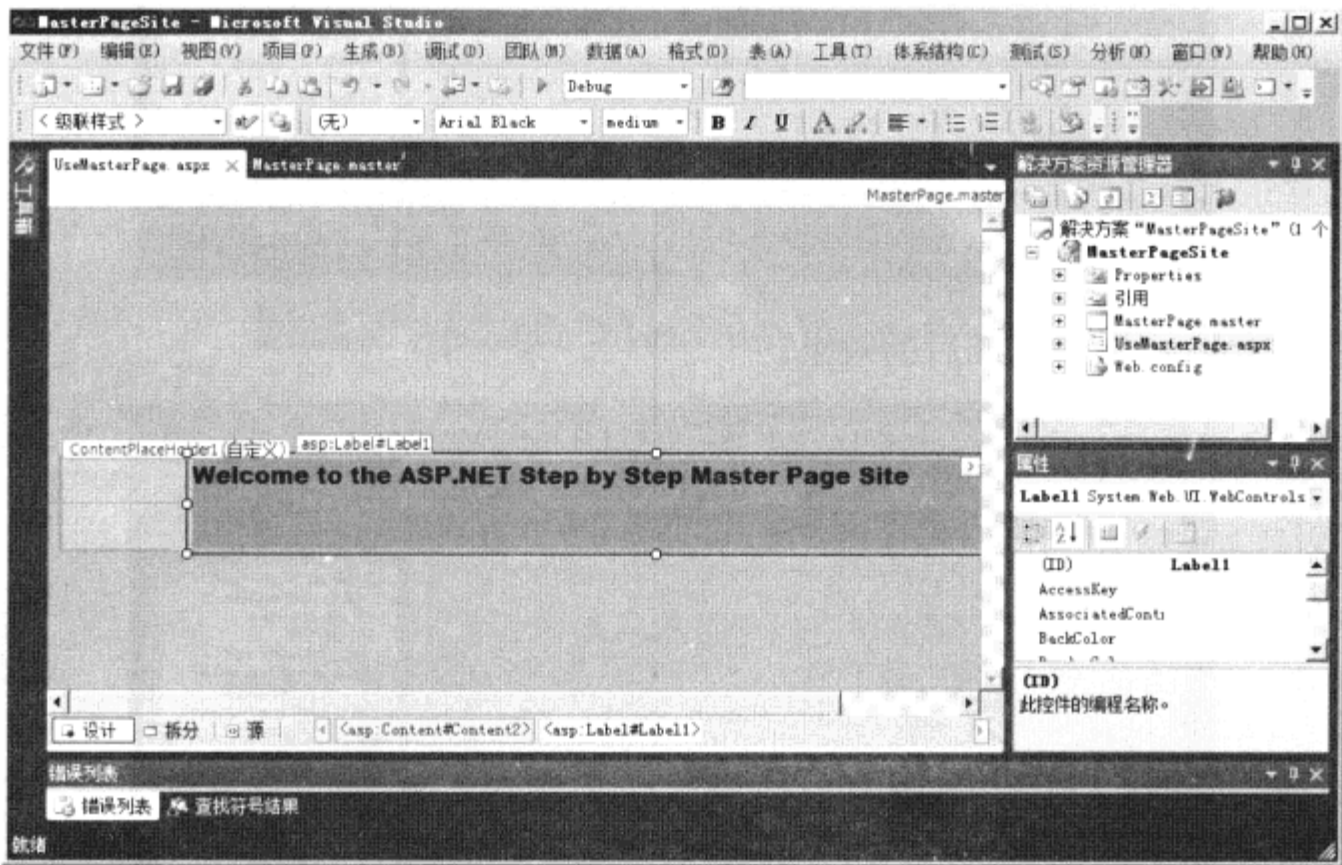
```
<%@Page Title="" Language="C#"
MasterPageFile="~/MasterPage.master"
AutoEventWireup="true"
CodeFile="UseMasterPage.aspx.cs"
Inherits="UseMasterPage"%>

<asp:Content ID="Content1"
ContentPlaceHolderID="head" runat="server">
</asp:Content>
```

① 译者注：这里经作者确认，为 Inherits 属性添加命名空间，否则会导致编译错误。

```
<asp:Content ID="Content2"
    ContentPlaceHolderID="ContentPlaceHolder1"
    Runat="Server">
</asp:Content>
```

5. 下面我们为 UseMasterPage.aspx 添加一些内容。在内容占位符控件中添加一个标签控件，输入一些文本以便能够将其与其他页面区分开来(见下图)。



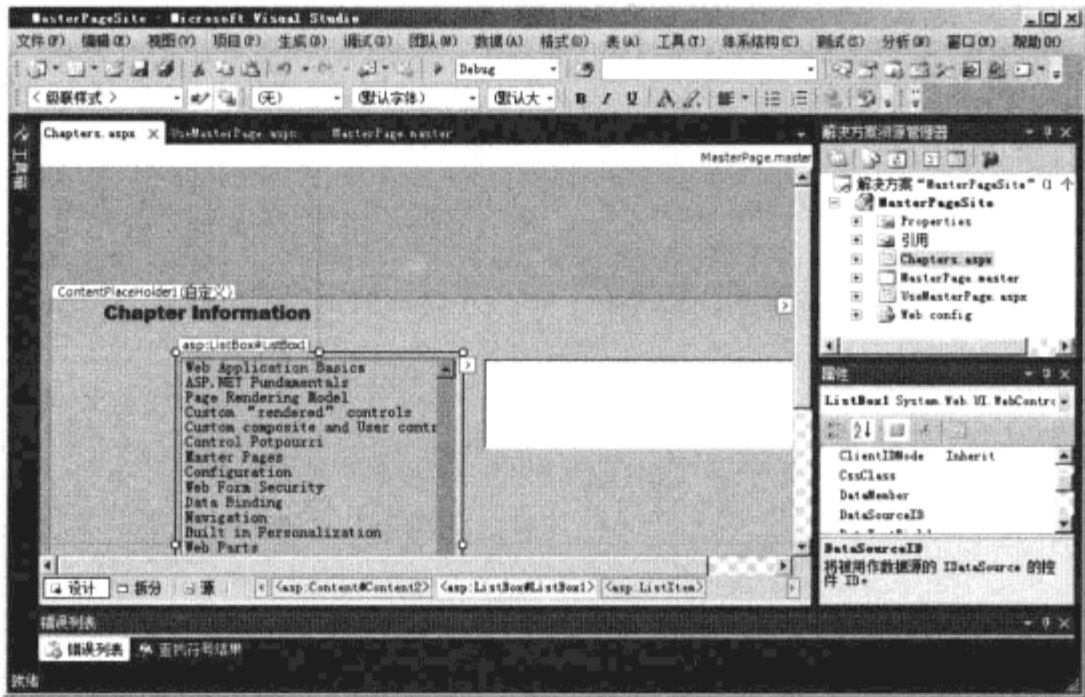
6. 再为网站添加两个页面。本示例用其中的一个来显示本书的章节，用另一个来显示各示例项目。读者也可以填充其他内容。在两页面的内容占位符中添加一些内容，以便将两者区分开来(稍后将在此添加导航支持)。

这里需要注意的是，添加这两个页面时要为其应用母版页(即通过“使用母版页的 Web 窗体”模板来创建页面)。

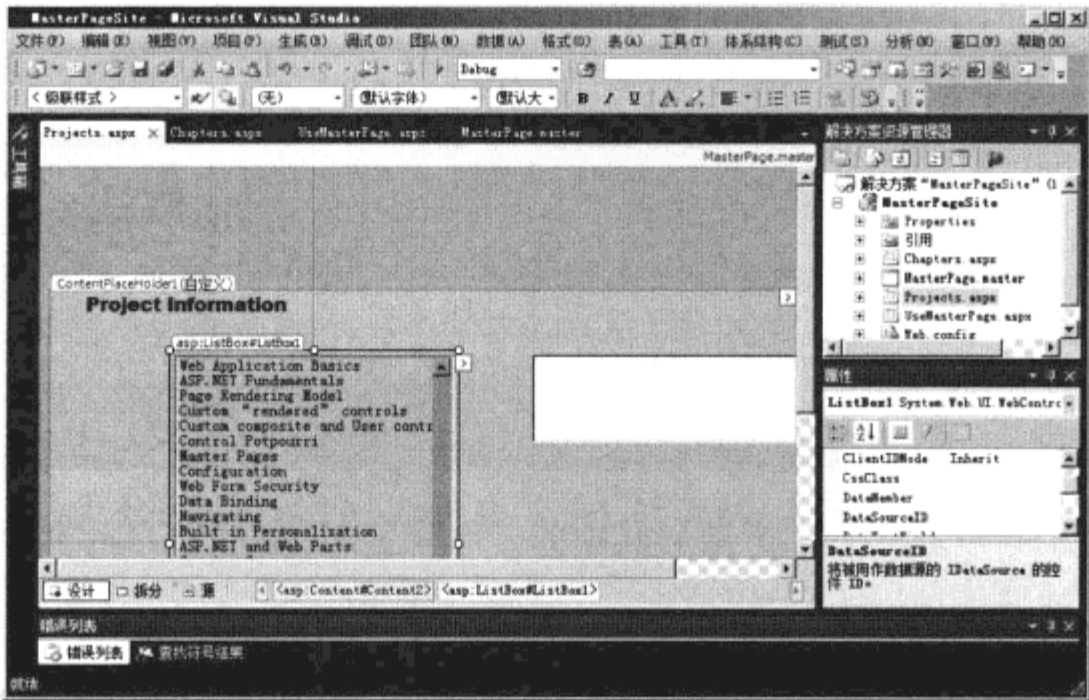
按下面两张截图所示，用 ListBox 来显示主题，用 TextBox 来显示选定主题的相关信息。将元素的定位方式设置为 absolute 可以简化布局工作。在这里我们也使用这种方式。此外，本示例会为两页面的 ListBox 分别填充章节(Chapters 页面)和项目名称(Projects 页面)。在 ListBox 的处理程序中为 TextBox 填充章节或项目相关的内容。这样，我们便可以实际体验到母版页带来的一致观感效果。

这里介绍一种为 ListBox 手动添加元素的方法。(第 10 章会介绍另一种技术。)首先，在“设计器”中选择 ListBox。单击 ListBox 右上角的小箭头，然后选择“编辑项”。我们可以在这个“ListItem 集合编辑器”中添加键/值对。本示例使用了两个 ListBox——一个用来罗列本书的章节，另一个用来罗列示例项目。下图展示了 Chapters 页面完成后的效果：

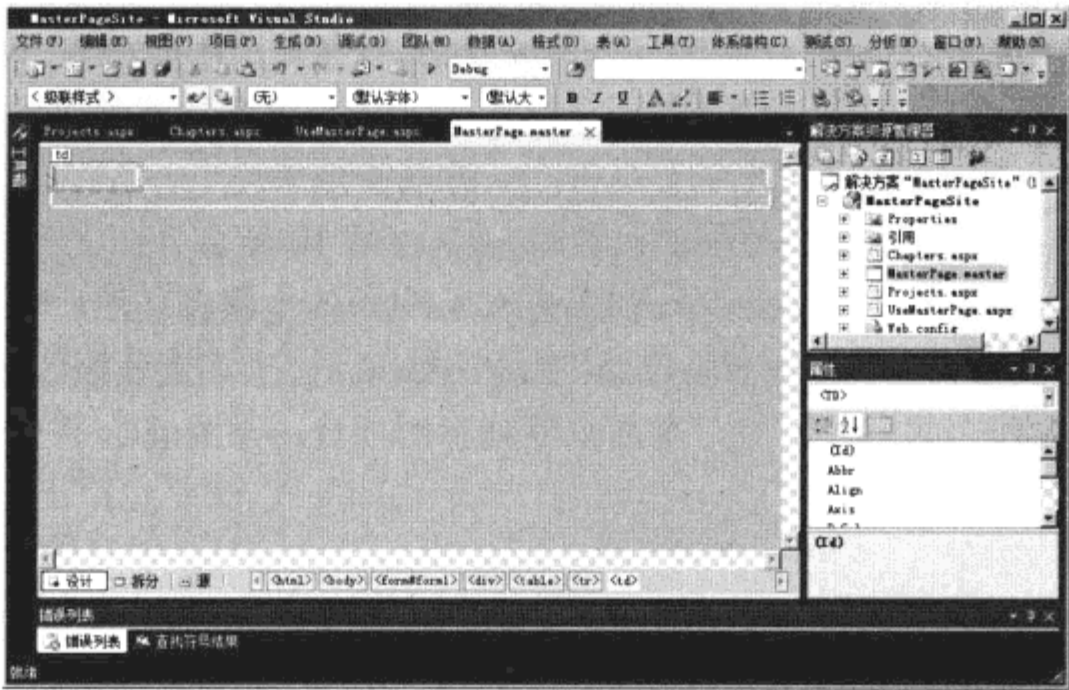
ASP.NET 4 从入门到精通



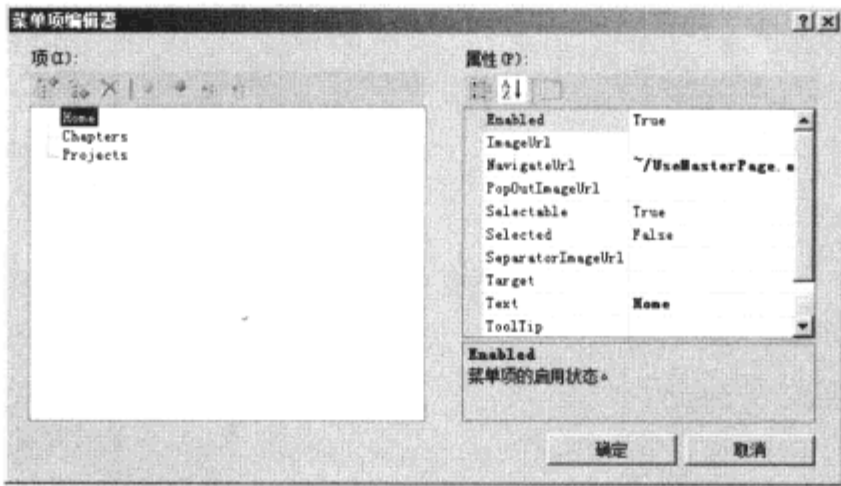
下图是 Projects 页面完成后的效果。



7. 为 MasterPage.master 页面再添加些内容。在“设计”模式下，单击“表”|“插入表”，在母版页内容窗格的上方添加一个一行两列的表。调整单元格的大小，使左侧的窄一些，右侧的宽一些。此时的母版页如下图所示。



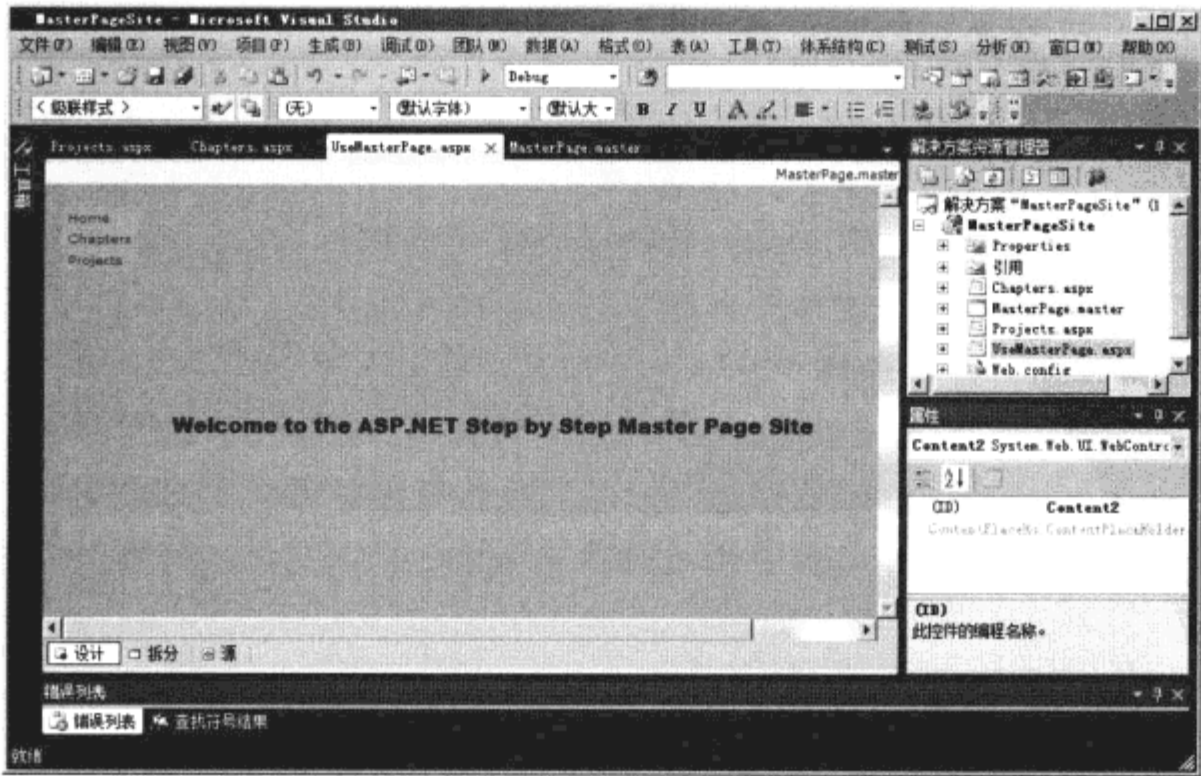
8. 在表格左侧的单元格中添加一个 Menu(菜单)控件。要定制菜单，可为菜单设置“自动套用格式”。本示例选择了“传统型”。在菜单中添加 3 个菜单项，分别用于导航到网站中的 3 个页面——Home、Chapters 和 Projects。为添加菜单项，在“设计器”中选择菜单控件，单击控件右上角的小箭头，选择“编辑菜单项”。此时会显示“菜单项编辑器”对话框(如下图所示)。在这里添加那 3 个菜单项。



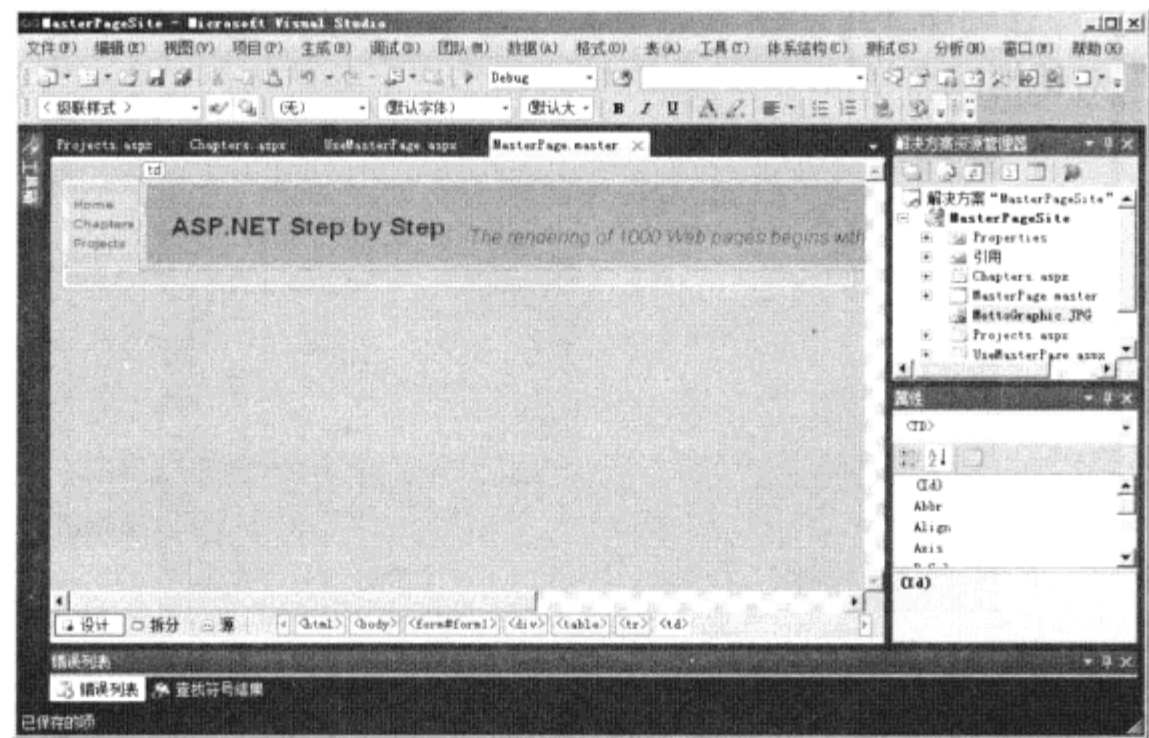
为每个菜单项设置导航参数。使 Home 菜单项导航至 UseMasterPage.aspx 页面(见下图)，使 Chapters 菜单项导航至 Chapters.aspx 页面，将 Projects 菜单项导航至 Projects.aspx 页面。可以单击“属性页”中 NavigateUrl 字段中的导航按钮，在这里单独设置导航用的 URL。



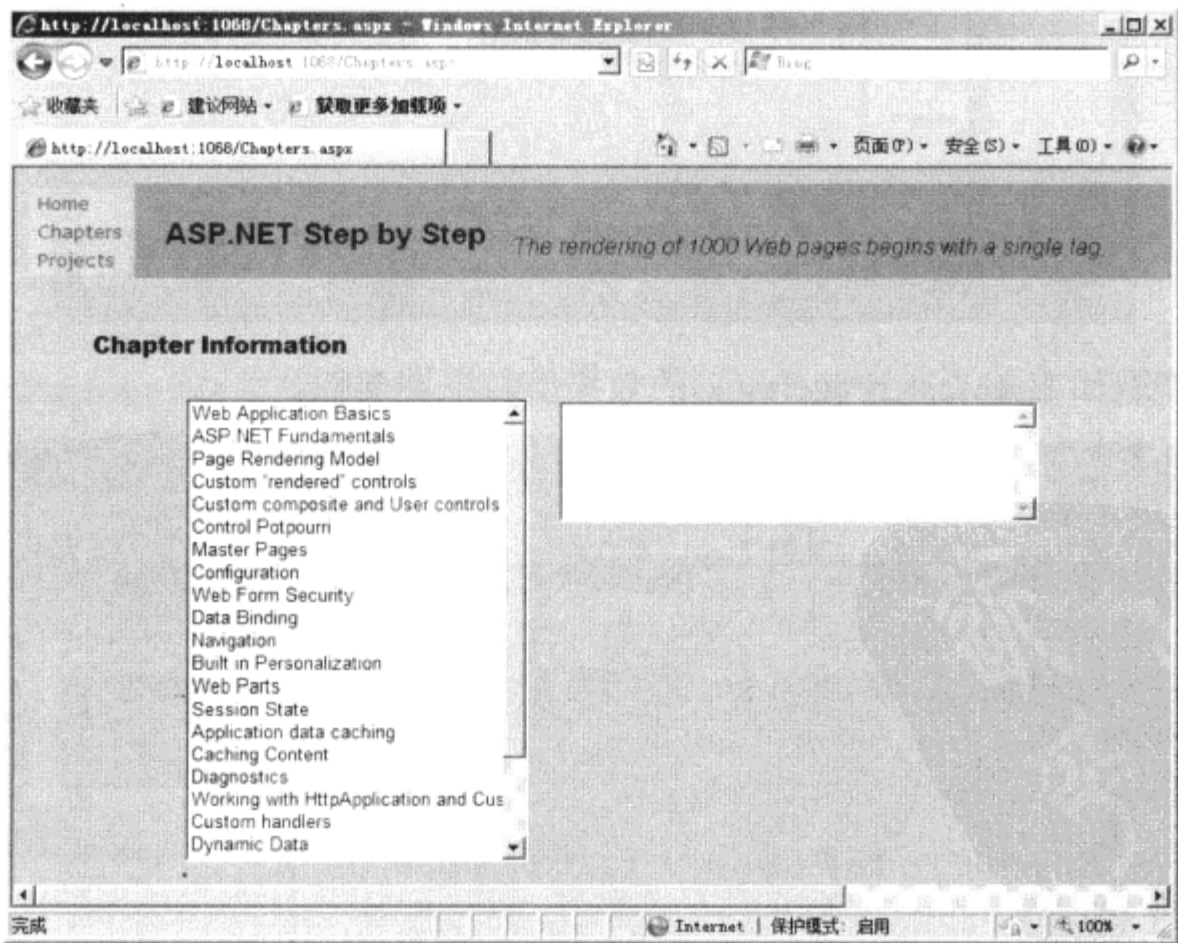
此时内容页(如 UseMasterPage.aspx)的效果应与下图类似。



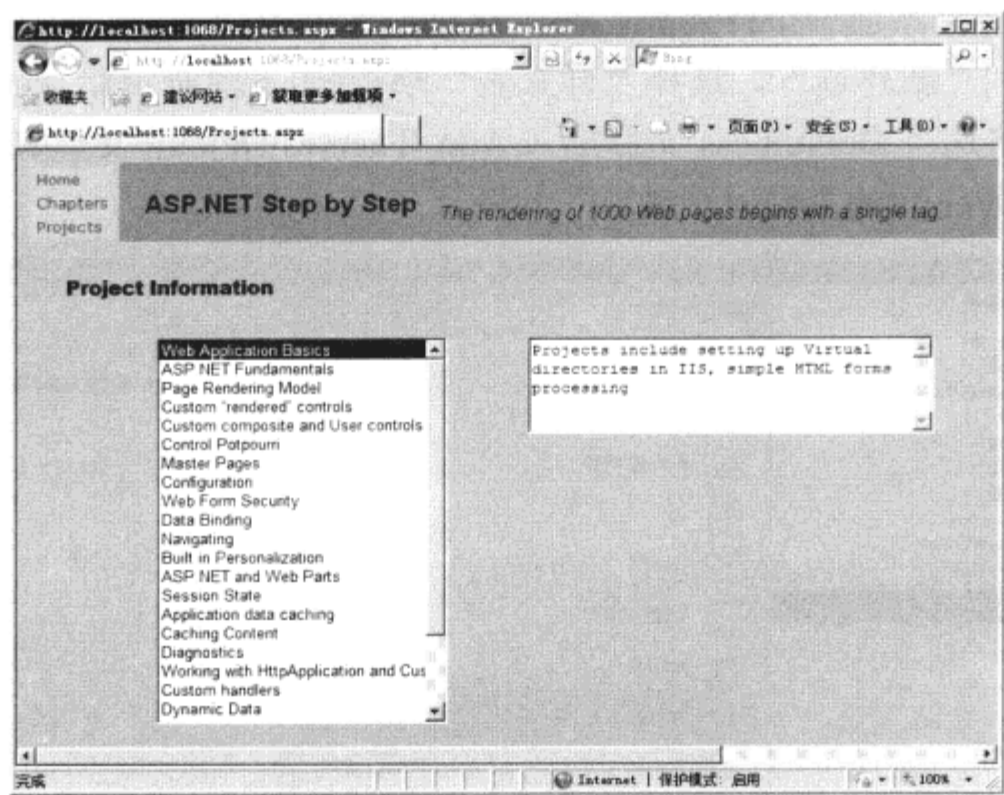
9. 最后，添加一个横幅。笔者认为绝大多数页面都应该有横幅。使用 Visual Studio 中的位图编辑器(或 Windows 中的“画图”)绘制一个横幅。本示例使用的图片宽 1000 像素，高 90 像素。将制作好的横幅拖放到表格右侧的单元格中。此时的母版页如下图所示。



由于 UseMaster.aspx、Chapters.aspx 和 Projects.aspx 是基于母版页创建的，因此菜单和横幅会被自动添加到各页面中。浏览 UseMaster.aspx 文件，并通过菜单项打开其他页面。我们会发现，所有页面都具有一致的界面，并且格式正确。Chapters 页面如下图所示。



Projects 页面如下图所示。



母版页极大地改进了传统 ASP 和 ASP.NET 早期版本对实现网站中页面一致观感的支持。当然，也可以在一个项目中使用多个母版页，甚至可以使用嵌套的母版页。

另一种管理 Web 应用程序观感的方法是使用 ASP.NET 主题。

7.3 主 题

对于应用程序中同一系列的页面，母版页能够控制其一般性布局。然而，开发者往往要使某些元素保持不变(而改变另一些)。主题提供了一种为页面上的元素应用公共样式的途径。

如果熟悉“级联样式表”(Cascading Style Sheets, CSS)，那么对“主题”也不会感到陌生。这两种技术类似，都是对网页定义可视样式。不过，主题要比 CSS 更高级。我们通过主题来为应用程序中的页面指定样式、图形，以及 CSS 文件。ASP.NET 主题的作用范围包括应用程序、页面和服务器控件。

ASP.NET 中的主题是基于文本的样式定义。开发者可以定义和使用自己的主题。下面我们来看看如何创建并使用主题。

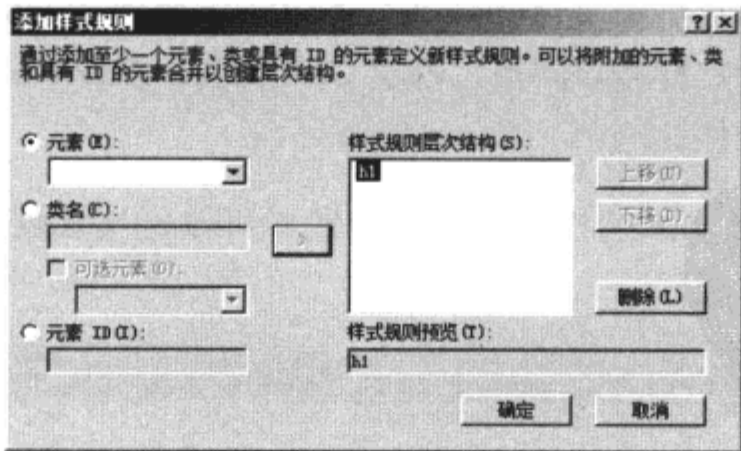
➤ 创建并使用主题

1. 为 MasterPagesSite 的项目添加一个“不使用”母版页的页面，并将其命名为 UseThemes.aspx。
2. 添加一个主题文件夹。在“解决方案资源管理器”中的网站节点上右击，依次选择“添加”|“添加 ASP.NET 文件夹”|“主题”。此时，Visual Studio 会创建一个名为 App_Themes 的目录。

- 3. 添加 App_Themes 后, Visual Studio 会自动创建一个子文件夹。我们将其命名为 Default。
- 4. 为 Default 文件夹添加一个样式表。在“解决方案资源管理器”中的项目节点上右击, 选择“添加”|“新建项”。选择“样式表”模板(如下图所示), 将这个样式表命名为 Default.css。将这个 Default.css 文件拖入 App_Themes\Default 文件夹。此样式表会在页面应用名为 Default 的主题时生效。^①

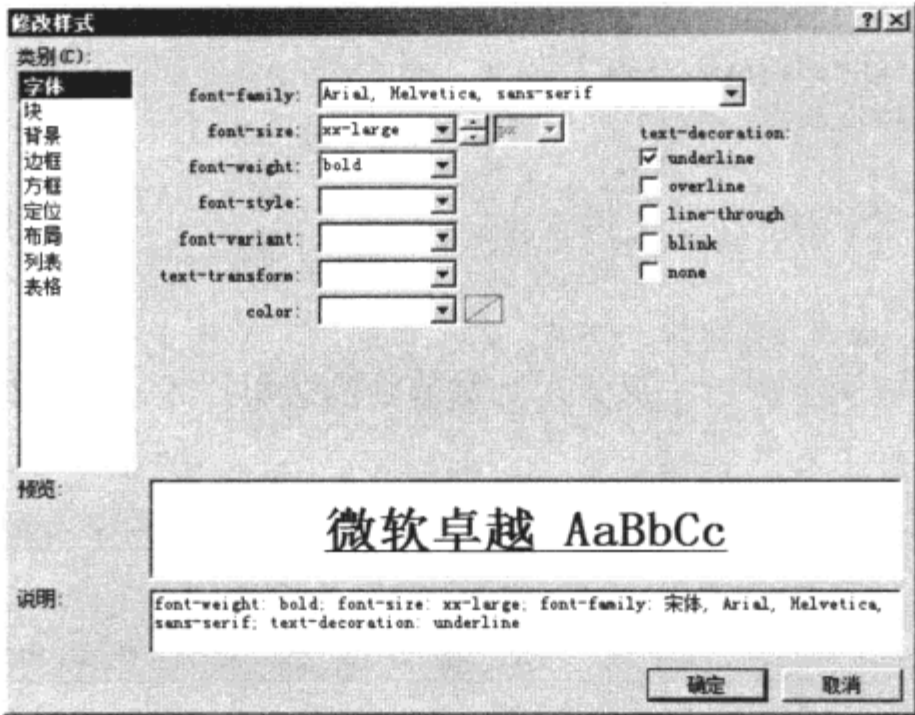


- 5. 下面来修改这个样式表。这个样式表文件默认会包含一个 body 标签。在 Visual Studio 中打开这个样式表文件, 在“样式”菜单中选择“添加样式规则”。此外, 也可以在标签上右击来修改其样式。例如, 如果要修改针对<h1>标签的样式, 则可以在该元素节点上右击, 然后选择“生成样式”。这里要添加一个针对<h1>标签的样式。从“元素”组合框中选择该元素, 单击“>”按钮将其添加到“样式规则层次结构”列表中(如下图所示), 然后单击“确定”按钮。



为修改 h1 的样式, 在“CSS 大纲”窗口中单击“h1”节点, 在“属性”窗口中选择“Style”, 单击旁边的省略号按钮(⋮)。此时会打开“修改样式”对话框, 按下图修改字体的大小和粗细(或其他属性), 然后单击“确定”按钮。

^① 译者注: App_Themes 文件夹的直属子文件夹的名称就是主题的名称, 也就是引用主题时使用的名称(详见第 8 步)。



本示例为字体指定了 Arial(字体)、xx-large(字体大小)、bold(粗细)和下划线。

在 CSS 文件中，我们会发现原样式定义已被修改：

```
body
{
}
h1
{
    font-family: Arial,Helvetica,sans-serif;
    font-size: xx-large;
    font-weight: bold;
    text-decoration: underline;
}
```

- 6. 下面我们来测试一下这个主题的效果。在页面中用 Theme 属性声明，并使用<h1>标签输入一个标题，如下所示：^①

```
<%@Page Language="C#" AutoEventWireup="true"
    Theme="Default"
    CodeFile="UseThemes.aspx.cs"
    Inherits="UseThemes"%>

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

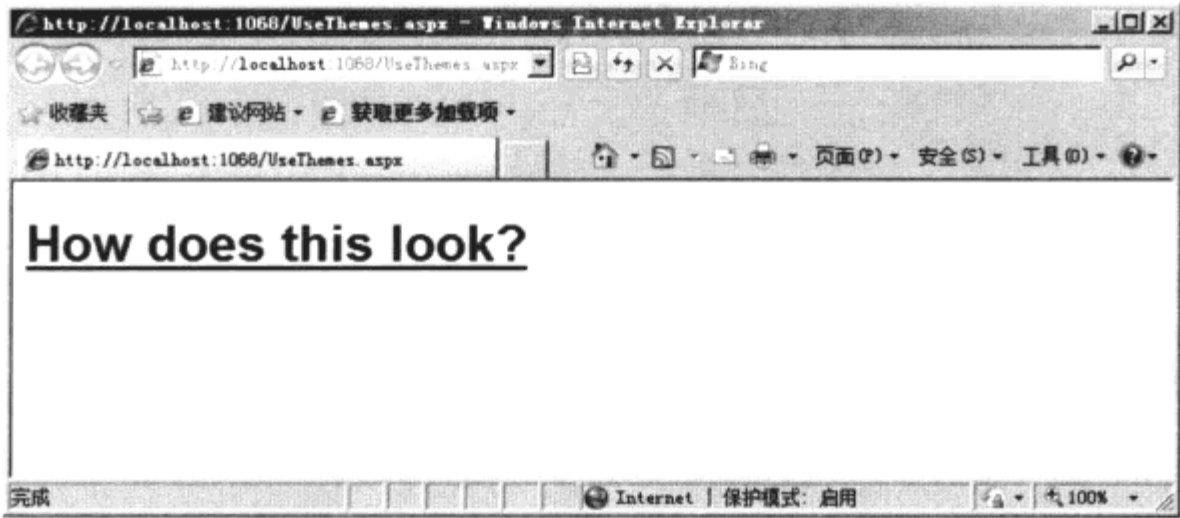
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
```

① 译者注：这里经作者确认，为 Inherits 属性添加命令空间，否则会导致编译错误。

```
<div>
    <h1>How does this look?</h1>
</div>
</form>
</body>

</html>
```

下图展示了应用新的主题后，页面在浏览器中的效果。本示例创建的主题为<h1>标签应用了新的字体和下划线。



- 7. 在项目中再添加一个主题，将其命名为 SeeingRed。在这个主题文件夹下添加一个名为 SeeingRed 的样式表。这次将<h1>标签的字体颜色设置为红色。然后，在页面引用名为 SeeingRed 的主题(也可以在 Visual Studio 的“属性窗口”中设置):

```
<%@ Page Language="C#" AutoEventWireup="true"
Theme="SeeingRed"
CodeFile="UseThemes.aspx.cs"
Inherits="MasterPageSite.UseThemes"%>
```

浏览这个页面，我们会发现<h1>标签被显示为红色。

本节只是简单展示了主题的效果，使读者获得一个感性认识。定义了主题后，可在 Page 指令中引用，也可以在 PreInit 事件的处理程序中修改页面的 Theme 属性。

下一节将要介绍的“皮肤”是一种与主题配合使用的技术。

7.4 皮 肤

皮肤是对母版页和主题在修改网站样式方面的补充。使用皮肤类似于将基于 WebControl 的控件与 CSS 关联。我们还可将皮肤看作是集中设置控件属性的一种方式。举例来说，皮肤可以为某个控件设置不同的颜色组合(如像 TextBox 和 Calendar 这样的具有多种功能和属性的控件)。通过为控件设置皮肤，开发者可以方便地为不同控件设置外观，而不必深入网站中的每个控件实例，逐一管理各自的属性。

事实上，我们已经在使用皮肤了。许多服务器端控件都支持样式模板。例如，在上一章使

用 TreeView 时，有多种样式可供选择。本章在使用 Menu 控件时，也在“自动套用格式”窗口中为其选择了“传统型”样式。本节将介绍如何自行创建和使用皮肤。

皮肤文件能够定义特定的控件及针对该控件的属性。皮肤文件是一种扩展名为.skin 的文件，包含服务器端控件的声明。皮肤文件旨在为控件预先设置属性。皮肤文件应放在主题文件夹中，与必要的 CSS 文件在一起。

我们将在下面的练习中学习如何为网站的某些控件创建皮肤。

➤ 创建皮肤

- 1. 创建一个皮肤文件。在“解决方案资源管理器”的 App_Theme\SeeingRed 文件夹上右击，选择“添加”|“新建项”。在模板列表中选择“外观文件”，将其命名为 SeeingRed.skin。
- 2. 在 SeeingRed.skin 文件中，先声明几个控件，并分别设置一组属性。本示例在 SeeingRed.skin 文件中为以下控件分别定义了一组属性。这些控件将具有不同的颜色，默认使用红色或粉红色。

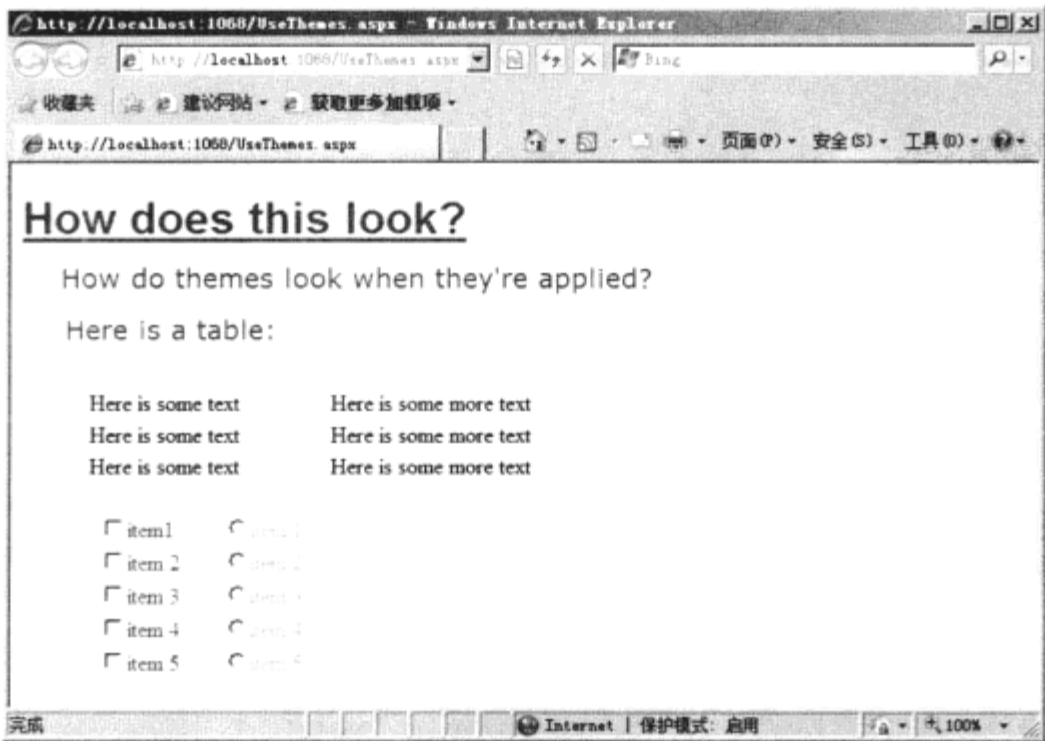
```
<asp:Label runat="server" ForeColor="red"
Font-Size="14pt" Font-Families="Verdana"/>

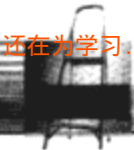
<asp:button runat="server" borderstyle="Solid"
borderwidth="2px" bordercolor="#ff0000" bgcolor="#cc0000"/>

<asp:CheckBoxList runat=server ForeColor="#ff0000"/>

<asp:RadioButtonList runat=server ForeColor="#ff9999"/>
```

- 3. 根据 SeeingRed.skin 文件中的定义，在 UseThemes.aspx 页面中逐一添加每种控件，看看皮肤是如何发挥作用的。在下图中，红色的控件会呈现为浅灰色。实际运行应用程序便可以看到效果。





在页面中声明使用 SeeingRed 主题后，SeeingRed.skin 文件会自动生效。在运行时，我们也可以 在页面的 PreInit 事件处理程序中设置每个控件的皮肤。

7.5 快速参考

目 标	操 作
在网站中定义一系列具有一致感观的页面	为网站添加母板页
创建基于母板页的页面	添加“使用母版页的 Web 窗体”，并选择所要应用的母板页
在母版页中添加会最终显示在内容页的元素	将这些元素添加到 ContentPlaceholder 控件之外的位置
添加内容页特有的元素	将这些元素添加到内容页 Content 控件中
为页面创建主题	在应用程序的 App_Themes 目录为主题添加一个文件夹。通过“级联样式表”(CSS)来为主题定义样式和类别
将主题应用到页面	可以在 ASPX 文件中设置 Page 指令的 Theme 属性，也可以在页面的 PreInit 事件被引发时设置 Theme 属性
创建皮肤	在主题所在文件夹下创建一个文本文件，为其添加.skin 扩展名，并添加带有默认属性值的控件声明

第 8 章 配 置

学习目标

- 理解 Microsoft .NET Framework 的配置机制
- 为 ASP.NET 应用程序添加配置
- 通过“网站管理工具”来管理 ASP.NET 配置
- 使用“Internet 信息服务(IIS)管理器”来配置 ASP.NET 应用程序

本章介绍 ASP.NET 管理配置信息的方式。您将理解 ASP.NET 的配置机制。在开发和部署网站方面，ASP.NET 是一个具有丰富功能的系统。在之后的章节中，我们还会接触到 ASP.NET 的配置，其中涉及以下功能。在学习这些功能后，相信您会对配置有一个较为清晰的认识：

- 会话状态
- 通过缓存内容来优化网站的响应
- 跟踪请求
- 管理针对特定文件扩展名的自定义处理程序
- 用户身份验证

上面的每种功能都受许多专门的可配置参数控制。例如，如果为应用程序启用会话状态，则可以选择应用程序会话状态的位置(可以是进程中的，可以通过 Windows 服务放置到其他计算机上，也可以使用 Microsoft SQL Server)。我们还可以配置会话状态的生命周期，以及应用程序对会话状态的跟踪方式(使用 cookie 或其他方法)。

对输出进行缓存是另一项可以通过配置文件控制的功能。如果缓存网站的内容，则可以控制缓存时间的长短及缓存的位置(服务器、客户端或代理)。

这里提到的两种功能(和其他功能)，都受配置文件的掌控。本章先介绍 Windows 配置的特点，然后重点介绍 ASP.NET 的配置机制。在 ASP.NET 1.x 中，修改应用程序的配置需要手动修改基于 XML 的配置文件。幸运的是，后来的 ASP.NET(2.0 和更高版本)提供了两个工具，能够方便地修改配置。其中之一是使用 Internet 信息服务(IIS)7.x 功能视图中的 ASP.NET 配置工具。另一个是基于 Web 的“网站管理工具”，通过 Microsoft Visual Studio 的“网站”

/“项目”^①菜单下的“ASP.NET 配置”打开。本章将依次介绍这两个工具。

8.1 Windows 的配置机制


所有计算环境都需要一种配置机制来控制其行为。有许多参数可以控制操作系统和程序的运行。修改这些参数可能是为了优化性能，可能是为了安全，也有可能只是为了控制常规的运行方式。例如，Windows 操作系统提供了一个名为 PATH 的环境变量，用于设置可执行程序的搜索路径。此外还有名为 TEMP 和 USERPROFILE 的环境变量，分别用于设置临时文件和当前用户配置文件的位置。

除操作系统的参数以外，每个应用程序也可能有特定的设置。例如，许多应用程序要求运行在某一特定的 Windows 操作系统中，或者要求具有某个动态链接库(DLL)。不同的安装设置会产生不同的行为，因而设置不应硬编码在程序中。相反，应该将这些值存储在一个伴随应用程序的文件中。

在早期的 Windows 中，可以使用初始化(initialization)文件(.ini 文件)来配置每个应用程序和 Windows 操作系统本身(甚至有一套专门管理配置参数的 Windows 应用程序编程接口)。这种文件包含分别表示属性及设置的键/值对。例如，如果要在 Win.INI 文件中启用“对象链接和嵌入”(Object Linking and Embedding, OLE)，则可以使用这样的键/值对：

```
OLEMessaging=1
```

如今，XML 成为了主流配置技术。.NET 使用 XML 文件(machine.config、web.config 和其他可信赖的文件)来存储配置信息。

 **提示** 在过去，还可以通过注册表(registry)来配置应用程序。注册表是一种集中式的数据库，应用程序可以在其中存储键/值对。ASP.NET 没有使用注册表来存储配置信息是因为注册表是全局的，直接影响 ASP.NET 在开发上的灵活性。存储在注册表中的设置要使用 Registry API 来复制，而配置文件更易于复制。此外，为防止黑客的攻击，运行 ASP.NET 网站的帐户一般都不具有注册表的访问权限。

8.2 .NET 的配置机制

.NET 配置文件是规范的 XML 文件，其关键字受 .NET 运行库支持。配置目录中(稍后会介绍)包含所有类型的配置文件。

.NET 运行库会在必要时将这些配置文件读入内存来设置各 .NET 运行时参数，而且这些参数是层叠的。例如，ASP.NET 启动时加载的是 web.config，而服务器会先加载 machine.config。

^① 译者注：根据项目类型(如“网站项目”和“Web 应用程序”项目)的不同，菜单项会有一定的差异。

8.2.1 Machine.Config

计算机默认的.NET 配置是在一个名为 `machine.config` 的文件中声明的。`machine.config` 文件可以在 `C:\Windows\Microsoft.NET\Framework\vxxxxx\Config` 路径下找到(`xxxxx` 为.NET 版本, 在本书截稿时最新的版本为 4, 对应的目录名为 `v4.0.30319`^①)。`machine.config` 包含计算机中所有.NET 应用程序行为的默认设置。

最近的几个.NET 版本改进了 `machine.config` 的组织方式。.NET 的 1.x 版将所有配置信息都存储在 `machine.config` 文件中——甚至包括注释和某些计算机用不到的系统所使用的配置信息(如浏览器定义信息, 因为有些计算机不承载 ASP.NET)。从 2.0 开始, `machine.config` 在很大程度上得到了精简。注释被移到一个单独的文件中——`machine.config.comments`。浏览器定义被移到专门的配置文件中。理解这一点很重要, 因为 `machine.config` 的注释往往比.NET 配置方面的在线文档更为重要。在配置 ASP.NET 应用程序时, `machine.config` 的注释往往是最先要查看的。`machine.config` 文件的 4.0 版要略大于之前的 3.0 版本。

8.2.2 配置节处理程序

`machine.config` 的最顶端罗列了许多配置节处理程序(configuration section handler)。这些处理程序能够解释配置.NET(和 ASP.NET)所使用的关键字。`machine.config` 包含计算机范围的设置, 而 ASP.NET 应用程序通过名为 `web.config` 的文件管理设置。稍后会详细介绍 `web.config`, 这里先给出某 Web 应用程序 `web.config` 文件的一部分:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <sessionState mode="SQLServer" cookieless="UseUri" timeout="25" />
  </system.web>
</configuration>
```

这个片段告诉 ASP.NET 运行库采用“Forms 身份验证”(ASP.NET 身份验证方式之一)来对网站的用户进行身份验证。此设置还会使 ASP.NET 使用 SQL Server 来管理会话状态, 使会话状态信息在 25 分钟后过期, 并使用嵌在“统一资源标识符”(URI)中的会话 ID 来跟踪会话信息。第 14 章会详细介绍会话状态——这里只是作为 ASP.NET 配置参数的一个例子。

从这个例子中我们可以看出, 为配置 ASP.NET, 运行库需要能够理解某些关键字。上面给出的配置为使 ASP.NET 理解如何管理身份验证, 使用了 `authentication`、`mode` 和 `Forms` 关键字; 而为了通过相应的关键字来配置会话状态, ASP.NET 需要正确解释 `sessionState`、

① 译者注: 这是.NET Framework 4 在本书截稿前的最新版本。

mode、SQLServer、cookieless、UseURI 和 timeout。

能够解释这些关键字的组件列于 machine.config 的顶部。

```
<configuration>
  <configSections>
    <section name="appSettings"
      type="{entire strong assembly name here...}"
      restartOnExternalChanges="false" />
    <section name="connectionStrings"
      type="{entire strong assembly name here...}" />
    ...
    <sectionGroup name="system.web"
      type="{entire strong assembly name here...}">
      <section name="authentication"
        type="{entire strong assembly name here...}"
        allowDefinition="MachineToApplication" />
      <section name="sessionState"
        type="{entire strong assembly name here...}"
        allowDefinition="MachineToApplication" />
      ...
    </sectionGroup>
  </configSections>
</configuration>
```

这段代码是被精简过的。不妨亲自查看一下 machine.config，其中包含节处理程序的详细信息。(在大多数计算机上，machine.config 位于 C:\Windows\Microsoft.NET\Framework\vxxxxx\Config 路径下。)在这些配置节处理程序中，我们会看到 sessionState 的配置使用了程序集的强名称——System.Web.Configuration.SessionStateSection，System.Web，Version=4.0.0.0，Culture=neutral，PublicKeyToken=b03f5f7f11d50a3a。强名称(strong name)能够“准确”指定程序集的名称，其中包含版本信息(保证版本兼容性)和公钥权标 public key token(确保程序集未被篡改)。

8.2.3 web.config

machine.config 文件为计算机提供了默认设置(这些设置会最终提供给特定的应用程序)。这些默认设置适用于一般情况，而特定的应用程序可能需要特殊的配置。例如，sessionState 默认会被设置成以“进程内”(in process)方式处理。这种方式对开发的影响不大，但很难适应于为不同类型的客户端提供服务的企业级应用程序。

同一台计算机上的所有.NET 应用程序都依赖于 machine.config 文件的配置，修改该文件可能会影响到多个应用程序。因而直接修改这个文件并不明智。

每个独立的.NET 应用程序为配置自身，依赖于其自身的配置文件，这些文件是根据应用程序的名称来命名的。例如，名为 MyApp.EXE 的应用程序可能要使用名为 MyApp.EXE.config 的配置文件。当然，ASP.NET 应用程序并没有遵循这种命名规范。ASP.NET 运行库总会在

名为 web.config 的文件中查找配置信息。

Microsoft Visual Studio 2010 引入了一个新功能——分别为应用程序的 debug 版本和 release 版本提供配置支持。之前的 Visual Studio 只提供一个 web.config 文件，应用程序的 debug 版本和 release 版本使用相同的设置。而在 Visual Studio 2010 中，Visual Studio 会在生成 Web 应用程序时提供 3 个配置文件——web.config、web.debug.config 和 web.release.config。web.config 中包含 debug 和 release 共享的设置，其他两个配置文件则分别针对 debug 和 release 版本(如 Trace 设置)。

为覆盖默认设置，只需要在应用程序的虚拟目录中包含 web.config 文件即可。例如，下面的配置信息能够启用“Forms 身份验证”和跟踪：

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <trace enable="true"/>
  </system.web>
</configuration>
```

应用程序实际使用的设置(可能)继承于一系列 web.config 文件。machine.config 文件提供 .NET 的默认设置。最顶层的 web.config 文件(.NET 配置目录下的)提供 ASP.NET 的初始配置。然后，在请求路径中，开发者可以通过子 web.config 文件来为单个应用程序提供针对特定文件夹的设置。

这种方式能够很好地管理配置信息。许多一般的默认设置可以用在大多数场景中，而有时只需要对其中的几项进行修改。此时，只需要在虚拟目录和/或子目录中添加 web.config。

不过，如果使用较多单独的配置，那么人工管理硬盘上的 web.config 文件会很不方便。ASP.NET 配置模式中包含一个名为 location 的元素，能够在一个主配置文件中为不同目录指定不同设置(而对应用程序来说，所有设置都将汇集到主配置文件)。

例如，下面的配置节会禁止 AppSubDir 目录处理标准的 ASP.NET Web 服务。这条 remove 指令会使 ASP.NET 忽略带有 .asmx 扩展名的所有文件。

```
<configuration>
  <location path="AppSubDir">
    <system.web>
      <httpHandlers>
        <remove verb="*" path="*.asmx" />
      </httpHandlers>
    </system.web>
  </location>
</configuration>
```

我们也可以在子目录中应用特定设置(如出于安全目的)。(第 9 章会具体介绍安全性方面的内容。)ASP.NET 配置文件包含管理身份验证和授权的关键字似乎不足为奇。这一特性非常

适合采用 `location` 元素来进行配置。下面的配置代码允许所有用户访问主(虚拟)目录，但要求对访问 `PagesRequiringAuth` 子目录的用户进行身份验证：

```
<configuration>
  <system.web>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
  <location path="PagesRequiringAuth">
    <system.web>
      <authorization>
        <deny users="?" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

8.2.4 ASP.NET 1.x 的配置管理

在 ASP.NET 1.x 中，需要在目标 `web.config` 文件中以手动输入的方式来修改设置。例如，如果要采用 `SQLServer` 来作为应用程序的会话状态数据库，则需要在应用程序的 `web.config` 文件中逐字输入正确的关键字。不幸的是，没有能够确保配置文件语法正确的配置编译器。如果输入有误，则很难意识到错误的存在，直到运行应用程序时 ASP.NET 显示错误消息。

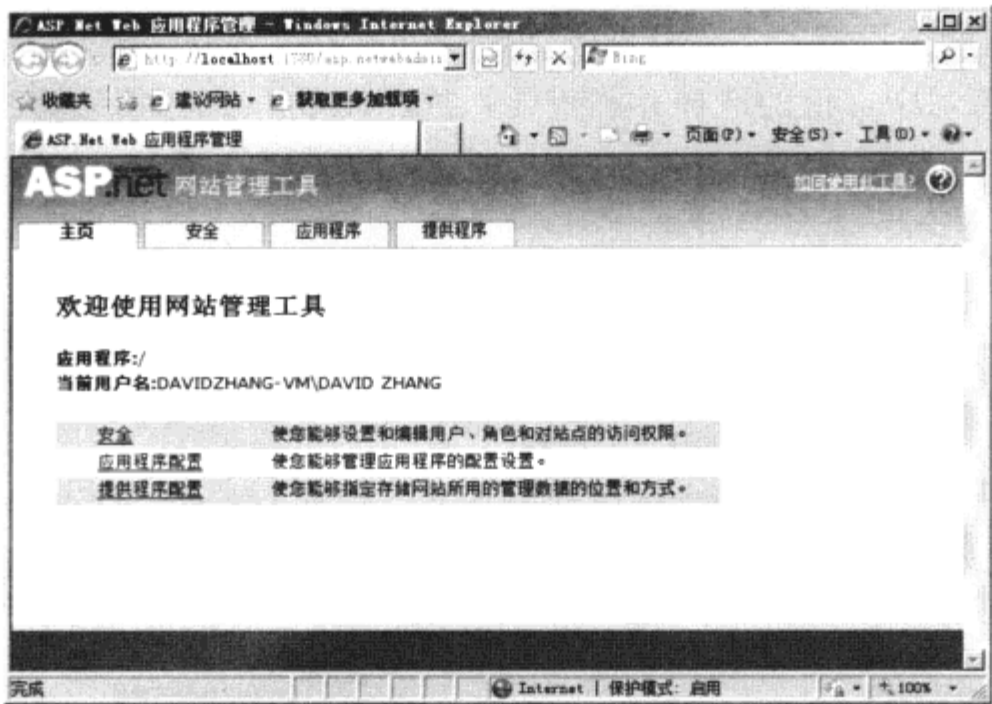
8.2.5 ASP.NET 后续版本的配置管理

ASP.NET 2.0 在 ASP.NET 应用程序的管理流程上做了几项重要的改进，这些改进一直延续至 ASP.NET 的当前版本。虽然仍可以手动在 `web.config` 文件中输入配置信息，但 ASP.NET 2.0 和后续的版本提供了一些新的配置工具，其中包括 Visual Studio 中的“网站管理工具”(Web Site Administration Tool, WSAT)和 IIS 7.x 中的 ASP.NET 配置工具。

在下面的练习中，我们将使用“网站管理工具”来修改应用程序的配置，并观察其对 `web.config` 文件的影响。

➤ 配置应用程序

1. 在 Visual Studio 中创建一个名为 `ConfigORama` 的“ASP.NET 空 Web 应用程序”项目。
2. 在 Visual Studio 生成应用程序后，单击“项目”|“ASP.NET 配置”。此时会打开“网站管理工具”，如下图所示：



其他选项卡

“网站管理工具”包含 4 个选项卡：“主页”、“安全”、“应用程序”和“提供程序”。“安全”选项卡用于管理身份验证和授权设置。也就是说，我们可以使用“安全”选项卡来添加用户并为其分配角色。下一章会具体介绍有关的操作。

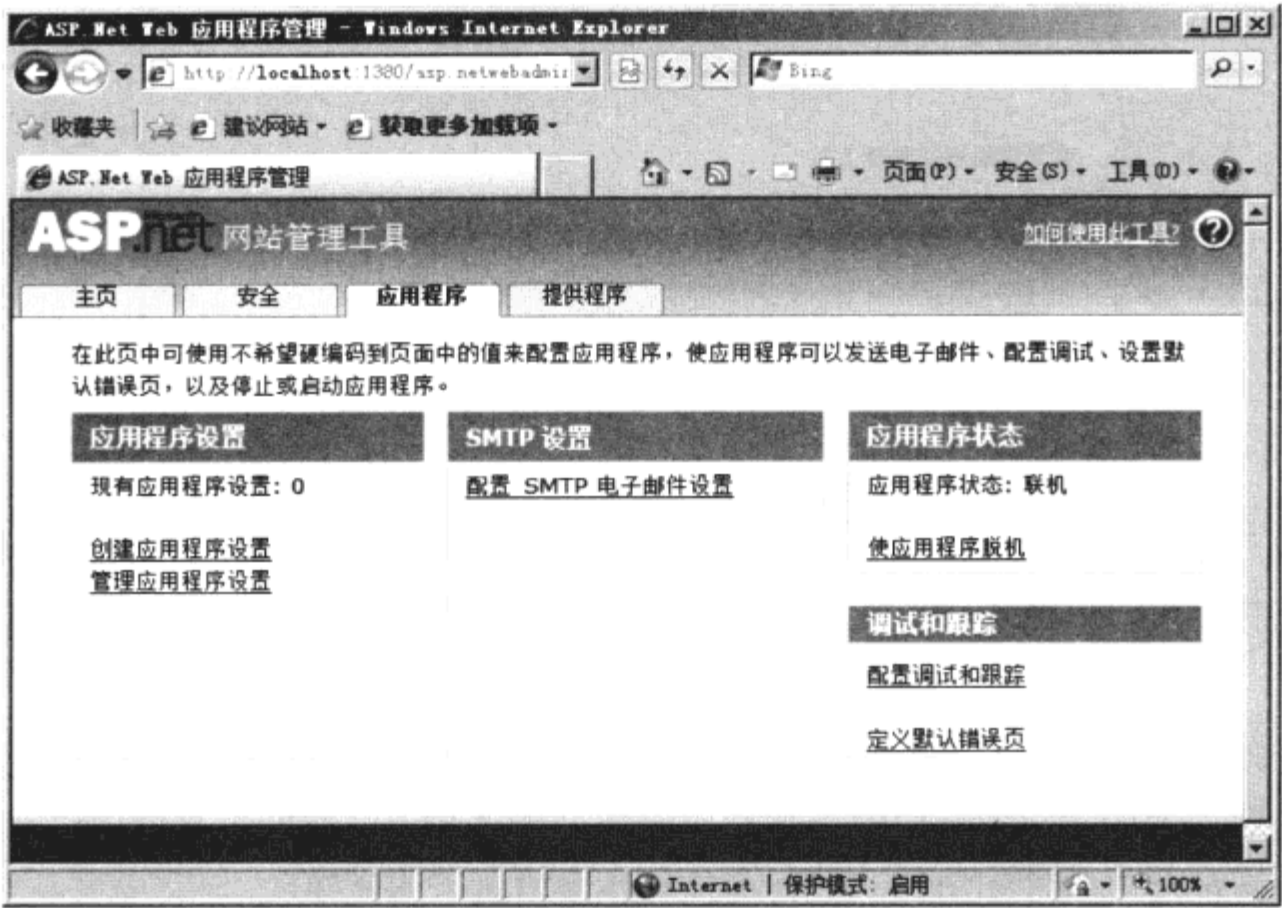
“应用程序”选项卡用于管理与应用程序有关的各种设置。在这里可以控制基本的设置，其中包括维护针对应用程序的键/值对、配置网站发送电子邮件所使用的“简单邮件传输协议” (Simple Mail Transfer Protocol, SMTP)，以及开启和关闭调试和跟踪的设置。在需要对网站进行维护时，还可以通过该选项卡使应用程序脱机。

最后，我们可以使用“提供程序”选项卡来管理各种数据提供程序。在 ASP.NET 2.0 中，Microsoft 引入了提供程序(provider)的概念，旨在使 ASP.NET 子系统的数据访问更易于使用和更为标准化。例如，用户可能有个性化的设置，此时便可以使用“成员资格”(Membership)提供程序来获取相关数据，以显示给用户或在管理时使用。“角色”(Role)提供程序可以为使用 Web 应用程序的用户分配角色。使用“提供程序”选项卡，可以分别设置每种提供程序。我们一般使用 ASP.NET 内建的提供程序来访问数据库，进行数据的存储和获取。此外，我们也可以使用自定义的提供程序、第三方的提供程序，或者混合使用这两类提供程序。“提供程序”选项卡中的提供程序配置允许我们选择使用哪种提供程序(如果存在多个)。如果使用数据库，还可以设置数据库连接字符串。

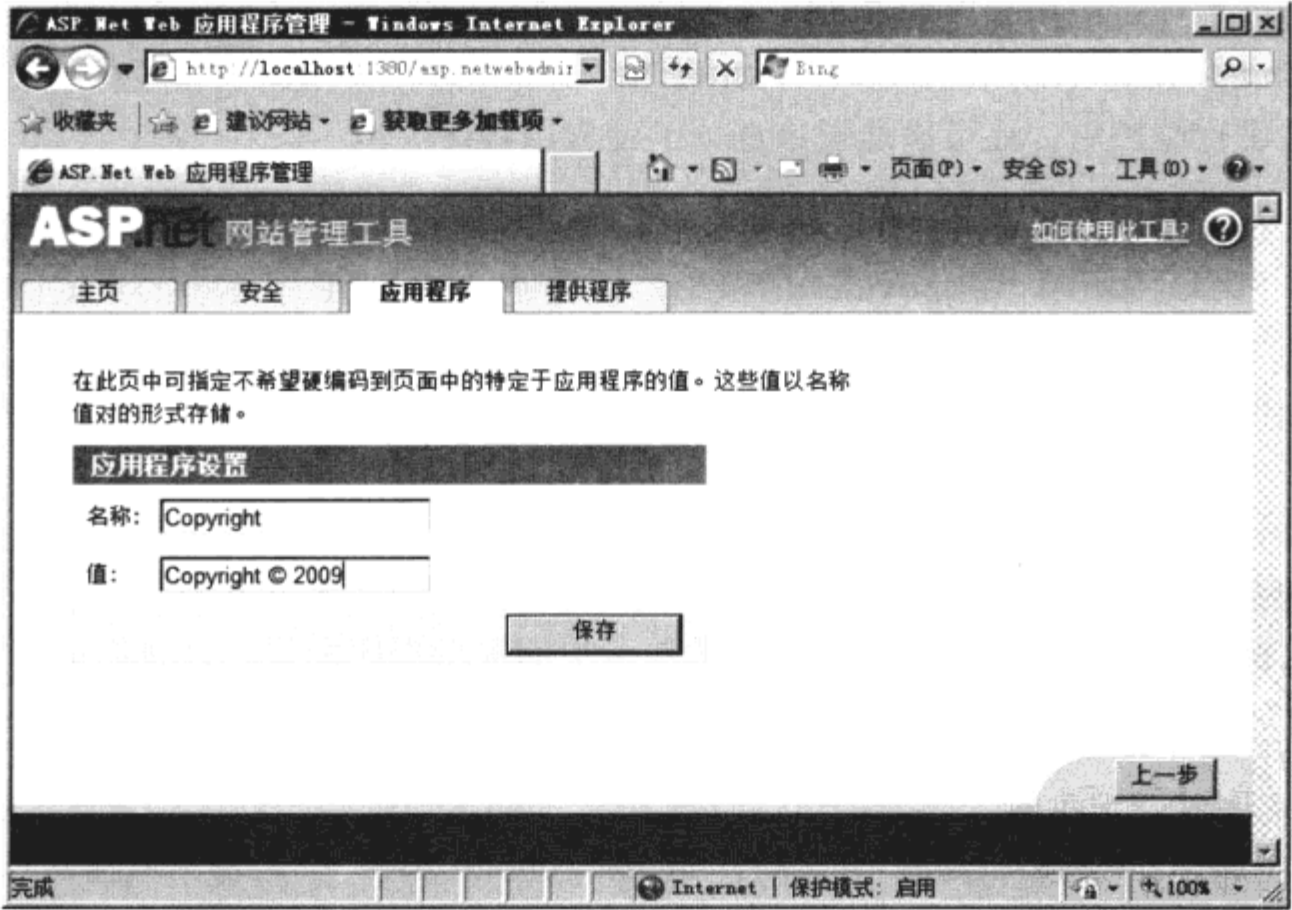
使用“网站管理工具”，我们可以管理部分 web.config，而无需手动输入设置。这个工具可以从 Visual Studio 中访问。Visual Studio 2010 会为每个网站默认创建一个 web.config 文件。但如果由于某种原因导致该文件没被创建，那么“网站管理工具”会自动创建它。在管理身份验证和角色时，该工具还会在网站的 App_Data 文件夹(该文件夹用于存储应用程序的数据)下隐式地创建受 SQL Server Express 支持的数据库。(本书后面会具体介绍 ASP.NET 的个性化、授权和其他功能。)

3. 下面我们在“应用程序”选项卡中，通过“创建应用程序设置”链接为应用程序添加

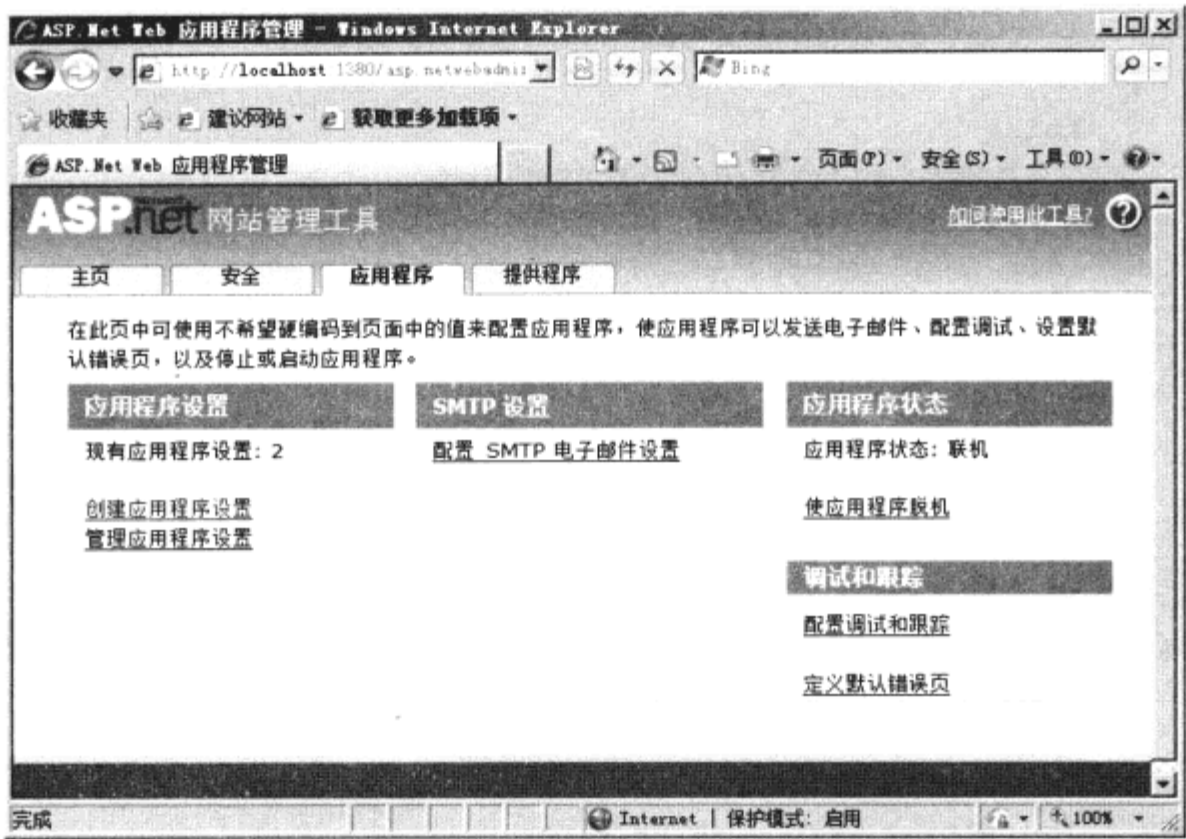
两个设置。这两个设置的名称分别为 Copyright 和 CompanyName(请参考下面几张截图)。在这个练习中，值的选择并不重要。首先，单击“应用程序”选项卡(见下图)。



然后，单击“创建应用程序设置”链接来添加设置(见下图)。



返回“应用程序”选项卡后，页面会显示配置文件中应用程序设置的数目(见下图)。



4. 打开应用程序的 web.config 文件。我们会看到 Copyright 和 CompanyName 数据项。此时，该文件的内容应如下所示(Visual Studio 插入的部分内容已被删减)：

```
<?xml version="1.0" ?>
<configuration>

    <appSettings>
        <add key="Copyright" value="Copyright © 2009 " />
        <add key="Company" value="ThisIsACompanyName" />
    </appSettings>

    <connectionStrings/>
</configuration>
```

5. 下面我们以编程方式来访问刚刚添加的应用程序设置。为此，我们要用到名为 ConfigurationManager 的类。在 Default.aspx 表单上添加一个下拉列表来保存应用程序设置的“键”(将该控件的 ID 设置为 DropDownListApplicationSettings)，用一个标签来显示“值”(将该控件的 ID 设置为 LabelSetting)。添加一个 ID 为 ButtonLookupSetting 的按钮，以便用户可以查找与应用程序设置的键关联的值。在 Page_Load 处理过程中通过 ConfigurationManager 来获取所有应用程序设置：

```
using System.Configuration

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
        {
            foreach (String strKey
```

```
        in ConfigurationManager.AppSettings.AllKeys)
    {
        this.
            DropDownListApplicationSettings.
                Items.Add(strKey);
    }
}

protected void ButtonLookupSetting_Click(object sender, EventArgs e)
{
    string strSetting;
    strSetting =
        ConfigurationManager.AppSettings[this.
            DropDownListApplicationSettings.
                SelectedItem.Text];
    this.LabelSetting.Text = strSetting;
}
}
```

6. 编译并运行网站。在页面被打开时，它会在下拉列表中加载 ConfigurationManager.AppSettings 中的所有键。用户可以通过下拉列表来选择应用程序设置，单击按钮后，上一步插入的代码会查找应用程序设置的值，并将其显示在下方的标签中(见下图)。



如果 ASP.NET 网站受 IIS 承载，那么我们还能够以图形方式来管理应用程序设置。

8.3 在 IIS 中配置 ASP.NET

如果网站在虚拟目录中(通过 IIS)运行，则可以在 IIS 中使用“功能视图”来编辑配置信息。虽然只能在承载 ASP.NET 应用程序的计算机上对其进行配置，但此时可控制的配置要比 Visual Studio 的“网站管理工具”中的多得多。而且配置的更改会立即对 Web 应用程序

生效。

我们将通过下面的练习来了解 IIS 对 ASP.NET 配置的支持。

➤ 使用 IIS 来配置 ASP.NET

- 1. 在创建新网站之前，确保 Visual Studio 是以管理员身份运行的(在“开始”菜单的 Visual Studio 上右击，然后选择“以管理员身份运行”)。为创建(或编辑)由 IIS 直接承载的网站，以管理员身份运行是个前提。将这个网站命名为 ConfigORamaIIS(见下图)。



为使 IIS 直接管理这个网站，可单击“浏览”按钮。随后打开的对话框(见下图)允许我们选择网站的位置。这里选择本地计算机(localhost)。Visual Studio 会为该网站创建一个虚拟目录，并将自身引向这个虚拟目录。



“创建新 Web 应用程序”按钮

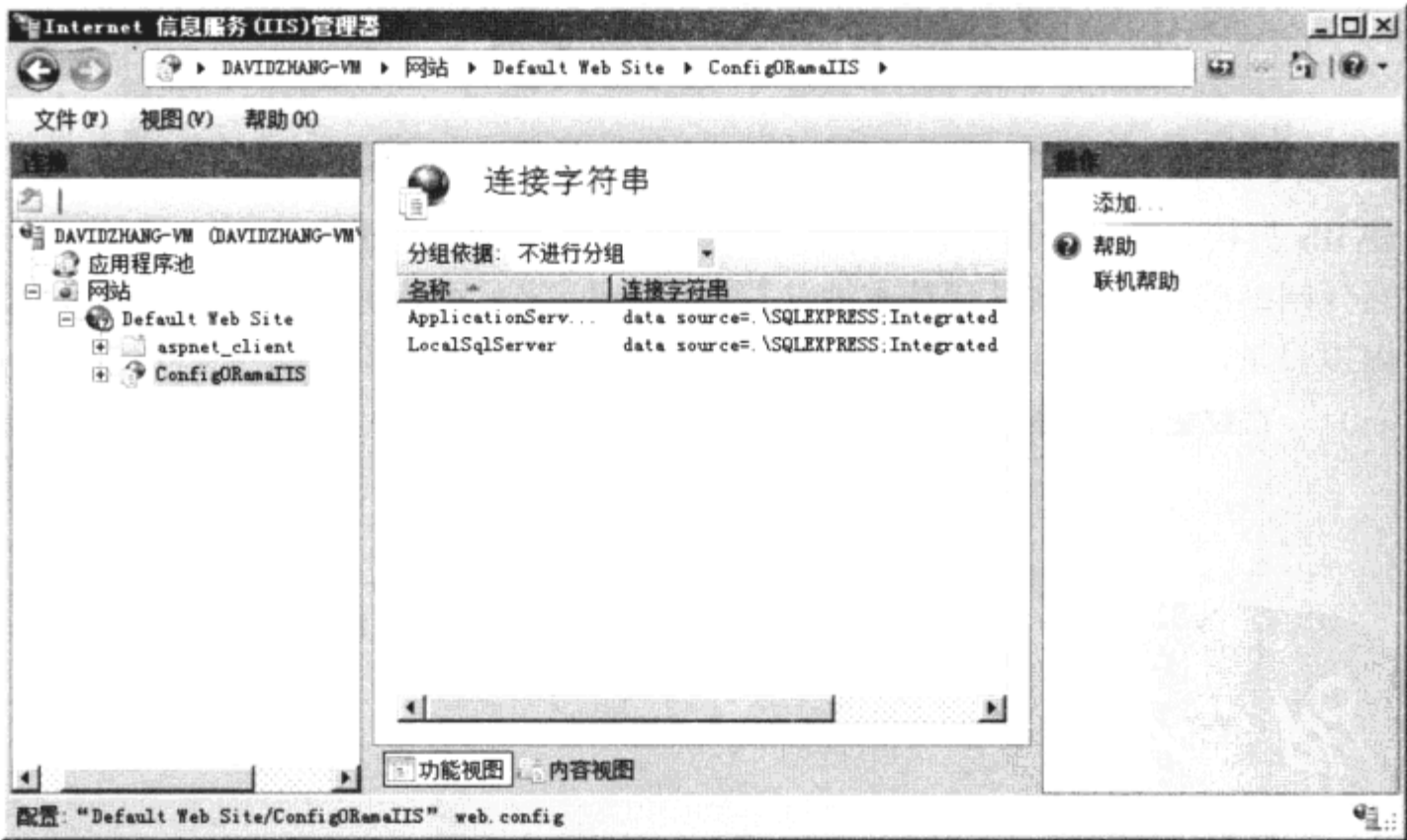
- 2. 打开“Internet 信息服务(IIS)管理器”。为此，可以先打开“控制面板”，然后找到“管理工具”。如果使用 Windows Vista 或 Windows 7，则可以在“系统和安全”设置选项中找到“管理工具”。打开“Internet 信息服务(IIS)管理器”。在“连接”窗格中找到

ASP.NET 4 从入门到精通

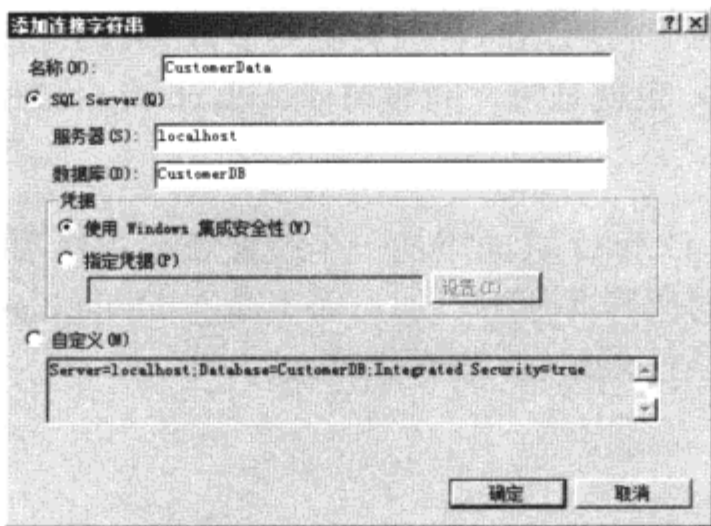
“ConfigORamaIIS” 虚拟目录，单击它。此时，我们会在下图所示的“功能视图”中看到 ASP.NET 相关的设置。



3. 双击一两个功能，查看其配置界面。例如，双击“连接字符串”图标会打开“连接字符串”窗格(见下图)。



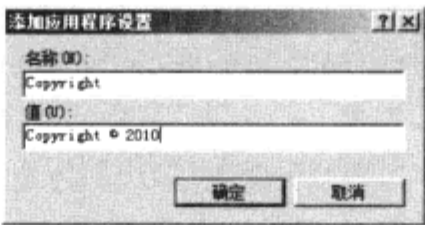
4. 下面将添加一个新的连接字符串。在“连接字符串”窗格的空白区域右击，选择“添加”。我们可以方便地通过随后打开的“添加连接字符串”对话框(见下图)来输入连接字符串信息(下图中数据库的名称是虚构的):



在“功能视图”中除了管理连接字符串，还可以管理应用程序设置。在“功能视图”窗格中双击“应用程序设置”。此时会显示“应用程序设置”窗格，如下图所示。



在“应用程序设置”窗格中的空白区域右击，然后选择“添加”。此时会打开“添加应用程序设置”对话框(见下图)，可以在这里添加应用程序设置——这与“网站管理工具”非常类似。可以像这样添加键/值对。



- 5. 打开应用程序的 web.config 文件。该文件现在应包含一个名为 Copyright 的项：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <appSettings>
    <add key="Copyright" value="Copyright © 2010" />
  </appSettings>
</configuration>
```

- 6. 使用 IIS 的 ASP.NET 配置编辑器添加一个名为 BackgroundColor, 值为 #00FF00 的设置。

添加该设置的目的是为了使网站的管理员能够修改 Default.aspx 的背景颜色(稍后添加修改背景颜色的代码)。也就是说, 能够访问 web.config 文件的人员, 都能够修改背景颜色。

7. 返回 Visual Studio, 为 Default 页面添加一个能够获取背景颜色的属性。(如果使用了母版页, 则应在母版页的代码文件中添加, 而不是在 Default.aspx.cs 中。)该属性应通过 ConfigurationManager.AppSettings 集合来获取。

```
using System.Configuration;

public partial class SiteMaster : System.Web.UI.MasterPage
{
    protected string BackgroundColor
    {
        get
        {
            return ConfigurationManager.AppSettings["BackgroundColor"];
        }
    }

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

8. 在默认情况下, Visual Studio 会在应用程序中包含一个母版页。在“源”视图中打开 Site.master 页面, 更新<body>标签, 使其从应用程序设置中获取背景颜色。通过<%和%>括号来标记可执行的代码。此外, 再为 Site.master 文件添加一行显示背景颜色值的代码。

```
<%@Master Language="C#" AutoEventWireup="true"
    CodeFile="Site.master.cs" Inherits="SiteMaster"%>

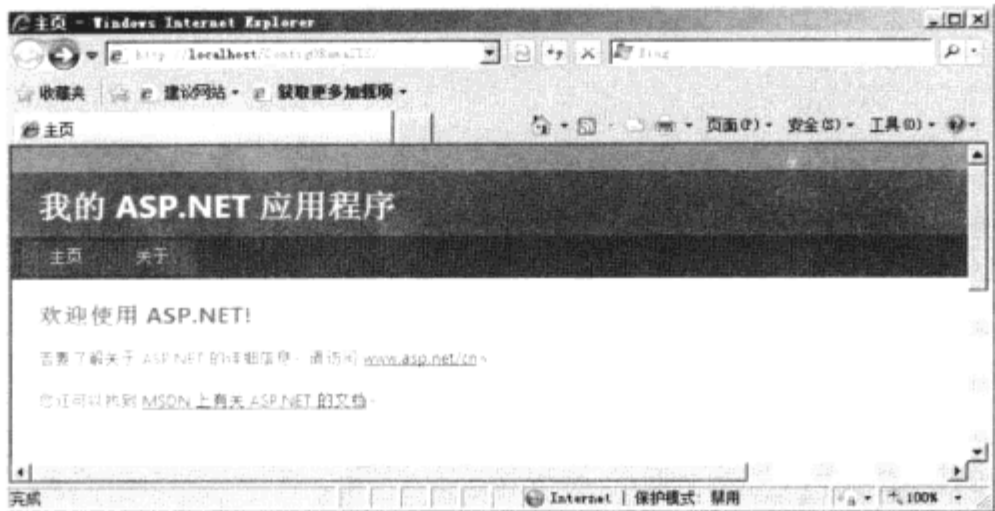
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head runat="server">
    <title></title>
    <link href="~/Styles/Site.css" rel="stylesheet" type="text/css"/>
    <asp:ContentPlaceHolder ID="HeadContent" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body style="background-color: <%=BackgroundColor%>">

<!-- 这里是其他内容 -->
</body>
</html>
```

9. 编译整个项目, 并运行该页面。结果如下图所示。#00FF00 代表绿色, 因而母版页的

背景会呈现这个颜色。



10. 查看 IIS 中针对 ASP.NET 的其他设置。在后续章节继续探索 ASP.NET 时，我们还会遇到其中的许多设置。
- 可以在“.NET 授权规则”窗格中为应用程序设置用户的访问权限并为其分配角色
 - 可以在“.NET 全球化”窗格中管理全球化设置
 - “会话状态”窗格可以用来管理会话状态设置。可以使 ASP.NET 将会话状态存储在任何位置，包括在本地计算机的进程中、通过指定的状态服务器存储在进程外，或者存储在指定的 SQL Server 数据库中
 - “页面和控件”页面允许我们管理应用程序的 UI 设置(如主题和母版页)

ASP.NET 的大部分设置都可以存储在 web.config 文件中。不过关于配置的话题尚未结束。本章只是介绍了其中的几种设置，在本书的后面还会遇到其他设置，如安全性、会话状态、错误消息和 HttpHandler/HttpModule 功能都会用到配置。

8.4 快速参考

目 标	操 作
查看全局配置文件	在 Windows 目录的 Microsoft.NET\Framework\xxxxxx\config 文件夹下查找，其中 xxxxx 代表网站所使用的 ASP.NET 的版本
修改特定 ASP.NET 应用程序的设置	在应用程序目录下添加 web.config 文件，并在该文件中进行设置
修改虚拟目录下子目录的设置	在子目录中添加单独的 web.config 文件，或者在虚拟目录下的 web.config 文件中使用 location 关键字
通过“网站管理工具”(WSAT)来修改应用程序的设置	在 Visual Studio 中，在主菜单中选择“网站”/“项目” “ASP.NET 配置”
通过 IIS 的 ASP.NET 配置工具来修改应用程序的设置	打开 IIS 管理器。选择 Web 应用程序的虚拟目录，双击要查看或修改的设置所对应的图标
获取配置文件中的设置	使用 ASP.NET 中的 ConfigurationManager 类

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569




第 9 章 登 录


学习目标

- 管理基于 Web 的安全性
- 实现 “Forms 身份验证”
- 以编程方式使用 “Forms 身份验证”
- 通过 ASP.NET 中的登录控件来简化登录页面的构建
- 使用 ASP 基于角色的授权方式

本章将介绍 ASP.NET 应用程序访问的控制方法。网站的安全性是绝大多数企业关心的话题。如果不为网站采用任何安全措施，那么网站就会将企业不希望公开的部分暴露出去。本章将简单介绍与 Web 应用程序相关的安全措施，以及 ASP.NET 在用户身份验证与授权方面提供的各种服务。

 **提示** 用户身份验证的意思是判断用户是否的确是他们所声称的人(验证用户的身份)。这一般是通过某种达成共识的秘密信息(如密码)来实现的。用户的授权是根据用户的身份或为其分配的角色来允许或限制用户的访问。例如，相比只拥有普通用户角色的用户，拥有管理员角色的用户往往能够访问更多内容。

本章还会介绍几种新的登录控件。在保护网站方面，这些控件能够在很大程度上降低开发者的工作量。最后，本章会介绍 ASP.NET 对授权的支持(为用户分配角色)。

 **注意** 执行本章配套的示例代码需要 IIS 的支持。有关运行示例代码的注意事项，请阅读本书前言“示例代码”部分。

9.1 基于 Web 的安全性

软件安全一直都是一个重要的话题，尤其是在当今公众的安全意识(如隐私)逐渐增强的情况下。如果 Web 应用程序运行于 Microsoft 的软件环境下，那么安全方面需要关注：(1)IIS 安全性上下文；(2)确认客户端的身份；(3)指定客户端对应用程序能做什么，不能做什么。

管理一般网络的安全性涉及用户的身份验证和授权，而管理基于 Web 的安全性与之类似。不过，基于 Web 的安全性需要管理开放式系统的不同平台上运行的客户端。也就是说，基于 Web 平台的编程需要处理来自客户端的请求，而这些客户难以像封闭式网络(如基于

Windows 系统的办公网络)那样控制。

虽然配置 Windows 的安全性并不简单,但至少已经有了解决方案。搭建过 Windows 网络的人员一般都知道,为所有用户正确设置权限会牵扯许多问题。Windows 网络是一种封闭式系统,网络上的所有用户都处于连接状态,最初只具有基本的信任级别(也就是说,用户都属于该网络)。在登录 Windows 网络时,用户应通过用户名和密码来证明其身份(身份验证)。如果安全子系统确认该用户的身份,那么它会为 Windows 会话颁发安全令牌,该用户运行的所有程序都将使用这个安全令牌。

计算机和网络上的资源(如文件、文件夹、驱动器、应用程序等)都与各自的“自由访问控制列表”(discretionary access control lists, DACL)关联。如果应用程序所处的安全上下文包含在某个资源的 DACL 中,那么它就可以访问该资源。否则,系统会阻止它使用这个资源。这种机制通常被称为授权。

对于 Windows 网络这种封闭式系统,管理员能够有效地控制整个系统,并能够为用户分配各种资源的访问权限。由于是封闭式的,因此系统很容易判断用户是否属于本系统,以及该用户什么能做,什么不能。

而对于 Web 应用程序而言,其用户非常广泛,并且不必是本地系统的一部分。这也就意味着,我们需要另一种(在 Windows 基础设施之外)对使用 Web 应用程序的用户进行身份验证和授权的方法。换言之,Windows 身份验证不适用于一般的 Internet 环境。

9.1.1 IIS 的保护

编写 Windows 环境中的 Web 应用程序,首先要理解的是 IIS 的安全上下文。几乎网站的所有访问都要通过 IIS 进行定向。与 Web 应用程序一样,IIS 也运行在一个特定的环境下。在计算机上安装 IIS 时,安装程序会特别为 IIS 创建一个单独的安全标识帐户。

我们可以通过以下操作来查看运行 IIS 所使用的帐户:在“控制面板”中打开“Internet 信息服务(IIS)管理器”,选择某个虚拟目录,在“功能视图”中,双击“身份验证”图标。此时会打开“身份验证”窗格。在“匿名用户标识”上右击,选择“设置”。在笔者的计算机上,这个用户的名称为“IUSR”(如图 9.1 所示)。

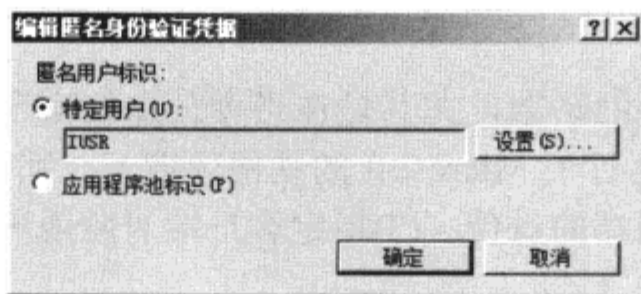


图 9.1 在 IIS 中管理身份验证设置

在默认情况下,IIS 会为虚拟目录采用“匿名身份验证”。如果采用这种身份验证方法,IIS

会使用“特定用户”字段指定的标识来作为安全主体 (principal)。也就是说，IIS 只允许用户访问允许 IUSR 访问的资源。

IIS 也支持其他类型的身份验证，其中包括“Windows 身份验证”。如果使用这种身份验证方式，则需要为所有可能的客户端分配 Windows 用户名和密码。只有在基于 Windows 的平台上运行的客户端才支持这种身份验证方式。网站会提示用户输入用户名和密码，以便对其进行身份验证。在登录网站时，用户会看到一个 Windows 登录对话框(读者可能遇到过这种网站)。Windows 身份验证适用于企业级的网站，因为此时可以确保网站的用户使用的都是基于 Windows 的浏览器。然而，对于那些不使用 Windows 操作系统的用户，则需要采用其他身份验证方式。这是因为其他操作系统不具备 Windows 的底层安全机制，因而不能对这些系统的用户进行身份验证。^①

幸运的是，ASP.NET 支持“Forms 身份验证”——一种能够方便地对客户端进行身份验证的方法。ASP.NET 1.0 和 1.1 中的“Forms 身份验证”子系统是对之前技术的巨大改进，免去了开发者自行编写身份验证子系统的麻烦。ASP.NET 的后续版本也包含“Forms 身份验证”模型，而且还添加了“授权”子系统，对之前的身份验证模型做了改进。

下面先让我们了解一下基本的“Forms 身份验证”。

9.1.2 基本的“Forms 身份验证”

ASP.NET 1.0 和 1.1 引入了一种简单的用户身份验证方法。这种“Forms 身份验证”受应用程序中 web.config 文件的驱动。除控制会话状态、跟踪、调试和应用程序设置外，web.config 还可以包含与身份验证和授权相关的节点。

实现用户身份验证只需要在 web.config 文件中添加一些指令。我们可以直接编辑这个文件，也可以使用 Microsoft Visual Studio 的“网站管理工具”。(有关“网站管理工具”的详细内容，请阅读第 8 章。)

web.config 包含用于指定网站如何处理身份验证和授权的节点。如果没有这两个节点，ASP.NET 就不会限制用户对网站的访问。不过，如果在 web.config 文件中添加了这两个节点，ASP.NET 便会强制将用户重定向到一个用于进行身份验证的 URI。在大多数情况下，这个 URI 是 Web 应用程序中的一个登录页面，用户必须在该页面接受身份验证(如输入用户名和密码)。

在阅读实际的代码之前，先来看一下图 9.2。此图展示了通过 web.config 启用“Forms 身份验证”后，网站的控制流程。

^① 译者注：在基于 Windows 的企业环境中，用户的凭据是统一管理的(使用 Active Directory)。用户的身份信息可供企业中的所有系统共享，而用户身份验证也受专门的服务器支持。这样用户便不必每次都进行注册，极大地提高了生产效率。

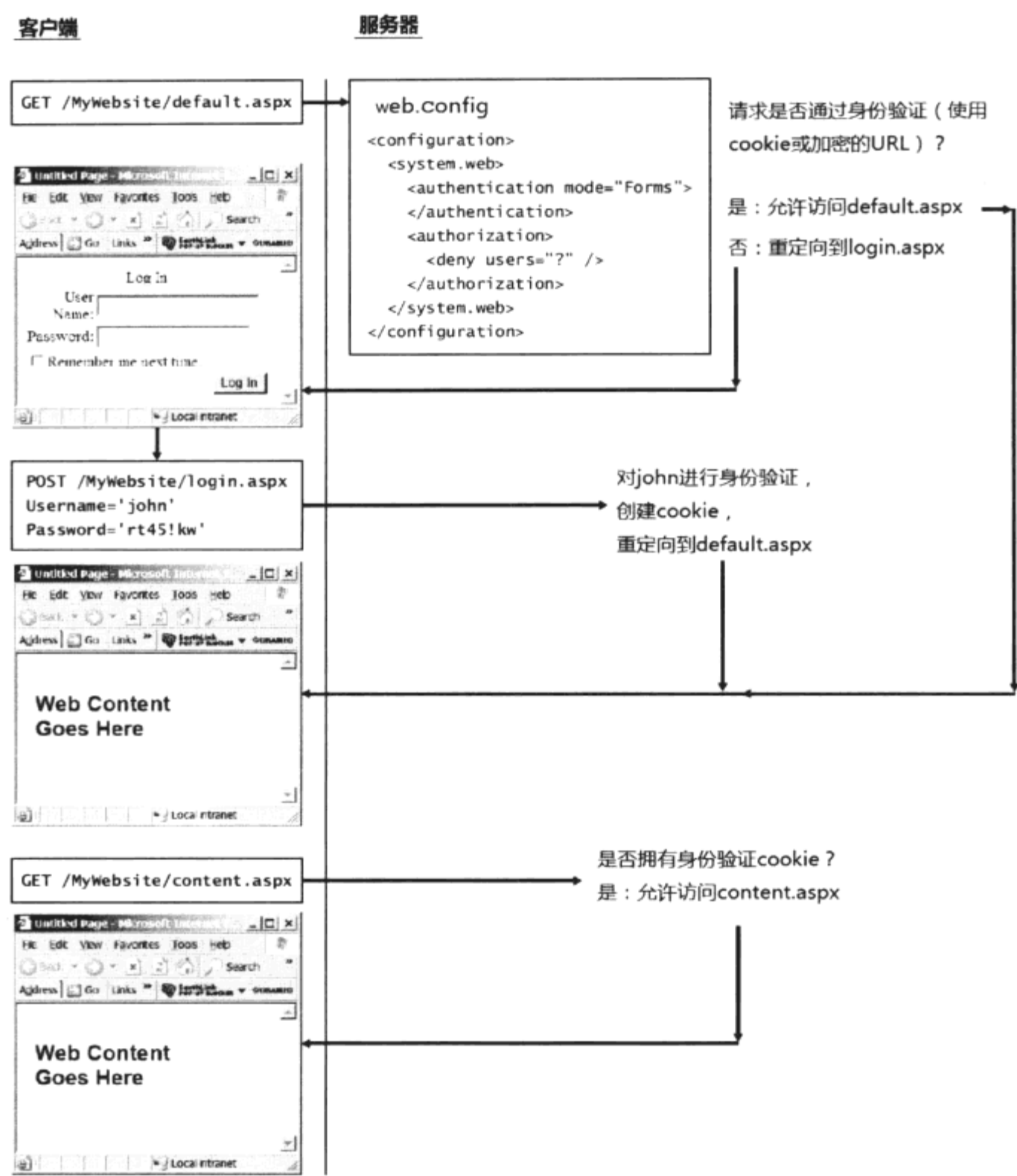


图 9.2 网站“Forms 身份验证”的控制流程

本书的配套资源包含这个登录页面。如果希望在应用程序中使用这种最基本的身份验证方法，则可以参考 Login.aspx 和 web.config 文件。为支持“Forms 身份验证”，web.config 应包含 authentication 和 authorization 节点。清单 9.1 给出了为启用这种身份验证而需要在 web.config 中添加的设置。

清单 9.1 为启用“Forms 身份验证”而需要在 web.config 文件中添加的设置

```
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms loginUrl="login.aspx" />
    </authentication>

    <authorization>
```

```

        <deny users="?" />
    </authorization>
</system.web>
</configuration>

```

清单 9.2 给出了相应的登录页面的代码。

清单 9.2 基本的 ASP.NET 登录页面

```

<%@Page language="C#" %>
<%@Import namespace="System.Web.Security" %>

<html>
    <script runat="server">

protected bool AuthenticateUser(String strUserName,
                                String strPassword)
{
    if (strUserName == "Gary") {
        if (strPassword == "K4T-YYY") {
            return true;
        }
    }
    elseif (strUserName == "Jay") {
        if (strPassword == "RTY!333") {
            return true;
        }
    }
    else if (strUserName == "Susan") {
        if (strPassword == "erw3#54d") {
            return true;
        }
    }
    return false;
}

public void OnLogin(Object src, EventArgs e) {
    if (AuthenticateUser(m_textboxUserName.Text,
                        m_textboxPassword.Text)) {
        FormsAuthentication.RedirectFromLoginPage(
            m_textboxUserName.Text, m_bPersistCookie.Checked);
    } else {
        Response.Write("Invalid login: You don't belong here...");
    }
}
</script>
<head>
    <title>Login Page</title>
</head>
<body>
    <form runat="server">
        <h2>A most basic login page</h2>
        User name:
        <asp:TextBox id="m_textboxUserName" runat="server"/><br/>

```

```

Password:
<asp:TextBox id="m_textboxPassword"
    TextMode="password" runat="server"/>
<br/>
Remember password and weaken security?:
<asp:CheckBox id="m_bPersistCookie" runat="server"/>
<br/>
<asp:Button Text="Login" OnClick="OnLogin"
    runat="server"/>
<br/>
</form>
</body>
</html>
```

这个简单的登录页面能够验证 3 个用户——Gary、Jay 和 Susan。

在这个示例中，如果用户要访问虚拟目录中的页面，ASP.NET 会先阻止他们，并将他们重定向到一个登录页面(如图 9.3 所示)。



图 9.3 从客户端获取用户名和密码的简单登录页面

这个简单的登录页面能够对用户进行身份验证(使用 3 个账户之一)。在实际的网站中，所采用的身份验证算法可能会通过数据库来查找用户是否包含在其中，并判断其密码是否匹配。本章后面会介绍 ASP.NET 身份验证服务。这个登录页面使用了工具类 FormsAuthentication(位于 System.Web.Security 命名空间下)来创建身份验证 Cookie。

图 9.4 展示了启用跟踪后的页面。我们可以在“请求 Cookie 集合”中找到身份验证 Cookie 的值。

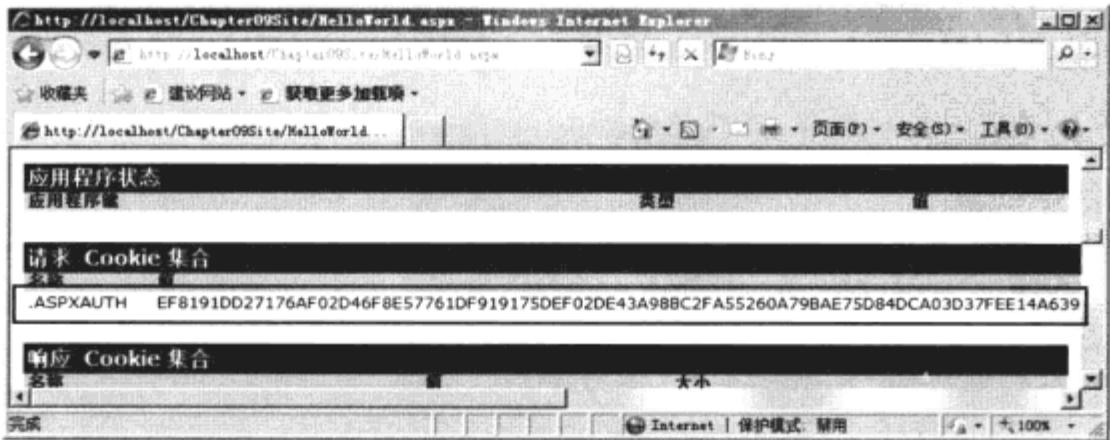


图 9.4 页面跟踪中可以看到为“Forms 身份验证”创建的身份验证 Cookie

下面我们来看看如何为网站启用“Forms 身份验证”。

► 运行“Forms 身份验证”示例

1. 为运行“Forms 身份验证”示例(一个网站)，需要创建一个用于承载它的 IIS 应用程序。这个示例已尽可能地被简化(ASP.NET 需要一些目标和版本信息)，因此该网站要基于 ASP.NET 2.0 版本运行。在 IIS “连接”窗格中右键单击网站目录，选择“管理应用程序”|“高级设置”。将“应用程序池”属性设置为“Classic .NET AppPool”。在当前目录中添加一个名为 Default.htm 的 HTML 文件，简单地使其显示“Hello World”。为使“Forms 身份验证”能够真正生效，需要浏览这个网页(作为目标页面)。
2. 将配套资源中本章示例代码中的 Login.aspx 页面复制到刚刚创建的虚拟目录下。
3. 将配套资源中本章示例代码中的 Web.ConfigForceAuthentication 文件也复制到该目录下。在复制完毕后，将该文件重命名为 web.config。
4. 尝试浏览这个虚拟目录中的页面。在呈现实际内容之前，ASP.NET 会将浏览器重定向到 Login.aspx 页面，让用户填写身份信息。
5. 输入一对有效的用户名和密码。(可以从第 2 步添加的 Login.aspx 页面中选择一对用户名和密码。)对这个虚拟目录的后续访问将变得顺畅，因为此时身份验证票证(Authentication ticket)已与请求和响应关联。

虽然可以自行构建身份验证算法，但 ASP.NET 提供了许多特性，能够以直观而标准的流程来验证用户身份。稍后会介绍这方面内容。

ASP.NET 还支持另外两种身份验证方法——“Passport 身份验证”和“Windows 身份验证”，在这里简单介绍一下。“Passport 身份验证”已经演变为 Windows Live ID，它需要使用 Microsoft 提供的一种集中式身份验证服务。曾经使用过 Hotmail.com 的话也就接触过 Windows Live ID 了。Windows Live ID 身份验证的优势在于，它是将登录信息和个性化信息集中存储在一处。虽然这项服务并不免费，但用户可以在不同网站中使用统一的用户 ID。Windows Live ID 提供了许多便利，开发者不再需要自行管理用户的身份验证，从而降低了开发成本。

“Windows 身份验证”也是 ASP.NET 支持的一种身份验证方式。如果指定 Windows 身份验证，那么 ASP.NET 要依赖于 IIS 和 Windows 身份验证来管理用户。任何通过 IIS 进行身份验证的用户(使用基本身份验证、摘要式身份验证或集成 Windows 身份验证)都将授权网站的访问。为支持其他身份验证方式，可以对 IIS 进行配置。对于大多数 ASP.NET 网站来说，如果只是为了获得更好的伸缩性，可以用 ASP.NET 身份验证来替代 IIS 身份验证。ASP.NET 能够根据通过身份验证的标识来管理授权。

9.2 ASP.NET 身份验证服务

ASP.NET 包含许多对用户身份验证方面的支持(不需要 IIS 参与)，其中的大部分来源于 FormsAuthentication 类。这个类支持哈希密码(能够安全存储密码)，支持字符串的加密与解密，支持创建身份验证 Cookie，支持为验证用户身份而对请求进行重定向，还支持管理身份验证参数(如过期时间)。

9.2.1 FormsAuthentication 类

许多 ASP.NET 身份验证服务都是围绕 FormsAuthentication 类进行的。清单 9.1 和清单 9.2 给出的示例展示了一种简单的身份验证方法，能够在响应中添加身份验证 Cookie，并将用户重定向回所请求的页面。这正是 FormsAuthentication.RedirectFromLoginPage 方法实现的功能。FormsAuthentication 类还包括其他方法，可以对身份验证过程进行粒度更细的控制。例如，我们能够以编程方式对用户进行身份验证，而不强制执行重定向。这种特性适合创建可选的登录页面，其内容可以根据客户端身份验证级别的不同而不同。

FormsAuthentication 还具有许多其他服务。表 9.1 给出了 FormsAuthentication 类的一些常用成员。

表 9.1 FormsAuthentication 类的常用成员

成 员	说 明
CookiesSupported	用于指示当前身份验证方法是否支持 Cookie
FormsCookieName	包含“Forms 身份验证”Cookie 的名称
FormsCookiePath	包含“Forms 身份验证”Cookie 的路径
LoginUrl	登录页面的 URL
RequireSSL	指示是否使用“安全嵌套字层”(SSL)
SlidingExpiration	指示可调过期是否被设置
Authenticate	用于对用户进行身份验证
Encrypt	用于生成代表“Forms 身份验证”票证的加密字符串，该字符串适合在 HTTP Cookie 中使用
Decrypt	用于对已加密的“Forms 身份验证”票证进行解密
GetAuthCookie	为特定用户创建身份验证 Cookie
GetRedirectUrl	用于获取客户端原始请求的 URL
HashPasswordForStoringInConfigFile	用于生成适合存储在凭据存储区中的哈希密码
RedirectFromLoginPage	用于将通过身份验证的用户重定向回原来请求的页面
SignOut	用于使身份验证票证失效

9.2.2 可选的登录页面

本书的配套资源中包含一个示例，演示了如何单独进行身份验证。清单 9.3 中的页面使用了之前硬编码密码的身份验证算法(只支持 Gary、Jay 和 Susan 这 3 个用户)。不过，该页面会对用户进行身份验证，并将其重定向到当前页面(OptionalLogin.aspx)。

清单 9.3 OptionalLogin.aspx

```
<%@Page language="C#"trace="false"%>
<html>
  <script runat=server>

    protected bool AuthenticateUser(String strUserName,
                                    String strPassword)
    {
        if (strUserName == "Gary.")
        {
            if(strPassword== "K4T-YYY")
            {
                return true;
            }
        }
        else if (strUserName == "Jay")
        {
            if(strPassword== "RTY!333")
            {
                return true;
            }
        }
        else if(strUserName == "Susan")
        {
            if(strPassword== "erw3#54d")
            {
                return true;
            }
        }
        return false;
    }

    public void OnLogin(Object src, EventArgs e) {
        if (AuthenticateUser(m_textboxUserName.Text,
                              m_textboxPassword.Text))
        {
            FormsAuthentication.SetAuthCookie(
                m_textboxUserName.Text,
                m_bPersistCookie.Checked);
            Response.Redirect("optionallogin.aspx");
        }
    }
  }
</script>
</html>
```



```

    } else {
        Response.Write("Invalid login: You don't belong here...");
    }
}

protected void ShowContent()
{
    if(Request.IsAuthenticated)
    {
        Response.Write("Hi, you are authenticated. <br>" );
        Response.Write("You get special content...<br>" );
    } else
    {
        Response.Write("You're anonymous. Nothing special for you... ");
    }
}
</script>
<body><form runat=server>

<h2>Optional Login Page</h2>

    User name:
    <asp:TextBox id="m_textboxUserName" runat=server/><br>
    Password:
    <asp:TextBox id="m_textboxPassword"
        TextMode="password" runat=server/>
    <br/>
    Remember password and weaken security?:
    <asp:CheckBox id= "m_bPersistCookie" runat="server"/>
    <br/>
    <asp:Button text="Login" OnClick="OnLogin"
        runat=server/>

    <br/>

    <%ShowContent(); %>
</form></body>
</html>

```

请注意，这个页面采用编程方式来实现页面的身份验证。它的做法是调用 `FormsAuthentication.SetAuthCookie`，并将请求重定向回原页面。在每次要显示页面内容时，页面都会调用 `ShowContent` 方法。该方法会检查页面的身份验证属性，以此来判断是否显示为通过身份验证的用户所准备的内容。由于页面要在完成身份验证后手动进行重定向，因此需要特别修改 `web.config` 文件。为使页面正确运行，应保留 `authentication` 节点，但需要移除拒绝匿名用户的指令。这样，任何用户都能够访问 `OptionalLogin.aspx`(而不会被拒绝)，并在通过身份验证后继续往下。清单 9.4 给出了这里使用的 `web.config` 文件。在配套资源中，该文件的名称为 `Web.ConfigForOptionalLogin`。为使其生效，应在复制该文件后将

其重命名为 web.config。

清单 9.4 支持可选登录的 web.config 文件

```
<configuration>
  <system.web>
    <authentication mode="Forms">
    </authentication>
  </system.web>
</configuration>
```

图 9.5 展示了这个可选登录页面在用户通过身份验证之前显示的内容。



图 9.5 可选登录页面在用户通过身份验证之前显示的内容

下面给出了具体运行这个可选登录页面的方法。

➤ 运行示例中的可选登录页面

1. 为运行这个可选登录页面，需要创建一个虚拟目录。或者，也可以将该页面添加到现有的虚拟目录中。
2. 从配套资源中本章的示例代码中将 OptionalLogin.aspx 页面复制到这个虚拟目录下。
3. 从配套资源中本章的示例代码中将 Web.ConfigForOptionalLogin 文件复制到这个虚拟目录下。记住将它重命名为 web.config，以便 ASP.NET 能够加载相应的配置信息。
4. 尝试浏览这个虚拟目录中的某个页面。ASP.NET 应该会允许我们查看这个页面，但会按匿名用户对待。
5. 输入有效的用户名和密码。此时我们会看到为通过身份验证的用户所准备的内容。网站后续的请求/响应都会与同一身份验证票证关联，因此之后总是能够顺畅地看到这些内容。

在用户通过身份验证后，这个可选登录页面会显示为通过身份验证的用户所准备的内容。图 9.6 展示了用户通过身份验证(登录)后的页面。



图 9.6 用户通过身份验证(登录)后的页面

9.3 用户的管理

目前为止，我们可以看到，“Forms 身份验证”的基础内容相对容易掌握。在之前的示例中，用户经过身份验证后才能够访问页面。这些示例演示了最基本的身份验证方法——将用户名和密码硬编码在 ASPX 文件中。这样做只是为了演示，而在生产环境中，无疑要为授权访问网站的用户分配标识。

ASP.NET 和 Visual Studio 包含管理用户标识和角色的工具。在下面的练习中，我们将学习如何建立一个安全的网站——仅当用户验证了其身份后才允许访问。

➤ 管理用户的访问

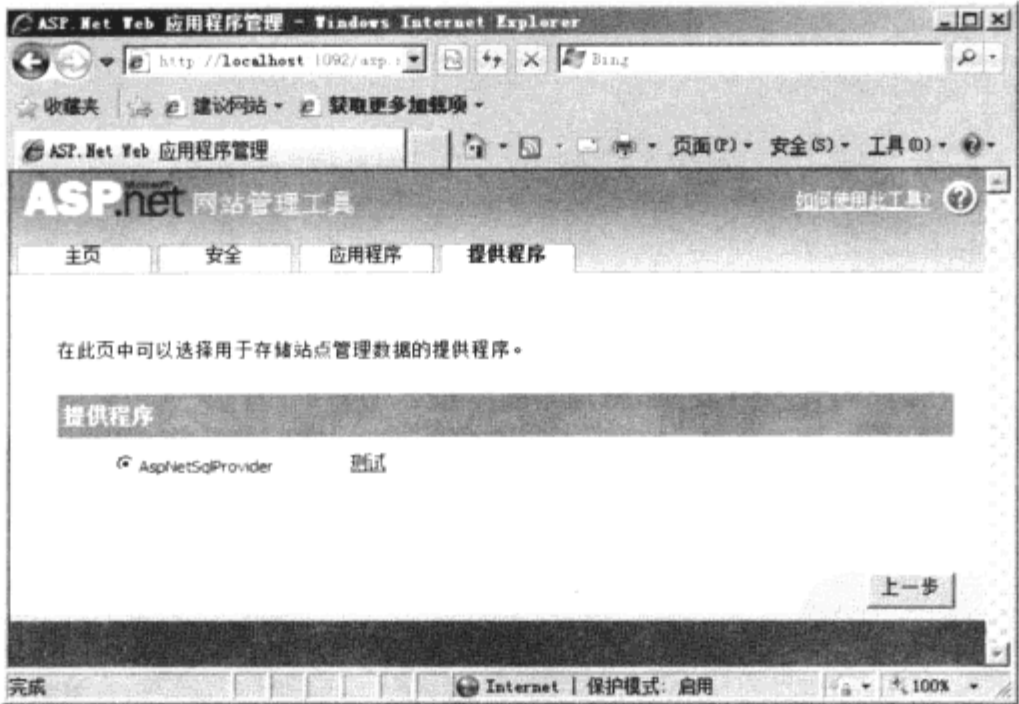
- 1. 创建一个名为 SecureSite 的“ASP.NET 应用程序”。
- 2. 为 Default.aspx 页面添加一个标签，将其文本设置为“Congratulations. You made it in.”。这样，我们便能够在登录后认出这个默认页面。
- 3. 在主菜单中选择“项目”|“ASP.NET 配置”，打开“网站管理工具”。单击“提供程序”选项卡。单击“为所有站点管理数据选择同一提供程序”链接。单击“测试”链接来对相应的提供程序进行测试，以确保能够连接到数据源。^①

技巧 第 8 章介绍过，IIS 中也包含 ASP.NET 配置工具。如果网站位于虚拟目录下，则可以打开 IIS 管理器，选择相应的虚拟目录，然后在“功能视图”中进行配置。

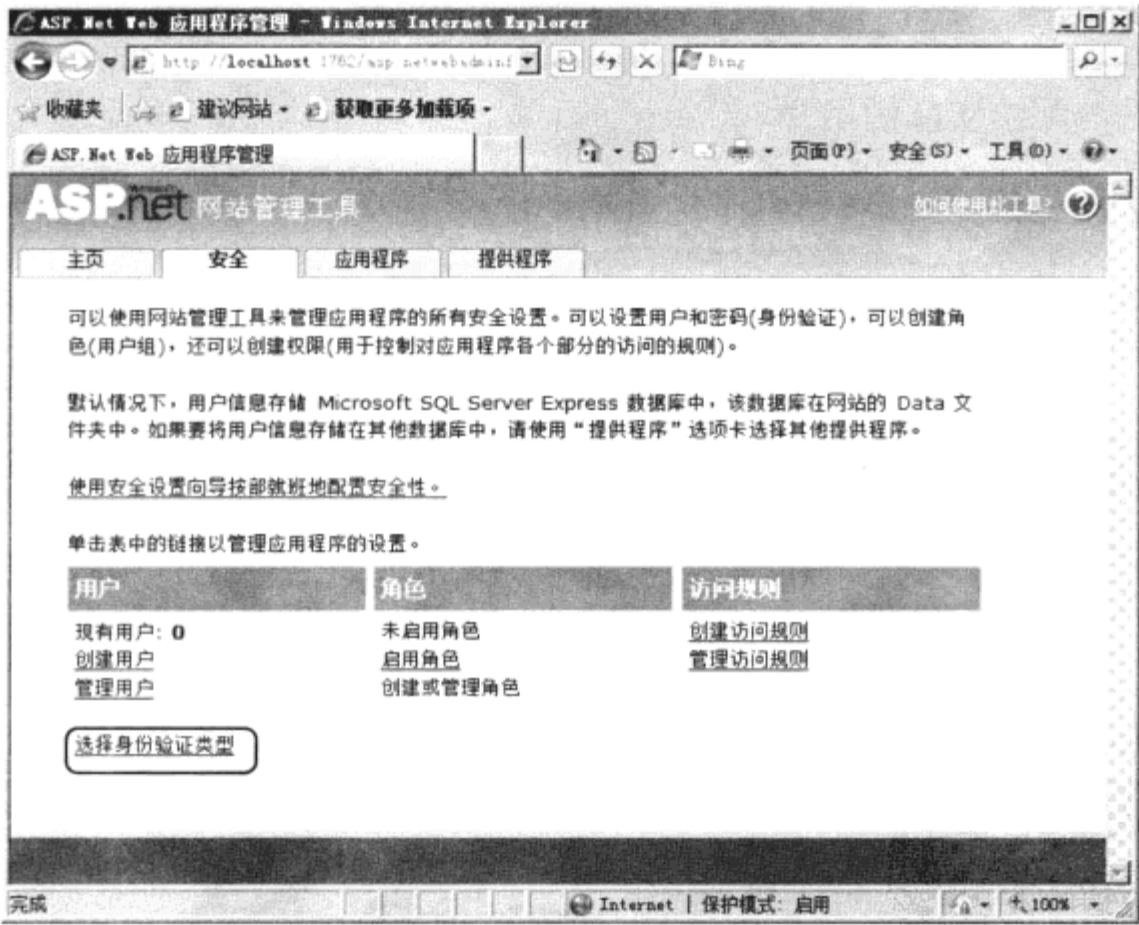
- 4. 为确保数据库正常工作，单击“安全”选项卡，并在此直接添加一个用户。这样，相应的 SQL Express 数据库就会被创建。另外，也可以使用 aspnet_regsql.exe 来创建存储成员资格信息的数据存储。该工具位于 C:\Windows\Microsoft.NET\Framework\

^① 译者注：在测试后，“网站管理工具”可能会提示“未能建立数据库连接”。这是因为数据库不存在或未能成功创建。ASP.NET 能够在运行网站后第一个用户注册时创建数据库。如果使用第 4 步中的方法仍不能通过测试，不妨尝试这种方法。

v4.0.30319^① 目录下。

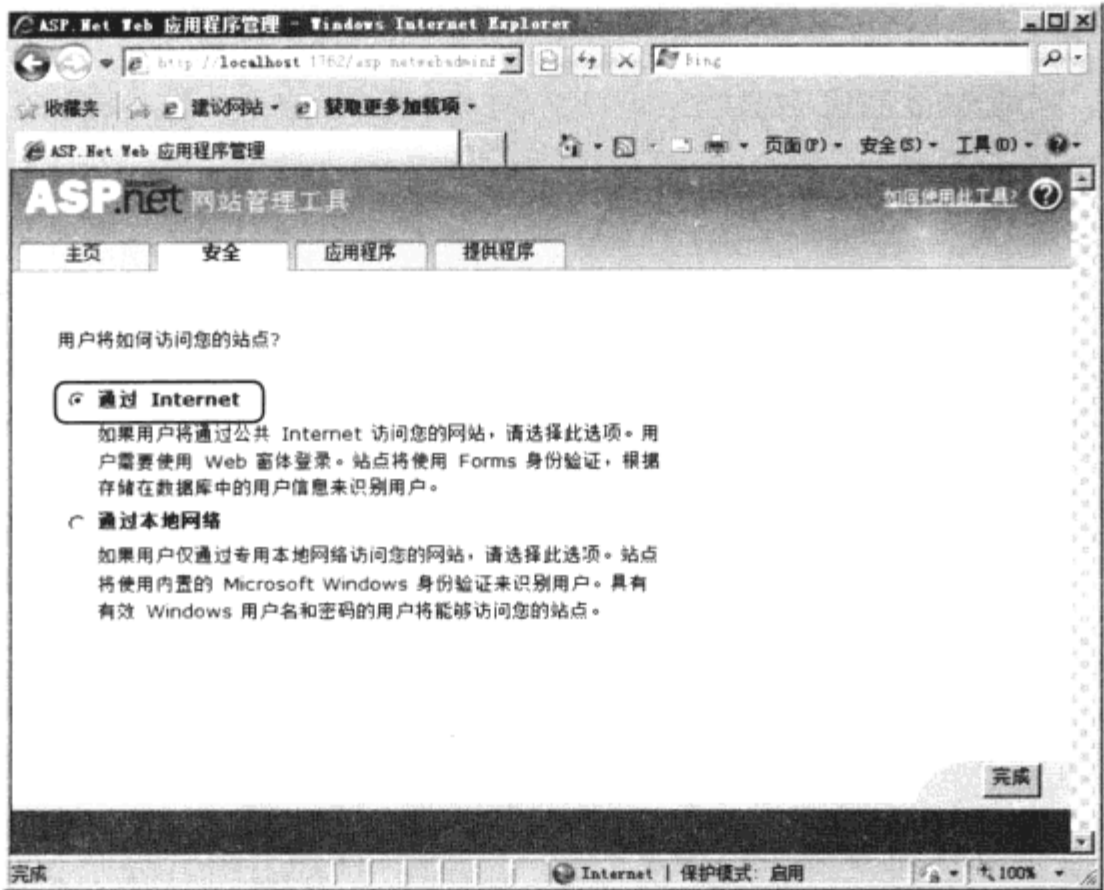


5. 单击“安全”选项卡来修改身份验证类型。此时会看到下图所示的页面。单击“选择身份验证类型”链接。(Visual Studio 可能已经选择了“Forms 身份验证”。)



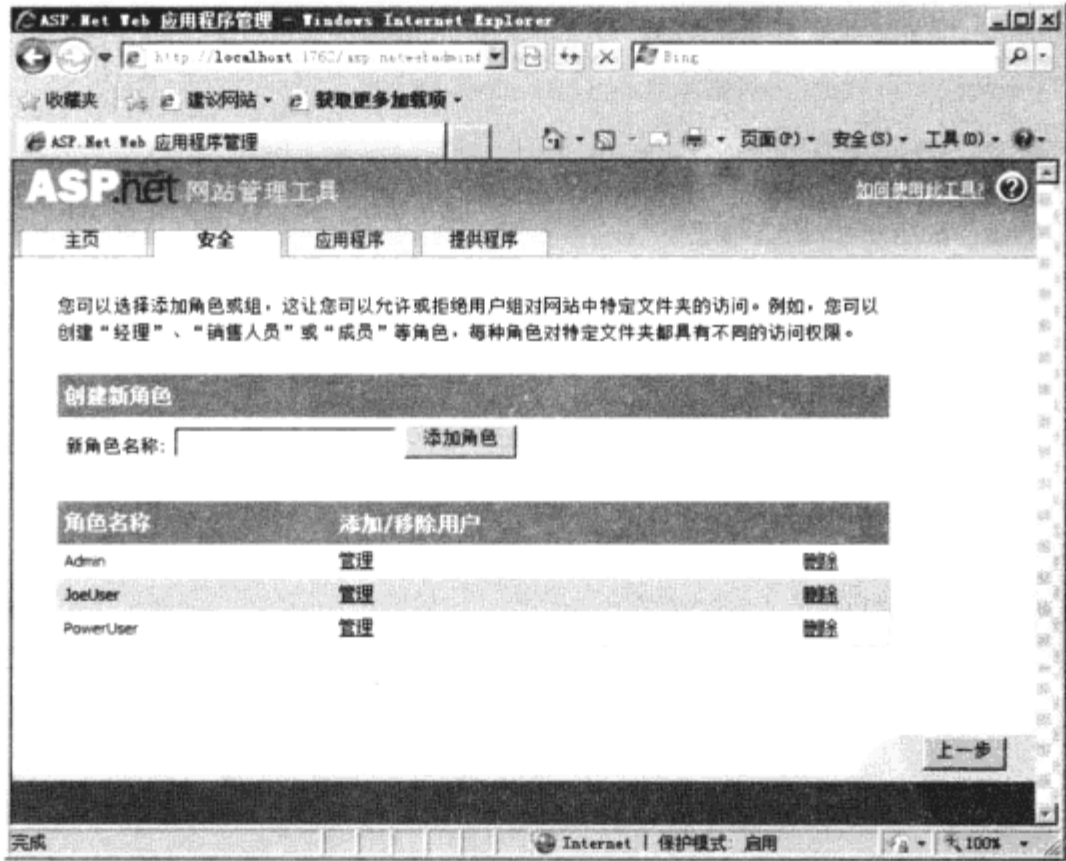
6. 确保“通过 Internet”选项已被选中(如下图所示)。然后，单击“完成”按钮。这样会使网站采用“Forms 身份验证”。

① 译者注：这里根据 Visual Studio 2010 RTM 版本对应的 .NET Framework 修正路径。

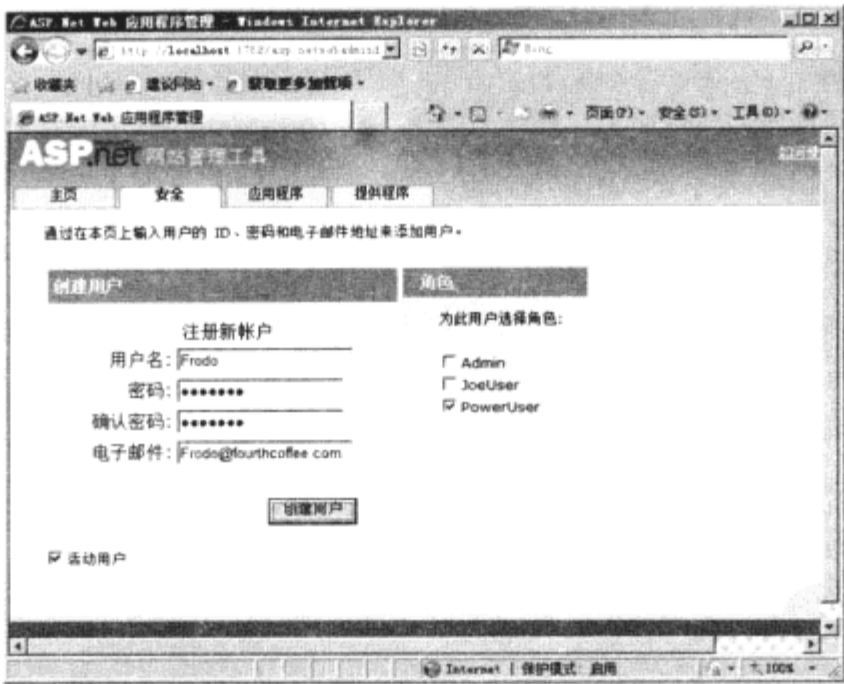


7. 单击“启用角色”链接，然后选择“创建或管理角色”。为网站添加几个角色。本示例为网站添加了 3 个角色：Admin、JoeUser 和 PowerUser。我们先在这里添加角色，稍后会将其分配给实际的用户。

下图展示了角色创建完毕后的页面。



8. 在“安全”选项卡中，单击“创建用户”链接并添加一些用户。如果愿意，我们可以在这里为用户分配角色(见下图)。



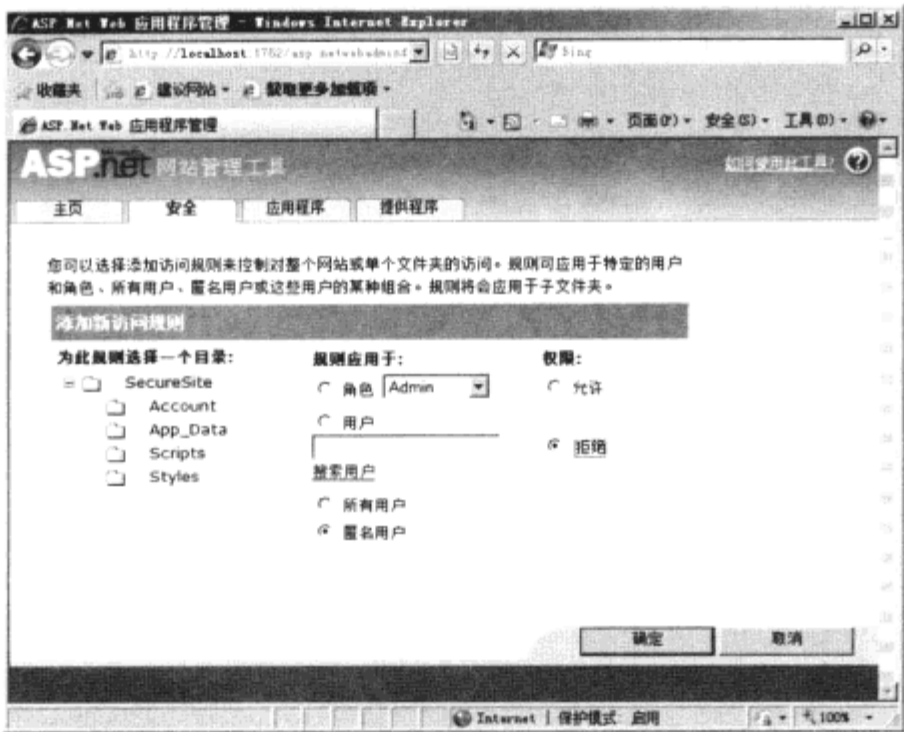
在经过一番配置后，我们应该能够在 web.config 文件中找到有关身份验证的设置。(稍后会对 authorization 节进行设置。)

```
<?xml version="1.0"?>
<configuration>
  <!-- other config info -->
  <system.web>

    <!-- other config info -->
    <authentication mode="Forms">
      <forms loginUrl="~/Account/Login.aspx" timeout="2880" />
    </authentication>
    <membership>
      <providers>
        <clear/>
        <add name="AspNetSqlMembershipProvider"
            type="System.Web.Security.SqlMembershipProvider"
            connectionStringName="ApplicationServices"
            enablePasswordRetrieval="false"
            enablePasswordReset="true"
            requiresQuestionAndAnswer="false" requiresUniqueEmail="false"
            maxInvalidPasswordAttempts="5"
            minRequiredPasswordLength="6"
            minRequiredNonalphanumericCharacters="0" passwordAttemptWindow="10"
            applicationName="/" />
      </providers>
    </membership> </system.web>
  </configuration>
```

9. 至此，我们已经可以验证用户身份了，但可能还需要控制网站中的哪些区域允许(授权)用户访问。为管理授权，需要创建一些访问规则。在“安全”选项卡中单击“创建访问规则”链接。^①拒绝匿名用户的访问，如下图所示。

① 译者注：如果拒绝匿名用户访问网站的根目录，那么用户就无法访问设置网站样式的样式表(Styles\Site.css)。为了网站的美观，可以授权匿名用户访问该文件或该文件所在目录。

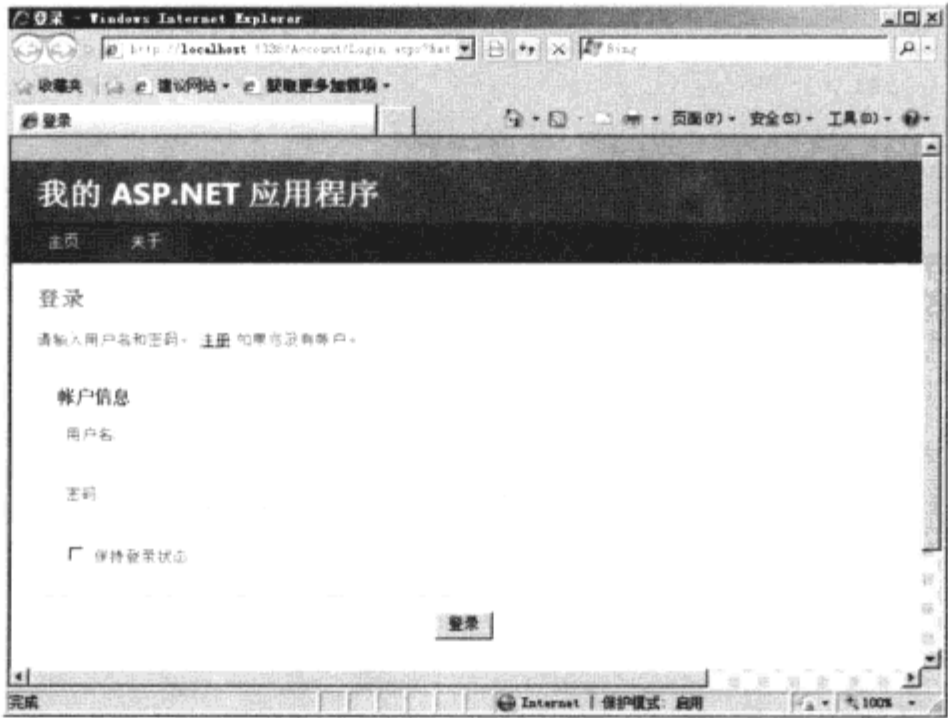


拒绝匿名用户的访问会使 web.config 发生如下变化(注意 authorization 元素):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
    <!--more config info here -->
  </system.web>
</configuration>
```

10. 运行网站。ASP.NET 应当会拒绝我们访问网站，并将我们定向至登录页面，如下图所示。



ASP.NET 会查找验证用户身份的方式。由于这是匿名用户发起的新会话，并且网站采用的是“Forms 身份验证”方法，因此网站会拒绝用户的访问。在默认情况下，Visual Studio

所生成的项目会使用 Account 文件夹下一个名为 login.aspx 的预先建好的页面。

下面我们将通过 Visual Studio 创建的这个登录页面来认识一下 ASP.NET 的登录控件。

9.4 ASP.NET 登录控件

在本章的前面，我们手动制作了几个不同的登录页面。那是 ASP.NET 1.1 时代实现“Forms 身份验证”的方式。ASP.NET 的后续版本添加了许多登录控件来满足网站对登录功能的一般需求。

这些登录控件包括：Login、LoginView、PasswordRecovery、LoginStatus、LoginName、ChangePassword 和 CreateUserWizard 控件。下面简单介绍一下这些控件：

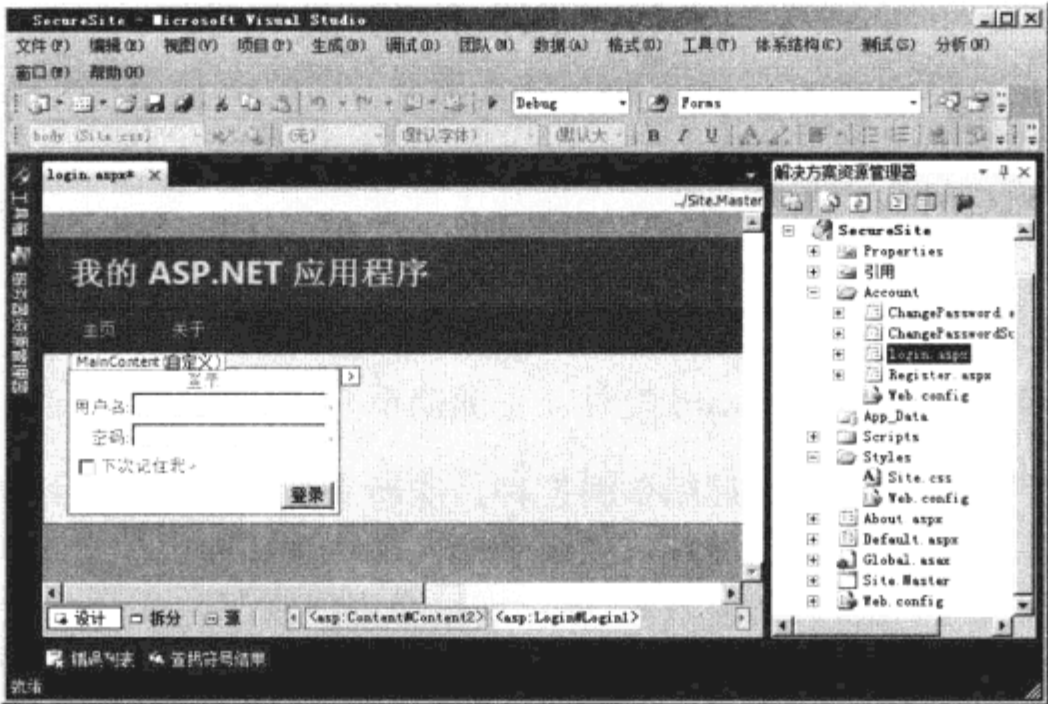
- **Login** 该控件是登录控件中最简单的控件，能够满足绝大多数网站对登录的需求——通过用户名和密码登录。该控件包括用于输入用户名和密码的文本框，以及用于选择是否在计算机上保存密码的复选框(保存密码会降低密码的安全性)。该控件暴露了用于修改文本和外观的属性。开发者可以添加提示注册和恢复密码的链接。在默认情况下，该控件会调用 ASP.NET 成员资格提供程序组件。开发者也可以通过处理 Authenticate 事件来自行管理身份验证。
- **LoginView** 该控件非常类似之前介绍的可选登录页面。如果希望对通过身份验证和未通过身份验证的用户显示不同的内容，则可以使用该控件。该控件具有 AnonymousTemplate 和 LoggedInTemplate 显示模板，可以在其中添加文本和链接。该控件会根据用户的登录状态选择要显示的模板。
- **PasswordRecovery** 该控件能够帮助用户在忘记密码的情况下将密码发送给他/她。该控件会提示用户输入用户的帐号名，并询问安全问题(以便核对用户身份)。该控件会将用户当前的密码或新生成的密码通过电子邮件发送给用户。
- **LoginStatus** 该控件能够显示当前用户是否已登录。对未登录的用户会提示登录，而对已登录的用户会提示注销。
- **LoginName** 该控件能够显示用户的登录名。
- **ChangePassword** 该控件允许用户修改密码。为此，通过身份验证的用户需要提供原密码、新的密码，并对新密码进行确认。
- **CreateUserWizard** 该控件用于收集创建 ASP.NET 成员资格帐户所需要的信息。在默认情况下，该控件会收集用户名、密码、电子邮件地址、安全问题和安全答案。针对应用程序当前采用的成员资格提供程序，该控件能够收集不同的信息。

下面我们来看看如何利用登录控件构建登录页面。

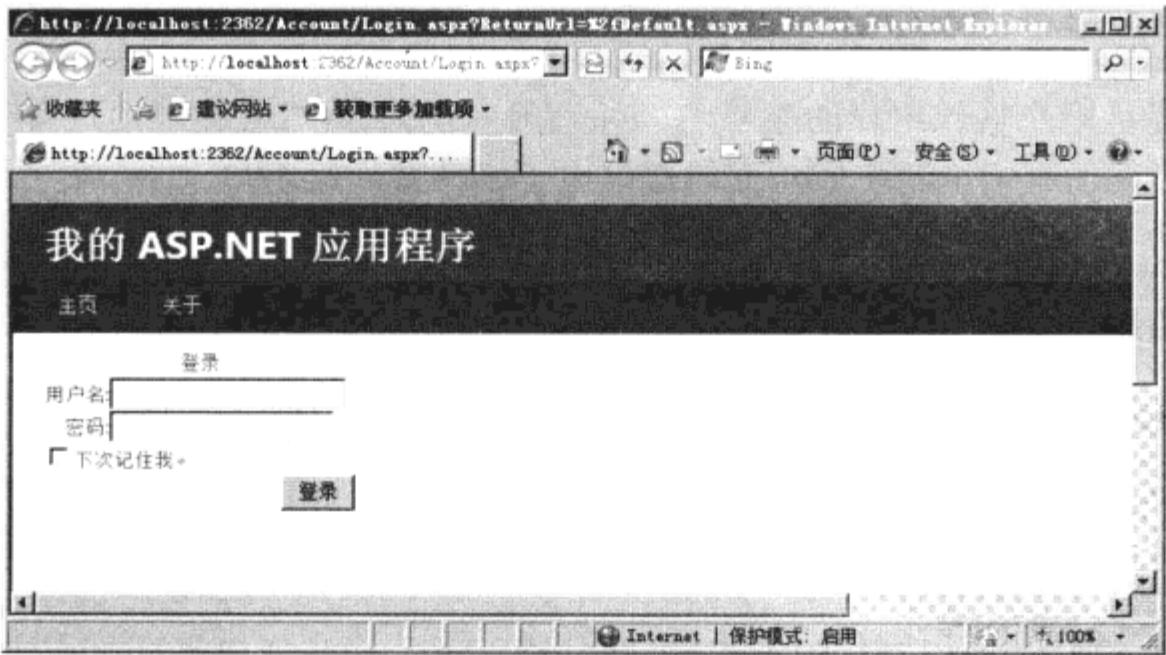
➤ 使用登录控件构建登录页面

1. Visual Studio 会默认创建一个登录页面。因此为了解登录控件的使用，最好自己创建

一个登录页面。移除 Accounts 文件夹下的 Login.aspx 页面，并在该文件夹右击，选择“添加”|“新建项”，选择“使用母版页的 Web 窗体”。将其命名为 login.aspx。在“选择母版页”窗口中选择“Site.Master”，单击“确定”。^①拖放一个 Login 控件到内容区域(见下图)。



- 2. Visual Studio 在生成网站时会默认采用“Forms 身份验证”。(为启用这种身份验证方式，可以在“网站管理工具”中选择“通过 Internet”。)在 web.config 中，登录 URL 被指定为~/Account/Login.aspx(已指向新建的登录页面)。
- 3. 浏览默认页面。此时，ASP.NET 会阻止用户并提示登录。在下图所示的窗口中，尝试输入在上一练习中设置的用户名和密码。



如果登录成功，则会看到下图所示的这个默认页面。

身份验证是实现网站安全的重要步骤。而对已通过身份验证的用户进行访问管理也同等重

^① 译者注：这里补充原书缺少步骤。

要，这种机制一般被称为“授权”。

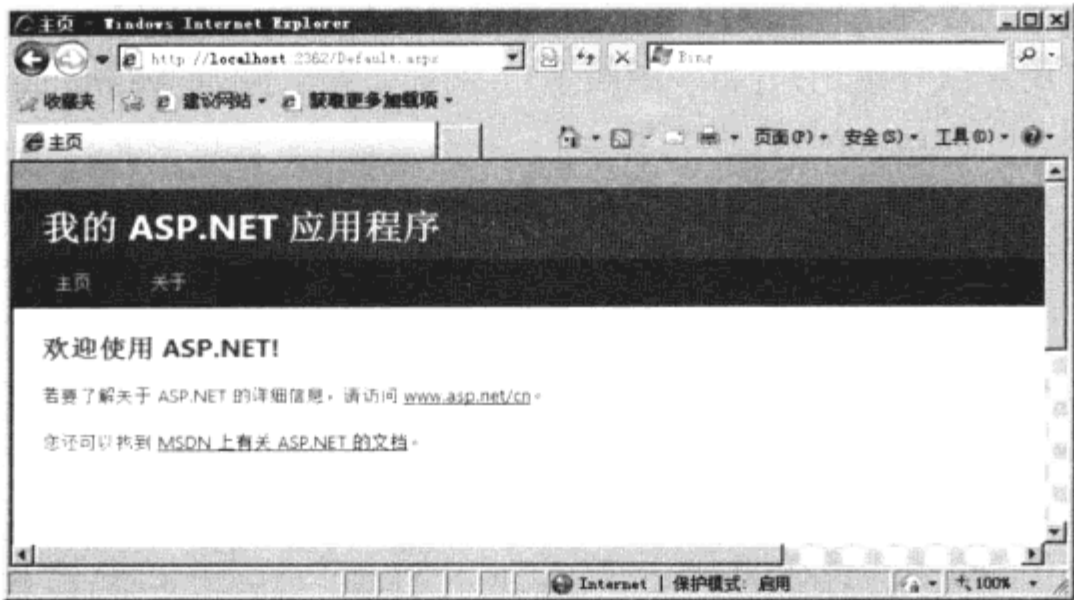


表 9.2 给出了本章中的练习所涉及的 3 组用户名和密码。

表 9.2 本章中的练习所涉及的用户名和密码

用 户 名	密 码
George	abc!123
Joe	abc!123
Frodo	abc!123

9.5 用户的授权

在对用户进行身份验证后，便确认了他/她的身份。虽然这个信息本身也有价值，但将身份验证和授权结合使用，系统会更安全。身份验证旨在确定身份，而授权旨在确定用户在登录后能够做什么。

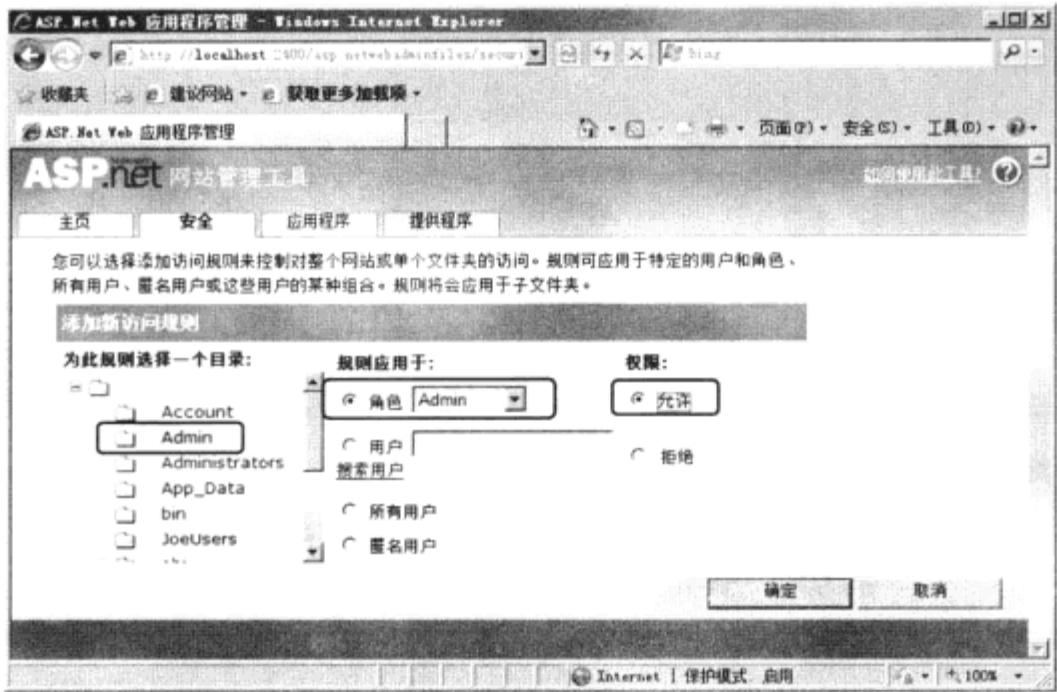
在之前的练习中，我们为网站添加了几个用户角色。下面我们将看看如何根据用户的身份来限制其对特定区域的访问。

➤ 实践授权

1. 添加一个只允许管理员访问的文件夹。在网站节点上右击，依次选择“添加”|“新建文件夹”。将这个文件夹命名为 Admin。在该文件夹中添加一个页面，并创建一个文本为“Administrators Only”的标签。类似地，分别创建名为 JoeUsers 和 PowerUsers 的文件夹。为每个目录添加一个页面 (Administrators.aspx、JoeUsers.aspx 和 PowerUsers.aspx)，以便稍后进行浏览。为区分这些页面，分别插入内容不同的文本标签。
2. 现在我们确定了角色和新资源之间的关联。在“网站管理工具”中，再添加一些用户，并为每个用户分配一些角色。例如，配套资源中的示例包含拥有 Administrator 角色的用户 George，拥有 JoeUser 角色的用户 Joe，以及拥有 PowerUser 角色的用户 Frodo。

ASP.NET 4 从入门到精通

- 3. 在添加新角色后，可以设置新的访问规则。为此，可以在“网站管理工具”的“安全”选项卡中选择“管理访问规则”链接，单击“添加新访问规则”。我们可以选择允许或拒绝单个用户或一组用户(如下图所示)。例如，被分配有 Administrators 权限的用户是可以访问 Administrators 文件夹的。



在完成上述操作后，我们应该能够在每个文件夹下的 web.config 文件的 authentication 和 authorization 节中看到相应的变化。在应用访问规则后，“网站管理工具”会为相应的目录添加 web.config 文件。对于示例中的 Admin 目录，该工具会在 allow 节点下面添加<deny users="*">。

- 4. 为使用户能够尝试浏览受限制的页面，在母版页中使用“菜单项编辑器”添加 3 个菜单项——依次指向第 1 步中添加的 Administrators.aspx、JoeUsers.aspx 和 PowerUsers.aspx 页面。为每个菜单项的 Text 属性设置一些有意义的文本(如对于指向 Administrators.aspx 页面的链接，其文本可以是“Go to Administrator Page”)。设置菜单项的 NavigationUrl 属性，使其链接指向对应的页面。
- 5. 运行网站。在登录后，我们应能够看到默认页面(见下图)和新建的几个菜单项。根据用户身份的不同，ASP.NET 可能允许或拒绝其访问刚刚创建的子目录。单击不同菜单项，看看 ASP.NET 能否根据角色允许或拒绝页面的访问。



如果成功以 JoeUsers 角色的用户登录，ASP.NET 将允许查看 JoeUsers 子目录的页面，如下图所示。

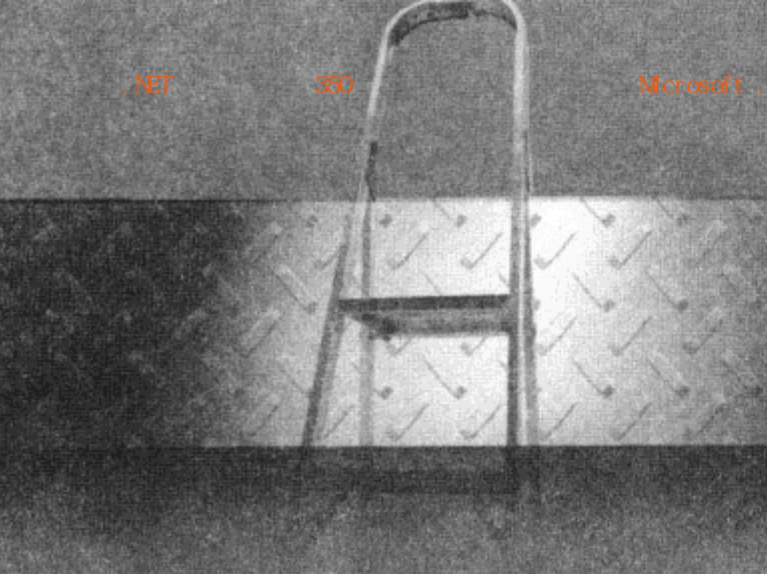


这个练习只涵盖了登录控件的一小部分功能。管理网站除了对用户进行筛选还有很多内容。完善的登录还涉及密码恢复和可选登录等功能，而这要用到其他登录控件。

9.6 快速参考

目 标	操 作
在应用程序中使用“Forms 身份验证”	可以采用以下任意一种方法。 (1) 使用“网站管理工具”(在 Visual Studio 的菜单中单击“网站”/“项目” “ASP.NET 配置”) (2) 使用 IIS 管理器中的 ASP.NET 配置功能。在这种情况下，身份验证类型必须为“Forms 身份验证”
配置网站的安全设置	可以采用以下任意一种方法。 (1) 使用“网站管理工具”(在 Visual Studio 的菜单中单击“网站”/“项目” “ASP.NET 配置”) (2) 使用 IIS 管理器中的 ASP.NET 配置功能。在这里，可以管理用户和角色，并为用户分配角色
以编程方式对请求进行身份验证	通过 FormsAuthentication 类来创建身份验证 Cookie
使用用户的身份验证 Cookie 失效	调用 FormsAuthentication 类的 SignOut 方法
确认身份验证 Cookie 是否存在	启用跟踪

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569



第 10 章

数据绑定

学习目标

- 使用数据绑定控件来显示集合的内容
- 使用 ASP.NET 中的数据库提供程序
- 配置数据绑定控件

本章将介绍 ASP.NET 中最主要的功能之一——数据绑定。ASP.NET 中有许多控件能够解析集合的结构与内容，并通过正确的标签来将其呈现为界面元素(如列表框、单选按钮列表和组合框)。本章将介绍这些控件的工作方式及其在网页中的使用方法。

10.1 在不使用数据绑定的情况下显示集合的内容

通过用户界面(UI)元素来显示集合，是在构建许多软件(尤其是网站)时需要处理的最常见的问题之一。不妨回忆一下自己最近浏览的网站。例如，如果通过某个商业网站来下订单，它很有可能会提示输入邮寄地址。还记得其中的省份是如何填写的？大多数网站会显示一个下拉列表框，可以从中选择省份名称。

这个下拉列表是如何被填充的呢？在 HTML 中，<select>标签可以嵌套多个<option>标签，所有要被列出的元素通过<option>标签呈现。省份名称可能来自数据库或其他已建立的数据源。在某处(最有可能是服务器上)有代码来对省份名称集合的数据进行遍历来呈现<select>和<option>标签。

一些 ASP.NET 服务器端控件(如 ListBox 和 DropDownList)具有 Items 属性(集合)。例如，以下拉列表的形式呈现集合的方法之一，是在 ASP.NET 页面上添加一个下拉列表框控件，然后像下面这样通过 Items.Add 方法来逐一添加每个元素(当然，这里假定了对对象的 ToString 能够返回一些有价值的内容——不是对象的类型名，而是对象的内容)^①：

```
protected void BuildDropDownList(IList techList)
{
    for (int i = 0; i < techList.Count; i++)
    {
```

① 译者注：在下拉列表控件显示每个元素的内容时，该控件会自动调用其 ToString 方法。ToString 是 .NET 中最常用的方法之一，有时被显式地调用，有时被隐式地调用。通过重写该方法来体现对象本身的内容，是一个良好的 .NET 编程习惯(有时甚至是一种规范)。


```
        this.DropDownList2.Items.Add(techList[i]);  
    }  
}
```

由于通过 UI 元素显示集合内容是一项十分常见的编程任务，因而最好让框架具备这种功能。ASP.NET 包含许多数据绑定控件，能够接受集合并呈现正确的标签。下面几节将介绍这种机制。

10.2 通过数据绑定来显示集合

ASP.NET 中的每种数据绑定控件都包含数据源相关属性。对于支持简单数据绑定方式的控件，可以将控件的 `DataSource` 属性设置为任何实现了 `IEnumerable`、`ICollection` 和 `ICollection` 接口的集合(其中包括稍后将要介绍的 `DataSet` 和 `DataTable` 类)。在将集合附加到控件上之后，便可以调用页面(或控件)的 `DataBind` 方法来使控件对该集合进行遍历。

有些控件支持复杂一些的数据绑定方式，它们使用名为 `DataSourceID` 的属性。这是一种被称为“声明式数据绑定”(declarative data binding)的新型数据绑定方式。声明式数据绑定不是简单地对集合进行遍历，而是通过单独的 `DataSource`(数据源)控件为数据绑定控件管理数据。我们可以将 `DataSource` 控件看作预先配置的数据库命令。使用 `DataSource` 控件时，并不需要在代码中对数据库执行命令和查询，它会代为提交。这种数据管理控件能够为数据绑定控件提供标准功能的支持(如排序、分页和编辑)。声明式数据绑定在很大程度上简化了集合的呈现。在页面上，数据绑定控件只需要引用 `DataSource` 控件的 ID。.NET Framework 中包含几种 `DataSource` 控件，分别针对 Microsoft Access 数据库、Microsoft SQL Server 数据库、临时集合(`ObjectDataSource`)、“语言集成查询”(LINQ `DataSource`)，以及 XML 数据(`XmlDataSource`)。第 11 章还会介绍另外一种 `DataSource` 控件——`SiteMapDataSource`。使用声明式数据绑定时不必特意调用 `DataBind` 方法。控件会自动在页面的 `PreRendering` 事件中调用该方法。

ASP.NET 中包含许多支持简单数据绑定的控件，此外还有一部分支持声明式数据绑定。这些声明式数据绑定控件包括基于 `ListControl` 的控件——`CheckBoxList`、`RadioButtonList`、`DropDownList` 和 `ListBox`。此外，还有一些更高级的控件——`TreeView`、`Menu`、`GridView`、`DataGrid`、`Repeater`、`FormView` 和 `DetailsView`。

下面简单介绍一下每种控件。

10.2.1 基于 ListControl 的控件

最常见的数据绑定控件中有许多派生于 `ListControl` 类。这些控件包括 `ListBox`、`BulletedList`、`RadioButtonList`、`CheckBoxList` 和 `DropDownList`。稍后会具体介绍这些控件。这些控件大多数都可以“顾名思义”。这些控件都可以在 Windows 桌面编程和标准的 HTML 控件标签中找到对应的元素。例如，ASP.NET 中的 `DropDownList` 与 Windows Forms 应用程序的

ComboBox 类似。RadioButtonList 能够显示互斥的单选按钮。CheckBoxList 能够显示一系列复选框。

10.2.2 TreeView 控件

第 6 章的示例演示过 TreeView 控件。该控件能够表现层次型数据，能够与 XML 数据源完美结合。TreeView 支持可折叠的节点，用户可以通过这些节点在逻辑上较泛化的元素与较具体的元素之间穿梭。TreeView 是支持声明式数据绑定的控件之一。

10.2.3 Menu 控件

Menu 控件也能够处理层次型数据的绑定。使用 Menu 控件，用户可以像在桌面应用程序中使用菜单那样来对网站进行导航。Menu 是支持声明式数据绑定的控件之一。

10.2.4 FormView 控件

FormView 控件支持任意形式的布局，通过支持数据绑定的控件(TextBox 或 ListBox)来呈现每个数据项。FormView 还支持通过控件来编辑数据源中的数据。FormView 是支持声明式数据绑定的控件之一。

10.2.5 GridView 控件

ASP.NET 1.x 只支持 DataGrid 控件，而后续的 ASP.NET 则提供了该控件的升级版——GridView(网格视图)。顾名思义，GridView 控件能够在行和列构成的网格中呈现集合的内容。网格中的每一行代表集合中的一个记录，每一行中的每一列代表记录的一个字段。此外，DataGrid 要求开发者自行实现数据分页与排序，而 GridView 控件本身就支持这两种功能。GridView 还具有编辑功能(而 DataGrid 控件则需要单独对其编写代码)。GridView 是支持声明式数据绑定的控件之一。

10.2.6 DetailsView 控件

GridView 控件用于展示整个数据源的概要，而 DetailsView 控件则每次只关注其中的一条记录。DetailsView 控件经常与 ListBox、DropDownList 和 GridView 这样的控件配合使用。用户在这些控件中选择一条记录，再通过 DetailsView 查看具体的信息。DetailsView 是支持声明式数据绑定的控件之一。

10.2.7 DataList 控件

DataGrid 和 GridView 控件以规则的行和列的形式来显示数据，这是一种显示方式。而如果要更细致地控制最终呈现的格式，则可以使用 DataList 控件。DataList 控件能够通过定制的模板来控制数据源中记录的显示方式。

10.2.8 Repeater 控件

Repeater 控件也能够以指定的格式显示数据源中的数据(但不一定采用表格形式)。Repeater 控件会通过 HTML 和服务器端控件来显示每行数据，显示每一行时都会使用预先定义的格式(模板)。


10.3 简单数据绑定

最简单的数据绑定只需要将集合附加到控件(如基于 ListControl 的控件)的 DataSource 属性上。如果集合已获得，那么只需将其赋给控件的 DataSource 属性，控件就会自动选择正确的标签来呈现。


下面的练习会将同一个 List 与几种 ListControl 控件关联，以此来体验几种数据绑定控件。

➤ 使用集合的数据绑定

1. 新建一个名为 DataBindORama 的空网站项目。
2. 为项目添加 App_Code 目录。在“解决方案资源管理器”中的项目节点上右击，选择“添加”|“添加 ASP.NET 文件夹”|“App_Code”。右击它，然后从弹出的菜单中选择“添加新项”，然后在其中添加一个名为 TechnologyDescriptor 的类。为该类添加两个类型为字符串，名称分别为 TechnologyName 和 Description 的自动实现的属性。该类将用于表示技术名称和对应的说明。

 **技巧** 在 .NET 3.5 之前的版本中，可能需要创建私有或受保护的字段来存储基于字符串的信息，然后通过公共属性将字符串暴露出来以供外界使用。.NET 3.5 和后续版本支持自动实现的属性(Auto-Implemented Property)。自动实现的属性只不过是一种实现属性的简便方法，适用于只把私有(或受保护)字段暴露出来的情况。^①

① 译者注：有的资料将这种属性成为“隐式属性”(implicit property)或“自动属性”(automatic property)，而将常规的属性称为“显式属性”(explicit property)。为了便于理解，这里采用的是 Microsoft 官方承认的译法——“自动实现的属性”。

 **注意** 为使数据绑定能够正确工作，需要将成员变量以属性的形式暴露出来。当控件与某个由类组成的集合绑定后，控件会根据集合元素的属性名称来获取内部字段的内容。在使用数据绑定控件时，可以指定一个显示名称（相应的值会显示在控件中）和一个隐藏的值，两者与控件的项目关联在一起。在呈现托管对象的集合时，数据绑定的方式与通过属性暴露的字段有关。

清单 10.1 给出了 TechnologyDescriptor 类的代码，它通过属性暴露了“技术名称”和“相关说明”。这个类还有一个用于创建 TechnologyDescriptor 集合的静态方法。

清单 10.1 TechnologyDescriptor 类的代码

```
public class TechnologyDescriptor
{
    public string TechnologyName { get; set; }
    public string Description { get; set; }

    public TechnologyDescriptor(string strTechnologyName,
                                string strDescription)
    {
        this.TechnologyName = strTechnologyName;
        this.Description = strDescription;
    }

    public static List<TechnologyDescriptor> CreateTechnologyList()
    {
        List<TechnologyDescriptor> lTechnologies =
            new List<TechnologyDescriptor>();

        TechnologyDescriptor technologyDescriptor;

        technologyDescriptor =
            new TechnologyDescriptor("ASP.NET",
                                      "Handle HTTP Requests");
        lTechnologies.Add(technologyDescriptor);

        technologyDescriptor =
            new TechnologyDescriptor("Windows Forms",
                                      "Local Client UI technology");
        lTechnologies.Add(technologyDescriptor);

        technologyDescriptor =
            new TechnologyDescriptor("ADO.NET and Linq",
                                      "Talk to the database");
        lTechnologies.Add(technologyDescriptor);

        technologyDescriptor =
            new TechnologyDescriptor(".NET CLR",
                                      "Modern runtime environment for manage code");
        lTechnologies.Add(technologyDescriptor);
    }
}
```



```

        technologyDescriptor =
            new TechnologyDescriptor(".NET IL",
                "Intermediary representation for .NET applications");
        lTechnologies.Add(technologyDescriptor);

        technologyDescriptor =
            new TechnologyDescriptor("WPF",
                "Advanced rendering technology");
        lTechnologies.Add(technologyDescriptor);

        technologyDescriptor =
            new TechnologyDescriptor("Silverlight",
                "Advanced rendering on the Web");
        lTechnologies.Add(technologyDescriptor);

        technologyDescriptor =
            new TechnologyDescriptor(".NET Compact Framework",
                "Modern runtime environment for small devices");
        lTechnologies.Add(technologyDescriptor);

        return lTechnologies;
    }
}

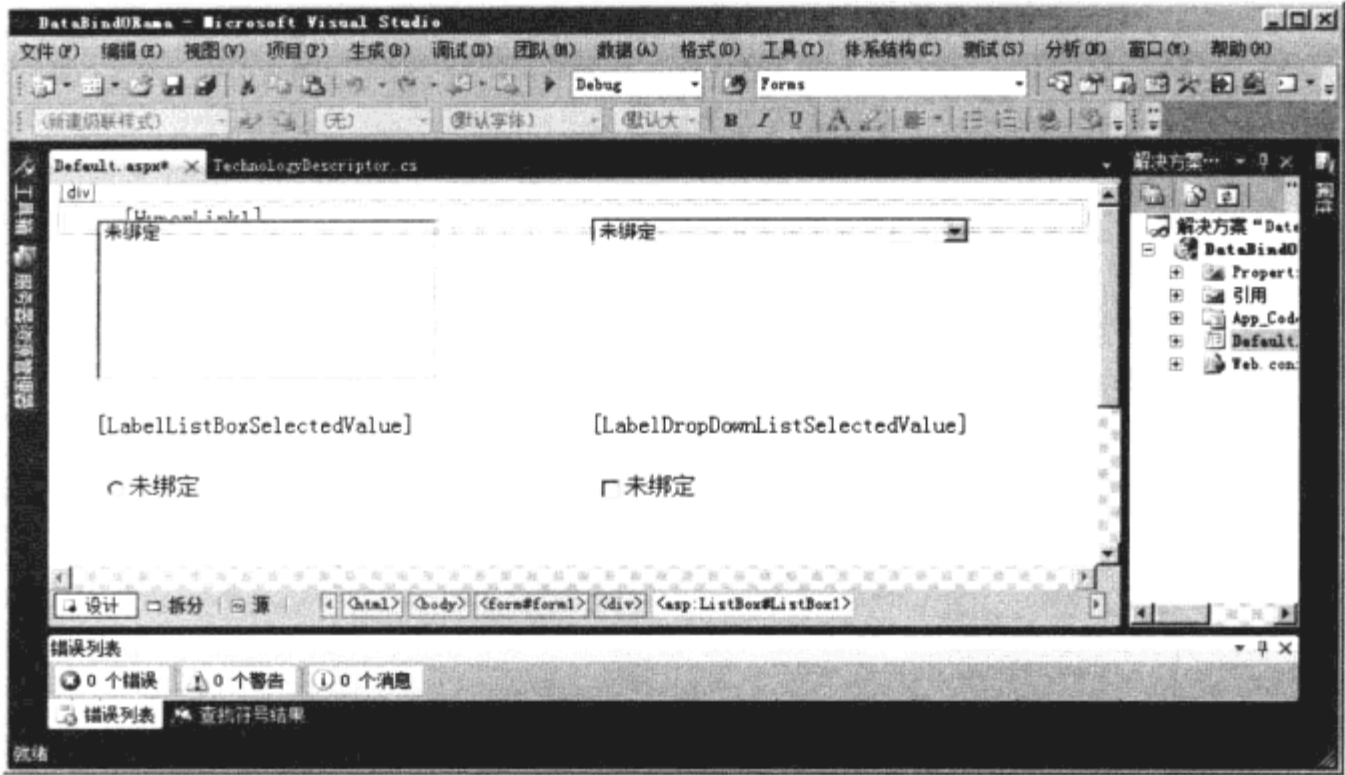
```

3. 在项目中添加一个“Web 窗体”，将其命名为 Default.aspx。在这个默认页面的内容区域添加 4 个数据绑定控件：ListBox、DropDownList、RadioButtonList 和 CheckBoxList。本示例采用了绝对定位方式来对控件进行布局。
4. 在每个控件的下面添加一个 Label。稍后将通过这些标签控件来显示与选定项关联的值。为这些标签依次进行命名，以便能够在代码中使用它：LabelListBoxSelectedValue、LabelDropDownListSelectedValue、LabelRadioButtonListSelectedValue 和 LabelCheckBoxListSelectedValue。
5. 将 ListBox、DropDownList 和 CheckBoxList 控件的 AutoPostBack 属性设置为 true(这是 RadioButtonList 的默认设置)。这样，在这些控件中进行选择时，都会触发页面回发，而选定项会在回发期间被检测到。
6. 对页面进行更新，使其初始化 TechnologyDescriptor 集合，并将这个集合附加到每个控件上。将每个控件的 DataTextField 属性设置为 TechnologyName(以便将控件映射到 TechnologyDescriptor 的 TechnologyName 属性)。这样能够确保控件显示的是技术名称。然后将每个控件的 DataValueField 设置为 Description，以便将 Description 属性映射为与技术名关联的值。

清单 10.2 给出了创建 TechnologyDescriptor 集合并将该集合附加到每个控件上的代码。

7. 双击每个数据绑定控件来添加选择事件的处理程序。在处理程序中访问控件中选定的值。清单 10.2 也列出了这些处理程序。除标签控件外，这个代码清单中其他控件的名

称均为 Microsoft Visual Studio 自动生成的。



清单 10.2 为支持数据绑定和控件事件而对 Default.aspx.cs 的修改

```
using System.Collections.Generic
protected void Page_Load(object sender, EventArgs e)
{
    if (!this.IsPostBack)
    {
        List<TechnologyDescriptor> techList =
            TechnologyDescriptor.CreateTechnologyList();
        this.ListBox1.DataSource = techList;
        this.ListBox1.DataTextField = "TechnologyName";

        this.DropDownList1.DataSource = techList;
        this.DropDownList1.DataTextField = "TechnologyName";

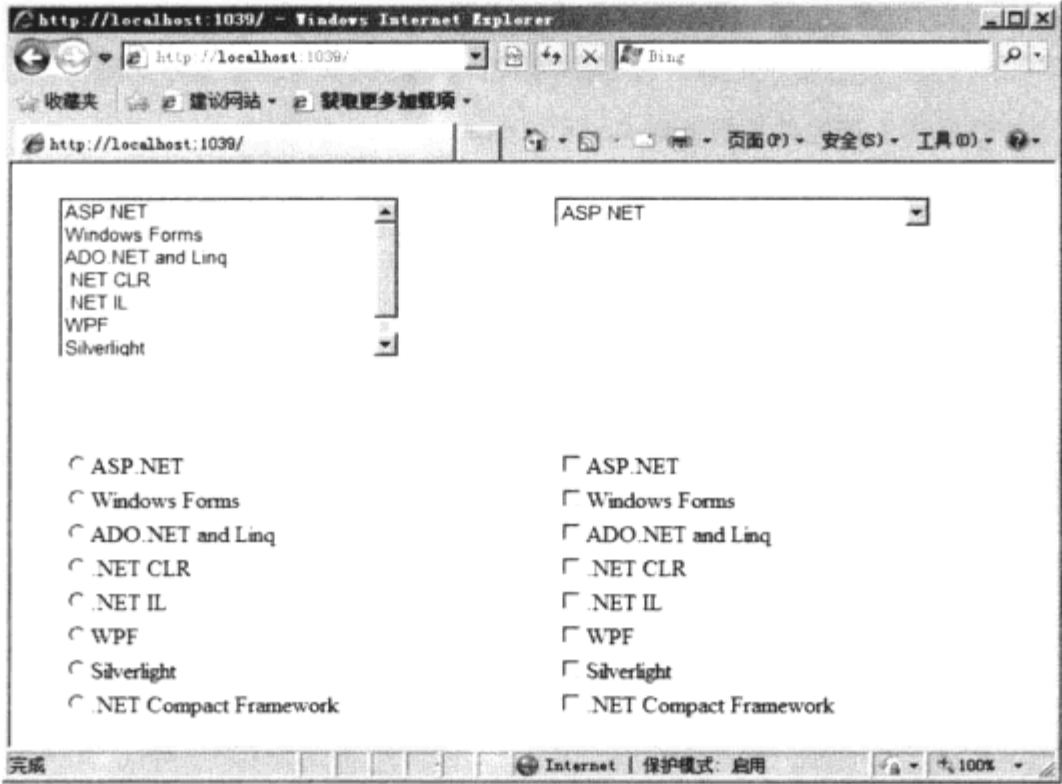
        this.RadioButtonList1.DataSource = techList;
        this.RadioButtonList1.DataTextField = "TechnologyName";

        this.CheckBoxList1.DataSource = techList;
        this.CheckBoxList1.DataTextField = "TechnologyName";

        this.DataBind();
    }
}
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.LabelListBoxSelectedValue.Text = this.ListBox1.SelectedValue;
}
protected void DropDownList1_SelectedIndexChanged(object sender,
    EventArgs e)
{
    this.LabelDropDownListSelectedValue.Text =
```

```
        this.DropDownList1.SelectedValue;
    }
    protected void RadioButtonList1_SelectedIndexChanged(object sender,
        EventArgs e)
    {
        this.LabelRadioButtonListSelectedValue.Text =
            this.RadioButtonList1.SelectedValue;
    }
    protected void CheckBoxList1_SelectedIndexChanged(object sender,
        EventArgs e)
    {
        this.LabelCheckboxListSelectedValue.Text =
            this.CheckBoxList1.SelectedValue;
    }
}
```

8. 编译网站，并浏览该页面。



在本示例中，选择数据绑定控件中的某一项时，控件下方的标签便会显示相应的值。

在某些情况下，我们可能会遇到这种数据绑定方式。例如，显示我国的省份，或显示雇员或联系人姓名这种简短的列表，就非常适合采用基于 ListControl 的控件。然而，有时需要处理格式更为复杂的数据(而不仅仅是标准的集合)。有许多控件能够处理更为复杂的 DataSet。不过，为此我们需要先了解一下 ADO.NET，因为这是掌握复杂数据结构最直接的方式。

10.4 数据库的访问

上面的示例将内存中的集合(如 ArrayList 和 List)附加到服务器端的控件上，这些控件会为客户端呈现适当的标签。虽然这种方式十分常见，但服务器端控件还支持其他集合——包括来自数据库中的集合。在学习如何使用 UI 元素呈现数据库查询之前，我们先来了解一

下.NET对数据库的支持。本章会介绍ASP.NET与SQL Server配合使用时的一些基础知识。第22章会演示用于创建对象模型的新的实体框架。本章后面还将介绍与数据库访问有关的语言集成查询(Language Integrated Query, LINQ)技术。

10.5 .NET对数据库的支持

除了包含管理富客户端UI(Windows Forms)和处理HTTP请求(ASP.NET)的类库外，.NET还包含一个支持多种数据库的类库——ADO.NET。

ADO.NET与Microsoft之前名为ADO的数据库技术类似。ADO代表“活动数据对象”(Active Data Objects)。Microsoft已抛弃了“活动”的概念，而将ADO这个名称后面追加了“.NET”，以强调托管的数据库技术(这同样也是产品名称)。ADO.NET包含一组功能与传统ADO功能类似的托管提供程序。ADO.NET主要关注3个方面：数据库连接、数据库操纵和结果的使用。

10.5.1 连接

如果要与某个特定的数据库进行交互，首先要进行连接。在大多数情况下，连接数据库至少需要提供数据库的位置。很多情形下还需要设置连接的安全性(通过用户名和密码)。而有时可能还涉及连接池和事务这种高级话题。这些设置都会在连接数据库时处理。连接信息通常包含在一个字符串中，在ADO.NET内部解析这个字符串时，会通过字符串的内容来设置各种连接参数。

ADO.NET中包含几个用于连接数据库的类。ADO.NET 1.x中只有两个，一个针对Microsoft SQL Server，另一个针对支持OLEDB的数据库。ADO.NET的后续版本添加了几种对特定数据库的支持，并引入了一套采用“提供程序模式”(provider pattern)^①的数据库服务。

ADO.NET 1.x的数据访问代码大多使用的是ADO的接口(而不是直接初始化数据库类)。这样，开发者可以在代码中的某处(管理连接的部分)隔离出厂商特定的技术细节。然后，执行查询所需的对象(如命令对象)可以通过连接获得。虽然现在仍可以采用ADO.NET 1.x风格的代码来连接数据库，但已经有了一种更好的方式——ADO.NET数据库提供程序工厂。

ADO.NET提供程序模式对连接和使用数据库做出了重大改进。这种模式可以通过调用一次“提供程序工厂”来避免暴露不同数据库的差异。在某一处选定数据库的类型后，提供程

① 译者注：这里所谓的“提供程序模式”实际上指的是“设计模式”中的“策略模式”(下文还涉及另一种设计模式——“工厂方法”)。它通过派生在接口不变的基础上使不同算法之间可互换。不同类型的数据库可以看作不同“算法”的焦点。例如，在连接SQL Server和Access数据库时会采用不同的交互方式(一种基于网络，另一种基于文件系统)，而连接的调用者并不会察觉两者的不同。

序会帮助开发者选择类型正确的连接和命令。这在 ADO.NET 1.x 中并不重要,因为它只将数据库分为两类: SQL Server 和 OLEDB 数据库。然而,随着对新的数据库类型的支持,提供程序模式的作用便凸显出来。

在 machine.config 文件中,我们可以找到针对以下几种数据库的提供程序:

- ODBC 数据提供程序
- OLE DB 数据提供程序
- OracleClient 数据提供程序
- SqlClient 数据提供程序
- SQL Server CE 数据提供程序

前 4 种数据库提供程序工厂的设置由 DbProviderConfigurationHandler 处理,而针对 SQL Server CE 数据库的提供程序工厂的设置则由单独组件处理。清单 10.3 给出了 machine.config 中将提供程序名称与提供程序工厂建立映射的部分。

清单 10.3 machine.config 中定义的默认提供程序工厂

```
<configuration>
  <configSections>
    <section name="system.data.odbc"
      type="System.Data.Common.DbProviderConfigurationHandler, ..." />
    <section name="system.data.oledb"
      type="System.Data.Common.DbProviderConfigurationHandler, ..." />
    <section name="system.data.oracleclient"
      type="System.Data.Common.DbProviderConfigurationHandler, ..." />
    <section name="system.data.sqlclient"
      type="System.Data.Common.DbProviderConfigurationHandler, ..." />
  </configSections>
  <system.data>
    <DbProviderFactories>

      <add name="Microsoft SQL Server Compact Data Provider"
        invariant="System.Data.SqlServerCe.3.5"
        type="System.Data.SqlServerCe.SqlCeProviderFactory ..." />

    </DbProviderFactories>
  </system.data>
</configuration>
```

为连接数据库,需要获得相应的工厂对象,然后通过工厂创建连接(如清单 10.4 所示)。为获得工厂对象,可以使用数据库类型的名称(如 System.Data.SqlClient 或 System.Data.SqlServerCe.3.5),然后可以通过工厂对象来创建连接。

清单 10.4 获取数据库提供程序工厂

```
DbConnection GetConnectionUsingFactory()
{
    DbProviderFactory dbProviderFactory =
```

```

        DbProviderFactories.GetFactory("System.Data.SqlClient")
    return dbProviderFactory.CreateConnection();
}

```

在获得连接后，便可以使用它来连接数据库。如果本地计算机上有一个名为 AspDotNetStepByStep 的 SQL Server 数据库，则可以在 web.config 文件中添加一个连接字符串(如清单 10.5 所示)。

清单 10.5 连接字符串设置的例子

```

<configuration>
  <connectionStrings>
    <add name="AspDotNetStepByStep"
      connectionString=
        "server=(local);integrated security=sspi;database=AspDotNetStepByStepDB"/>
  </connectionStrings>
</configuration>

```

在获得数据库连接的引用后，便可以打开该链接，然后通过它来运行命令。

10.5.2 命令

在建立连接后，数据库会等待程序发送数据库命令。这些命令一般用于查询数据、更新或删除现有数据，以及插入新数据。大多数数据库支持“结构化查询语言”(Structured Query Language, SQL)来管理命令(有些数据库只支持特定的 SQL 版本，所以实际的命令可能在不同的实现中有所差异)。对数据库执行的命令一般是下面这样的 SQL 语句：

```
SELECT * FROM DotNetReferences WHERE AuthorLastName = 'Smith'
```

清单 10.6 给出了一段示例代码。这段代码首先会连接名为 AspDotNetStepByStepDB 的 SQL Server 数据库，并在 DotNetReferences 表中查询姓为“Smith”的人。

清单 10.6 通过数据读取器查询数据库的例子

```

class UseDBApp
{
    static void Main()
    {
        DbProviderFactory dbProviderFactory =
            DbProviderFactories.GetFactory("System.Data.SqlClient");
        using (DbConnection conn = dbProviderFactory.CreateConnection())
        {
            string s =
                ConfigurationManager.ConnectionStrings["AspDotNetStepByStep"].ConnectionString;
            conn.ConnectionString = s;
            conn.Open();

            DbCommand cmd = conn.CreateCommand();
            cmd.CommandText =
                "SELECT * FROM DotNetReferences WHERE AuthorLastName='Smith'";

```



```
        DbDataReader reader = cmd.ExecuteReader();
        // do something with the reader
    }
}
```

`ExecuteReader` 方法会把查询发送给数据库，结果会通过实现了 `IDataReader` 接口的对象返回。上述代码不包含使用结果的代码，但我们后面几节中看到它的作用。

10.5.3 结果的管理

当连接到数据库并发起查询后，便可以使用结果了。ADO.NET 主要支持两种对象来管理结果集——`IDataReader` 接口和 `DataSet` 类。

10.5.3.1 IDataReader

上面的示例通过查询操作获得了一个 `IDataReader` 对象。我们可以通过 `IDataReader` 接口来对查询的结果进行遍历。清单 10.7 展示了 `IDataReader` 接口的常用成员。

清单 10.7 `IDataReader` 接口的常用成员

```
public interface IDataReader
{
    bool IsClosed { get; }
    int RecordsAffected { get; }
    void Close();
    bool NextResult();
    bool Read();
    //...
}
```

在对查询的结果进行遍历时，可以通过 `Read` 方法获取下一行，通过 `NextResult` 方法获取下一个结果集。

`IDataReader` 访问数据的这种方法一般被称为“流水模式” (fire hose mode)，因为数据是以一种只进的、一次一行的方式被访问的。除非重置读取器并重新开始，否则是无法后退的。此外，读取器返回的数据行是只读的。不论出于何种原因而获取数据，都不能通过 `IDataReader` 对象来修改数据库(插入、更新和删除)。除使用 `IDataReader` 接口外，还可以使用 `DataSet` 来访问数据。

10.5.3.2 DataSet

除 `IDataReader` 外，ADO.NET 还支持一种以无连接方式访问数据的记录集——`DataSet`(数据集)。ADO.NET 旨在帮助开发者编写大型的、高伸缩性的应用程序。妨碍伸缩性的主要因素之一是数据库连接性。数据库通常会限制同时活动的连接的数目。如果某一时刻所有

连接都处于使用状态，那么其他要连接数据库的应用程序就必须等待。如果系统中用户的数目与可用连接的数目相等，那么这可能不是问题。然而，如果系统中用户的数目远大于数据库可用的连接的数目，那么系统的性能则会受到严重影响。

为增强可伸缩性，ADO.NET 引入了 DataSet 类，旨在为应用程序提供一个便于导航的数据库快照。这背后的思想是连接数据库后尽快获得数据副本并断开连接。使用 DataSet，我们可以插入记录、更新字段，甚至删除记录，并在最后将这些更改提交给数据库。

DataSet 类通常使用 DataAdapter 填充。DataSet 中可以包含多个 DataTable(数据表)——每个 DataTable 对应查询中的一条选择语句。一旦 DataAdapter 为 DataSet 获得数据后，我们便在内存中得到了最新数据的快照。DataSet 包含一个 DataTable 集合，每个 DataTable 对应查询中的一条 SELECT 语句。这个集合通过 Tables 属性暴露出来，支持数字或字符串类型的索引。在获得数据表后，便可以对其中的行和列进行遍历。这时，可以使用数字索引来选择行，使用数字或字符串索引来选择列。清单 10.8 给出了一个使用 SqlDataAdapter 来填充 DataSet 的例子。

清单 10.8 通过 DataSet 和 DataAdapter 执行数据库查询的例子

```
class UseDBApp2
{
    static void Main()
    {
        DataSet ds = new DataSet();
        DbProviderFactory dbProviderFactory =
            DbProviderFactories.GetFactory("System.Data.SqlClient");
        using (DbConnection conn = dbProviderFactory.CreateConnection())
        {
            string s =
                ConfigurationManager.ConnectionStrings["AspDotNetStepByStep"].ConnectionString;
            conn.ConnectionString = s;
            conn.Open();

            DbCommand cmd = conn.CreateCommand();
            cmd.CommandText =
                "SELECT * FROM customer; SELECT * FROM country";

            DbDataAdapter adapter = dbProviderFactory.CreateDataAdapter();
            adapter.SelectCommand = cmd;
            adapter.Fill(ds);
        }

        foreach (DataTable t in ds.Tables)
        {
            Console.WriteLine("Table " + t.TableName + " is in dataset");
            Console.WriteLine("Row 0, column 1: " + t.Rows[0][1]);
            Console.WriteLine("Row 1, column 1: " + t.Rows[1][1]);
            Console.WriteLine("Row 2, column 1: " + t.Rows[2][1]);
        }
        ds.WriteXml("dataset.xml");
    }
}
```

```
ds.WriteXmlSchema("dataset.xsd");

// Also-may bind to the tables here:
;
} }
```

清单 10.8 中的代码演示了 `DataAdapter` 和 `DataSet` 的使用方法。这段代码会打印 `DataSet` 中每个表前 3 行的第二列(即索引为 1 的列)。清单 10.8 中的代码说明 `DataTable` 可以作为数据绑定控件的数据源。这段代码还说明了可以将 `DataSet` 对象以 XML 的形式序列化。架构与内容都能以相同的方式被序列化——从而可以通过 `DataSet` 在不同系统之间传送数据。虽然 `DataSet` 适合传送数据，但其体积相对庞大。使用 `DataSet` 将数据和架构传给另一个系统，会增加带宽消耗或传输时间。因此，采用这种方式应谨慎。

关于 `DataSet` 类中的项，还有一点需要说明：在数据访问的方式上，该对象是无连接的、非流水模式的。我们可以对 `DataSet` 中的任意表、列或行进行随机的访问。事实上，`DataSet` 类中的对象还能够跟踪对数据的修改。使用 `CommandBuilder` 和 `DataAdapter`，还可以将 `DataSet` 中的更改更新到数据库中。`CommandBuilder` 能够自动生成简单的 SQL 语句，从而简化了通过 `DataAdapter` 对数据库的更新。^①

在获得 `IDataReader` 或 `DataSet` 后，数据绑定控件能够自动将内容呈现给浏览器。虽然可以通过标准的连接/命令方式来手动连接数据库，获取数据及呈现数据，但 ASP.NET 和 Visual Studio 提供了另一种更为便捷的方式——声明式数据绑定。

10.6 ASP.NET 数据源

我们可以使用 ADO.NET 以传统的方式来访问数据，但有一种更便捷的方法。ASP.NET 提供了几种控件，能够掩盖管理连接和获取数据的复杂性。这就是 `DataSource` 控件。

`DataSource` 控件对整个连接和命令机制进行了抽象，开发者所要做的只是确定数据源，将该控件指向这个数据源，然后定义所需的查询。Visual Studio 提供了帮助开发者完成此过程的向导。在 `DataSource` 就绪后，可以将其附加到使用它的数据绑定控件上。

下面的练习演示了如何进行查询并把结果填充到控件中。

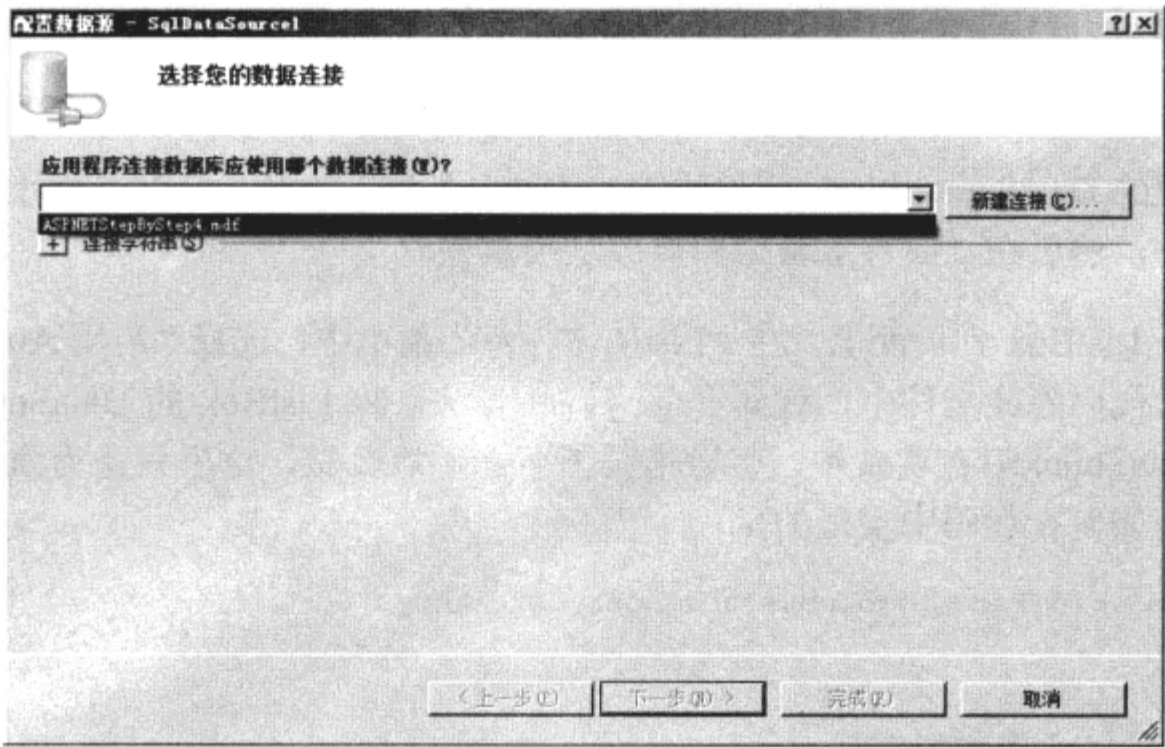
► 使用 `DataSource` 和 `DataReader` 来为控件填充数据

1. 为 `DataBindORama` 项目添加一个名为 `DataBindingWithDB` 的“Web 窗体”。
2. 配套资源上本章的示例(名为 `DataBindORama`)包含一个名为 `ASPNETStepByStep4.mdf`

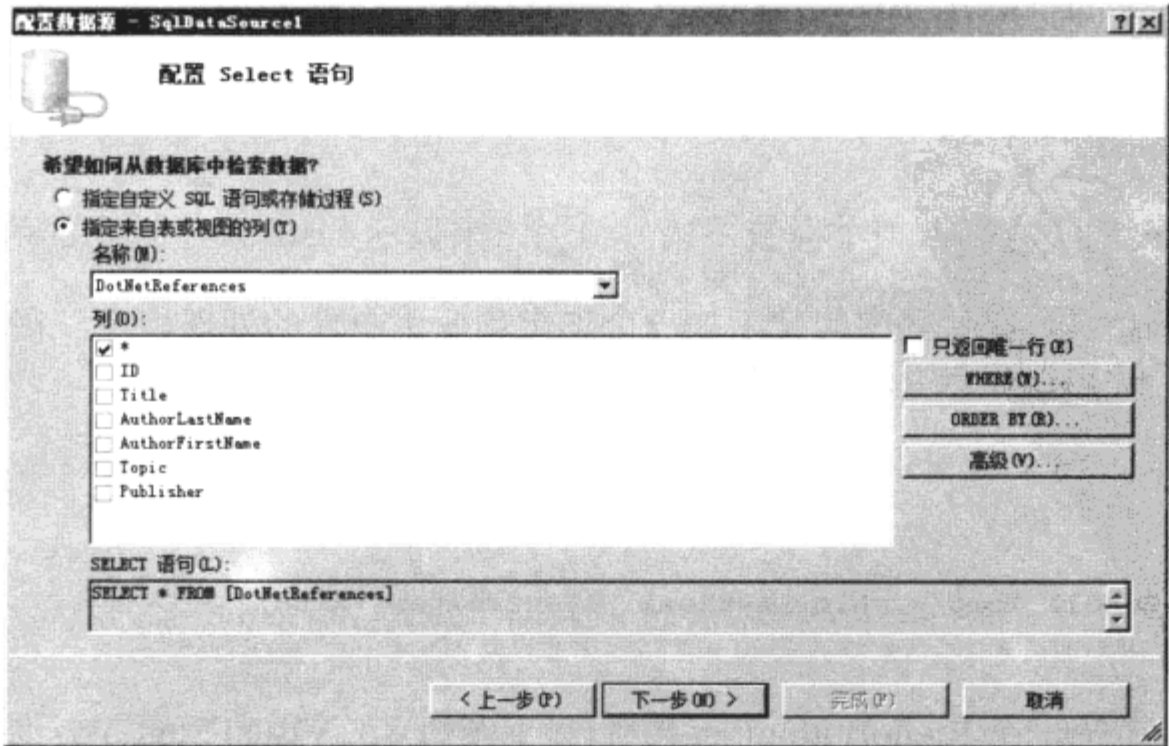
① 译者注：虽然这里将 `DataSet` 与数据库一起讨论，但实际上它的功能非常强大，能够完全独立于数据库工作。甚至可以将它看作一种存储于内存的微型数据库。

的数据库。先创建一个 App_Data 文件夹。在“解决方案资源管理器”中的项目节点上右击，选择“添加”|“添加 ASP.NET 文件夹”|“App_Data”。为将数据库添加到这个项目中，在项目中右键单击“App_Data”节点，然后选择“添加”|“现有项”，找到配套资源中的 ASPNETStepByStep4.mdf 数据库文件。下面我们添加并配置一个针对这个数据库的数据源控件。

在“工具箱”中找到“数据”选项卡。从该选项卡中拖出一个 SqlDataSource 到页面上。单击 Visual Studio 上下文菜单中的“配置数据源”。此时会显示“配置数据源”对话框，我们可以在其中选择数据库——刚刚添加的数据库会显示在组合框中(如下图所示)。为连接到 ASPNETStepByStep4.mdf 数据库，在组合框中选择它，并单击“下一步”。此时会提示设置连接字符串的名称，接受 Visual Studio 生成的默认设置，单击“下一步”(见下图)。

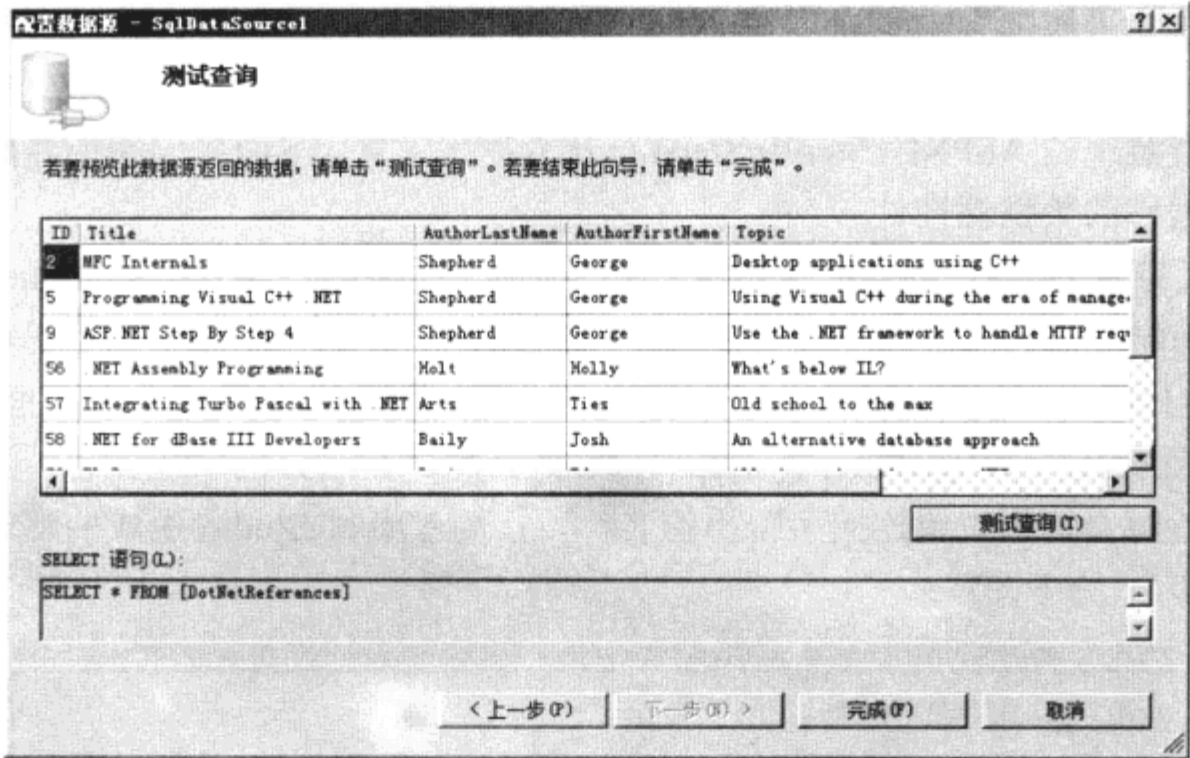


3. 在配置查询时，选择 DotNetReferences 表的所有列和行。为获得所有列，选择星号(*) 即可(见下图)。单击“下一步”。



ASP.NET 4从入门到精通

4. 如果愿意，可以单击“测试查询”按钮来预览查询结果(见下图)。单击“完成”。



5. 将数据源的 DataSourceMode 属性设置为 DataReader。为此，可以在设计器中单击该数据源控件，然后在“属性”窗口中修改这个属性。
6. 拖动一个 ListBox 到页面上。在“ListBox 任务”菜单中，选择“启用 AutoPostBack”。在页面的代码隐藏文件中，找到 Page_Load 方法，将 ListBox 的 DataSource 属性设置为 SqlDataSource1(在实践中，多数情况下可能不这么做，这里只是为演示这种数据绑定方式是如何在代码中实现的)：

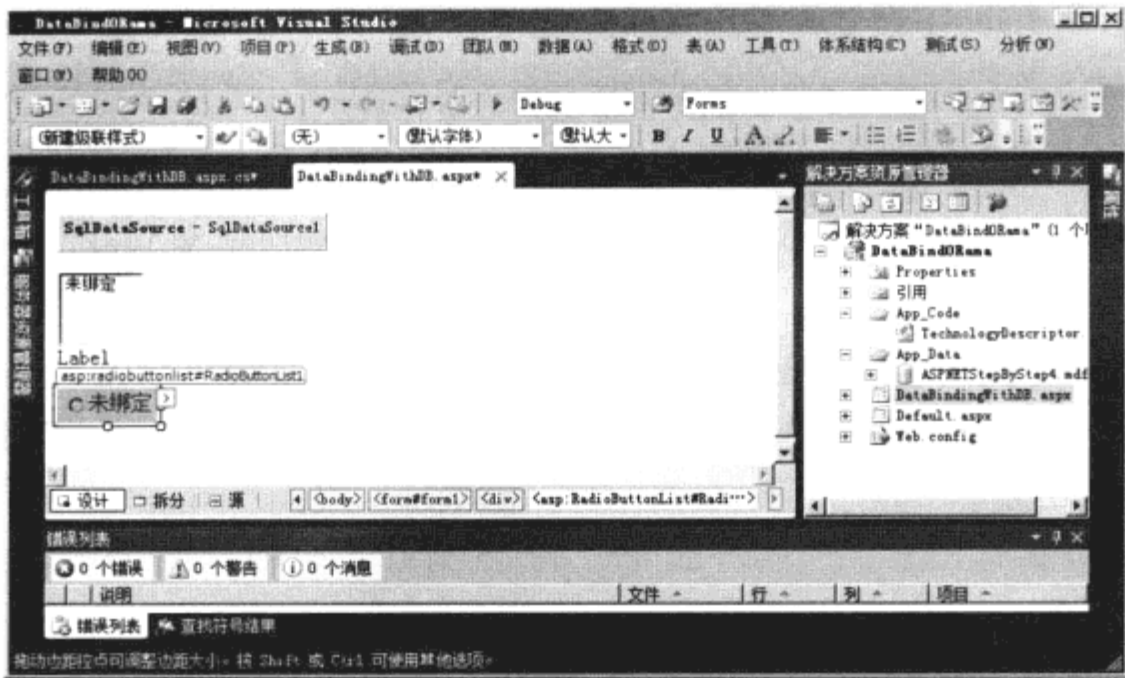
```
protected void Page_Load(object sender, EventArgs e)
{
    if (!this.IsPostBack)
    {
        this.ListBox1.DataSource = this.SqlDataSource1;
        this.ListBox1.DataTextField = "AuthorLastName";
        this.ListBox1.DataValueField = "Title";
        this.ListBox1.DataBind();
    }
}
```

7. 在 ListBox 的下方插入一个标签，稍后将用它来显示 ListBox 中选定的值。
8. 双击 ListBox1，在代码中为其添加一个选定项变更事件的处理程序。在这个事件处理程序中，将 Label1 的 Text 属性设置为选定项的值。

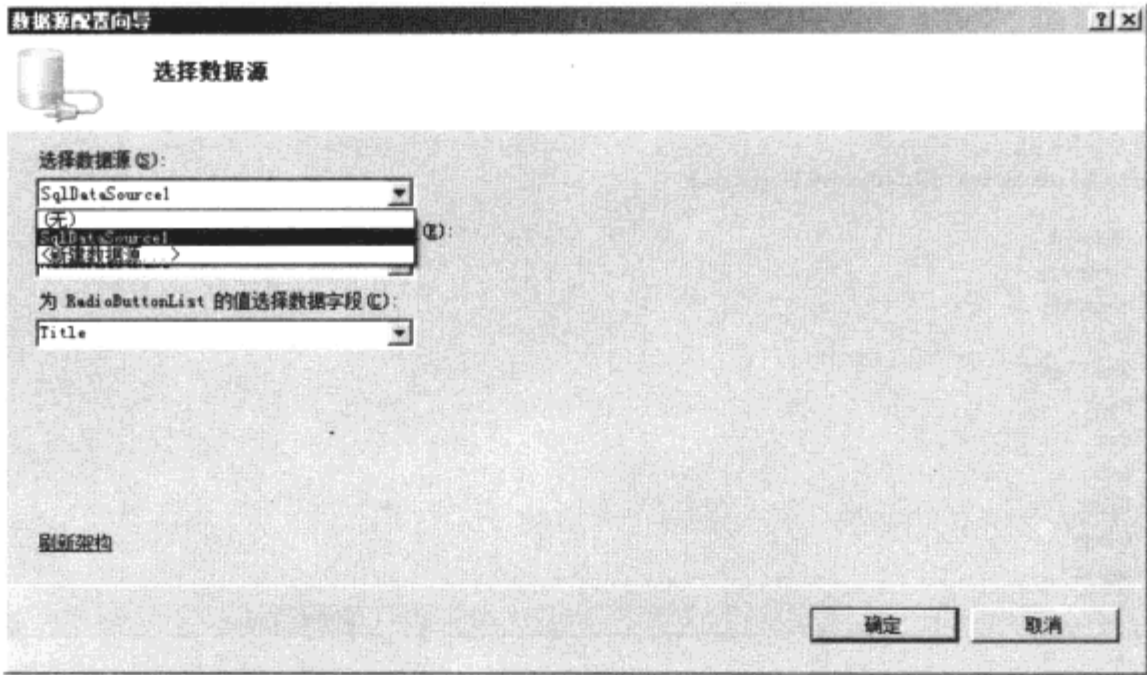
```
Protected void ListBox1_SelectedIndexChanged(object sender,
    EventArgs e)
{
    this.Label1.Text = this.ListBox1.SelectedItem.Value;
}
```

9. 在页面上添加一个 RadioButtonList 控件(如下图所示)。Visual Studio 会询问是否配置该

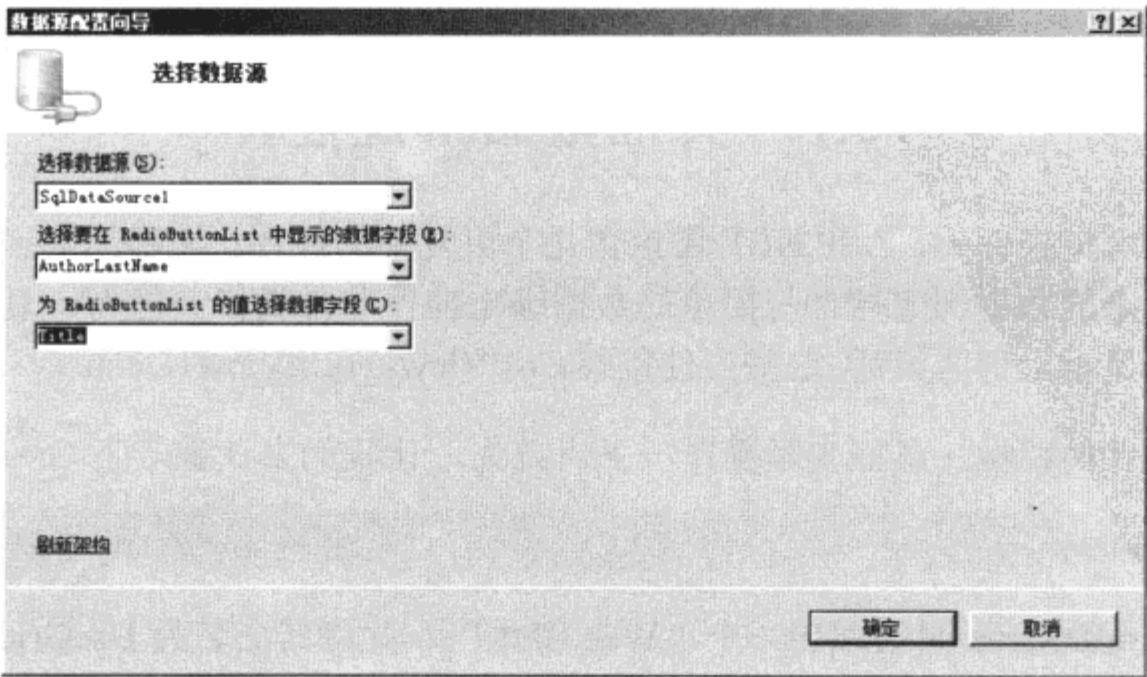
控件。先选择“启用 AutoPostBack”复选框，然后单击“选择数据源”。



10. 如下图所示，让这个控件引用刚刚添加的 SqlDataSource1 数据源。



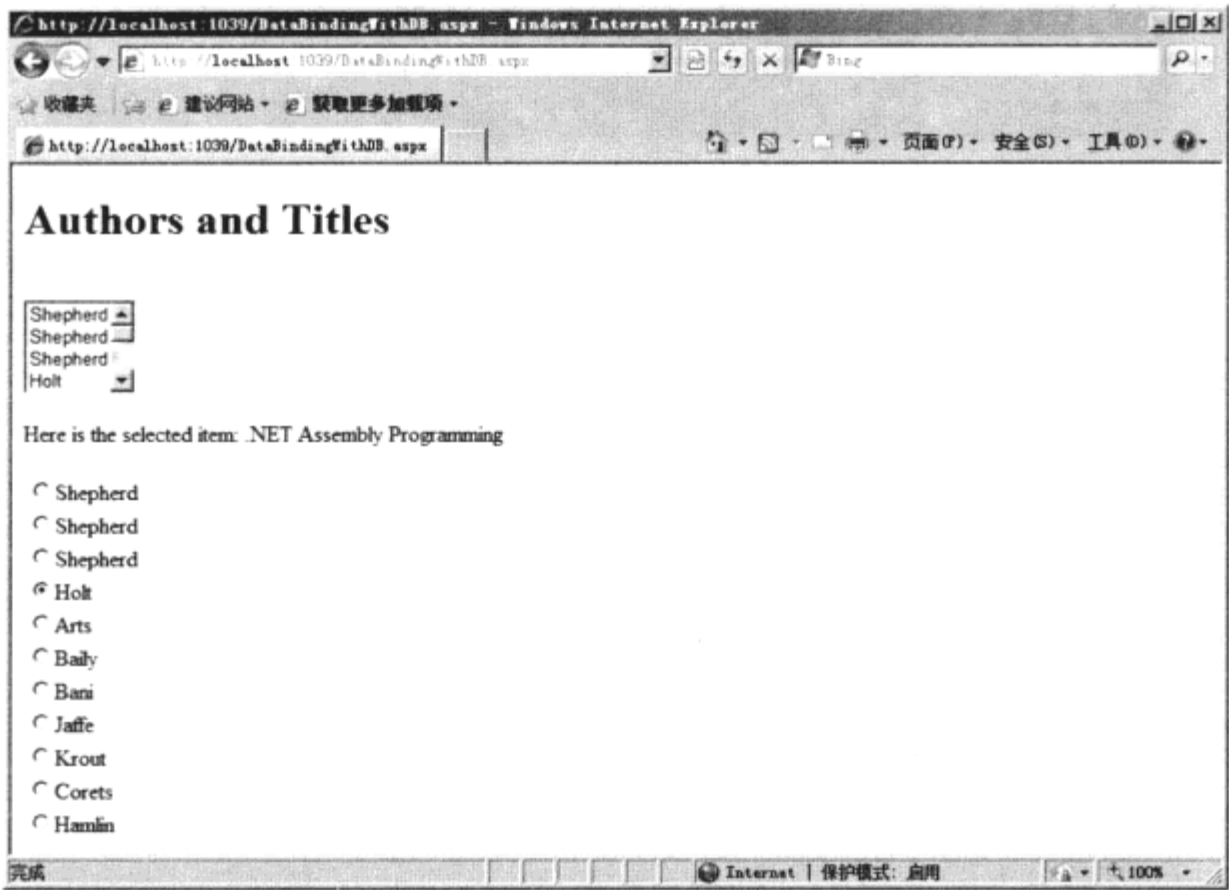
11. 配置该控件，将文本字段设置为 AuthorLastName，将值字段设置为 Title(见下图)。单击“确定”。



12. 在页面上双击 RadioButtonList1 控件，为其添加一个事件处理程序。在新建的事件处理程序中，将 Label1 设置为单选按钮选定项的值。

```
protected void RadioButtonList1_SelectedIndexChanged(object sender,
    EventArgs e)
{
    this.Label1.Text = this.RadioButtonList1.SelectedItem.Value;
}
```

13. 运行这个页面。此时会发现 ListBox 和 RadioButtonList 都会显示 AuthorLastName 字段 (如下图所示)。在这两个列表中进行选择都会导致页面回发，并将相应的书名 (Title 字段的值) 显示在标签中。



这个示例让我们对简单的数据绑定控件有了直观的认识。虽然这些控件很常用，但数据绑定控件并不只这些。ASP.NET 还提供了另外一些控件，能够以多种形式呈现更为复杂的 UI (如网格和控件组合)。

10.7 其他数据绑定控件

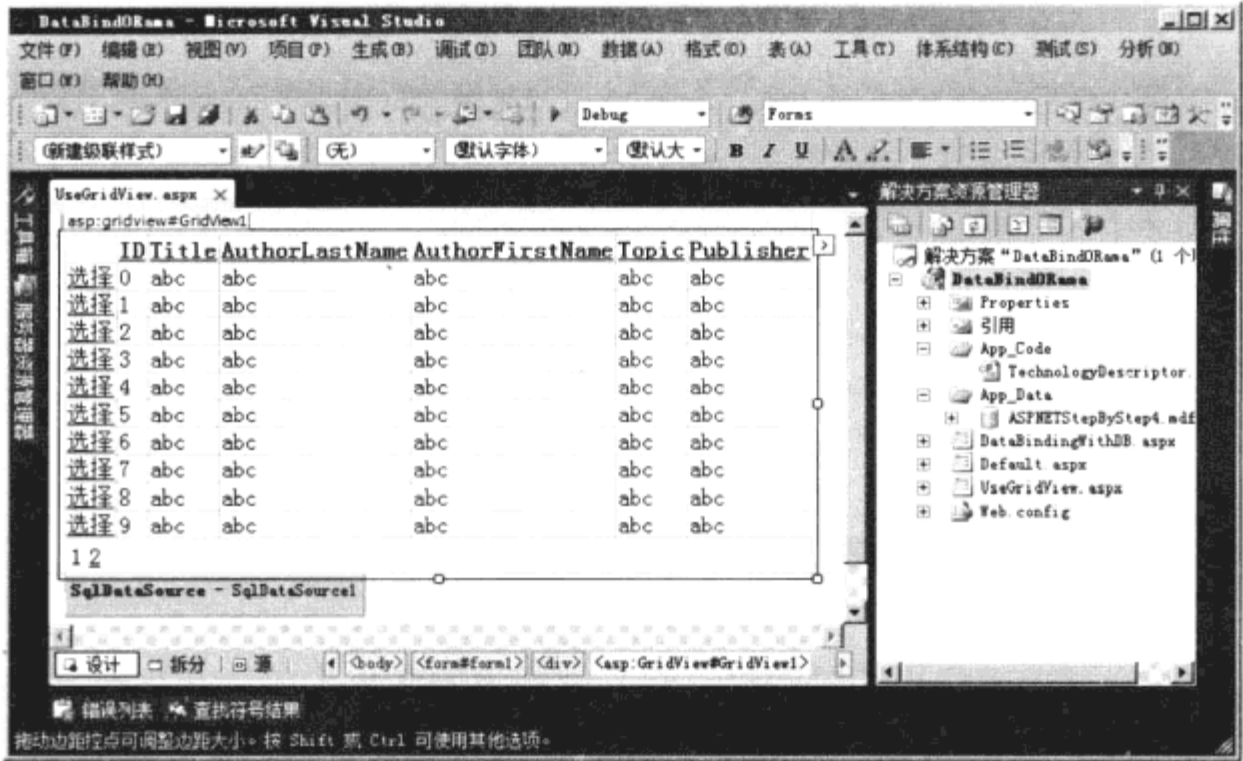
除简单的数据绑定控件外，ASP.NET 还包含几个更复杂的控件。在附加数据源和自动呈现方面，这些复杂的数据绑定控件与简单的数据绑定控件非常类似。然而，复杂的数据绑定控件在数据的显示上更为灵活。这些控件包括 GridView、FormView、DetailsView 和 DataList。

为理解这些控件的特性，最好实际操作一下。首先，让我们来认识一下 GridView。

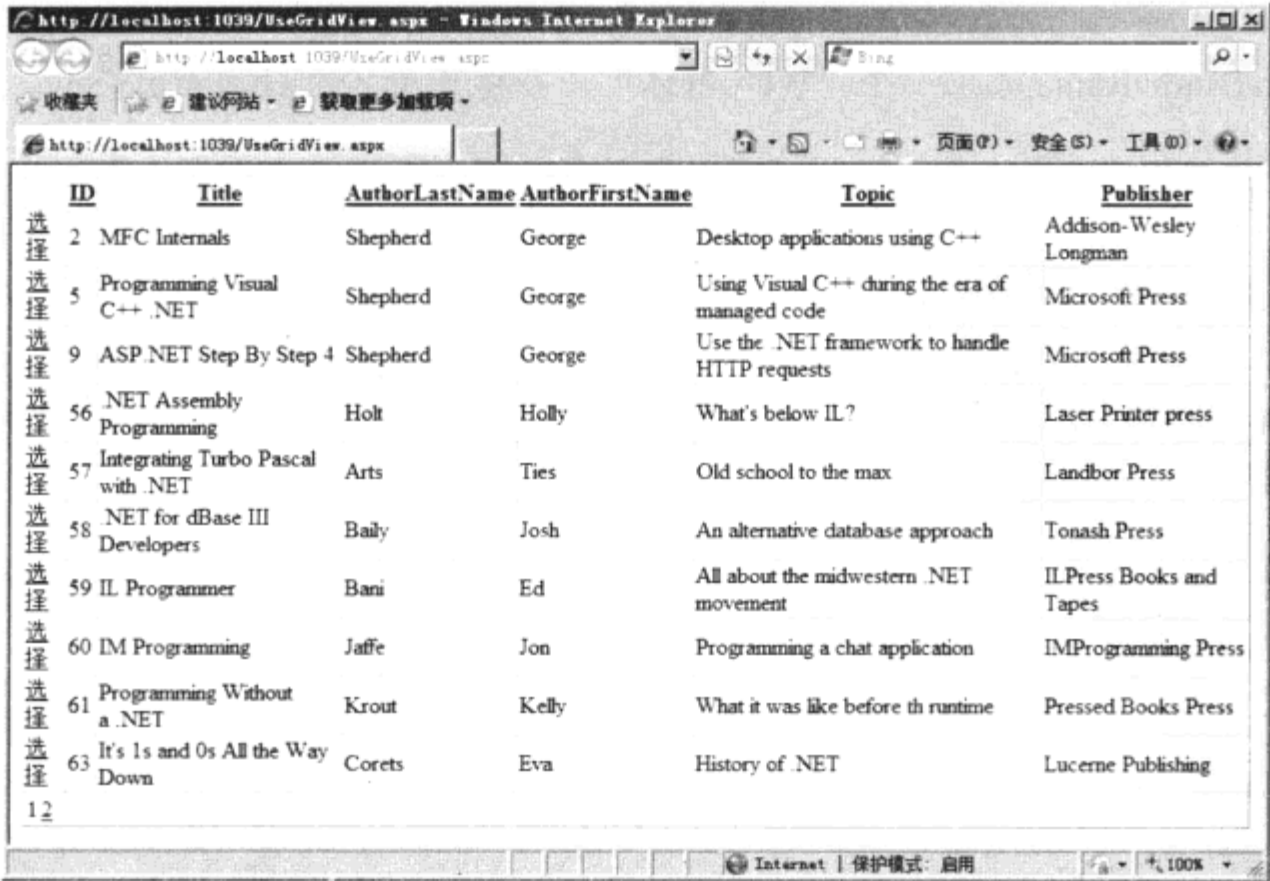
➤ 使用 GridView

1. 为 DataBindORama 网站添加一个“Web 窗体”，并将其命名为 UseGridView。

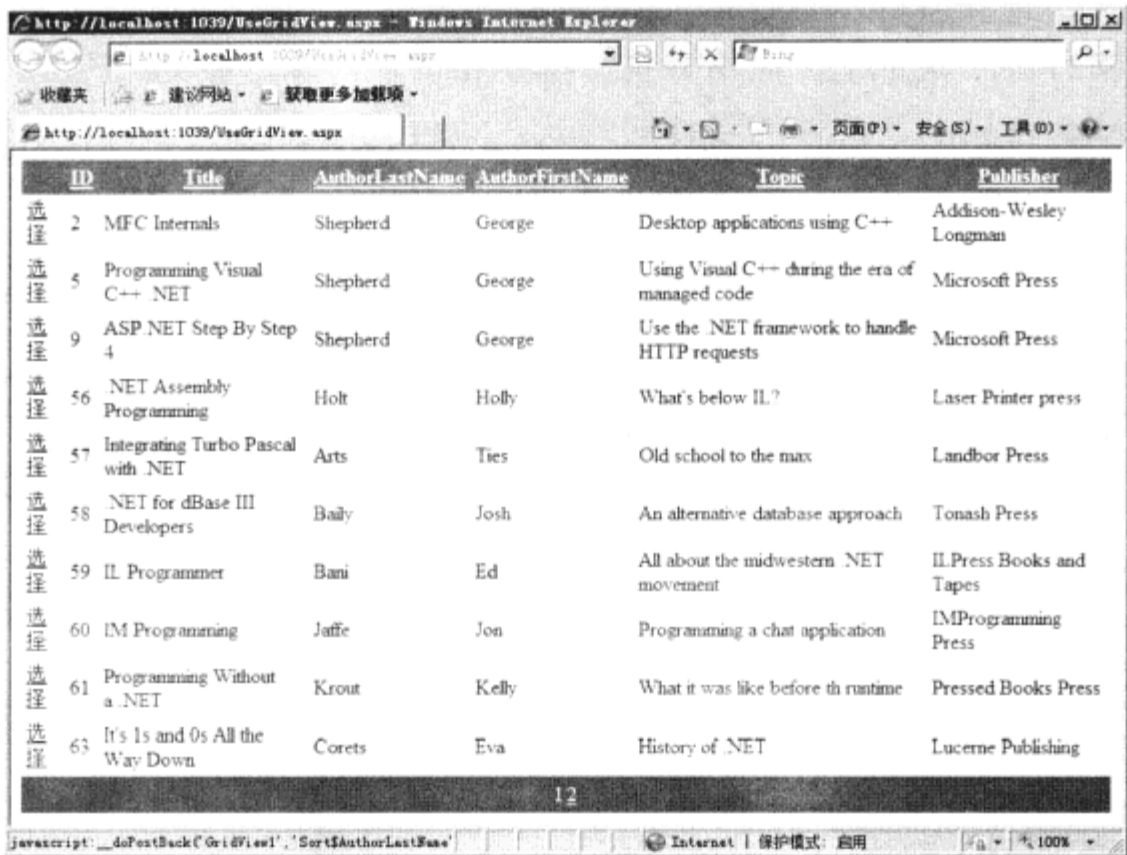
2. 从“工具箱”中拖动一个 GridView 控件到页面上(该控件位于“数据”选项卡中)。Visual Studio 会询问是否配置 GridView。在“选择数据源”组合框中单击“新建数据源”。然后选择“数据库”，单击“确定”。此时，Visual Studio 会创建一个 SqlDataSource，并将其添加到页面上(这个数据源的默认名称为 SqlDataSource1)。Visual Studio 会提示指定连接字符串。可以使用上一个练习创建的连接字符串(它存储于配置文件中)。在设置查询时，选择 DotNetReferences 表，并选择星号(*)来获取所有列。最后，在“GridView 任务”菜单上，启用分页、排序和选定内容。在 GridView 配置完毕后，Visual Studio 会显示该控件的设计时样式，该样式与在浏览器中显示的基本一致(见下图)。



3. 运行这个页面，结果如下图所示。试试不同选项(如分页和排序)，实际体验一下 GridView 的特性。



4. 返回 Visual Studio 修改 GridView 的外观。与其他 ASP.NET 控件类似，GridView 也有许多可配置的属性(如前景和背景颜色)。该控件特有的属性有 `AlternateRowStyle`、`PagerSettings` 和 `PagerStyle`。如果愿意，可以选择自动套用格式。许多 ASP.NET 控件都支持该功能(包括 GridView 在内)。下图展示了 `UseGridView.aspx` 页面在应用“传统型”样式后的效果：



GridView 非常适合显示表格形式的数据，它能够通过同时显示的行和列清晰地展示数据。虽然传统的 DataGrid 仍可用，但 GridView 提供了按行选择和按列排序这样的功能。

下面让我们来了解一下另一个复杂控件——FormView。

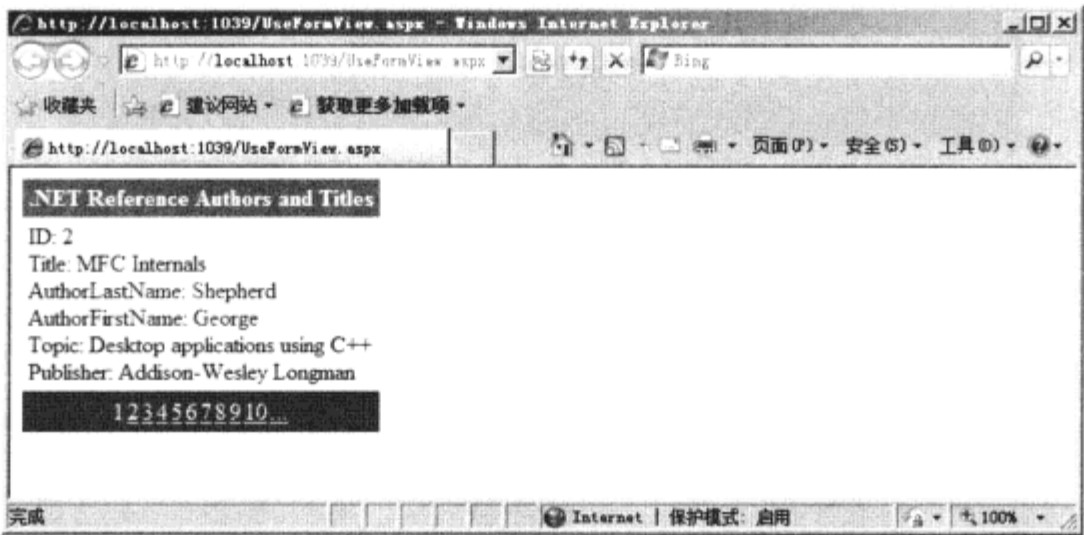
➤ 使用 FormView

1. 为 `DataBindORama` 添加一个“Web 窗体”，并将其命名为 `UseFormView`。
2. 从“工具箱”拖放一个 `FormView` 控件到页面上(该控件位于“数据”选项卡中)。Visual Studio 会提示对这个 `FormView` 进行配置。在“选择数据源”组合框中单击“新建数据源”，选择“数据库”，并单击“确定”。选择一个现有的连接字符串(在组合框中选择“`ConnectionString`”)，单击“下一步”。在设置查询时，选择 `DotNetReferences` 表，并选择星号(*)来获取所有列，单击“下一步”。在“测试查询”页面中，单击“完成”。
3. 在“FormView 任务”菜单中选择“自动套用格式”。此时会列出一些为 `FormView` 预定义的样式。这里选择“传统型”。
4. 在“FormView 任务”菜单中选择“启用分页”。在 Visual Studio 的“属性”窗口中，将 `FormView` 的 `HeaderText` 属性设置为一个标题(如“.NET Reference Authors and Titles”)。

5. 在 FormView 配置完毕后，Visual Studio 会显示该控件的设计时样式，该样式与在浏览器中显示的基本一致，见下图。



6. 运行这个页面，结果如下图所示。试试不同选项(如分页)，实际体验一下 FormView 的特性。



FormView 适合在一处集中显示单条记录的信息。用户可以在不同记录之间导航，但每次只在关注当前的记录。

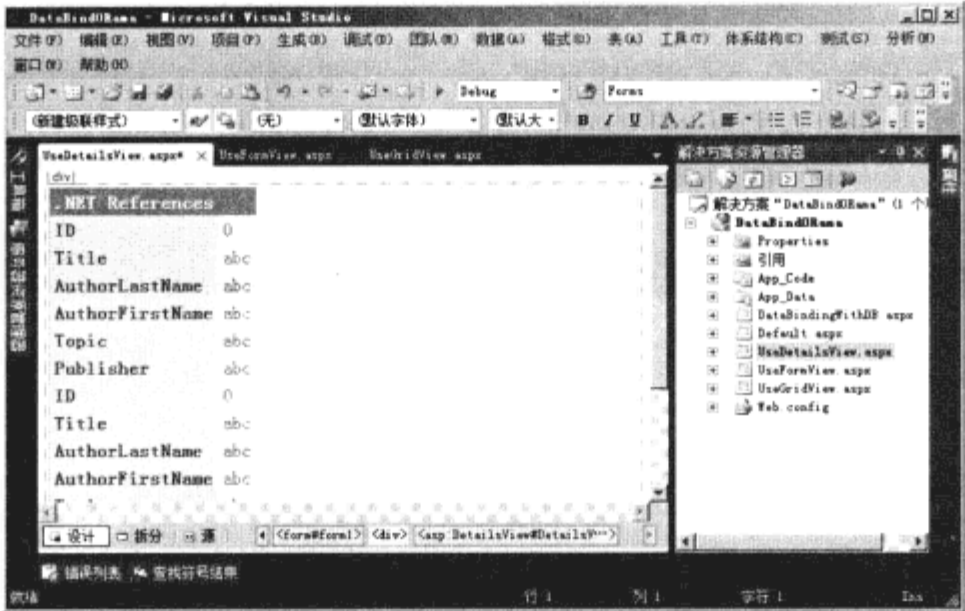
DetailsView 是 ASP.NET 中一个与 FormView 类似的控件。该控件允许开发者以表格形式呈现每条记录，但格式的设置不如 FormView 那样灵活。

► 使用 DetailsView

1. 为 DataBindORama 添加一个“Web 窗体”，并将其命名为 UseDetailsView。
2. 从“工具箱”拖放一个 DetailsView 控件到页面上(该控件位于“数据”选项卡中)。Visual Studio 会提示对这个 DetailsView 进行配置。在“选择数据源”组合框中单击“新建数据源”，配置方法与之前一样。例如，还是选择之前 Visual Studio 创建的那个连接字符串。在设置查询时，还是选择 DotNetReferences 表，并选择星号(*)来获取所有列。
3. 在“DetailsView 任务”菜单中选择“自动套用格式”。与之前一样，可以在这里为

DetailsView 选择预定义的样式。这里选择“传统型”。此外，通过 HeaderText 属性为该控件添加一个标题(本示例输入的是“.NET References”)。

- 4. 在“DetailsView 任务”菜单中选择“编辑字段”。在“字段”对话框中选择“自动生成字段”复选框，如果它尚未被选中的话。
- 5. 在“DetailsView 任务”菜单中选择“启用分页”。
- 6. 在 DetailsView 配置完毕后，Visual Studio 会显示该控件的设计时样式，该样式与在浏览器中显示的基本一致，如下图所示。



- 7. 运行这个页面，结果如下图所示。试试不同选项(如分页)，实际体验一下 DetailsView 的特性。^①

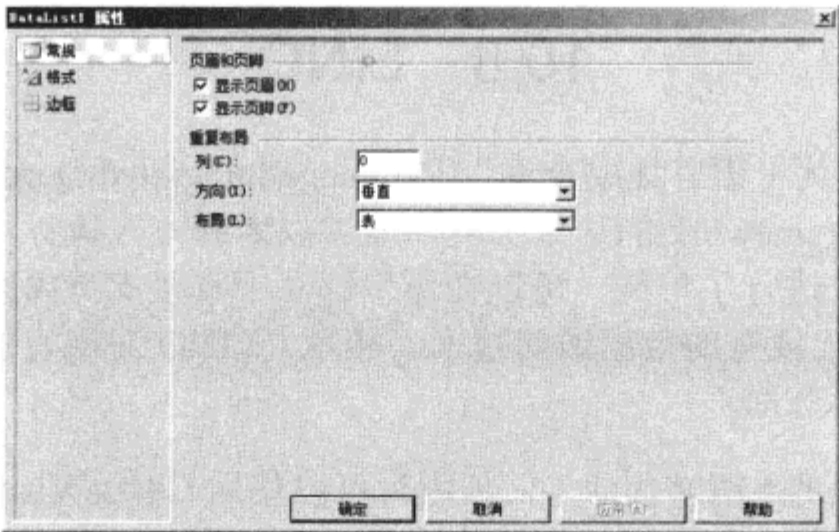


^① 译者注：不难发现，页面把同一条记录显示了两次。上部呈现的字段是 Visual Studio 在添加数据源时逐一添加到代码中的，而下部是 ASP.NET 在运行时自动生成的。前者可以在第 4 步打开的对话框中可以看到，而自动生成是因为选择了该对话框中的“自动生成字段”。

下面让我们来认识一下 DataList。ASP.NET 1.x 就已经包含 DataList 控件了。在之后的版本中，该控件得到了更新，以支持声明式数据绑定。DataList 能够以表格形式呈现数据，这与 DetailsView 类似，而不同在于，DataList 没有内建的分页功能。

➤ 使用 DataList

1. 为 DataBindORama 添加一个“Web 窗体”，并将其命名为 UseDataList。
2. 从“工具箱”拖放一个 DataList 控件到页面上(该控件位于“数据”选项卡中)。Visual Studio 会提示对这个 DataList 进行配置。在“选择数据源”组合框中单击“新建数据源”，配置方法与之前一样。选择之前 Visual Studio 创建的那个连接字符串。在设置查询时，选择 DotNetReferences 表，并选择星号(*)来获取所有列。
3. 在“DataList 任务”菜单中选择“自动套用格式”。与之前一样，可以在这里为 DataList 选择预定义的样式。这里选择“石板”。
4. 在“DataList 任务”菜单中选择“属性生成器”。此时会打开下图所示的“DataList 属性”对话框，确保“显示页眉”和“显示页脚”复选框已被选中。



5. 通过 Caption 属性为这个 DataList 设置一个标题(如“.NET References and Titles”)。
6. 在 DataList 配置完毕后，Visual Studio 会显示该控件的设计时样式，该样式与在浏览器中显示的基本一致(见下图)。





7. 运行这个页面，查看 DataList 呈现的效果(见下图)。



虽然传统的数据访问技术仍可以使用，但 .NET 3.0 和后续版本提供了另一种数据访问和管理方法——语言集成查询(LINQ)。下面让我们一探究竟。

10.8 LINQ

.NET 3.0 引入了一项名为“语言集成查询”(Language Integrated Query, LINQ)^①的数据访问技术。LINQ 是 .NET Framework 的一组扩展，能够以内联方式执行数据查询。LINQ 对 C# 和 Visual Basic 的语法进行了扩展，可以在原生语法中直接支持内联查询(相对于 SQL 或 XPath)。LINQ 并未取代现有的数据访问技术。相反，LINQ 对现有的数据查询技术进行了改进，能够执行高效的查询。

这种技术之所以特别强调“语言集成”，是因为可以使用 C#(或 Visual Basic)语言构造来构建查询的选择语句。下面我们看看如何构建 LINQ 查询语句。

► 使用 LINQ

1. 为 DataBindORama 添加一个“Web 窗体”，并将其命名为 UseLinq。
2. 拖放一个 GridView 到页面上。稍后将使用该控件来呈现 LINQ 查询返回的结果。
3. 在 Page_Load 方法中执行 LINQ 查询。将本章前面提到的 TechnologyDescriptor 集合作为 LINQ 查询的数据源。将针对 TechnologyDescriptor 集合的 LINQ 查询所返回的结果赋给 GridView 的 DataSource 属性。LINQ 语句的格式如下(C#):

```
from <与集合元素类型相同的变量> in <集合>
where <条件>
orderby <条件>
select <选定项的属性>
```

① 译者注：LINQ 一般读作[lɪŋk]，与英文单词 link 的发音相同。

在 TechnologyDescriptor 集合中选择名称中包含 “.NET” 字样的记录，并按 TechnologyName 属性的长度进行排序。代码如下所示：

```
public partial class UseLinq : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
        {
            List<TechnologyDescriptor> techList =
                TechnologyDescriptor.CreateTechnologyList();

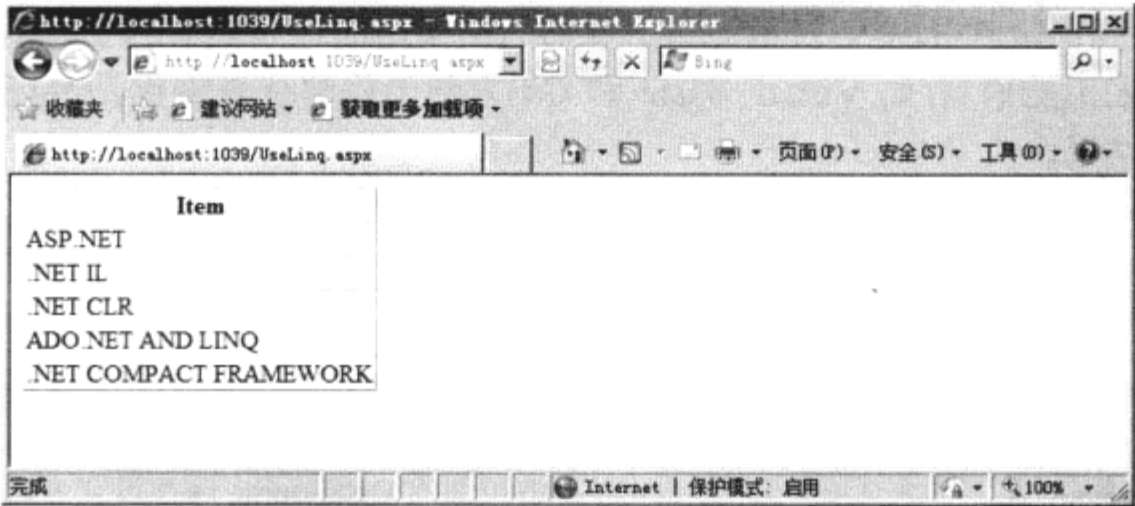
            GridView1.DataSource =
                from technologyDescriptor in techList

                where technologyDescriptor.TechnologyName.Contains(".NET") == true

                orderby technologyDescriptor.TechnologyName.Length

                select technologyDescriptor.TechnologyName.ToUpper();
            GridView1.DataBind();
        }
    }
}
```

4. 运行 UseLinq.aspx 页面，看看这个查询是如何填充 GridView 的：



5. 注意，GridView 此时只显示了 TechnologyDescriptor 中的一个属性。下面我们对查询语句进行更新，使其显示 TechnologyDescriptor 的整个结构。代码如下所示：

```
public partial class UseLinq : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
        {
            List<TechnologyDescriptor> techList =
                TechnologyDescriptor.CreateTechnologyList();

            GridView1.DataSource =
```

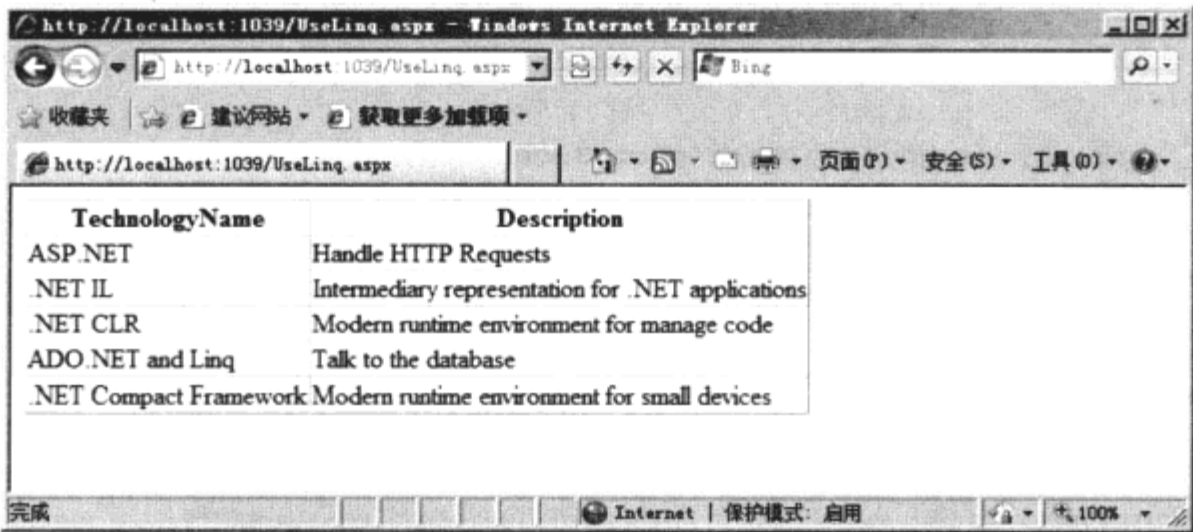
```
from technologyDescriptor in techList

where
    technologyDescriptor.TechnologyName.Contains(".NET") == true

orderby technologyDescriptor.TechnologyName.Length

select technologyDescriptor;
GridView1.DataBind();
}
}
}
```

6. 再次运行这个页面。此时，GridView 会把整个 TechnologyDescriptor 都显示出来。



这个示例演示了在 GridView 中显示 LINQ 查询的结果，而这只是 LINQ 的基本功能。我们可以使用项目采用的语言(如 Visual Basic 和 C#)方便地构建任意查询，并可以在任意上下文中使用查询结果。

10.9 快速参考

目 标	操 作
将集合绑定到控件	将控件的 DataSource 属性设置为这个集合
选择在控件中显示的列	将控件的 DataTextField 设置为列的名称(部分控件具有这个属性)
选择程序使用的列(不显示在控件中)	将控件的 DataValueField 属性设置为列的名称(部分控件具有这个属性)
以网格的形式呈现 DataTable	使用 DataGrid 或 GridView 控件(后者更好)
将 DataTable 呈现为带有格式的、重复的列表	使用 DataList
使类的成员变量能够分配给控件的 DataTextField 和 DataValueField 属性	将这些成员以属性的形式暴露出来
以“主/从”视图的形式呈现数据	使用 FormView 控件



第 11 章

网站的导航

学习目标

- 理解 ASP.NET 对导航与站点地图的支持
- 实现基于 XML 数据源的站点地图
- 将站点地图数据绑定到 ASP.NET 导航控件
- 捕获和响应站点地图导航事件

使用户能够有效地在页面间进行导航，是网站开发所要面对的主要问题之一。网站通常采用层次型结构，页面可能有几层嵌套，因而用户可能会有“现在在哪儿”或“如何从当前位置链接到目标”的困惑。ASP.NET 对网站导航方面的支持可以使这些问题迎刃而解。

网站设计已经超越了传统意义上的导航概念，成为了一种艺术，变幻莫测。浏览不同平台承载的网站便不难发现，网站一般有许多方式来对其内容进行导航。例如，许多网站在每个页面的顶部设有菜单栏，能够链接到网站中的不同页面。有些网站提供某种树形结构的视图来提供导航服务。还有些网站能够显示导航路径(或称为“面包屑”， breadcrumb)。ASP.NET 提供了上述所有功能。

11.1 ASP.NET 对导航的支持

ASP.NET 对导航的支持主要体现在 3 方面：导航控件、站点地图数据源和站点地图提供程序架构。“导航控件”(Menu、TreeView 和 SiteMapPath)能够为用户显示便于阅读的 URL 的名称，HTTP 请求会被发送到这些 URL。“站点地图数据源”用于存储有关网站层次型组织结构的信息。“站点地图提供程序”能够解释物理数据(一般来自 XML 文件)，并实现了一种类似数据库游标的机制，能够提供当前页面在网站的层次结构中的位置。

11.1.1 导航控件

ASP.NET 包含 3 种支持导航的服务器端控件：SiteMapPath、Menu 和 TreeView。TreeView 和 Menu 控件能够维护显示名称与 URL 映射的集合。开发者可以手动对这个集合进行编辑。此外，这些控件能够根据站点地图数据源提供的信息来构建显示名称与 URL 映射的层次型集合。SiteMapPath 控件能够根据站点地图数据源构建显示名称与 URL 映射的集合。表 11.1 总结了这几个 ASP.NET 导航控件。

表 11.1 ASP.NET 的导航控件

导航控件	说 明
Menu	该控件能够解释站点地图 XML 文件包含的站点导航信息，并将其以菜单的形式呈现出来。顶层的 XML 节点对应顶层的菜单项，子 XML 节点对应子菜单项
TreeView	该控件能够解释站点地图 XML 文件中包含的站点导航信息，并将其以树形视图的形式呈现出来。顶层的 XML 节点对应树根，子 XML 节点对应树的子节点
SiteMapPath	该控件能够解释站点地图 XML 文件中包含的站点导航信息，并将其以“面包屑”的形式呈现出来。这个控件只会显示当前 XML 节点的路径(从根节点到当前节点)

在实现导航功能时可以使用这三个控件，SiteMapPath 只能根据网站的站点地图 XML 文件实现导航功能，但 Menu 和 TreeView 则不止于此。Menu 控件能够以层次形式显示链接项，并在某一项被选择后执行回发。还可以为 Menu 控件中的项目分配导航 URL。TreeView 能够显示任何实现了 IHierarchicalDataSource 或 IHierarchicalEnumerable 接口的层次型数据源的内容，也能够重定向到其他 URL(适用于站点导航)。

结构简单的网站变动可能不大，因此从头构建导航基础设施的代价也不大。然而，随着网站越来越复杂，管理导航结构的难度也会越来越大。

在规划网站并确定页面的布局后，可以方便地通过母版页来实现布局，而在母版页中可以包含一个链接到其他页面的菜单。(第 7 章介绍了母版页。)为此，需要创建一个菜单并添加所需的链接(通过菜单项的 NavigateUrl 属性)。最初，手动实现这种导航基础设施非常简单。然而，随着时间的推移，网站逐渐被扩展，情况会变得更为复杂。此时，可能就必须提供导航支持。

ASP.NET 支持导航和站点地图的原因正在于此。使用 ASP.NET 导航支持的优势在于，可以先确定网站组织结构，然后通过层次型数据结构(如 XML 文件或数据库表)来表示它。Menu、TreeView 和 SiteMapPath 控件能够引用站点地图数据源，并根据其中的数据来填充自身。在将站点地图数据源关联到导航控件后，导航控件便可以通过这个数据源来生成所有链接。

在建立站点地图后，如果要更新导航链接，更新站点地图即可。所有使用站点地图数据源的控件会自动反映所有变更。

11.1.2 XML 站点地图

ASP.NET 内建了一种导航功能，支持通过描述站点布局的 XML 文件来实现导航。这种文件被称为“XML 站点地图”(XML site map)。ASP.NET 默认的站点地图支持由一个 XML 文件和 SiteMapProvider 组成，XML 文件用来描述站点组织结构，SiteMapProvider 则会读取这个 XML 文件，并为相关的控件(如 Menu 或 TreeView 控件)生成 SiteMap 节点。

11.1.3 SiteMapProvider

SiteMapProvider 是一个基类，导航控件会使用其子类的对象。ASP.NET 对这个类的默认实现是 XmlSiteMapProvider，这个类(默认)会读取名为 web.sitemap 的 XML 文件。

虽然默认的 XML 站点地图适用于一般情况，但 ASP.NET 导航控件也支持来自其他位置的数据源(而不仅仅是 XML 数据源)。例如，可以自定义一种基于数据库中数据的站点地图提供程序。XML 网站地图为站点导航提供了基本的功能。如果不希望使用基于默认 XML 架构的站点地图数据，那么可以实现自定义的提供程序。

本章将介绍默认的 XML 站点地图提供程序——它足以适用于大多数情况了。

11.1.4 SiteMap 类

SiteMap 类是 ASP.NET 导航基础架构的核心组成部分。为支持导航基础架构，SiteMap 类提供了一套用于管理站点导航的静态方法。SiteMap 类是站点导航结构在内存中的表示，它的具体功能由一个或多个站点地图提供程序实现。该类是抽象类，因而必须派生实现类。

SiteMap 类具有几种功能。首先，它可以作为网站层次结构的根节点。其次，它能够确定主要的站点地图提供程序。最后，它能够跟踪构成站点地图的对象。

SiteMap 包含一个由 SiteMapNode 对象构成的层次型集合。不论站点地图数据是以何种形式组织的，都可以通过 SiteMap 的接口来访问网站的导航信息。

ASP.NET 的默认配置文件中指定了一个默认的站点地图。当然，与 ASP.NET 的其他设置一样，开发者可以方便地覆盖默认配置来指定不同的提供程序。

SiteMap 类只提供了一些静态成员。使用静态成员有助于改善性能。此外，在 Web 应用程序的页面和服务端控件的任意位置，都可以使用站点地图功能。

表 11.2 列出了 SiteMap 类的属性和特有的事件。

表 11.2 SiteMap 的事件及属性

名 称	类 型	说 明
SiteMapResolve	事件	该事件会在 CurrentNode 属性被访问时引发。这使开发者能够在创建代表当前页面的 SiteMapNode 时插入自定义的逻辑，而无须编写自定义的提供程序
CurrentNode	属性	用于获取代表导航层次结构中当前页面的 SiteMapNode 实例。如果 XML 站点地图中不包含节点，该属性则会返回 null

续表

名 称	类 型	说 明
Enabled	属性	该属性能够返回一个布尔值，指示 web.config 文件中是否指定并启用了某个站点地图提供程序
Provider	属性	用于获取当前站点地图的默认 SiteMapProvider
Providers	属性	能够返回 SiteMapProvider 对象的只读集合，这些对象是在 web.config 文件中指定的(可以同时指定多个)，但在初始化阶段只会使用默认的提供程序
RootNode	属性	能够返回代表网站导航层次结构顶层页面的 SiteMapNode 对象

11.1.5 SiteMapNode

SiteMapNode(站点地图节点)代表了站点地图的层次结构元素。也就是说，每个 SiteMapNode 实例对应网站中的一个页面。在 Web 应用程序启动时，SiteMap 会通过网站的 web.config 指定的提供程序加载 SiteMapNode 集合。

SiteMapNode 的常用属性包括：ChildNodes、Description、HasChildNodes、Key、NextSibling、ParentNode、PreviousSibling、Provider、ReadOnly、ResourceKey、Roles、RootNode、Title 和 Url。SiteMapNode 的常用方法包括：GetAllNodes、GetDataSourceView、GetHierarchicalDataSourceView、IsAccessibleToUsers 和 IsDescendentOf。后面的练习会用到这些属性中的一部分。例如，在练习中，SiteMapResolve 事件的处理程序会通过这些属性来动态修改导航结构。

11.2 导航控件的使用

运行 Microsoft Visual Studio 2010 后，我们便可以在“工具箱”的“导航”选项卡中找到 3 个控件：Menu、TreeView 和 SiteMapPath。下面将介绍这些控件。

11.2.1 Menu 控件与 TreeView 控件

Menu 控件和 TreeView 控件能够绑定到实现了 IHierarchicalDataSource 或 Ihierarchical-Enumerable 的层次型数据源。虽然这里将这两个控件与站点地图关联，但两者也都适用于其他数据源。图 11.1 和图 11.2 分别展示了 Menu 控件和 TreeView 控件在浏览器中的效果。图中的这两种控件的数据均来自站点地图数据源。



图 11.1 浏览器中的 Menu



图 11.2 浏览器中的 TreeView

11.2.2 SiteMapPath 控件

我们有时会在网站上遇到与 SiteMapPath 控件类似的用户界面(UI)元素——尤其是在浏览层次较深的页面时。SiteMapPath 控件能够显示当前页面处在网站的层次结构中的位置，并提供最终到根节点的返回路径(与“面包屑”路径很像)。对于层次结构较深而导致 Menu 控件和 TreeView 控件都不适用的情况，可以使用 SiteMapPath 控件。

虽然 SiteMapPath 控件在某些方面类似 Menu 控件和 TreeView 控件(SiteMapPath 控件能够反映 SiteMap 的状态)，但有几点需要特别注意。SiteMapPath 控件和提供程序中的站点地图是紧密耦合的。举例来说，如果某个页面不在站点地图中，那么在用户浏览该页面(通过其他导航方式)时，他/她就不会在该页面上看到 SiteMapPath 控件。(正因为此，这个控件在许多情况下会被嵌在母版页中。)图 11.3 展示了浏览器中的 SiteMapPath 控件。在图中，用户可以使用 SiteMapPath 控件下的 Menu 控件来导航到具体的页面。(如果没有 Menu 控件或 TreeView 控件，用户将无法进入站点地图的更深层次。)



图 11.3 浏览器中的 SiteMapPath

11.2.3 站点地图的配置

ASP.NET 的全局配置采用的是默认的 XmlSiteMapProvider。清单 11.1 展示了默认 web.config 的部分配置信息。当然，这一切都是可配置的。在特定的网站中，可以通过修改自身的 web.config，将默认的站点地图提供程序替换为其他的。

清单 11.1 站点地图的默认配置

```
<siteMap>
  <providers>
    <add siteMapFile="Web.sitemap" name="AspNetXmlSiteMapProvider"
      type="System.Web.XmlSiteMapProvider, System.Web, Version=4.0.0.0,
      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  </providers>
</siteMap>
```

除了在 web.config 中添加配置信息外，Visual Studio 2010 还会在网站地图中添加一个空的顶层节点(如清单 11.2 所示)。

清单 11.2 Visual Studio 2010 添加的默认站点地图

```
<?xml version="1.0" encoding="utf-8" ?>
<sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode url="" title="" description="">
    <siteMapNode url="" title="" description="" />
    <siteMapNode url="" title="" description="" />
  </siteMapNode>
</sitemap>
```

站点地图的修改非常容易——例如，如果要在其中添加几个新节点，只需要以文本(XML)方式进行编辑即可。清单 11.3 展示了带有几个额外节点的 XML 站点地图。

清单 11.3 XML 形式的站点地图

```
<?xml version="1.0" encoding="utf-8" ?>
<sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
```



```
<siteMapNode url=""
  title="Navigation Menu" description="">
  <siteMapNode url="~/Default.aspx"
    title="Home" description="" />
  <siteMapNode url="~/Products.aspx"
    title="Products" description="" />
  <siteMapNode url="~/Support.aspx"
    title="Support" description="" />
  <siteMapNode url="~/Contact.aspx"
    title="Contacts" description="" />
</siteMapNode>
</siteMap>
```

11.3 实现可导航的网站

为网站添加导航支持非常简单。一旦确定网站的层次结构，便可以使用 XML 站点地图文件来进行描述。然后，只需要将页面中的所有导航控件都指向新的 XML 站点地图文件即可。导航控件会自动被填充，并呈现可导航的网站。下面的示例将演示如何为网站添加导航支持，以及如何在应用程序中使用 ASP.NET 导航控件。

➤ 创建站点地图

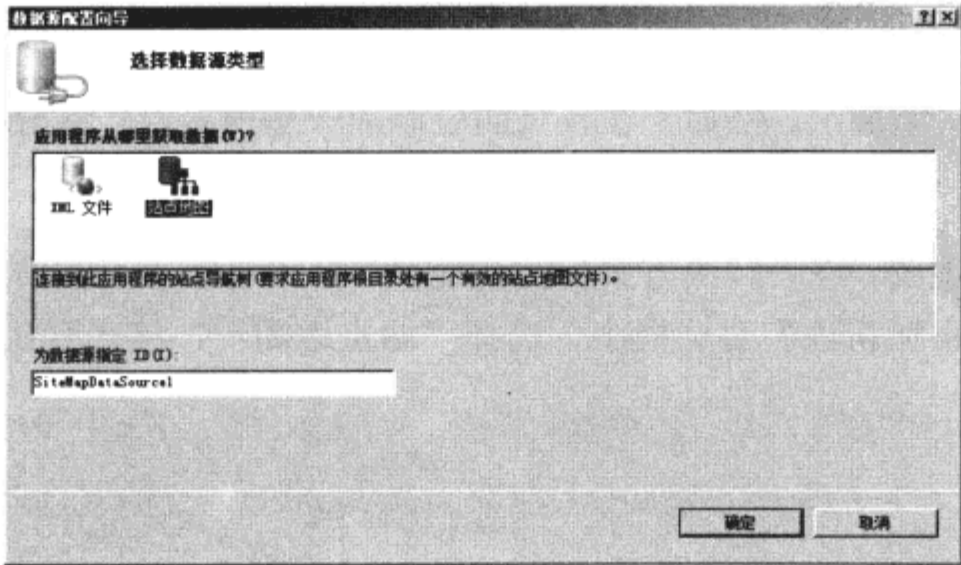
1. 打开 Visual Studio，创建一个“ASP.NET Web 应用程序”，将其命名为 NavigateMeSite。
2. 注意，Visual Studio 会创建一个新网站，其中包含一个可在其中放置控件的母版页。稍后我们将在这个页面添加导航控件。此外，Visual Studio 还会添加 Default.aspx 和 About.aspx 页面。
3. Visual Studio 创建的样式不便于我们看清菜单和导航控件。打开 Site.css，修改 hideSkiplink，将背景从 Visual Studio 设置的深蓝色改为浅灰色。本示例将 hideSkiplink 样式的背景颜色设置为 #C0C0C0。这个样式用于修饰母版页中的菜单背景。
4. 添加 4 个基于母版页的页面：一个默认页面、一个产品页面、一个支持页面和一个联系页面。Visual Studio 提供了一个基于母版页创建页面的模板。为添加以上页面，在项目节点上右击，选择“添加”|“新建项”。在对话框中选择“使用母版页的 Web 窗体”，单击“添加”。然后，在各页面中添加一些内容以作区分(添加一些简单的文本即可)。
5. 添加一个站点地图。在“解决方案资源管理器”的项目节点上右击，选择“添加”|“新建项”。在“添加新项”对话框中，选择“站点地图”，接受默认名称 Web.sitemap，如下图所示。



6. 在这个站点地图中添加以下数据(如果页面的文件名不同，可以修改相应的 URL)。直接编辑(或覆盖)Visual Studio 创建的两个空节点即可：

```
<?xml version="1.0" encoding="utf-8" ?>
<sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode url="~/Default.aspx" title="Home"
    description="This is the home page">
  <siteMapNode url="~/Products.aspx" title="Products"
    description="This is the products page" />
  <siteMapNode url="~/Support.aspx" title="Support"
    description="This is the support page" />
  <siteMapNode url="~/Contact.aspx" title="Contacts"
    description="This is the contacts page" />
</siteMapNode>
</sitemap>
```

7. 为使站点地图生效，需要将站点地图与主菜单关联。在“设计器”中打开 Site.master 文件，选择导航菜单控件(目前只有到 Home 和 About 页面的链接)。单击菜单右上角的箭头，打开“Menu 任务”。在“选择数据源”下拉列表中选择“新建数据源”。此时会显示“数据源配置向导”对话框。在该对话框中，选择“站点地图”，然后单击“确定”。下图展示了如何为 Menu 控件选择站点地图数据源。



8. 为看到 Menu 控件的实际效果, 运行这个网站。单击 Menu 上的选项, 看看是否会导航至相应的页面。
9. 接下来, 从“工具箱”中拖放一个 TreeView 控件到母版页上。使该控件指向默认的站点地图数据源。再次运行这个网站, 看看效果如何。
10. 现在为母版页添加一个 SiteMapPath 控件。将 SiteMapPath 的 DataSource 属性设置为 XML 站点地图数据源。
11. 在项目中再添加两个页面, 以提供两种通过网站联系业务的方式——一个显示公司的物理地址, 另一个显示联系信息(如电子邮件地址和电话号码)。首先, 为这两个页面创建两个文件夹, 分别命名为 ContactAddress 和 ContactEmailPhone。在每个文件夹中添加一个基于母版页的页面, 分别命名为 ContactAddress.aspx 和 ContactEmailPhone.aspx。与之前一样, 为每个页面添加一些标签或文本, 以便在运行这个 Web 应用程序时区分不同页面。
12. 在站点地图 XML 文件(Web.sitemap)中添加两个用于反映新页面的元素。将它们作为 Contact 节点的子节点:

```
<?xml version="1.0" encoding="utf-8" ?>
<sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode url="Default.aspx" title="Home"
    description="This is the home page">
  <siteMapNode url="Products.aspx" title="Products"
    description="This is the products page" />
  <siteMapNode url="Support.aspx" title="Support"
    description="This is the support page"
    ImageURL="supportimage.jpg"/>
  <siteMapNode url="Contact.aspx" title="Contacts"
    description="This is the contacts page">
    <siteMapNode url="~/ContactAddress/ContactAddress.aspx"
      title="Contact using physical address"
      description="This is the first contact page" />
    <siteMapNode url="~/ContactEmailPhone/ContactEmailPhone.aspx"
      title="Contact by email or phone"
      description="This is the second contact page" />
  </siteMapNode>
</siteMapNode>
</sitemap>
```

13. 运行这个网站, 看看更改是否生效。此时应该可以在 Menu 和 TreeView 中看到的新导航选项, 而新页面也可以反映在 SiteMapPath 控件中。
14. 探究一下 SiteMapDataSource 的属性对 Menu 和 TreeView 的影响。例如, 将 SiteMapDataSource.ShowStartingNode 设置为 false 会禁用根节点(一般为“主页”节点)。SiteMapDataSource.StartFromCurrentNode 能够决定从数据源层级结构的哪里开始生成

数据。

15. 探究一下 Menu 的属性对该控件的影响。例如，Menu.StaticDisplayLevels 和 MaximumDynamicDisplayLevels 属性能够决定 Menu 从 SiteMapDataSource 提取多少数据。

至此，我们不难发现，为网站添加导航功能是比较容易的。使用站点地图文件(和底层基于提供程序的架构)，如果要更新站点导航，只需在一处修改即可。

11.4 SiteMapResolve 事件的捕获

ASP.NET 具有很多扩展点，它们无处不在，而导航架构也不例外。ASP.NET 对站点地图的支持包括一个应用程序范围的事件，用户在使用通过连接到网站地图数据源的控件进行导航时，监听该事件的程序会得到通知。下面让我们通过示例了解一下如何处理这个事件。

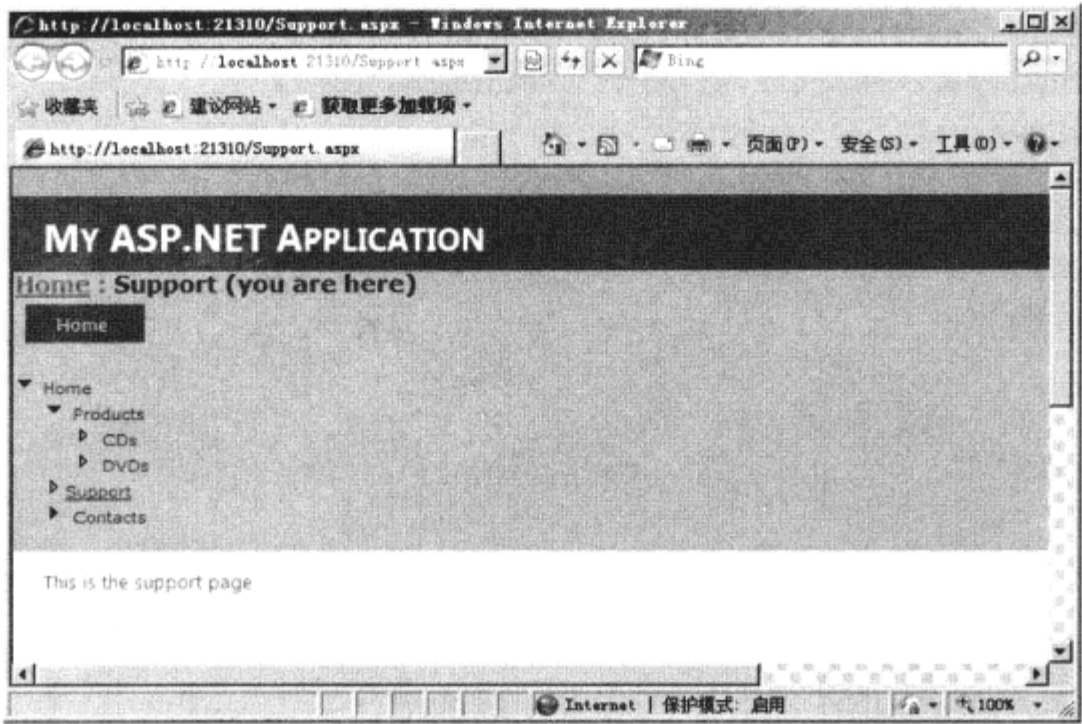
► 处理 SiteMapResolve 事件

1. 可以在项目中的任意位置添加 SiteMapResolve 处理程序。在这个示例中，我们将其添加到全局应用程序对象中。打开 Visual Studio 创建的 Global.asax。
2. 将 SiteMapResolve 事件处理程序添加到 Global.asax 文件中。这个处理程序可以做任何我们所希望做的事。这个示例将克隆通过事件参数传入的 SiteMapNode 对象。(处理程序通过克隆节点可以避免修改底层数据结构。)然后，我们通过处理程序修改节点的 Title 字段，添加一条短语：“(you are here)”。(注意，我们只能够在导航控件中看到 Title 属性的内容。)在编写完这个处理程序后，需要在 Global.asax 的 Application_Start 中将这个处理程序与 SiteMapResolve 事件关联：

```
void Application_Start(object sender, EventArgs e)
{
    SiteMap.SiteMapResolve +=
        new SiteMapResolveEventHandler(ResolveNode);
}

SiteMapNode ResolveNode(object sender,
                        SiteMapResolveEventArgs e)
{
    SiteMapNode n = e.Provider.CurrentNode.Clone();
    n.Title = n.Title + " (you are here)";
    return n;
}
```

3. 运行这个网站并导航到不同页面。在这个过程中，我们会在 SiteMapPath 控件的显示名称上发现 SiteMapNode 的变化，如下图所示。



11.5 为节点添加自定义特性

对 Web 应用程序导航机制的另一个扩展点是在 Web.sitemap 中为节点添加自定义特性 (attribute)，并在运行时获取它们。假设要为网站中的每个页面设置一张图片，则可以为站点地图的 siteMapNode 添加一个新特性并指定其值。

ASP.NET 站点地图导航功能支持为节点添加任意特性。在下面的示例中，我们要为站点地图节点添加 JPEG 图片的 URL。在每个页面加载时，母版页会通过 Image 控件显示相应图片。下面我们来看看如何添加和访问这个特性。

➤ 为站点地图添加自定义特性

1. 在项目中添加 6 个 JPEG 图片——每张图片代表一个页面。为主页、产品页面、3 个联系页面和支持页面各设计一个 JPEG 图片。修改 Web.sitemap 文件，使其中的每个 siteMapNode 元素包含一个 ImageURL 特性，如下所示。

```
<?xml version="1.0" encoding="utf-8" ?>
<sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
<siteMapNode url="~/Default.aspx" title="Home"
  description="This is the home page"
  ImageURL="~/homeimage.jpg">
  <siteMapNode url="~/Products.aspx" title="Products"
    description="This is the products page"
    ImageURL="~/productsimage.jpg"/>
  <siteMapNode url="~/Support.aspx" title="Support"
    description="This is the support page"
    ImageURL="~/supportimage.jpg"/>
  <siteMapNode url="~/Contact.aspx" title="Contacts"
    description="This is the contacts page"
    ImageURL="~/contactimage.jpg">
    <siteMapNode url="~/ContactAddress/ContactAddress.aspx"
      title="Contact using physical address"
```

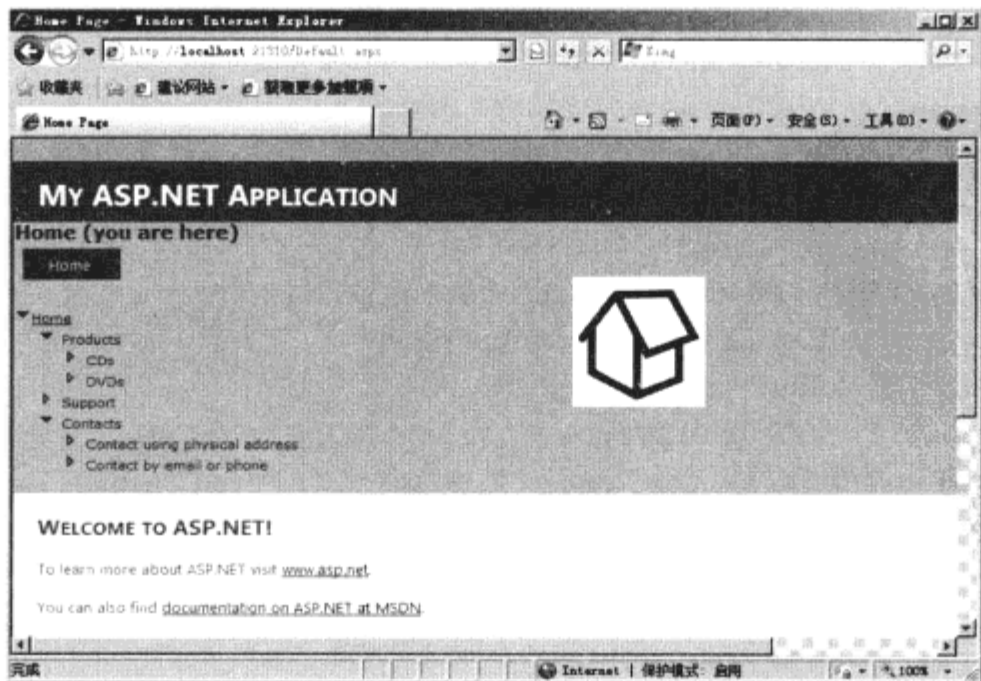
```
description="This is the first contact page"
ImageURL="~/contactPhysicalAddressimage.jpg"/>
<siteMapNode url="~/ContactEmailPhone/ContactEmailPhone.aspx"
title="Contact by email or phone"
description="This is the second contact page"
ImageURL="~/contactPhoneimage.jpg" />
</siteMapNode>

<SiteMapNode>
</siteMap>
```

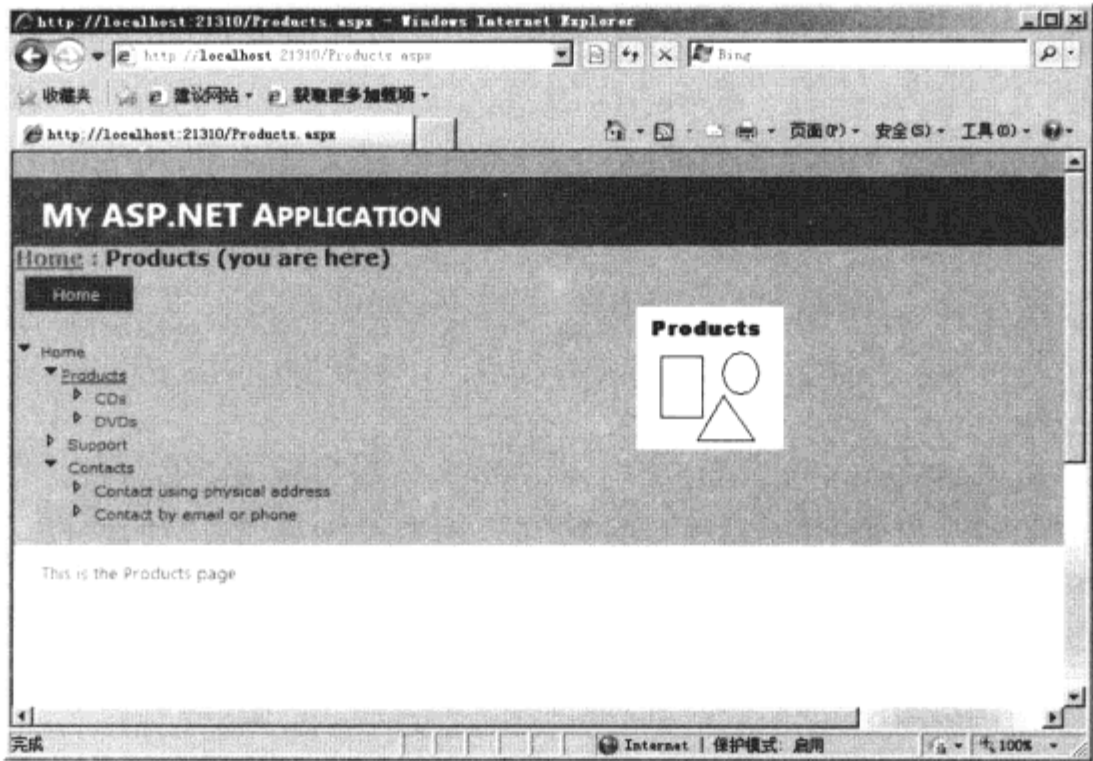
2. 访问节点时，以编程方式获取该节点的自定义特性 ImageURL。使用这个新特性的方式有多种。最简单的方式也许是在母版页中添加一个 Image 控件，并在 Page_Load 方法中将该控件的 ImageUrl 设置为从节点获取的值。

```
Public partial class SiteMaster : System.Web.UI.MasterPage
{
    protected void Page_Load(object sender, EventArgs e)
    {
        SiteMapNode current = SiteMap.CurrentNode;
        string strImageURL = current["ImageURL"];
        if (strImageURL != null)
        {
            this.Image1.ImageUrl = strImageURL;
        }
    }
}
```

3. 虽然在母版页的 Page_Load 方法中设置图像非常简单，但这并不是根据特定 SiteMapNode 的信息更新 UI 的唯一方式。例如，我们还可以处理 OnMenuItemDataBound 事件，在这个处理程序中设置自定义特性。下面两个截图展示了母版页中的图标，根据页面的不同，这个图标也会不同。



下图为产品页面。



11.6 安全性调整

ASP.NET 的导航机制能够与身份验证与授权机制结合，实现安全性调整。所谓“安全性调整”(security trimming)是指根据当前用户的角色显示菜单。当然，前提是网站必须对用户进行身份验证。(有关身份验证的内容，请阅读第 9 章。)

为使安全性调整功能生效，需要在 web.config 中打开 securityTrimmingEnabled 属性。可以通过每个 SiteMapNode 节点的特性来设置与导航有关的角色列表。

有关安全性调整的更多内容，可以阅读“ASP.NET 站点地图安全性调整”一文，网址为 <http://msdn.microsoft.com/zh-cn/library/ms178428.aspx>。

11.7 URL 映射

ASP.NET 导航架构支持 URL 映射。所谓“URL 映射”(URL mapping)是指在 web.config 文件中通过 urlMappings 元素将虚拟的(不存在的)URL 映射到实际存在的 ASPX 文件。建立 URL 映射会使 ASP.NET 将请求的 URL 交给映射的 URL 进行处理。这是在 HttpApplication 中通过 HttpContext.RewritePath 实现的。

例如，假设某网站包含一个简单的产品页面，其中同时包含 CD 和 DVD。然而，当前的 UI 模型要求为 CD 和 DVD 在菜单结构中分别添加一个单独的选项。此时便可以使用 URL 映射了。

下面的练习会将单个页面映射到两个菜单项上。在这种情况下，页面的内容由 URL 参数决定。

► 实现 URL 映射

1. 更新 Products 页面，使它能够根据 ID 参数的值来显示不同内容。本示例将产品分为



CD 和 DVD。该页面会根据 ID 参数的值(1、2 或其他值)显示不同内容。在 Products 页面中添加一个 Label 控件, 将其 ID 设置为 LabelProductType。再添加一个 ListBox 控件, 将其 ID 设置为 ListBoxProducts。在代码旁置文件的 Page_Load 处理程序中实现 URL 映射功能, 如下所示:

```
public partial class Products : System.Web.UI.Page
{
    protected void AddCDsToListBox()
    {
        this.ListBoxProducts.Items.Add("CD- Snakes and Arrows");
        this.ListBoxProducts.Items.Add("CD- A Farewell To Kings");
        this.ListBoxProducts.Items.Add("CD- Moving Pictures");
        this.ListBoxProducts.Items.Add("CD- Hemispheres");

        this.ListBoxProducts.Items.Add("CD- Permanent Waves");
        this.ListBoxProducts.Items.Add("CD- Counterparts");
        this.ListBoxProducts.Items.Add("CD- Roll the Bones");
        this.ListBoxProducts.Items.Add("CD- Fly By Night");
        this.ListBoxProducts.Items.Add("CD- 2112");
    }

    protected void AddDVDsToListBox()
    {
        this.ListBoxProducts.Items.Add("DVD- A Show Of Hands");
        this.ListBoxProducts.Items.Add("DVD- Exit Stage Left");
        this.ListBoxProducts.Items.Add("DVD- Rush In Rio");
        this.ListBoxProducts.Items.Add("DVD- R30");
    }

    protected void Page_Load(object sender, EventArgs e)
    {
        if (this.Request.Params["ID"] == "1")
        {
            this.LabelProductType.Text = "CDs";
            AddCDsToListBox();
        }
        elseif (this.Request.Params["ID"] == "2")
        {
            this.LabelProductType.Text = "DVDs";
            AddDVDsToListBox();
        }
        else
        {
            this.LabelProductType.Text = "All CDs and DVDs";
            AddCDsToListBox();
            AddDVDsToListBox();
        }
    }
}
```

2. 更新 web.sitemap 文件, 添加映射虚拟文件(如 CDs.aspx 和 DVDs.aspx)的节点。可以将以下代码添加到 web.sitemap 文件中:

```
<?xml version="1.0" encoding="utf-8" ?>
  <sitemap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
    <siteMapNode url="/Default.aspx" title="Home"
      description="This is the home page"
      ImageURL="/homeimage.jpg">
    <siteMapNode url="/Products.aspx" title="Products"
      description="This is the products page"
      ImageURL="/productsimage.jpg">
      <siteMapNode url="/CDs.aspx" title="CDs"
        description="This is the CDs page"
        ImageURL="/productsimage.jpg"/>
      <siteMapNode url="/DVDs.aspx" title="DVDs"
        description="This is the DVDs page"
        ImageURL="/productsimage.jpg"/>
    </siteMapNode>
    <siteMapNode url="/Support.aspx" title="Support"
      description="This is the support page"
      ImageURL="/supportimage.jpg"/>
    <siteMapNode url="/Contact.aspx" title="Contacts"
      description="This is the contacts page"
      ImageURL="/contactimage.jpg">
    <siteMapNode url="/ContactAddress/ContactAddress.aspx"
      title="Contact using physical address"
      description="This is the first contact page"
      ImageURL="/contactPhysicalAddressimage.jpg"/>
    <siteMapNode url="/ContactEmailPhone/ContactEmailPhone.aspx"
      title="Contact by email or phone"
      description="This is the second contact page"
      ImageURL="/contactPhoneimage.jpg" />
    </siteMapNode>
  </siteMapNode>
</sitemap>
```

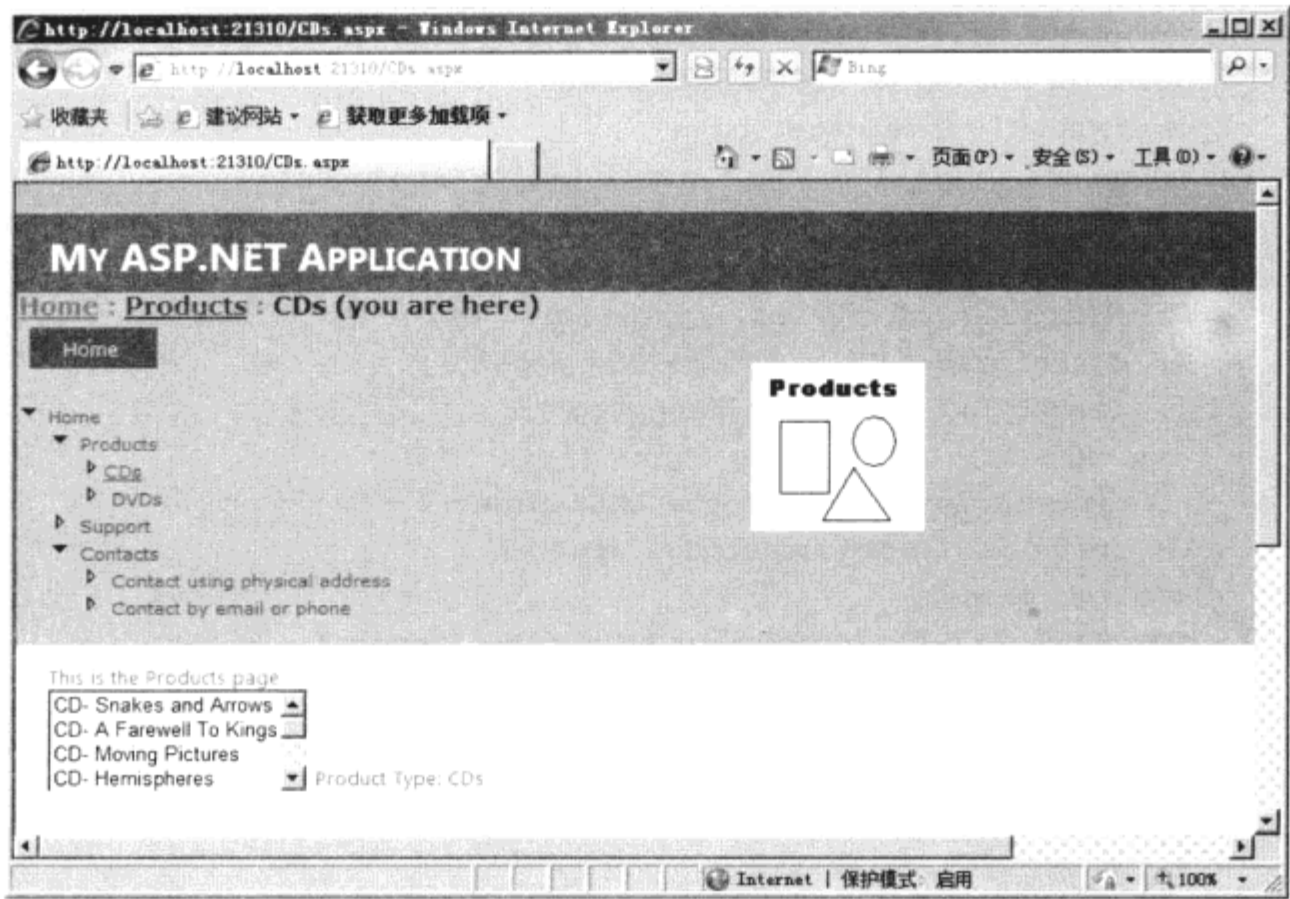
3. 在 web.config 文件中添加以下设置:

```
<configuration>
  <system.web>
    <urlMappings enabled="true">
      <add url="/CDs.aspx" mappedUrl="/Products.aspx?ID=1"/>
      <add url="/DVDs.aspx" mappedUrl="/Products.aspx?ID=2"/>
    </urlMappings>
  </system.web>
</configuration>
```

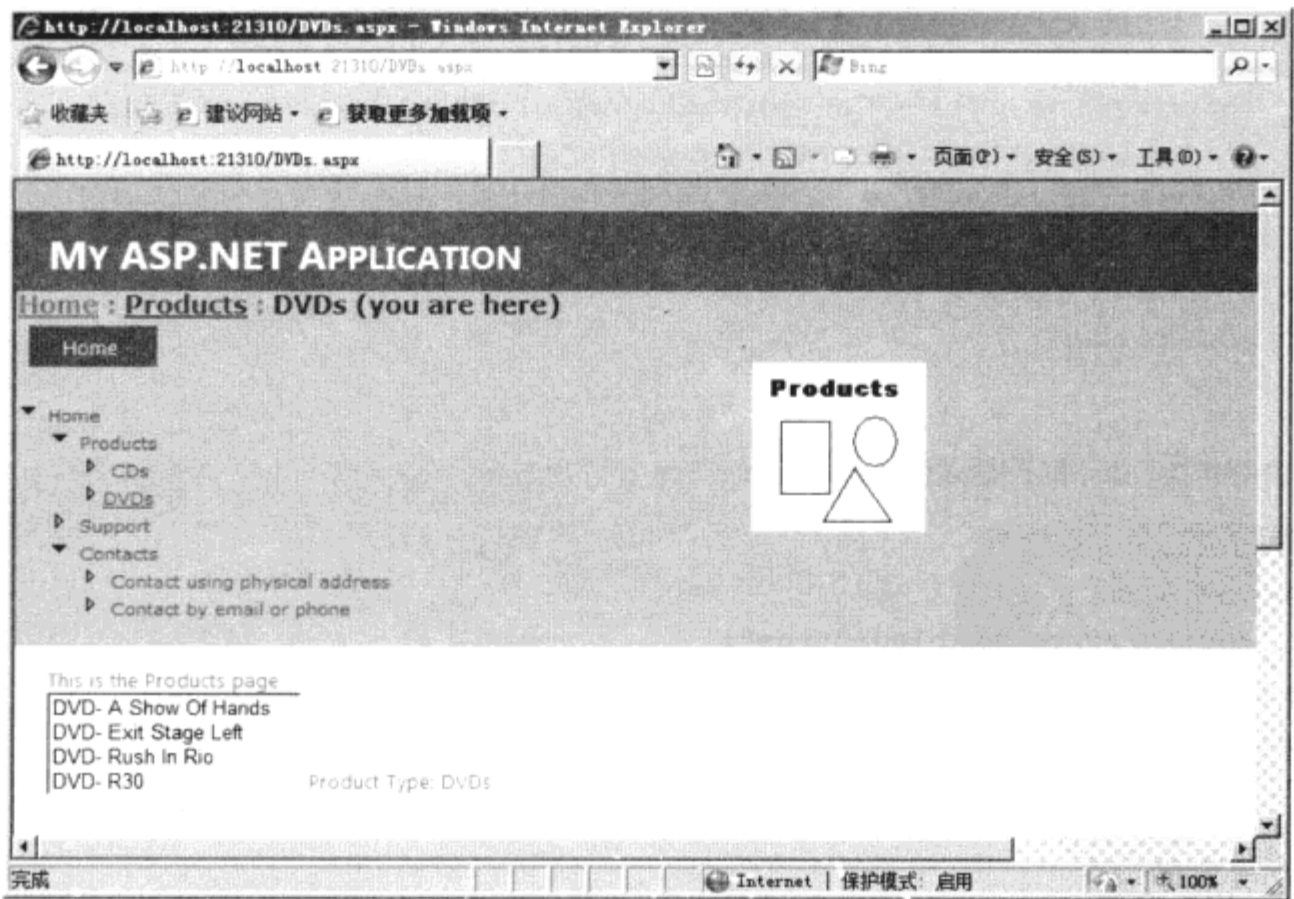
4. 运行这个网站。此时我们会发现, 刚刚添加的两个节点显示在了 Products 菜单中。站点地图将这两个节点指向 CDs.aspx 和 DVDs.aspx 文件。虽然这个 Web 应用程序中并不包含这两个文件, 但用户可以通过菜单项来查看它们。web.config 文件会将请求定

向给 Products.aspx，并传入具有特定值的 URL 参数。如果 Products.aspx 页面被加载并且 ID 参数为 1 或 2，那么页面会相应地加载包含 CD 名称或 DVD 名称的列表框。

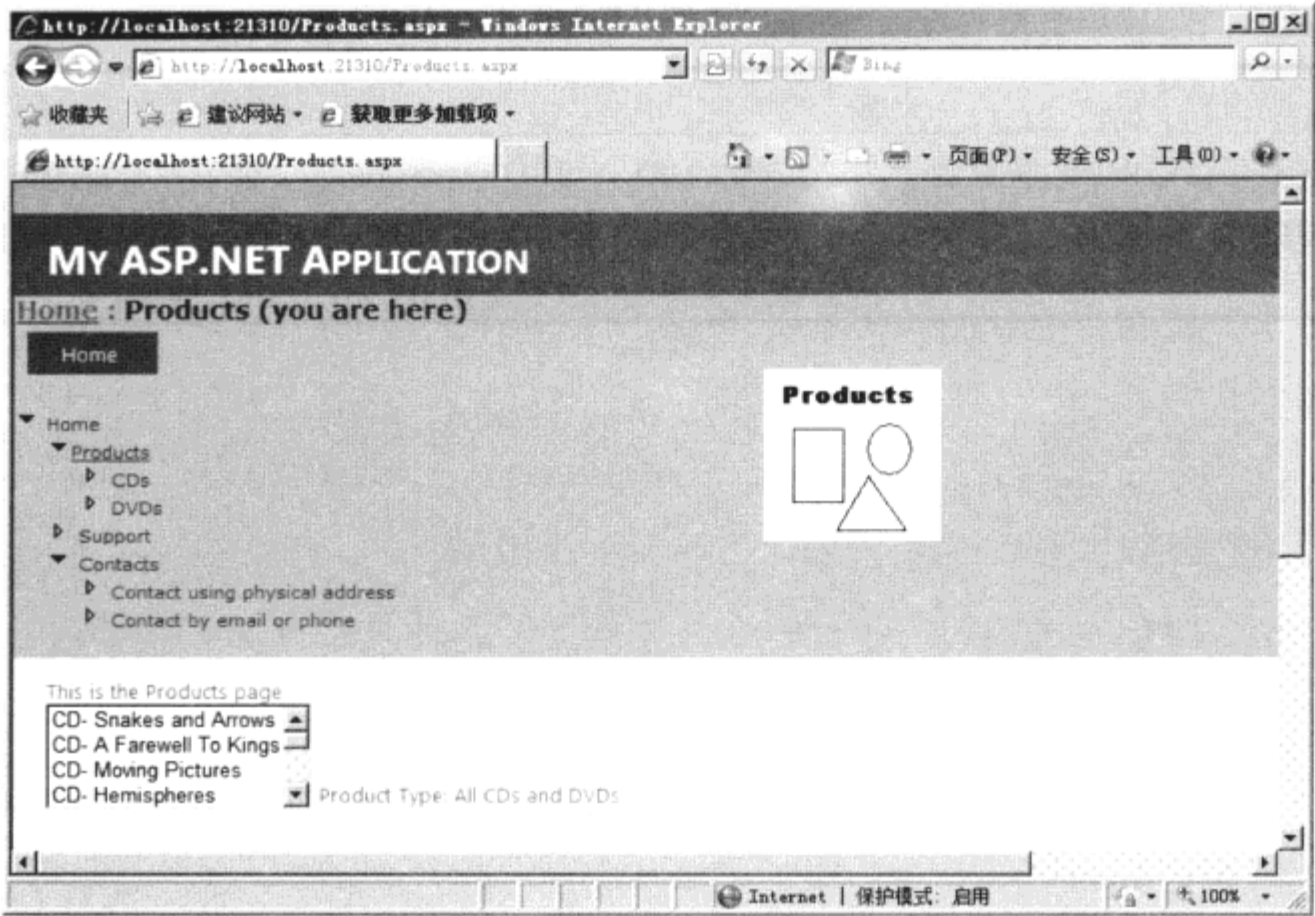
下图展示了从站点地图数据中选择 CDs 后显示的产品页面。



下图展示了从站点地图数据中选择 DVDs 后显示的产品页面。



下图展示了从站点地图数据中选择 Products 后显示的产品页面。



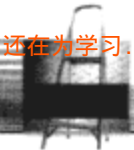
在没有物理页面支持的情况下，适合用 URL 映射为站点导航添加节点。

11.8 URL 重写

Microsoft Internet 信息服务(IIS) 7.0 支持一种“URL 重写模块”(URL Rewrite Module)，提供了当今比较前沿的 URL 重写技术。URL 重写并不仅仅是在配置文件中添加 urlMappings 那么简单，它实际上是一种动态请求重定向技术，例如，可以利用 URL 重写来基于某种运行时条件(如服务器变量或 HTTP 标头)进行重定向，也可以根据 URL 重写规则来控制重定向。

11.9 快速参考

目 标	操 作
为应用程序添加 XML 站点地图	在“解决方案资源管理器”的项目节点上右击。选择“添加” “新建项”，从模板中选择“站点地图”。该模板适合为网站添加基于 XML 的站点地图
为网站中的页面添加导航控件	打开“工具箱”中的“导航”选项卡。选择 Menu、TreeView 或 SiteMapPath 控件，并将其置于页面上 此时，Visual Studio 会通过“任务”菜单提示为控件选择数据源。如果页面上已具有可用的数据源，则选中它。如果已创建了基于 XML 的站点地图，则选择“新建数据源”，然后根据导航数据的包装方式，选择“XML 文件”或“站点地图”
监听导航请求	在 Global.asax 文件中添加 SiteMapResolve 事件的处理程序



续表

目 标	操 作
将虚拟的(不存在的)URL 映射到真实的 URL	为映射虚拟 URL，可以在 web.config 文件中添加 urlMappings 节。在站点地图数据中添加虚拟 URL，以便用户能够更方便地导航到指定页面(更好的方式是使用 IIS 7.0 的 URL 重写)

第 12 章

个 性 化

学习目标

- 使用 ASP.NET 个性化功能
- 为网站添加个性化支持

本章将介绍 ASP.NET 内建的个性化功能。ASP.NET 旨在提供一个框架，为实现大多数网站所需的功能提供支持。例如，正如第 7 章介绍的，ASP.NET 通过母版页和主题提供了一致观感的支持。第 9 章介绍了新的登录控件，这些控件在一定程度上使得开发者不必做额外的工作。此外还有身份验证与授权、站点地图等。ASP.NET 如今已包含了许多简化开发的功能。

个性化往往也是优秀网站所具有的特征之一。在 ASP.NET 2.0 之前，我们必须自行提供对网站个性化的支持，而如今，这些功能也被 ASP.NET 收入囊中。

本章将重点介绍网站的个性化。

12.1 为访客提供个性化服务

在 Internet 和 Web 刚刚开始流行时，大多数网站只支持静态内容。也就是说，这些网站只提供文本、图形和链接。而当时的网络冲浪社区只有可怜的几个懂得如何使用浏览器来浏览早期 Web 服务器内容的用户。

在出现可交互式站点后，网站除了生成内容，其余的几乎都不需要。然而，一些精明的商人发现，为不同的用户提供定制的、有针对性的内容会带来更多商机。

例如，在下一次在线购物或访问某个订阅式网站(subscription-type site)^①时，可以注意观察一下这些站点对我们了解多少。例如，在提供登录信息后，网站往往会用我们的名字向我们问好。有的网站甚至会向我们显示我们所感兴趣的信息或产品。这些都是网站应用个性化的案例。

在早期，网站的个性化要求开发者自行编写代码。例如，在 Cookie 中管理用户偏好，并在数据库中存储个人信息。除了存储和管理个人信息外，还需要将个人信息管理与所采用的身份验证和授权机制结合。也就是说，在用户通过身份验证后，可以根据用户的个人信息

① 译者注：订阅式网站是指一种页面内容可由用户所定制的网站。这种网站一般要求用户注册。

来定制页面。

ASP.NET 现在包含一些为网站提供个性化支持的服务，能够满足不同用户对个性化的需求。当然，开发者仍可以自行建立提供此功能的数据库和服务。不过，与其他服务一样，ASP.NET 也为此提供了一种一致的方法，使开发者不必自己编写所有代码。

12.2 ASP.NET 中的个性化

与身份验证和站点导航一样，ASP.NET 个性化服务同样遵循提供程序模式，但定义网站的个性化需要先定义用户配置文件。本节将从这里开始说起。

12.2.1 用户配置文件

用户配置文件(user profile)是 ASP.NET 个性化服务的核心。我们可以通过用户配置文件来定义网站中个人信息的类型。例如，网站用户的姓名、性别、登录次数等个人数据。我们还可以通过用户配置文件来存储用户的偏好设置。例如，可以在用户配置文件中添加主题信息，以便根据用户的偏好来选择页面的样式。

在 web.config 中定义了个性化属性后，ASP.NET 中的组件便能够读取和使用它们。这也正是 ASP.NET 个性化提供程序的主要任务。

12.2.2 个性化提供程序

第 8 章介绍了 .NET 中提供程序模式的应用。提供程序隐藏了支持相应服务的基础代码，并能够在对网站影响最小的前提下切换不同的底层存储媒介。例如，可以先使用 XML 文件来存储数据，而后迁移到 Microsoft SQL Server 数据库，或者，我们有希望连接到 ASP.NET 网站的遗留的数据库(如用于身份验证)。ASP.NET 个性化也不例外。事实上，ASP.NET 内建了两种个性化提供程序——针对用户数据的配置文件提供程序，以及针对 Web Parts 的个性化提供程序。(第 13 章会详细介绍 Web Parts 方面的内容。)

ASP.NET 在一个名为 PersonalizationProvider 的抽象类中定义了个性化提供程序中所涉及的基础功能。这些功能包括加载和存储个性化属性，以及管理这些属性与网站中 Web Parts 的关系。ASP.NET 通过 SqlPersonalizationProvider 类(派生自 PersonalizationProvider)提供了这些功能的默认实现。

12.3 个性化功能的使用

个性化功能的使用非常简单。我们在 web.config 中定义个性化属性。ASP.NET 会自动生成一个类，以方便开发者管理个性化设置。之后，个性化信息的使用方式与会话状态的使用

方式基本一致的。

12.3.1 在 web.config 中定义配置文件

网站的配置文件架构是在 web.config 文件中通过成对的键和类型来定义的。假设我们在设计网站时决定跟踪用户的以下信息：

- 登录次数(整型)
- 用户名(字符串)
- 性别(布尔值)
- 生日(日期)

那么以上属性可以这样在 web.config 中定义：

```
<system.web>
  <profile automaticSaveEnabled="true">
    <properties>
      <add name="NumVisits" type="System.Int32"/>
      <add name="UserName" type="System.String"/>
      <add name="Gender" type="System.Boolean"/>
      <add name="Birthday" type="System.DateTime"/>
    </properties>
  </profile>
</system.web>
```

在 web.config 文件中定义完属性后，便可以通过当前 HttpContext(或基类 Page)的 Profile 属性来使用配置文件信息。

12.3.2 配置文件信息的使用

在网站中访问配置文件与访问会话状态类似。会话状态将在第 14 章详细介绍，这里只需要知道访问特定的会话状态需要使用页面的 Session 成员。Session 成员是一种键/值对字典，能够存储任何与会话有关的信息。ASP.NET 2、3、3.5 版的编译器会根据 web.config 文件中定义的架构生成一个配置文件类^①。ASP.NET 4 并未延续这种做法，而是通过 ProfileBase 类以键/值对的形式来访问配置文件信息^②。

ProfileBase 能够表示 web.config 中定义的配置文件信息。可以像下面这样使用其 GetPropertyValue 和 SetPropertyValue 方法来访问配置文件属性：

① 译者注：这类似于桌面应用程序中 Visual Studio 自动生成的 Settings 类。

② 译者注：该类出现于 ASP.NET 2.0，提供了一种以弱类型的方式访问配置文件信息的方法。


```
Protected void Page_Load(object sender, EventArgs e)
{
    ProfileBase profile = HttpContext.Current.Profile;
    string name = (string)profile.GetPropertyValue("Name");
    if (name != null)
    {
        Response.Write("Hello " + name);
        DateTime dateTime = (DateTime)profile.GetPropertyValue("Birthday");
        Response.Write("Your birthday is "
            +dateTime);
    }
}
```

12.3.3 配置文件变更的保存

上述代码片段假定与用户关联的个性化信息已经存在。要为特定用户新增配置文件数据，只需要设置 `ProfileBase` 对象的属性即可。例如，可以为页面实现这样一个存储配置文件的处理程序：

```
protected void ProfileSaveClicked(object sender, EventArgs e)
{
    ProfileBase profile = HttpContext.Current.Profile;
    profile.SetPropertyValue("Name", this.TextBoxName.Text);

    profile.Save();
}
```

为确保个性化属性总能够被保存，最简单的方法是将 `automaticSaveEnabled` 设置为 `true`。这样，配置文件数据会通过提供程序自动保存。

在必要时，也可以使用 `ProfileBase.Save` 方法来手动保存个性化属性。除了保存和加载配置文件外，还可以通过调用 `ProfileBase.DeleteProfile` 方法来删除特定用户的配置文件。

12.3.4 配置文件与用户

用户和配置文件信息是通过用户的标识来关联的。在默认情况下，ASP.NET 会使用 `HttpContext` 中的 `User.Identity.Name` 来作为数据存储的键。也正因为此，配置文件一般只提供给通过身份验证的用户使用。

不过，ASP.NET 也支持匿名配置文件。这也是在 `web.config` 中配置的。匿名配置文件跟踪机制默认会采用 `Cookie`。然而，可以使ASP.NET采用重整的URL。重整的URL(mangled URL)是一种包含某种键的URL，这个键能够标识客户端，并随请求回发至服务器。

下面的练习演示了基于用户的登录ID来访问个性化配置文件的方法。


➤ 使用配置文件

1. 创建一个“ASP.NET Web 应用程序”项目，将其命名为 MakeItPersonal。
2. 为使个性化功能正常工作，Visual Studio 会创建一个本地数据库。^①
3. 更新 web.config 文件，在<profile>元素中添加配置文件属性。本示例的配置文件属性包含用户姓名、主题名称、生日和地址。下面的代码展示了如何使用 group 元素来实现声明中配置文件结构的分组和嵌套。Visual Studio 会在 web.config 文件中添加<profile>节。配置信息应在<properties>开始和结束标签之间添加。

```
<system.web>

  <profile>
    <providers>
      <clear/>
      <add name="AspNetSqlProfileProvider"... />
    </providers>
    <properties>
      <add name="Theme" type="System.String"/>
      <add name="Name" type="System.String"/>
      <add name="Birthdate" type="System.DateTime"/>
      <group name="Address">
        <add name="StreetAddress" type="System.String"/>
        <add name="City" type="System.String"/>
        <add name="State" type="System.String"/>
        <add name="ZipCode" type="System.String"/>
      </group>
    </properties>
  </profile>

</system.web>
```

 **提示** 这个示例将用户的名称作为查找个性化信息的键。为此，用户要通过身份验证。不过，ASP.NET 也支持匿名(anonymous)个性化。也就是说，ASP.NET 能够为匿名用户提供个性化信息——通过 Cookie 来跟踪用户。为启用匿名个性化跟踪功能，需要将 anonymousIdentification 元素的 enable 属性设置为 true，并像这样设置 Cookie 参数：^②

```
<anonymousIdentification enabled="true"
  cookieName=".ASPXANONYMOUSUSER"
  cookieTimeout="120000"
  cookiePath="/"
  cookieRequireSSL="false"
  cookieSlidingExpiration="true"
```

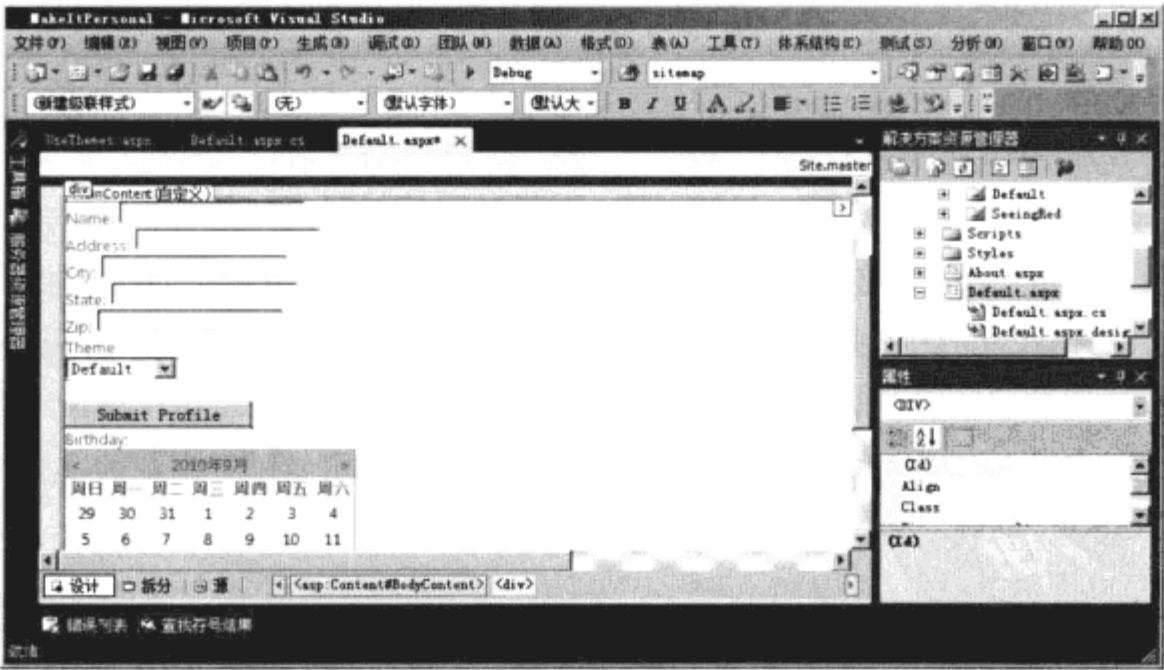
① 译者注：如果 Visual Studio 未创建这个数据库，可以按第 9 章的方法创建一个。

② 译者注：这个元素在 Web.config 文件中的路径为 configuration\system.web\anonymousIdentification。

```
cookieProtection="Encryption"
cookieless="UseDeviceProfile" />
```

除了在 web.config 中进行匿名访问设置外，也需要设置配置文件中必要属性的 allowAnonymous 特性。
在按上述方式完成 web.config 中的设置，并在配置信息中添加 allowAnonymous 特性后，ASP.NET 便会根据用户首次访问网站时生成的 Cookie 来存储个性化信息。

- 4. 从 MasterPageSite 项目(第 7 章)复制 Default 和 SeeingRed 主题到当前项目中。为此，先在应用程序的 App_Themes 目录下添加 Default 和 SeeingRed 文件夹。然后，在每个主题的文件夹上右击，选择“添加”|“现有项”。通过“添加现有项”对话框导航到第 7 章的项目，选择相应的主题文件。
- 5. 从 MasterPageSite 项目复制 UseThemes.aspx 和对应的.cs 文件到当前项目中。如果将它们置于单独的文件夹下(如 Secured 文件夹)，则可以单独管理这个文件夹的访问权限。
- 6. 更新 Default.aspx 页面，以便用户录入配置文件信息。
分别为姓名、地址、城市、省份和邮编创建一个文本框。
添加一个用于选择主题的下拉列表框，添加 Default 和 SeeingRed 两项。
再添加一个用于选取生日的日历(Calendar)控件。
- 7. 添加一个用于提交配置文件信息的按钮。双击这个按钮，为其添加处理程序。我们将通过这个处理程序来将用户输入的值保存到配置文件中。
下图展示了此时的页面。



提示 这个示例将用户的名称作为查找个性化信息的键。为此，用户要通过身份验证。使用 ASP.NET 网站配置工具为应用程序启用“Forms 身份验证”(详见第 8 章)。至少添加一个用户，以便可以为其提供个性化服务。自动生成的网站在 Accounts 文件夹下会有一个名为 Login.aspx 的登录页面。修改网站的访问规则，强制用户通过该页面进行身份验证，以便我们能观察个性化信息的存储和获取。

8. 更新 Page_Load，如果存在配置信息，则将其显示出来。为此需获取配置文件对象，并设置每个文本框和日历控件，如下所示：

```
using System.Web.Profile;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!this.IsPostBack)
        {
            ProfileBase profile = HttpContext.Current.Profile;
            string theme = (string)profile.GetPropertyValue("Theme");
            this.TextBoxName.Text =
                (string)profile.GetPropertyValue("Name");
            this.TextBoxAddress.Text = (string)profile.GetPropertyValue
                ("Address.StreetAddress");
            this.TextBoxCity.Text =
                (string)profile.GetPropertyValue("Address.City");
            this.TextBoxState.Text =
                (string)profile.GetPropertyValue("Address.State");
            this.TextBoxZipCode.Text =
                (string)profile.GetPropertyValue("Address.ZipCode");
            this.DropDownList1.SelectedValue =
                (string)profile.GetPropertyValue("Theme");
            this.Calendar1.SelectedDate =
                (DateTime)profile.GetPropertyValue("Birthdate");
        }
    }
}
```

9. 更新按钮的处理程序，使其保存配置文件信息：

```
public partial class _Default : System.Web.UI.Page
{
    //...
    protected void ButtonSubmitProfile_Click(object sender, EventArgs e)
    {
        if (this.User.Identity.IsAuthenticated)
        {
            ProfileBase profile = HttpContext.Current.Profile;
            profile.SetPropertyValue("Theme", "SeeingRed");
            profile.SetPropertyValue("Name", this.TextBoxName.Text);
            profile.SetPropertyValue("Address.StreetAddress",
                this.TextBoxAddress.Text);
            profile.SetPropertyValue("Address.City", this.TextBoxCity.Text);
            profile.SetPropertyValue("Address.State", this.TextBoxState.Text);
            profile.SetPropertyValue("Address.ZipCode", this.TextBoxZipCode.Text);
            profile.SetPropertyValue("Theme", this.DropDownList1.SelectedValue);
            profile.SetPropertyValue("Birthdate", this.Calendar1.SelectedDate);
            profile.Save();
        }
    }
}
```

```
    }  
    }  
}
```

10. 最后，更新 UseThemes.aspx 文件以应用相应主题。重写页面的 OnPreInit 方法，让代码根据配置文件中的设置来切换主题：

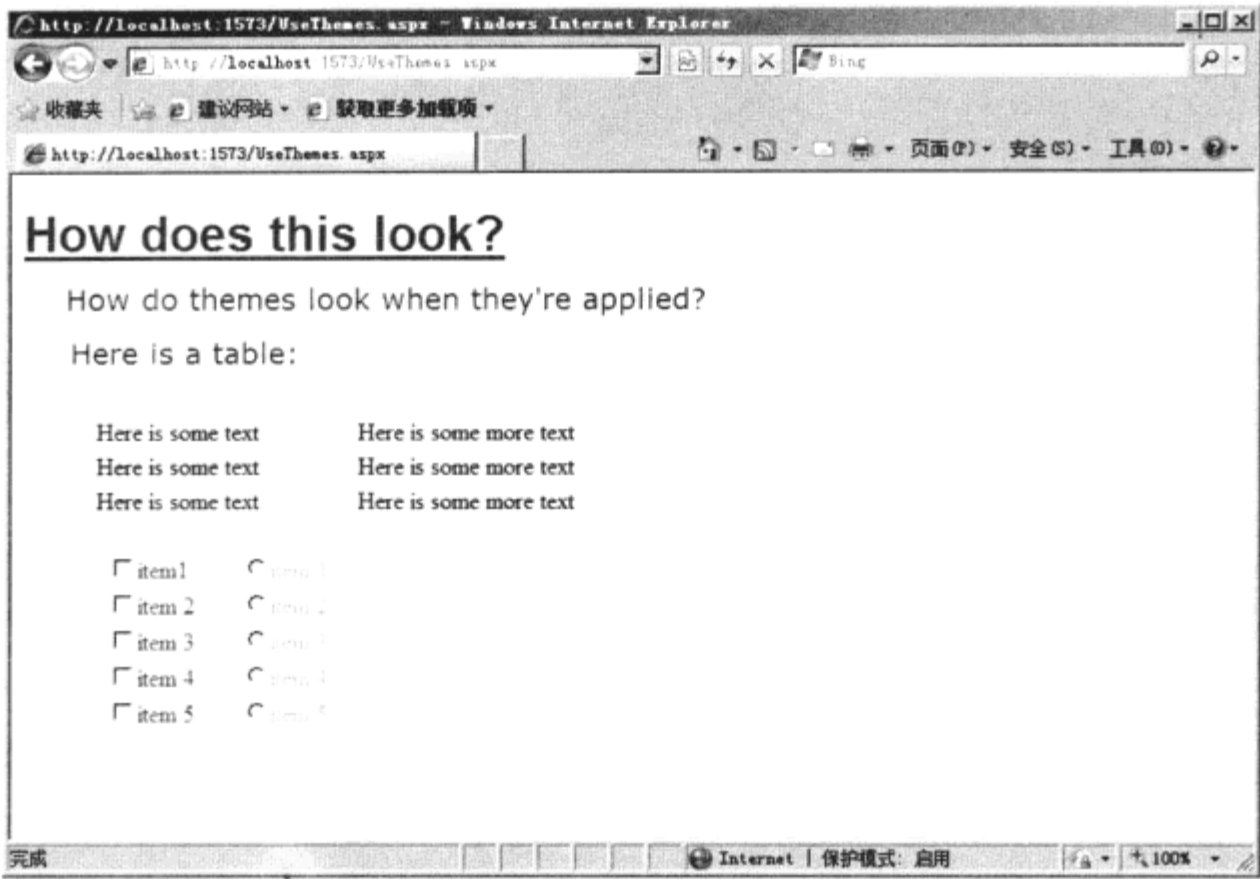
```
protected override void OnPreInit(EventArgs e)  
{  
    ProfileBase profile = HttpContext.Current.Profile;  
    if (profile != null)  
    {  
        String strTheme = (string)profile.GetPropertyValue("Theme");  
        if (strTheme != null &&  
            strTheme.Length> 0)  
        {  
            this.Theme = strTheme;  
        }  
    }  
    base.OnPreInit(e);  
}
```

11. 在 Default.aspx 页面中添加一个 Hyperlink 控件。将其 Text 属性设置为“Go to Themes Page”，使其 NavigateURL 属性指向 UseThemes.aspx 页面，以便在用户输入配置文件信息并提交后能够查看主题的变化。

在用户首次访问网站时，网站会自动添加配置文件。下图展示了浏览器中包含配置信息的默认页面。



12. 在访问 UseThemes.aspx 页面时，该页面会采用用户在配置文件中设置的主题。下图展示了根据配置文件呈现 SeeingRed 主题的 UseThemes.aspx 页面。



12.4 快速参考

目 标	操 作
定义个性化配置文件的设置	在 web.config 中通过<profile>来定义用于建立配置文件架构的名称和类型对
访问配置文件的属性	通过当前的 HttpContext 来获取配置文件对象，并使用 GetPropertyValue 和 SetPropertyValue 方法进行访问
通过 Cookie 跟踪匿名配置文件	在 web.config 中启用 anonymousIdentification，并为必要的配置文件属性添加 allowAnonymous 特性

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 13 章

Web 部件

学习目标

- 理解 ASP.NET “Web 部件”
- 在网页中使用标准 “Web 部件”
- 创建自定义的 “Web 部件”
- 在网页中使用自定义的 “Web 部件”

前几章讨论了自定义控件、复合控件和 ASP.NET 内建的控件。由于 ASP.NET “Web 窗体”的呈现由小粒度的、可管理的组件负责，因而通过添加新控件来扩展这个框架是非常容易的。服务器端控件能够以较小的粒度来控制 HTML 的呈现。

本章将介绍 “Web 部件”。“Web 部件”(Web Parts)将网站的交互方式带到了另一个层面，是一个较大的话题(甚至可以写成一本书)。“Web 部件”在许多方面都与自定义控件类似，它们允许对页面输出的 HTML 进行自定义，而不必进行硬编码。

自定义控件派生自 System.Web.UI.Control 或 System.Web.UI.WebControl，而 “Web 部件”则派生自 System.Web.UI.WebControls.WebParts.WebPart。虽然 WebPart 也继承于 System.Web.UI.Control，但它已超越了一般控件的功能，可以与 WebPartPage 和 WebPartZone 类相结合，从而支持用户在页面上添加、删除、自定义、连接和个性化 “Web 部件”。

或许 ASP.NET 服务器端控件和 “Web 部件”之间最大的不同是后者提供了一种允许 “最终用户”按个人喜好配置网站的手段。相反，ASP.NET 服务器端控件针对的则是 ASP.NET 开发者。使用 “Web 部件”，初级开发者也能够轻松构建可交互的页面；使用 “Web 部件”，网站用户也可以在某种程度上灵活地管理网站视图。

为更好地理解 “Web 部件”，不妨回想近几年热起来的几个社交网站(如 Windows Live Spaces)。虽然网站的主要结构由服务器掌控，但最终用户创建自己的帐户后，能够在很大程度上定制个人页面所呈现的内容，如能够添加好友和同事，以及集成其他站点的内容。

除了创建可由最终用户自定义的网站外，“Web 部件”对初级 Web 开发者也很有帮助。“Web 部件”能够使自定义控件支持 “拖放”(drag-and-drop)功能。开发者能够将一个完整的 “Web 部件”从 “Web 部件”库(目录)拖放至 “Web 部件”区域(zone)，还能够修改一组 “Web 部件”的公共属性并使它们固定不变。除了包装用户界面(UI)组件外，“Web 部件”还能够通过标准接口彼此互连。

“Web 部件”技术在构建门户和协作网站方面非常有用。Microsoft SharePoint 就是这类网

站最典型的例子之一。SharePoint 能够通过高级组件(“Web 部件”)来实现文档的协作和共享，而不是将这些功能内建到应用程序中。建立一个门户网站只不过是应用程序中组装这些高级组件而已。

13.1 “Web 部件”简史

在 21 世纪的最初几年，SharePoint 成为了组织构建门户网站和协作环境的有力工具。例如，如果要为一个庞大的团队制定共同的目标，此时便可以创建一个门户网站。像软件开发这样的团队，其工作往往需要系统具备版本控制和缺陷跟踪这样的功能。如果团队成员在地理位置上分散，或者有些员工不在办公网络中，则需要通过 Web 来共享信息。

如果没有 SharePoint 这样的框架，开发者可能要为建立这种工具而重新“造轮子”。SharePoint 引入了一些预定义的组件，极大地降低了构建协作网站的难度(不需要从头构建)。SharePoint 中的页面都基于“Web 部件”类型的组件。“Web 部件”能够包装信息和功能。

SharePoint 是一个单独的框架，旨在构建协作门户网站，而如今的 ASP.NET 是一种用途广泛的 Web 开发框架，并恰好内建了一套门户框架。也就是说，SharePoint 专门用于构建门户，而 ASP.NET 包含一些构建门户网站的类。虽然两者的开发环境不同，但两者都涉及一个概念——“Web 部件”。虽然 ASP.NET “Web 部件”和 SharePoint “Web 部件”并不是完全相同的东西，但二者的行为是类似的。

13.2 “Web 部件”的优点

“Web 部件”控件可以用来实现门户网站。工作流和协作管理如今已成为网站主要应用方向之一。由于不同门户之间具有很多共有的功能，因而可以通过框架来构建，而不必从头做起。这些功能包括文件传输、用户配置文件、用户管理等。

ASP.NET “Web 部件”的开发主要面向以下 3 种场景：

- 开发“Web 部件”控件
- 构建采用“Web 部件”控件的常规页面
- 实现门户网站中的“Web 部件”页面和“Web 部件”

13.3 “Web 部件”控件的开发

所有“Web 部件”控件都可以看作是现有 ASP.NET 服务器端控件(包括自定义控件、用户控件和复合控件)的一个超集。为能够在特定环境中以编程方式控制“Web 部件”控件，开发者还可以通过从 `System.Web.UI.WebControls.WebParts.WebPart` 类派生来创建自定义的“Web 部件”控件。

13.3.1 “Web 部件”页面的开发

一般的网页就能够使用“Web 部件”。Visual Studio 支持创建包含 WebPart 控件的页面。开发 WebPart 页面需要为页面添加 WebPartManager 和若干区域，并为这些区域添加 WebPart 控件。

13.3.2 “Web 部件”应用程序的开发

最后，我们可以通过 WebPart 控件来开发完整的应用程序。例如，假设要构建一个门户。如果使用 WebPart 控件，则可以实现可自定义的个性化页面。“Web 部件”还适合构建常用的应用程序(如共享记录或文档的应用程序)，并可以将其打包发布，然后批量部署到其他公司的网站上。

13.4 “Web 部件”的架构

“Web 部件”的架构承担着多方面任务。“Web 部件”的职责是充当更高层次的 UI 元素，所以功能组件被分解为页面管理和区域管理。不同 WebPart 控件需要协调一致。此外，页面中的每个功能区域通常需要作为一组控件来处理(如为了管理布局)。

让我们通过对象关系来理解“Web 部件”(如图 13.1 所示)。页面中的“Web 部件”要嵌在区域中，而这些区域由能够与应用程序数据存储交互的一个 WebPartManager(或 ProxyPartManager)来管理。

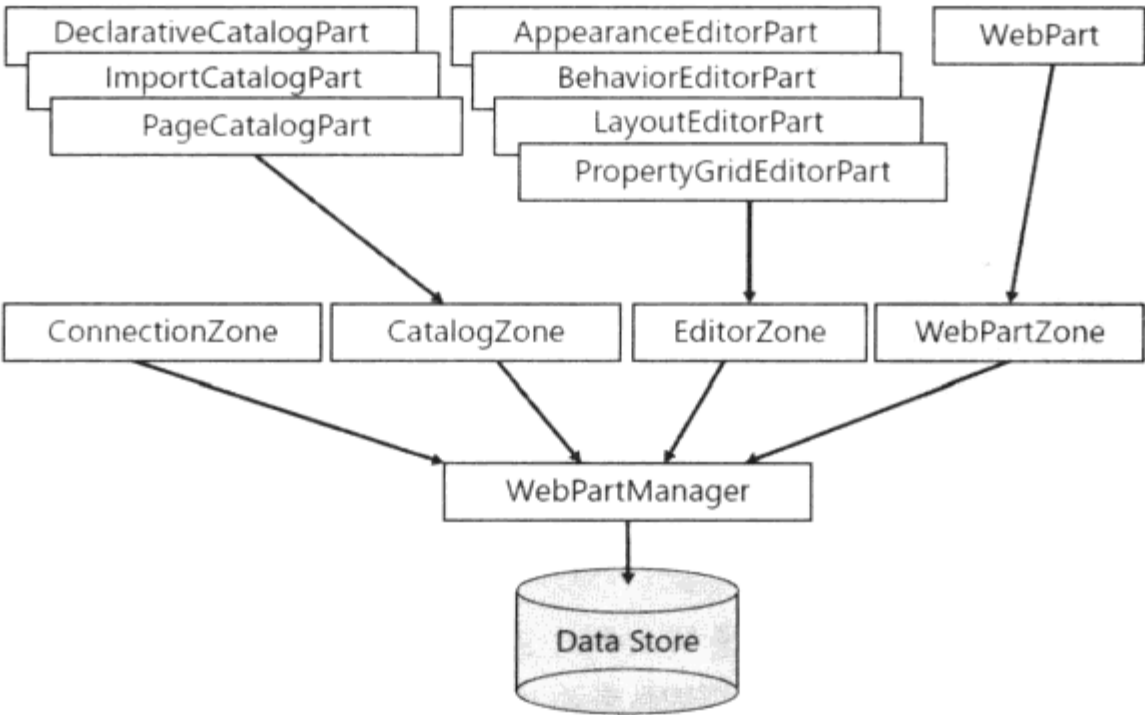


图 13.1 “Web 部件”由区域管理，而区域由 WebPartManager 管理

13.4.1 WebPartManager 与 WebPartZone

如图 13.1 所示，“Web 部件”由区域管理，而区域由 WebPartManager 管理。只要页面使用了 WebPart，不论多少，就都要添加 WebPartManager。WebPartManager 负责管理和协调区域和区域中的控件。WebPartZone 能够管理自身控件集合中的 UI 元素。

在区域中，ZoneTemplate 用于容纳“Web 部件”。如果 ZoneTemplate 中有常规的 ASP.NET 控件，那么 ASP.NET 会按 WebPart 的形式来包装它。

13.4.2 内建的区域

“Web 部件”区域用于管理一组控件的布局。ASP.NET 中内建了以下 4 种区域：

- **WebPartZone** WebPartZone 类包含了管理区域中服务器端控件的基本功能。这种区域能够容纳常规的服务器端控件和 WebPart 控件。常规的控件在运行时会被由 GenericWebPart 控件包装，以使它们具有 WebPart 的行为。
- **CatalogZone** 这种区域(目录区域)用于承载 CatalogPart 控件。“目录”一般用于管理页面中部件的可见性。CatalogZone 控件能够根据显示模式来显示和隐藏其内容。之所以称其为“目录”，是因为它充当了最终用户可以选择的控件目录。
- **EditorZone** 该控件允许最终用户根据个人偏好修改和个性化网页。对网站进行个性化包括设置个人信息(如生日、性别特定的称呼、网站的访问次数等)、颜色搭配和布局……EditorZone 旨在实现这方面的特性，它也能够保存和加载相关的设置，以便用户在下次登录时能够继续使用。
- **ConnectionZone** “Web 部件”之间往往需要动态连接和通信。ConnectionZone 正是为此而设立的。

13.4.3 内建的“Web 部件”


除了区域外，ASP.NET 还提了几种 WebPart 控件。这些 WebPart 控件按功能分类，有些用于管理目录(catalog)，有些用于管理编辑。不同的 WebPart 由不同区域承载。下面列出了“工具箱”中提供的 WebPart：

- **DeclarativeCatalogPart** 在构建 WebPart 页面时，可以动态添加或声明部件。向页面动态添加部件是指在运行时通过代码来添加部件。例如，假设有一个名为 MyWebPart 的“Web 部件”(派生自 System.Web.UI.Controls.WebParts.WebPart)。我们可以创建这个部件的实例，并使用 WebPartManager.AddWebPart 方法将其添加到 WebPartManager

中。以声明方式添加部件是指在包含 WebPart 的 ASPX 文件中添加标签声明。DeclarativeCatalogPart 控件能够管理以声明方式添加到部件目录中的服务器端控件。

- **PageCatalogPart** 最终用户可能会通过打开和关闭控件来自定义网站。PageCatalogPart 代表了页面目录, 其中包含之前添加到页面中, 但现在已被关闭的控件。使用 PageCatalogPart, 最终用户能够将这些控件重新放回到页面中。
- **ImportCatalogPart** 使用 ImportCatalogPart, 用户可以从 XML 数据导入“Web 部件”的描述。
- **AppearanceEditorPart** 该控件用于编辑与某个 WebPart 或 GenericWebPart 关联的外观属性。
- **BehaviorEditorPart** 该控件实现了对 WebPart 或 GenericWebPart 行为的编辑。
- **LayoutEditorPart** 该控件用于编辑与某个 WebPart 或 GenericWebPart 关联的布局属性。
- **PropertyGridEditorPart** 该控件支持用户编辑 WebPart 的自定义属性(而其他 EditorPart 控件只支持对 WebPart 类现有属性进行编辑)。

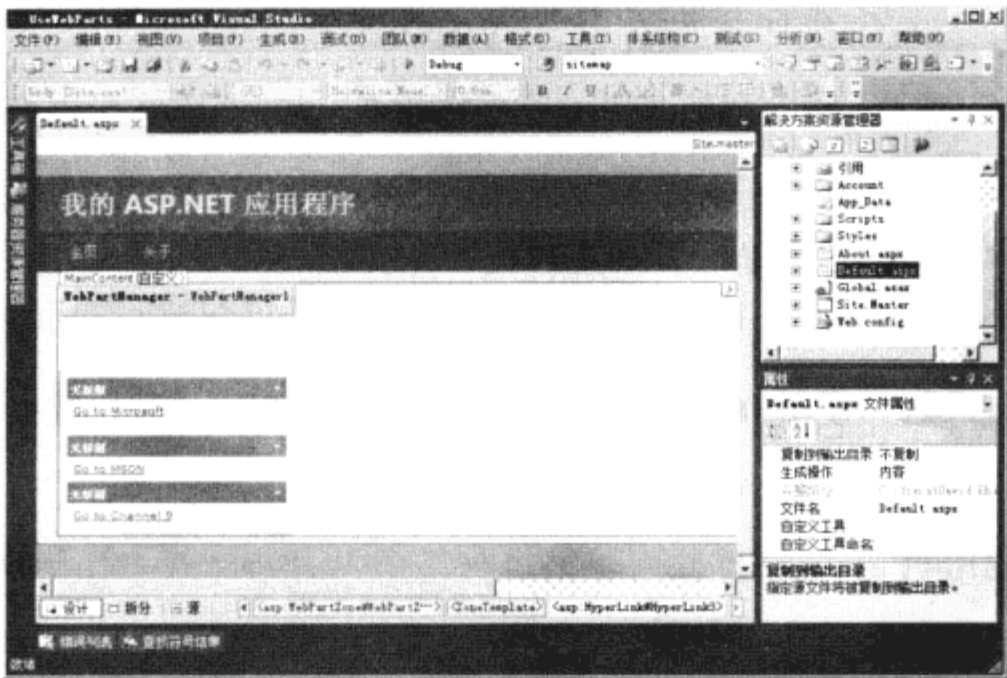
为更好地理解 WebPart 控件的使用, 请完成以下练习。它演示了如何通过 WebPart 控件来构建网页。

 **提示** 使用“Web 部件”必须启用个性化。也就是说, 要建立支持个性化的数据库(详见第 12 章)。此外, “Web 部件”要求对用户进行身份验证, 因而应确保网站至少具有一个用户(可以使用 ASP.NET 网站配置工具来添加)。

➤ 使用“Web 部件”

1. 在 Visual Studio 中创建一个“ASP.NET Web 应用程序”, 将其命名为 UseWebParts。
2. 从“工具箱”拖放一个 WebPartManager 实例到默认页面上。
3. 拖放一个 WebPartZone 到页面上。将其 ID 设置为 WebPartZoneLinks, 将 HeaderText 属性设置为 Try These Links, 将 HeaderStyle 属性组中的 ForeColor 属性设置为 Blue(以便编辑模式下可以更清晰地看到这个标题)。通过该控件的“自动套用格式”编辑器(在“设计器”中选择该控件, 单击其右上角的小箭头), 将其样式设置为“专业型”。
4. 在 WebPartZone 中添加几个 HyperLink 控件(如下图所示)。链接可以是任意的(这里只是为了演示功能)。

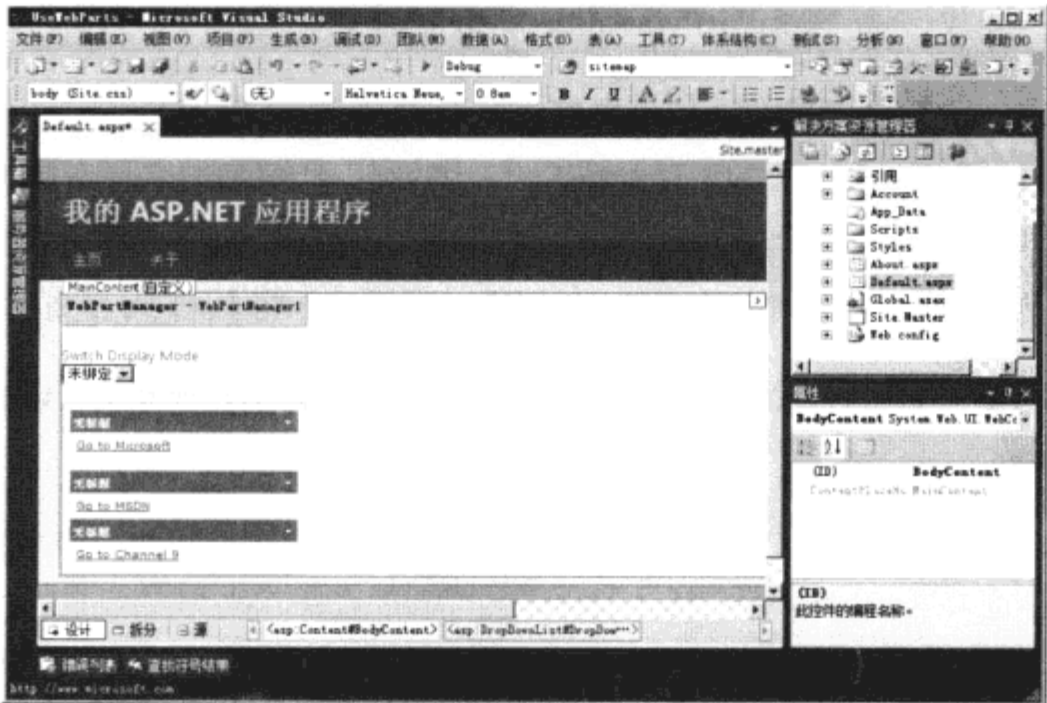
ASP.NET 4 从入门到精通



5. 运行这个页面。此时，页面的左部会显示刚刚添加的链接(见下图)。



6. 如下图所示，为页面添加一个 Label 控件。将 Text 设置为 Switch Display Mode(切换显示模式)。再为页面添加一个 DropDownList 控件。将其命名为 DropDownListDisplayModes，将 AutoPostBack 属性设置为 true。下面将通过这个下拉列表来切换显示模式。



ASP.NET “Web 部件”支持 5 种显示模式。我们将在下一步中添加其中的几个显示模式：

- **BrowseDisplayMode** 这是一般的显示模式。在这种模式下不能够进行个性化或编辑。
- **DesignDisplayMode** 这种模式支持以拖放方式对布局进行个性化设置。
- **EditDisplayMode** 这种模式支持 WebPart 属性的个性化和自定义，并允许用户动态删除已添加的“Web 部件”。
- **ConnectDisplayMode** 这种模式允许用户在运行时连接“Web 部件”。
- **CatalogDisplayMode** 这种模式允许用户在运行时向 WebPartZone 添加“Web 部件”。

7. 更新_Default 类以支持模式切换。Visual Studio 会为页面添加一个 WebPartManager 实例。在 Page_Load 方法中，对 WebPartManager1 的成员进行遍历，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!this.IsPostBack)
    {
        foreach (WebPartDisplayMode mode in
            this.WebPartManager1.SupportedDisplayModes)
        {
            String modeName = mode.Name;
            // Make sure a mode is enabled before adding it.
            if (mode.IsEnabled(this.WebPartManager1))
            {
                ListItem item = new ListItem(modeName, modeName);
                DropDownListDisplayModes.Items.Add(item);
            }
        }
    }
}
```

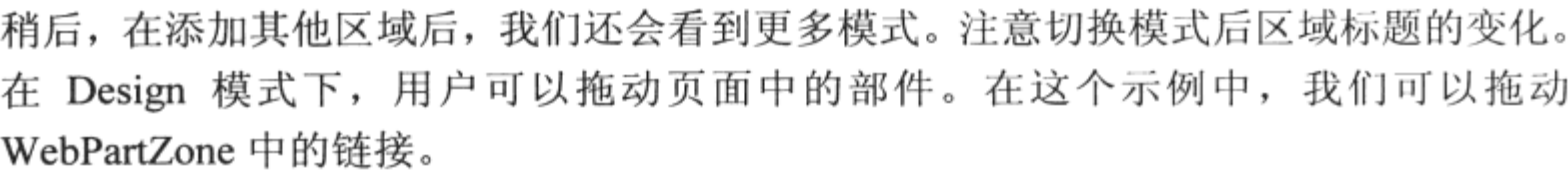
Page_Load 中的代码会对当前的 WebPartManager 进行查询，获取所有受支持的模式，并将启用的模式添加到 DropDownListDisplayModes 下拉列表中。

8. 为 DropDownListDisplayModes 下拉列表的 SelectedIndexChanged 事件添加一个处理程序。在这个处理程序中，将 WebPart 页面切换至选定的模式，如下所示：

```
protected void
DropDownListDisplayModes_SelectedIndexChanged(
    object sender, EventArgs e)
{
    string selectedMode = DropDownListDisplayModes.SelectedValue;
    WebPartDisplayMode mode =
        this.WebPartManager1.SupportedDisplayModes[selectedMode];
    if (mode != null)
    {

```

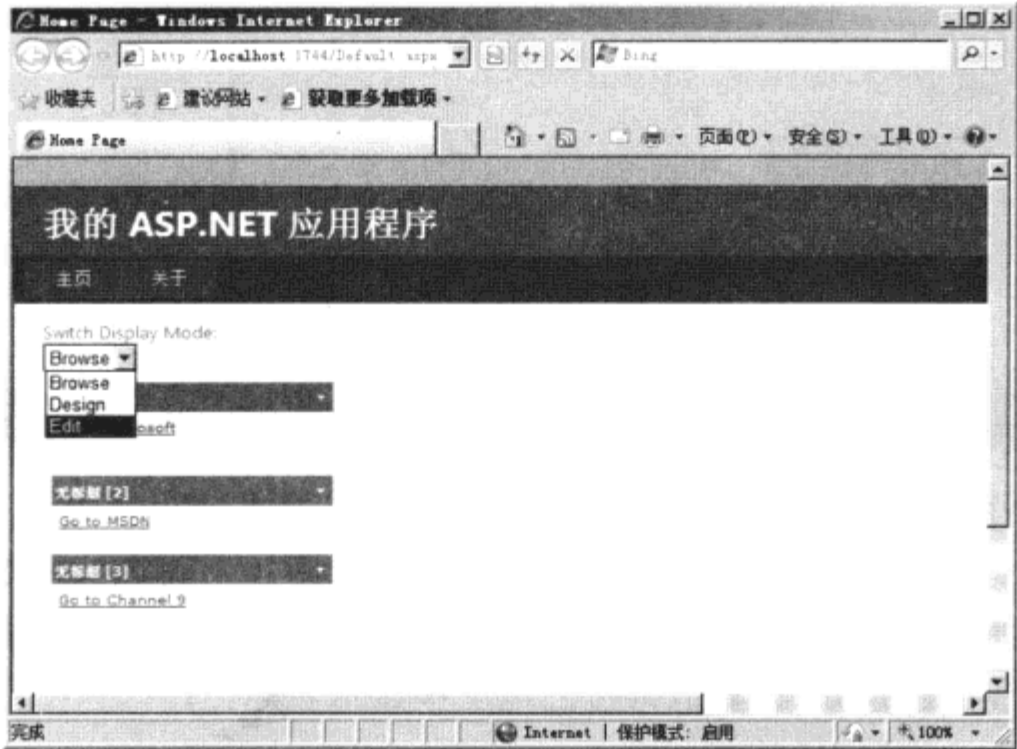
- 运行这个网站。确保至少有一个注册用户(可以使用“项目”菜单上的“ASP.NET 配置”来添加), 并且已登录。
- 如下图所示, 用户可以选择 Browse 模式和 Design 模式。



11. 现在加入更多功能。为页面添加一个 `EditorZone`。在这个 `EditorZone` 中，添加 `AppearanceEditorPart`，如下图所示(“设计器”的默认布局会按顺序安排每个组件——在这个示例中，`EditorZone` 区域采用的是绝对布局方式，因而可以将其放在任意位置)。



12. 运行这个网站，以注册用户的身份登录。此时，“Switch Display Mode” 下拉列表中会多出一个新选项——“Edit” (见下图)。

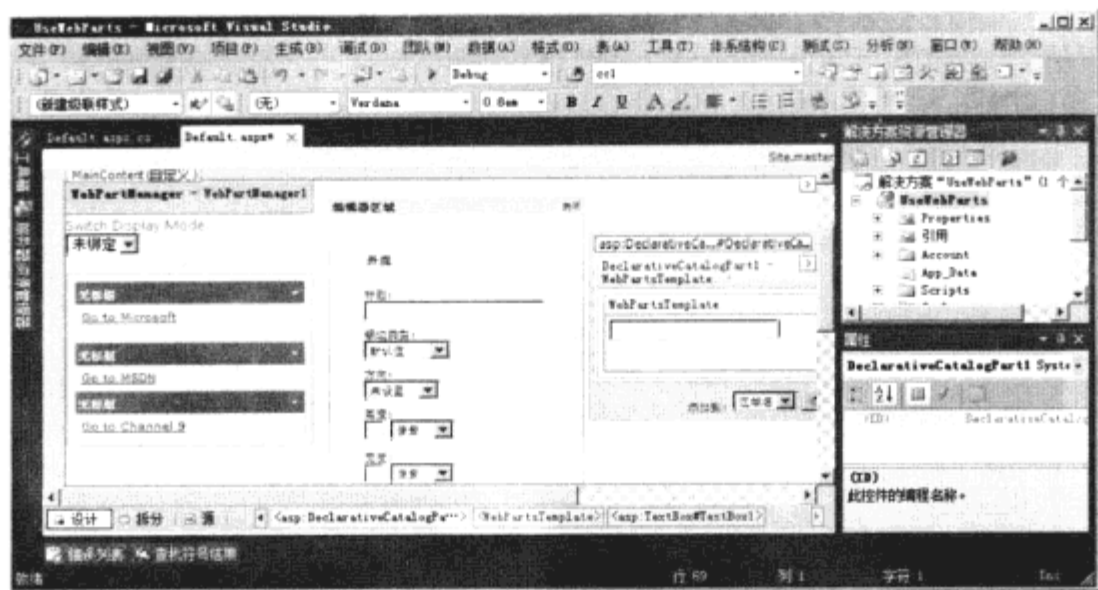


13. 回到 Visual Studio，添加一个 CatalogZone。拖放一个 DeclarativeCatalogPart 到这个新的区域，然后在“DeclarativeCatalogPart 任务”菜单中选择“编辑模板” (见下图)。

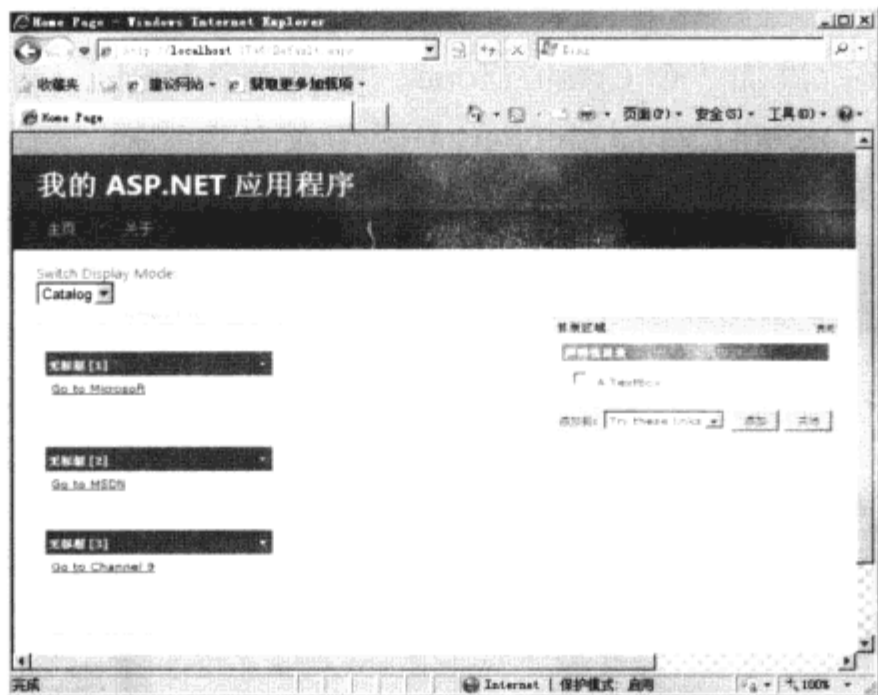


14. 在“模板编辑”模式下，从“工具箱”拖放一个 TextBox 到 DeclarativeCatalogPart 中(见下图)。然后，在页面的“源”视图中，为这个文本框添加一个 Title 属性，如下所示：

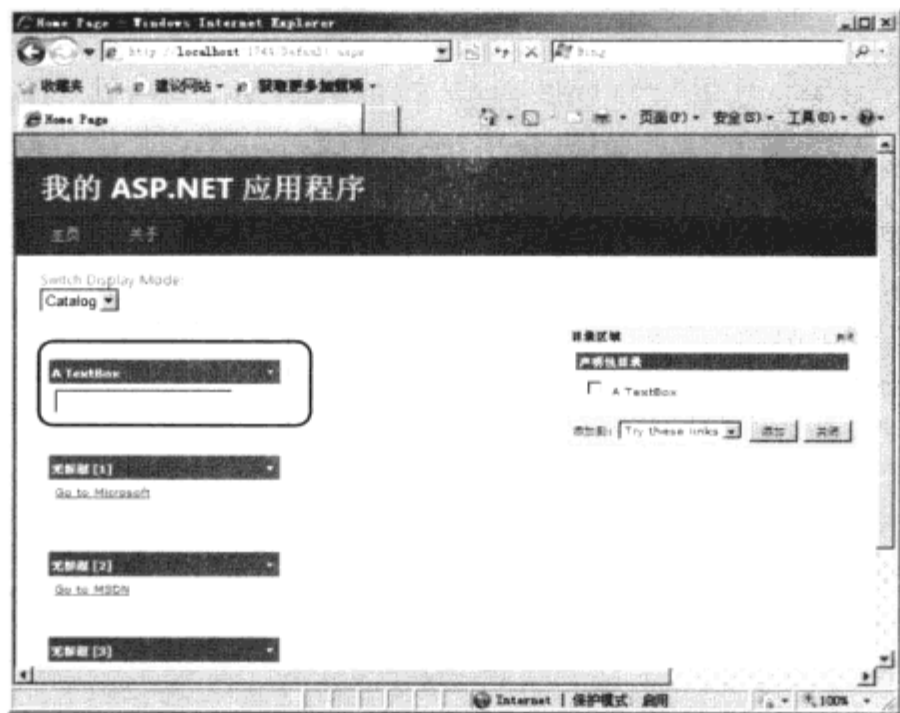
```
<ZoneTemplate>
  <asp:DeclarativeCatalogPart ID="DeclarativeCatalogPart1" runat="server">
    <WebPartsTemplate>
      <asp:TextBox ID="TextBox1"
        Title="A TextBox"
        runat="server">
      </asp:TextBox>
    </WebPartsTemplate>
  </asp:DeclarativeCatalogPart>
</ZoneTemplate>
```



15. 再次运行这个页面。切换到“Catalog”模式。选中“A TextBox”复选框，然后单击“添加”，将一个文本框添加到“Try These Links”区域中。（仅添加一个文本框意义似乎并不大。在后面的练习中，我们会编写一个“超链接 Web 部件”，并将其从目录添加到链接页面上，然后动态设置该链接的地址和显示名称。）



通过目录添加了一个 TextBox 后，页面效果如下图所示。



16. 运行这个页面，切换到编辑(Edit)模式下。在“Try these links”区域打开其中某个“Web 部件”的菜单。(可以单击“Web 部件”右上角的箭头来打开这个上下文菜单。)选择“编辑”。此时，“编辑器区域”中会显示一组用于编辑该“Web 部件”的控件，如下图所示。



这个示例演示了如何在页面中添加“Web 部件”区域，以及如何使普通的 ASP.NET 服务器端控件(如 HyperLink 控件)具有“Web 部件”的行为。下一节将介绍如何开发真正的“Web 部件”。

13.5 “Web 部件”的开发

上一节的示例演示了如何在页面中使用“Web 部件”，以及如何在运行时切换显示模式。该页面中的目录包含一个可以添加到 WebPartZone 中的 TextBox 控件。这个示例体现了“Web 部件”的灵活性和强大功能。然而，只向 WebPartZone 添加 TextBox 并没有太大意义。在下面的示例中，我们将构建一个“超链接 Web 部件”，并将其添加到 WebPartZone 中。

开发“Web 部件”实际上非常简单，与自定义控件的开发十分类似(详见第 4 章和第 5 章)。“Web 部件”并不继承于 System.Web.UI.Controls.WebControl 或 System.Web.UI.Controls.CompositeControl，而继承于 System.Web.UI.WebControls.WebParts.WebPart。从这一点看，我们可以呈现 HTML，也可以使用 ASP.NET 控件来构建“Web 部件”。WebPart 类包含许多与“Web 部件”架构集成所涉及的功能。例如，在下面的示例中，“超链接 Web 部件”的导航 URL 和显示名称属性都以属性形式暴露，最终用户可以通过 PropertyGridEditorPart 进行修改。

下面的练习将演示如何创建这个“超链接 Web 部件”并将其添加到 WebPartZone 中。虽然

可以添加常规的 HyperLink 控件，但这种控件不允许用户修改链接。例如，在之前的示例中编辑 HyperLink 控件时，只能修改部件共有的属性。为给 Web 应用程序的用户提供更多可修改的属性，需要以“Web 部件”的形式实现这个控件。

► 开发 HyperLinkWebPart

1. 为 UseWebParts 解决方案添加一个类库项目，将其命名为 WebPartLib。^①将默认添加的类文件重命名为 HyperLinkWebPart.cs。
2. 使这个新项目引用 System.Web 程序集。为此，在“解决方案资源管理器”中右键单击 WebPartLib 节点，选择“添加引用”，在“.NET”选项卡中选择这个程序集。
3. 使这个类继承于 System.Web.UI.WebControls.WebParts.WebPart，如下所示：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace WebPartLib
{
    public class HyperLinkWebPart : WebPart
    {

    }
}
```

4. 为 HyperLinkWebPart 类添加两个字符串类型的成员变量——一个用作“Web 部件”的显示名称，另一个用作实际的 URL。用合理的值对其进行初始化：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace WebPartLib
{

    public class HyperLinkWebPart : WebPart
```

^① 译者注：这里根据 Visual Studio 2010 RTM 版对原书步骤作稍事修改。

```

    {

        string _strURL = "http://www.microsoft.com";
        string _strDisplayName = "This is a link";
    }
}

```

5. 为这个类添加一个类型为 `HyperLink` 的字段。这个“Web 部件”能够利用 `HyperLink` 控件现有的功能。重写 `CreateChildControls` 方法，创建一个 `HyperLink` 实例，并将其添加到 `HyperLinkWebPart` 的控件集合中。分别用表示显示名称和 URL 的成员变量对 `HyperLink.Text` 属性和 `HyperLink.NavigateUrl` 属性进行初始化：

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace WebPartLib
{

    public class HyperLinkWebPart : WebPart
    {

        HyperLink _hyperLink;

        string _strURL = "http://www.microsoft.com";
        string _strDisplayName = "This is a link";
        protected override void CreateChildControls()
        {
            _hyperLink = new HyperLink();
            _hyperLink.NavigateUrl = this._strURL;

            _hyperLink.Text = this._strDisplayName;
            this.Controls.Add(_hyperLink);
            base.CreateChildControls();
        }
    }
}

```

6. 最后，以属性的形式暴露显示名称和 URL，以便“Web 部件”架构能够发现并使用这两个信息。为使暴露的属性能够通过稍后添加的 `PropertyGridEditorPart` 与“Web 部件”架构配合，应为这些属性添加 `Personalizable`、`WebBrowsable` 和 `WebDisplayName`(特性)，如下所示：

```

using System;
using System.Collections.Generic;

```

```
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace WebPartLib
{

    public class HyperLinkWebPart : WebPart
        System.Web.UI.WebControls.WebParts.WebPart
    {

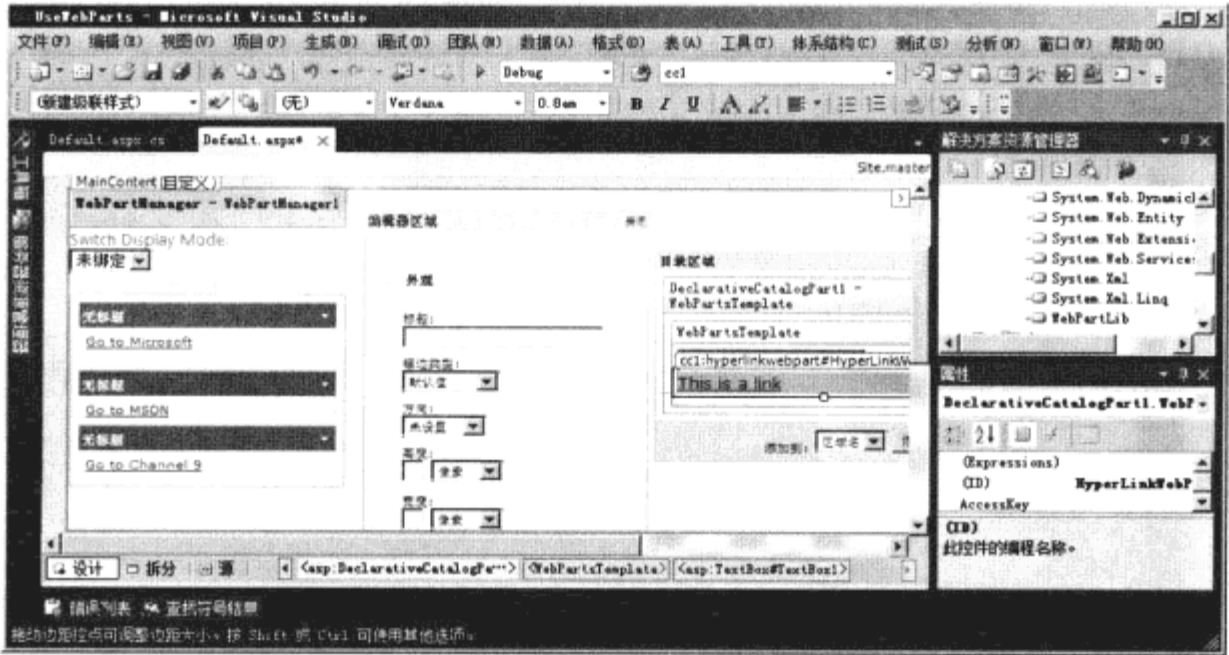
        HyperLink _hyperLink;

        string _strURL = "http://www.microsoft.com";
        string _strDisplayName = "This is a link";
        [Personalizable(), WebBrowsable, WebDisplayName("Display Name")]
        public string DisplayName
        {
            get
            {
                return this._strDisplayName;
            }
            set
            {
                this._strDisplayName = value;
                if (_hyperLink != null)
                {
                    _hyperLink.Text = this.DisplayName;
                }
            }
        }
        [Personalizable(), WebBrowsable, WebDisplayName("URL")]
        public string URL
        {
            get
            {
                return this._strURL;
            }
            set
            {
                this._strURL = value;
                if (_hyperLink != null)
                {
                    _hyperLink.NavigateUrl = this.URL;
                }
            }
        }
    }
}
```



```
protected override void CreateChildControls()
{
    _hyperLink = new HyperLink();
    _hyperLink.NavigateUrl = this._strURL;
    _hyperLink.Text = this._strDisplayName;
    this.Controls.Add(_hyperLink);
    base.CreateChildControls();
}
}
```

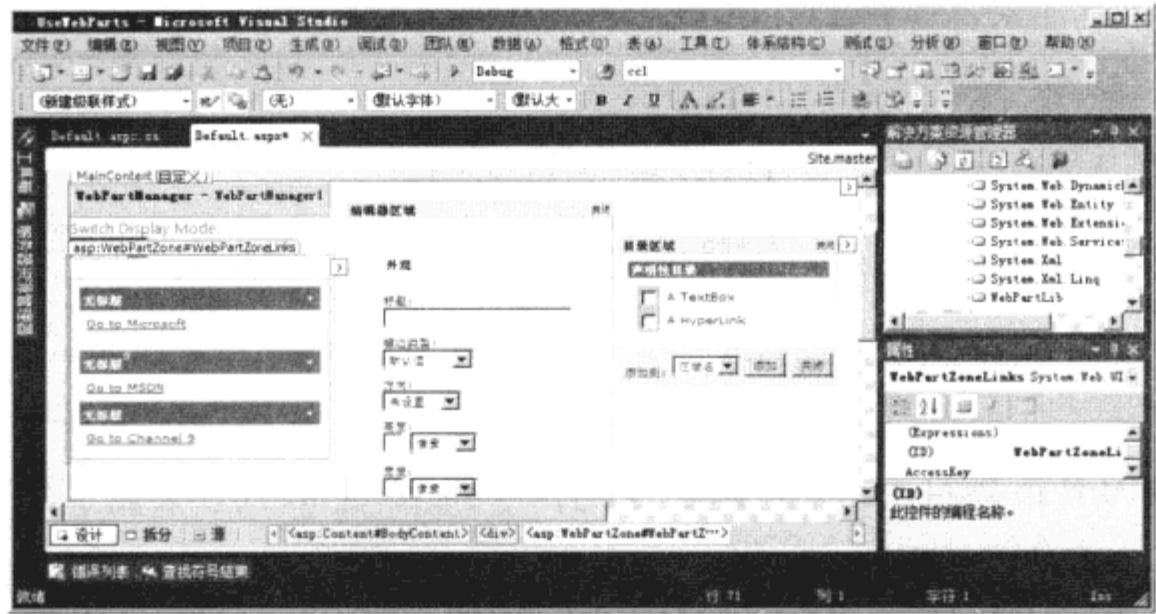
- 7. 编译 WebPartLib 项目。此时，新建的 HyperLinkWebPart 控件会显示在“工具箱”中。
- 8. 现在将 HyperLinkWebPart 添加到目录中。单击 CatalogZone 右上角的小箭头，将该控件切换到“编辑模板”模式。从“工具箱”拖放一个 HyperLinkWebPart 实例到目录中，这与前面添加 TextBox 的方法类似，如下图所示。



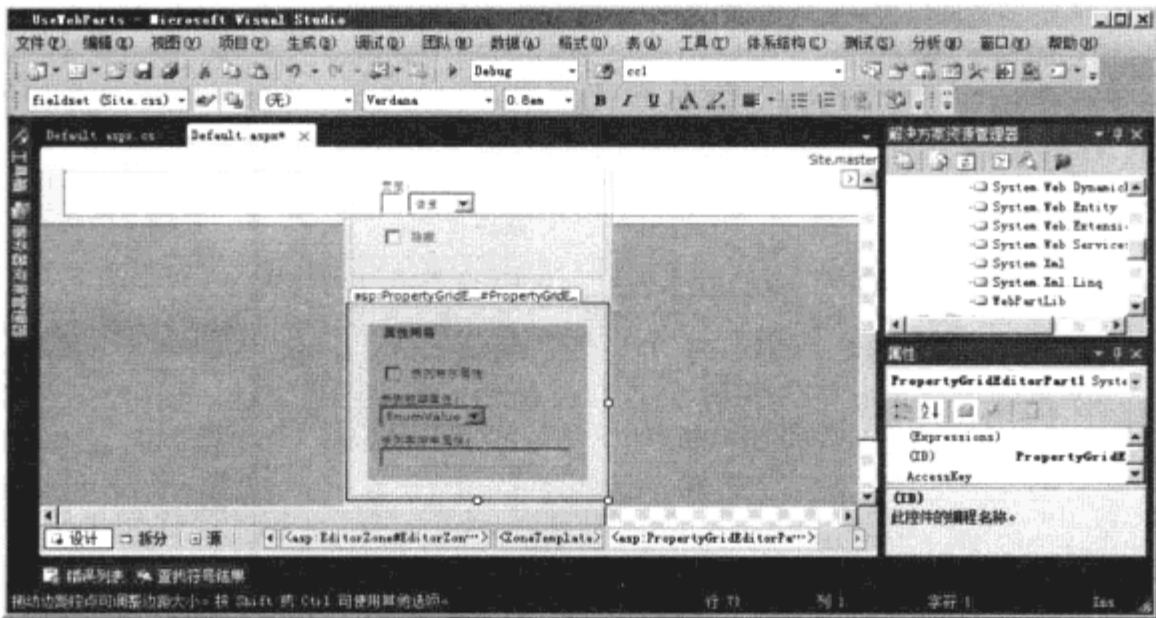
- 9. 为这个新目录项添加一个标题。将 Visual Studio 中的窗口切换到“源”视图。在标记中，添加一个属性：

```
<ZoneTemplate>
  <asp:DeclarativeCatalogPart ID="DeclarativeCatalogPart1" runat="server">
    <WebPartsTemplate>
      <asp:TextBox ID="TextBox1"
        Title="A HyperLink"
        runat="server">
      </asp:TextBox>
      <ccl:HyperLinkWebPart
        Title="A HyperLink"
        ID="HyperLinkWebPart1"
        runat="server"/>
    </WebPartsTemplate>
  </asp:DeclarativeCatalogPart>
</ZoneTemplate>
```

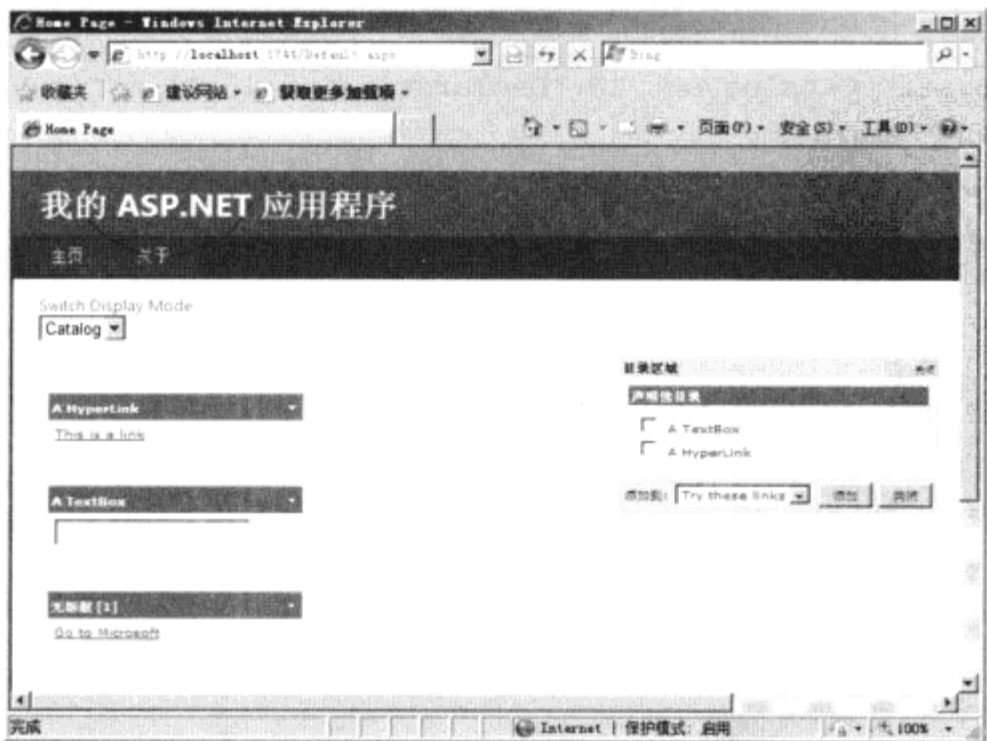
此时，目录中的 HyperLinkWebPart 应该会显示一个标题，如下图所示。



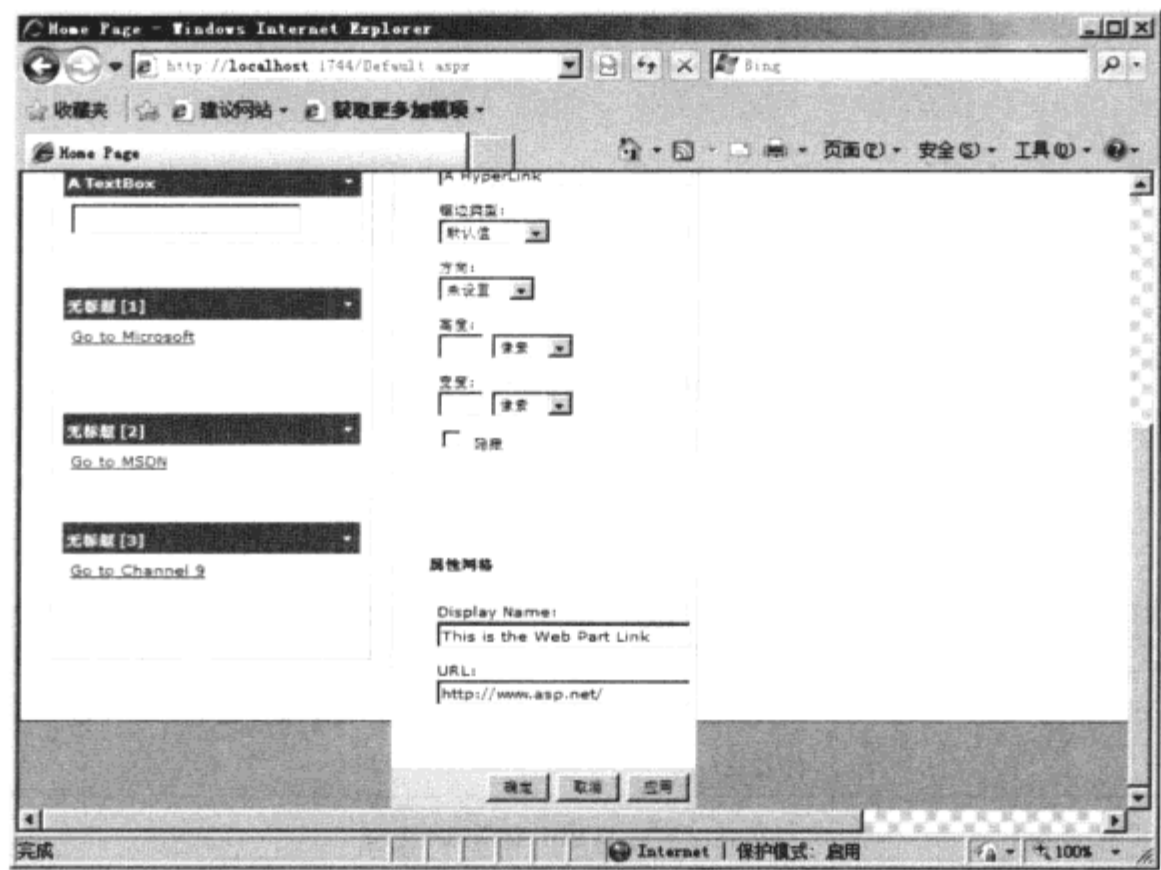
10. 在页面的 EditorZone 中添加一个 PropertyGridEditorPart。为此，只需要从“工具箱”拖放一个相应实例到 EditorZone 中即可，如下图所示。



11. 运行这个页面。如下图所示，从 Switch Display Mode 下拉列表选择 Catalog，将页面切换到目录模式。



- 12. 在“目录区域”中选择“A Hyper Link”，将其添加到“链接 Web 部件”区域。
- 13. 在“Switch Display Mode”下拉列表中选择“Edit”，将页面切换到编辑模式。
- 14. 单击新添加的链接右上角的箭头，选择“编辑”。此时会显示“编辑器区域”，其中包含新添加的“属性网格”(见下图)，我们可以通过这个“Web 部件”来编辑链接的 DisplayName 和 URL 属性。文本框中最初显示的是 DisplayName 和 URL 的默认值(可以输入其他值)。



- 15. 输入一个新的显示名称和 URL。(本示例使链接指向了 www.asp.net。)单击“确定”。此时，相应的 HyperLinkWebPart 会反映新的属性(见下图)。



如下图所示，单击这个链接会导航到相应的网站。



13.6 快速参考

目 标	操 作
使网站支持“Web 部件”	针对应用程序的数据库运行 aspnet_regsql，以确保启用了配置文件和角色
使页面支持 WebPart 控件	为页面添加 WebPartManager
使“Web 部件”页面支持编辑功能	为页面添加 EditorZone
添加一个能使服务器端控件接受“Web 部件”架构管理的区域	为页面添加 WebPartZone
允许用户动态地从控件集合中添加控件	为页面添加 CatalogZone，在“编辑模板”模式下添加希望在目录中出现的控件
创建“Web 部件”	从 System.Web.UI.WebControls.WebParts.WebPart 类派生子类，然后选择以下任意一种方法来呈现内容： <ul style="list-style-type: none">在“Web 部件”的 Render 方法中呈现 HTML创建 ASP.NET 控件，然后将其添加到“Web 部件”的 Controls 集合中，以实现自动呈现



第Ⅲ部分

状态管理与缓存

- ▶ 第 14 章 会话状态
- ▶ 第 15 章 应用程序数据的缓存
- ▶ 第 16 章 输出缓存

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569



第 14 章

会话状态

学习目标

- 理解会话状态的管理对于应用程序的重要性
- 使用会话状态管理程序(Session 对象)
- 配置会话状态
- 了解 ASP.NET 会话状态的几种存储形式

本章将介绍如何管理 Web 应用程序的会话状态(session state)。Web 应用程序的开发者应意识到应用程序的状态在任何时候的分布情况。^①会话状态(与特定会话关联的状态)是 Web 应用程序中最重要的状态之一。由于 Web 应用程序是分布式的，而 HTTP 协议是无状态的，因而 Web 应用程序不得不对客户端进行跟踪。

ASP.NET 的会话状态是可扩展、可靠且灵活的——相比其他平台(如传统的 ASP)具有诸多优势。最初，ASP.NET 会话状态由 Session 对象负责处理，系统会为每个新的会话自动创建一个对象字典(如果启用会话状态)。Session 对象可以通过 HttpContext 对象轻松访问，可以在处理请求期间随时引用它。ASP.NET 会自动将用户状态与会话关联。在需要访问会话状态时，只需要从上下文获取(也可以通过页面的成员获取)。我们可以选择 ASP.NET 跟踪会话状态的方式，甚至指定会话状态的存储位置。

本章首先会介绍 ASP.NET 管理的各种状态，以及会话状态管理程序的作用。

注意 安装本书的示例代码要求用户具有计算机的 Administrator 权限。如果使用家用计算机，则可能已具有 Administrator 权限。如果使用某个组织的计算机，并且没有 Administrator 权限，则需要咨询计算机支持人员或 IT 人员。相关内容，请参考本书“前言”部分“示例代码”一节。

14.1 何为会话状态

在经过前几章对 ASP.NET 使用的学习后，不妨整理一下思路。基于 Web 的编程是一种特殊的编程方式，需要开发者对服务于多区域用户的应用程序进行管理。此外，Web 应用程

^① 译者注：所谓“状态”是应用程序在运行时所使用的各类数据的统称。对于 Web 应用程序，“状态”可能分布客户端、Web 服务器、数据库服务器、文件服务器、网络间、应用服务器等位置。

序基于的是一种无连接的(无状态的)协议。

假设要编写一个购物门户,那么其中的某些应用程序数据(如库存信息和供货商列表)一般会保存在中央数据库中。

`System.Web.UI.Page` 和服务器端控件都能够管理视图状态。然而,在设计购物中数据的存储方案时,则会发现这些介质并不适用。

这种数据不应存储在页面的 `ViewState` 中。虽然简单的应用程序可以这样做,但如果在视图状态中存储大量数据,则会极大地降低网站用户的体验(网站的速度会变慢),而且每个请求中都包含这些数据也会暴露一些安全性问题。此外,只有可序列化的类型才能存储在视图状态中最后在重定向到其他页面后,视图状态便会丢失。

不幸的是,用户的会话状态也并不存储在应用程序的数据库中。如果应用程序只由一个用户使用,那么或许可以这样做。然而,请不要忘记,Web 应用程序旨在为尽可能多的客户端提供服务。显然,我们需要一小片空间来存储用户会话期间的数据,而这种数据就是会话状态(session state)。

14.2 ASP.NET 与会话状态

ASP.NET 自问世以来就支持会话状态。在启用会话状态的情况下,ASP.NET 会为每个新请求创建一个 `Session` 对象。`Session` 对象是上下文的一部分(也可以通过页面成员获取)。ASP.NET 会为每个 `Session` 对象分配一个标识符(稍后会作介绍),在包含有效会话标识符的请求传入后,相应的 `Session` 对象便会被激活。`Session` 对象会伴随该页面,以方便地存储伴随会话生命周期的信息,而不仅仅是在页面的生命周期中使用。

`Session` 对象是键/值对形式的字典。可以通过键来关联任何基于“公共语言运行库”(CLR)的对象,并将其存储在 `Session` 对象中,以便会话期间后续的请求能够访问。然后,可以使用相同的键来获取 `Session` 对象存储的数据。例如,如果要在 `Session` 对象中存储用户提供的数据,则可以这样编写代码:

```
void StoreInfoInSession()
{
    String strFromUser = TextBox1.Text;
    Session["strFromUser"] = strFromUser;
}
```

在后续的请求到达后,可以通过这样的代码来获取这个字符串:

```
void GetInfoFromSession()
{
    String strFromUser = Session["strFromUser"]; // NOTE: may be null
    TextBox1.Text = strFromUser;
}
```

`Session` 对象的方括号用于调用索引器。“索引器”(indexer)语法非常适合通过键来设置和

获取值——也可以通过这种方式来向 Session 对象插入和从中获取数据。但应注意的是，如果指定的键未与会话字典中的数据建立映射，那么 Session 对象会返回 null。在生产环境中，应总是检查其值是否为空并做相应的处理。

管理 ASP.NET 中的会话状态非常简单。ASP.NET 提供了多种会话状态的管理方式，其中包括(1)将会话状态存储在“进程中”(in proc)，即 ASP.NET 工作进程；(2)将会话状态存储在运行 Windows Service 进程的状态服务器中；(3)将会话状态存储在 Microsoft SQL Server 数据库中。由于会话状态的管理也遵循之前章节提到的 provider 模式，因此将 ASP.NET 内建的会话状态管理机制替换为其他实现是相对容易的。

下面，让我们通过示例对会话状态有一个初步认识。

14.3 会话状态简介

下面的练习会创建一个网站，将一个成员变量存储在状态中，以加深对会话状态的理解。这个练习将揭示页面状态和会话状态之间的差异。

➤ 实践会话状态

- 1. 创建一个“ASP.NET 空 Web 应用程序”，将其命名为 SessionState。
- 2. 添加一个“Web 窗体”，将其命名为 Default.aspx。^①在这个页面中添加一个文本框，以便输入要存储在会话状态的值。为这个文本框添加一个标签，如下图所示：



- 3. 在页面上添加两个按钮和两个标签。一个按钮用于向会话状态添加数据，另一个用于显示会话状态的内容。将第一个按钮的 ID 设置为 AddStringToSessionState，将 Text

^① 译者注：由于创建的是“ASP.NET 空 Web 应用程序”，Visual Studio 创建的项目不会包含这个文件，需要自行添加。这里根据 Visual Studio 2010 RTM 对原书进行修正。

属性设置为“Add String To Session State”。在“设计器”中双击这个按钮，为其在代码隐藏文件中添加一个事件处理程序。(这个按钮最终会将文本框中的字符串添加到页面的会话状态中。)这样便可以将其与后面的按钮区分开来。为第二个按钮设置一个名称(这里使用了“Just Submit”，并保持其默认 ID)。第一个按钮会将文本框中的字符串提交给服务器，并将其存储在一个局部变量中(暂时先这样做)，而另一个按钮只用于触发页面的回发。通过这种方式，我们可以观察页面成员变量的短暂行为。将第一个标签的 Text 属性设置为“Value of string held in MEMBER VARIABLE:”，将第二个标签的 ID 设置为 LabelShowString。稍后会用第二个标签来显示字符串。

4. 为页面添加一个名为 sessionString 的字符串成员变量。在 Page_Load 中，将文本框的内容设置为这个字符串变量。然后，将注意力放在 AddStringToSessionState 按钮的事件处理程序上。在这个处理程序中，获取 TextBox1 的 Text 属性的值，将其赋给刚刚添加的成员变量。然后，将 LabelShowString 标签的文本设置为这个字符串的值，如下所示。

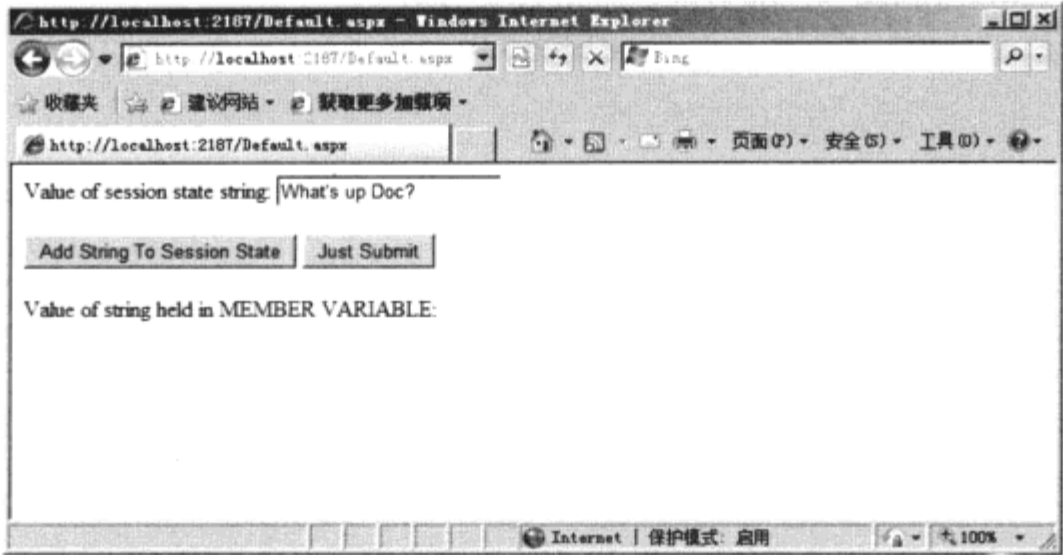
```
public partial class _Default : System.Web.UI.Page
{
    string sessionString;
    protected void Page_Load(object sender, EventArgs e)
    {
        this.LabelShowString.Text = this.sessionString;
    }
    protected void AddStringToSessionState_Click(object sender, EventArgs e)
    {
        this.sessionString = this.TextBox1.Text;
        this.LabelShowString.Text = this.sessionString;
    }
}
```

5. 运行这个页面。在文本框中键入一些文本，然后单击“Add String To Session State”。当页面回发后，标签会像下图这样显示所输入的内容。



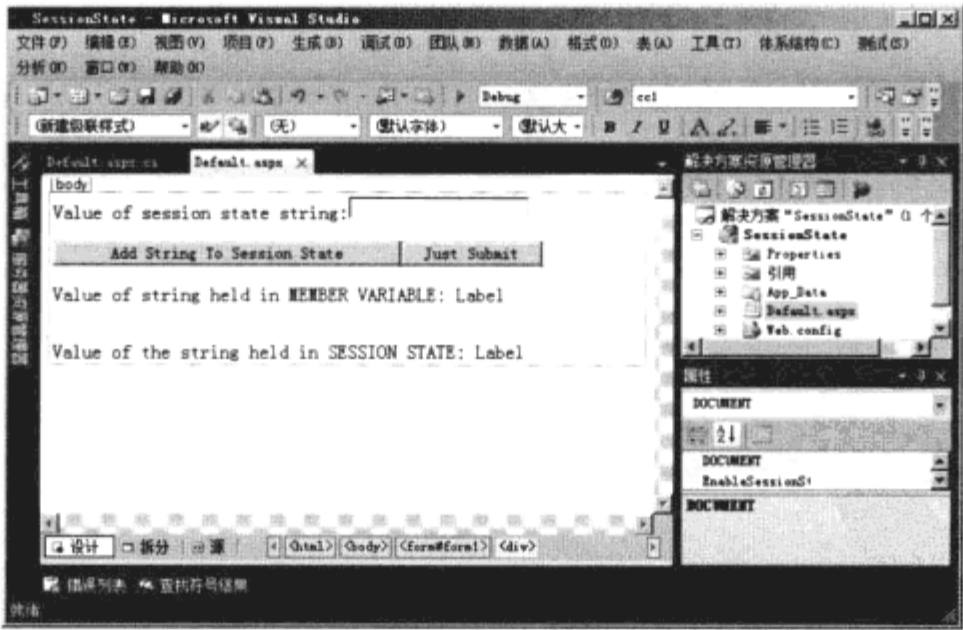
6. 单击“Just Submit”按钮。这时发生了什么？请不要忘记，Page_Load 只是将成员变量 sessionString 的值赋给标签。页面对象(和一般的 HTTP 处理程序)的生命周期十分短暂。

它们只在请求的处理期间有效，随后便会被销毁——其所持有的数据也会被一同销毁。成员变量 sessionString 在请求处理完毕后便会被立即清除。页面重新被创建后，成员变量 sessionString 会被重新初始化(null)(见下图)：



第 4 章介绍了控件管理自身状态的方式。与之不同的是，这个示例从文本框获取数据，然后将其存储在 Page 类的成员变量中。页面的生命周期十分短暂，在生成响应后便会消失。存储在页面成员中的数据也会随之消失。这就是单击“Just Submit”按钮后输入的字符串没有被显示出来的原因。在单击“Add String To Session State”按钮后能看到这个字符串，是因为这个成员变量的生命周期足以支持该按钮的 Click 事件处理程序。

- 7. 使用会话状态可以有效地解决这个问题。再添加两个标签，将第一个标签的 Text 属性设置为“Value of the string held in SESSION STATE:”，将第二个标签的 ID 设置为 LabelShowStringAsSessionState。这个标签用于显示从 Session 对象获取的数据(见下图)。



- 8. 编写代码，将字符串存储在会话状态中。使 AddStringToSessionState 按钮的处理程序从 TextBox1 获取文本，并将其存储在 Session 对象中。然后更新 Page_Load 方法，使其将会话状态的值显示出来，如下所示。

```
public partial class _Default : System.Web.UI.Page
{
    string sessionString;
```

```
protected void Page_Load(object sender, EventArgs e)
{
    this.LabelShowString.Text = this.sessionString;
    this.LabelShowStringAsSessionState.Text =
        (string)this.Session["sessionString"];
}

protected void AddStringToSessionState_Click(object sender, EventArgs e)
{
    // store in member variable
    this.sessionString = this.TextBox1.Text;

    // store in session state
    this.Session["sessionString"] = this.TextBox1.Text;

    // show member variable
    this.LabelShowString.Text = this.sessionString;

    // show session state
    this.LabelShowStringAsSessionState.Text =
        (string)this.Session["sessionString"];
}
}
```

9. 运行这个页面。键入一些文本，并单击 Add String To Session State 按钮。此时，两个标签应该都包含数据。LabelShowString 标签有数据是因为 Page_Load 将这个字符串赋给了成员变量。LabelShowStringAsSessionState 标签有数据是因为 Page_Load 将字符串存储在会话状态中(见下图)。



10. 现在单击下图所示的 Just Submit 按钮，看看会发生什么。
在这种情况下，页面仅仅被提交，因此只有 Page_Load 被执行。Page_Load 会显示成员变量 sessionString(其值为空，因为它与页面的生命周期相同)和来自 Session 对象(该对象与页面的生命周期无关)的数据。



从这个示例可以看出，会话状态的使用非常方便。然而，只是存储字符串或标量(scalar)往往是不够的。幸运的是，会话状态字典能够存储任何类型的 CLR 对象。

14.4 会话状态与复杂的数据类型

ASP.NET 中的 Session 对象能够存储运行在 CLR 中的任何(可序列化的)对象。这涉及的数据就多了——而不仅仅是字符串和其他标量类型。通过 Session 对象实现的最常见的功能之一是存储伴随客户端的数据(如购物车)。例如，在某个面向商业的网站中，用户可以购买商品。我们可能要建立一个存储库存信息的中央数据库。然后，用户登录后可以从中挑选商品，然后将其置于与会话状态关联的临时存储区中。对于 ASP.NET 网站，这个存储区一般就是 Session 对象。

有许多种集合可以实现类似购物车的功能。使用最简单的是我们比较熟悉的 ArrayList，它能够自动调整大小，支持随机访问和 IList 接口。然而，在某些情况下可能要使用 DataTable、DataSet 和其他更为复杂的类型。不过，DataTable 和 DataSet 对于某些情况显得过于复杂，可能会使会话状态变得臃肿，应谨慎使用。

第 10 章简单介绍了 ADO 和数据访问。下面的示例会用到那一章介绍的数据绑定控件(DataList 和 GridView)和 DataTable。这个示例将演示使用 ADO.NET 对象、数据绑定控件和会话状态来实现从库存(用 DataList 显示)中挑选商品(用 GridView 显示)。

► 使用 ADO.NET 对象、数据绑定控件和会话状态

1. 在 SessionState 项目中添加一个页面，将其命名为 UseDataList.aspx。
将以下 DataList 控件的代码复制到该页面的<div>节。这个 DataList 用于显示第 10 章中数据库的 DotNetReferences 表。

```
<asp:DataList ID="DataList1"
    runat="server" BackColor="White" BorderColor="#E7E7FF"
    BorderStyle="None" BorderWidth="1px" CellPadding="3"
    GridLines="Horizontal"
    Style="z-index: 100; left: 8px; position: absolute; top: 16px"
    OnItemCommand="DataList1_ItemCommand" Caption="Items in Inventory">
```

```
<FooterStyle BackColor="#B5C7DE" ForeColor="#4A3C8C"/>
<SelectedItemStyle BackColor="#738A9C"
    Font-Bold="True" ForeColor="#F7F7F7"/>
<AlternatingItemStyle BackColor="#F7F7F7"/>
<ItemStyle BackColor="#E7E7FF" ForeColor="#4A3C8C"/>
    <ItemTemplate>
        ID:
        <asp:Label ID="IDLabel"
            runat="server" Text='<%=Eval("ID") %>'></asp:Label><br/>
        Title:
        <asp:Label ID="TitleLabel"
            runat="server" Text='<%=Eval("Title") %>'></asp:Label><br/>
        AuthorLastName:
        <asp:Label ID="AuthorLastNameLabel"
            runat="server" Text='<%=Eval("AuthorLastName") %>'></asp:Label><br/>
        AuthorFirstName:
        <asp:Label ID="AuthorFirstNameLabel"
            runat="server" Text='<%=Eval("AuthorFirstName") %>'></asp:Label><br/>
        Topic:
        <asp:Label ID="TopicLabel"
            runat="server" Text='<%=Eval("Topic") %>'></asp:Label><br/>
        Publisher:
        <asp:Label ID="PublisherLabel"
            runat="server"

        Text='<%=Eval("Publisher") %>'></asp:Label><br/>
    <br/>

<asp:Button ID="SelectItem"

    runat="server" Text="Select Item"/>
    &nbsp;
</ItemTemplate>
    <HeaderStyle BackColor="#4A3C8C" Font-Bold="True" ForeColor="#F7F7F7"/>
</asp:DataList>
```

此时的 Visual Studio “设计器” 应如下图所示。



2. 为“Select Item”按钮添加一个事件处理程序。选择页面上的 DataList1。为查看这个 DataList 可用事件，单击 Visual Studio “属性”窗口的闪电图标。在 ItemCommand 事件旁边的编辑框会显示一条名为 DataList1_ItemCommand 的命令。双击这个编辑框，Visual Studio 会生成一个处理程序。按钮的处理程序名称为 DataList1_ItemCommand，与 DataList1 中的标识符是一致的。我们可以通过这个处理程序将库存中的货物添加到选定项表中。

```
public partial class UseDataList : System.Web.UI.Page
{
    protected void DataList1_ItemCommand(object source,
        DataListCommandEventArgs e)
    {
    }
}
```

3. 为页面添加一个方法，将其命名为 GetInventory。通过这个方法打开数据库并填充 DataList。配套的示例代码中有一个名为 AspNetStepByStep4 的 SQL Server 数据库。从第 10 章的项目中复制这个数据库到当前项目的 App_Data 文件夹下。将其添加到项目中后，我们可以在“服务器资源管理器”中看到它。在“服务器资源管理器”中选择这个数据库，“属性”窗口会显示相应的连接字符串。

```
using System.Data;
using System.Data.Common;

public partial class UseDataList : System.Web.UI.Page
{
    protected DataTable GetInventory()
    {
        string strConnection =
            @"Data Source=
            .\SQLEXPRESS;
            AttachDbFilename=|DataDirectory|\ASPNETStepByStep4.mdf;
            Integrated Security=True;
            User Instance=True";

        DbProviderFactory f =
            DbProviderFactories.GetFactory("System.Data.SqlClient");

        DataTable dt = new DataTable();
        using (DbConnection connection = f.CreateConnection())
        {
            connection.ConnectionString = strConnection;
            connection.Open();
            DbCommand command = f.CreateCommand();
            command.CommandText = "Select * from DotNetReferences";
            command.Connection = connection;

            IDataReader reader = command.ExecuteReader();
        }
    }
}
```



```

        dt.Load(reader);
        reader.Close();
        connection.Close();
    }
    return dt;
}

protected DataTable BindToInventory()
{
    DataTable dt;
    dt = this.GetInventory();
    this.DataList1.DataSource = dt;
    this.DataBind();
    return dt;
}

// More goes here...
}

```

4. 添加一个名为 `CreateSelectedItemData` 的方法。来自数据库的架构信息将包含列的名称和数据类型。这个方法接受一个描述数据库表架构的 `DataTable` 对象(稍后会获取这个表)。该方法将根据这个表创建一个新表，以便将选定的商品放在这个新表中。为此，可以先创建一个空的 `DataTable`，然后将列添加到 `Columns` 集合中。^①

```

public partial class UseDataList : System.Web.UI.Page
{
    protected DataTable CreateSelectedItemTable(DataTable tableSchema)
    {
        DataTable tableSelectedItemData = new DataTable();
        foreach (DataColumn dc in tableSchema.Columns)
        {
            tableSelectedItemData.Columns.Add(dc.ColumnName,
                dc.DataType);
        }
        return tableSelectedItemData;
    }
}

```

5. 更新 `Page_Load` 方法。在页面的首个请求到达时(即请求不是以回发方式发起的)，`Page_Load` 要调用 `BindToInventory`，该方法会获取 `DotNetReferences` 数据库表的快照并存储在 `DataTable` 对象中。存储选定项的表将以这个 `DataTable` 的数据架构为基础。为此，声明一个 `DataTable` 变量，将 `CreateSelectedItemTable` 的返回值赋给它。然后，通过 `tableSelectedItem` 变量将这个表(此时是空的)存储在 `Session` 对象中。

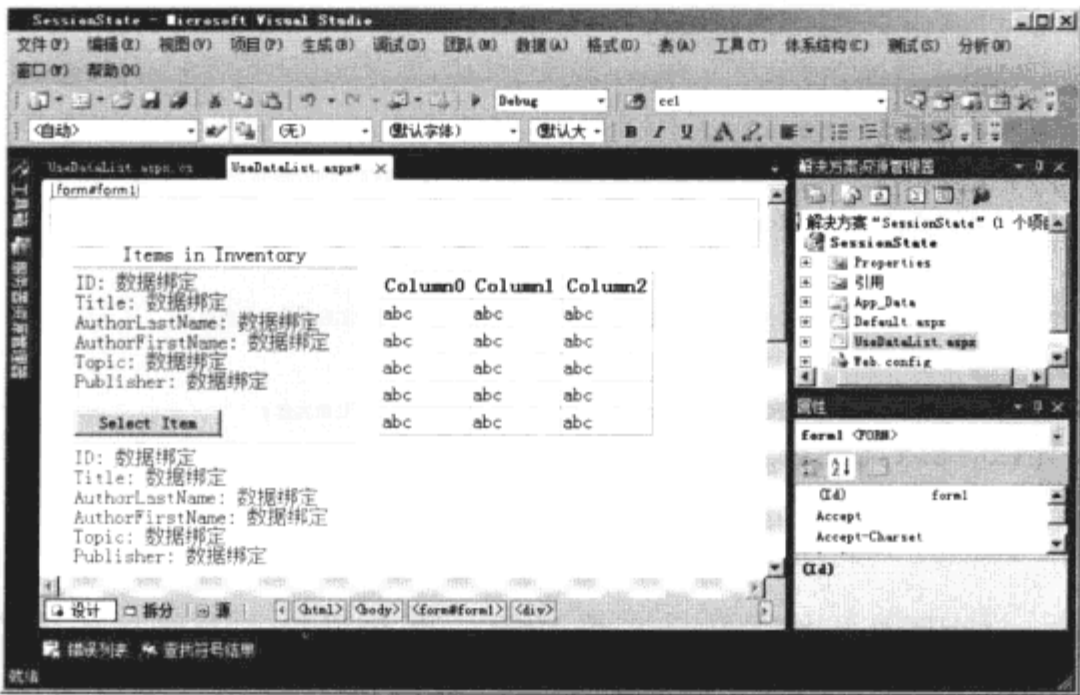
① 译者注：复制 `DataTable` 或 `DataSet` 架构的另一种方法是调用原对象的 `Clone()` 方法。该方法只复制自身的架构，而不会复制其中的数据。

```
public partial class UseDataList : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            DataTable dt = BindToInventory();
            DataTable tableSelectedItems =
                this.CreateSelectedItemsTable(dt);
            Session["tableSelectedItems"] = tableSelectedItems;
        }
    }
}
```

为确保数据库连接正确，运行这个页面。此时的页面如下图所示。



6. 如下图所示，在页面中添加一个 GridView。我们用它来显示会话状态中的选定项表。先不要急于为其设置数据源(稍后添加)。先要确保 AutoGenerateColumns 属性被设置为 true。页面之所以这样布局是因为它采用了绝对定位方式。



7. 完成“SelectItem”按钮的处理程序。这个方法需要将库存中的商品添加到选定项表。选定项的索引值可以从这个处理程序的 `DataListCommandEventArgs` 参数获得。调用 `BindToInventory` 来设置 `DataList` 的数据源，以便从中获取选定项。可以使用数字索引来访问列的值。用列的值构建新的 `DataRow`，并将新建的 `DataRow` 添加到选定项表中。将修改后的表存储在会话状态中。最后，将 `GridView1` 的 `DataSource` 设置为这个表，并调用 `DataBind` 方法。

```
public partial class UseDataList : System.Web.UI.Page
{
    protected void DataList1_ItemCommand(object source,
        DataListCommandEventArgs e)
    {
        int nIndex = e.Item.ItemIndex;
        this.DataList1.SelectedIndex = nIndex;

        BindToInventory();

        // Order of the columns is:
        // ID, Title, FirstName, LastName, Topic, Publisher

        DataTable dt = (DataTable)DataList1.DataSource;
        String strID = (dt.Rows[nIndex][0]).ToString();
        String strTitle = (dt.Rows[nIndex][1]).ToString();
        String strAuthorLastName = (dt.Rows[nIndex][2]).ToString();
        String strAuthorFirstName = (dt.Rows[nIndex][3]).ToString();
        String strTopic = (dt.Rows[nIndex][4]).ToString();
        String strPublisher = (dt.Rows[nIndex][5]).ToString();

        DataTable tableSelectedItems;
        tableSelectedItems = (DataTable)Session["tableSelectedItems"];

        DataRow dr = tableSelectedItems.NewRow();
        dr[0] = strID;
        dr[1] = strTitle;
        dr[2] = strAuthorLastName;
        dr[3] = strAuthorFirstName;
        dr[4] = strTopic;
        dr[5] = strPublisher;

        tableSelectedItems.Rows.Add(dr);

        Session["tableSelectedItems"] = tableSelectedItems;

        this.GridView1.DataSource = tableSelectedItems;
        this.GridView1.DataBind();
    }
}
```

8. 运行这个页面，结果如下图所示。在最开始，页面只会在左侧显示库存列表。单击其中的几个“Select Item”按钮。每次单击后，页面都会回发，并重新呈现 `DataList`，而

GridView 会显示新添加的选定项。



这个示例实现了一个使用会话状态的应用程序。下面我们来看看如何配置 ASP.NET 的会话状态。

14.5 会话状态的配置

ASP.NET 提供了多种管理会话状态的方法。我们可以将其完全禁用，可以将其存储在 ASP.NET 工作进程中，可以将其存储在单独的状态服务器上，也可以将其存储在 SQL Server 数据库中。下面列出了这几种选择：

- **完全禁用** 如果禁用会话状态，那么页面则无需在启动时加载会话状态，也不用存储会话状态，因而应用程序的性能会有所提升。然而，在切换不同页面时，我们无法将数据与特定用户关联。
- **在“进程中”存储会话状态** 这是会话状态的默认处理方式。在这种情况下，会话字典(Session 对象)受处理页面和代码的进程管理。在进程中存储会话状态的优点在于速度快、使用方便。然而，其中的数据并不是持久的。例如，如果重启“Internet 信息服务”(IIS)或服务器计算机，那么所有会话状态都将丢失。在某些情况下，这可能并无大碍。然而，如果数据相对重要(如购物车中包含大量订单)，那么丢失数据可能就不偿失了。此外，进程中会话管理程序只适合单台计算机环境，而并不适合 Web 场(Web farm)环境。(Web 场是一组绑定在一起的服务器，将网页作为一个单独的应用进行服务。)^①

^① 译者注：在 Web 场中，同一客户端的每次回发可能由不同的计算机处理，因此需要集中管理会话状态。

- **在状态服务器中存储会话状态** 在这种配置下，ASP.NET 运行库会将所有会话状态管理任务交给指定计算机上专门的 Windows 服务进程。采用这种会话状态管理方式的优势在于可以在 Web 场中运行应用程序服务器。ASP.NET 中有专门支持在 Web 场中使用会话状态的工具。在 Web 场中，应用程序的每个实例要从同一处获取会话信息。这种管理方式的劣势在于，其在某种程度上对性能有负面影响——在加载和存储会话信息时，应用程序要通过网络与状态服务器进行交互。ASP.NET 4 支持压缩。为降低状态服务器存储和传输的数据量，可以启用该功能。为此，只需要在 Web.config 的 sessionState 节中添加 compressionEnabled=true 设置。
- **在数据库中存储会话状态** 在这种配置下，可以实现将会话信息存储在网络上的 SQL Server 数据库中。如果需要采用 Web 场，或者希望持久地、安全地存储会话状态，则可以采用这种方式。除了可以将数据存储在 SQL Server 数据库中，还可以通过压缩来降低传输和存储的数据量。同样，只需要在 Web.config 的 sessionState 节中添加 compressionEnabled=true 设置。

如果在开发阶段需要配置 ASP.NET 会话状态，则可以直接编辑配置文件。如果网站已部署，则可以通过 IIS 中的“会话状态”页面(见下图)来进行配置。



14.5.1 禁用会话状态

我们可以通过 IIS 的 ASP.NET 会话状态配置工具在 Web.config 文件中插入相应的配置字符串。如果要完全禁用会话状态，请在“会话状态模式设置”中选择“未启用”。

14.5.2 在进程中存储会话状态

如果要在 ASP.NET 的工作进程中存储会话状态，则在“会话状态模式设置”中选择“在进程中”。在这种情况下，应用程序能够快速存取会话信息，但会话状态信息只对存储它

的服务器可用。(也就是说，Web 场中的其他服务器将无法使用这些信息。)

14.5.3 在状态服务器中存储会话状态

为使 ASP.NET 将会话状态存储在网络上的其他服务器，请在“会话状态模式设置”中选择“状态服务器”。在选择这个选项后，“连接字符串”和“超时”文本框会被启用。在“连接字符串”中设置所采用的协议、状态服务器的 IP 地址和端口。例如：

```
tcpip=localhost:42424
```

采用该设置，会话状态会存储在本地计算机的 42424 端口。如果要将会话状态存储在其他计算机上，只需要用该计算机的 IP 地址替换为“localhost”即可。在实际将数据存储在目标服务器中之前，需要确保其“ASP.NET 状态服务”处于运行状态。为此，可以通过“控制面板”的“管理工具”打开“服务”进行查看。下图展示了“服务”控制台。



14.5.4 在数据库中存储会话状态

有关会话状态的最后一个选项是用 SQL Server 数据库来存储会话状态。在“会话状态模式设置”中选择“SQL Server”后，会启用它下面的组合框。我们可以在此输入 SQL Server 状态数据库的连接字符串。默认的连接字符串如下：

```
data source=localhost;Integrated Security=SSPI
```

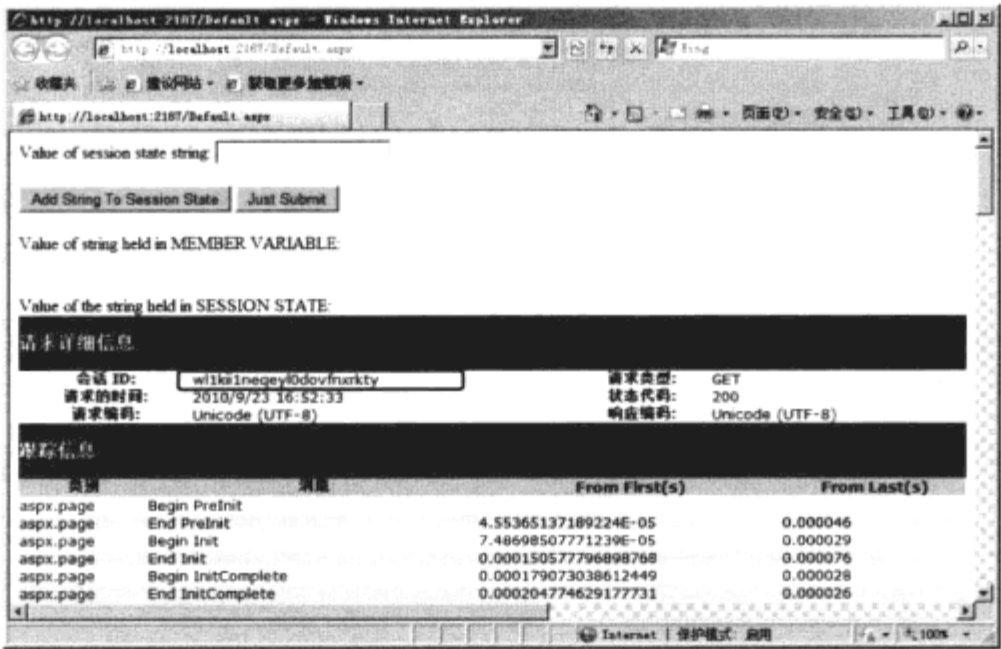
可以使 ASP.NET 引用其他服务器的数据库。当然，目标服务器需要安装 SQL Server。 .NET 系统目录(在本书截稿前，这个目录为 C:\Windows\Microsoft.NET\Framework\v4.0.30319)提供了创建该数据库的 SQL 脚本。aspnet_regsql.exe 工具也能够帮助我们建立这个数据库。

14.6 会话状态的跟踪

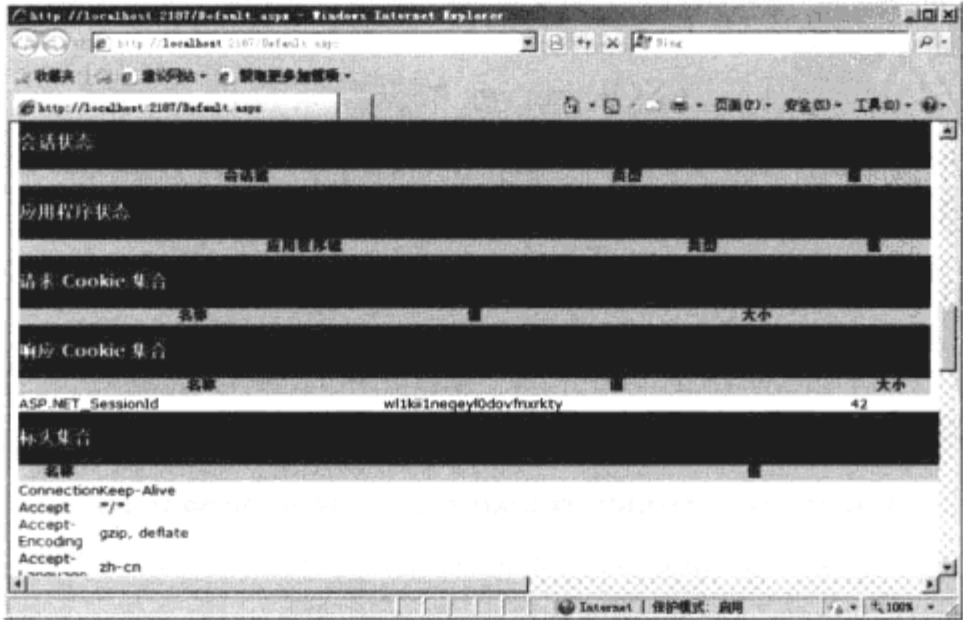
由于 Web 应用程序依赖于 HTTP 协议来在浏览器与服务器之间进行通信，并且需要 HTML 来表示应用程序的状态，因而 ASP.NET 本质上采用的是一种非连接的架构。在应用程序需要使用会话状态时，运行库需要跟踪请求的源头，以便能够将数据与特定的客户端关联起来。ASP.NET 提供了 3 种通过会话 ID 来跟踪会话状态的方法——Cookie、URL 和设备配置文件。

14.6.1 通过 Cookie 跟踪会话状态

这是 ASP.NET 网站的默认设置。在这种情况下，ASP.NET 会生成无规律的标识符，并基于它来存储 Session 对象。如果启用跟踪，则能够看到这个会话标识符。注意 ASP.NET 是如何在请求 Cookie 中存储会话 ID 的。跟踪信息包含会话变量的名称和值。下图展示了跟踪信息的“请求详细信息”中的会话 ID。

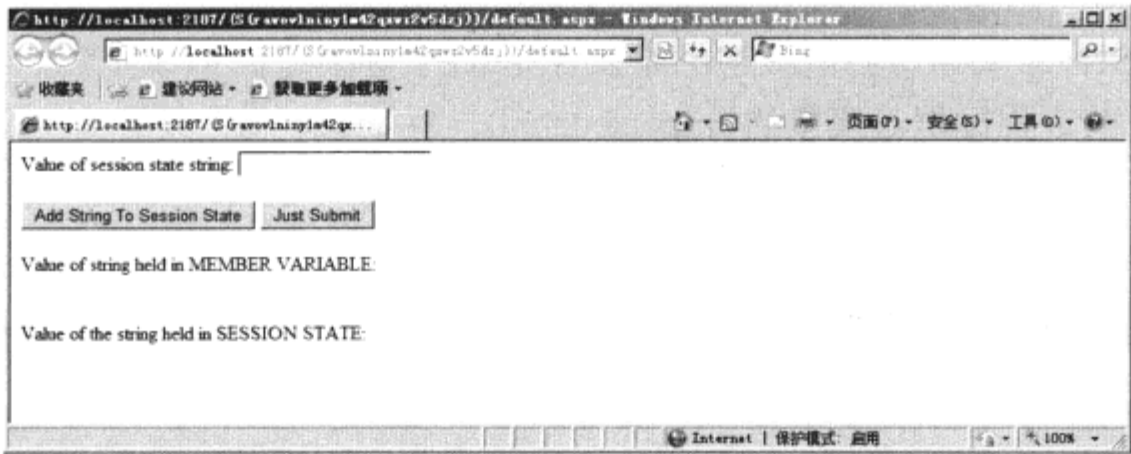


下图展示了跟踪信息中 Cookie 集合的内容，这表明会话 ID 只不过是 Cookie 中的一条记录。



14.6.2 通过 URL 跟踪会话状态

另一种主要的会话状态跟踪方法是在请求字符串中嵌入会话 ID。在客户端禁用或不支持 Cookie 的情况下(因而无法使用基于 Cookie 的会话跟踪)，就可以采用这种方式。注意下图中嵌在导航 URL 中的会话 ID。



14.6.3 自动检测

如果采用 AutoDetect(自动检测)方式，那么 ASP.NET 会检查客户端浏览器是否启用了 Cookie。如果 Cookie 被启用，那么会话标识符会由 Cookie 承载。否则，会话 ID 会由 URL 承载。

14.6.4 使用设备配置文件

UseDeviceProfile 选项会使 ASP.NET 根据针对请求而建立的 HttpBrowserCapabilities 对象的 SupportsRedirectWithCookie 属性来判断客户端浏览器是否支持 Cookie。如果该属性为 true，那么相应请求的会话标识符会由 Cookie 承载，否则将由 URL 承载。

14.6.5 会话状态超时

timeout 设置能够决定会话的生命周期。会话的生命周期(单位为分钟)是指在 ASP.NET 移除会话信息并使会话 ID 失效之前，允许会话空闲的时间。超时的最大值为 525601 分钟(一年)，默认值为 20 分钟。

14.7 会话的其他设置

ASP.NET 还支持一些无法通过 IIS 工具修改的设置，这些设置需要直接修改 Web.config 文件。

如果不希望使用 ASP.NET 在 Cookie 中生成的会话 ID 的名称(默认为 ASP.NET_SessionId)，可以通过 cookieName 来修改它。为防止黑客截获会话 ID，有时需要修改会话 ID 的名称。

如果要自动续订过期的会话 ID，则将 regenerateExpiredSessionId 设置为 true 即可。该设置只适用于无 Cookie 会话。

如果不希望使用 ASP.NET 提供的 SQL Server 数据库来支持会话状态，则可以选择其他数据库。为此，启用 allowCustomSqlDatabase 即可。

如果使用 SQL Server 来存储会话状态，那么 ASP.NET 必须要作为 SQL Server 的客户端。通常 ASP.NET 的进程标识是模拟的(impersonated)。将 mode 属性设置为 Custom，可以使 ASP.NET 采用 Web.config 中 identity 配置元素提供的用户标识。如果将 mode 设置为 SQLServer，那么 ASP.NET 会使用信任的连接。

stateNetworkTimeout 用于设置 Web 服务器与状态服务器之间或 Web 服务器与 SQL Server 之间的 TCP/IP 网络连接的最长空闲时间(单位为秒)。

最后，我们还可以使用自定义的会话状态提供程序。为此，要将 custom 元素设置为提供程序的名称，并在 Web.config 中的其他位置指定这个提供程序(providers 元素中)。

14.8 Wizard 控件——会话状态的一种替代方案

用户提供的信息往往跨越多个页面，跟踪这些信息是会话状态最常见的用途之一。例如，采集邮寄地址、获取安全凭据及在线支付，都会涉及这样的问题。

有时，这种信息可能很少，一个页面就能够满足要求。然而，如果要通过多页的表单来采集数据，则需要跟踪每次回发间的信息。例如，大多数商业网站的支付过程都涉及多个阶段。用户在购物车添加所需的商品后，他/她会单击“付款”，网站会将其带到付款页面。在付款页面中，用户通常要执行以下几步操作：选择支付方法，确认订单，接收订单确认信息。

虽然可以采用 ASP.NET 1.x 风格的代码来实现多阶段的数据采集，但 ASP.NET 的后续版本提供了 Wizard 控件，可以更方便地实现该特性。

如果要实现多阶段的数据采集，可能需要实现导航逻辑，并跟踪过程状态。Wizard 控件提供了一个模板，可以通过多个指定的输入页面来完成导航任务。Wizard 控件的逻辑需要通过指定的若干步骤来实现，该控件能够对这些步骤进行管理。Wizard 控件支持线性的和非线性的导航。

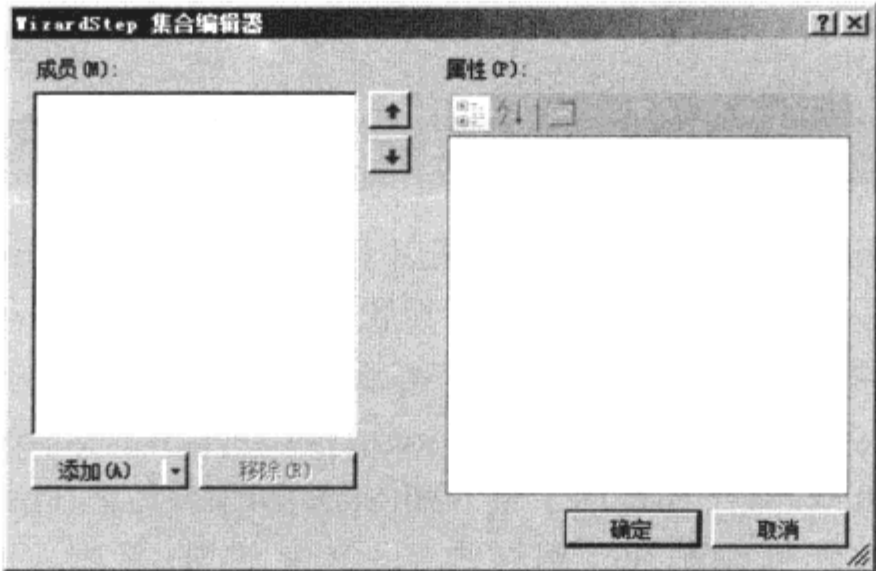
下面的练习将演示如何使用 Wizard 控件来从客户端收集各方面的信息：姓名、地址、他/她所使用的软件和硬件。可以通过这些信息来确定是否允许用户进入网站的某个区域或是否允许其订阅网站内容。

➤ 使用 Wizard 控件

- 1. 在 SessionState 项目中添加一个新页面，将其命名为 UseWizard.aspx。
- 2. 从“工具箱”拖放一个 Wizard 控件实例到页面上。
- 3. 当“设计器”中显示“Wizard 任务”菜单后，单击“自动套用格式”。这里选择“专业型”。

本示例会定制 StartNavigationTemplate 和 SidebarTemplate，以便更好地控制 Wizard 控件的外观。(本示例不会介绍两者的使用方法，这里提到它们只是为了说明其在 Wizard 控件中的作用。)我们可以为这两个模板添加控件来定义 Wizard 的外观。为将这两个区域转换为模板，单击 Wizard 右上角的小箭头，然后单击“转换为 StartNavigationTemplate”和“转换为 SideBarTemplate”。

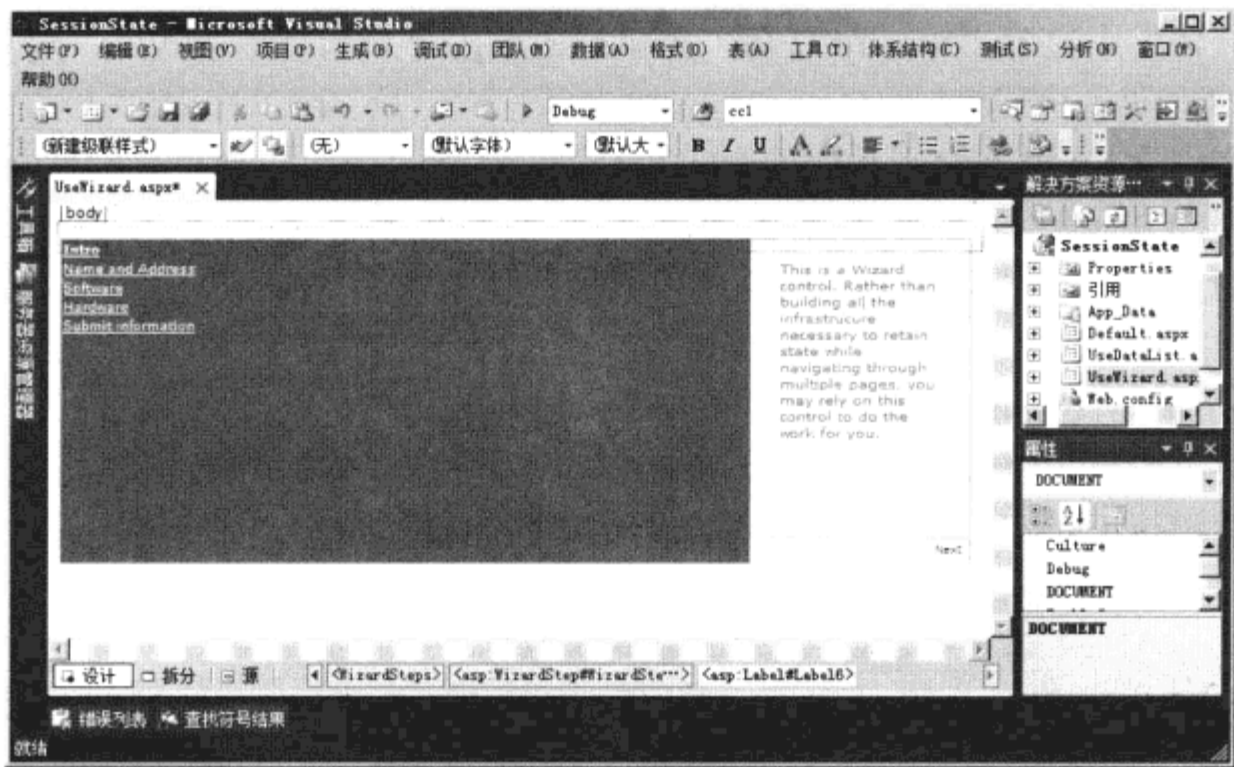
在这个菜单中单击“添加/移除 WizardSteps”。此时会打开“WizardStep 集合编辑器”对话框。Visual Studio 默认会添加两个示例步骤。将其删除，如下图所示。



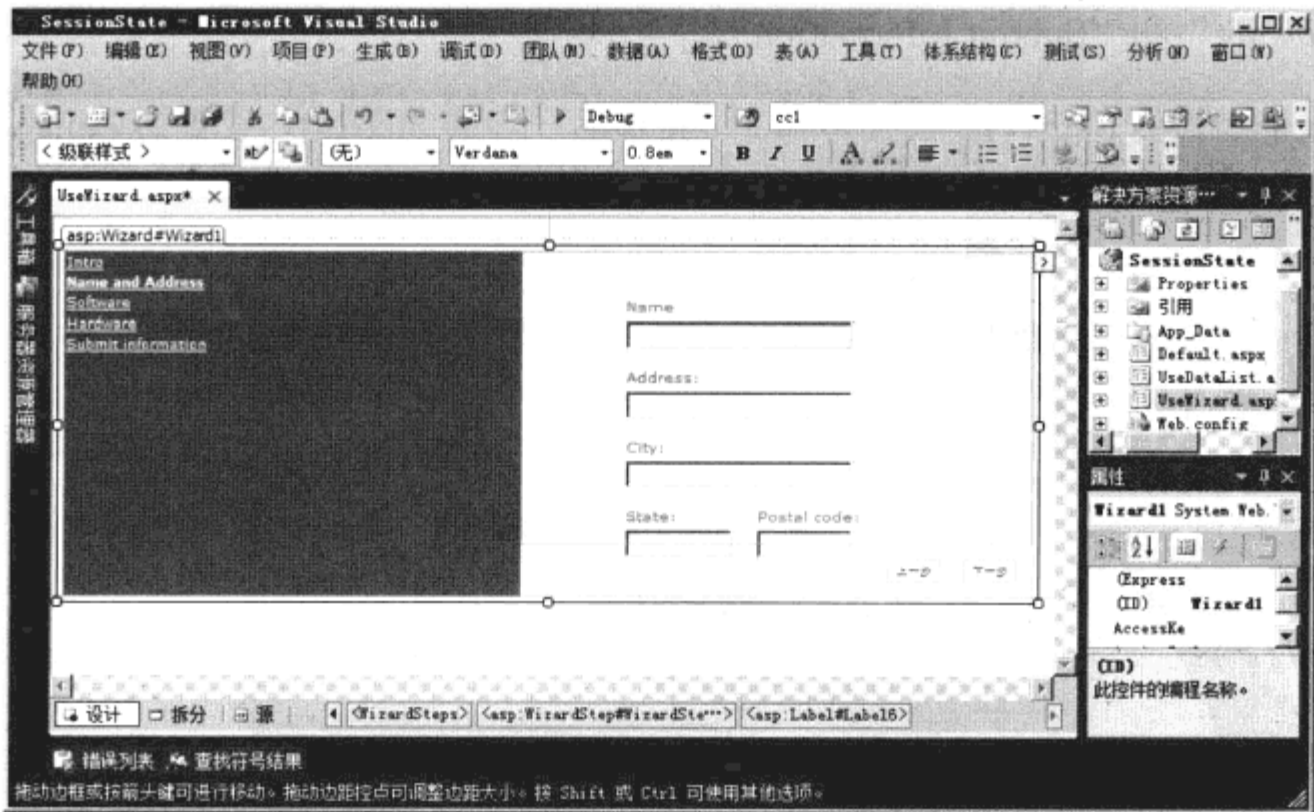
- 4. 单击“添加”按钮，分别添加 Intro、Name and Address、Software、Hardware 和 Submit 步骤。将这些步骤的 Title 分别设置为“Intro”、“Name and Address”、“Software”、“Hardware”和“Submit Information”。
- 5. 将第一个步骤的 StepType 设置为 Start，将最后一步的 StepType 设置为 Finish，如下图所示。设置完毕后，单击“确定”。



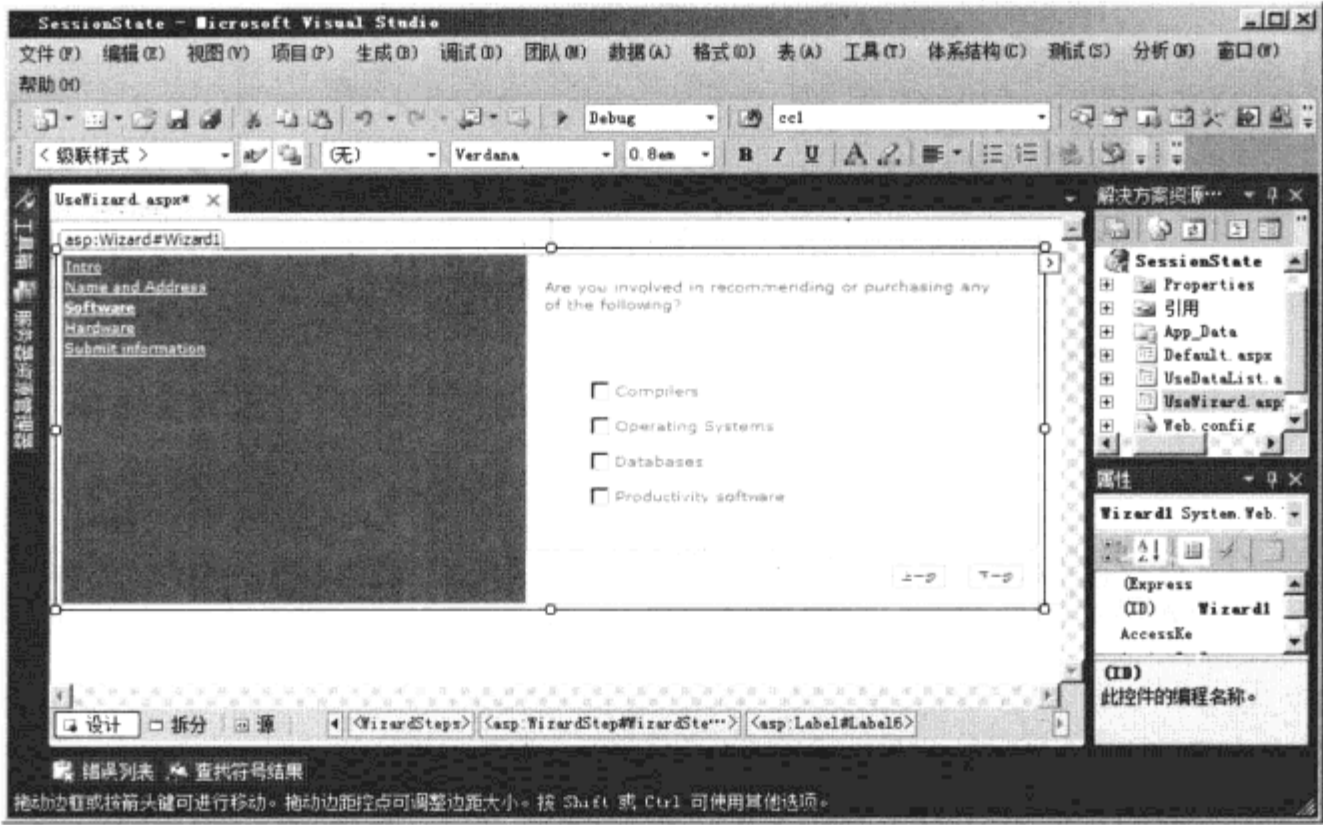
6. 在步骤中添加控件。首先，在“设计器”中选中 Wizard 控件，然后在“格式”菜单中选择“设置位置”|“绝对”。这时，我们可以修改 Wizard 控件的大小。将高度设置为 240 像素(px)，将宽度设置为 650 像素。单击 Wizard 控件右上角的小箭头，选择“Intro”步骤。在这个步骤中添加一个标签来描述所要输入的信息(见下图)。



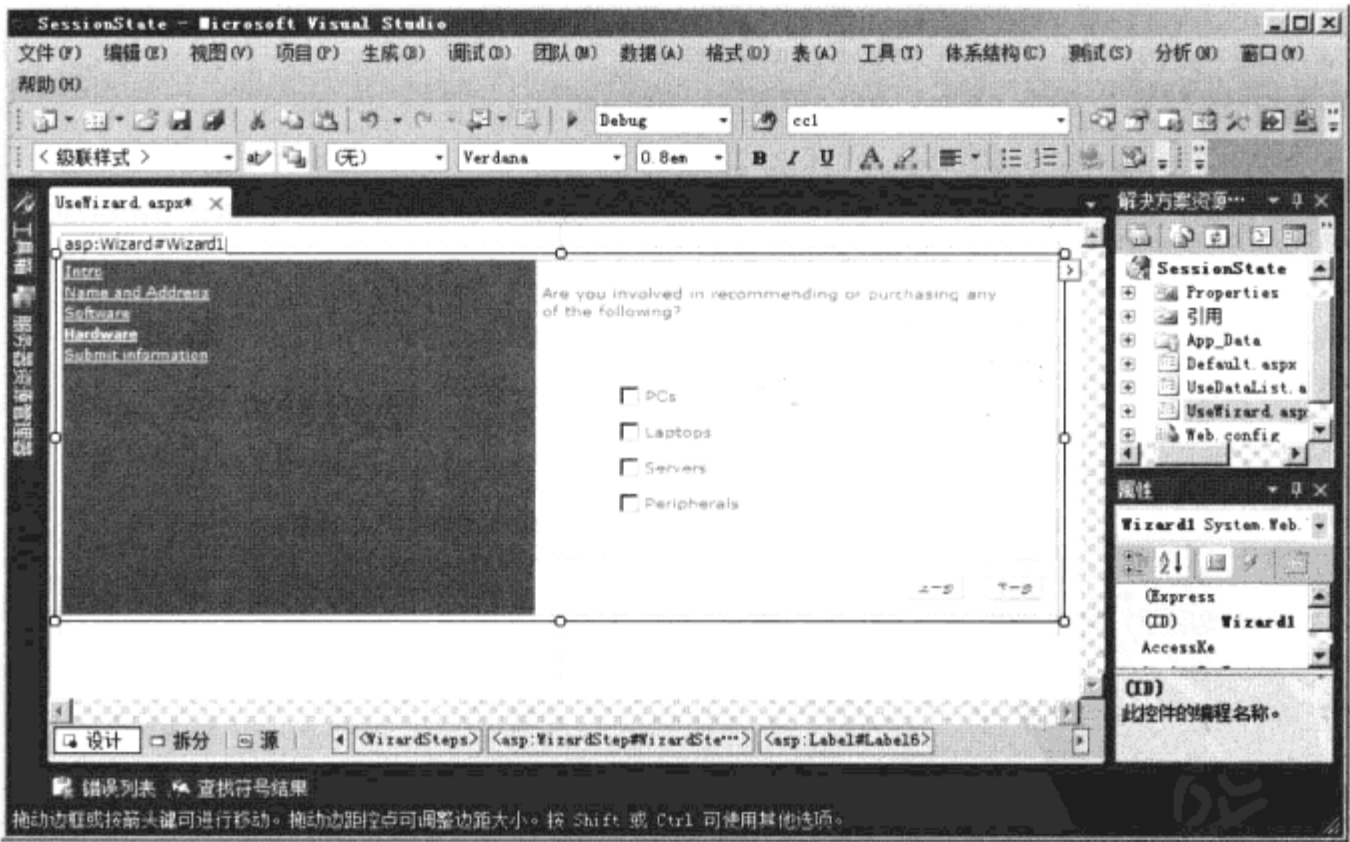
7. 选择“Name and Address”步骤，为其添加用于获取个人信息的标签和文本框。对于添加的每个控件，在“格式”菜单中选择“设置位置”|“绝对”，以便修改其长度和宽度。如下图所示，拖放一个 Label 控件到 Wizard 控件的右侧。在其下面添加输入姓名(Name)的 TextBox。继续添加输入地址(Address)的 Label 和 TextBox。添加输入城市(City)的 Label 和 TextBox，最下面是用于输入州(State)和邮政编码(Postal code)的 Label 和 TextBox。为每个文本框设置 ID。将这些 TextBox 的 ID 依次设置为 TextBoxName、TextBoxAddress、TextBoxCity、TextBoxState 和 TextBoxPostalCode。在提交步骤中需要用到这些控件。



8. 选择并修改 “Software” 步骤。“Software” 步骤应包含罗列了常用软件类型的复选框列表。添加一个 CheckBoxList 控件，将其 ID 设置为 CheckBoxListSoftware。按下图填充。

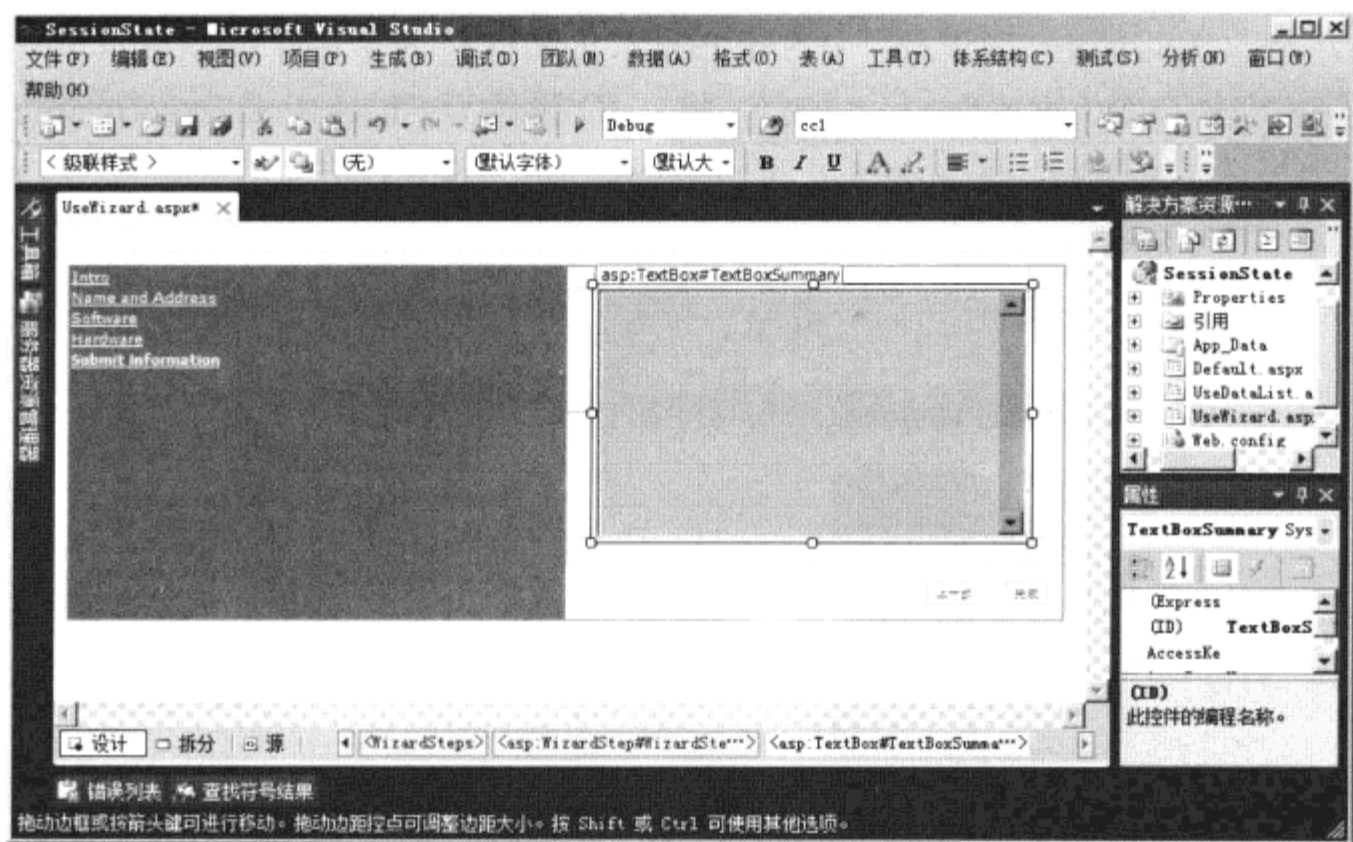


9. 选择 “Hardware” 步骤，为其添加列出了常用硬件类型的复选框。添加一个 CheckBoxList，将其 ID 设置为 CheckBoxListHardware。按下图填充。



10. “Submit information” 步骤(用来在提交前显示信息)应包含一个多行的 TextBox，用于显示所采集的信息的摘要。将这个 TextBox 的 ID 设置为 TextBoxSummary(见下图)。

ASP.NET 4 从入门到精通



11. 在 Page_Load 方法中添加以下代码，以便获取 Wizard 中各控件的信息。我们能够以页面的成员的形式访问所有控件。这样，每次加载页面都会获取用户的输入。然而，在到达最后一步之前，用户是无法看到这些信息的。双击 Wizard 控件，为“完成”按钮添加一个处理程序，并在这里处理向导生成的信息。

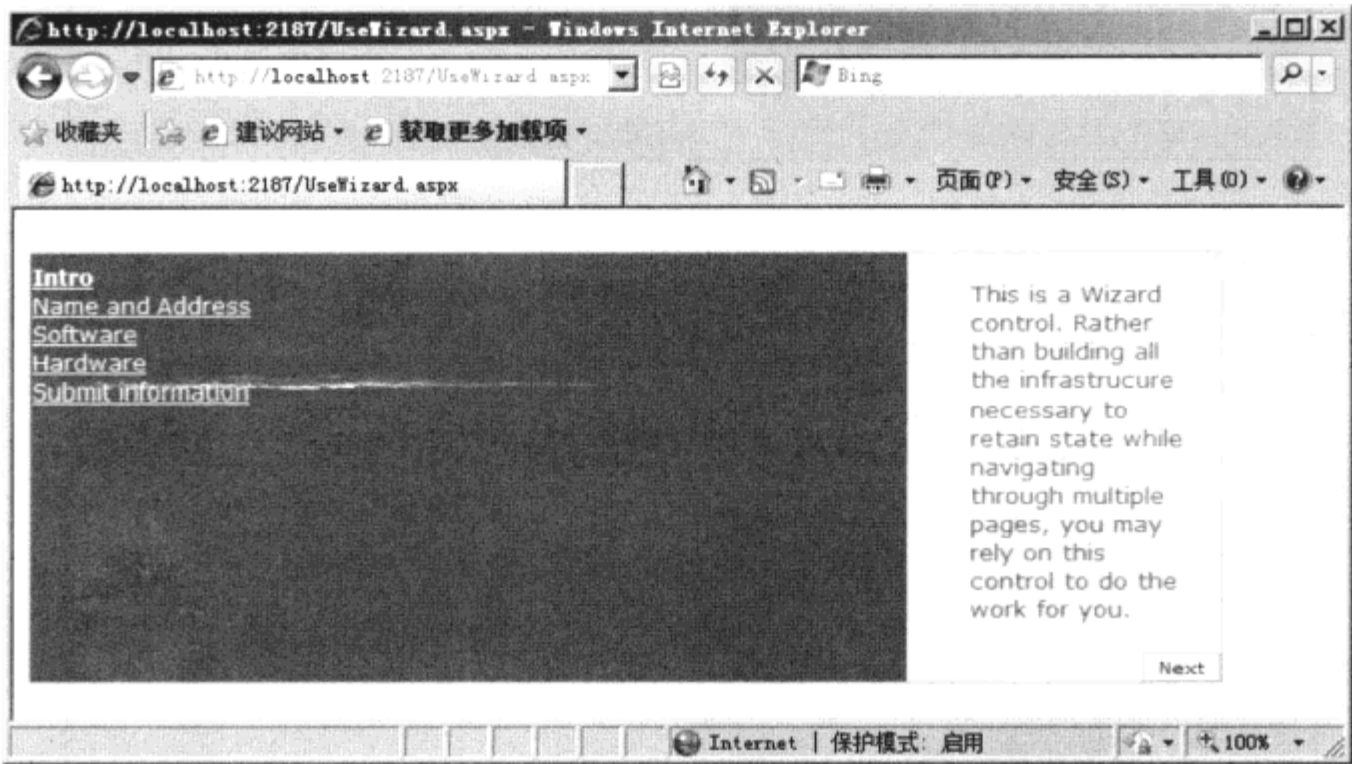
```
protected void Page_Load(object sender, EventArgs e)
{
    StringBuilder sb = new StringBuilder();
    sb.Append("You are about to submit. \n");

    sb.Append(" Personal: \n");
    sb.AppendFormat(" {0}\n", this.TextBoxName.Text);
    sb.AppendFormat(" {0}\n", this.TextBoxAddress.Text);
    sb.AppendFormat(" {0}\n", this.TextBoxCity.Text);
    sb.AppendFormat(" {0}\n", this.TextBoxState.Text);
    sb.AppendFormat(" {0}\n", this.TextBoxPostalCode.Text);
    sb.Append("\n Software: \n");
    foreach (ListItem listItem in CheckBoxListSoftware.Items)
    {
        if (listItem.Selected)
        {
            sb.AppendFormat(" {0}\n", listItem.Text);
        }
    }

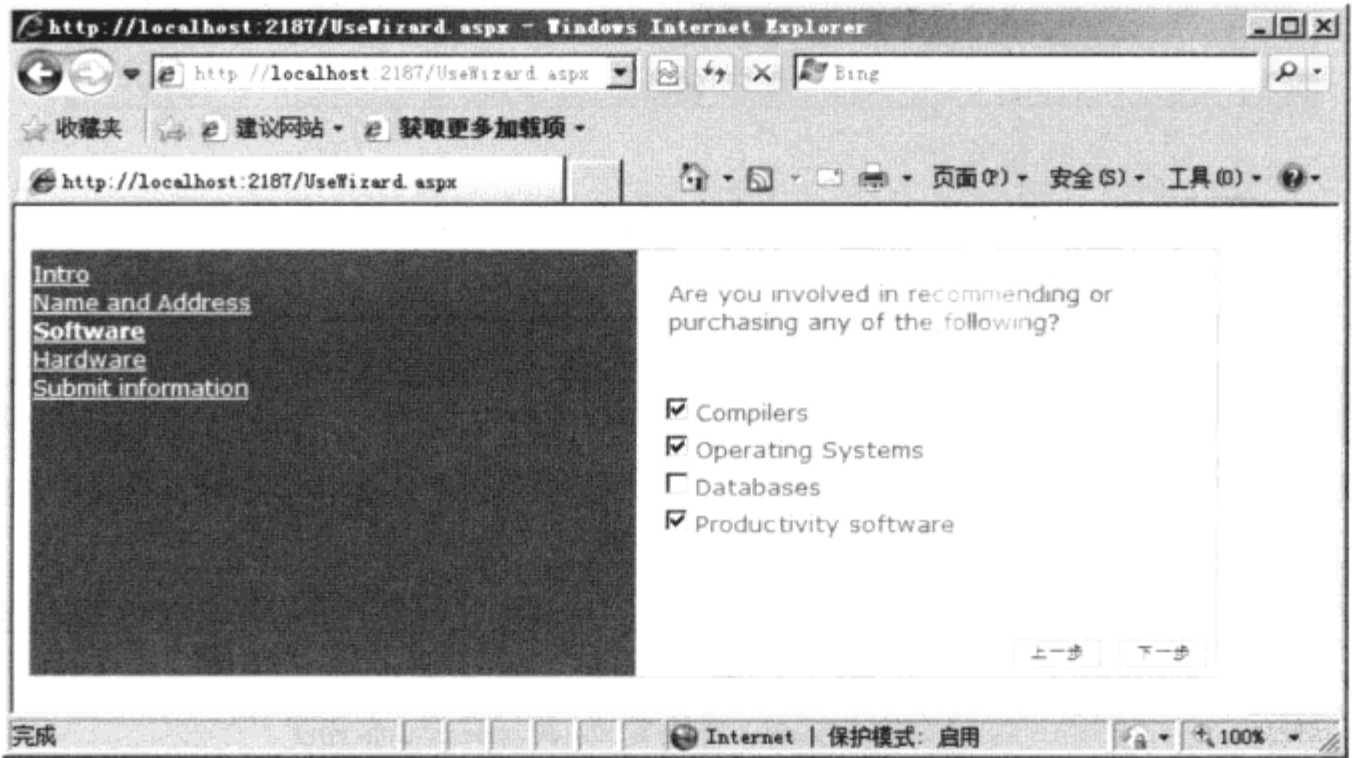
    sb.Append("\n Hardware: \n");
    foreach (ListItem listItem in CheckBoxListHardware.Items)
    {
        if (listItem.Selected)
        {
            sb.AppendFormat(" {0}\n", listItem.Text);
        }
    }
}
```

```
}
this.TextBoxSummary.Text = sb.ToString();
}
protected void Wizard1_FinishButtonClick(object sender,
    WizardNavigationEventArgs e)
{
    // Do something with the data here
}
```

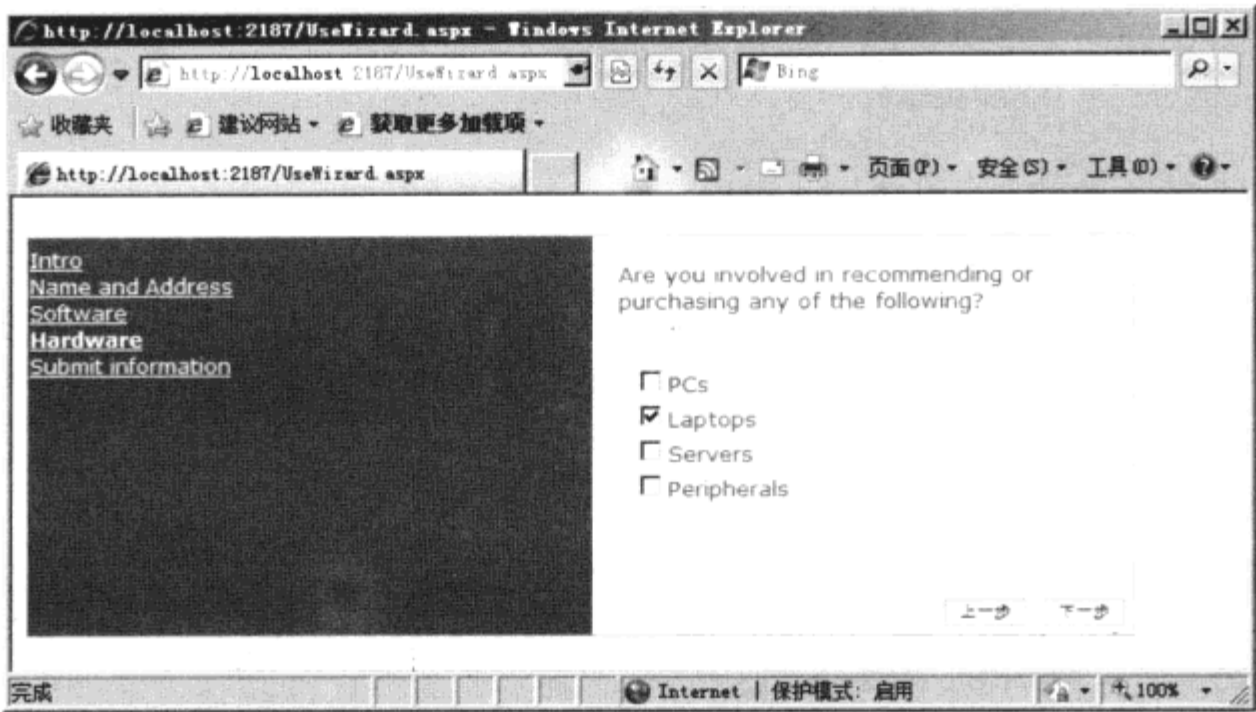
12. 运行这个页面，并测试所有步骤。我们将历经所有步骤，而最后会看到摘要。如果显示的步骤不是第一步(“Intro”), 则可能是由于页面运行在调试模式，且没有在“设计器”中选择“Intro”步骤。此时，只需要在“设计器”中选择该步骤，并重新运行页面。下图展示了浏览器中的“Intro”步骤。



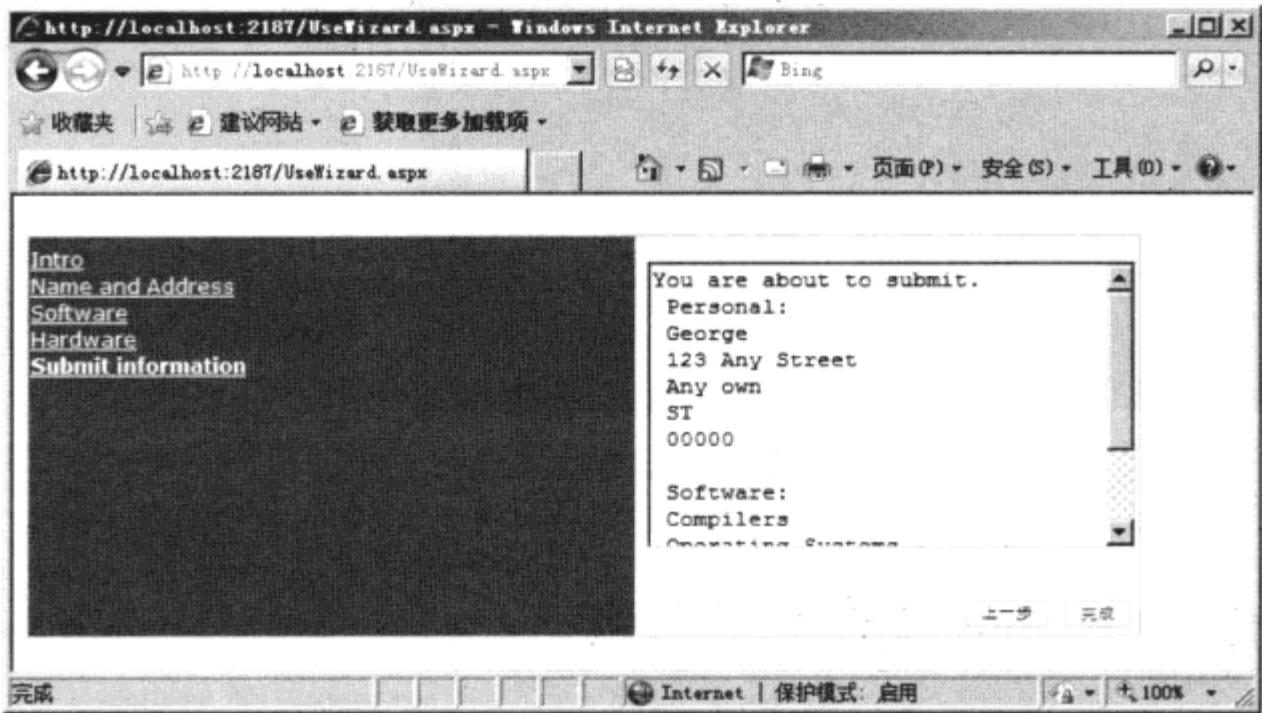
“Software” 步骤如下图所示。



“Hardware” 步骤如下图所示。



最后的步骤如下图所示。



14.9 快速参考

目 标	操 作
访问当前客户端的会话状态	使用 Page.Session 属性或当前上下文的 HttpContext.Session 属性
访问当前客户端会话状态中的值	会话状态是一种键/值对集合。可以通过存储数据时使用的键(字符串)来访问相应的数据
在进程中存储会话状态	在 Web.config 中编辑 sessionState 节的特性。将 mode 设置为 InProc
在状态服务器中存储会话状态	在 Web.config 中编辑 sessionState 节的特性。将 mode 设置为 StateServer，并设置 stateConnectionString

续表

目 标	操 作
在 SQL Server 中存储会话状态	在 Web.config 中编辑 sessionState 节的特性。将 mode 设置为 SQLServer，并设置 sqlConnectionString
禁用会话状态	在 Web.config 中编辑 sessionState 节的特性。将 mode 设置为 Off
使用 Cookie 来跟踪会话状态	在 Web.config 中编辑 sessionState 节的特性。将 cookieless 设置为 false(默认值)
使用 URL 来跟踪会话状态	在 Web.config 中编辑 sessionState 节的特性。将 cookieless 设置为 true
设置会话状态超时	在 Web.config 中编辑 sessionState 节的特性。将 timeout 设置为超时数(以分钟为单位)

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 15 章

应用程序数据的缓存

学习目标

- 通过缓存应用程序数据来改善应用程序性能
- 避免不必要的数据库访问
- 管理应用程序数据缓存中的元素

本章将介绍 ASP.NET 中内建的数据缓存功能。缓存一直是改善软件系统性能的方法之一。其主要思想是将频繁使用的数据置于能够快速访问的介质中。虽然大容量存储器的访问速度在不断改善，但从标准的硬盘访问数据的速度要远低于从内存中访问的速度。将常用数据挑出来，然后使其能够被快速访问，这样可以在很大程度上提高应用程序的性能。

ASP.NET 运行库中有一个公共语言运行库对象的字典(键/值映射)——Cache(缓存)。Cache 与应用程序的生命周期相同，可以通过 HttpContext 和 System.Web.UI.Page 访问。Cache 与 Session 的使用方法非常类似。可以使用索引器来访问其中的数据项。此外，可以控制缓存中对象的生命周期，甚至可以在已缓存的对象和物理数据源之间建立关联。为稍后练习如何使用缓存，下面先创建一个需要缓存数据的应用程序。

15.1 前期准备

ASP.NET 的缓存功能非常容易使用。可以为缓存项(cached item)设置过期时间和回调方法。这样，在缓存项被移除后，应用程序会得到相应的通知。下面的练习会创建一个利用缓存数据的应用程序。

➤ 创建一个利用缓存数据的应用程序

1. 创建一个“ASP.NET 空 Web 应用程序”，将其命名为 UseDataCaching。(也可以利用第 14 章创建的项目，因为这个项目会用到其中的数据库。)
2. 借用 UseDataList 页面。如果是新建的 Web 应用程序，则从第 14 章的项目复制该页面。为此，右键单击“解决方案资源管理器”的项目节点，选择“添加”|“现有项”。导航到第 14 章项目的位置，选择 UseDataList.aspx、UseDataList.aspx.cs 和 UseDataList.aspx.designer.cs 这 3 个文件，单击“添加”。这些文件会被复制到当前项目中。

刚刚导入的代码引用了 SessionState 项目中的数据库。可以将连接字符串改为指向该应用的 App_Data 目录中的数据库。当然，也可以指向位于其他位置的数据库。

3. 阅读一下 GetInventory、BindToInventory 和 Page_Load 方法的代码(见清单 15.1)。

清单 15.1 绑定库存数据的代码

```
protected DataTable CreateSelectedItemTable(DataTable tableSchema)
{
    DataTable tableSelectedItemData = new DataTable();

    foreach (DataColumn dc in tableSchema.Columns)
    {
        tableSelectedItemData.Columns.Add(dc.ColumnName,
            dc.DataType);
    }
    return tableSelectedItemData;
}

protected DataTable GetInventory()
{
    DataTable dt = null;

    dt = new DataTable();
    string strConnection =
        @"Data Source=
        .\SQLEXPRESS;
        AttachDbFilename=|DataDirectory|\ASPNETStepByStep4.mdf;
        Integrated Security=True;User Instance=True";
    DbProviderFactory f =
        DbProviderFactories.GetFactory("System.Data.SqlClient");
    using (DbConnection connection = f.CreateConnection())
    {
        connection.ConnectionString = strConnection;
        connection.Open();
        DbCommand command = f.CreateCommand();
        command.CommandText = "Select * from DotNetReferences";
        command.Connection = connection;

        IDataReader reader = command.ExecuteReader();
        dt.Load(reader);
        reader.Close();
        connection.Close();
    }
    return dt;
}

protected DataTable BindToInventory()
{
    DataTable dt;
    dt = this.GetInventory();
    this.DataList1.DataSource = dt;
    this.DataBind();
    return dt;
}
```

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        DataTable dt = BindToInventory();
        DataTable tableSelectedItems =
            this.CreateSelectedItemsTable(dt);
        Session["tableSelectedItems"] = tableSelectedItems;
    }
}
```

4. 运行这个页面，确保其能够正常工作。也就是说，它应能够连接数据库的 DotNetReferences 表，并将从该表获取的数据绑定到 DataList 控件上。

Page_Load 方法调用了 GetInventory 和 BindToInventory 方法。这种调用有多频繁？每次创建新页面，都会调用这两个方法(也就是说，每次发起对 UseDataList 页面的 HTTP 请求，这两个方法都会被调用)。如果这个应用程序在一台计算机上运行，并且只有一个客户端(测试环境)，每次请求都连接数据库并无大碍。然而，如果数以千计的用户频繁访问该页面，那么重复访问数据库的代价就不能忽视了。数据库的访问是非常耗资源的操作。稍后会看到，连接数据库和读取 DotNetReferences 表中记录的时间较长(对于计算机来说)。当数据库中的表增长后，访问数据会更耗资源。计算机半秒的处理过程中会进行亿万次的计算。

再让我们看看库存表本身。它的更新有多频繁？当然，练习中的应用程序过于简单。不妨设想一下实际的应用程序。库存中的商品可能不像其他数据那样更新那么频繁(并且更新的时机往往是有规律的、可预测的)。如果是这样，应用程序为什么每次加载页面都要连接数据库呢？这样做无疑得不偿失。如果能够将这些数据元素存储在访问速度比数据库快的介质中(例如，计算机的内存)，那么相比每次加载页面都访问数据库，网站将能够处理更多请求。这种情况非常适合将数据缓存。(这里应当注意的是，如果库存数据变动频繁，如每 5 秒就变动一次，那么这种数据则不适合缓存。)

15.2 数据缓存的使用

一般情况下，ASP.NET 中的 Cache 对象与 Session 对象的访问方式非常类似。请不要忘记，访问 Session 对象可以使用索引器(方括号语法)，通过一致的索引来存储和获取数据。数据缓存也是一样(但它还有管理缓存项的功能)。

数据缓存策略一般包含以下 4 个步骤。

1. 在缓存中查找所需的数据元素
2. 如果已存在，则使用它(从而避免了耗资源的数据库操作)
3. 如果缓存中不存在这个数据元素，则需要从数据库获取
4. 在取得该数据后，将其添加到缓存中，下次便可以从缓存获取

下面的练习将修改 UseDataList 页面，在首次使用数据项时，将其存储在缓存中。虽然 Page_Load 首次被调用时会花费一定的时间(对于计算机时间概念来说)，但后续的访问则会加快很多。

➤ 使用缓存

1. 打开 UseDataList.aspx.cs 文件，找到 GetInventory 方法。
2. 使这个方法利用缓存非常简单。首先，检查所需的 DataSet 是否存在于缓存中。如果在缓存中找到该数据项(DataSet)，则会得到一个有效的对象引用，之后便可以绕过查询数据库的代码，并直接获得这个 DataSet。如果查询缓存后返回的结果是 null，则需要查询数据库。在数据库查询完成后，则会得到所需的 DataSet。在将其引用返回给调用者之前，先将其缓存。使用 Trace 语句有助于观察缓存带来的变化。下面代码中加粗的部分是需要修改的：

```
protected DataTable GetInventory()
{
    DataTable dt = null;

    Trace.Warn("Page_Load", "looking in cache");
    dt = (DataTable)Cache["InventoryDataTable"];
    Trace.Warn("Page_Load", "done looking in cache");
    if (dt == null)
    {
        Trace.Warn("Page_Load", "Performing DB lookup");
        dt = new DataTable();

        string strConnection =
            @"Data Source=
            .\SQLEXPRESS;
            AttachDbFilename=|DataDirectory|ASPNETStepByStep4.mdf;
            Integrated Security=True;
            User Instance=True";
        DbProviderFactory f =
            DbProviderFactories.GetFactory("System.Data.SqlClient");

        using (DbConnection connection = f.CreateConnection())
        {
            connection.ConnectionString = strConnection;
            connection.Open();
            DbCommand command = f.CreateCommand();
            command.CommandText = "Select * from DotNetReferences";
            command.Connection = connection;

            IDataReader reader = command.ExecuteReader();
            dt.Load(reader);
            reader.Close();
            connection.Close();
        }
    }
}
```



```
Cache["InventoryDataTable"] = dt;
Trace.Warn("Page_Load", "Done performing DB lookup");
}
return dt;
}
```

这段代码极大地降低了加载页面的开销(当然，要在数据添加到缓存之后)。当页面再次被加载时，则可以使用缓存中的数据——使用 Cache 中的数据可以极大地降低开销。与不缓存的差距有多少？是非常大的。下一节让我们通过应用程序的跟踪页面来做个实验。

15.3 缓存的影响

通过在 GetInventory 方法中添加 Trace 语句，可以在跟踪页面看到缓存数据后的效果。UseDataCaching 应用程序页面的 Trace 属性默认是被禁用的，但可以启用应用程序级的跟踪。为此，在 Web.config 中添加以下设置：

```
<configuration>
  <system.web>
    <trace enabled="true"/>
  </system.web>
</configuration>
```

为查看这种跟踪信息，可以浏览虚拟目录下名为 Trace.axd 的文件。为此，先浏览 UseDataList.aspx，再浏览同目录下的 Trace.axd。

图 15.1 展示了首次访问页面后跟踪语句生成的结果。最右侧一列的数据是与执行的上一条跟踪语句所间隔的时间。这些数据表明，页面加载至少花费了 0.316 秒。



图 15.1 访问数据库花费了近半秒钟

使 UseDataList 页面进行几次回发(例如，从库存向选定项列表添加几件商品)。然后，再查

看这些回发(POST)的跟踪信息。图 15.2 展示了此时的跟踪语句。从 Cache 获取数据要比访问数据库快得多——这种差距达数千倍之多！如果只有一个客户端不时地浏览页面，效果可能并不明显。然而，如果多个客户端同时浏览这个页面，那么相比每次都访问数据库，使用缓存的响应速度要快很多。



图 15.2 从缓存读取数据只耗费了 0.000057 秒

15.4 缓存的管理

上一练习选择了最简单的方式来缓存数据——只通过索引将数据保存在缓存中。然而，有时需要对缓存项进行更多控制。例如，某个缓存项的源被更新该怎么办？如果有必要为用户提供准确的信息，则需要在更改发生时获得通知，以便做相应的处理(也许是将信息重新加载到缓存中)。又如，某段时间过后，或者到达某个时刻后，缓存中的信息变为无效，此时该怎么办？你可能需要确认缓存中的这部分数据是无效的，然后用新数据对其进行刷新(正如下面的示例将要介绍的)。

除了使用索引器来添加缓存项之外，Cache 对象还提供了一个带参数的重载方法——Insert。我们可以使用它来从多个方面控制缓存项，其中包括：

- 设置绝对过期时间(absolute expiration time)
- 设置可调过期时间(sliding expiration time)
- 设置缓存项的依赖项(如数据库、文件和目录依赖项，甚至缓存项之间的依赖项)
- 管理缓存项的相对失效优先级
- 设置在缓存项被移除时调用的回调方法

Cache 的 Insert 方法包含 4 个重载(见表 15.1)。

表 15.1 Cache.Insert 方法的重载

Insert 方法的重载	说 明
Insert(String, Object)	该方法与索引器的设置方法等价，该方法仅通过第一个参数提供的键将 Cache 中的对象替换
Insert(String, Object, CacheDependency)	该方法用于在 Cache 中插入对象并将其与一个依赖项关联
Insert(String, Object, CacheDependency, DateTime, TimeSpan)	该方法用于在 Cache 中插入对象并将其与一个依赖项和过期策略关联
Insert(String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority, CacheItemRemovedCallback)	该方法用于在 Cache 中插入对象并将其与一个依赖项和过期策略关联。还可以通过该方法将缓存项与一个回调委托关联，以便在缓存项被移除后通知应用程序

后面的练习会演示这些设置的使用。此外，接下来的示例将演示 DataTable 和 DataSet 的另一种使用方法。事实上，我们能够以编程方式创建它们。后面有几个示例中的 DataTable 就是通过程序创建的，而不是从数据库获取的。虽然使用程序创建的 DataTable 不能明显突出缓存的效果，但仍可以看到缓存前后的差别(同时还能够了解到另一种管理数据的方法)。下一节还会介绍如何将 DataTable 序列化为 XML(为了观察缓存项与文件依赖项)。

15.4.1 内存中的 DataSet

第 10 章介绍了如何通过访问数据库来获取适合绑定到控件上的数据。上一章介绍了如何通过 Session 对象来维护请求之间的数据。Session 对象能够存储任何可序列化的 .NET CLR 对象——其中包括 DataReader。然而，长时间存储 DataReader 并不明智，因为那意味着使连接长时间处于打开状态。打开过多的连接最终将导致应用程序的性能下降。最好是进行一次数据库访问，然后保存得到的 DataTable 或 DataSet。

除了从数据库获取 DataTable 外，我们还能够以编程方式填充 DataTable。为此，需要创建一个 DataTable，然后添加 DataColumn 来描述架构。创建 DataTable 就绪后，可以创建所需类型的列，然后将其添加到表的列集合中。清单 15.2 给出了在内存中创建 DataTable 的代码。(清单 15.1 与之前章节介绍的方法都能够创建 DataTable。当时没有提及这种方法是因为本节要讨论它。)这个表包含一些名言及出处。我们将在后面的练习中用到这个数据表。

清单 15.2 QuotesCollection 对象

```
public class QuotesCollection : DataTable
{
    public QuotesCollection()
    {
        //
        // TODO: Add constructor logic here
    }
}
```


ASP.NET 4从入门到精通

```
//
}

public void Synthesize()
{
    // Be sure to give a name so that it will serialize as XML
    this.TableName = "Quotations";
    DataRow dr;

    Columns.Add(new DataColumn("Quote", typeof(string)));
    Columns.Add(new DataColumn("OriginatorLastName",
        typeof(string)));
    Columns.Add(new DataColumn("OriginatorFirstName",
        typeof(string)));

    dr = this.NewRow();
    dr[0] = "Imagination is more important than knowledge.";
    dr[1] = "Einstein";
    dr[2] = "Albert";
    Rows.Add(dr);

    dr = this.NewRow();
    dr[0] = "Assume a virtue, if you have it not";
    dr[1] = "Shakespeare";
    dr[2] = "William";
    this.Rows.Add(dr);

    dr = this.NewRow();
    dr[0] = @"A banker is a fellow who lends you his umbrella
        when the sun is shining, but wants it back the
        minute it begins to rain.";
    dr[1] = "Twain";
    dr[2] = "Mark";
    this.Rows.Add(dr);

    dr = this.NewRow();
    dr[0] = @"A man cannot be comfortable without his own
        approval.";
    dr[1] = "Twain";
    dr[2] = "Mark";
    this.Rows.Add(dr);

    dr = this.NewRow();
    dr[0] = "Beware the young doctor and the old barber";
    dr[1] = "Franklin";
    dr[2] = "Benjamin";
    this.Rows.Add(dr);

    dr = this.NewRow();
    dr[0] = @"Reality is merely an illusion, albeit a
        very persistent one.";
```

```
dr[1] = "Einstein";
dr[2] = "Albert";
this.Rows.Add(dr);

dr = this.NewRow();
dr[0] = "Beer has food value, but food has no beer value";
dr[1] = "Sticker";
dr[2] = "Bumper";
this.Rows.Add(dr);

dr = this.NewRow();
dr[0] = @"Research is what I'm doing when I don't know
        what I'm doing";
dr[1] = "Von Braun";
dr[2] = "Wernher";
this.Rows.Add(dr);

dr = this.NewRow();
dr[0] = "Whatever is begun in anger ends in shame";
dr[1] = "Franklin";
dr[2] = "Benjamin";
this.Rows.Add(dr);

dr = this.NewRow();
dr[0] = "We think in generalities, but we live in details";
dr[1] = "Whitehead";
dr[2] = "Alfred North";
this.Rows.Add(dr);

dr = this.NewRow();
dr[0] = "Every really new idea looks crazy at first.";
dr[1] = "Whitehead";
dr[2] = "Alfred North";
this.Rows.Add(dr);

dr = this.NewRow();
dr[0] = @"The illiterate of the 21st century will not be
        those who cannot read and write, but
        those who cannot learn,
        unlearn, and relearn.";
dr[1] = "Whitehead";
dr[2] = "Alfred North";
this.Rows.Add(dr);
}
}
```

在内存中构建 **DataTable** 相对容易——只不过是定义列架构，然后向表中添加行。这个类可以在本书配套的示例代码中找到，因而不必人工输入。在进行后面的练习时，将其导入即可。

下面我们来看看如何管理缓存。

15.4.2 缓存过期

为缓存项设置超时阈值是管理缓存项的方法之一。在某些情况下，可能希望对缓存的数据进行计时。Cache 对象支持两种计时方式——“绝对过期” (absolute expiration) 和“可调过期” (sliding expiration)。

➤ 设置绝对过期

- 1. 为实践绝对过期的使用，在 UseDataCaching 项目中添加一个新页面，将其命名为 CacheExpirations.aspx。
- 2. 单击“项目”|“添加现有项”，将示例代码中的 CacheExpirations 文件导入。
- 3. 拖放一个 GridView 控件到 CacheExpirations 页面(如下图所示)。先不要将其绑定到数据源，稍后会在 Page_Load 方法中以编程方式绑定数据。



- 4. 在 CacheExpirations 页面的 Page_Load 方法中，检查缓存中是否存在 QuotesCollection 对象的实例(与上一个练习的做法相同)。如果该数据表不存在，则创建 QuotesCollection 类的实例并调用其 Synthesize 方法来填充它。最后，通过 Insert 方法的重载，将这个表添加到缓存中。可以使用 DateTime 类来生成绝对过期时间。将 QuotesCollection 对象绑定到 GridView。将缓存策略设置为 Cache.NoSlidingExpiration。添加跟踪语句，以便观察绝对过期策略的效果。

```
Protected void Page_Load(object sender, EventArgs e)
{
    QuotesCollection quotesCollection;

    DateTime dtCurrent = DateTime.Now;
```



```
Trace.Warn("Page_Load",
    "Testing cache at: " +
    dtCurrent.ToString());
quotesCollection = (QuotesCollection)Cache["QuotesCollection"];

if (quotesCollection == null)
{
    quotesCollection = new QuotesCollection();
    quotesCollection.Synthesize();

    DateTime dtExpires = new DateTime(2008, 5, 31, 23, 59, 59);
    dtCurrent = DateTime.Now;

    Trace.Warn("Page_Load",
        "Caching at: " +
        dtCurrent.ToString());
    Trace.Warn("Page_Load",
        "This entry will expire at: " +
        dtExpires);
    Cache.Insert("QuotesCollection",
        quotesCollection,
        null,
        dtExpires,
        System.Web.Caching.Cache.NoSlidingExpiration,
        System.Web.Caching.CacheItemPriority.Default,
        null);
}

this.GridView1.DataSource = quotesCollection;
this.DataBind();
}
```

5. 适当修改日期和时间，看看过期时间能否强制缓存重新加载。

绝对过期时间会使 ASP.NET 在指定的时间过后将缓存项移除。下面我们实验另一种过期策略——可调过期。使用可调过期后，ASP.NET 会在指定的一段时间后移除缓存。这段时间内没有被访问的缓存项会过期。^①

➤ 使用可调过期

1. 为给缓存项设置可调过期策略，需修改 CacheExpirations 页面的 Page_Load 方法。设置可调过期只需要调整 Insert 方法的参数。创建一个 TimeSpan，在这段时间过后缓存项会过期。将绝对过期时间设置为 DateTime.MaxValue，将这个 TimeSpan 对象作为最后的参数传入，如下所示：

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

^① 译者注：可调过期是从最后一次访问缓存项的时刻开始计时，指定的一段时间过后，该缓存项便会过期。

```
{
    QuotesCollection quotesCollection;

    DateTime dtCurrent = DateTime.Now;
    Trace.Warn("Page_Load",
        "Testing cache: " + dtCurrent.ToString());

    quotesCollection =
        (QuotesCollection)Cache["QuotesCollection"];

    if (quotesCollection == null)
    {
        quotesCollection = new QuotesCollection();
        quotesCollection.Synthesize();

        TimeSpan tsExpires = new TimeSpan(0, 0, 15);
        dtCurrent = DateTime.Now;

        Trace.Warn("Page_Load",
            "Caching at: " + dtCurrent.ToString());
        Trace.Warn("Page_Load",
            "This entry will expire in: " +
            tsExpires.ToString());
        Cache.Insert("QuotesCollection",
            quotesCollection,
            null,
            DateTime.MaxValue,
            tsExpires);
    }

    this.GridView1.DataSource = quotesCollection;
    this.DataBind();
}
```

2. 运行这个页面。如果在指定的时间段内没有访问这个缓存项，那么缓存会被重新加载。

缓存依赖项是另一种管理缓存项的方法。下面让我们来认识几种缓存依赖项。

15.4.3 缓存依赖项

除了通过时间来使缓存项过期，还可以为缓存项设置依赖项。例如，假设应用程序从文件中加载某些数据，并将这些数据置于缓存中。如果后端文件(即缓存项的源)被更新，那么缓存中的数据就失效了。ASP.NET 能够在缓存项和文件之间建立依赖关系，修改文件会使缓存项失效。依赖项使缓存项失效的条件包括文件和目录的变更、其他缓存项被移除，或者 Microsoft SQL Server 中的表被更新(该功能是 ASP.NET 2.0 之后开发者最希望引入的功能之一)。

下面让我们看看如何设置缓存依赖项。

➤ 设置缓存依赖项

- 1. 为 UseDataCache 项目添加一个页面，将其命名为 CacheDependencies.aspx。
- 2. 在页面上添加一个 GridView 控件，再添加一个按钮，将这个按钮的 ID 设置为 ButtonSaveAsXML。我们将用这个按钮来使页面根据 QuotesCollection 生成 XML 文件。此时的页面如下图所示。



- 3. 双击这个按钮，为其添加一个事件处理程序。在这个处理程序中，将 DataTable 以 XML 架构(XSD 文件)和 XML 数据(XML 文件)的形式保存到 App_Data 目录下。
- 4. 在这个处理程序中，初始化一个 QuotesCollection 对象，调用其 Synthesize 方法来为自身填充数据。这个页面有一个 Server 对象的引用。可以调用 Server 对象的 MapPath 方法来获得保存文件的物理路径。然后使用这个路径来创建 XML 文件和架构文件。为此，分别调用 DataTable 的 WriteXmlSchema 和 WriteXml 方法即可。

```
protected void ButtonSaveAsXML_Click(object sender, EventArgs e)
{
    QuotesCollection quotesCollection = new QuotesCollection();
    quotesCollection.Synthesize();
    String strFilePathXml =

    Server.MapPath(Request.ApplicationPath +
    "\\App_Data\\QuotesCollection.xml");
    String strFilePathSchema =
    Server.MapPath(Request.ApplicationPath +
    "\\App_Data\\QuotesCollection.xsd");
    quotesCollection.WriteXmlSchema(strFilePathSchema);
    quotesCollection.WriteXml(strFilePathXml);
}
```

- 5. 编写一个方法，将 XML 加载到 QuotesCollection 对象中，并缓存该对象。可以使用

XML 文件的路径来创建相应的依赖项。如果该文件被更新，ASP.NET 会将对应的缓存项移除。传入 `Cache.NoAbsoluteExpiration` 和 `Cache.NoSlidingExpiration` 来禁用绝对过期和可调过期策略。为了观察将文件加载到缓存后更新该文件而产生的效果，可以添加跟踪语句。最后，将 `QuotesCollection` 绑定到 `GridView` 控件上。

```
protected void CacheWithFileDependency()
{
    QuotesCollection quotesCollection;

    Trace.Warn("Page_Load", "Testing cache ");
    quotesCollection = (QuotesCollection)Cache["QuotesCollection"];

    if (quotesCollection == null)
    {
        Trace.Warn("Page_Load", "Not found in cache");
        quotesCollection = new QuotesCollection();

        String strFilePathXml =
            Server.MapPath(Request.ApplicationPath +
                "\\App_Data\\QuotesCollection.xml");
        String strFilePathSchema =
            Server.MapPath(Request.ApplicationPath +
                "\\App_Data\\QuotesCollection.xsd");

        quotesCollection.ReadXmlSchema(strFilePathSchema);
        quotesCollection.ReadXml(strFilePathXml);

        System.Web.Caching.CacheDependency cacheDependency =
            new System.Web.Caching.CacheDependency(strFilePathXml);

        Cache.Insert("QuotesCollection",
            quotesCollection,
            new
                System.Web.Caching.CacheDependency(strFilePathXml),
                System.Web.Caching.Cache.NoAbsoluteExpiration,
                System.Web.Caching.Cache.NoSlidingExpiration,
                System.Web.Caching.CacheItemPriority.Default,
                null);
    }

    this.GridView1.DataSource = quotesCollection;
    this.DataBind();
}
```

6. 在 `Page_Load` 方法中调用 `CacheWithFileDependency()`。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        ButtonSaveAsXML_Click(null, null);
    }
}
```

```

    }
    CacheWithFileDependency();
}

```

7. 运行这个页面。该页面会加载 XML 数据和架构到 QuotesCollection 中，将 QuotesCollection 对象保存在缓存中，并通过网格来显示数据。单击“Generate XML File”按钮来刷新 XML 文件(缓存依赖项所监视的文件)。由于磁盘上的文件被更新，因此 ASP.NET 会移除相应的缓存项。当页面再次被加载时，新的 XML 文件会被加载到缓存中。

下面让我们了解一下最后一种缓存依赖项——SQL Server 依赖项。

15.4.4 SQL Server 依赖项

ASP.NET 1.0 的缓存依赖项具有一些不可回避的缺陷。它缺少一种最常用的依赖项，即能够在来自 SQL Server 的缓存项与物理数据库之间建立依赖关系的 SQL Server 依赖项。由于许多网站将 SQL Server 作为 DataGrid 和其他控件的后端数据提供程序，因此实现这种依赖项无疑是管理缓存项的最有效的方式。

为启用 SQL Server 依赖项，首先可以使用 aspnet_regsql.exe 来配置 SQL Server。依赖项可以在配置文件中描述，然后将依赖项的名称传给 SqlCacheDependency 的构造函数。SqlCacheDependency 类能够监视指定的表。如果某些操作更新了这个表，那么 ASP.NET 则会从 Cache 移除相应的缓存项。

清单 15.3 给出了一个 SQL Server 依赖项的配置示例。

清单 15.3 配置文件中的 SQL Server 缓存依赖项设置

```

<キャッシング>
  <sqlCacheDependency enabled="true">
    <databases>
      <add name="DBName" pollTime="500"
        connectionStringName="connectionString"/>
    </databases>
  </sqlCacheDependency>
</キャッシング>

```

清单 15.4 展示了一个 ASP.NET 页面，该页面从 SQL Server 数据库加载数据，并在数据库和缓存项之间建立依赖关系。

清单 15.4 使用 SqlCacheDependency 的页面

```

<%@Page Language="C#" %>
<%@Import namespace="System.Data"%>
<%@Import namespace="System.Data.SqlClient"%>
<script runat="server">
    protected void Page_Load(Object sender, EventArgs e)
    {

```

```
DataSet ds = null;
ds = (DataSet)Cache["SomeData"];
if (ds == null)
{
    string connectionString =
        ConfigurationManager.ConnectionStrings["connectionString"]
            .ConnectionString;
    SqlDataAdapter da =
        new SqlDataAdapter("select * from DBName.tableName",
            connectionString);
    ds = new DataSet();
    da.Fill(ds);
    SqlCacheDependency sqlCacheDependency =
        new SqlCacheDependency("DBName", "tableName");
    Cache.Insert("SomeData",
        ds,
        sqlCacheDependency);
}
GridView1.DataSource = ds;
DataBind();
}
</script>
<html>
<body>
    <form id="form1" runat="server">
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
    </form>
</body>
</html>
```

一旦数据被添加到缓存，并且其生命周期受过期策略和缓存依赖项控制，那么就只剩下一项管理工作了——在缓存项被移除时做出相应的反应。

15.4.5 缓存项的清除

通过之前的示例可知，ASP.NET 支持通过以下 3 种方式移除缓存项：

- 显式地调用 `Cache.Remove`
- 由于内存占用而移除优先级较低的缓存项
- 移除过期的缓存项

`Insert` 方法的重载之一接受一个回调委托参数。ASP.NET 能够通过这个委托来通知缓存项已被移除。为接收通知，只需要实现一个与其签名一致的(回调)方法，用委托将其包装，并将这个委托传入 `Insert` 方法。当缓存项被移除后，ASP.NET 会调用这个回调方法。

下面的练习演示了这种移除回调。

➤ 实现移除回调

1. 实现移除回调的主要任务之一是找到一个适合放置回调的位置。如果把它作为 Page 类的普通实例成员会怎样？这是行不通的——在页面的生命周期过后，回调将会失效。回调应一直保持可用状态。（可以以静态方法的形式实现它。）最适合承载这种回调方法的类是全局应用程序类。第 18 章会详细介绍应用程序类，这里暂时不做说明，只在项目中直接添加一个即可。选择“项目”|“添加新项”。选择“全局应用程序类”模板，单击“添加”将其添加到项目中。Visual Studio 会在应用程序中添加一个名为 Global.asax 的文件。
2. Global.asax.cs 包含应用程序范围的代码。在 Global.asax.cs 文件中编写一个处理回调的方法。在这个示例中，这个方法将设置一个缓存失效的标志。Application_BeginRequest 处理程序中的代码会将数据重新添加到缓存中，其行为与之前提到的 CacheWithFileDependency 方法十分类似。Cache 对象可以通过当前的 HttpContext 获得。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;
using System.Web.Caching;

namespace UseDataCaching
{
    public class Global : System.Web.HttpApplication
    {
        bool _bReloadQuotations = false;
        public void OnRemoveQuotesCollection(string key, object val,
            CacheItemRemovedReason r)
        {
            // Do something about the dependency Change
            if (r == CacheItemRemovedReason.DependencyChanged)
            {
                _bReloadQuotations = true;
            }
        }

        protected void ReloadQuotations()
        {
            QuotesCollection quotesCollection = new QuotesCollection();
            String strFilePathXml =
                Server.MapPath(HttpContext.Current.Request.ApplicationPath +
                    "\\App_Data\\QuotesCollection.xml");
            String strFilePathSchema =
                Server.MapPath(HttpContext.Current.Request.ApplicationPath +
```

```

        "\\App_Data\\QuotesCollection.xsd");

quotesCollection.ReadXmlSchema(strFilePathSchema);
quotesCollection.ReadXml(strFilePathXml);

System.Web.Caching.CacheDependency cacheDependency =
    new System.Web.Caching.CacheDependency(strFilePathXml);

HttpContext.Current.Cache.Insert("QuotesCollection",
    quotesCollection,
    cacheDependency,
    System.Web.Caching.Cache.NoAbsoluteExpiration,
    System.Web.Caching.Cache.NoSlidingExpiration,
    System.Web.Caching.CacheItemPriority.Default,
    this.OnRemoveQuotesCollection);
}

protected void Application_BeginRequest(object sender, EventArgs e)
{
    if (_bReloadQuotations == true)
    {
        ReloadQuotations();
        _bReloadQuotations = false;
    }
}

// VS-provided code
}
}

```

3. 修改 CacheWithFileDependency 方法，使其在向缓存添加 QuotesCollection 对象时传入回调方法。可以通过 HttpContext.Current.ApplicationInstance 来访问刚刚实现的回调方法。

```

protected void CacheWithFileDependency()
{
    QuotesCollection quotesCollection;
    Trace.Warn("Page_Load", "Testing cache ");
    quotesCollection = (QuotesCollection)Cache["QuotesCollection"];

    if (quotesCollection == null)
    {
        Trace.Warn("Page_Load", "Not found in cache");
        quotesCollection = new QuotesCollection();

        string strFilePathXml =
            Server.MapPath(Request.ApplicationPath +
                "\\App_Data\\QuotesCollection.xml");
        string strFilePathSchema =
            Server.MapPath(Request.ApplicationPath +
                "\\App_Data\\QuotesCollection.xsd");
    }
}

```

```
quotesCollection.ReadXmlSchema(strFilePathSchema);
quotesCollection.ReadXml(strFilePathXml);

System.Web.Caching.CacheDependency cacheDependency =
    new System.Web.Caching.CacheDependency(strFilePathXml);

Global global = HttpContext.Current.ApplicationInstance as Global;
Cache.Insert("QuotesCollection",
    quotesCollection,
    cacheDependency,
    System.Web.Caching.Cache.NoAbsoluteExpiration,
    System.Web.Caching.Cache.NoSlidingExpiration,
    System.Web.Caching.CacheItemPriority.Default,
    global.OnRemoveQuotesCollection);
}
this.GridView1.DataSource = quotesCollection;
this.DataBind();
}
```

运行 CacheDependencies 页面后不难发现，页面本身从不会刷新缓存。因为在 XML 文件被更新后，ASP.NET 会立即调用全局对象的 ReloadQuotations 方法(该方法会重新加载缓存)。

15.5 快速参考

目 标	操 作
访问数据缓存	数据缓存可以通过页面的 Cache 属性和当前 HttpContext 的 Cache 属性访问
在缓存中插入数据	通过索引符号向缓存插入对象或值
在缓存中插入带依赖项的数据	创建 CacheDependency 对象，通过 Cache.Insert 方法重载来一并添加缓存项到这个对象
在缓存中插入带过期策略的数据	创建 DateTime 对象，通过 Cache.Insert 方法重载来添加缓存项到这个对象
删除缓存项	调用 Cache.Remove 方法
在缓存项被移除后获得通知	在添加缓存项时传入回调委托

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 16 章

输出缓存

学习目标

- 缓存页面的内容
- 通过缓存输出来改善 Web 应用程序的性能
- 通过 OutputCache 指令来管理缓存的内容
- 通过 HttpCachePolicy 类来管理缓存的内容

本章将介绍 ASP.NET 对输出的缓存。第 15 章演示了缓存数据给应用程序性能带来的积极影响。通过避免不必要的数据库操作，可以在很大程度上改善网站中部分内容的性能。除了缓存应用程序数据外，ASP.NET 还支持“输出缓存”。

如果了解一下页面呈现过程就会发现，其中涉及很多处理。从页面被加载到最后一个闭标签被发送到浏览器，这之间涉及许多内容。页面可能要访问数据库，可能包含许多控件，还可能包含像 DataList 和 GridView 这样呈现过程复杂的控件。处理所有这些元素通常要花费较长时间。

正如可以通过在内存中缓存数据来避免不必要的数据库操作一样，可以使 ASP.NET 绕过整个页面呈现过程，将已被呈现过的内容直接发送给客户端。这就是所谓的“输出缓存”(output caching)。

16.1 页面内容的缓存

在网上，我们可以看到各种页面。有些网站对内容的更新十分频繁，而有些则不然。有些页面只更新局部内容。如果遇到更新并不频繁的页面，则可以通过缓存输出来避免为每个请求重新生成页面。

在最简单的情况下，启用输出缓存非常简单——在页面中添加 OutputCache 指令(directive)即可。OutputCache 是一个单独的指令(这一点与 Page 指令一样)，可以通过它来启用缓存，并可以控制缓存的大部分行为。下面我们通过练习来实际认识一下输出缓存。

➤ 创建可缓存的页面

1. 创建一个“ASP.NET 空 Web 应用程序”，将其命名为 OutputCaching。
2. 添加一个名为 Default.aspx 的“Web 窗体”。Visual Studio 会在创建后自动打开这个文

件。暂时将 Page 指令的 Trace 特性设置为 false。(稍后在缓存用户控件时会启用它。)在页面的顶部，Page 指令的后面添加 OutputCache 指令。我们至少要为 OutputCache 指令添加两个特性——Duration 和 VaryByParam。Duration 特性用于指定内容被缓存的时长(以秒为单位)。VaryByParam 特性用于指定是否缓存页面的多个版本，这里将它设置为 none。稍后将更详细地讲解这两个特性。下面这段代码展示了 OutputCache 指令的语法。该页面会将内容缓存 15 秒。OutputCache 指令后面的代码是 Visual Studio 生成的：

```
<%@Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="-Default" Trace="false"%>
<%@OutputCache Duration="15" VaryByParam="none"%>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title> Untitled Page </title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            </div>
        </form>
    </body>
</html>
```

3. 修改 Page_Load 方法，使其输出页面生成的日期和时间。如下所示：

```
public partial class Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Response.Write("This page was generated and cached at: " +
            DateTime.Now.ToString());
    }
}
```

当页面首次被请求时，Page_Load 方法会被执行并生成以下内容：



从页面首次被加载后的 15 秒内，不论刷新多少次，ASP.NET 都会获取并显示缓存的

内容。15 秒过后，ASP.NET 会按常规方式运行页面，调用 `Page_Load`，重新呈现内容，并将新内容缓存。下图是页面首次被访问后不久(15 秒内)刷新后的结果。此时的日期和时间与之前的页面完全相同，尽管这是全新的请求(两张截图是两个不同请求获得的结果)：



4. 为说明输出缓存对性能有多少积极影响，在 `Page_Load` 方法中添加一行代码，使当前线程休眠 10 秒(以此来模拟大量内容的生成过程)。为使用线程，需要引用 `System.Threading` 命名空间：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Threading;

namespace OutputCaching
{
    public partial class Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Thread.Sleep(10000);
            Response.Write("This page was generated and cached at: " +
                DateTime.Now.ToString());
        }
    }
}
```

5. 运行这个页面。页面的首次加载会花费很长时间(大约 10 秒钟)。立即刷新会发现该页面立即显示其内容(没有很长的延迟)。大多数页面的加载不会花费这么长的时间，这里只是为了说明缓存内容能够有效地改善 Web 应用程序的性能。对于呈现耗资源且更新又相对不频繁的页面，缓存其内容可以有效改善网站的性能，尤其是在客户端的数目增加时。

16.2 缓存内容的管理

在某些情况下，只需要在页面中添加 `OutputCache` 指令便可以缓存页面的参数。然而，有时需要对输出缓存进行一定的控制。ASP.NET 提供了许多用于管理这种缓存的参数。修改 `OutputCache` 指令的参数或 `Response` 对象的 `HttpCachePolicy` 属性，都能改变输出缓存的行为。

16.2.1 `OutputCache` 指令的使用

对输出缓存的管理通常非常有用。例如，如果页面对所有用户显示的内容都是相同的，那么缓存输出的一个版本即可。然而，有时不适合为不同用户提供相同的内容。控制输出缓存最简单的方式是修改 `OutputCache` 指令。

为不同浏览器的请求提供不同版本的内容是控制输出缓存的重要作用之一。不同的浏览器一般具有不同功能。如果页面中包含并非所有浏览器都支持的功能，那么有些浏览器发送请求后，可能会得到无法完全被正确处理的响应。使用 `OutputCache` 指令的 `VaryByCustom` 参数，我们可以对不同类型浏览器缓存不同的内容。

如果页面呈现的内容会由于查询字符串中参数的不同而不同，那么也需要相应地控制输出缓存。例如，假定某页面要求用户在一个文本框中输入其姓名，并通过查询字符串的参数发送给服务器。则可以使输出缓存根据查询字符串中参数的不同而缓存页面的不同版本。例如，名为“John Doe”的用户所获得的缓存内容的版本，将与名为“Jane Smith”的用户获得的版本不同。`VaryByParam` 属性可以控制这种行为。

表 16.1 列出了所有参数。^①

表 16.1 `OutputCache` 指令的参数

属 性	有 效 值	说 明
CacheProfile	字符串	配置文件的名称(在 <code>Web.config</code> 中定义)。默认值为空字符串
Duration	整数	页面或用户控件被缓存的时间(以秒为单位)
NoStore	true false	用于指定是否发送“no store”缓存控制标头(用于禁用敏感信息的二级存储)。对用户控件不适用。默认值为 false

① 译者注：每添加一个缓存变量相当于在输出缓存版本的创建和选择上增加了一个维度。

续表

属 性	有 效 值	说 明
Location	Any Client Downstream Server None	用于指定发送给客户端的标头和元数据(标签)，每种设置的具体含义如下： Any——允许在任意位置缓存输出(默认值) Client——只允许浏览器缓存输出 Downstream——允许下游服务器和客户端缓存输出 Server——只允许在服务器上缓存 None——禁用缓存
Shared	true false	用于指定用户控件的输出是否可以由多个页面共享
SqlDependency	数据库/表的名称对	用于指定输出缓存所依赖的数据库和表的名称对
VaryByContentEncoding	编码类型列表	可以通过逗号分隔的字符串列表来指定客户端可能提交的编码类型，输出缓存会据此区分缓存内容的版本
VaryByCustom	browser 自定义字符串	ASP.NET 会根据浏览器的名称和版本，或者自定义的字符串来缓存不同的版本，必须在 Global.asax 中重写 GetVaryByCustomString 来进行处理
VaryByHeader	* 标头名	用于通过由分号分隔的字符串列表来指定客户端可能提交的标头。用户控件不支持此属性。默认值为空字符串(无标头)
VaryByParam	None * 参数名	可以通过由分号分隔的字符串列表来指定 GET 请求中的查询字符串或 POST 请求中的变量

下面的练习将演示如何根据不同的用户姓名来缓存不同版本的内容。

➤ 根据查询字符串参数来区分缓存的内容

1. 回到 OutputCache 项目，为 default.aspx 页面添加一个 TextBox 和一个 Button。将 TextBox 的 ID 设置为 TextBoxName，将 Button 的 ID 设置为 ButtonSubmitName。TextBoxName 用于输入用户的姓名，该参数将控制缓存的页面版本数目。
2. 双击按钮，为其添加一个 Click 事件的处理程序。在这个处理程序中，使用文本框的内容显示一条问候语来作为对用户请求的响应。另外，修改页面加载时间，减少当前线程睡眠的时间(或直接将其注释掉)：

```
public partial class Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Thread.Sleep(0);
        Response.Write("This page was generated and cached at: " +
```



```

        DateTime.Now.ToString());
    }

    protected void ButtonSubmitName_Click(object sender, EventArgs e)
    {
        Response.Write("<br><br>");
        Response.Write("<h2> Hello there, " +
            this.TextBoxName.Text + "</h2>");
    }
}

```

3. 延长页面缓存的时间(这里设置为 1 分钟), 以便修改 TextBox 的内容并查看缓存后的效果。另外, 在 OutputCache 指令中, 将 VaryByParam 设置为 TextBoxName, 以便将该参数作为区分内容的依据。

```

<%@Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="-Default"
Trace="false"%>

<%@OutputCache Duration="60" VaryByParam="TextBoxName"%>

```

4. 在 TextBox 和 Button 控件后面添加一个 Substitution 控件(可以从“工具箱”拖放一个到页面上)。Substitution 控件能够调用代码隐藏类中的方法来显示任意字符串。这里用 Substitution 控件来显示请求的时间, 以便将其与页面缓存的时间做比较。在代码隐藏类中编写以下方法来处理这个控件:

```

public partial class Default : System.Web.UI.Page
{
    // Existing code ...
    protected static string SubstituteDateAndTime(HttpContext c)
    {
        return "Request occurred at :" + DateTime.Now;
    }
}

```

5. 在 ASPX 文件中, 为 Substitution 控件添加一个 MethodName 属性, 将其设置为 SubstituteDateAndTime 方法(如下所示)。

```

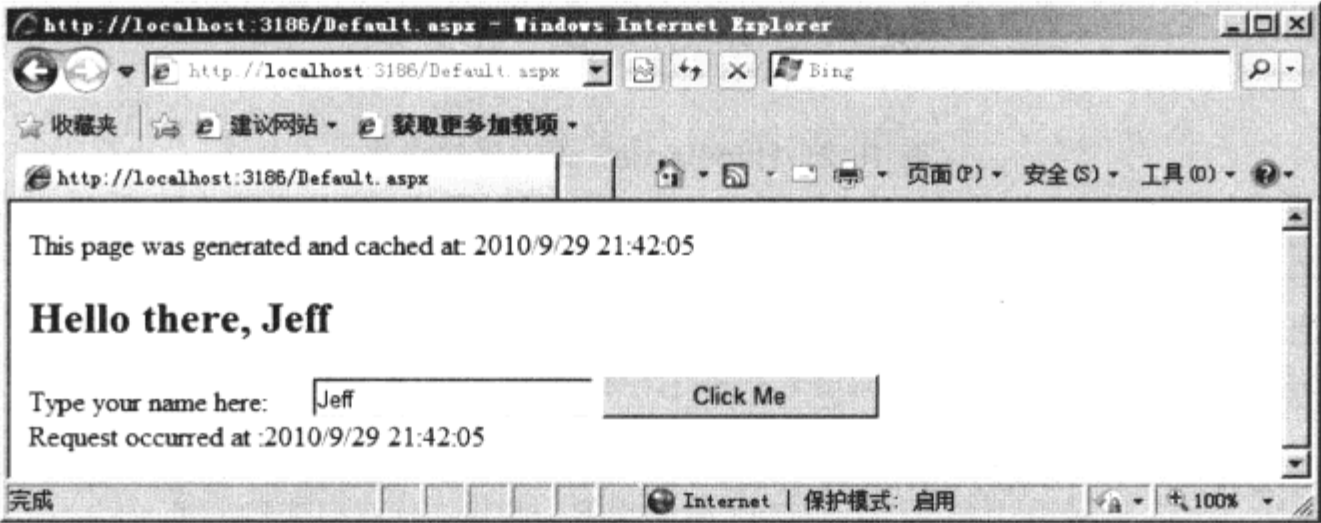
<asp:Substitution ID="Substitution1" MethodName="SubstituteDateAndTime"
    runat="server"/>

```

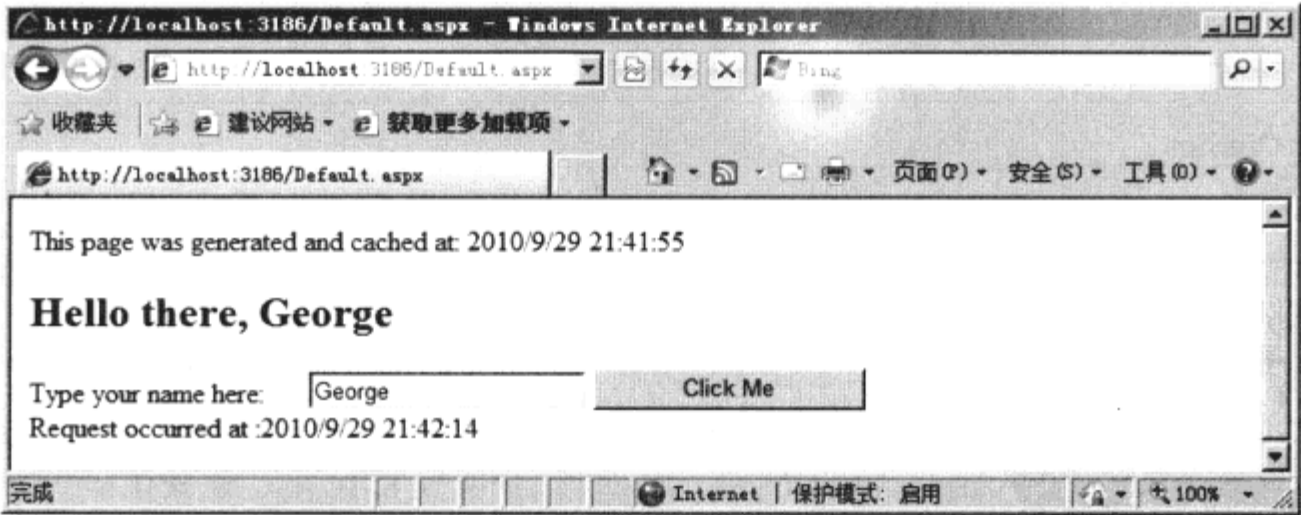
6. 运行这个页面, 在文本框中输入一个名字, 单击按钮将表单提交。注意页面的时间戳。在文本框中输入另一个名字, 单击按钮将表单提交。同样请注意页面的时间戳。然后, 重新输入第一次输入的名字, 并提交表单。如果在 60 秒的周期内完成所有操作, 则会看到页面的缓存版本(通过页面的时间戳来分辨)。下面 3 张图展示了根据 TextBoxName 参数的值进行缓存的页面。第一张图是在 TextBox 中第一次输入值并提交后的页面。Substitution 控件展示的是请求的时间, 它与 Page_Load 方法显示的时间相同。



第二张图是为 TextBoxName 参数设置另一个值的页面。Substitution 显示的请求时间和 Page_Load 显示的时间也是一致的：



第三张图是使用首次使用的参数请求后返回的页面。此时，Substitution 显示的时间和 Page_Load 方法显示的时间不同了。请求的时间要晚于 Page_Load 方法呈现的时间，这表明此页面已被缓存。



VaryByParam 属性还有几种设置方法。其中之一是将其设置为 none。在这种设置下，ASP.NET 只会为每种请求(如 GET、POST 和 HEAD)各缓存页面的一个版本。将 VaryByParam 设置为星号(*)会使 ASP.NET 针对每种查询字符串和 POST 主体各缓存一个版本的页面。上面的这个示例会根据文本框内容的不同来缓存不同版本的页面。

在 OutputCache 指令中添加 VaryByHeader 属性会使 ASP.NET 针对传入的每种标头字符串缓存一个版本的页面。(例如，客户端可能会发送 UserAgent 和 UserLanguage 这样的 HTTP

标头。)

使用 VaryByControl 属性，ASP.NET 便可以根据用户控件属性的不同而缓存不同的页面内容。稍后会练习缓存用户控件。

VaryByCustom 属性能够使 ASP.NET 根据几方面因素来管理缓存项。一种是浏览器的类型和版本。另一种是在 Global.asax 中提供自定义的 GetVaryByCustomString 方法，这样可以使 ASP.NET 根据自定义的字符串来缓存不同版本的页面。

16.2.2 HttpCachePolicy

使用 HttpCachePolicy 能够从另一个角度来管理输出缓存。该对象可以通过 Response 类获得。表 16.2 列出了 HttpCachePolicy 类的主要成员。

表 16.2 HttpCachePolicy 类的成员

成 员	说 明
AppendCacheExtension	用于向 Cache-Control HTTP 标头追加指定的文本
SetCacheability	用于设置 Cache-Control HTTP 标头，该标头能够控制文档在网络上缓存的方式
SetETag	用于将 ETag HTTP 标头设置为指定的字符串
SetExpires	用于将 Expires HTTP 标头设置为指定的绝对日期和时间
SetLastModified	用于设置最后修改的日期和时间(修改 Last-Modified HTTP 标头的内容)
SetMaxAge	用于将 Cache-Control: max-age HTTP 标头设置为指定的时间跨度
SetRevalidation	用于将 Cache-Control HTTP 标头设置为 must-revalidate 或 proxy-revalidate 指令
SetValidUntilExpires	用于指定 ASP.NET 缓存是否忽略客户端发送的使缓存失效的各种 HTTP Cache-Control 标头
SetVaryByCustom	用于指定区分缓存输出版本的自定义文本字符串
VaryByHeaders	用于设置区分缓存输出版本的所有 HTTP 标头的列表
VaryByParam	用于设置 GET (查询字符串)或 POST(HTTP 请求主体)中参数的列表，ASP.NET 将据此来区分缓存输出的版本

在设置 OutputCache 指令后，ASP.NET 会在 Page 类的 InitOutputCache 方法中填充 HttpCachePolicy 对象。Response 通过 Cache 属性将 HttpCachePolicy 暴露出来。这里的 Cache 这个名称可能会使人们将其与应用程序的数据缓存相混淆(最好称其为 CachePolicy)。在任何情况下都可以通过 HttpCachePolicy 类来控制服务器端输出缓存的行为和修改控制内容缓存的标头。OutputCache 指令也提供了 HttpCachePolicy 类的部分控制能力，但某些控制(如可调过期时间或修改页面的“上次修改”时间戳)则必须通过 HttpCachePolicy 类完成。

清单 16.1 给出了一个示例，禁用了源服务器对当前响应的缓存，并将“上次修改”时间戳设置为当前日期和时间。

清单 16.1 修改输出缓存策略

```
public partial class Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Thread.Sleep(0);
        Response.Write("This page was generated and cached at: " +
            DateTime.Now.ToString());

        Response.Cache.SetNoServerCaching();
        Response.Cache.SetLastModified(DateTime.Now);
    }
}
```

16.2.3 缓存的位置设置

除了区分页面缓存的版本外，还可以选择缓存的位置。可以通过 `OutputCache` 指令的 `Location` 属性或 `HttpCachePolicy` 类的 `SetCacheability` 方法来修改此设置。

`OutputCache` 指令支持以下几种输出缓存的位置设置：

- `Any` 允许浏览器、下游服务器和服务器缓存页面
- `Client` 只允许在客户端缓存页面
- `Downstream` 只允许下游服务器和客户端缓存页面
- `Server` 只允许服务器缓存页面
- `None` 禁用缓存

如果使用 `HttpCachePolicy`，则可以以编程方式设置缓存内容的位置。为此要用到 `HttpCachePolicy.SetCacheability` 方法(或 `HttpResponse.CacheControl` 属性)，它接受一个 `HttpCacheability` 枚举参数。这个枚举包含以下成员，它们要比 `OutputCache` 指令中的 `Location` 属性值更容易理解：

- `NoCache` 禁用缓存
- `Private` 只允许在客户端上缓存
- `Public` 允许在客户端和共享的代理上缓存
- `Server` 允许在服务器上缓存
- `ServerAndNoCache` 只允许在服务器上缓存输出，禁止在其他位置上进行缓存
- `ServerAndPrivate` 只允许在服务器和客户端上缓存输出，禁止在其他位置上进行缓存(不允许代理服务器缓存输出)

16.2.4 输出缓存依赖项

第15章介绍了ASP.NET对数据缓存的支持。ASP.NET可以通过多种依赖项来刷新应用程序数据缓存。ASP.NET输出缓存也具有类似的功能。Response对象具有许多针对不同缓存内容的依赖项。例如，假设某页面从一个文本文件加载数据，则可以设置针对这个文本文件的CacheDependency(缓存依赖项)。这样，在这个文本文件被更新后，缓存的输出便会失效而这个文件的内容则会被重新加载。

16.2.5 缓存配置文件

直接使用OutputCache指令会带来一个问题——所有设置都是硬编码的。修改缓存的行为需要更新页面的源代码。ASP.NET 2.0和后续版本支持缓存配置文件(cache profile)。该功能使我们可以将缓存的行为参数写在配置文件中。这样，配置缓存输出就变成了一种管理任务，而不是编程任务(事实也应当如此)。

为设置缓存配置文件，要在web.config文件中添加包含outputCacheProfiles列表的outputCacheSettings节。outputCacheProfiles包含的只是简单的键/值对，其中的键为输出缓存变量(如Duration)。在OutputCache指令中(通过键)引用配置文件后，ASP.NET会从配置文件读取相应的值，并将其应用到OutputCache指令。

下面的练习演示了如何通过缓存配置文件来避免硬编码缓存参数。

➤ 设置缓存配置文件

1. 在网站的web.config文件中添加缓存配置文件。如果网站中没有web.config文件，直接添加一个即可。然后，将缓存配置文件嵌在system.web节中。将缓存配置文件命名为profile。

```
<configuration>
  <system.web>
    <caching>
      <outputCacheSettings>
        <outputCacheProfiles>
          <add name="profile"
              duration="60"
              varyByParam="TextBoxName" />
        </outputCacheProfiles>
      </outputCacheSettings>
    </caching>
  </system.web>
</configuration>
```

2. 修改Default.aspx页面中的OutputCache指令，使其引用刚刚添加的配置文件：

```
<%@Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="-Default"
Trace="false"%>
```

```
<%@OutputCache CacheProfile="profile"%>
```

- 3. 运行这个页面。重复上一练习 6 的页面操作。如果在指定的周期内完成所有操作，则会发现其行为与采用硬编码设置的页面是一样的。

16.3 用户控件的缓存

正如可以缓存整个页面一样，ASP.NET 也能够缓存 UserControl。例如，假设要创建一个大型网站，通过使用许多控件(如菜单、超链接等)将用户导航到最近的新闻、摘要或其他位置。页面实际的内容可能会被更新，但链接一般不会。如果链接并不经常被更新，并且生成这部分内容要耗费较多资源，则可以用 UserControl 来包装它们，并为这个 UserControl 添加 OutputCache 指令。这样便可以使 ASP.NET 只缓存页面的局部(用户控件所属区域)。

可以在定义 UserControl 的 ASCX 文件中添加 OutputDirective 指令。UserControl 的 OutputDirective 指令支持一个名为 Shared 的属性(默认值为 false)，它能够使 ASP.NET 只缓存一个版本，并由所有引用此用户控件的页面共享。这样可以使缓存的用户控件服务于多个页面，进而提高应用程序的性能。

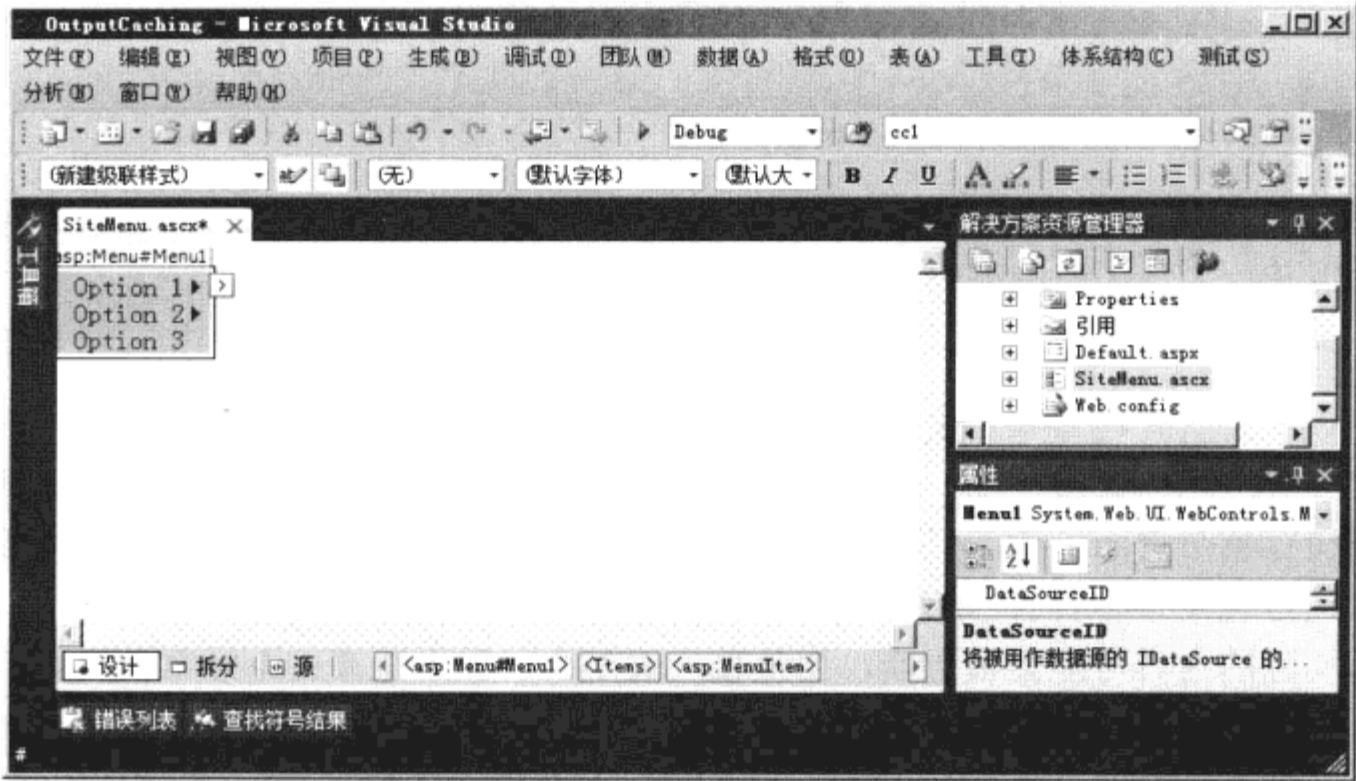
下面的练习演示了缓存 UserControl 输出的方法。

➤ 缓存用户控件的输出

- 1. 在 OutputCaching 项目中创建一个简单的 UserControl。导航控件非常适合被缓存，因此我们创建一个包含菜单的用户控件。将这个控件命名为 SiteMenu.ascx。拖放一个 Menu 控件到这个 UserControl 中，如下图所示。

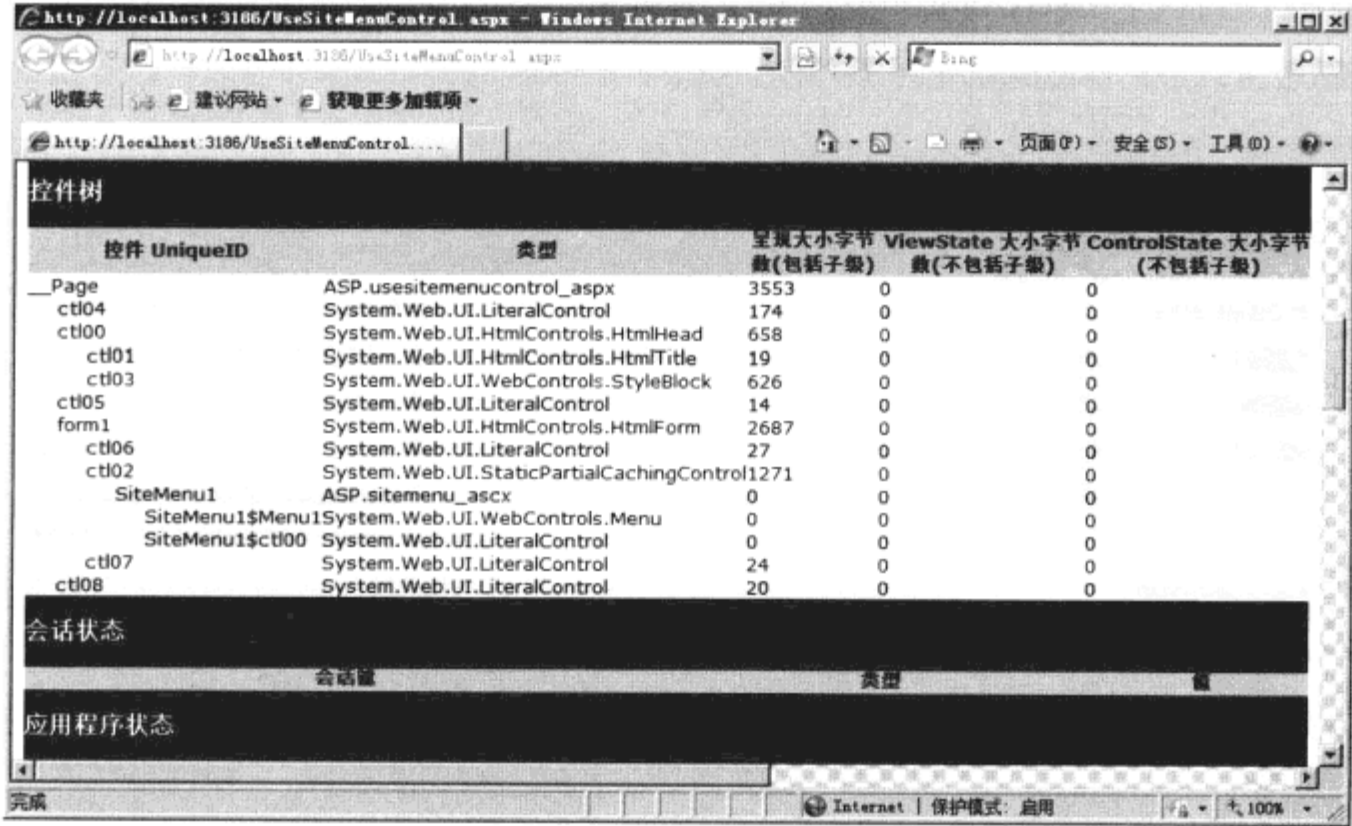


添加几个菜单项，如下图所示。

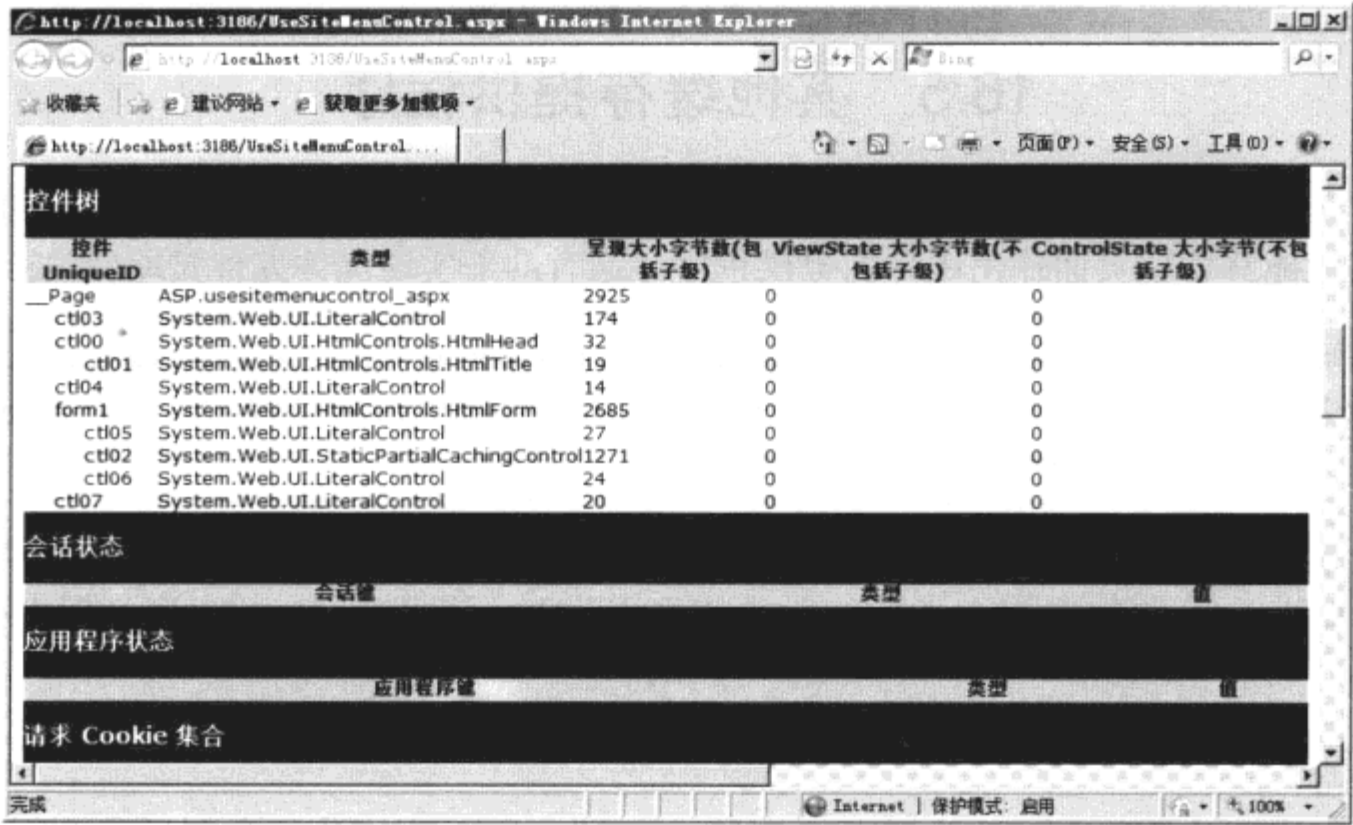


- 2. 在控件的源代码中，为 OutputCache 指令设置以下参数：

```
<%@ControlLanguage="C#" AutoEventWireup="true"
CodeFile="SiteMenu.ascx.cs" Inherits="OutputCaching.SiteMenu"%>
<%@OutputCache Duration="60" VaryByParam="none"%>
```
- 3. 在项目中添加一个新页面，将其命名为 UseSiteMenuControl.aspx。
- 4. 从“解决方案资源管理器”将新建的 SiteMenu 用户控件拖放到 UseSiteMenuControl 页面的设计视图上。
- 5. 启用 UseSiteMenuControl.aspx 页面的跟踪。(为 Page 指令添加 Trace="true" 属性。)运行这个页面。首次运行这个页面并查看 Trace 的控件树节，会看到下图所示的内容。



可以看到，此时整个控件树都会被呈现。在浏览时，刷新 UseSiteMenuControl.aspx 页面(在 Internet Explorer 中按 F5 键)，再查看 Trace 输出的“控件树”部分(见下图)。此时我们会发现，ASP.NET 使用缓存的控件来替代了整个 SiteMenu 控件。



16.4 适合应用输出缓存的场景

与其他缓存技术类似，最有效的输出缓存策略之一是缓存那些访问频繁且呈现复杂的页面。此外，还应确保只缓存那些不频繁更新的页面(否则会适得其反)。

页面中的控件生成大量 HTML 是耗费资源的。此时可以启用输出缓存。假设某页面通过 DataGrid 来显示员工电话簿。那么该页面会非常适合缓存，原因有 3 个。首先，这需要访问数据库(或访问内存中的缓存)。其次，DataGrid 的呈现非常耗费资源——尤其是在需要动态获取员工电话簿表的架构信息时。最后，雇员电话簿的更新并不非常频繁。缓存该页面可以在很大程度上降低系统开销。

在为缓存参数(如 VaryByParam)设置星号(*)时需要注意。如果查询字符串每次都发生变化，那么 VaryByParam=*会使 ASP.NET 为每种查询字符串都缓存一个版本的页面。这与不缓存的效果几乎是一样的(而且缓存输出还会占用内存)。不过，如果网站的用户数有限，并且请求的参数也有限，那么这样做也未尝不可。

此外，还应注意缓存对页面在不同浏览器中的显示效果的影响。在大多数情况下，不同浏览器会显示相同的内容。然而，如果缓存的内容涉及特定浏览器的功能(如动态 HTML)，那么不支持该控件的浏览器则会表现出不确定的行为。

调整输出缓存的行为也要谨慎。有效的缓存策略往往需要权衡多方面内容。虽然可以通过缓存输出来改进性能，但这也会增加内存的占用。可以使用性能监视工具来平衡性能与开销。

最后，用户控件是缓存输出的主要方式之一(特别是在其内容的更新不频繁的情况下)。将页面中不频繁更新的部分包装在用户控件中，并缓存整个用户控件。这样，由于只有该用户控件被缓存，因而资源的占用较低，最重要的是这样会改善应用程序的性能。

16.5 其他缓存提供程序

ASP.NET 4 引入了一种新的输出缓存功能，允许将输出存储在内存以外的位置。开发者可以指定其他能够管理页面输出缓存的提供程序。这有助于实现高伸缩性策略(如云计算)。

自定义的输出缓存提供程序应派生自 `OutputCacheProvider` 类，并至少重写 `Add`、`Get`、`Remove` 和 `Set` 方法。`Add` 方法用于添加缓存项，`Get` 方法用于获取指定的缓存项，`Remove` 用于删除缓存项，`Set` 方法用于替换现有的缓存项。

在完成自定义的提供程序后，还要在 `web.config` 文件中设置 `caching` 节。下面的代码指定了 `MyCacheLibrary` 程序集中的一个名为 `CacheWithFileBackingProvider` 的自定义提供程序。^①

```
<caching>
  <outputCache defaultProvider="CacheWithFileBacking">
    <providers>
      <add name="CacheWithFileBacking"
        type="MyCacheLibrary.CacheWithFileBackingProvider"/>
    </providers>
  </outputCache>
</caching>
```

在 `web.config` 文件中设置好缓存提供程序后，ASP.NET 便会在需要缓存特定页面时启用新的提供程序。

16.6 快速参考

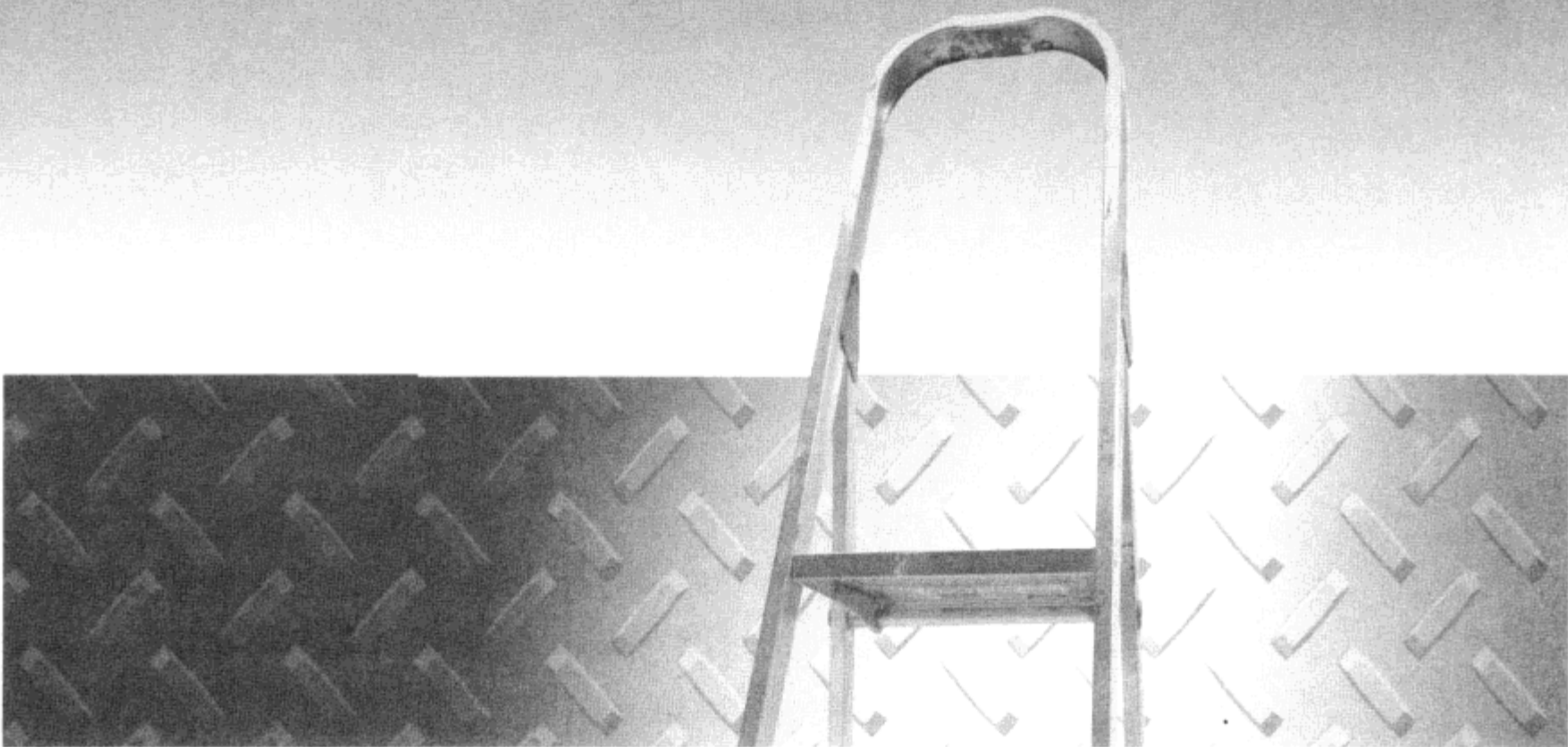
目 标	操 作
缓存页面的输出	为页面添加 <code>OutputCache</code> 指令
根据查询字符串参数的不同来缓存不同版本的页面	设置 <code>OutputCache</code> 指令的 <code>VaryByParam</code> 特性
根据标头的不同来缓存不同版本的页面	设置 <code>OutputCache</code> 指令的 <code>VaryByHeader</code> 特性
根据浏览器的不同来缓存不同版本的页面	将 <code>OutputCache</code> 指令的 <code>VaryByCustom</code> 特性设置为 <code>browser</code>
指定缓存内容的位置	可以通过 <code>OutputCache</code> 指令的 <code>Location</code> 特性指定
以编程方式访问缓存属性	使用 <code>Response</code> 对象的 <code>Cache</code> 属性，该对象是 <code>HttpCachePolicy</code> 类的实例

① 译者注：为使指定的提供程序生效，除了添加 `add` 标签外，还需要通过 `outputCache` 标签的 `defaultProvider` 属性来进行引用。

续表

目 标	操 作
通过 web.config 来管理输出缓存的行为	在 web.config 文件中添加 outputCacheProfile 元素，并在需要时引用
缓存用户控件	在用户控件的.ascx 文件中添加 OutputCache 指令

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569



第 IV 部分

诊断与插件

- ▶ 第 17 章 诊断与调试
- ▶ 第 18 章 HttpApplication 类与 HTTP 模块
- ▶ 第 19 章 HTTP 处理程序

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 17 章

诊断与调试

学习目标

- 启用页面跟踪
- 在页面跟踪中插入自定义的跟踪消息
- 启用应用程序跟踪
- 管理自定义的错误页面
- 管理应用程序中的异常

如今，虽然有许多软件架构技术和开发实践，但软件开发仍然是一项复杂的工作。软件框架(如 ASP.NET 和 Windows Forms)一直在向开发的规范化和可预测性方面努力(两者都是软件实践的目标)。然而，应用程序不可避免会存在问题，使其行为偏离我们的目标，这就需要排查。

本章将介绍 ASP.NET 对排除应用程序故障方面的支持。调试 Web 应用程序无疑要面临诸多挑战。还记得吗？HTTP 基本上是无连接的。客户端得到的输出只不过是应用程序的快照(snapshot)。本章将介绍如何监视应用程序的运行，以及如何跟踪任意请求的状态。本章还将介绍 ASP.NET 应用程序错误页面的管理方法和应用程序异常的捕获手段。

17.1 页面跟踪

调试 ASP.NET 应用程序一般从页面跟踪开始。Page 类有一个名为 Trace 的属性，用于暴露支持跟踪的对象。如果启用跟踪，ASP.NET 则会在发送给客户端的 HTML 尾部呈现整个请求和响应的上下文。

之前的章节用到了一些页面跟踪。在探索 ASP.NET 服务器端控件架构时，页面跟踪能够为我们理解页面的结构提供宝贵的信息。请不要忘记，页面是由许多服务器端控件以层次结构组合而成的。Page 能够嵌套多个子控件，而这些子控件可能会继续嵌套子控件(事实上，可能存在多层嵌套)。页面跟踪提供了一个专门用于显示页面中服务器端控件的区段。

17.1.1 跟踪

启用页面跟踪非常简单。只需要将页面中 Page 指令的 Trace 特性设置为 true 即可。为此，

可以直接编辑 ASPX 代码，也可以在 Visual Studio “设计器”的“属性”窗口中设置 Trace 特性。下面是 ASPX 代码中一条启用跟踪后的 Page 指令：

```
<%@Page Language="C#" AutoEventWireup="true" CodeFile="TraceMe.aspx.cs"
Inherits="TraceMe" Trace="true"%>
```

启用跟踪后再运行页面，HTML 流的尾部便会被追加跟踪信息。清单 17.1 是本章示例 DebugORama 项目中的代码。TraceMe.aspx 页面能够根据用户输入的字符串生成一个表格。这些字符串存储在会话状态中。在每次提交新的字符串后，表格都会被刷新。

清单 17.1 在页面加载期间构造表格的代码

```
public partial class TraceMe : System.Web.UI.Page
{
    ArrayList alTableEntries = null;

    protected void Page_Load(object sender, EventArgs e)
    {
        alTableEntries = (ArrayList)this.Session["TableEntries"];
        if (alTableEntries == null)
        {
            alTableEntries = new ArrayList();
        }
        AssembleTable();
    }

    protected void AssembleTable()
    {
        this.Table1.Rows.Clear();
        foreach (String s in alTableEntries)
        {
            TableRow row = new TableRow();
            TableCell cell = new TableCell();
            cell.Text = s;
            row.Cells.Add(cell);
            this.Table1.Rows.Add(row);
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        alTableEntries.Add(this.TextBox1.Text);
        this.Session["TableEntries"] = alTableEntries;
        AssembleTable();
    }
}
```

图 17.1 展示了该页面启用跟踪后的效果。

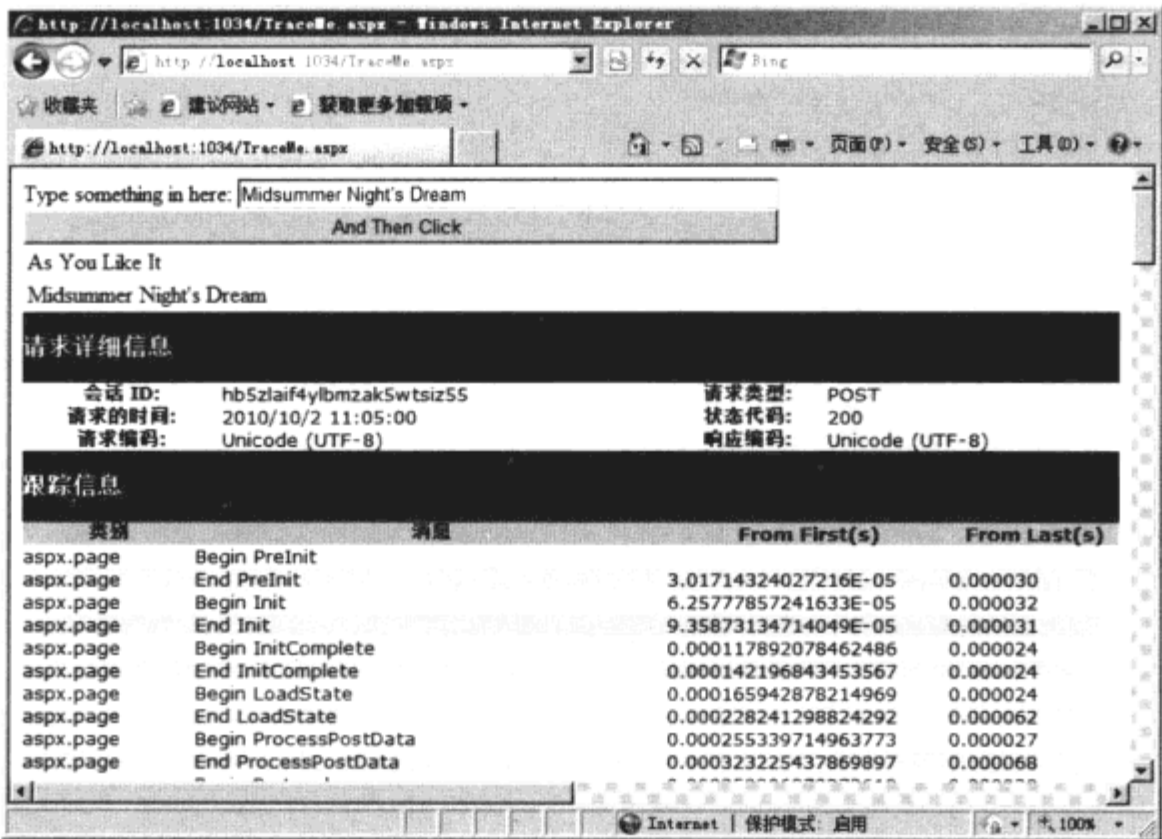


图 17.1 TraceMe.aspx 页面启用跟踪后的效果

向下滚动跟踪输出的内容，我们会看到控件树(正如在之前章节中看到的)。图 17.2 展示了 TraceMe.aspx 页面的控件树。



图 17.2 TraceMe.aspx 页面跟踪信息中的控件树

最后，继续向下滚动跟踪信息，我们会看到与请求关联的上下文信息(如图 17.3 和图 17.4 所示)。该应用程序在会话状态中保存了一个字符串数组。可以看到，“会话状态”节显示了这个会话状态字典的内容。读者还可以从中了解到其他一些上下文信息。例如，可以查看会话 ID 和导航到当前页面的 URL。

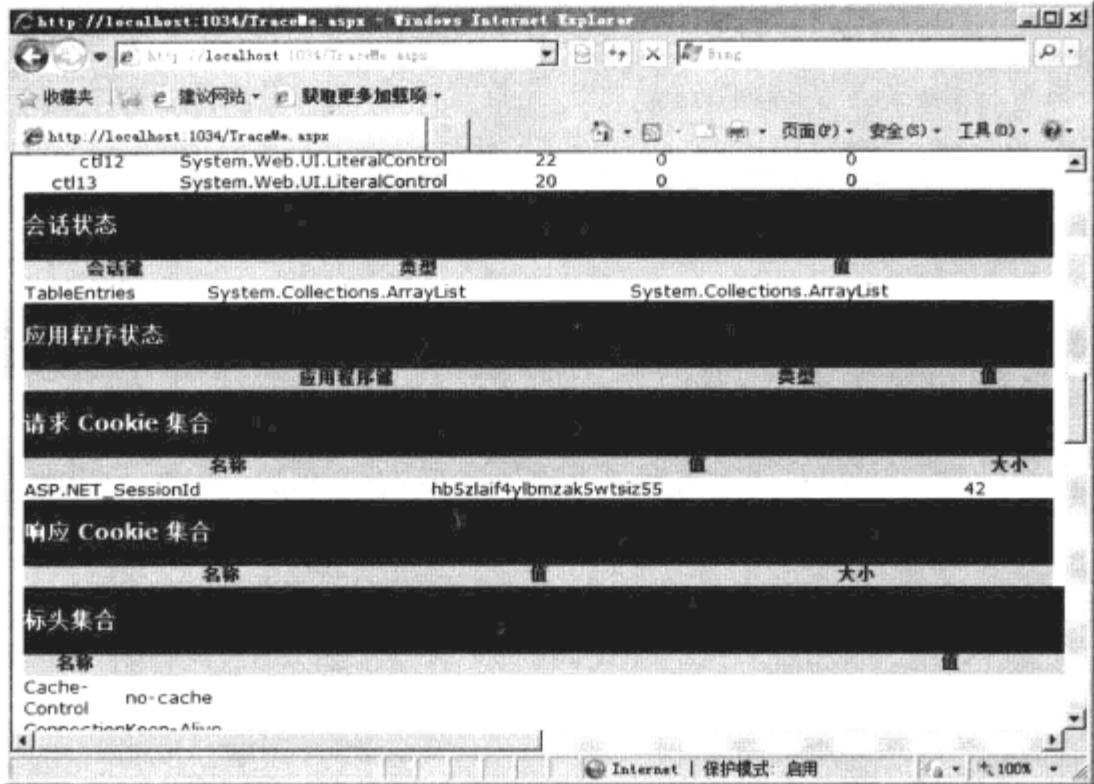


图 17.3 清单 17.1 中应用程序跟踪信息中的请求上下文信息



图 17.4 清单 17.1 中应用程序跟踪信息中的其他信息

当然，其中的许多信息在网站出现问题时会更有用。例如，如果由于某种原因，会话状态中存储字符串的列表被移除，那么页面中的表格则不能被生成。通过查看页面跟踪信息，便可以很快找到症结所在。如果用户反映网站的布局存在问题，则可以通过请求中的 User-Agent 信息来了解用户在使用哪种应用程序不兼容的浏览器。

17.1.2 跟踪语句

除了 ASP.NET 输出的请求上下文信息，页面跟踪还包含了执行过程中跟踪语句的输出。我们可以在“跟踪信息”节看到这些内容(如图 17.5 所示)。

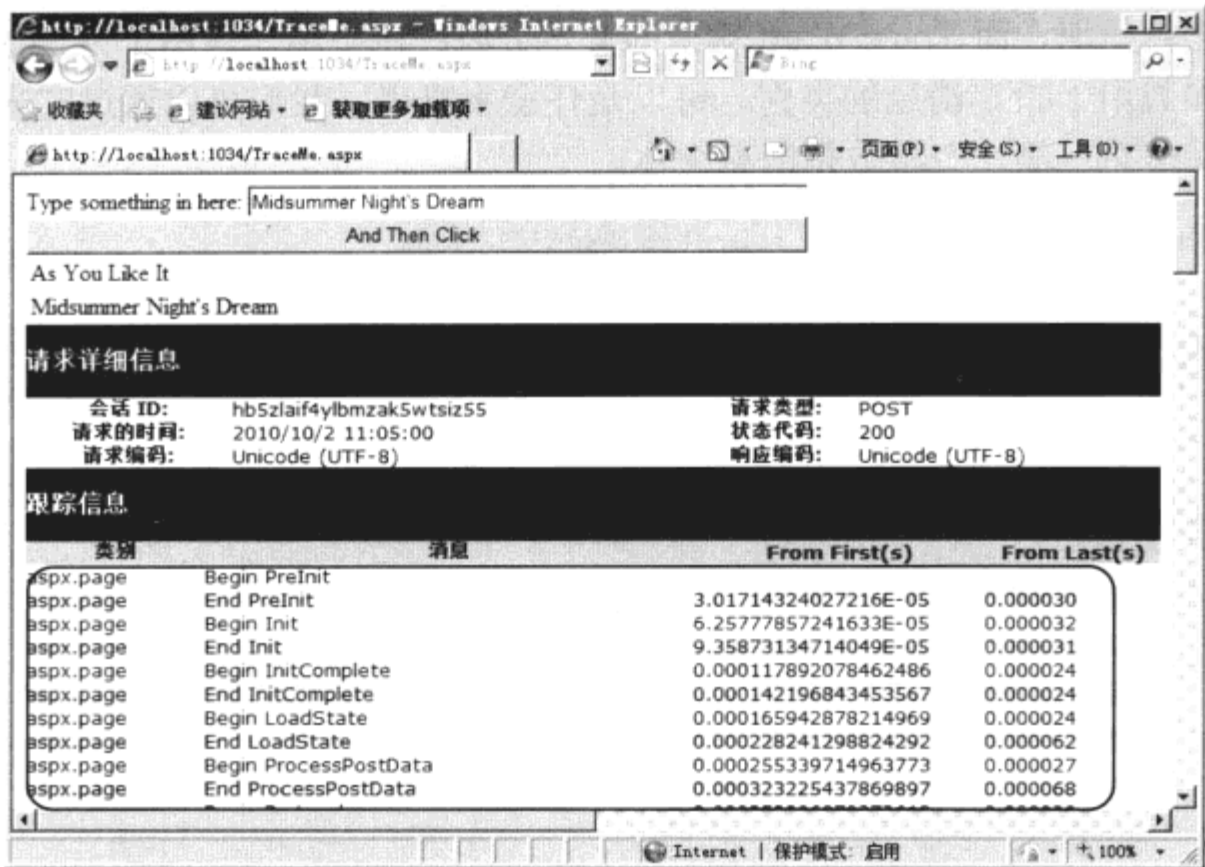


图 17.5 清单 17.1 中应用程序跟踪信息中“跟踪信息”节的消息能够跟踪页面的执行

图 17.5 展示了 ASP.NET 框架本身生成的跟踪消息。从中我们可以看到页面执行过程中的各个阶段(如 PreInit、Init、LoadState 等)。

除了获得 ASP.NET 生成的跟踪消息外，还可以插入自定义的跟踪信息。Page 类有一个 Trace 特性，可以用来添加任意跟踪消息。下面让我们来看看如何使用它。

注意 安装本书的示例代码要求用户具有计算机的 Administrator 权限。如果使用家用计算机，则可能已具有 Administrator 权限。如果使用某个组织的计算机，并且没有 Administrator 权限，则需要咨询计算机支持人员或 IT 人员。相关内容，请参考本书“前言”部分“示例代码”一节。

➤ 添加跟踪语句

1. 创建一个名为 DebugORama 的网站(可以是基于文件系统的网站)。添加一个名为 TraceMe.aspx 的页面。
2. 打开 TraceMe.aspx 页面。按前面几张截图所示，分别添加 Label(将其内容设置为“Type something in here:”)、TextBox、Button 和 Table 控件。双击按钮控件，为其添加一个 Click 事件的处理程序。添加清单 17.1 中的代码(在 Page 的 Load 事件执行期间生成表格)。为 Page 指令添加 Trace="true"特性。运行这个页面，确保其能够显示跟踪信息。
3. 使用页面的 Trace 对象在各方法的关键位置添加跟踪语句。例如，可以监视表格的生成过程。为此，在页面中调用 Trace.Write 或 Trace.Warn 都可以。Trace.Write 输出的文

① 临时译注：请根据原图画个矩形框。

本会以黑色呈现，而 `Trace.Warn` 输出的文本会以红色呈现。这两个方法的签名相同。第一个参数用于指定消息的类别，可以是任意字符串。我们可以通过它来区分不同类型的跟踪信息。第二个参数是消息主体。

```
public partial class TraceMe : System.Web.UI.Page
{
    ArrayList alTableEntries = null;

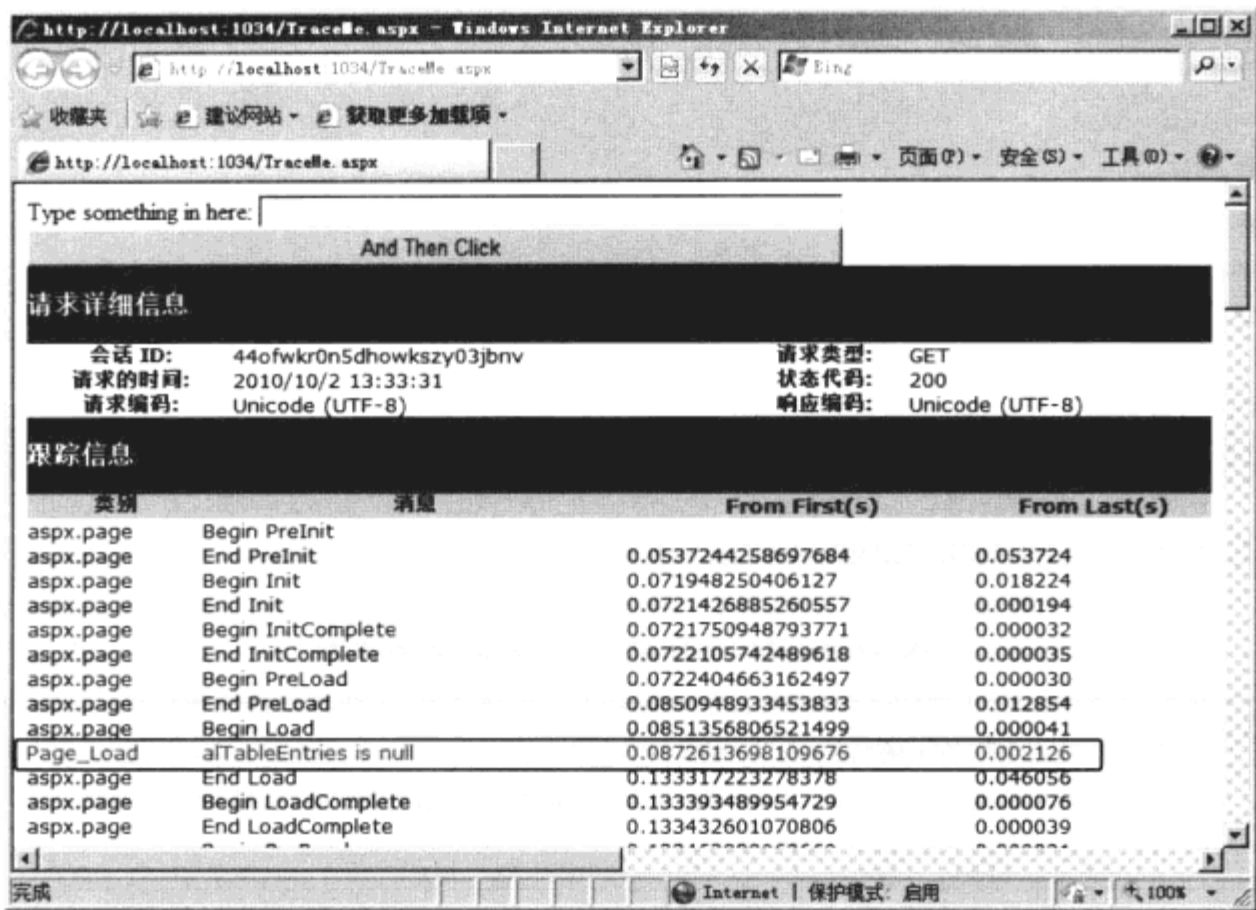
    Protected void Page_Load(object sender, EventArgs e)
    {
        alTableEntries = (ArrayList)this.Session["TableEntries"];
        if (alTableEntries == null)
        {
            Trace.Warn("Page_Load", "alTableEntries is null");
            alTableEntries = new ArrayList();
        }
        AssembleTable();
    }

    protected void AssembleTable()
    {
        this.Table1.Rows.Clear();

        foreach (String s in alTableEntries)
        {
            Trace.Write("AssembleTable", "String found: " + s);
            TableRow row = new TableRow();
            TableCell cell = new TableCell();
            cell.Text = s;
            row.Cells.Add(cell);
            this.Table1.Rows.Add(row);
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        Trace.Write("Button1_Click", "Adding string: " + this.TextBox1.Text);
        alTableEntries.Add(this.TextBox1.Text);
        this.Session["TableEntries"] = alTableEntries;
        AssembleTable();
    }
}
```

4. 编译并运行这个页面。只要跟踪处于启用状态，我们就会看到输出中的跟踪消息。在计算机屏幕上，这些消息会显示为红色(虽然在下图中显示为灰色)。下图中的跟踪消息说明 `alTableEntries` 变量为 `null`。



下图中的跟踪消息说明有一个字符串被添加到了表格中。



17.2 应用程序跟踪

虽然我们有时会使用单个页面的跟踪(尤其是要快速定位问题所在时),但这种技术有一个主要缺点——它会在页面尾部追加许多无用的信息。使用应用程序跟踪便可以避免这个问题。应用程序跟踪与页面跟踪所提供的信息是完全相同的,只不过前者存储在内存中,通过单独的页面和 ASP.NET 提供的特殊 HTTP 标头呈现。

为启用应用程序跟踪，需要这样设置 web.config:

```
<configuration>
  <system.web>
    <trace enabled="true"/>
  </system.web>
</configuration>
```

这只是启用了应用程序跟踪。此外，我们还可以对其进行设置。例如，可以只允许在宿主计算机上显示跟踪信息(而禁止客户端访问)。也可以控制内存中保留的响应数目。

表 17.1 列出了配置文件支持的跟踪设置。

表 17.1 web.config 中有关跟踪的设置

名 称	有 效 值	说 明
enabled	true false	用于启用或禁用应用程序级的跟踪
localOnly	true false	用于指定是只允许在本地主机显示跟踪信息，还是可以在任意位置显示
mostRecent	true false	用于指定是丢弃超出 requestLimit 限制的跟踪数据，还是只保留其限制的最新数据
pageOutput	true false	用于指定除缓存应用程序级的跟踪外是否在每个页面上显示跟踪输出
requestLimit	整数值	用于指定允许在内存中保留的跟踪信息的数目(默认值为 10)
traceMode	SortByTime SortByCategory	用于指定显示跟踪信息的顺序
writeToDiagnosticsTrace	true false	用于指定是否将跟踪信息发到 System.Diagnostics.Trace

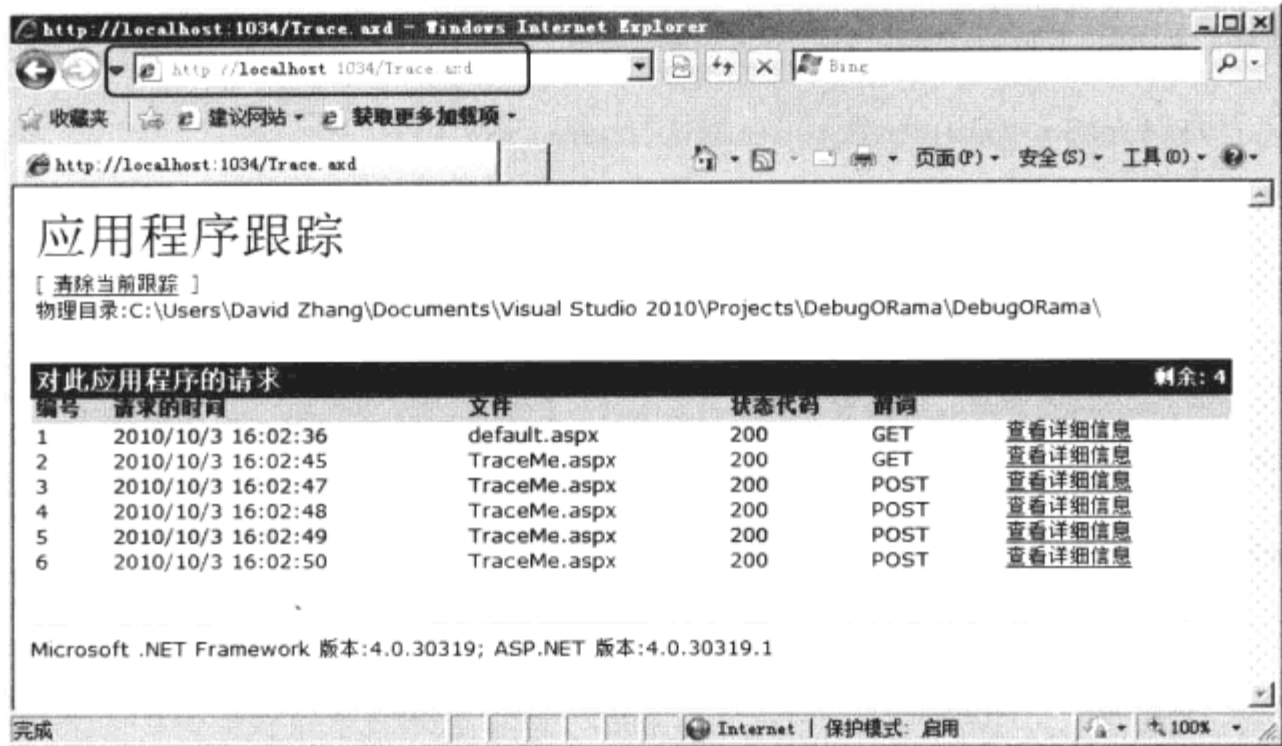
下面的练习演示了应用程序级跟踪的使用，以及如何查看相应的跟踪输出。

➤ 使用应用程序级的跟踪

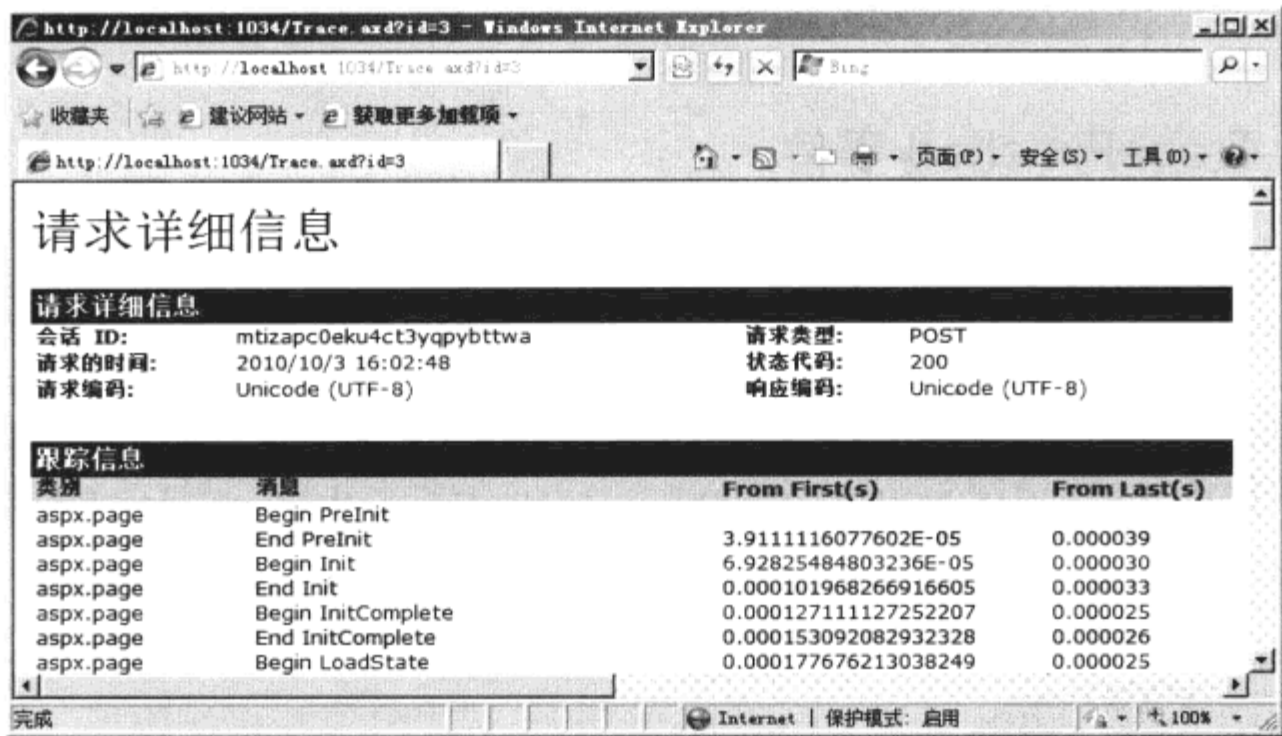
1. 打开 DebugORama 项目。在 TraceMe.aspx 中将 Page 指令的 Trace 特性删除^①，从而禁用页面级跟踪。
2. 在 web.config 中启用应用程序级跟踪。为此，打开 web.config 文件，添加 trace 元素(正如前文所述)。如果应用程序中没有配置文件，则在当前项目菜单中选择“添加新项”，添加一个即可。
3. 按 Ctrl+F5 组合键，运行 TraceMe.aspx 页面。为表格添加几个字符串。

① 译者注：将 Trace 特性移除可以避免其对应用程序缓存的影响。

4. 在地址栏中，将 TraceMe.aspx 修改为 Trace.axd(如下图所示)。这个 URL 会请求一个特殊的处理程序，它能够呈现存储在内存中的跟踪结果。



5. 此时，我们会看到一个请求列表。为查看每个请求跟踪信息的详细内容(如下图所示)，请单击相应的“查看详细信息”链接。



此时我们会发现，该页面与之前页面跟踪示例的输出完全一致，不过，它并没有与网页的内容混在一起。

17.2.1 以编程方式启用跟踪

虽然在许多情况下都可以通过“设计器”来启用跟踪，但有时可能要在运行时基于某种条件来(以编程方式)管理跟踪。例如，可以为一般的用户提供常规内容，而对特殊的用户显示跟踪信息；也可以在请求中出现特定参数时修改跟踪设置。

DebugORama 的 EnableTracing.aspx 演示了这种控制跟踪的方法。该页面能够以编程方式启用或禁用跟踪。如果用户输入的密码正确，那么跟踪则会被启用。

```
public partial class EnableTracing : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        if (this.TextBoxSecretCode.Text == "password")
        {
            this.Trace.IsEnabled = true;
        }
    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        this.Trace.IsEnabled = false;
    }
}
```

17.2.2 TraceFinished 事件

跟踪的上下文中有一个名为 TraceFinished 的事件——这是记录和以其他方式处理跟踪信息的最后机会。Trace 对象会在收集到所有请求信息后引发 TraceFinished 事件。

该事件可以在 Page_Load 事件中订阅。示例代码 DebugORama 中的 TraceFinished.aspx 页面演示了跟踪信息是如何通过 System.Diagnostics.Debug 被收集并写入调试控制台的。

```
public partial class TraceFinished : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Trace.TraceFinished +=
            new TraceContextEventHandler(TracingFinished);
    }

    void TracingFinished(object sender, TraceContextEventArgs e)
    {
        foreach (TraceContextRecord traceContextRecord in e.TraceRecords)
        {
            System.Diagnostics.Debug.WriteLine(traceContextRecord.Message);
        }
    }
}
```

17.2.3 融合其他跟踪消息

上一个示例通过 Trace 上下文中的 TraceFinished 事件向调试控制台添加跟踪消息。System.Diagnostics.Debug 是标准的 .NET 类型，用于管理跟踪和调试信息。在 2.0 版本后，ASP.NET 能够利用 WebPageTraceListener(网页跟踪监听程序)类型，这样，调用 System.Diagnostics.Trace 也能够添加 ASP.NET 跟踪消息。为设置 WebPageTraceListener，只需要在 web.config 中添加一行代码(详见表 17.1 中的 writeToDiagnosticsTrace 特性)。这在需要记录编译器的输出时比较有用。为此，将 writeToDiagnosticsTrace 选项设为 true，然后再启用编译器跟踪。编译器跟踪也可以在 web.config 中设置，但该设置位于 System.web 节之外。

```
<configuration>
  <system.codedom>
    <compilers>
      <compiler compilerOptions="/d:TRACE"/>
    </compilers>
  </system.codedom>
</configuration>
```

17.3 使用 Visual Studio 进行调试

ASP.NET 内建的跟踪功能非常易用，是调试(debug)应用程序的重要手段——尤其是在应用程序被部署之后。然而，在开发阶段，在页面中添加跟踪消息并运行，这样调试应用程序未免有些繁杂，因此有时并不是最有效的调试方式。Microsoft Visual Studio 在集成开发环境中提供了出色的调试功能，可以通过它来监视代码的执行和进行单步跟踪。事实上，开发 Web 应用程序能够利用所有 Visual Studio 的调试功能。

ASP.NET 和 Visual Studio 的结合使开发者可以像开发桌面应用程序一样来开发 Web 应用程序。这在调试方面也不例外。下面的练习将演示 Visual Studio 调试环境的使用。

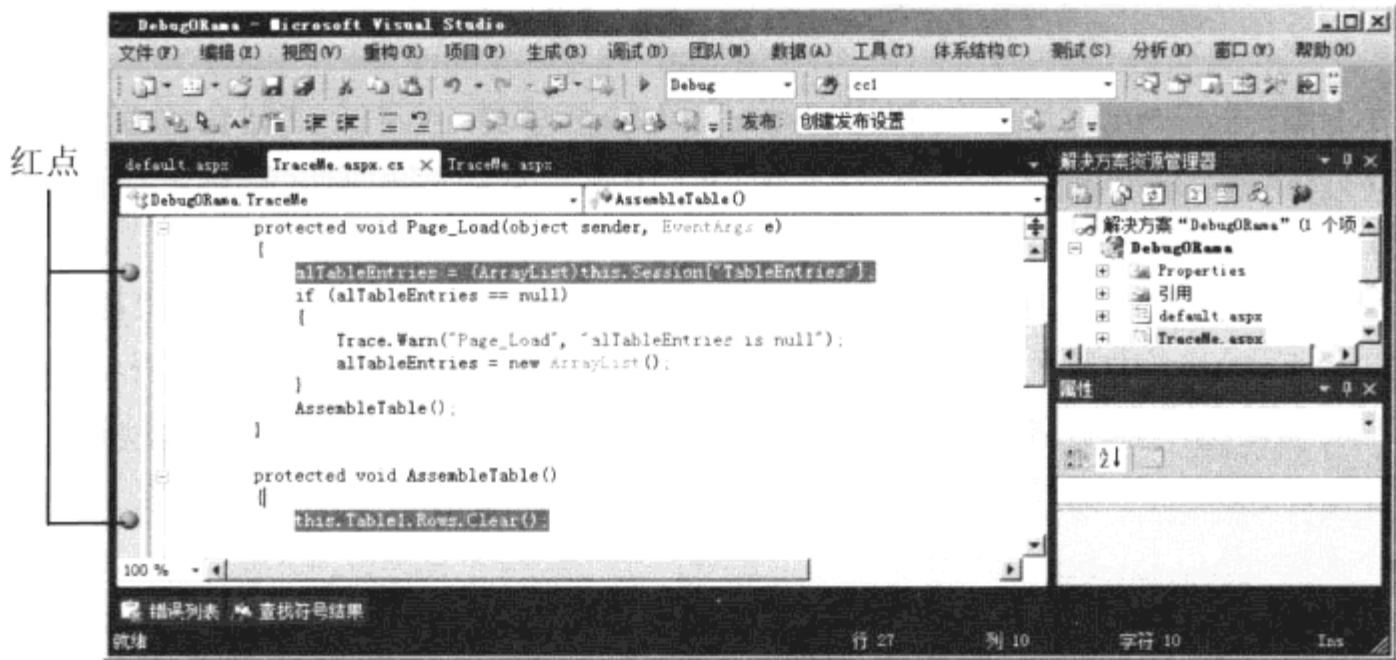
► 调试应用程序

1. 打开 DebugORama 项目。为启用调试，需要在 web.config 中添加相应的设置。这个设置可以手动添加，但 Visual Studio 能够在开始调试之前自动插入。

```
<configuration>
  <system.web>
    <compilation debug="true"/>
  </system.web>
</configuration>
```

2. 打开 TraceMe.aspx 页面，在 Page_Load、AssembleTable 和 Button1_Click 方法中设置断点。为插入断点，首先要在编辑窗口中单击要插入断点的那行代码，然后可以按 F9

键，或者在主菜单中依次单击“调试”|“切换断点”，也可以在代码编辑器左侧的灰色区域(显示断点标记的位置)单击。此时，Visual Studio 会在该行的左端显示一个红点：

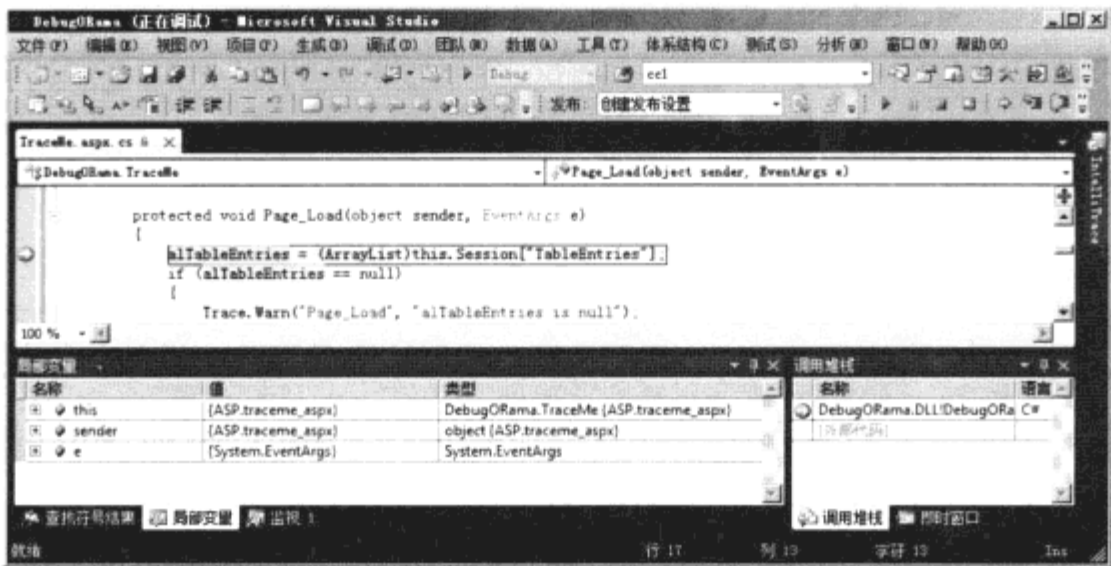


- 3. 按 F5 键，或者在主菜单中单击“调试”|“启动调试”，在调试模式下运行当前页面。如果 web.config 文件中没有添加调试设置，Visual Studio 会在添加调试属性之前进行询问，之后便运行网站。当执行到断点时，Visual Studio 会中止执行，并在窗口中用黄色突出显示即将要执行的语句(如下图所示)。



- 4. 在这个示例中，Page_Load 方法中的断点是 Visual Studio 遇到的第一个断点。此时，我们可以对代码进行单步跟踪。按 F10 键可以逐过程调试，按 F11 键可以逐语句调试。此外，也可以在主菜单中单击“调试”|“逐语句”或“逐过程”，或通过工具栏中的按钮来完成相应的操作。
- 5. 将鼠标指针悬停在要探查的变量上，Visual Studio 会在提示框中显示该变量的值。
- 6. 按 F5 键使程序继续执行。Visual Studio 会跳到另一个断点处。按同样的方式继续运行程序，直到历经所有断点。

- 7. 单击按钮来触发回发。此时，断点会再次被激活。首先是 Page_Load，然后是 Button_Click。这反映了网页的固有特性——客户端每次发出请求，ASP.NET 都会创建新的页面实例。
- 8. 最后，在主菜单中单击“调试”|“窗口”，尝试几个不同的调试窗口。不同的窗口能够从不同角度监视应用程序。下图展示了“局部变量”窗口^①。



- 9. 下图展示了“调用堆栈”窗口。该窗口能够反映方法的调用层次，可以在此跟踪和追溯整个程序的执行。



此外，我们可以通过“监视”窗口来查看变量的值，还可以通过“线程”窗口来查看当前所有线程的相关信息(其中包括线程 ID)。

17.4 错误页面

ASP.NET 一直以来所秉承的理念之一是尽可能地提供 Web 开发所需的功能。在这个意义上，“Internet 信息服务”(IIS)其实只是整个架构的一个中间管理程序。现在的 ASP.NET 引入了许多之前需要 IIS 支持的功能(尽管 IIS 在 7.0 版本中也引入了许多 ASP.NET 的特性)。

^① 译者注：“局部变量”窗口能够显示当前上下文中所有局部变量的信息。

管理自定义的错误页面是其中的功能之一。在 ASP.NET 中，我们可以用自定义的错误页面来避免向用户显示 ASP.NET 的错误消息。

我们可以通过修改 web.config 使 ASP.NET 在应用程序发生错误时返回指定的页面。表 17.2 列出了 web.config 中设置自定义错误页面的相关设置。

表 17.2 web.config 中有关错误页面的设置

属 性	说 明
defaultRedirect	发生异常时显示的错误页面
mode	on = 显示自定义的页面 off = 显示 ASP.NET 的错误页面
remoteOnly	向客户端显示自定义的错误页面，而只在本地显示 ASP.NET 的错误页面

下面的练习将演示如何设置自定义的错误页面。在这个示例中，我们将为应用程序添加几个错误页面，并探索引发这些错误页面的原因。

➤ 设置错误页面

1. 打开 DebugORama 项目。
2. 在这个项目中添加一个“Web 窗体”，将其命名为 ThrowErrors.aspx。
3. 在这个页面中添加两个按钮，将其中一个命名为 ButtonThrow404，另一个命名为 ButtonThrowOther。ButtonThrow404 用于引发 404 错误(常见的解释是“无法找到资源”)，ButtonThrowOther 用于抛出其他异常。
4. 为应用程序添加两个用作自定义错误页面的“HTML 页”，分别命名为 404Error.htm 和 SomethingBadHappened.htm。(本示例采用的是简单的 HTML 文件，而并没有使用 ASPX 文件。)下图展示了添加 404Error.htm 时的“添加新项”窗口。



下图展示了添加 SomethingBadHappened.htm 时的“添加新项”窗口。



- 5. 在每个错误页面中添加一些内容。示例代码中的 404 错误页面会显示三行错误消息，而用于其他错误的页面将只提示有错误发生。
- 6. 为使 ASP.NET 启用这两个自定义的错误页面，需要在 web.config 中添加 customErrors 节(如下所示)。

```
<configuration>
  <system.web>
    <customErrors
      defaultRedirect="SomethingBadHappened.htm" mode="On">
      <error statusCode="404"
        redirect="404Error.htm"/>
    </customErrors>
  </system.web>
</configuration>
```

这样一来，如果系统无法找到某个文件，ASP.NET 就会返回 404Error.htm。如果发生其他错误，ASP.NET 则会返回 SomethingBadHappened.htm。

- 7. 添加几个生成错误的处理程序。在第一个按钮的处理程序中，将客户端导航到一个不存在的页面。在这个示例中，我们使用一个名为 NonExistent.aspx 的页面。由于这个页面不存在，因而会引发 404 错误。第二个按钮的处理程序会抛出任意一个异常。

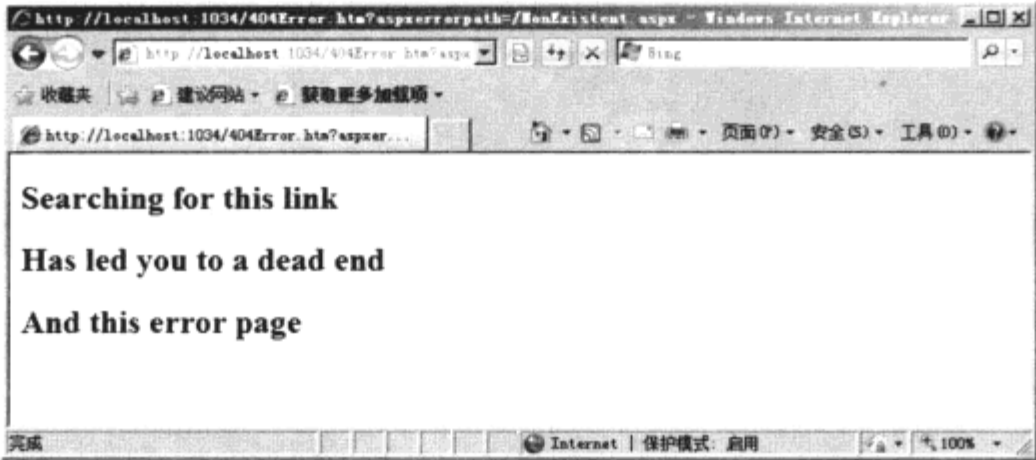
```
public partial class ThrowErrors : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void ButtonThrow404_Click(object sender, EventArgs e)
```

```
{
    this.Response.Redirect("NonExistent.aspx");
}

protected void ButtonThrowOther_Click(object sender, EventArgs e)
{
    throw new Exception();
}
}
```

在尝试重定向到一个不存在的文件时，服务器会返回“无法找到资源”错误页面(见下图)。



抛出泛化的异常会使服务器返回下图所示的错误页面。



如果在调试模式下运行这个项目，调试器则会在异常发生时中止程序的执行。为在 Visual Studio 报告错误后继续执行并显示错误页面，可以按 F5 键。

在这个示例中，错误页面对最终用户并没有什么实际的作用，因为其中没有包含有关异常的详细信息。实际的错误页面应包含一些有价值的信息，甚至提供寻求帮助的联系人。在结束调试和诊断的话题之前，让我们看看如何优雅地捕获未处理的异常。

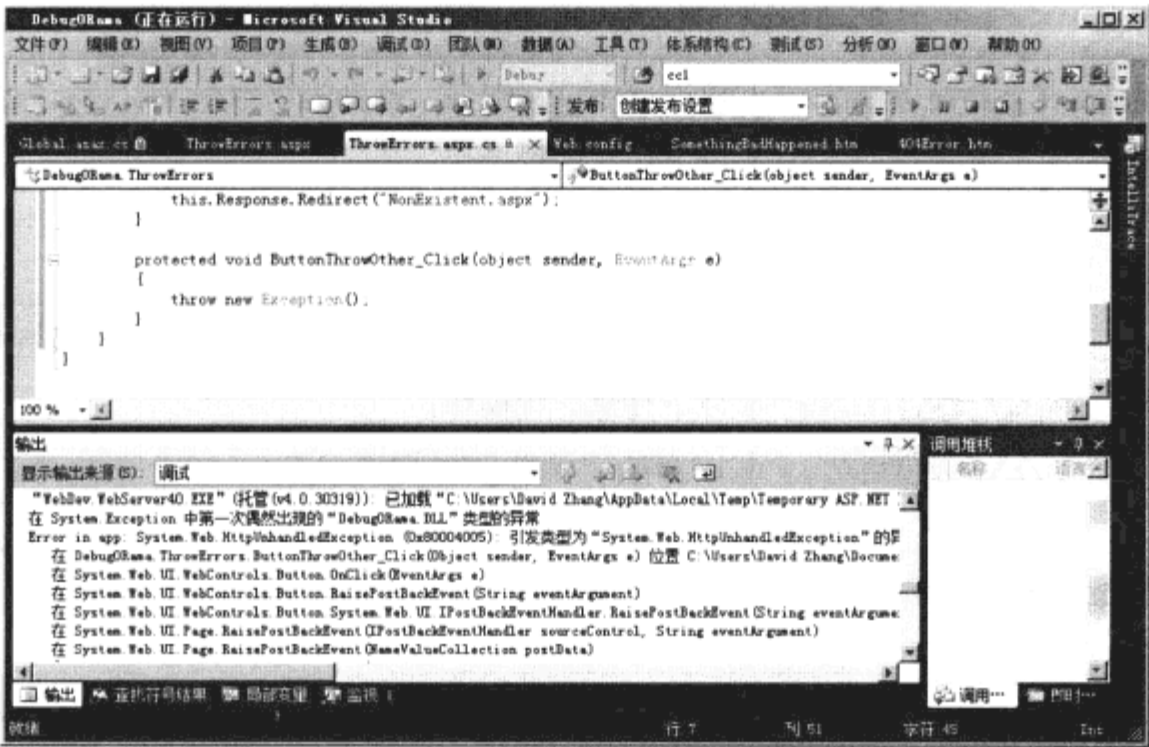
17.5 未处理的异常

对于上一个练习，在抛出异常后，ASP.NET 会将用户重定向到默认的错误页面。在 ASP.NET 中，我们可以通过订阅 HttpApplication 的 Error 事件来捕获异常，从而进行适当的处理。

为实现这种异常处理，最简单的方法是在 Global.asax.cs 中通过 `HttpApplication` 的派生类来定义处理程序。在订阅 `Error` 事件后，应用程序会在异常发生时收到通知，开发者就能进行相应的处理。例如，可以在将用户重定向到错误页面之前将错误记录到日志当中，或者将错误发送到调试控制台。下面的示例将异常重定向到一个错误页面：

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Error(object sender, EventArgs e)
    {
        Exception ex = Server.GetLastError();
        // display the exception
        System.Diagnostics.Debug.WriteLine("Error in app: " + ex);
        // display the exception
        System.Diagnostics.Debug.WriteLine("Error in app: " + ex);
        if (ex is HttpUnhandledException)
        {
            Context.ClearError();
            Server.Transfer("somethingbadhappened.htm");
        }
        else
        {
        }
    }
}
```

上述代码能够在重定向发生之前捕获异常，这便给了我们记录异常的机会(或者像这段代码一样将异常显示在 `System.Diagnostics.Debug` 上下文中)。下图展示了 Visual Studio “输出”窗口中给出的异常的详细信息。



我们也可以在 ASP.NET 重定向到 `web.config` 文件指定的页面之前截断异常处理，并将用户重定向到其他页面。为此，应首先调用 `Context.ClearError` 来清除错误，以免 ASP.NET 生成标准的错误页面。

17.6 快速参考

目 标	操 作
为调试网站做好准备	在 web.config 中添加以下设置： <code><system.web></code> <code><compilation debug="true"/></code> <code></system.web></code>
启用应用程序级的跟踪	在 web.config 中添加以下设置： <code><system.web></code> <code><trace enabled="true"/></code> <code></system.web></code>
启用页面级的跟踪	在 Visual Studio 的“属性”窗口中将 Page 类的 Trace 属性设置为 true，或者在 Page 指令中添加 Trace="true"
在 Visual Studio 中调试 Web 应用程序	确保 web.config 中添加了 debug 属性。选择以下任意一种方式，以调试模式运行程序： 1. 在主菜单上单击“调试” “启动调试” 2. 按 F5 键
在 Visual Studio 中为程序设置断点	确定要中止执行的那行代码，然后选择以下任意一种方式来设置或取消断点： 1. 单击此行代码，在主菜单中选择“调试” “切换断点” 2. 单击此行代码，按 F9 键 3. 单击文本编辑器该行左侧的灰色区域
在 Visual Studio 调试器中每次执行一个过程	在调试器正在运行，并且程序中止于所要执行的代码时，进行以下任意一种操作： 1. 单击“调试” “逐过程” 2. 按 F10 键
在 Visual Studio 调试器中进入某条语句的内部	在调试器正在运行，并且程序中止于所要执行的代码时，进行以下任意一种操作： 1. 单击“调试” “逐语句” 2. 按 F11 键
在发生某种特定 HTTP 错误时使 ASP.NET 显示指定的页面	在 web.config 文件中添加<customErrors>节来指定该页面
捕获 ASP.NET 中特定的.NET 异常和未处理的异常	在 Global.asax 中的 Application_Error 中处理异常(包括未处理的异常)。然后，一般可将用户重定向到指定的页面(注意，具体的错误信息位于 HttpUnhandledException 的 InnerException 属性中。)

第 18 章

HttpApplication 类 与 HTTP 模块

学习目标

- 将 HttpApplication 对象作为应用程序的全局访问点
- 管理 HttpApplication 对象中的数据
- 管理 HttpApplication 对象中的事件
- 使用 HTTP 模块

本章将介绍 ASP.NET 中应用程序范围的状态和事件。桌面应用程序的全局访问点比较常见。例如，“Microsoft 基础类”(MFC)(一个支持 Windows 底层开发的 C++ 类库)中有一个名为 CWinApp 的类，能够保存可以在应用程序范围内使用的状态。这种状态包括当前应用程序的实例句柄、主窗口的句柄、应用程序启动时传入的参数。CWinApp 类还能够运行消息循环——这种机制只能在 Windows 应用程序的全局访问点运行。Windows 应用程序有且只有一个 CWinApp 类，在应用程序的任意位置都可以使用。

Windows Forms 和 Windows Presentation Foundation(二者都是用于开发 Windows 桌面应用程序的 .NET 类库)中都有名为 Application 的类，所包含的状态类型也是相同的：命令行参数、顶层窗口，以及程序运行过程中所需的其他状态。Application 类也具有消息循环。事实上，Microsoft Silverlight 引入的 WPF 模型也包含 Application 对象。

Web 开发也需要与桌面应用程序中 Application 对象类似的“全局访问点”。为 Web 应用程序添加全面访问点便可以实现“数据缓存”和“会话状态”这样的功能。本章将重点介绍 ASP.NET 应用程序全局访问点的开发。

18.1 Application 对象——全局访问点

在学习过这么多内容后，不难发现 Web 开发的一个特点——开发者要时刻注意应用程序的状态。在早期，Web 应用程序本身并不包含对状态的支持。Web 请求总是通过无连接的协议发送出去，请求的大部分状态会在请求抵达端点(endpoint)后消失。

第 4 章介绍了 ASP.NET “Web 窗体”应用程序中视图状态的表示方式。ASP.NET 的服务器端控件都支持视图状态。视图状态嵌于浏览器与服务器之间传输的数据当中，(多数)用于保存用户界面(UI)信息，使得从外界看就像浏览器和服务器之间有持续性连接一样。举例来说，如果没有视图状态(或在服务器程序中实现特殊功能)，那么像下拉列表这样的 UI 元素在回发后其状态便会丢失，从而总是会显示成选中了第一个选项(即便它其实并不是所

选择的项)。

第 14 章介绍了会话状态——与会话关联的数据。会话状态可以用来存储与特定客户端关联的数据(如购物车中的商品)。

最后，第 15 章讨论了如何通过缓存状态来避免访问数据源。从内存加载数据要比从数据库加载或重新生成数据快得多。为使应用程序的各部分都能够访问这些数据，它们不能存储在视图状态和会话状态中。在能够使用 `HttpContext` 的任意位置，我们都可以访问缓存。`HttpContext` 包含 `HttpApplication` 实例的引用。除了可以缓存数据外，应用程序对象也有一个用来存储数据的字典，其工作方式与 `Cache` 的基本一致。然而，`Cache` 和 `HttpApplication` 字典之间存在一些重要的差异，本章最后会加以说明。

为应用程序的其他部分提供数据缓存和字典不是引入中央应用程序对象的唯一原因。另一个原因是要提供处理应用程序范围事件的机制。我们知道，`Page` 类专门用于处理与请求有关的事件。然而，不妨考虑一下整个 ASP.NET 管线的工作方式。其中的一些事件与处理页面或请求无关，因而不能在常规的页面处理机制中订阅这些事件。

例如，第 14 章介绍了会话状态，如果某个网站启用了会话状态，那么当用户的首个请求到达后，应何时创建会话对象呢。显然，应在页面处理开始之前创建它。第 9 章讨论了 ASP.NET 的安全模型，但身份验证和授权应何时处理呢。我们当然也希望在常规请求处理的上下文之外进行。第 16 章介绍的输出缓存也是一个例子。为使输出缓存正常工作，ASP.NET 需要监听管线，等待用户的首个请求，以便能够绕过页面的整个创建过程而呈现缓存的内容。

ASP.NET 中的 `HttpApplication` 对象能够管理上述各种操作。在应用程序运行时，`HttpApplication` 对象充当整个 Web 应用程序的全局访问点。`HttpApplication` 采用了设计模式中的“单件”(singleton)模式，因而可以认为应用程序中该对象只有一个实例。可以在任意位置通过 `HttpContext` 类的 `Current` 属性来访问它。

18.2 `HttpApplication` 的重写

为添加自定义的状态和事件处理程序而重写 `HttpApplication`，只需要为网站添加一个全局应用程序对象。如果通过 Visual Studio 来创建网站(即单击“文件”|“新建”|“网站”|“ASP.NET 网站”)，Visual Studio 会在项目中添加 `Global.asax` 文件。我们可以在 `Global.asax` 中为应用程序对象添加任意服务器端脚本块。

如果通过 Visual Studio 创建“ASP.NET Web 应用程序”项目(即单击“文件”|“新建”|“项目”)，Visual Studio 会添加两个文件——`Global.asax` 和 `Global.asax.cs`。`Global.asax` 和 `Global.asax.cs` 的关系，与 ASPX 文件和对应的 CS 文件的关系相同。事实上，如果项目中尚不存在全局应用程序对象，也可以通过 Visual Studio 来添加。在为应用程序添加 `Global.asax` 和 `Global.asax.cs` 文件后，便可以处理应用程序范围的事件。与 `.aspx` 文件的顶部包含 `Page` 指令类似，`Global.asax` 文件也包含一个指令——`Application`。`Application` 指令

能够使运行库将指定的对象作为应用程序对象。与页面不同的是，应用程序中只能有一个 Global.asax 文件。

清单 18.1 展示了一个 Global.asax.cs 文件，它派生自在单击“文件”|“新建”|“项目”|“ASP.NET Web 应用程序”后，Visual Studio 自动生成的 HttpApplication 类。Visual Studio 提供的 Global.asax.cs 能够处理 Application_Start、Application_End、Application_Error、Begin_Request、Authenticate_Request、Session_Start 和 Session_End 事件。

清单 18.1 Global.asax.cs 文件中应用程序事件处理程序的存根

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(object sender, EventArgs e) { }
    protected void Session_Start(object sender, EventArgs e) { }
    protected void Application_BeginRequest(object sender, EventArgs e) { }
    protected void Application_AuthenticateRequest(object sender, EventArgs e) { }
    protected void Application_Error(object sender, EventArgs e) { }
    protected void Session_End(object sender, EventArgs e) { }
    protected void Application_End(object sender, EventArgs e) { }
}
```

为进一步理解这些事件的工作方式，请完成以下练习。我们将一些数据存储在应用程序的字典中，并稍后在页面加载时获取。

➤ 管理应用程序的状态

- 1. 创建一个 Web 应用程序项目(在 Visual Studio 中单击“文件”|“新建”|“项目”|“ASP.NET 空 Web 应用程序”)，将其命名为 UseApplication。
- 2. 添加一个名为 Default.aspx 的页面。在该页面中添加一个 GridView 控件。暂时不要为其设置数据源，稍后会使用存储在应用程序中的数据来填充它。
- 3. 在“解决方案资源管理器”中添加 Global.asax 和 Global.asax.cs 文件。可以右键单击项目节点，也可以在主菜单中选择“网站”|“添加新项”。选择“全局应用程序类”(如下图所示)。



4. 在为应用程序添加 Global.asax 和 Global.asax.cs 这两个文件后，Application_Start 事件的处理程序存根会被自动添加(虽然其不包含任何代码)。
5. 为在应用程序对象中存储数据，从第 15 章的 UseDataCaching 项目导入 QuotesCollection 数据和代码文件。如果尚未生成 XML 和 XSD 文件，可以运行一下 UseDataCaching 项目。单击 CacheDependencies.aspx 页面的“Generate XML File”按钮便会生成 XML 文件和架构文件。选择“项目”|“添加现有项”。以相同的方式导入 UseDataCaching\App_Data 目录下的 QuotesCollection.xml 和 QuotesCollection.xsd 文件。
6. 为 Application_Start 事件添加代码，加载数据并将数据存储在应用程序字典中。为加载 XML 和 XSD 文件，可以通过 Server.MapPath 类取得当前应用程序的路径。向应用程序字典添加数据的方法与向缓存添加数据的方法相同：

```
protected void Application_Start(object sender, EventArgs e)
{
    QuotesCollection quotesCollection = new QuotesCollection();

    String strAppPath = Server.MapPath("");

    String strFilePathXml =
        strAppPath + "\\app_data\\QuotesCollection.xml";
    String strFilePathSchema = strAppPath +
        "\\app_data\\QuotesCollection.xsd";
    quotesCollection.ReadXmlSchema(strFilePathSchema);
    quotesCollection.ReadXml(strFilePathXml);

    Application["quotesCollection"] = quotesCollection;
}
```

7. 修改 Default.aspx 的 Page_Load 方法，使其从应用程序字典加载数据。应用程序状态可以通过页面提供的 Application 对象来访问。访问字典中的数据只需使用相应的索引即可。在从字典中获得数据后，通过 DataSource 属性将其绑定到 GridView 控件：

```
protected void Page_Load(object sender, EventArgs e)
{
    QuotesCollection quotesCollection =
        (QuotesCollection)Application["quotesCollection"];

    GridView1.DataSource = quotesCollection;
    GridView1.DataBind();
}
```

18.3 使用应用程序状态的注意事项

正如您所看到的，应用程序状态和应用程序数据缓存在功能上似乎有些重复。确实，两者具有相同的访问范围(可以在应用程序的任意位置访问)，并且设置和获取数据都使用索引。然而，应用程序状态和缓存有几点重要差异。

首先，应用程序状态中的数据在被显式地移除之前会一直被保留。应用程序数据缓存则更为灵活，可以设置过期和其他移除/刷新条件。

此外，在应用程序状态字典中添加过多数据会影响应用程序的伸缩性。为以线程安全的方式使用全局的应用程序状态，需要使用 `HttpApplicationState` 类提供的 `Lock` 方法。虽然 `Lock` 方法能够确保数据不被损坏(corrupt)，但频繁地锁定应用程序会极大地降低其处理请求的效率。

在理想情况下，加载到应用程序状态中的数据应该是只读的，而不要频繁更新。如果明白了这些问题，那么便可以通过应用程序状态来存储应用程序全局范围使用的数据。

18.4 事件的处理

应用程序对象还能够处理应用程序范围的事件。正如之前的练习所展示的，我们可以在 `Global.asax.cs` 文件中插入全局事件的处理程序。在添加此文件时，Visual Studio 会自动添加几个处理程序。Visual Studio 在 `Global.asax.cs` 中生成的事件处理程序存根包括：`Application_Start`、`Application_End`、`Application_Error`、`Application_BeginRequest`、`Application_AuthenticateRequest`、`Session_Start` 和 `Session_End`。下面将逐一进行介绍。

18.4.1 Application_Start

`Application_Start` 在应用程序初始化时运行——即第一个请求到达时。由于 `Application_Start` 在应用程序生命周期开始时运行(且只运行一次)，因而该事件一般用于在应用程序启动时加载和初始化数据(正如之前的示例所展示的)。

18.4.2 Application_End

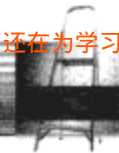
ASP.NET 运行库会在关闭应用程序时运行 `Application_End`。可以通过该事件来清理需要特别进行释放的资源。

18.4.3 Application_Error

Web 应用程序有时会发生错误。如果现有的应用程序发生了错误，我们可能会看到标准的 ASP.NET 错误页面。在部署应用程序后，我们也许不希望客户端看到这种页面。此时，可以通过订阅 `Application_Error` 事件来处理错误。异常一般就近处理，而其他未处理的异常可以在这里处理。

18.4.4 Application_BeginRequest

`Application_BeginRequest` 事件会在每次用户请求应用程序时被引发，它适合在处理请求的



最初阶段执行某些操作。

18.4.5 Application_AuthenticateRequest

发起请求的用户的身经 ASP.NET 确认后，Application_AuthenticateRequest 事件便会被引发。该事件引发后，用户的身份就明确了。

18.4.6 Session_Start

Session_Start 事件会在每个用户最初请求应用程序(建立新的会话)时被引发。该事件适合初始化会话变量(在页面加载之前)。

18.4.7 Session_End

该事件会在会话被释放后引发。会话被终止可能是由于超时，也可能是由于 Abandon 方法被显式地调用。只有在进程中维护会话状态的应用程序，才会引发该事件。

18.5 HttpApplication 的事件

之前列出的事件都位于 Visual Studio 生成的 HttpApplication 派生类中(根据项目类型，可能位于 Global.asax，也可以位于 Global.asax/Global.asax.cs 文件中)。应用程序对象还能够引发其他许多事件。表 18.1 总结了应用程序对象能够引发的所有事件，其中的一些只能在 Global.asax 中处理，其他的则可以在 HttpModule(HTTP 模块)中处理。

表 18.1 应用程序范围的事件

事 件	引发的原因	顺 序	是否只能在 Global.asax 中处理
Application_Start	应用程序启动	应用程序启动时运行一次	是
Application_End	应用程序结束	应用程序结束时运行一次	是
Session_Start	会话开始		是
Session_End	会话结束		是
BeginRequest	收到新的请求	1	否
AuthenticateRequest/ PostAuthenticateRequest	用户通过身份验证(即用户的安全身份已确定)	2	否

续表

事 件	引发的原因	顺 序	是否只能在 Global.asax 中处理
AuthorizeRequest/ PostAuthorizeRequest	用户已被授权使用请求的资源	3	否
ResolveRequestCache/ PostResolveRequestCache	在授权用户后、调用处理程序之前引发。输出缓存就是在这里被处理的。如果存在缓存的输出，那么应用程序会绕过整个页面呈现过程	4	否
AcquireRequestState/ PostAcquireRequestState	在需要初始化会话状态时引发	5	否
PreRequestHandlerExecute	在请求即将被发送到处理程序前引发。在输出发送给客户端之前，这是对其进行修改的最后机会	6	否
PostRequestHandlerExecute	在输出发送给客户端之后引发	7	否
ReleaseRequestState/ PostReleaseRequestState	在开始处理请求之后引发。在必要时，系统会通过该事件来保存其使用的状态	8	否
UpdateRequestCache/ PostUpdateRequestCache	在处理程序执行后引发。缓存模块通过该事件来缓存输出	9	否
EndRequest	请求处理完毕	10	否
Disposed	在应用程序结束之前引发	应用程序结束时引发	否
Error	出现未处理异常	出现未处理异常时引发	否
PreSendRequestContent	在将内容发送给客户端之前引发		否
PreSendRequestHeaders	在将 HTTP 标头发送给客户端之前引发		否

下面我们通过练习，来了解一下如何在 Global.asax 中通过监听 BeginRequest 和 EndRequest 事件来实现请求处理的计时。

➤ 对请求处理计时

1. 打开 UseApplication 项目中的 Global.asax.cs 文件。
2. 找到 Application_BeginRequest 处理程序。该处理程序是 Visual Studio 自动生成的。不过，Application_EndRequest 并没有自动添加，因而需要手动输入：

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

3. 实现 `Application_BeginRequest` 处理程序。将当前的日期和时间存储在当前 `HttpContext` 的 `Items` 属性中。`Items` 属性是一个键/值集合，可以通过索引值进行访问，这与缓存、会话状态和 `HttpApplication` 字典一样。实现 `EndRequest` 处理程序。将请求开始的时间戳与当前日期和时间进行比较。通过 `Response.Write` 输出时间差。

```
protected void Application_BeginRequest(object sender, EventArgs e)
{
    DateTime dateTimeBeginRequest = DateTime.Now;

    HttpContext ctx = HttpContext.Current;
    ctx.Items["dateTimeBeginRequest"] = dateTimeBeginRequest;
}

protected void Application_EndRequest(object sender, EventArgs e)
{
    DateTime dateTimeEndRequest = DateTime.Now;

    HttpContext ctx = HttpContext.Current;
    DateTime dateTimeBeginRequest =
        (DateTime)ctx.Items["dateTimeBeginRequest"];

    TimeSpan duration = dateTimeEndRequest - dateTimeBeginRequest;

    Response.Write("<b>From Global.asax: This request took " +
        duration.ToString() + "</b></br>");
}
```

这个时间间隔会被输出到发送给浏览器的响应中。

18.6 HttpModule

通过 `Global.asax.cs` 中的全局事件可以方便地管理应用程序中的数据和事件。Visual Studio 会生成 `Global.asax.cs` 和重要事件的处理程序存根。然而，使用 `Global.asax.cs` 不是存储状态和处理应用程序范围事件的唯一方式。我们还可以编写 HTTP 模块。

HTTP 模块与针对传统 ASP 的“Internet 服务器应用程序编程接口”(Internet Server Application Programming Interface, ISAPI)筛选程序的作用类似——用于扩展请求处理。HTTP 模块能够作为插件与 ASP.NET 处理过程融合来处理应用程序范围的事件(就像在 `Global.asax` 中一样)。事实上，许多 ASP.NET 功能都是通过 HTTP 模块实现的。在我们继续下面的内容之前，有一点需要注意：在请求处理过程中添加过多插件会降低应用程序的性能。不过，也要知道，HTTP 模块体现了 ASP.NET 的灵活性。

18.6.1 现有的模块

ASP.NET 通过 HTTP 模块来支持输出缓存和会话状态这样的功能。要想了解 HTTP 模块所实现的功能，可以查看计算机的主配置文件(找到 Windows 目录、Microsoft.NET 目录，然后打开最新的版本)。清单 18.2 列出了主 web.config 文件 httpModules 节定义的模块。为简明起见，这个列表没有提供程序集的完整名称，但可以从了解到 ASP.NET 管线中现有的模块。

清单 18.2 主 web.config 文件中的 HttpModules 节

```
<httpModules>
  <add name="OutputCache" type="System.Web.Caching.OutputCacheModule" />
  <add name="Session" type="System.Web.SessionState.SessionStateModule" />
  <add name="WindowsAuthentication"
type="System.Web.Security.WindowsAuthenticationModule" />
  <add name="FormsAuthentication"
type="System.Web.Security.FormsAuthenticationModule" />
  <add name="PassportAuthentication"
type="System.Web.Security.PassportAuthenticationModule" />
  <add name="RoleManager" type="System.Web.Security.RoleManagerModule" />
  <add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationModule" />
  <add name="FileAuthorization" type="System.Web.Security.FileAuthorizationModule" />
  <add name="AnonymousIdentification"
type="System.Web.Security.AnonymousIdentificationModule" />
  <add name="Profile" type="System.Web.Profile.ProfileModule" />
  <add name="ErrorHandlerModule"
type="System.Web.Mobile.ErrorHandlerModule, System.Web.Mobile,
    Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  <add name="ServiceModel" type="System.ServiceModel.Activation.HttpModule,
    System.ServiceModel.Activation,
Version=4.0.0.0,
    Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
  <add name="UrlRoutingModule-4.0" type="System.Web.Routing.UrlRoutingModule" />
  <add name="ScriptModule-4.0"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
    Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
</httpModules>
```

httpModules 节包含每个模块的名称，随后便是实现该功能的类型。这些模块实现了以下功能：

- 输出缓存
- 会话状态
- Windows 身份验证

- Forms 身份验证
- Passport 身份验证
- 角色管理
- URL 授权
- 文件授权
- 匿名身份
- 配置文件
- ErrorHandlerModule
- ServiceModule-4.0
- ScriptModule-4.0

第2章简单介绍了ASP.NET 管线。模块能够融入请求处理过程中,并在处理 `HttpApplication` 对象之前执行。事实上,“Internet 信息服务”(IIS)7.x 广泛地应用了模块(尤其是在“集成”模式下运行时)。虽然这些功能本身需要很多代码来实现(例如,可以设想一下实现会话状态管理程序的代码),但将模块融入应用程序却十分简单,只有以下4个基本步骤:

1. 编写实现 `IHttpModule` 接口的类
2. 编写相关事件的处理程序
3. 订阅事件
4. 在 `web.config` 中配置模块

18.6.2 模块的实现

下面的示例将演示 HTTP 模块的工作方式。本章之前的练习演示了如何通过 `Global.asax` 中的事件处理程序来对请求处理进行计时:在请求处理开始时创建时间戳,将其存储在当前的 `HttpContext` 中,并在请求处理结束后将这个时间戳与当前时间进行比较。

下面的练习实现的功能与之前练习实现的功能类似,只不过这里使用 HTTP 模块来处理相关事件。

► 实现计时模块

1. 为实现计时模块,打开本章的 Web 应用程序解决方案——`UseApplication`。模块需要包含于程序集中才能起作用,而最简单的方式是单独为该模块创建一个程序集。在主

菜单上单击“文件”|“添加”|“新建项目”。选择“类库”项目，并将该项目命名为 TimingModule。

2. Visual Studio 会在 Class1.cs 文件中添加一个名为 Class1 的类。将该文件重命名为 Timer.cs，将其中的类重命名为 Timer。确保这个类位于 TimingModule 命名空间中。
3. Visual Studio 生成的模块默认不识别 ASP.NET 类型，因此需要添加对 System.Web 程序集的引用。
4. 为请求的开始和结束事件添加处理程序。一部分代码可以从 Global.asax 复制过来。这两个事件处理程序接受 object 和 EventArgs 类型的参数，返回类型为 void。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;

///<summary>
/// Summary description for Timer
///</summary>
namespace TimingModule {
    public class Timer
    {
        public Timer()
        {
        }

        public void OnBeginRequest(object o, EventArgs ea)
        {
            DateTime dateTimeBeginRequest = DateTime.Now;

            HttpContext ctx;
            ctx = HttpContext.Current;
            ctx.Items["dateTimeBeginRequest"] = dateTimeBeginRequest;
        }

        public void OnEndRequest(object o, EventArgs ea)
        {
            DateTime dateTimeEndRequest = DateTime.Now;

            HttpContext ctx;
            ctx = HttpContext.Current;
            DateTime dateTimeBeginRequest =
                (DateTime)ctx.Items["dateTimeBeginRequest"];

            TimeSpan duration = dateTimeEndRequest - dateTimeBeginRequest;

            ctx.Response.Write("<b>From the TimingModule: This request took " +
                duration.ToString() + "</b><br>");
        }
    }
}
```

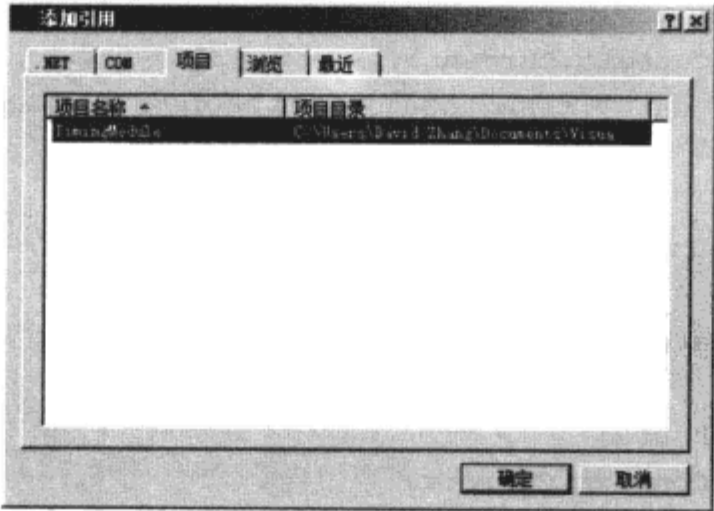


```
    }  
}
```

5. 使这个类继承于 IHttpModule 接口。在编辑器中右击 IHttpModule，然后选择“实现接口” Init 执行的工作是订阅事件，Dispose 执行的工作是释放模块占用的资源。(在本示例中，Dispose 方法不需要执行任何操作。)

```
public class Timer  
    : IHttpModule  
{  
    public Timer()  
    {  
    }  
  
    public void Init(HttpApplication httpApp)  
    {  
        httpApp.BeginRequest +=  
            new EventHandler(this.OnBeginRequest);  
  
        httpApp.EndRequest +=  
            new EventHandler(this.OnEndRequest);  
    }  
    public void Dispose() { }  
  
    //...  
}
```

6. 使网站引用刚刚创建的模块。为此，需要为 UseApplication 项目添加项目级的引用(引用新的模块)，以便在页面代码中使用它。在“解决方案资源管理器”中右键单击“UseApplication”节点，选择“添加引用”。在“添加引用”对话框中，单击“项目”选项卡，从列表中选择“TimingModule”，然后单击“确定”(如下图所示)。



7. 最后，需要在 web.config 文件中引用 TimingModule 模块。在 httpModules 节添加这个引用。此节位于 system.web 下，如下所示：

```
<configuration>  
  <system.web>  
    <httpModules>  
      <add name="TimingModule">
```

```
type="TimingModule.Timer, TimingModule" />
</httpModules>
</system.web>
</configuration>
```

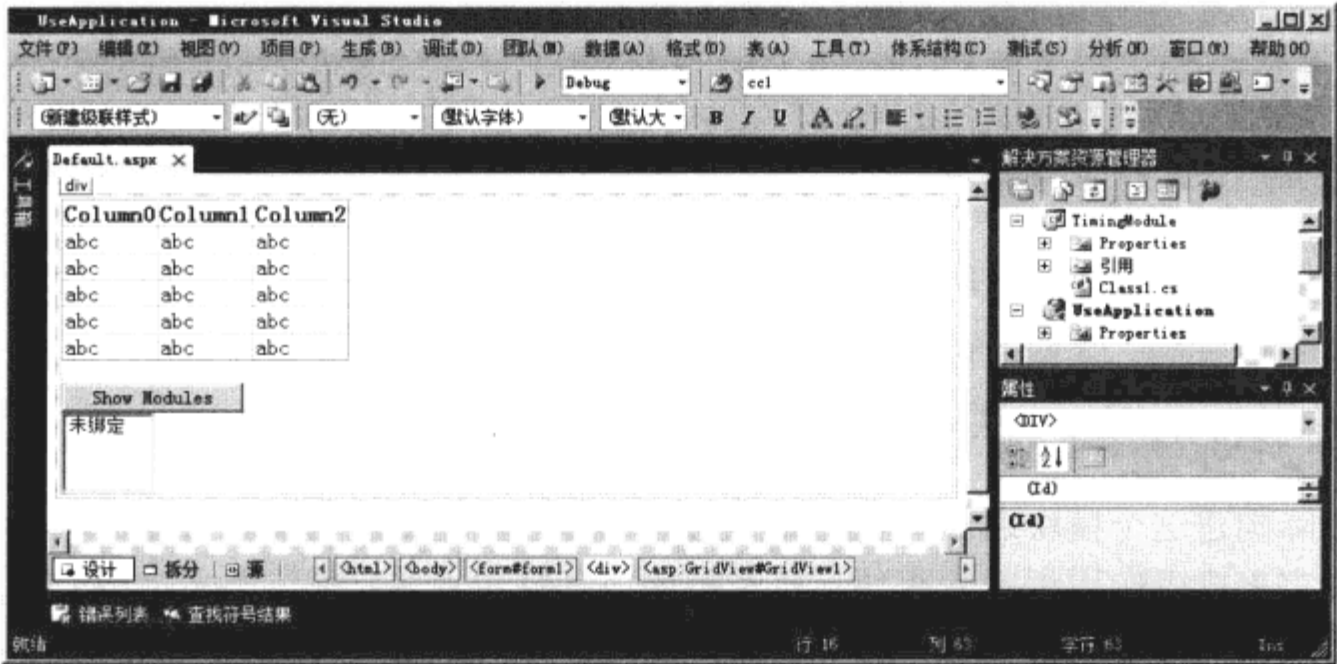
只要应用程序包含 TimingModule 程序集(即只要它位于虚拟目录的 Bin 子目录下), 那么该程序集便会融入管线。在运行程序后, 我们会看到来自 Global.asax.cs 和计时模块的计时信息。

18.6.3 查看活动的模块

ASP.NET 通过模块实现了许多功能。我们可以在主配置文件中看到这些模块, 也可以在运行时对其进行查看。为此, 我们需要使用当前应用程序的实例。下面的练习演示了如何查看活动的模块。

➤ 列出活动的模块

- 1. 为 UseApplication 项目的 Default.aspx 页面添加一个按钮(如下图所示)。这个按钮用于显示模块的列表。将这个按钮的 ID 设置为 ButtonShowmodules, 将 Text 属性设置为“Show Modules”。此外, 再添加一个用于显示模块名称的列表框。



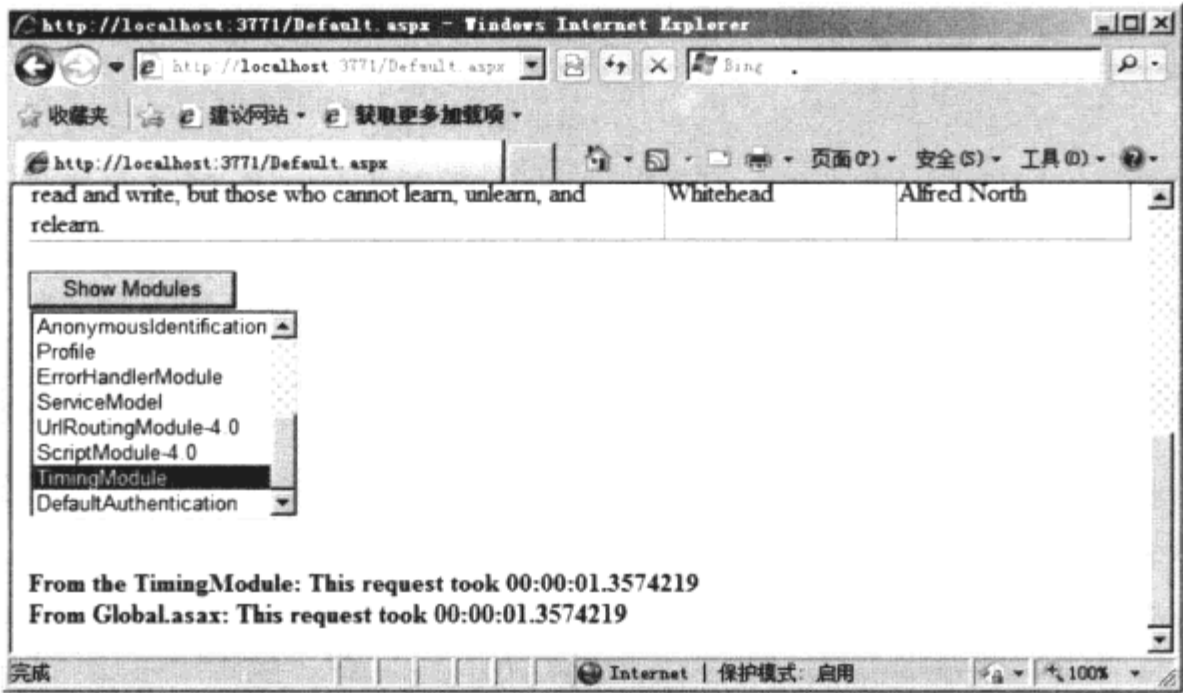
- 2. 双击按钮, 为其添加 Click 事件的处理程序。
- 3. 在这个处理程序中, 通过应用程序的实例获取模块列表。这个列表是以集合形式返回的, 这里将其赋给列表框的 DataSource 属性。调用 ListBox 的 DataBind 方法, ListBox 便会显示所有模块的名称。

```
protected void ButtonShowmodules_Click(object sender, EventArgs e)
{
    HttpApplication httpApp = HttpContext.Current.ApplicationInstance;
    HttpModuleCollection httpModuleColl = httpApp.Modules;

    Response.Write("<br>");
}
```

```
String[] rgstrModuleNames;  
rgstrModuleNames = httpModuleColl.AllKeys;  
  
this.ListBox1.DataSource = rgstrModuleNames;  
this.ListBox1.DataBind();  
}
```

4. 运行这个页面，单击“Show Modules”按钮。此时，列表框会列出附加在应用程序上的所有模块的名称(如下图所示)。TimingModule 模块也会列于其中。



18.6.4 在模块中存储状态

HTTP 模块非常适合存储应用程序的全局状态。下面的练习演示了如何跟踪请求处理的平均持续时间。为此，我们要在应用程序状态中存储每次请求处理的持续时间。

➤ 跟踪平均请求持续时间

1. 在实际构建这个模块之前，不妨先考虑一下如何来使用平均请求持续时间。我们可以通过它来排查应用程序的瓶颈。虽然将该信息发送到客户端浏览器会很有用，但有时需要以编程方式来使用它。为从模块中获取这个信息，我们需要为 TimingModule 添加若干方法(除 Init 和 Dispose 方法外)。实现这个功能的最佳方式是定义一个用于操纵此模块的接口。下面的代码定义了一个获取平均请求持续时间的接口。创建一个名为 ITimingModule.cs 的文件，并将其添加到 TimerModule 子项目中：

```
public interface ITimingModule  
{  
    TimeSpan GetAverageLengthOfRequest();  
}
```

2. 在 Timer 类中实现 ITimingModule 接口。为 Timer 类添加一个 ArrayList 类型的成员来存储每次请求处理的持续时间。(为此，需要通过 using 指令来引用 System.Collections 命名空间。)通过 OnEndRequest 处理程序，在每个请求结束时存储请求的持续时间。

这里用“计时周期”(tick)为单位来计算请求的平均持续时间。最后，我们实现 `GetAverageLengthOfRequest` 方法(`ITimingModule` 接口定义的方法)，具体而言，就是所有元素都添加到 `ArrayList` 中，所有持续时间的总和除以 `ArrayList` 的大小。根据计算结果创建一个 `TimeSpan` 对象，并将其返回给客户端。

```
using System.Collections;

public class Timer : IHttpModule, ITimingModule
{
    public Timer()
    {
    }

    protected ArrayList _alRequestDurations = new ArrayList();
    public void OnBeginRequest(object o, EventArgs ea)
    {
        DateTime dateTimeBeginRequest = DateTime.Now;
        HttpContext ctx;
        ctx = HttpContext.Current;
        ctx.Items["dateTimeBeginRequest"] = dateTimeBeginRequest;
    }

    public void OnEndRequest(object o, EventArgs ea)
    {
        DateTime dateTimeEndRequest = DateTime.Now;

        HttpContext ctx;
        ctx = HttpContext.Current;
        DateTime dateTimeBeginRequest =
            (DateTime)ctx.Items["dateTimeBeginRequest"];

        TimeSpan duration = dateTimeEndRequest - dateTimeBeginRequest;

        ctx.Response.Write("<b>From the TimingModule: This request took " +
            duration.ToString() + "</b></br>");

        _alRequestDurations.Add(duration);
    }

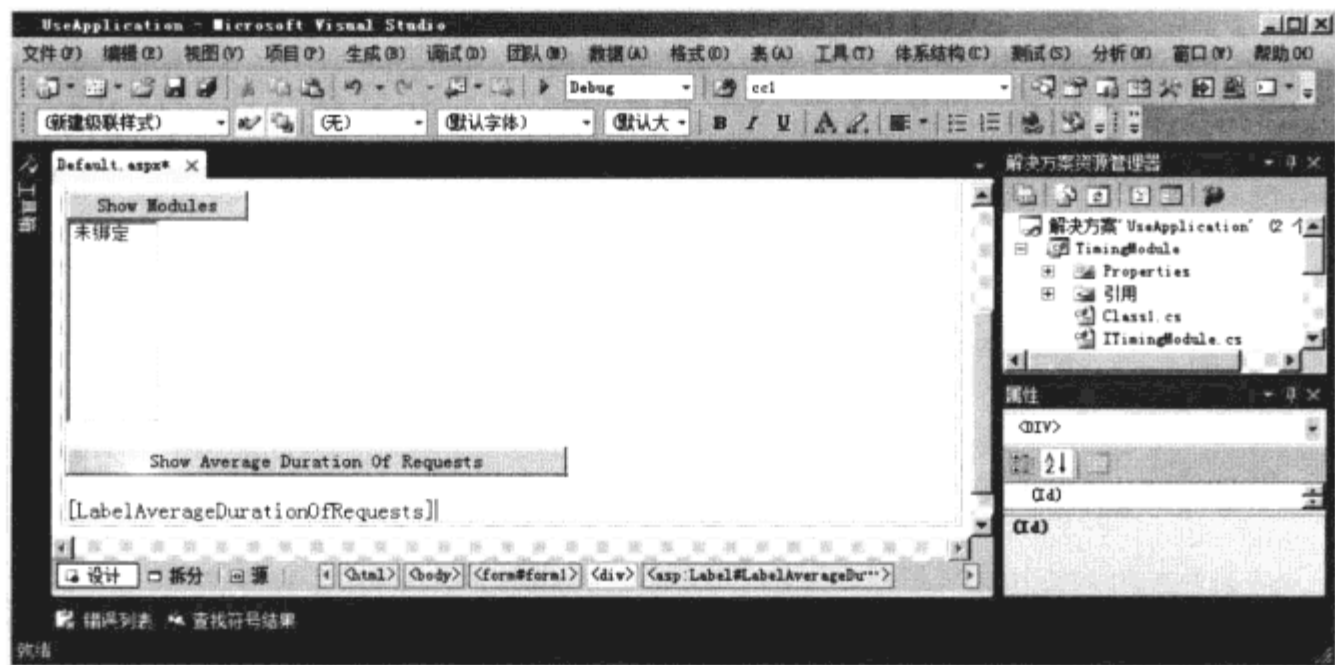
    public void Init(HttpApplication httpApp)
    {
        httpApp.BeginRequest +=
            new EventHandler(this.OnBeginRequest);

        httpApp.EndRequest +=
            new EventHandler(this.OnEndRequest);
    }

    public void Dispose() { }
```

```
public TimeSpan GetAverageLengthOfRequest()  
{  
    long lTicks = 0;  
    foreach (TimeSpan timespanDuration in this._alRequestDurations)  
    {  
        lTicks += timespanDuration.Ticks;  
    }  
  
    long lAverageTicks = lTicks / _alRequestDurations.Count;  
    TimeSpan timespanAverageDuration = new TimeSpan(lAverageTicks);  
    return timespanAverageDuration;  
}  
}
```

3. 在 Default.aspx 页面中添加一些代码来获取请求的平均持续时间。添加一个用于获取平均持续时间的按钮，再添加一个显示这个值的标签。将按钮的 Text 属性设置为“Show Average Duration Of Requests”（如下图所示），将 ID 设置为 ButtonShowAverageDurationOfRequests。将标签的 Text 属性置空，将 ID 设置为 LabelAverageDurationOfRequests。此外，在 Default.aspx 页面中添加对 TimingModule 的引用，以便通过代码访问该模块的接口。



4. 在 Visual Studio 的“设计器”中双击 Show Average Duration Of Requests 按钮，为其添加 Click 事件的处理程序。在这个处理程序中，从模块集合获取 TimingModule 的实例。由于模块是基于名称索引的，因而可以通过名称来查找 TimingModule（这个名称是在 web.config 中指定的）。

```
using TimingModule;
```

```
...
```

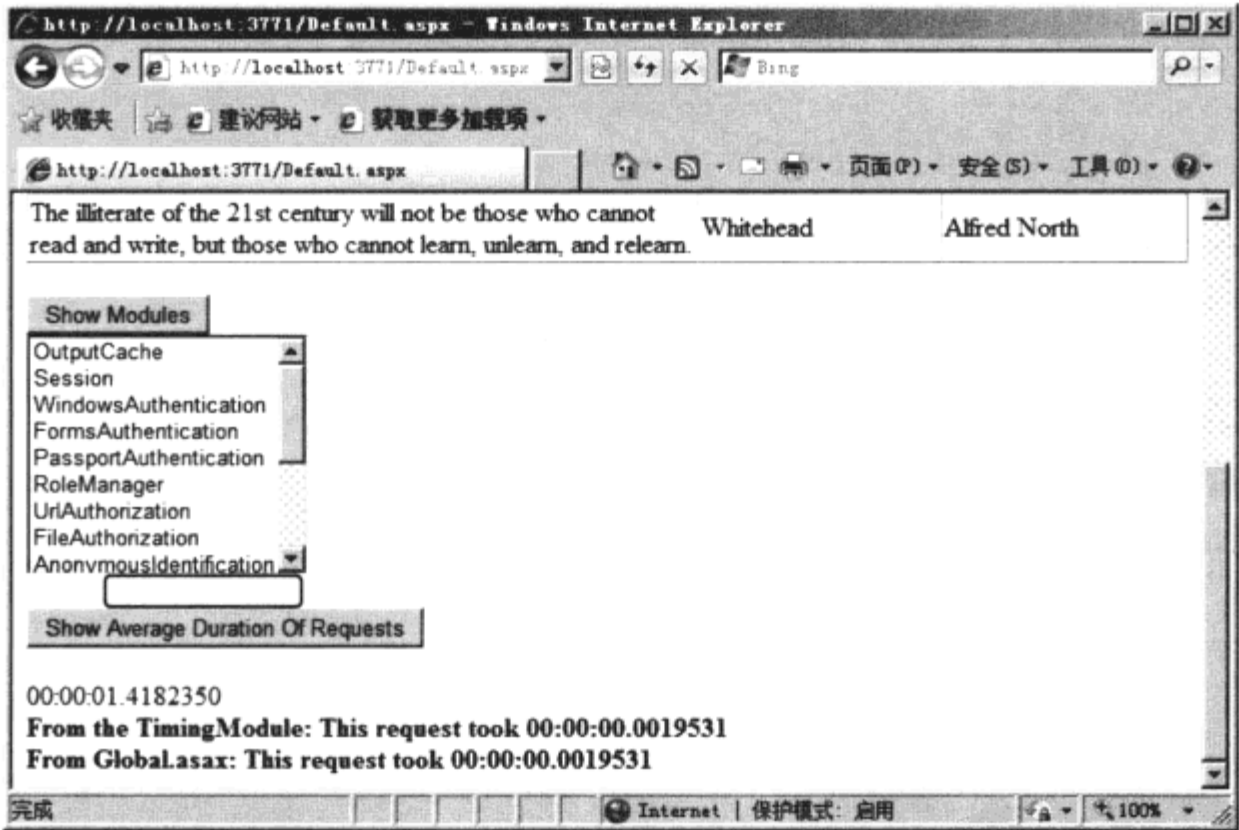
```
protected void  
    ButtonShowAverageDurationOfRequests_Click(  
        object sender,  
        EventArgs e)
```

```
{
    HttpApplication httpApp =
        HttpContext.Current.ApplicationInstance;

    HttpModuleCollection httpModuleColl = httpApp.Modules;
    IHttpModule httpModule =
        httpModuleColl.Get("TimingModule");
    ITimingModule TimingModule =
        (ITimingModule)httpModule;

    TimeSpan timeSpanAverageDurationOfRequest =
        TimingModule.GetAverageLengthOfRequest();
    LabelAverageDurationOfRequests.Text =
        timeSpanAverageDurationOfRequest.ToString();
}
```

模块集合返回的是 HttpModule 对象。为通过 ITimingModule 接口来操纵它，需要进行类型转换。之后，我们便可以调用 GetAverageLengthOfRequest 方法，并通过标签来显示返回值(如下图所示)。



18.7 Global.asax 与 HttpModule

Global.asax 定义的应用程序对象和通过 HTTP 模块创建的应用程序对象都能够提供应用程序的全局访问点。两者都可以存储请求间的全局状态和响应应用程序范围的事件。在选择时应记住，Global.asax 文件只能伴随应用程序，它旨在管理特定于应用程序的状态和事件。HTTP 模块则是完全独立的程序集，不必与特定应用程序绑定。这种程序集甚至可以被签名并部署到全局程序集缓存。因此，模块更适合实现多个应用程序通用的功能。

18.8 快速参考

目 标	操 作
创建自定义的模块(程序集)	添加一个实现 IHttpModule 接口的类，并实现其 Init 和 Dispose 方法
将模块融入 ASP.NET 管线	在应用程序的 web.config 文件中配置 httpModules 节
在模块中处理应用程序事件	在模块中为每个要处理的事件编写处理程序。在 Init 方法中订阅这些事件
重写 Global.asax 文件中的应用程序对象	单击“项目”/“网站” “全局应用程序类”文件模板。添加用于响应应用程序范围事件的代码
使用应用程序字典	获取应用程序对象(通过当前的 HttpContext 对象)，使用索引符号来访问字典

第 19 章

HTTP 处理程序

学习目标

- 理解自定义处理程序在 ASP.NET 中的作用
- 编写自定义的处理程序
- 编写即时编译的自定义处理程序
- 在网站中引用自定义的处理程序


本章将介绍如何编写自定义的 HTTP 处理程序。第 2 章解释了 ASP.NET 管线。ASP.NET 中处理请求的端点 Cendpoint 都是 IHttpHandler 接口的实现。

ASP.NET 包含几个常用的请求处理程序。例如，Page 类会解释查询字符串，并返回有价值的、面向用户界面的 HTML，以这种方式来处理请求。Service 类能够以方法调用的方式来解释传入的查询字符串，并调用相应的过程。到目前为止，本书只介绍了一种处理程序——System.Web.UI.Page。然而，我们有时需要改变请求处理过程，甚至采用一种完全不同的处理方式。此时，我们会发现 System.Web.UI.Page 或 System.Web.Services.Service 类无法满足请求处理的要求。那该怎么办呢？为此，ASP.NET 提供了对自定义 HTTP 处理程序的支持。

19.1 ASP.NET 请求处理程序

本书到目前为止一直将重点放在 Page 类上。Page 类主要负责管理应用程序的 UI。UI 的处理非常复杂(大部分是模板式的代码)，但 Page 类内建了许多功能来简化开发，能够满足绝大多数用户界面处理的需要。

接触过 Web 服务的开发者知道，WebService 类能够以方法调用的形式处理 HTTP 请求。客户端通过基于 SOAP 格式的 XML 来包装 Web 服务调用信息。

 **提示** SOAP 代表“简单对象访问协议”(Simple Object Access Protocol)，而在 SOAP 1.2 版本之后，SOAP 有了更多内涵——不再强调对象，而且这些对象也并不一定简单(至少在实现上)。

客户端调用 Web 服务的方式与其对网页发起 HTTP 请求的方式是相同的，只不过后者采用的是基于 GET 和 POST 命令的 HTTP 请求。当请求到达服务器后，服务器会将参数解包，并将其传入实际的或虚拟的调用堆栈，并最终调用目标方法。通过 HTTP 调用方法的大部分任务都会由 WebService 类处理，这非常好理解，基本原理也与 HTTP 请求相同。

正如第 2 章所介绍的，ASP.NET 中 HTTP 请求的端点是实现了 `IHttpHandler` 接口的类。`IHttpHandler` 是一个非常简单的接口，只包含两个方法。然而，实现了这个接口的类都能够以 HTTP 处理程序的形式融入 HTTP 管线。本章会详细介绍这个接口。

HTTP 处理程序只不过是实现 `IHttpHandler` 的类(这与 HTTP 模块是实现 `IHttpModule` 的类一样)。HTTP 处理程序需要在 `web.config` 中引用。与 HTTP 模块一样，ASP.NET 提供了几个默认的 HTTP 处理程序(如跟踪和阻止对网站中敏感文件的访问)，并已在主 `web.config` 文件中引用(这个文件与主配置目录中的 `machine.config` 在一起)。

到目前为止，`ASPX`、`ASAX` 和 `ASCX` 文件在 ASP.NET 中能够奇迹般地工作。例如，正如之前介绍的，浏览 `ASPX` 文件会使 ASP.NET 立即生成派生于 `System.Web.UI.Page` 的类，并对其进行编译。`ASPX` 文件之所以这样工作是因为 ASP.NET 包含相应功能的处理程序。

ASP.NET 在 `web.config` 中引用处理程序的方式与引用 HTTP 模块类似。处理程序要通过 `add` 特性来添加，它涉及 4 个特性。首先是 `path`，用于指定处理程序所针对的文件名和/或扩展名。请记住，HTTP 请求会以资源请求的形式传入服务器(HTTP 协议允许请求包含资源名)。第二个特性是 `verb`，用于指定处理程序针对的谓词的列表。这些谓词遵循 HTTP 规范。例如，我们可以使处理程序只响应 `GET` 请求，而不响应 `POST` 请求。也可以使处理程序响应所有类型的请求。第三个特性是 `type`，用于指定实现处理程序的 .NET 类型的名称。最后是 `validate` 特性，用于指定处理程序对象是在应用程序启动时加载，还是请求到达时加载。

清单 19.1 展示了 ASP.NET 主 `web.config` 文件中的部分 HTTP 处理程序。

清单 19.1 主 `web.config` 文件中引用的部分处理程序

```
<httpHandlers>
  <add path="trace.axd" verb="*"
      type="System.Web.Handlers.TraceHandler" validate="True" />
  <add path="WebResource.axd" verb="GET"
      type="System.Web.Handlers.AssemblyResourceLoader" validate="True" />
  <add verb="*" path="*_AppService.axd"
      type="System.Web.Script.Services.ScriptHandlerFactory,
      System.Web.Extensions, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35" validate="False" />
  <add verb="GET,HEAD" path="ScriptResource.axd"
      type="System.Web.Handlers.ScriptResourceHandler,
      System.Web.Extensions, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=31bf3856ad364e35" validate="False"/>
  <add path="*.axd" verb="*" type="System.Web.HttpNotFoundHandler" validate="True" />
  <add path="*.aspx" verb="*" type="System.Web.UI.PageHandlerFactory" validate="True" />
  <add path="*.ashx" verb="*" type="System.Web.UI.SimpleHandlerFactory" validate="True" />
  <add path="*.asmx" verb="*"
      type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
      Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" validate="False" />
  <add path="*.rem" verb="*"
      type="System.Runtime.Remoting.Channels.Http.HttpRemotingHandlerFactory,
      System.Runtime.Remoting, Version=4.0.0.0, Culture=neutral,
      PublicKeyToken=b77a5c561934e089" validate="False" />
  <add path="*.soap" verb="*" />
```



```
type="System.Runtime.Remoting.Channels.Http.HttpRemotingHandlerFactory,
System.Runtime.Remoting, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" validate="False" />
<add path="*.asax" verb="*" type="System.Web.HttpForbiddenHandler"
validate="True" />
<add path="*.ascx" verb="*" type="System.Web.HttpForbiddenHandler" validate="True" />
<add path="*.master" verb="*" type="System.Web.HttpForbiddenHandler" validate="True" />
<add path="*.skin" verb="*" type="System.Web.HttpForbiddenHandler" validate="True" />
<add path="*.browser" verb="*" type="System.Web.HttpForbiddenHandler" validate="True" />
<add path="*.sitemap" verb="*" type="System.Web.HttpForbiddenHandler" validate="True" />
<add path="*.dll.config" verb="GET,HEAD" type="System.Web.StaticFileHandler"
validate="True" />
<!-- More handlers follow... -->
</httpHandlers>
```

下面将通过两个处理程序(Trace 和 Forbidden)来说明实现单独的请求处理功能(即与 UI 或 Web 服务没有必然联系的请求)是非常有好处的。

19.2 内建的处理程序

学习 ASP.NET 内建的 Trace 处理程序有助于理解自定义处理程序。第 17 章介绍了跟踪。我们可以通过在 web.config 中插入跟踪元素(<trace enabled=true />)来启用跟踪。如果跟踪被启用，那么 ASP.NET 运行库会存储网站请求的摘要，供开发者在诊断问题时查看。

ASP.NET 会在内存中缓存跟踪信息。为查看跟踪结果，只需要请求网站虚拟目录的一个特殊资源——Trace.axd。清单 19.1 中的第一项就是针对 Trace.axd 资源的 HTTP 处理程序。ASP.NET 幕后的跟踪功能不涉及 UI 处理，因而最适合以自定义处理程序的形式来实现。

在浏览 Trace.axd 资源时，处理程序会呈现图 19.1 所示的输出。此处理程序的功能非常单一——呈现最新请求的列表。



图 19.1 Trace.axd 处理程序的输出

如图 19.2 所示，单击图 19.1 中的第四个“查看详细信息”链接会提交查询字符串中包含 id=3 参数的请求。此时，Trace.axd 处理程序会呈现该请求的详细跟踪信息。



图 19.2 Trace.axd 针对特定请求的输出

图 19.3 展示了“Internet 信息服务”(IIS)针对.axd 文件的映射。此图反映出 ASP.NET 能够处理多种请求，不过在部署网站之前我们无法看到这些信息。IIS 处理 Trace.axd 请求的方式与其他 ASP.NET 请求的处理方式完全相同。也就是说，IIS 会将带有.axd 扩展名的资源交给 ASP.NET。一旦进入 ASP.NET 管线，ASP.NET 便会查询 web.config 文件，并通过 Trace 处理程序来处理请求。



图 19.3 IIS 中 Trace.axd 映射的处理程序

如果仔细查看默认的 web.config 文件，还会发现一些重要的 ASP.NET 处理程序。显然，在默认情况下，Web 服务器应禁止客户端访问源代码。从配置文件中可以看到，*.cs、*.config 和*.vb 都会由 Forbidden 处理程序处理。如果在浏览器中请求这些类型的资源，那么 ASP.NET 默认会返回图 19.4 所示的错误。

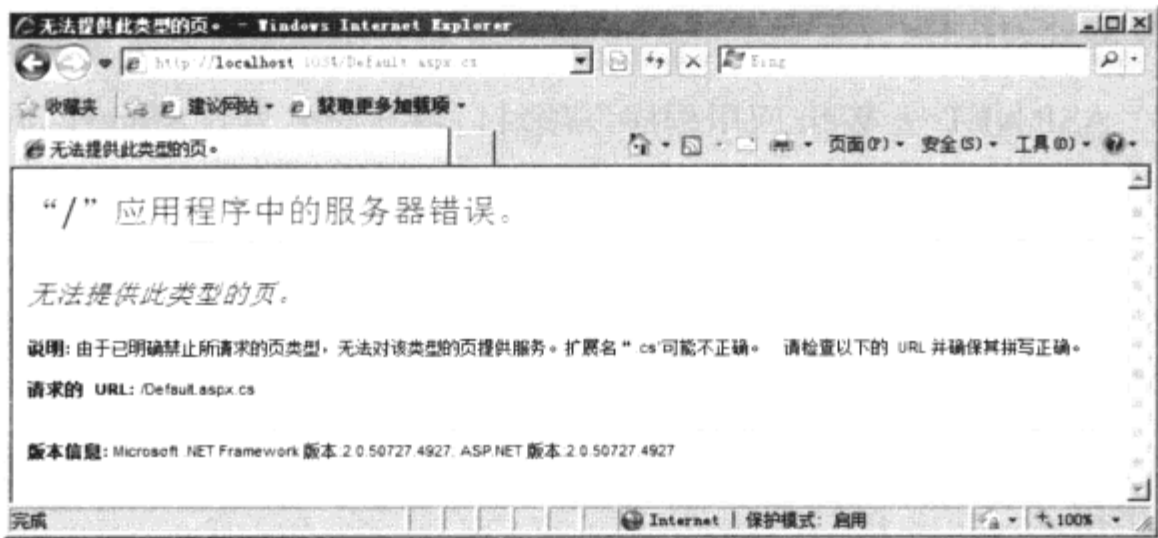


图 19.4 尝试查看禁止的资源时显示的页面

ASP.NET 的配置机制十分灵活，可以通过两种方法来允许客户端查看源代码：可以移除 IIS 中源代码扩展名与 ASP.NET 处理程序之间的映射，也可以编写自定义的源代码查看器处理程序，并在应用程序的 web.config 文件中引用。

融入管线的处理程序需要实现 IHttpHandler 接口。下一节就将介绍这个重要的接口。

19.3 处理程序与 IHttpHandler

清单 19.2 展示了 IHttpHandler 接口的定义，这是一个非常简单的接口。

清单 19.2 IHttpHandler 接口

```
public interface IHttpHandler
{
    void ProcessRequest(HttpContext ctx);
    bool IsReusable {get;}
}
```

这个接口虽然简单，但还是有几点需要说明。它包含一个名为 ProcessRequest 的方法和一个名为 IsReusable 的属性。如果处理程序的实例可以被利用多次，则 IsReusable 应返回 true。如果处理程序通常返回静态的内容，则可以复用其实例。如果内容是动态的，那么它的实例可能是不可复用的。处理程序的核心是接受一个 HttpContext 参数的 ProcessRequest 方法。

当请求(通过 ProcessRequest 方法)传入处理程序后，ProcessRequest 便可以以任意方式进行响应。Trace.axd 处理程序的响应方式是列出运行库跟踪的请求；Forbidden 处理程序的响应方式是阻止客户端查看被禁止的资源；自定义 Web 服务的响应方式是解析 XML 有效载荷，生成调用堆栈，并调用内部方法。

实现 IHttpHandler 非常简单(至少从结构上是这样)。ProcessRequest 方法接受一个参数——当前的 HttpContext。不过，ProcessRequest 方法内部的代码可以执行任意操作，而某些实现可能非常复杂。下面的练习将完成整个呈现过程：通过组合框显示一些选项，允许最终用户进行选择，并呈现选定的项目。

➤ 编写自定义的处理程序

1. 创建一个“ASP.NET 空 Web 应用程序”项目，将其命名为 CustomHandlers。
2. 在 CustomHandlers 解决方案中添加一个类库子项目(就像在上一章创建 HTTP 模块时所做的)。将这个子项目命名为 CustomFormHandlerLib。Visual Studio 会自动生成一个名为 Class1 的类。将其重命名为 CustomFormHandler，将所属文件重命名为 CustomFormHandler.cs。
3. 由于 Visual Studio 生成的类库默认不识别 ASP.NET 类型，因而需要添加 System.Web 程序集的引用。
4. 为使 CustomFormHandler 成为处理程序，需添加 IHttpHandler 接口，并实现 ProcessRequest 方法。添加一个名为 ManageForm 的方法，使其接受一个 HttpContext 类型的参数。使 ManageForm 方法通过 Response.Write 依次输出<html>、<body>和<form>标记。输出一个问句：“Hello there. What’s cool about .NET?”。输出<select>标签，将其 name 特性设置为“Feature”。通过<option>标签来添加一些选项。这样，用户的浏览器便会显示一个下拉列表框。输出一个<input>标签，将其 type 特性设置为“submit”，将 name 和 value 特性都设置为“Lookup”。然后，在 HttpContext.Request.Params(参数)集合中查找“Feature”(<select>标签)的新值。如果这个值不是 null，则说明用户选中了某一项。将“Feature”中选定的项输出。最后输出闭标签——</form>、</body>和</html>。

使 ProcessRequest 方法调用 ManageForm 方法，如下所示：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;

public class CustomFormHandler : IHttpHandler
{
    public void ProcessRequest(HttpContext ctx)
    {
        ManageForm(ctx);
    }

    public void ManageForm(HttpContext context)
    {
        context.Response.Write("<html><body><form>");

        context.Response.Write(
            "<h2>Hello there. What's cool about .NET?</h2>");

        context.Response.Write(
            "<select name='Feature'>");
        context.Response.Write(
            "<option> Strong typing</option>");
        context.Response.Write(
            "<option> Managed code</option>");
```

```

        context.Response.Write(
            "<option> Language agnosticism</option>");

        context.Response.Write(
            "<option> Better security model</option>");
        context.Response.Write(
            "<option> Threading and async delegates</option>");
        context.Response.Write(
            "<option> XCOPY deployment</option>");
        context.Response.Write(
            "<option> Reasonable HTTP handling framework</option>");
        context.Response.Write("</select>");
        context.Response.Write("</br>");

        context.Response.Write(
            "<input type=submit name='Lookup' value='Lookup'></input>");
        context.Response.Write("</br>");

        if (context.Request.Params["Feature"] != null)
        {
            context.Response.Write("Hi, you picked: ");
            context.Response.Write(
                context.Request.Params["Feature"]);
            context.Response.Write(
                " as your favorite feature.</br>");
        }

        context.Response.Write("</form></body></html>");
    }

    public bool IsReusable
    {
        get
        {
            return true;
        }
    }
}


```

ProcessRequest 中的代码会呈现一个 form 元素和一个 select 元素，浏览器将显示一个可提交的表单。在表单被提交给服务器后，参数集合将包含“Features”元素。这段代码会检查参数集合是否包含选中的项目，如果存在，则将其显示出来。

5. 刚刚创建的这个类库的输出(CustomHandlerLib.dll)位于自身的项目目录下。为使ASP.NET 调用它，需要将可执行程序复制到网站的 bin 子目录中。这个动态链接库可以以项目引用的方式添加到网站中。在“解决方案资源管理器”中右键单击网站节点，选择“添加引用”。在“项目”选项卡中选择 CustomFormHandlerLib 项目，单击“确定”。
6. 更新 web.config 文件，以便应用程序在用户请求 CustomFormHandler 资源时予以处理。如果网站中没有 web.config 文件，添加一个即可。然后，在 httpHandlers 节中添加

CustomFormHandler 类的声明。

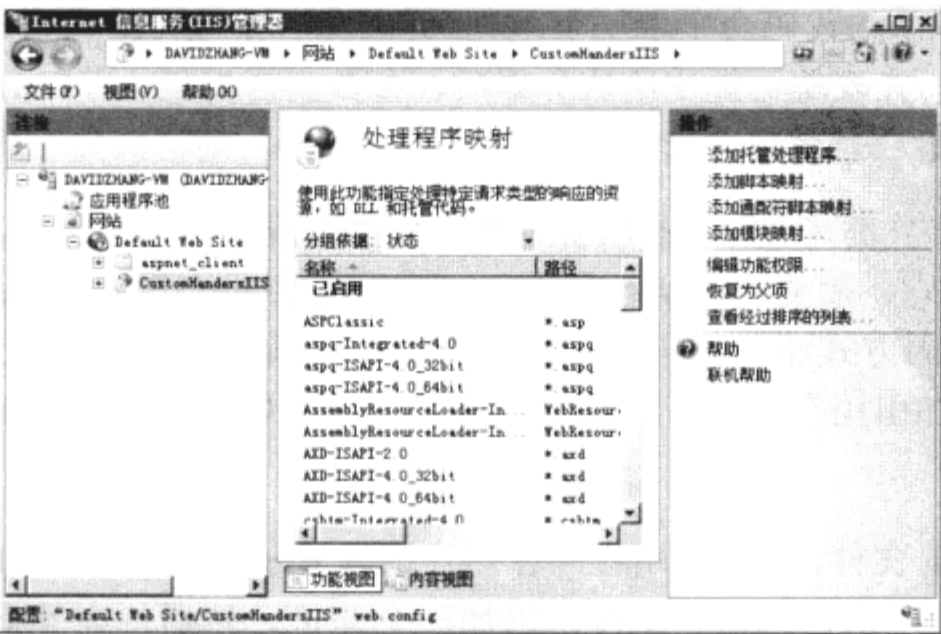
```
<configuration>
<system.web>
  <httpHandlers>
    <add path="*.cstm" verb="*"
      type="CustomFormHandlerLib.CustomFormHandler, CustomFormHandlerLib"
      validate="true" />
  </httpHandlers>
</system.web>
</configuration>
```

 **提示** 如果网站由 IIS 承载，则需要对 IIS 进行配置来使指定的文件类型由 CustomFormHandler 处理。如果决定在 IIS 下运行应用程序(而不使用 Visual Studio 的 Web 服务器)，则可以在了解 config.sys 的前提下对其直接进行编辑，或者通过以下方法进行设置：

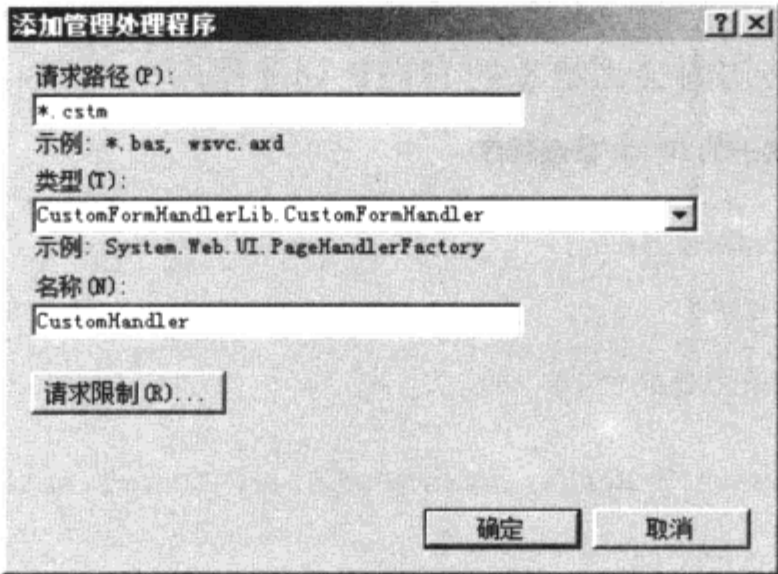
- (1) 打开 IIS，导航到网站的虚拟目录。
- (2) 打开“功能视图”，找到“处理程序映射”图标(如下图所示)。



(3) 双击“处理程序映射”便可以打开“处理程序映射”页面(见下图)。



- (4) 在“处理程序映射”页面的列表中右击，选择“添加托管处理程序”。
- (5) 输入希望映射到自定义处理程序的扩展名(如下图所示)。然后，指定处理程序的类型。IIS 会查找当前应用程序可用的处理程序(包括应用程序本身所引用的)。从下拉列表中选择图中所示的处理程序，为这个处理程序设置一个别名。这样，在用户浏览指定类型的文件时，这个处理程序便会被调用。



- (6) 最后，在项目中添加一个名为 CustomHandler.cstm 的空文本文件。可以通过这个带有.cstm 扩展名的文件来浏览刚刚创建的处理程序。
- (7) 浏览 CustomHandler.cstm 资源，ASP.NET 会调用刚才创建的这个自定义处理程序(如下图所示)。



当然，通过“Web 窗体”很容易实现这种功能，但这个示例体现了 ASP.NET 处理程序架构的灵活性，也可以加深我们对“Web 窗体”和自定义控件机制的理解。

19.4 处理程序与会话状态

第 14 章介绍了会话状态。System.Web.UI.Page 上下文中默认就可以使用会话状态。然而，如果要在自定义处理程序中使用会话状态，则需要显式地启用。



.NET 架构采用了一种名为“标记接口”(marker interfaces)的方法。标记接口是空的接口(没有定义任何方法或属性)，作用是为运行库提供某种信息。例如，ASP.NET 运行库经常使用这种接口来启用或禁用某些功能。如果运行库检测到对象的类层次结构中包含某个标记接口，那么运行库便会启用相应的功能。

处理程序要利用会话状态，它必须实现 `System.Web.SessionState.IRequiresSessionState` 接口。这样，运行库便会在请求的开始时加载会话状态，并在结束时将其保存。

清单 19.3 列举了一个能够访问会话状态的 HTTP 处理程序。

清单 19.3 一个能够访问会话状态的 HTTP 处理程序

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.SessionState;

public class HandlerWithSessionState : IHttpHandler, IRequiresSessionState
{
    public void ProcessRequest(HttpContext ctx)
    {
        string strData = (string)ctx.Session["SomeSessionData"];

        if (String.IsNullOrEmpty(strData))
        {
            strData = "This goes in session state";
            ctx.Session["SomeSessionData"] = strData;
        }
        ctx.Response.Write("This was in session state: " + strData);
    }

    public bool IsReusable
    {
        get
        {
            return true;
        }
    }
}
```

19.5 一般处理程序(ASHX 文件)

ASPX 文件能够被即时(just in time)编译，而这也适用于处理程序。承载一般处理程序的文件扩展名为 ASHX。这种处理程序与用 C#或 Visual Basic 实现的 `IHttpHandler` 自定义处理程序是等价的，并且实现这种处理程序与实现 ASPX 文件一样方便。在用户浏览 ASHX 文件时，它们会被自动编译。

下面的练习演示了以“一般处理程序”形式实现的 CustomFormHandler。

➤ 编写一般处理程序

1. 为网站添加一个“一般处理程序”。在“解决方案资源管理器”的“CustomHandlers”项目节点上右击，选择“添加”|“新建项”。从模板中选择“一般处理程序”，将这个处理程序命名为 CustomFormHandler.ashx。



2. Visual Studio 会生成一个处理程序，其中包括 ProcessRequest 方法的存根和完整的 IsReusable 属性。打开这个处理程序的代码隐藏文件(文件名为 CustomFormHandler.ashx.cs)，编写一个处理表单的方法(可以从上一个练习复制过来)。在 ProcessRequest 中调用这个方法。根据实际的实现修改存根方法和属性。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CustomHandlers
{
    ///<summary>
    ///CustomFormHandler 的摘要说明
    ///</summary>
    public class CustomFormHandler : IHttpHandler
    {
        public void ProcessRequest(HttpContext context)
        {
            ManageForm(context);
        }

        public void ManageForm(HttpContext context)
```




```

{
    context.Response.Write("<html><body><form>");
    context.Response.Write(
        "<h2>Hello there. What's cool about .NET?</h2>");
    context.Response.Write("<select name='Feature'>");
    context.Response.Write("<option> Strong typing</option>");
    context.Response.Write("<option> Managed code</option>");
    context.Response.Write("<option> Language agnosticism</option>");
    context.Response.Write("<option> Better security model</option>");
    context.Response.Write(
        "<option> Threading and async delegates</option>");
    context.Response.Write("<option> XCOPY deployment</option>");
    context.Response.Write(
        "<option> Reasonable HTTP handling framework</option>");
    context.Response.Write("</select>");
    context.Response.Write("<br>");
    context.Response.Write(
        "<input type=submit name='Lookup' value='Lookup'></input>");
    context.Response.Write("</br>");
    if (context.Request.Params["Feature"] != null)
    {
        context.Response.Write("Hi, you picked: ");
        context.Response.Write(context.Request.Params["Feature"]);
        context.Response.Write(" as your favorite feature.</br>");
    }

    context.Response.Write("</form></body></html>");
}

public bool IsReusable
{
    get{ return true; }
}
}

```

3. 浏览 CustomFormHandler.ashx 文件。浏览器显示的页面与之前练习中 CustomFormHandler 类生成的页面相同(见下图)。



一般处理程序有两个优势。首先，它非常适合实现简单的处理程序，无需为处理请求而创建新的程序集。其次，不需要为这种处理程序配置 web.config 或 IIS(部署后)。也就是说，ASP.NET 和 IIS 知道在遇到.ashx 扩展名的请求时如何处理。安装 ASP.NET 时，便会在 IIS 中建立相关的映射。

然而，在 ASP.NET 项目中，ASHX 文件也具有 ASPX 和 ASCX 文件的某些限制。简单的一般处理程序要伴随项目，而自定义处理程序可以以程序集的形式部署，也可以以全局程序集的形式在企业中共享(即将强命名程序集置于全局程序集缓存中)。

19.6 快速参考

目 标	操 作
创建自定义处理程序程序集	创建一个继承于 IHttpHandler 接口的类，实现该接口中的 IsReusable 属性和 ProcessRequest 方法
在 ASP.NET 中添加文件与处理程序的映射	配置应用程序 web.config 中的 httpHandler 节
在 IIS 中添加文件与处理程序的映射	在“功能视图”中双击“处理程序映射”图标。在列表的任意位置右击，选择“添加托管处理程序”
创建简单的处理程序	在 Visual Studio 的主菜单中选择“项目” “网站” “添加新项”，在模板中选择“一般处理程序”模板。插入响应请求的代码

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

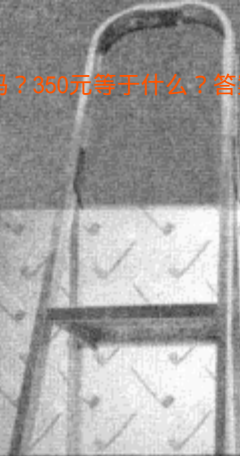


第 V 部分

动态数据、XBAP、MVC、 AJAX 和 Silverlight

- ▶ 第 20 章 动态数据
- ▶ 第 21 章 ASP.NET 与 WPF 内容
- ▶ 第 22 章 ASP.NET MVC 框架
- ▶ 第 23 章 AJAX
- ▶ 第 24 章 Silverlight 与 ASP.NET

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569



第 20 章 动 态 数 据

学习目标

- 理解“动态数据”的作用
- 理解“动态数据”的工作方式
- 使用“动态数据”功能创建网站
- 自定义“动态数据”网站的默认外观
- 为“动态数据”网站添加验证支持

ASP.NET 经过十余年的演进，其可用性和开发简易程度有了大幅提升。例如，ASP.NET 的母版页简化了一致观感的实现，避免开发者使用不成熟的继承机制。ASP.NET 身份验证和授权简化了安全性的管理。通过验证控件可以轻松实现数据验证。ASP.NET 数据源控件允许开发者在“设计器”中使用数据源。ASP.NET 4 还包含一个重要的可用性功能——“动态数据”(dynamic data)。“动态数据”通过丰富的框架扩展了 ASP.NET 中的 GridView 和 DetailsView 控件，从而简化了数据驱动网站的开发。(虽然“动态数据”技术已经存在一段时间了，但最近才得到 Visual Studio 的支持。)

第 10 章讨论了数据绑定。在 ASP.NET 中，某些控件可以直接绑定到数据源(如 DataGrid)。数据源控件的使用非常方便，因为不用编写太多代码便可以将数据呈现给最终用户。虽然 DataGrid 支持数据源并能够显示任何赋予它的数据，但其功能也仅限于此。许多开发者希望能够方便地为 DataGrid 添加验证支持。不幸的是 DataGrid 不支持该功能。不过，开发者现在可以通过 ASP.NET 动态数据构建数据驱动的网站，因为“动态数据”支持数据验证。

可以这样来理解 ASP.NET 动态数据：作为开发者，我们至少要知道网站应做成什么样子(至少有个概念)。在理解数据模型后，我们要创建这些数据的表示(例如人的表示)。DataGrid 能够通过反射推断数据结构，并以一种合理的方式将数据显示给最终用户。ASP.NET 动态数据功能延伸了这个过程，允许开发者根据数据模型来构建整个网站。

本章将先从相关控件开始介绍“动态数据”。

20.1 动态数据控件

ASP.NET 动态数据框架提供了 6 个专门支持“动态数据”的控件。这些控件是 ASP.NET 动态数据不可缺少的组成部分，是 ASP.NET 动态数据功能的核心。下面列出了这些“动态数据”控件：

- **DynamicControl** 该控件能够通过 ASP.NET 动态数据功能显示模板式的数据绑定控件(如第 10 章介绍的控件)所定义的内容
- **DynamicDataManager** 该控件是一个不可见的控件，旨在管理“动态数据”控件的动态行为
- **DynamicEntity** 该控件用于表示 ASP.NET 动态数据中的实体
- **DynamicFilter** 该控件能够显示一个用户界面，用于根据指定的列来筛选表中的记录
- **DynamicHyperLink** 该控件用于显示操纵表中数据的链接(如编辑、删除和插入)
- **DynamicValidator** 该控件能够处理数据模型抛出的异常和显示错误消息

与其他服务器端控件一样，这些控件也适应 ASP.NET 基础架构。下面的练习将演示如何开发一个“动态数据”网站。

➤ 开发一个“动态数据”网站

1. 打开 Visual Studio，使用“ASP.NET Dynamic Data Linq to SQL”模板创建一个 ASP.NET 项目，并将其命名为 DynamicDataLinqToSQLSite(如下图所示)。

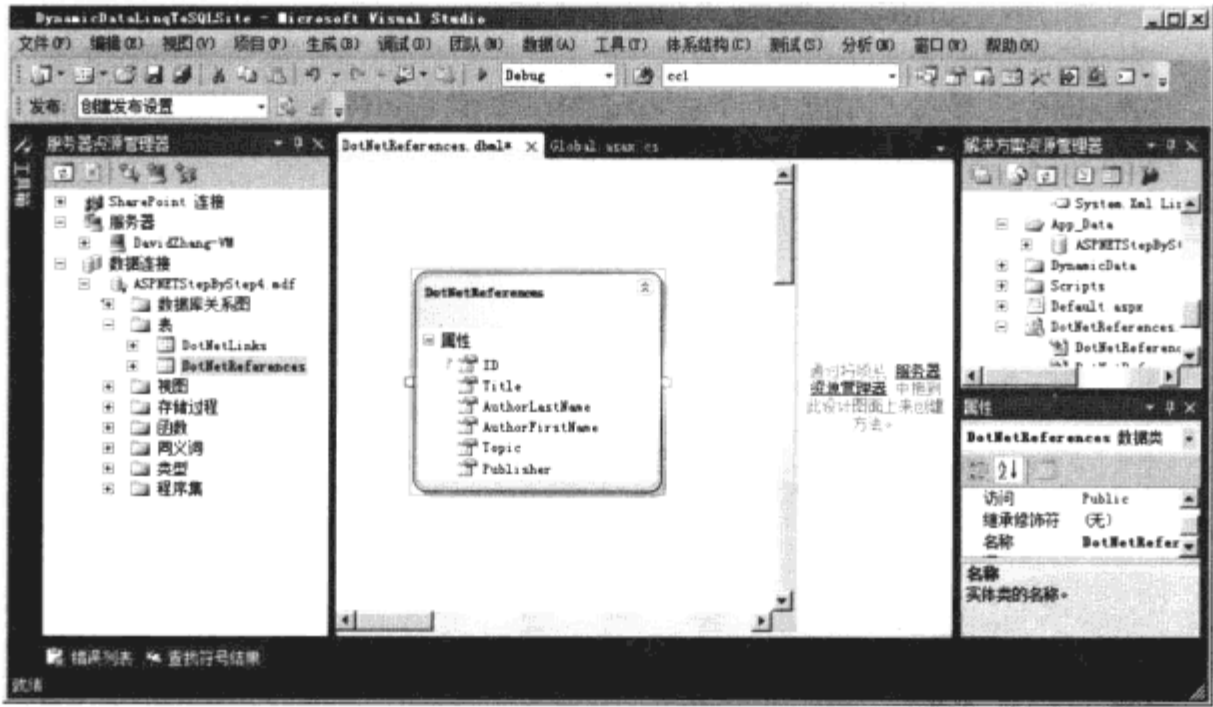


2. Visual Studio 会生成一个支持“动态数据”的网站项目。注意“解决方案资源管理器”中的文件项(本章将介绍其中的几个)。
3. 复制第 10 章项目中的 ASPNETStepByStep4.mdf 文件到当前项目。在“解决方案资源管理器”中，右键单击 App_Data 文件夹，选择“添加”|“现有项”。找到第 10 章的项目，从 App_Data 文件夹选择 ASPNETStepByStep4.mdf。
4. 为应用程序生成数据模型。在“解决方案资源管理器”的项目节点上单击右键，选择“添加”|“新建项”。选择“LINQ to SQL 类”(在“数据”节点下)，将这个 LINQ to SQL 类(Visual Studio 将要生成的类)命名为 DotNetReferences(如下图所示)。单击“确

定”后，Visual Studio 会自动显示“对象关系设计器”界面。



- 5. 打开“服务器资源管理器”，依次展开“数据连接”节点下的 ASPNETStepByStep4 数据库和“表”节点，找到 DotNetReferences 表。从“服务器资源管理器”将 DotNetReferences 表拖入当前的“对象关系设计器”(如下图所示，此图说明 Visual Studio 会创建一个名为 DotNetReference 的类)。



- 6. Visual Studio 会在项目中添加一个名为 DotNetReferences.dbml 的文件(其中包含一些有用的类)。保存这个文件。
- 7. 打开 Global.asax 文件。找到用默认数据模型来注册数据上下文的代码。这段代码是被注释掉的。去掉所在行的注释，让 DefaultModel 注册 DotNetReferencesDataContext(这个类是 Visual Studio 生成的)(如下所示)。此外，将 ScaffoldAllTables 属性设置为 true。

```
public static void RegisterRoutes(RouteCollection routes)
{
    DefaultModel.RegisterContext(typeof(DotNetReferencesDataContext),
```

ASP.NET 4从入门到精通

```
new ContextConfiguration() { ScaffoldAllTables = true });
// more registration code...
}
```

8. 运行这个网站。此时，浏览器会显示下图所示的默认页面。



9. 单击“我的表”标题下的“DotNetReferences”链接。此时，Visual Studio 会查询数据模型中的数据，并根据这些数据填充 GridView。(事实上，正如我们将要在后面的步骤中看到的，Default.aspx 的源代码中只包含一个 GridView。)下图展示了单击“DotNetReferences”链接后显示的页面。



10. 打开 Default.aspx 文件，查看 Visual Studio 生成的代码：

```
<asp:GridView ID="Menu1" runat="server" AutoGenerateColumns="false"
    CssClass="DDGridView" RowStyle-CssClass="td" HeaderStyle-CssClass="th"
    CellPadding="6">
    <Columns>
        <asp:TemplateField HeaderText="表名称" SortExpression="TableName">
            <ItemTemplate>
```



```
<asp:DynamicHyperLink ID="HyperLink1"
    runat="server"><%#Eval("DisplayName") %>
</asp:DynamicHyperLink>
</ItemTemplate>
</asp:TemplateField>
</Columns>
</asp:GridView>
```

11. 这段代码会根据默认的数据模型(目前只有一个表)生成超链接表格。在单击其中的链接后, ASP.NET 会将请求定向到名为 List.aspx 的文件。在“解决方案资源管理器”中, 我们可以在项目的 DynamicData\PageTemplates 节点下找到 List.aspx 文件。ASP.NET 在 URL 中嵌入了这个表的显示名称(DotNetReferences), 并根据 DotNetReferences 表的内容生成一个表格。如果打开 List.aspx 文件, 则会发现, 其中只包含一个 GridView——List.aspx 中的这个 GridView 会根据表的内容生成表格。

20.2 动态数据详解

至此, 在完成之前的示例后, 不免让人对所发生的事感到困惑。毕竟, 如果将 GridView 绑定到数据源, GridView 就会自动反射出数据源的列, 并呈现数据。然而, GridView 所显示的列都是绑定到使用服务器端控件 BoundField 的列。在 GridView 呈现后, 最终用户只能浏览, 而很难对其行为进行定制。ASP.NET “动态数据”功能则改变了这一切, 因为“动态数据”功能基于 DynamicControl 和 DynamicField 控件构建。

ListView 和 FormView 数据控件采用了 DynamicControl。通过 DynamicControl, ASP.NET 能够根据来自数据库的架构信息来呈现 UI。我们可以在 GridView 和 DetailsView 中使用 DynamicField(替代 BoundField)来呈现 UI。DynamicField 在内部依赖于 DynamicControl, 因而可以实现在 GridView 和 DetailsView 上下文中生成同种效果的数据驱动的 UI。

如果应用 DynamicControl(显式地或隐式地), ASP.NET 会直接根据来自数据库的表的结构来呈现 UI(而不依赖于 ASP.NET 提供的默认机制)。这样, 开发者便可以在 GridView、ListView、FormView 和 DetailsView 中轻松使用多种控件。

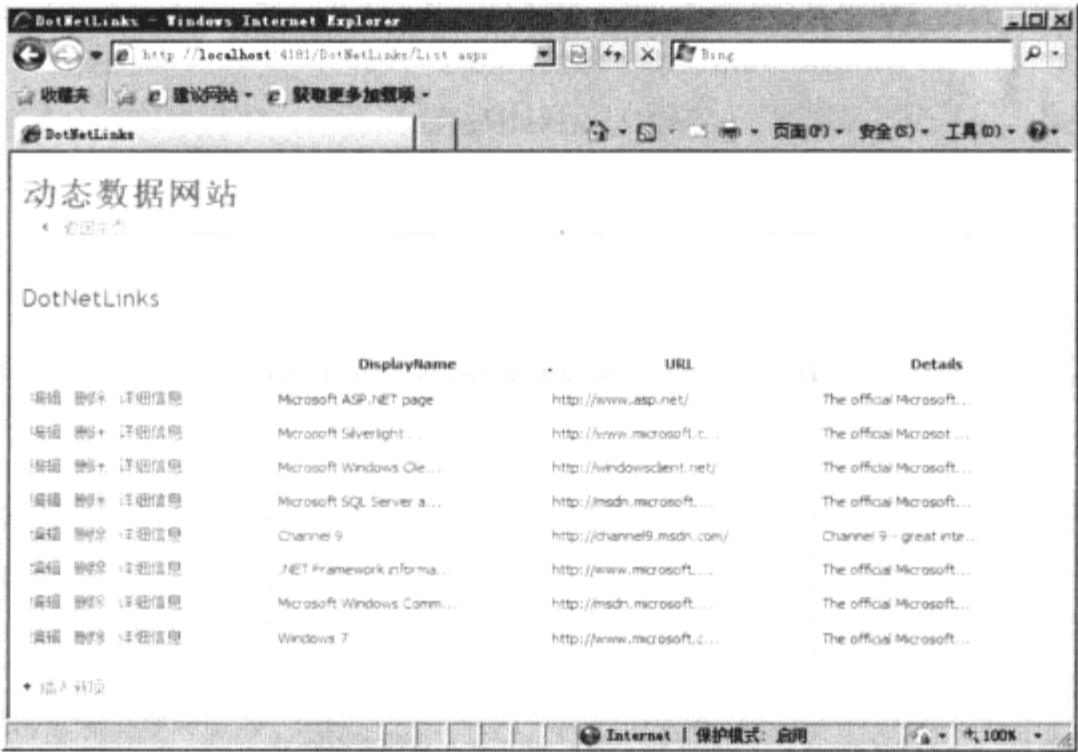
例如, 假设要允许用户通过 DataGrid 输入社保号(美国)。如果不使用“动态数据”, GridView 会通过标准的 TextBox 控件来呈现编辑界面, 而很难添加其他任何控件。现在假设要使用其他控件进行编辑, 如 AJAX 工具箱中的 MaskedTextBoxExtender。为 TextBox 应用 MaskedTextBoxExtender, 可以确保用户的输入遵循正确的格式(由 3 个数字、连字符、2 个数字、另一个连字符和 4 个数字依次组成)。直到现在, 通过 AJAX 来以这种方式扩展 GridView 在技术上还无法做到, 但我们可以使用 DynamicControl 或 DynamicField。Visual Studio 生成的“动态数据”基架(scaffolding)提供了完美替换默认控件的方式。而且, “动态数据”还支持自动数据验证。例如, 使用 ASP.NET 的“动态数据”功能可以避免重复编写验证程序, 因为数据库架构能够为特定列数据的验证提供信息。

下面的练习演示了如何扩展之前的应用程序, 使网站显示多个表并使用基架提供的自动数

据验证。

➤ 扩展应用程序

1. 打开 DotNetReferences.dbml 文件。Visual Studio 会显示“对象关系设计器”。打开“服务器资源管理器”。找到 DotNetLinks 表，将其拖入“对象关系设计器”。重新生成应用程序，这样网站便会显示 DotNetLinks 表(和 DotNetReferences 表)的链接。单击 DotNetLinks 链接。此时，浏览器会显示 DotNetLinks 表的内容。

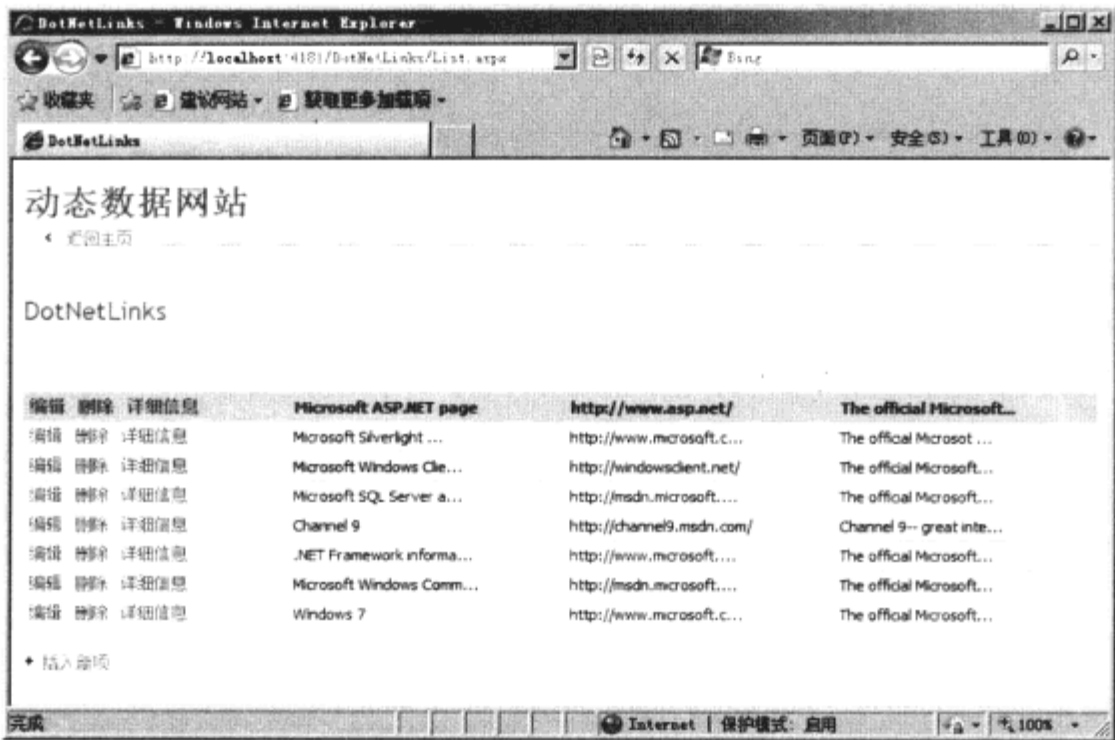


2. 关闭浏览器，在“解决方案资源管理器”中的项目节点下打开 DynamicData\PageTemplates\List.aspx。打开 List.aspx 后，如果 Visual Studio 未处于“设计”视图，则切换到视图下。这样，我们便可以在“设计”视图下看到 GridView 控件(见下图)。



3. 此时，这个页面无异于其他页面。如果要修改 GridView 的格式，只需要单击该控件的“智能标签”——控件右上角的尖括号(>)。例如，可以使 GridView 外观的颜色变得

更丰富。也可以使 GridView 采用交替行的样式(正如第 10 章所使用的)。这里选择“传统型”样式。此时，填充 DotNetLinks 表的 GridView 如下图所示。



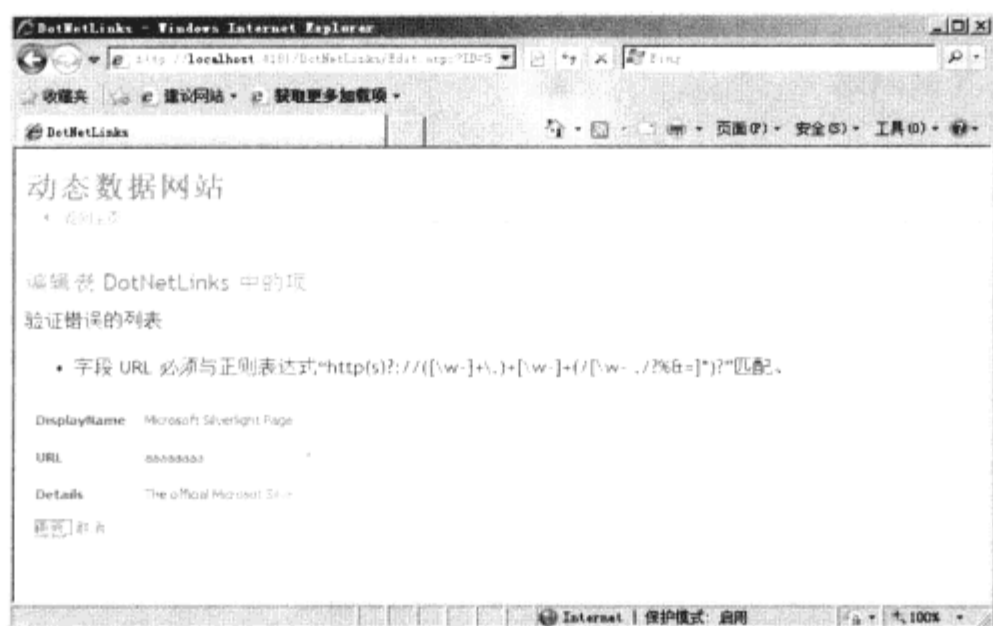
4. 更新 LINQ to SQL 数据模型，使其支持正则表达式验证。Visual Studio 创建了一个名为 DotNetReferences.Designer.cs 的文件。为编辑这个文件，在“解决方案资源管理器”中展开 DotNetReferences.dbml 节点。在 DotNetLink 类中找到 URL 属性，为其添加 RegularExpression 特性 (attribute)。下面的这个正则表达式来自 RegularExpressionValidator 控件的 ValidationExpression 对话框：

```
[global::System.Data.Linq.Mapping.ColumnAttribute(Storage = "_URL",
    DbType = "NVarChar(100) NOT NULL", CanBeNull = false)]
[System.ComponentModel.DataAnnotations.RegularExpression
    ("http(s)?://([\\w-]+\\.)+[\\w-]+(/[\\w- ./?%&=]*)?")]
```

```
public string URL
{
    get
    {
        return this._URL;
    }
    set
    {
        if ((this._URL != value))
        {
            this.OnURLChanging(value);
            this.SendPropertyChanging();
            this._URL = value;
            this.SendPropertyChanged("URL");
            this.OnURLChanged();
        }
    }
}
```

5. 运行网站。导航到 DotNetLinks 页面，单击某条记录的“编辑”链接。此时，浏览器

会显示 DotNetLinks 的编辑页面。在文本框中键入一个无效的 URL，并单击“更新”。正如第 6 章所介绍的，内建的正则表达式验证程序会被激活(如下图所示)。之所以有这种变化，是因为在属性值不匹配通过特性定义的正则表达式时，数据模型中的 URL 属性会抛出异常。



6. 如果要查看内建 `RegularExpressionValidator` 的设置，可以在“解决方案资源管理器”中打开 `DynamicData\\FieldTemplates\\Text_Edit.ascx` 文件。该文件包含以下代码：

```
<%@Control Language="C#" CodeBehind="Text_Edit.ascx.cs"
    Inherits="DynamicDataLinqToSQLSite.Text_EditField"%>

<asp:TextBox ID="TextBox1" runat="server"
    Text='<%# FieldValueEditString%>' CssClass="DDTextBox"></asp:TextBox>

<asp:RequiredFieldValidator runat="server"
    ID="RequiredFieldValidator1" CssClass="DDControl DDValidator"
    ControlToValidate="TextBox1" Display="Dynamic" Enabled="false"/>
<asp:RegularExpressionValidator runat="server"
    ID="RegularExpressionValidator1" CssClass="DDControl DDValidator"
    ControlToValidate="TextBox1" Display="Dynamic" Enabled="false"/>

<asp:DynamicValidator runat="server"
    ID="DynamicValidator1" CssClass="DDControl DDValidator"
    ControlToValidate="TextBox1" Display="Dynamic"/>
```

没错！页面的编辑功能就是通过用户控件实现的！

不难发现，ASP.NET 动态数据功能并没有隐藏什么或让人不可捉摸。事实上，如果仔细阅读 Visual Studio 生成的代码，则会发现许多本书之前所介绍的 ASP.NET 惯例和功能。“动态数据”的强大之处在于，开发者可以对网站进行定制，使其由数据模型驱动，而不必花费大量时间来创建匹配数据模型的 UI。如果数据模型在未来发生变化，“动态数据”功能的优势便会凸显出来。

20.3 快速参考

目 标	操 作
创建支持“动态数据”的网站	使用 Visual Studio 的“ASP.NET Dynamic Data LINQ to SQL”或“ASP.NET Dynamic Data 实体 Web 应用程序”项目模板
创建 LINQ to SQL 数据模型	使用 Visual Studio 的“LINQ to SQL 类”文件模板来创建 DBML 和类文件。将数据模型中的表拖到“对象关系设计器”界面上
使 ASP.NET 路由功能根据基架来自动处理请求	在调用 DefaultModel.RegisterContext 时传入正确的数据上下文类型(该类型是 Visual Studio 生成的)
为数据模型添加验证	为要验证的属性添加验证特性(如 RegularExpression 或 Required 特性)
修改生成的页面的观感	在“解决方案资源管理器”中展开 DynamicData\PageTemplates 节点。修改相应的.aspx 文件(如 List.aspx、ListDetails.aspx、Details.aspx、Edit.aspx 和 Insert.aspx)

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 21 章

ASP.NET 与 WPF 内容

学习目标

- 理解 Windows Presentation Foundation(WPF)相比传统 Windows 用户界面的优势
- 创建基于 XAML 的浏览器应用程序(XBAP)
- 为 ASP.NET 网站添加基于 WPF 的内容

前 20 章一直都在讲解 ASP.NET 如何通过将 HTML 的呈现交给 ASP.NET Control 类及其子对象来简化 Web 开发。此外，ASP.NET 管线也隐藏了许多 Web 请求的细节，以便开发者能够将注意力放在开发上。下面几章将介绍另一种为最终用户生成内容的方式，包括 ASP.NET 对 AJAX 的支持、模型-视图-控制器(MVC)模式，以及 Microsoft Silverlight 的工作方式。本章将首先讨论如何为浏览器呈现基于“可扩展应用程序标记语言”(XAML)的内容。

21.1 通过降低往返次数来改进界面性能

纵观 Web 的历史，改善最终用户体验的主要方式是降低与服务器的交互次数。很长时间以来，为达成这一目标只能使用客户端脚本。这样，应用程序的一部分代码可以在客户端浏览器执行，而这通常要比执行整个往返周期快得多。

第 23 章介绍了 AJAX，这是一种改善 Web 用户界面(UI)的主流技术。AJAX 能够为 Web 用户界面提供许多之前桌面应用程序所特有的元素。例如，AJAX 中的 AutoComplete 扩展程序允许用户在向 TextBox 输入文本的过程中从提示的选项中进行选择，这些选项由 Web 服务动态生成。使用 ModalPopupExtender，用户可以在一个面板中定制内容，该面板在运行时的行为类似 Windows 中标准的模式对话框。

然而，编写脚本不是将功能推入浏览器的唯一方式。AJAX 基本上仍依赖于 HTML。虽然 HTML 包含许多在浏览器中呈现标准用户界面元素的标记，但 WPF 内容更为丰富。WPF 提供了一种为网站添加富用户界面的新方式，彻底改变了标准的 Web(和 Windows)用户界面编程方式。在本章中，我们将学习 WPF 的工作方式，以及它与 Internet 和浏览器应用程序的关系。在学习 Silverlight(一种与 WPF 类似的技术)时，我们将回顾本章的部分内容。下面让我们先来认识一下 WPF。

21.2 WPF 是什么

基于 Windows 用户界面的编程所使用的架构二三十年以来一直没有本质变化。从 20 世纪 80 年代早期到今天，所有应用程序都采用一种机制：主程序运行消息循环，从消息队列中提取 Windows 消息，并将消息分发给窗口句柄。每个窗口负责呈现自身的内容——这包括所有窗口，从应用程序的顶层窗口到窗口中最基本的控件。

如今，几乎所有基于 Windows 的应用程序在底层仍使用 Win32 应用程序编程接口(API)。传统的 Win32 API 一直都在使用。然而，这种设计逐渐显露其陈旧。由于每个窗口和控件负责使用 Win32 “图形设备接口”(Graphics Device Interface, GDI, “Windows 窗体”应用程序采用的是 GDI+)来呈现自身，所以基本的用户界面限制都已内建到 Windows 操作系统的设计中。然而，这涉及许多基本的绘制和文本呈现工作。例如，变换、透明、视频播放集成等特殊效果很难用当前的 Windows 图形接口来实现。Windows 还支持一种功能更为丰富的图形接口——DirectX。然而，它并不适合大多数 Windows 应用程序，而更多适用于开发游戏。

传统 Windows API 的限制促使微软开发一种新的编程接口——Windows Presentation Foundation(WPF)。WPF 能够为基于 Windows 的应用程序(包括稍后要介绍的 Web 内容)添加特殊效果。WPF 库由许多类组成，其整体行为与“Windows 窗体”对应类的行为一致(至少在表面上是这样，但两者的底层工作机制完全不同)。

WPF 在用户界面的开发方面提供了非常丰富的编程接口。下面列出了 WPF 提供的几种功能(这只是个概要，并不详尽)：

- 各种用户界面元素。相比使用 Win32 和继承机制，这些元素更容易定制
- 路径、形状和二维几何图形
- 允许用户界面元素以一致的方式进行变换(拉伸、平移、旋转和扭曲)
- 管理每个元素的不透明度
- 内建的布局面板
- 画笔——能够显示域的图像、视频和画笔
- 动画

WPF 应用程序通过布局面板(layout panel)来安排 UI 元素。WPF 不依赖于绝对定位(如 Win32 应用程序)，也不依赖于流布局(如 ASP.NET 页面)，而是提供了很多布局选项，其中包括以下几种：

- **Grid** 允许在表格中放置元素
- **StackPanel** 允许元素纵向或横向叠加

- **Canvas** 允许元素采用绝对定位方式
- **DockPanel** 允许元素定位在容器的一边
- **WrapPanel** 允许元素依次排列，并适应宿主的大小

后面的示例采用的是 **Canvas**。

创建一般 WPF 应用程序和创建 ASP.NET 应用程序的方式相同，都可以从空白文件开始写起。独立的 WPF 应用程序包括一个运行消息循环的主应用程序对象和至少一个窗口，而基于浏览器的 WPF 应用程序由页面构成。WPF 应用程序组件通常由标记文件组成(与 ASP.NET 页面类似)。WPF 的布局通过 XAML 定义。

XAML 文件用于描述 WPF 布局的逻辑树——WPF 用户界面元素的集合。WPF 应用程序由“公共语言运行库”(CLR)类构成。XAML 文件能够引导运行库生成可视元素的逻辑树。对于基于浏览器的应用程序，逻辑树在浏览器中呈现。下面是一小段 XAML，包含一个文本为“Hello World”的按钮：

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:sys="clr-namespace:System;assembly=mscorlib"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
  <Button Height="100" Width="100">Hello World</Button>
</Page>
```

这段代码比较简单，只是为了说明 XAML 描述的 WPF 页面所采用的基础结构。在运行后，Visual Studio 会建立一个浏览器会话并显示带有“Hello World”字符串的按钮(需要安装 XAML 插件)。实际的应用程序不会这么简单。我们可以通过 WPF 中的布局面板实现各种布局方式(稍后会做演示)。

21.2.1 WPF 与 Web 的关系

所有这些对 Web 应用程序有何意义？Windows Internet Explorer 和其他运行在 Windows 操作系统上的浏览器都基于传统的 Windows 架构。浏览器负责通过 Windows 的图形接口(“图形设备接口”)来显示图形界面。因此，浏览器(和一般的 HTML)与传统 Windows 程序面临着同样的窘境。

Web 编程是将 HTTP 请求提交给服务器，服务器处理请求后再将响应发送回客户端。因此，与用户界面有关的响应受 HTML 的制约。Web 是动态的，而 HTML 是一种文档技术。

有支持丰富的标记又受 HTML 浏览器解释的语言吗？答案是肯定的。XAML 可以在 Web 应用程序上下文中使用。

还记得前文给出的代码吗？如果这段代码以 ASCII 方式保存在名为 HelloWorld.xaml 的文本文件中，那么在“Windows 资源管理器”中双击它，Internet Explorer 便会加载并解析其中

的 XAML 内容。图 21.1 展示了那段 XAML 文件加载到 Internet Explorer 浏览器后的效果。在“Windows 资源管理器”中双击这种文件，便可以运行其定义的应用程序。

在网站中直接添加 WPF 风格的内容有三种方式：(1)通过松散 XAML 文件定义内容；(2)创建基于浏览器的应用程序(XBAP)；(3)使用 Silverlight。(有关 Silverlight 的内容将在第 24 章中介绍。)



图 21.1 XAML 定义的按钮

21.2.2 松散 XAML 文件

正如刚刚介绍的，如果在网站中添加良构的 XAML 文件，并通过 Web 服务器将其暴露出来，那么任何支持 XAML 插件的浏览器(如 Internet Explorer)都能够接收并呈现它。这种技术非常适合呈现半动态(semi dynamic)的内容——即只用 XAML 文件描述的内容。

WPF 编程模型将 XAML 布局定义和代码模块结合在了一起，这种方式与 ASP.NET 所采用的机制非常类似。来自用户界面元素的事件由代码隐藏文件中的代码进行处理。松散的 XAML 文件不包含事件处理程序和其他代码。

然而，只有用纯粹的 XAML 来组织用户界面元素并能够生成动画，才能说 WPF 元素是动态的。因此说使用纯粹的 XAML 描述的 WPF 内容是半动态的。我们可以添加 XAML 定义的纯交互式元素，但这还不够。例如，虽然 XAML 能够通过用户界面元素呈现图像名称列表，并且允许用户通过滚动条来查看每个元素，但我们不能为这些控件实现事件处理程序。为添加处理程序，需要以 XBAP 应用程序的形式来创建和部署 WPF 应用程序。

21.2.3 XBAP 应用程序

XBAP 提供了另一种通过 Web 部署 WPF 内容的方式，但这要比松散 XAML 文件稍微复杂一点。除了定义布局外，XBAP 还支持为每个页面添加可执行的代码。部署了在 Web 上运

行的 WPF 应用程序后，客户端便可以下载 WPF 可视布局定义和相应的代码到客户端计算机上。XBAP 中的事件会由客户端处理。

从表面上看，以 XBAP 方式部署的应用程序与基于 Windows 的桌面应用程序很类似(但前者在很大程度上地降低了权限和安全性设置)。例如，这种应用程序能够处理鼠标的单击事件，并完全在客户端完成对控件事件的响应。

XBAP 与 ASP.NET 并不直接关联，但 XBAP 内容可以包含在 ASP.NET 页面中，与组织松散 XAML 内容的方式相同。也就是说，我们可以重定向到 XBAP 文件或在 HTML 元素 <iframe> 中嵌入 XBAP 文件。

Microsoft Visual Studio 包含一个向导，能够生成承载 WPF 内容的 XBAP。此外，WPF 内容中包含的用户界面元素能够像桌面应用程序一样响应事件和消息。当浏览器导航到 XBAP 文件后(最终要通过“Internet 信息服务”部署)，便能够提供与桌面应用程序类似的用户体验(用户界面和响应速度)，尽管程序是在浏览器中运行的。下面的练习演示了 XBAP 内容的创建。

➤ 创建 XBAP

- 1. 打开 Visual Studio，单击“文件”|“新建项目”。打开“Windows”应用程序模板，选择“WPF 浏览器应用程序”。将这个项目命名为 XBAPORama(如下图所示)。



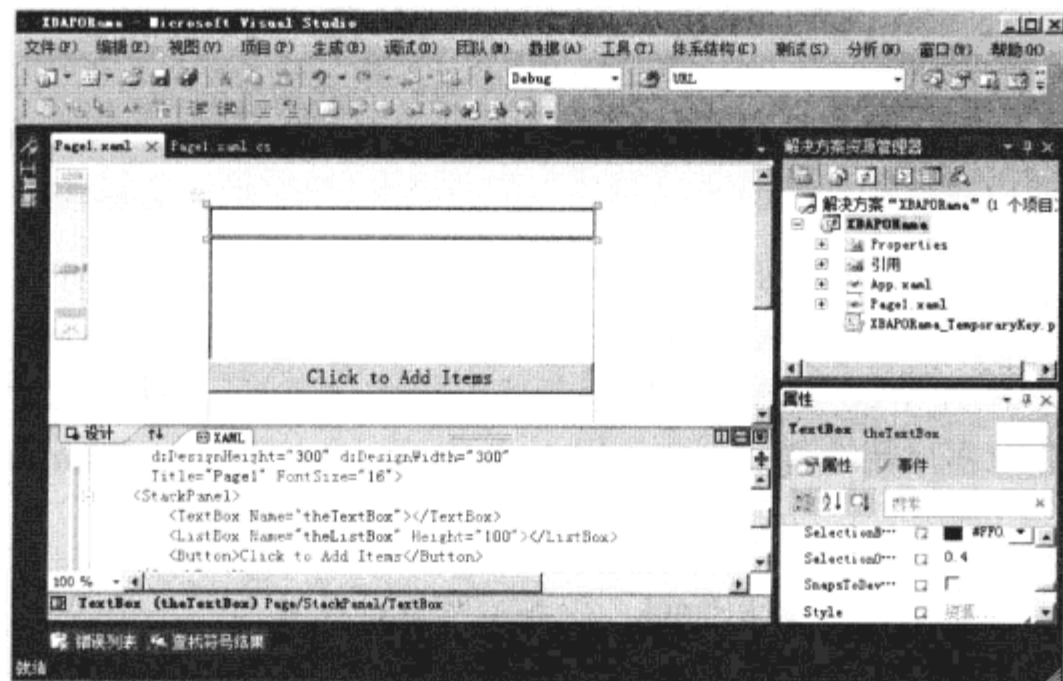
- 2. Visual Studio 会创建一个 XBAP 程序，其中包含一个页面 XAML 和一个应用程序 XAML。相关文件为 Page1.xaml/Page1.xaml.cs 和 App.xaml/App.xaml.cs。这种文件组织方式与 ASP.NET Web 窗体应用程序所采用的非常类似——标记文件包含 UI 元素，代码文件包含在客户端运行的功能。Visual Studio 会打开 Page1.xaml 文件，其中包含一个 Grid 布局面板。
- 3. 将 Grid 布局面板替换为 StackPanel。在 StackPanel 中添加控件相对简单，不必考虑行和列的定义。


```
<Page x:Class="XBAPORama.Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="300"
    Title="Page1">
    <StackPanel>
    </StackPanel>
</Page>
```

4. 稍事修改 XAML 文件。将 Page 的 FontSize 属性设置为 16。在 StackPanel 中添加以下控件: TextBox、ListBox 和 Button。WPF 支持在标记文件(XAML 文件)中设置控件的名称,而控件在代码隐藏类中便表现为可编程元素,这种机制与 ASP.NET 所采用的非常类似。将 TextBox 和 ListBox 的 Name 属性分别设置为 theTextBox 和 theListBox。这样,我们稍后在代码文件中便可以引用它们。最后,将 ListBox 的 Height 属性设置为 100,以便其在内容为空的情况下也能够被显示出来。相关代码如下所示:

```
<Page x:Class="XBAPORama.Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="300"
    Title="Page1" FontSize="16">
    <StackPanel>
        <TextBox Name="theTextBox"></TextBox>
        <ListBox Name="theListBox" Height="100"></ListBox>
        <Button>Click to Add Items</Button>
    </StackPanel>
</Page>
```

此时的“设计器”会像下图一样显示 StackPanel 中的控件。



5. 双击刚刚添加的按钮以添加处理程序。Visual Studio 会在当前页面的代码文件中为该按钮创建一个处理程序。由于尚未对这个按钮进行命名，因而 Visual Studio 会使用默认名称——Button_Click。这个方法与 ASP.NET 中按钮单击事件的处理程序类似，只不过这里将一般的 EventArgs 类型替换为了 RoutedEventArgs 类型。
6. 在这个处理程序中，将 TextBox 的内容添加到 ListBox。这种编程体验与“Web 窗体”一样——代码模型非常类似。

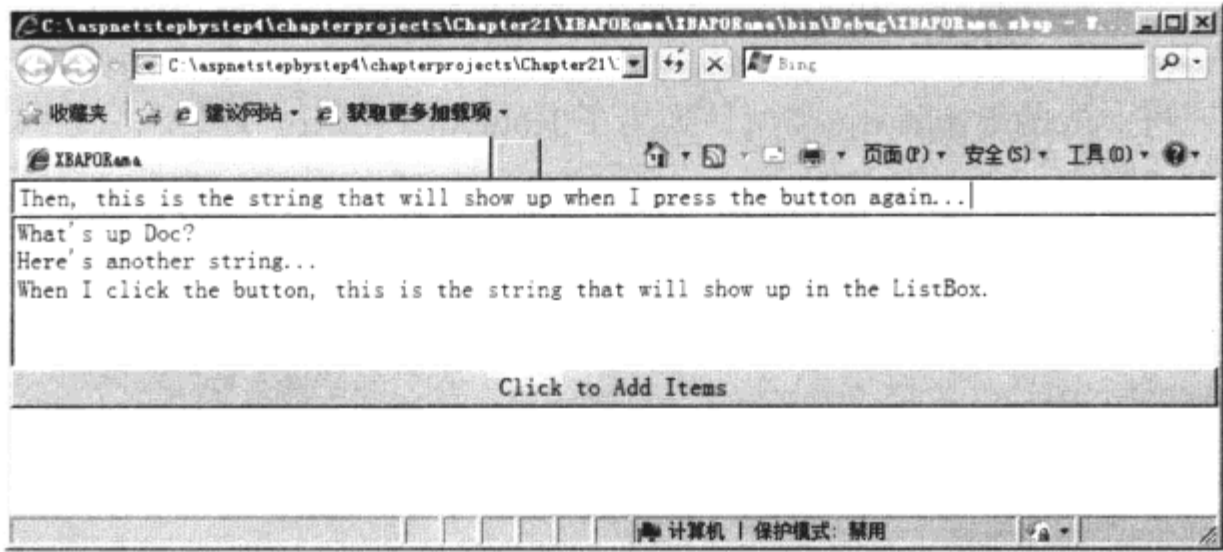
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace XBAPORama
{
    ///<summary>
    /// Page1.xaml 的交互逻辑
    ///</summary>
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }
        private void Button_Click(object sender, RoutedEventArgs e)
        {
            this.theListBox.Items.Add(this.theTextBox.Text);
        }
    }
}
```

7. 在 Visual Studio 中按 Ctrl+F5 组合键，通过浏览器运行这个应用程序。在向 TextBox 输入文本并单击按钮后，客户端代码会将 TextBox 的内容添加到 ListBox 中，如下图所示(注意 URL 末尾的.xbap 扩展名)。

虽然这个示例并没有严格在 ASP.NET 环境中运行，但它确实提供了另一种生成内容的方式。在编译应用程序后，Visual Studio 会创建包括 XBAPORama.xbap 和 XBAPORama.exe 在内的几个文件。为使这些内容成为 ASP.NET 网站的一部分，可以在 ASP.NET 应用程序的某个文件夹中添加编译后的 XBAP、EXE 和程序清单文件。(稍

后会有一个相关练习。)



21.3 WPF 内容与 Web 应用程序

ASP.NET 应用程序可以用来提供 WPF 内容，这与提供其他内容的方式相同。我们可以在 Web 应用程序中添加松散 XAML 文件，也可以在 HTML 元素<iframe>中包含某些指定的 WPF 内容。下面的练习将演示如何通过 ASP.NET 应用程序提供 WPF 内容。

➤ 为网站添加 XAML 内容

- 1. 在 Visual Studio 中创建一个“ASP.NET 空 Web 应用程序”，将其命名为 XAMLORama。
- 2. 在这个项目中添加一个文本文件。在 Visual Studio 中右键单击项目节点，单击“添加”|“新建项”。从模板中选择文本文件类型。
- 3. 重命名这个文件，使其具有.xaml 扩展名。这个文件将包含一个纸飞机的图形，不妨将这个文件命名为 PaperAirplane.xaml。Visual Studio 的 XAML 设计器会立即显示一个错误，因为该文件是空的。没关系，下面将为其添加内容。
- 4. 在文件中添加一些 XAML 内容。首先定义顶层布局节点。添加以下 XAML 命名空间，并将窗口宽度设置为 750：

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">

</Page>
```

所有 WPF 布局都以顶层节点为基础。在这个练习中，顶层节点是 Page(页面)，因此它将会显示在客户端浏览器中。

- 5. 在页面中添加一个 Grid(网格)，添加两个行定义和两个列定义：

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">
  <Grid>
    <Grid.RowDefinitions>
```

```

        <RowDefinition/>
        <RowDefinition Height="100"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition Width="25"/>
    </Grid.ColumnDefinitions>
</Grid>
</Page>

```

6. 在网格中添加 WPF 元素。在 Grid 的左上角添加一个 Canvas，将 Background 属性设置为 SkyBlue。添加两个 Slider(滚动条)控件。第一个 Slider 用于控制飞机的 x 轴，将这个 Slider 命名为 sliderX。将 sliderX 置于第一行，通过 ColumnSpan 属性使这个滚动条跨越两列。将 sliderX 的最大值设置为 500。第二个 Slider 用于控制飞机的方向，因而将其命名为 sliderRotate，将其最大值设置为 360。使 sliderRotate 横向呈现，将其置于 Grid 的第一列，并通过 RowSpan 属性使它跨越两行。

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">
    <Grid>
        <!--这里是行和列的定义... -->
        <Canvas Background="SkyBlue" Grid.Row="0"
                Grid.Column="0">
        </Canvas>
        <Slider x:Name="sliderRotate" Orientation="Vertical"
                Grid.Row="0"
                Minimum="0" Maximum="360"
                Grid.Column="1"></Slider>
        <Slider x:Name="sliderX" Maximum="500"
                Grid.Column="0" Grid.Row="1"
                Grid.ColumnSpan="2"></Slider>
    </Grid>
</Page>

```

7. 添加飞机图形，并通过 XAML 数据绑定将滚动条与这个图形关联。通过 WPF 元素 Path 添加一个飞机图案。Path 能够使用指定的画笔绘制一系列线段。将 Stroke 属性设置为 Black，将 StrokeThickness 属性设置为 3。Path 的数据需要将以下点连接起来。将光标移动到(0, 0)，绘制到(250, 50)的线段，再到(200, 75)，再到(0, 0)。然后，将光标移动到(200, 75)，绘制到(190, 155)的线段，再到(180, 85)，再到(0, 0)。接着，将光标移动到(180, 85)，绘制到(140, 105)，再绘制到(0, 0)。最后，将光标移动到(190, 115)，绘制到(158, 93)。将 Path 与 Canvas 上边缘(Top)的距离设置为 200。将 Path 与 Canvas 左边缘(Left)的距离值绑定到 sliderX 的 Value 属性。最后，为 Path 添加一个 RenderTransform 和 RotateTransform。将 RotateTransform 的 Angle 属性绑定到 sliderRotate 的 Value 属性。将 Path 的 RenderTransformOrigin 属性设置为 .5, .5。下面是 Path 的代码：

```

<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="750">

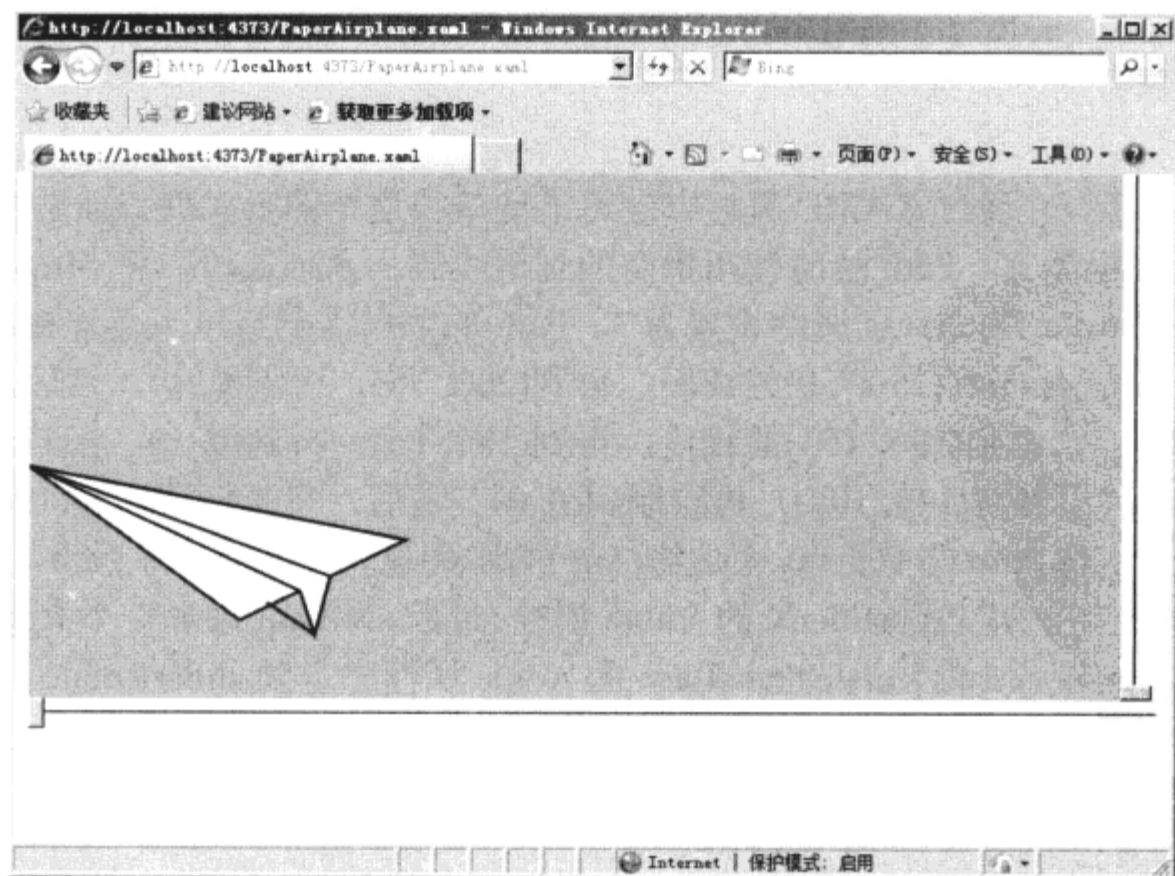
```



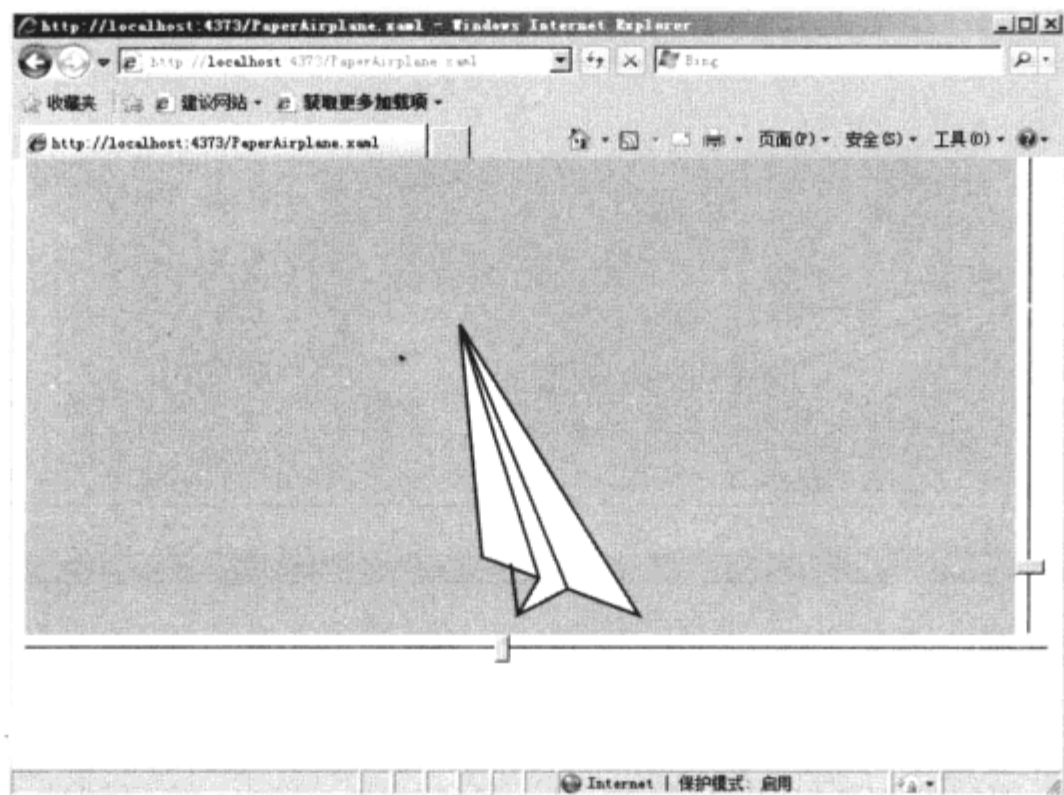
```
<Grid>
  <!--这里是行和列的定义... -->
  <Canvas Background="SkyBlue" Grid.Row="0"
    Grid.Column="0">
    <Path Stroke="Black" StrokeThickness="2" Fill="White"
      Data="M0,0 L250,50 L200,75 L0,0 M200,75 L190,115 L180,85
        L0,0 M180,85 L140,105 L0,0 M190,115 L158,93"
      RenderTransformOrigin=".5, .5"
      Canvas.Top="200"
      Canvas.Left="{Binding ElementName=sliderX,Path=Value}" >
    <Path.RenderTransform>
      <RotateTransform Angle=
        "{Binding ElementName=sliderRotate,Path=Value}"/>
    </Path.RenderTransform>
  </Path>
  <!--这里是滚动条... -->
</Canvas>
</Grid>
</Page>
```

在网格中设置 Canvas、Path 和 Slider 后，Visual Studio 会将它们显示出来。

8. 在“解决方案资源管理器”中的“引用”节点上右击，选择“添加引用”，在“添加引用”对话框的“.NET”节点中查找 WindowsBase、PresentationCore 和 PresentationFramework 这三个程序集，分别将其添加到项目中。运行这个页面。因为 Visual Studio 不支持直接运行松散 XAML 文件，所以我们需要从其他页面导航过来。为应用程序添加一个页面，将其命名为 Default.aspx。为 Default.aspx 页面添加一个 Hyperlink 控件，将其 NavigationUrl 属性设置为 PaperAirplane.xaml。打开这个默认页面并单击其中的超链接。此时，XAML 文件会加载到浏览器(如下图所示)。



9. 尝试滑动滚动条。由于垂直滚动条控制飞机的角度，向上滑动会使飞机顺时针旋转。水平滚动条与 Path 的 Canvas.Left 属性绑定，滑动这个滚动条会使飞机沿 x 轴移动。滑动这两个滚动条后的效果如下图所示。



10. 将 WPF 内容与 HTML 结合。在“解决方案资源管理器”中的“XAMLORama”节点上右击，添加一个新页面，将其命名为 PaperAirplane.aspx。在 Visual Studio 提供的 <div> 标签中添加一个 <iframe> 标签。将这个 <iframe> 的 height 设置为 500，将 width 设置为 750。最后，将 <iframe> 的 src 设置为 PaperAirplane.xaml。

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="PaperAirplane.aspx.cs"
    Inherits="XAMLORama.PaperAirplane"%>

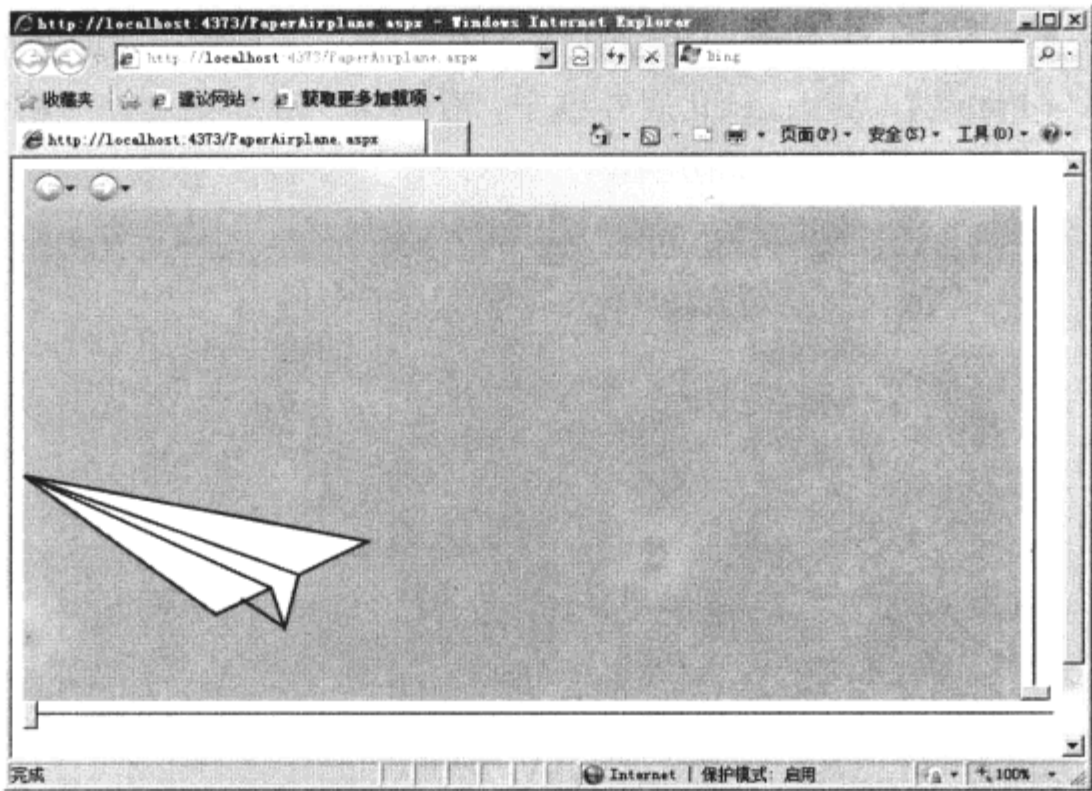
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>

            <iframe height="500"
                width="750"
                src="PaperAirplane.xaml"></iframe><br/>

        </div>
    </form>
</body>
</html>
```

11. 运行这个页面，结果如下图所示。PaperAirplane.xaml 的内容会包含在页面的框架(frame) 中。XAML 内容的特性与其直接在浏览器中运行的特性相同。



由于这是呈现在一般的 ASP.NET 页面中，因而可以在包含 WPF 内容的同时包含 ASP.NET 服务器控件。

- 12. 将前一个练习创建的 XBAP 内容复制到当前网站中。首先，在“解决方案资源管理器”的项目节点下添加一个文件夹，将其命名为 XBAPContent。在这个文件夹上右击，选择“添加”|“现有项”。导航到前一个练习创建的项目(如 C:\aspnetstepbystep4\chapterprojects\Chapter21\XBAPORama\XBAPORama\bin\Debug)。添加 XBAPORama.xbap、XBAPORama.exe 和 XBAPORama.exe.manifest 到当前的 XAMLORama 项目。
- 13. 在 Default.aspx 页面中添加一个新链接控件。使其 NavigationUrl 属性指向 XBAPContent 文件夹下的 XBAPORama.xbap 文件。运行这个应用程序，单击指向 XBAP 内容的链接。此时，浏览器会从 Web 服务器下载 XBAPORama.xbap 的内容，并显示其中的 XBAP 内容(页面会显示一个状态栏，如下图所示)。



下图是该 XBAP 内容在 ASP.NET 网站中的运行效果。



这个示例说明了 HTML 中可以包含基于 XAML 的内容，也说明了 ASP.NET 网站可以包含 XBAP 内容。虽然这些技术与 ASP.NET 管线的关系不大，但基于 XBAP 和基于 XAML 的 WPF 内容仍能够在许多情况下发挥作用。WPF 和 XAML 为最终用户显示内容提供了一种全新的方式。这是一项新技术，还有许多用法值得探索和学习，但有关 WPF 的具体内容不在本书的讨论范围之内。

21.4 关于 Silverlight

作为 Web 开发者，或多说少会听说过 Silverlight。至今，使用 Macromedia Flash 仍然是创建动态 Web 内容的有效方式。Flash 是在 Web 上呈现动态内容(即动画)的插件。然而，WPF 及其动态内容特性提供了一种支持在浏览器呈现的标记技术，这在底层功能上要优于 Flash。虽然还有其他动态内容技术，但这些技术在对开发者的支持上有诸多缺陷。Silverlight 改变了这一切。

Silverlight 是一种平台无关的 WPF 呈现引擎。如果不使用 Silverlight，那么在浏览器中呈现 WPF 内容的唯一方式是安装 XAML 插件。在 Microsoft 环境下，Silverlight 由 ActiveX 控件包装。Apple Safari 浏览器也支持 Silverlight。Visual Studio 2010 全面支持 Silverlight 应用程序。第 24 章将专门介绍 Silverlight。

21.5 快速参考

目 标	操 作
在网站中添加 XAML 文件	在 Visual Studio “解决方案资源管理器”的项目节点上右击。选择“添加” “新建项”。从模板列表中选择“文本文件”。重命名这个文件，确保其扩展名为.xaml
在 XAML 文件中声明 Page	在文件的顶部，添加开标签<Page>和闭标签</Page>。在 XAML 中使用 WPF 需要添加标准的 WPF 命名空间 “http://schemas.microsoft.com/winfx/2006/xaml/presentation” 和关键字命名空间 “http://schemas.microsoft.com/winfx/2006/xaml” (它一般映射到前缀“x”)
在 Page 中添加 Canvas	声明开标签<Canvas>和闭标签</Canvas>。将希望在画布中显示的对象置于这两个标签之间
为 Canvas 添加内容	将希望在画布中显示的对象置于开标签<Canvas>和闭标签</Canvas>之间。通过 Canvas.Top 和 Canvas.Right 属性设置对象的位置
在 Page 中添加 Grid	声明开标签<Grid>和闭标签</Grid>。通过 Grid 的 RowDefinitions 和 ColumnDefinitions 属性来定义行和列
为 Grid 添加内容	将希望在网格中显示的对象置于开标签<Grid>和闭标签</Grid>之间。使用 Grid.Row 和 Grid.Column 属性来设置对象在网格中的位置
创建基于 XAML 的浏览器应用程序	在 Visual Studio 的主菜单中选择“文件” “新建” “项目”。在“Windows”应用程序模板中，选择“WPF 浏览器应用程序”。Visual Studio 会创建一个 XBAP 应用程序，其中包含一个简单的页面。在这个页面中添加 WPF 控件和处理程序。如果要在 ASP.NET 网站中运行 XBAP 内容，请将 XBAP、EXE 和程序清单文件添加到目标 ASP.NET Web 项目中

第 22 章

ASP.NET MVC 框架

学习目标

- 理解“模型-视图-控制器”(MVC)幕后的软件模式
- 理解 ASP.NET 中 MVC 模式的实现方式
- 创建一个可用的 MVC 网站

虽然 ASP.NET 中的一些功能多年都没变化，但一直都有新特性加入。在这些新特性中，采用“模型-视图-控制器”(MVC)软件开发模式的 ASP.NET 应用程序备受关注。

ASP.NET MVC 框架是一种替代“Web 窗体”的环境。“Web 窗体”针对生成网页提供了结构化的、基于控件的框架。使用“Web 窗体”可以像开发桌面应用程序或富客户端应用程序一样进行开发，只不过这种应用程序的 UI 基于 HTML 构建，并通过无连接协议传输。

“Web 窗体”页面的基础是服务器端控件——每个控件负责输出流中一部分 HTML 的呈现(输出流最终会发送到客户端)。“Web 窗体”通过视图状态和事件模型为开发者提供了诸多便利。Visual Studio 中的“设计器”完全支持“Web 窗体”。开发者甚至可以在不需要处理任何 HTML 的情况下开发整个页面。

相对而言，MVC 框架“更接近底层”(closer to the metal)。也就是说，开发者往往要处理原始的 HTML。然而，为了避免开发者使用特殊的框架来完成主要的应用程序开发任务——处理数据和 UI，MVC 框架提供了一种工业级的软件模式来协调这两方面。MVC 模型将应用程序数据、数据的呈现和请求的处理这三方面内容严格界定。每种任务由专门的软件组件处理，而这些组件合在一起便可以完成 HTTP 请求的处理。

本章将介绍如何使用 ASP.NET MVC 框架来创建网页。

22.1 “模型-视图-控制器”(MVC)架构

现代软件往往比较复杂。读者们可能有所感触，即便使用像 ASP.NET 这样能够隐藏许多底层机制的框架，开发 Web 应用程序仍费时费力。将不同任务进行拆分，在多数情况下会降低问题的复杂度。例如，ASP.NET 应用程序一般涉及操作数据源(一般是数据库)，确保应用程序呈现的 HTML 与应用程序状态同步，以及管理传入和传出的消息。

ASP.NET 本身已包含处理这些任务的组件。ADO.NET、LINQ 和实体框架能够很好地与数据库交互。ASP.NET 数据绑定和数据绑定控件能够使客户端的数据视图显示当前数据。而

结合 ASP.NET 管线，ASP.NET 控件架构则隐藏了许多管理传入和传出消息的细节。虽然 ASP.NET 的这些功能在很大程度上简化了 Web 开发，但还有其他 Web 开发方法。MVC 提供了另一种处理 HTTP 请求的途径。

MVC 模式可以应用于 Web 开发。MVC 将典型 Web 应用程序的开发分为 3 个方面(管理数据、使应用程序的可视部分与内部状态同步，以及管理消息传递)，并通过 3 个专门的组件来分别进行管理——模型(model)、视图(view)和控制器(controller)。“模型”负责处理数据访问和管理应用程序状态，“视图”负责处理应用程序的可视表示，而“控制器”负责管理消息。

MVC 模型组件负责维护应用程序的状态，有时包含不直接与视图耦合的代码。应用程序状态一般通过持久性的数据库管理。在基于 MVC 的应用程序中，模型会封装底层数据库访问。例如，对于人力资源应用程序，模型可能通过 Employee 类来表示人力资源数据库中雇员信息表的记录，还可能在内存中维护 Employee 对象的集合。

MVC 视图组件负责呈现应用程序的用户界面。其中可能包含某些控件(如显示和编辑数据的控件)。这些控件通常与模型耦合。例如，应用程序可能要为显示和编辑雇员数据显示不同的视图。详细信息视图一般以只读方式显示详细信息，而编辑视图可能包含用于编辑雇员信息中各字段的控件。

MVC 控制器组件负责管理与最终用户的交互。虽然视图组件负责呈现，但页面呈现与用户交互是相互独立的。在 ASP.NET 上下文中，管理用户交互涉及管理 Web 流量、更新模型(应用程序状态)，以及管理 UI(即生成正确的 HTML)。

图 22.1 表示了这三个组件之间的关系。需要注意的是，控制器能够向模型和视图发送指令。确切地讲，控制器会在必要的时候更新模型，并在需要呈现时与视图交互。视图能够向模型发送指令，以确保显示应用程序当前最新的状态。

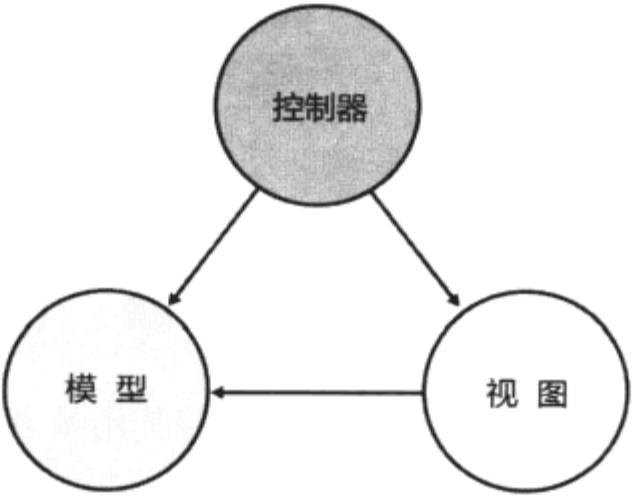


图 22.1 模型、视图和控制器之间的关系

软件设计模式

MVC 架构由几种软件模式(pattern)组成。在介绍 MVC 的 ASP.NET 版本之前，有必要简单地认识一下 MVC 中蕴含的软件设计模式。

当一个人做庞大的设计时，他是无暇从头做起的。

——Christopher Alexander，建筑师

建筑师 Christopher Alexander 的这句话最初只针对建筑结构。然而，这种思想在软件开发领域也适用。如果已经存在某种确定的、可靠的方法或组件能够帮助我们快速完成工作，那么从头去做设计的意义并不大。

有经验的软件开发者可能还记得 20 世纪 90 年代中期的“模式”运动。这大约起始于 1993 年年末。软件设计模式的思想来自于 Christopher Alexander 在建筑结构模式上的著作。作为建筑架构师，Christopher Alexander 发现建筑物中存在许多重复的设计。他在结构模式方面的研究成果是两本非常有名的书——*A Pattern Language* 和 *The Timeless Way of Building*。

在 1995 年，4 名计算机科学家合作完成了一部软件方面对应的著作：*Design Patterns: Elements of Reusable Object-Oriented Software*，Erich Gamma、Richard Helm、Ralph Johnson、John Vlissides，由 Addison Wesley 出版。此书包含多种解决常见软件问题的软件设计模式(方法)。

软件设计模式给出了一种形式化的方法来解决软件问题。软件模式一般是从一个较高的层面描述解决方案，避免提及特定实现。模式的描述包括模式背后的意图、它所解决的问题，以及模式的特定应用举例。模式一般具有一个描述性名称，如“命令”(Command)。“命令”模式背后的思想是将请求用对象包装。这样便可以通过对象包装用户的不同请求、使请求排队，并支持可以撤销的操作。“Microsoft 消息队列”(MSMQ)是“命令”模式的一个典型应用。

“模型-视图-控制器”(MVC)架构是一种常见软件架构。MVC 架构结合了多种模式，能够为创建应用程序奠定良好基础。总体说来，MVC 模式包括“组合”(Composite)模式、“观察者”(Observer)模式和“策略”(Strategy)模式。“组合”模式是将软件划分为不同任务(对于本书所讨论的话题，这些任务包括数据库访问、用户界面呈现和交互管理)。“观察者”模式描述了软件组件之间的发布与订阅关系。“策略”模式可以使程序在运行时选择其行为。

22.2 ASP.NET 与 MVC

如今，MVC 也被引入到了 ASP.NET 中。我们可以将 ASP.NET 看作是耦合度很低的类的集合，这些类能够协同工作来处理请求。ASP.NET 的管线作为一个基础，不同处理程序都可以挂接到它上面。管线经过配置后，对不同类型文件的请求会被引向对应的处理程序。

虽然 ASP.NET 适合处理 Web 请求，但 ASP.NET 并不是真正的框架。Web 窗体考虑到了 Web 开发人员的要求使之获得了极大的方便。而 MVC 清晰地分离了 Web 开发者关注的部分，它才是真正的框架。之前介绍过，在 MVC 中，模型负责处理应用程序状态，视图负责处理呈现，而控制器负责管理与最终用户的交互。

ASP.NET MVC 框架能够与 ASP.NET 的其他部分共存。该框架能够在脱离 .aspx、.ascx、.master 和 Global.asax 文件的情况下运行。MVC 也能够与 ASP.NET 现有的“Forms 身份验证”和标准的成员资格与角色提供程序协同工作。MVC 完全能够访问现有的数据缓存与输出缓存，以及现有的数据提供程序。开发者能够将 ASP.NET 中的任何功能融入 MVC 框架。

为使 MVC 与 ASP.NET 共存，MVC 框架将灵活性放在设计目标的首位，使它在各个方面都可被定制。例如，Visual Studio 为基于 MVC 应用程序生成的代码包括了一个标准的路由表(routing table)，使应用程序能够控制不同类型请求的处理。虽然内建的路由机制在多数情况下都可以使用，但它在某些情况下并不是最佳选择。幸运的是，对于 MVC 应用程序，更换路由策略是相对容易的。

通过将所有 MVC 路由架构放在控制器中，我们可以轻松替换路由策略。与普通的 ASP.NET 开发相比，这样做具有一些显著优势。例如，MVC URL 映射能够向最终用户隐藏混乱的 URL。URL 映射由 MVC 框架本身负责，开发者可以防止最终用户看到混乱的 URL 名称。这些 URL(如/contacts/edit/3256)能够在内部被清晰地映射。乍看起来，这似乎是微不足道的。然而，这确实使得 Web UI 更加简洁，掩盖了冗长的、杂乱的 URL，并有利于实现相对智能的路由选择。例如，基于 MVC 框架的应用程序不需要使用扩展名，这样便可以通过它来实现符合“搜索引擎优化”(Search Engine Optimization, SEO)和针对服务的“具象状态传输”(Representational State Transfer, REST)的命名模式。

MVC 框架仍能够利用现有的 ASP.NET 文件类型(如.aspx 文件、.ascx 文件和.master 文件)。MVC 能够将这些标记文件作为“视图模板”(view template)。视图模板允许开发者使用内联的编码语法(即<% %>片段)。然而，MVC 并不将每次交互回发直接定向至处理程序，而是将请求路由至某个 Controller 类。这使得 MVC 应用程序在一般情况下更容易测试。如果愿意，还可以使用“测试驱动设计”(Test-Driven Design, TDD)^①技术。由于传统“Web 窗体”的固有特性，它不直接支持自动化单元测试。

最后需要说明的是，在使用 MVC 框架时，ASP.NET 的其他功能仍然可用。这包括输出缓存、会话状态、提供程序架构和配置。

22.3 ASP.NET MVC 与 Web 窗体

ASP.NET MVC 框架从本质上区别于“Web 窗体”。MVC 清晰地分离了数据源、程序与数据的交互和数据的表示这 3 方面内容。MVC 会强制这样做，而在使用“Web 窗体”时则

^① 译者注：亦作“测试驱动开发”(Test-Driven Development, TDD)。

需要开发者自行分离。

MVC 有意避开了标准“Web 窗体”的某些功能。例如, ASP.NET MVC 不直接支持视图状态(此功能的出现源于服务器端控件)。因此,我们不会在 MVC 框架呈现的页面中找到任何隐藏字段。

MVC 与“Web 窗体”最显著的区别之一是在回发事件的处理方面。对于 MVC,事件会根据路由表进行定向,而与服务器端控件无关;而对于“Web 窗体”,事件一般通过页面的特定事件处理程序完成。

MVC 框架会区别对待数据、表示和程序逻辑,从而使组件之间彼此间的界限更为清晰。这样,测试和调试变得更为容易。事实上,ASP.NET MVC 框架还支持“测试驱动开发”。这为大型团队构建和支持项目提供了诸多便利。

MVC 并不依赖于某种功能(如视图状态和服务器端控件),而是将更多生成正确 HTML 的职责交给开发者。虽然这为开发者增加了一定的负担,但开发者可以更直接地控制 HTML 的呈现。

最后,“Web 窗体”和服务器端控件的事件处理机制,使得基于“Web 窗体”的应用程序的执行路径难以跟踪。MVC 则完全不同,因为所有请求都会经过应用程序的一处——路由表。利用路由表,开发者能够在一处控制请求的路由,而不需要在页面中添加多个控制点(即事件处理程序)。

22.4 MVC 与测试

MVC 框架将功能划分到职责相对单一的区域。这使得 MVC 应用程序比“Web 窗体”应用程序更容易测试。虽然“Web 窗体”模型通过分离表示(用户界面)与程序逻辑类改进了 Web 编程方式,但这种模型仍将其他部分混在一起。数据访问往往与程序代码的其他部分混在一起。例如,如果实现特殊身份验证逻辑,需要在用户单击登录按钮而进行回发时直接查找用户,那么查找逻辑通常就在代码旁置文件中。还有,请求会直接定向到当前的.aspx 文件。不使用 MVC,则很难选择其他路由方式。

MVC 框架将不同职责分离,从而实现了更适合进行单元测试的模块化架构。测试一般的 ASP.NET 程序通常要通过单击页面上的所有控件和 UI 元素来确保其正常工作。如果出现错误,则需要跟踪和修正。MVC 框架支持测试驱动开发。对于这种方法,我们首先定义代码的功能,然后在编写实际的代码之前编写测试用例。这样,我们不运行整个应用程序也能够通过 MVC 框架逐一进行用户交互。也就是说,我们可以使用现有的测试框架(如 MS Test 和 NUnit)对应用程序进行单元测试。

22.5 MVC 与 ASP.NET 的结合

在构建网站之前，最好先确定是否采用 MVC。虽然 ASP.NET 的标准组件能够完全独立于 MVC 框架工作，但使用 Visual Studio 来创建 MVC Web 应用程序可以免除许多琐碎的工作。为使应用程序采用 MVC 模式，Visual Studio 会生成几个组件。例如，必须注册路由表才能使 HTTP 请求由正确的方法进行处理。Visual Studio 会在应用程序的 Global.asax 文件的 Application_Start 事件中插入以下代码：^①

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        "Default",                                     // 路由名称
        "{controller}/{action}/{id}",                 // 带有参数的 URL
        new { controller = "Home", action = "Index", id = UrlParameter.Optional } // 参数默认值
    );
}

protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();

    RegisterRoutes(RouteTable.Routes);
}
```

MVC 框架包含一个名为 RouteTable 的类，其中包含许多数据项，这些数据项指定了 MVC 框架处理请求的方法。Visual Studio 会默认为 RouteTable 添加一个数据项，用于将 URL 映射到特定控制器中的特定方法，其中还包含处理请求所涉及的必要参数。Visual Studio 生成的路由会将默认请求映射到 Home 控制器中无参数的 Index 方法。通常，这个 Index 方法用于显示网站的默认信息，是用户使用网站的一个起点。我们可以根据特定场景在 RouteTable 中添加所需的数据项。

除了在 Global.asax 文件中设置 RouteTable 外，Visual Studio 还会生成特殊的 Default.aspx 文件。事实上，这个文件是必要的，Visual Studio 会生成一段注释，提示不要删除它。查看 Default.aspx 的 Page_Load 处理程序便不难发现，它会注册 MVC HttpHandler，使其监听基于 MVC 的请求。

请求的路径

在 MVC 框架各部分就绪后，应用程序便可以开始处理请求了。请求仍会以传统方式到达

^① 译者注：这里的代码根据 Visual Studio 2010 RTM 版做了更新。

网站。对于第一个请求，ASP.NET 会创建应用程序实例(Global.asax 中的类定义的)。此后，首次访问 Default.aspx 时，MVC 处理程序便会被注册，进而请求能够被正确路由。图 22.2 展示了请求在系统中可能经历的路径。

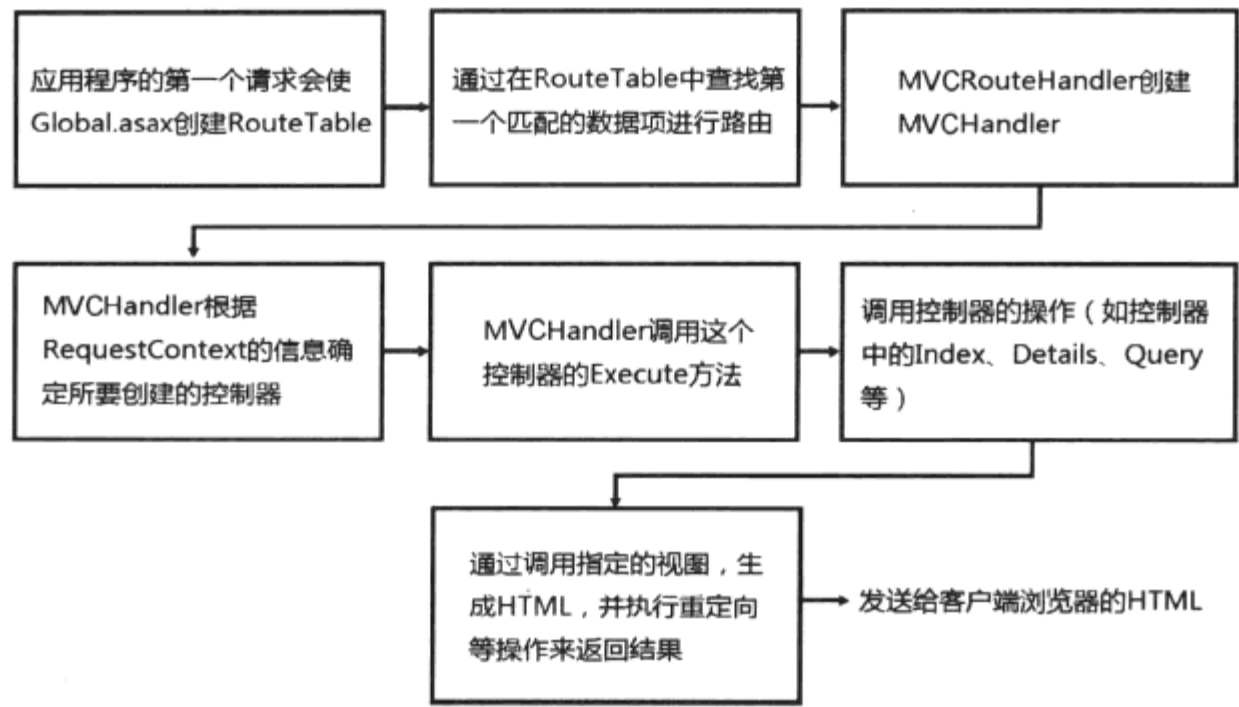


图 22.2 请求在 MVC 框架经历的路径

下面的练习将通过创建一个基于 MVC 的网站来演示 MVC 框架。

➤ 创建一个 MVC 网站

- 1. 启动 Visual Studio。在“文件”菜单中选择“新建”|“项目”。从模板中选择“ASP.NET MVC 2 Web 应用程序”(如下图所示)，并将这个网站命名为 MVCORama。Visual Studio 会询问是否为应用程序创建单元测试，单击“确定”。此时，Visual Studio 会创建一个 MVC 网站。



- 2. 完成后，Visual Studio 会基于 MVC 框架生成一个完整的 ASP.NET 项目。在“解决方案资源管理器”中，我们可以找到 MVC 的几个文件夹：Controllers、Models 和 Views。

3. 运行这个应用程序，体验一下这个网站最初的效果。
4. 为理解 MVC 与 ASP.NET 其他部分的结合方式，在设计器中打开 Views\Shared 文件夹下名为 Site.master 的母版页。其中包含许多常见的 HTML，还包含一些以内联代码形式出现的 HTML 辅助方法。例如，在这个母版页中，我们会发现应用程序使用了某种名为 LogOnUserControl 的工具。现在请将网站的标题由“我的 MVC 应用程序”改为“Dot Net References”。
5. 我们可以修改母版页的样式与颜色。主 CSS 文件是 Content 文件夹下名为 Site.css 的文件，打开它。为修改主体的颜色，在代码中的 body 样式描述中右击，从快捷菜单中选择“生成样式”。使用“修改样式”对话框，我们可以修改背景颜色和其他样式元素。修改完毕后，在 Visual Studio 中按 Ctrl+F5 来运行网站。浏览器应该反映出对页面样式元素的修改。
6. Visual Studio 会创建 AccountController、HomeController，以及为支持相应视图所需的文件，为后续开发奠定基础。Home 控制器和视图包含一行文本：“欢迎使用 ASP.NET MVC!”。为将其替换为别的内容，请打开 Controllers\Home 文件夹下的 HomeController.cs 文件。这条问候语是 Index 方法输出的，修改该方法使其输出“Welcome to the Dot Net References Site”。然后，打开 Views\Home 文件夹下的 Index.aspx 文件，将下面的代码

若要了解有关 ASP.NET MVC 的更多信息，请访问

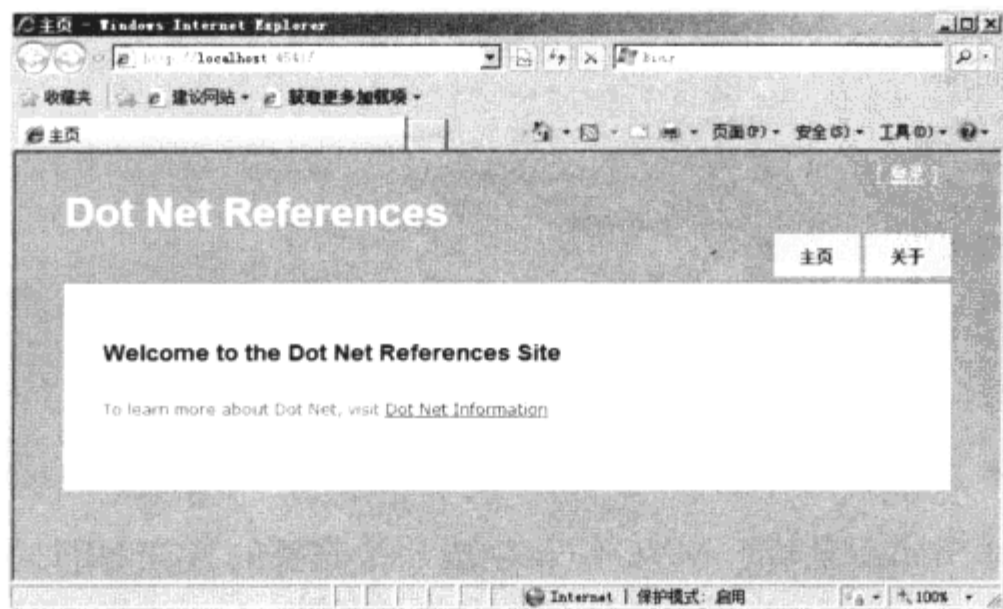
```
<a href="http://asp.net/mvc"
title="ASP.NET MVC 网站">http://asp.net/mvc</a>。
```

修改为：

To learn more about Dot Net, visit

```
<a href="http://msdn.microsoft.com/en-us/netframework/default.aspx"
title="Dot Net Framework Development Center"> Dot Net Information </a>
```

7. 运行网站后，我们会看到新修改的文本，其中的链接会将我们带到 Microsoft 的“.NET Framework 开发人员中心”(英文)(见下图)。



8. 接下来，为应用程序创建数据模型。从配套资源第 22 章的项目中复制 ASPNETStepByStep4.mdf 到当前项目。这是一个包含两个表的 Microsoft SQL Server 数据库文件。一个表包含一些 .NET 参考书的名称，另一个表包含一些指向 .NET 开发者网站的链接。可以将这些表作为这个 MVC 应用程序模型的基础。为复制 ASPNETStepByStep4.mdf 文件到 App_Data 目录，在“解决方案资源管理器”中右键单击 App_Data 文件夹，选择“添加”|“现有项”。在配套资源中找到这个文件。
9. 在为应用程序添加这个数据库后，需要创建一些访问数据的辅助类。最简单的方法是创建 LINQ to SQL 包装类。在“解决方案资源管理器”中右键单击 Models 文件夹，从快捷菜单中选择“添加”|“新建项”。在“添加新项”对话框的左侧选择“数据”，在右侧窗格中选择“LINQ to SQL”，将这个文件命名为 DotNetReferences.dbml。确认后，Visual Studio 会创建“数据库脚本语言”(Database Markup Language, DBML)源文件，并将其添加到项目中。从“服务器资源管理器”将 DotNetReferences 表拖放至“设计器”界面，在设计界面选中新建的类，将其 Name 属性修改为 DotNetReference。^①这样，Visual Studio 会创建一个名为 DotNetReference 的包装类来表示数据库表中的一条记录。稍后我们会用到这个类。
10. DotNetReference 类只在数据库上下文有效时有效。Visual Studio 除创建 DotNetReference 类之外，还会创建一个名为 DotNetReferencesDataContext 的类来表示 DotNetReferences 表。我们并不直接访问数据库，而是使用 LINQ 和数据库管理器类方便地使用数据。在“解决方案资源管理器”中，右键单击 Models 文件夹，添加一个类，将其命名为 DotNetReferencesManager。这里要通过 DotNetReferencesManager 类来包装 DotNetReferencesDataContext，因此在 DotNetReferencesManager 类中创建 DotNetReferencesDataContext 类的实例。然后，为 DotNetReferencesManager 类添加一个名为 GetAllReferences 的方法。将其返回类型定义为 IQueryable<DotNetReference>。我们可以这样通过 DotNetReferencesDataContext 类获取所有记录：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
namespace MVCORama.Models
{
    public class DotNetReferencesManager
    {
        DotNetReferencesDataContext dataContext =
            new DotNetReferencesDataContext();

        public IQueryable<DotNetReference> GetAllReferences()
        {
```

^① 译者注：Visual Studio 有时会将复数的表名改为单数名词，而有时则不会。在 DBML 中定义的类，其名称都应该是单数的，因为这些类代表表中的单条记录。

```

        return dataContext.DotNetReferences;
    }
}
}

```

11. 为 DotNetReferences 模型创建一个视图。在项目的 Views 文件夹上右击，选择“添加”|“新建文件夹”，添加一个新的视图文件夹，将这个文件夹命名为 DotNetReferences。在这个文件夹上右击，从快捷菜单中选择“添加”|“视图”。此时，Visual Studio 会显示用于配置视图的“添加视图”对话框。Visual Studio 会将这个视图命名为 Index。选择“创建强类型视图”，使其基于 DotNetReference 类创建(这个类会显示在对话框的下拉列表中)。保持有关母版页的默认设置。最后，Visual Studio 会通过对 DotNetReferences 模型的反射生成以下视图代码：

```

<%@Page Title=""
    Language="C#"
    MasterPageFile="~/Views/Shared/Site.Master"
    Inherits="System.Web.Mvc.ViewPage<IEnumerable<MVCORama.Models.DotNetReference>>"%>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Index
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
    <h2>Index</h2>
    <table>
        <tr>
            <th></th>
            <th>ID</th>
            <th>Title</th>
            <th>AuthorLastName</th>
            <th>AuthorFirstName</th>
            <th>Topic</th>
            <th>Publisher</th>
        </tr>
        <%foreach (var item in Model) { %>
            <tr>
                <td>
                    <%:Html.ActionLink("Edit", "Edit", new { id=item.ID }) %> |
                    <%:Html.ActionLink("Details", "Details", new { id=item.ID })%> |
                    <%:Html.ActionLink("Delete", "Delete", new { id=item.ID })%>
                </td>
                <td>
                    <%: item.ID %>
                </td>
                <td>
                    <%:item.Title%>
                </td>
                <td>
                    <%:item.AuthorLastName%>
                </td>
                <td>

```



```

        <%:item.AuthorFirstName%>
    </td>
    <td>
        <%:item.Topic%>
    </td>
    <td>
        <%:item.Publisher%>
    </td>
</tr>
<% } %>
</table>
<p>
    <%:Html.ActionLink("Create New", "Create") %>
</p>
</asp:Content>

```

12. Index.aspx 文件采用的是一般的 ASP.NET 语法。Page 指令在页面的最顶端，后面是 ASP.NET 中的 Content 控件(用于替换母版页中指定位置的内容)。这个视图已经与 DotNetReference 类定义的模型关联。如果阅读 Index 页面的代码，则会发现，该视图对 DotNetReferences 模型中的记录进行迭代。为使视图正确显示，只需要向模型添加一个控制器。此外，还可以添加一个编辑视图。
13. 创建一个附属于 DotNetReferences 模型的控制器来处理请求。右键单击 Controllers 文件夹，选择“添加”|“控制器”。在“添加控制器”对话框中输入 DotNetReferencesController^①。Visual Studio 会创建一个类，它继承于 MVC 框架的 Controller 类。找到名为 Index、返回类型为 ActionResult 的方法。在这个 Index 方法中，初始化一个 var 变量，用 DotNetReferencesManager.GetAllReferences 方法为其赋值。这个变量所引用的是一个无类型的集合，可以用于管理任意对象——有时可能无法预知集合中对象的类型。下面用加粗字体标出了要添加和修改的代码：

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Mvc.Ajax;
using MVCORama.Models;

namespace MVCORama.Controllers
{
    public class DotNetReferencesController : Controller
    {
        DotNetReferencesManager dotNetReferencesManager =
            new DotNetReferencesManager();
    }
}

```

① 译者注：为使 MVC 框架的路由机制能够正确识别控制器，需要使所有控制器的名称都有 Controller 后缀。

```

public ActionResult Index()
{
    var dotNetReferences =
        dotNetReferencesManager.GetAllReferences().ToList();
    return View("Index", dotNetReferences);
}
}
}

```

14. 在导航到 DotNetReferences 页面后，MVC 框架应呈现 DotNetReferences 视图。为此，我们要在母版页中添加一个选项卡标签，以便导航到 DotNetReferences 页面，如下所示：

```

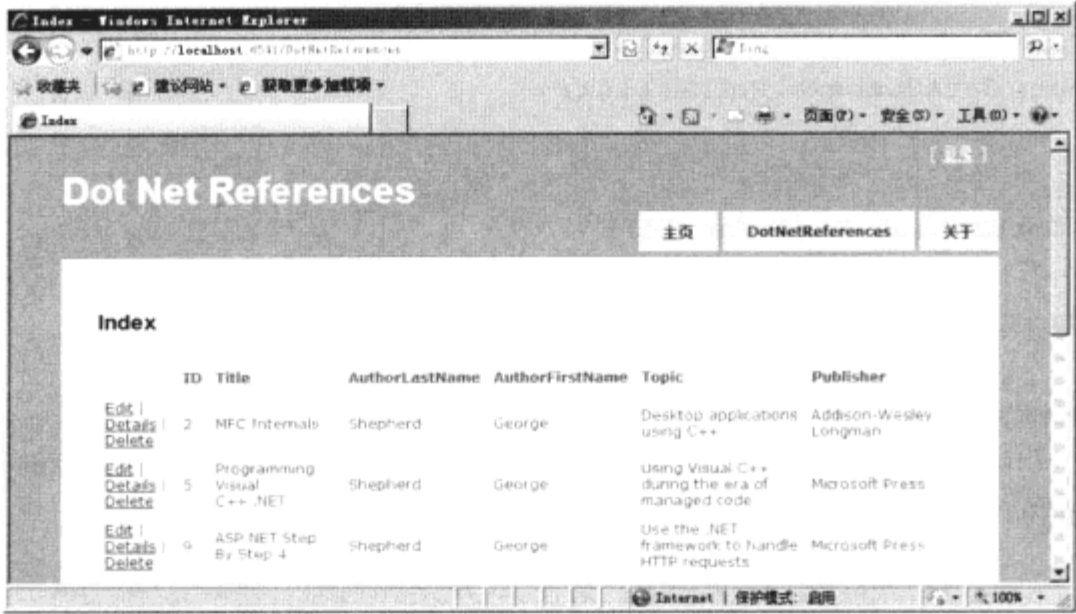
<%@Master Language="C#" Inherits="System.Web.Mvc.ViewMasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/
DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<metahttp-equiv="Content-Type"content="text/html; charset=utf-8"/>
    <title><asp:ContentPlaceholder ID="TitleContent" runat="server"/></title>
    <link href="../../../Content/Site.css" rel="stylesheet" type="text/css"/>
</head>
<body>
    <div class="page">
        <div id="header">
            <div id="title">
                <h1>Dot Net References</h1>
            </div>
            <div id="logindisplay">
                <%Html.RenderPartial("LogOnUserControl"); %>
            </div>
            <div id="menucontainer">
                <ul id="menu">
                    <li><%:Html.ActionLink("主页", "Index", "Home")%></li>
                    <li><%:Html.ActionLink("DotNetReferences", "Index",
                        "DotNetReferences")%></li>
                    <li><%:Html.ActionLink("关于", "About", "Home")%></li>
                </ul>
            </div>
        </div>

        <div id="main">
            <asp:ContentPlaceholder ID="MainContent" runat="server"/>
            <div id="footer">
            </div>
        </div>
    </div>
</body>
</html>

```

15. 编译应用程序，然后按 Ctrl+F5 快捷键运行。在默认页面显示后，通过刚刚添加的选项卡导航到 DotNetReferences 页面。此时，应用程序应显示 DotNetReferences 数据库

的内容：



在下面的练习中，我们将在这个应用程序中创建一个相对完善的场景，使应用程序更具交互性。在这个过程中我们将学习如何显示数据项的详细信息，以及如何编辑、删除和创建数据项。

➤ 实现添加、删除和更新

- 1. 打开之前练习创建的 MVCORama 应用程序。之前的练习只是显示了模型的内容，而没有其他功能。这里，我们将使用另一个数据库表(DotNetLinks)来完成几个额外的功能。
- 2. 为 DotNetLinks 表创建模型。与上一个练习一样，为 DotNetLinks 表添加一个 LINQ to SQL 类。右键单击 Models 文件夹，单击“添加”|“新建项”。在“添加新项”对话框中，单击“数据”节点，然后选择“LINQ to SQL 类”。将这个 LINQ to SQL 类命名为 DotNetLinks。从“解决方案资源管理器”拖动 DotNetLinks 表到设计界面上。选中这个类，在属性窗口中将 Name 修改为 DotNetLink。这样，Visual Studio 会生成名为 DotNetLink 的包装类。
- 3. 与 DotNetReferences 表一样，为 DotNetLinks 表创建数据管理器。右键单击 Models 文件夹，选择“添加”|“新建项”。从模板中选择“类”，将其命名为 DotNetLinksManager。Visual Studio 会为我们创建这个类。
- 4. 在 DotNetLinksManager 类中创建 DotNetLinksDataContext 的实例。添加 4 个方法，分别用于从表中获取所有记录、在 DotNetLinksDataContext 中查找某个 DotNetLink、向 DotNetLinksDataContext 添加 DotNetLink，以及从 DotNetLinksDataContext 中删除 DotNetLink。最后，添加一个将更改提交给底层表的方法。后面的控制器会用到这些方法。

```
public class DotNetLinksManager
{
    DotNetLinksDataContext dataContext =
        new DotNetLinksDataContext();
```



```

public IQueryable<DotNetLink> GetAllLinks()
{
    return DataContext.DotNetLink;
}

public DotNetLink Find(int id)
{
    DotNetLink dotNetLink;

    dotNetLink =
        DataContext.DotNetLink.SingleOrDefault(l => l.ID == id);
    return dotNetLink;
}

public void Add(DotNetLink dotNetLink)
{
    DataContext.DotNetLink.InsertOnSubmit(dotNetLink);
}

public void Delete(DotNetLink dotNetLink)
{
    DataContext.DotNetLink.DeleteOnSubmit(dotNetLink);
}

public void Save()
{
    DataContext.SubmitChanges();
}
}

```

5. 在 Controllers 文件夹中添加 DotNetLinks 控制器。右击 Controllers 文件夹，选择“添加”|“控制器”。Visual Studio 会自动创建一个控制器。以成员变量的形式在该控制器中创建 DotNetLinksManager 的实例。

```

Public class DotNetLinksController: Controller
{
    DotNetLinksManager dotNetLinksManager =
        new DotNetLinksManager();

    // more code
}

```

6. 在 Views 文件夹下创建一个名为 DotNetLinks 的文件夹，在其中添加一个名为 Index.aspx 的视图。右键单击 Views\DotNetLinks 文件夹，选择“添加”|“视图”。将视图数据类设置为 DotNetLink 类，并使视图以列表(List)形式显示链接。
7. 回到 DotNetLinksController，使 Index 方法根据所有可用链接创建一个新的视图(为此要调用 DotNetLinksManager.GetAllLinks 方法)。注意，这个方法会捕获所有异常，从而使运行不受影响。另一种策略是使异常沿管线传递。

```
public ActionResult Index()
{
    try
    {
        var dotNetLinks =
            dotNetLinksManager.GetAllLinks().ToList();
        return View("Index", dotNetLinks);
    }
    catch (Exception ex) {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return View();
    }
}
```

8. 打开 Index.aspx 文件，修改界面，使 URL 以链接形式呈现。这样会使得页面将 URL 显示为有效的链接。找到显示 URL 的 HTML 列，将以下代码

```
<td>
    <%: item.URL %>
</td>
```

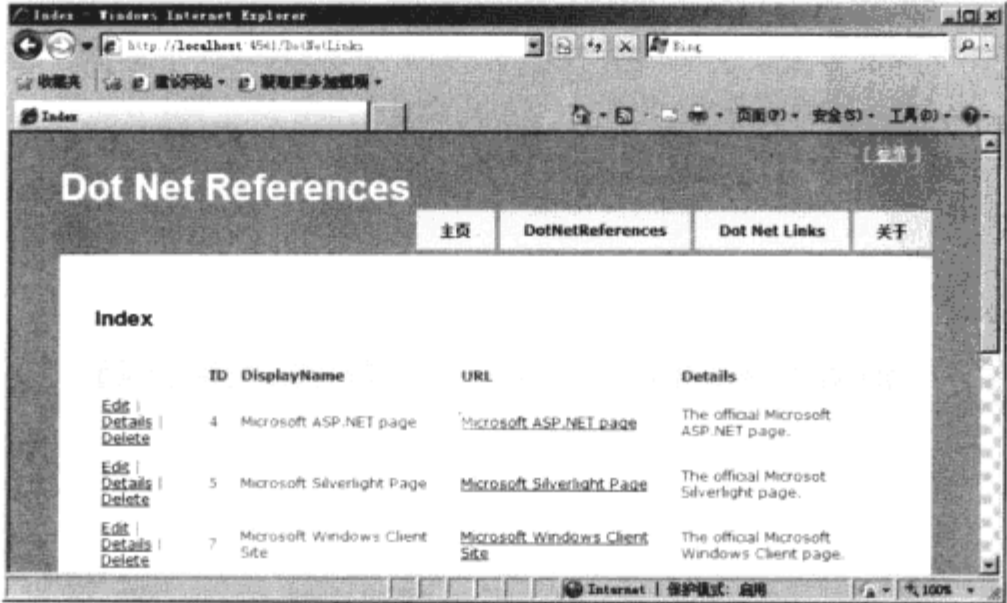
修改为：

```
<td>
    <a href="<%: item.URL %>"><%:item.DisplayName %></a>
</td>
```

9. 打开 Site.Master 文件(母版页)，在菜单中添加一个新的选项卡，以便显示 DotNetLinks 信息：

```
<ul id="menu">
    <li><%:Html.ActionLink("主页", "Index", "Home")%></li>
    <li><%:Html.ActionLink("DotNetReferences", "Index", "DotNetReferences")%></li>
    <li><%:Html.ActionLink("Dot Net Links", "Index", "DotNetLinks")%></li>
    <li><%:Html.ActionLink("关于", "About", "Home")%></li>
</ul>
```

10. 运行程序，导航到 DotNetLinks 页面。此时我们会发现，URL 已显示为常见的、可以单击的 HTTP 链接(如下图所示)。



11. 下面实现详细信息的显示——用户可以查看链接相关的详细信息。在 Views\DotNetLinks 文件夹上右击，选择“添加”|“视图”。在对话框中，将视图命名为 Details，将视图的数据类设置为 DotNetLink 类，并从“视图内容”组合框中选择“Details”。Visual Studio 会创建一个派生自 DotNetLink 类的视图。
12. 下面，我们需要使 MVC 框架知道如何响应针对特定链接的详细信息请求。为 DotNetLinksController 类添加一个公共方法，以便向视图返回单个链接的详细信息。调用 DotNetLinksManager.Find 方法，并将 ID 传入。ID 参数会以 URL 参数的形式传入，并由 MVC 框架中的托管类型进行包装。Visual Studio 生成的 Index 视图包含“Details”链接，用于为显示模型中的单条数据提供导航。在 DotNetLinksManager 中找到指定的 URL 信息后，调用控制器的 View 方法，分别传入字符串“Details”和 DotNetLink 这两个参数：

```
// Get the details for a single link and show them:
public ActionResult Details(int id)
{
    try
    {
        DotNetLink dotNetLink = dotNetLinksManager.Find(id);
        if (dotNetLink != null)
        {
            return View("Details", dotNetLink);
        }
        else
        {
            return View();
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return View();
    }
}
```

13. 运行网站，通过刚刚添加的选项卡标签导航到 DotNetLinks 的 Index 页面，然后单击某个 Details 链接。此时浏览器会显示下图中所示的以下内容。



14. 下面，我们实现在数据库中添加新数据。首先，在 Views\DotNetLinks 文件夹下添加一个强类型的视图。右键单击 Views\DotNetLinks 文件夹，选择“添加”|“视图”。通过对话框对视图进行配置，将这个视图命名为 Create，使视图采用 DotNetLink 类。然后，从“视图内容”组合框中选择“Create”。
15. 为控制器添加几个方法，以支持数据项的添加。首先，编写一个名为 Create 的方法，它只需简单地显示默认视图。下面的代码会显示默认的 Create 视图，其中有等待用户输入 DotNetLink 对象各属性的文本框。

```
public ActionResult Create()
{
    return View();
}
```

16. 添加一个名为 DotNetLinkFromFormsCollection 的静态辅助方法，它接受一个 FormCollection 参数，并返回 DotNetLink。FormCollection 类是一个键/值集合，代表回发的内容。它用于填充 DotNetLink 对象：

```
private static DotNetLink DotNetLinkFromFormsCollection(FormCollection collection)
{
    DotNetLink dotNetLink = new DotNetLink();
    dotNetLink.DisplayName = collection["DisplayName"];
    dotNetLink.URL = collection["URL"];
    return dotNetLink;
}
```

17. 添加另一个 Create 方法，它接受 FormCollection 参数，并返回 ActionResult。为这个方法添加 AcceptVerbs 特性，并传入枚举值 HttpVerbs.Post。这会引导 MVC 框架处理这种回发。MVC 框架会使用回发的结果填充 FormCollection。通过 DotNetLinkFromFormsCollection 方法来填充 DotNetLink 的实例。调用 DotNetLinksManager.Add 方法，将这个 DotNetLink 实例添加到集合中，然后使用 DotNetLinksManager.Save 方法将更改提交到底层数据库(如下所示)。注意，这不是最终产品的代码，它没有验证用户的输入。实际的应用程序一般要对用户的输入进行检查，以防止无效的输入导致错误或使得网站遭受攻击。

```
// Create scenario
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Create(FormCollection collection)
{
    try
    {
        DotNetLink dotNetLink =
            DotNetLinkFromFormsCollection(collection);
        if (dotNetLinksManager.Find(dotNetLink.ID) == null)
        {
            dotNetLinksManager.Add(dotNetLink);
            dotNetLinksManager.Save();
        }
    }
}
```

```

        return RedirectToAction("Index");
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return View();
    }
}

```

18. 运行应用程序，尝试向 DotNetLinks 集合中添加一个新链接。例如，单击“Create New”链接。在 Create 页面中，输入显示名称 **MSDN**，在“URL”字段中输入 **http://msdn.microsoft.com**，然后在“Details”字段输入一些注释。单击 Create 按钮后，此链接信息便会被插入到数据库中。
19. 接下来创建一个编辑视图。右键单击 Views\DotNetLinks 文件夹，添加一个基于 DotNetLink 类的强类型视图。将这个视图命名为 Edit，为视图选择 DotNetLink 类，在“视图内容”组合框中选择“Edit”。Visual Studio 会生成一个用于编辑现有数据项的新视图。
20. 为控制器添加一个处理编辑的方法。它接受一个整数值，这个值代表待编辑项的 ID。MVC 框架会在导航到 Edit 页面时调用控制器中的这个方法(为此，需要在 DotNetLinks 页面中单击某个“Edit”链接)。在这个方法中，通过 DotNetLinksManager.Find 方法来查找具有指定 ID 的 DotNetLink。然后调用控制器的 View 方法，传入字符串“Edit”(用于调用 Edit 视图)和从 DotNetLinksManager 获取的 DotNetLink 对象的引用：

```

// handle editing...
public ActionResult Edit(int id)
{
    try {
        DotNetLink dotNetLink =
            dotNetLinksManager.Find(id);
        if (dotNetLink != null)
        {
            return View("Edit", dotNetLink);
        }
        return View();
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return View();
    }
}

```

21. 在控制器中添加一个处理回发的方法。将这个方法命名为 Edit，并使它接受两个参数——一个是整型值，用于指定被编辑项的 ID，另一个是 FormCollection。使这个 Edit 方法返回 ActionResult。用 AcceptVerbs 特性来修饰此方法，以便响应回发。为获取指定的 DotNetLink，调用 DotNetLinksManager.Find 方法并传入 ID。通过控制器基类的

UpdateModel 方法来填充来自集合的 DotNetLink 对象(UpdateModel 是 MVC 框架的一部分，能够自动更新模型)。然后，调用 DotNetLinkManager.Save 方法来将信息保存到数据库：

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(int id, FormCollection collection)
{
    try
    {
        DotNetLink dotNetLink = dotNetLinksManager.Find(id);
        UpdateModel(dotNetLink);
        dotNetLinksManager.Save();
        return RedirectToAction("Index");
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return View();
    }
}
```

22. 最后，实现删除功能。在 Views\DotNetLinks 文件夹下添加一个基于 DotNetLink 类的强类型视图，将其命名为 Delete。在“视图内容”组合框中选择“Empty”。我们使用这个页面来确认用户操作。在这个页面中添加一些文本，要求用户确认指定的记录将要被删除。通过调用 Html.BeginForm 在页面中添加一个表单。在该表单中添加一个 Submit 按钮，单击这个按钮便会引起回发。

```
<%@Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<MVCORama.Models.DotNetLink>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent" runat="server">
    Delete
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

    <h2>
        Confirm Delete
    </h2>
    <div>
        <p>Do you want to delete this link?:
        <i><%=Html.Encode(Model.DisplayName) %>? </i> </p>
    </div>
    <%using (Html.BeginForm()) { %>
        <input name="confirmButton" type="submit" value="Delete"/>
    <% } %>

</asp:Content>
```


23. ^①为控制器添加用于删除指定的 DotNetLink 所需的方法。首先，添加一个名为 Delete 且只接受一个整型参数的方法，使这个方法返回 ActionResult。这个方法用于响应发起删除操作的 GET 请求。调用 DotNetLinksManager.Find 方法并传入 ID 来获取指定的 DotNetLink 的引用。然后，调用控制器的 View 方法，传入字符串“Delete”和 DotNetLink 的引用。这样，删除确认页面便会正确显示。

```
// Methods for deleting
public ActionResult Delete(int id)
{
    try
    {
        DotNetLink dotNetLink =
            dotNetLinksManager.Find(id);
        if (dotNetLink != null)
        {
            return View("Delete", dotNetLink);
        }
    }
    else
    {
        return View();
    }
}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine(ex.Message);
    return View();
}
```

24. 最后，为控制器添加一个接受一个整型值和一个 FormCollection 的 Delete 方法。使用 AcceptVerbs 特性对其进行修饰，并传入 HttpVerbs.Post 枚举，以指明这个方法用于处理回发。也就是说，用户单击删除确认按钮，这个方法会被调用。使用 DotNetLinksManager.Find 方法，根据传入控制器的 ID 来查找指定的 DotNetLink。调用 DotNetLinksManager.Delete 和 DotNetLinksManager.Save 方法，从数据库移除记录。最后，使用控制器的 RedirectToAction 方法来显示 Index 视图。

```
[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Delete(int id,
    FormCollection formsCollection)
{
    try
    {
        DotNetLink dotNetLink =
            dotNetLinksManager.Find(id);

        if (dotNetLink != null)
```

① 译者注：Visual Studio 2010 RTM 会自动添加 Delete 链接，这里将原书第 23 步删去。

```
        {
            dotNetLinksManager.Delete(dotNetLink);
            dotNetLinksManager.Save();
            return RedirectToAction("Index");
        }
        else
        {
            return View();
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine(ex.Message);
        return View();
    }
}
```

25. 运行程序，尝试删除一条链接。

22.6 快速参考

目 标	操 作
新建 MVC 网站	在创建 Web 应用程序项目时选择 MVC 模板
通过创建数据库表的包装类来支持模型	右键单击 Models 文件夹，单击“添加” “新建项”。从“数据”模板中选择“LINQ to SQL 类”。从“服务器资源管理器”将所要添加的数据库表拖动到设计界面上
为项目添加视图	在 Views 文件夹下添加一个文件夹，用于容纳不同类型的视图。右键单击这个文件夹，选择“添加” “视图”。如果希望通过 Visual Studio 反射数据库表并生成强类型的视图，则应指定数据类型。根据特定用途选择 Create、Update、Details 或 List，其他用途选择 Empty
为项目添加控制器	右键单击 Controllers 文件夹，选择“添加” “控制器”
为视图添加 HTML 标签	在视图的.aspx 文件中，直接输入 HTML 标签，也可以使用 HTML 辅助方法

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 23 章

AJAX

学习目标

- 理解 AJAX 所解决的问题
- 理解 ASP.NET 对 AJAX 的支持
- 构建应用 AJAX 的网站
- 在必要时利用 AJAX 来改善用户体验

本章将介绍 AJAX，这可能是 ASP.NET 近些年引入的最有趣的特性。AJAX 代表“异步 JavaScript 和 XML” (Asynchronous JavaScript and XML)，它力图使尽可能多的网站具有全新的感观。

23.1 富 Internet 应用程序(RIA)

软件似乎是以这样一个趋势发展的：只要某种技术深入人心(组件与架构之间能够良好地衔接)，那么就需要重视最终用户的体验。应用程序技术层出不穷，而如今“富 Internet 应用程序” (Rich Internet Application, RIA) 备受瞩目。AJAX 是构建 RIA 的流行技术之一。(Microsoft Silverlight 也是其中的一种。)

AJAX 的出现是为了改进 Web 用户所熟知的标准 HTTP GET/POST 方式。也就是说，原来在客户端与服务器之间传输整个表单(页面)的标准 Web 协议正逐步得到扩充。

虽然标准的 HTTP 仍然可用，并且 Web 开发者相对熟悉，但它存在一些弊端——主要弊端之一是用户在刷新页面时需要等待较长的时间。对于所有事件驱动的界面，这都是常见问题。(Windows 操作系统中的应用程序就是事件驱动的。)AJAX 引入的技术能够避免最终用户等待“整个”页面的回发。

为理解 AJAX，需要回顾 HTTP 的典型工作方式。在用户发起请求后(如使用 GET 或 POST)，Web 浏览器会代表用户向服务器发送请求。在此期间，用户只能等待请求处理完毕。也就是说，用户发起请求后需要等待(看着浏览器下面的小进度条)。在请求返回浏览器后用户才能使用应用程序，而在请求返回之前，应用程序几乎是不能被使用的。在某些情况下，浏览器的窗口甚至还会变成空白。在这期间，Web 浏览器要等待网站完成 HTTP 请求的处理，这相当于基于 Windows 的应用程序必须等待消息处理程序完成其任务一样。(事实上，如果客户端浏览器支持多线程的用户界面，如 Windows Internet Explorer，那么用户通常可以取消请求，但也仅限于此)这种体验很容易模拟，只需要在 ASPX 页面的 Page_Load 方法

中调用 `System.Threading.Thread.Sleep`。通过让线程睡眠,最终用户便只能等待服务器(苏醒)完成请求处理。

对于这种问题, AJAX 的解决方案是通过某种方式来进行异步请求处理。网站为什么不在后台以异步方式处理请求,进而使浏览器对用户的反应速度更快?对于某些应用程序,可否在 HTTP 请求处理期间不阻塞整个浏览器,而是从表面上看请求是在后台运行,使前台不受阻塞,并且只更新要呈现的那一部分页面?这样,网站的观感对用户来说便更连续、更平滑。举另外一个例子, ASP.NET 为什么不在呈现的页面中包含一些能够生成客户端脚本的控件,而通过这些脚本来更新 HTML“文档对象模型”(Document Object Model, DOM),进而为用户提供更具交互性的表单?这正是 ASP.NET AJAX 出现的原因。

23.2 何为 AJAX

AJAX 引入了一种编程风格,旨在改善网站 UI 的响应速度和外观。AJAX 中的许多功能早已投入应用。AJAX 汇集了多种优秀的思想,并根据这些思想形成了一种编程风格,扩展了标准 HTTP(Internet 的基础协议)机制。与大多数 Web 应用程序开发环境一样, ASP.NET 也以非常标准的方式使用 HTTP 的功能。浏览器中的页面一般通过 HTTP GET 请求来建立与服务器最初的联系,而之后使用一系列 POST 指令。从更高层面来看整个机制,浏览器发送完整的请求,然后等待来自服务器的完整响应。虽然 ASP.NET 服务器端控件架构革命性地改进了后端编程方式,但用户仍然每次都需要通过下载整个页面来获取自己所要的信息。这种方式很像 20 世纪 70 年代和 80 年代早期的主机/终端模型,只不过终端换成了现代更为复杂的浏览器,而主机换成了 Web 服务器(或 Web 场)。

标准的 HTTP 往返交互方式是一种不错的应用程序策略, Web 一直都采用这种方式。20 世纪 90 年代晚期 Web 被开发出来后,出现了许多种浏览器,功能各不相同。从早期针对蜂窝电话和个人数字助理(PDA)的 America Online Browser 浏览器(功能十分有限)到 Internet Explorer 和 Netscape Navigator 这些更为复杂的浏览器(功能更为丰富)。例如, Internet Explorer 支持 JavaScript 和“动态 HTML”这样的高级功能。此时需要综合考虑网站的可用性和丰富性,因此在 ASP.NET 这样的技术出现之前,网站是很难构建的。

不过,大多数现代计算平台都支持能够处理客户端脚本的浏览器。如今,大多数计算环境都运行现代操作系统(如 Windows Vista、Windows 7 操作系统或 Macintosh OS X)。这些环境中的浏览器都完全支持 XML 和 JavaScript。由于如此之多的 Web 客户端平台都支持此功能,利用它也是理所应当的。正如本章稍后将要介绍的, AJAX 充分利用了现代浏览器的功能来改善用户体验。

除了扩展标准 HTTP 协议外, AJAX 还能够利用 Web 服务。Web 服务最初是为“企业到企业”的商业通信而产生的。然而, Web 服务也适合在更小的范围内使用,以“带外”(out of band)方式来处理 Web 请求(所谓“带外”,是指通过特殊方式来处理 HTTP 请求,而不使用标准的页面回发机制)。相比传统的 HTTP GET/POST 方式, AJAX 能够在后台借助 Web 服务来加快客户端 UI 的响应速度。本章将介绍这种工作机制,特别是在 2.3.9 节中介绍了

ASP.NET AJAX Control Toolkit 中的“扩展程序控件”(extender control)。

23.3 ASP.NET 与 AJAX

AJAX 在 Web 编程方面带来的主要变化之一是使浏览器更多地参与到整个过程中来。AJAX 包含一些新的客户端脚本库，使浏览器能够利用异步回调方式与服务器交互，而不仅仅显示 HTML 流并执行简单的自定义脚本块。AJAX 还包含一些基本的服务器端组件来支持来自客户端的异步调用。我们甚至还可以利用受社区支持的 AJAX Control Toolkit——一套基于 ASP.NET 的 AJAX 实现。图 23.1 展示了 ASP.NET AJAX 的组织方式。

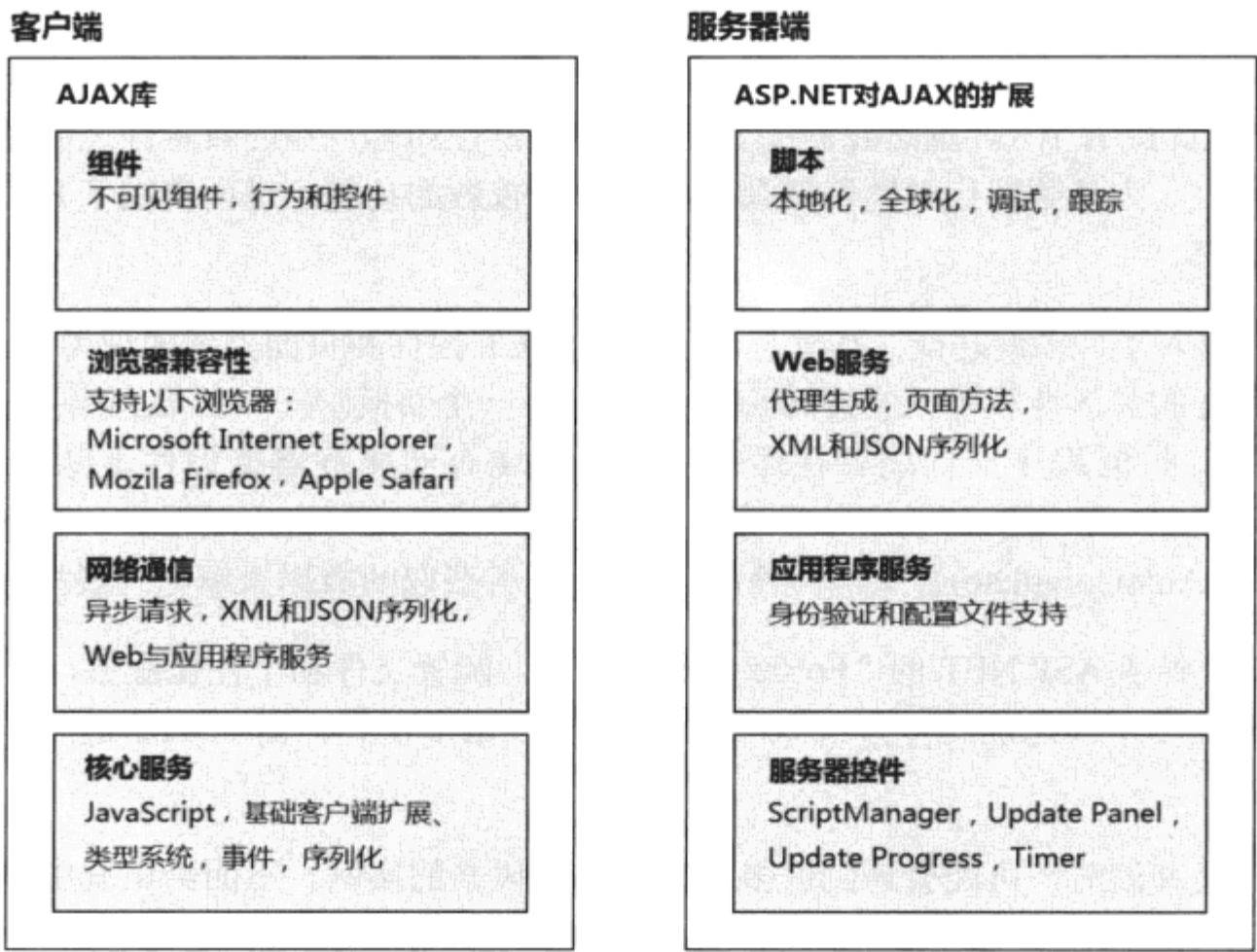


图 23.1 ASP.NET AJAX 的逻辑层次

23.3.1 使用 AJAX 的原因

如果传统的 ASP.NET 开发方式得到如此大的改进并被广泛接受，那么为什么还要使用 AJAX 呢？乍看起来，AJAX 似乎为 ASP.NET 编程增添了额外的复杂性。事实上，它似乎采用了一些 ASP.NET 曾经力图避免的方式(如频繁使用客户端脚本)。然而，AJAX 旨在为用户带来更丰富的体验。由于 ASP.NET 几乎能够无缝地支持 AJAX，因而所增加的复杂性在很大程度上被化解了。在构建网站时，可能有以下几方面原因促使某些 ASP.NET 网站支持 AJAX：

- 通过在浏览器中对网页的局部进行处理，AJAX 能够在总体上改进网站的效率。用户

不必等待通过 HTTP 协议获取完整的页面响应。开发者可以将页面的部分处理工作放在客户端，从而使客户端能够更快做出反应。当然，这种特性过去也存在——只要开发者愿意编写实现代码。ASP.NET 对 AJAX 的支持包括许多脚本，因此只需要借助几个服务器端控件，便可以发挥浏览器的诸多功能。

- ASP.NET AJAX 为网站带来了一些通常只在桌面应用程序中出现的 UI 元素，并且支持多种浏览器(虽然这需要用到许多浏览器脚本，但其中的大部分都已实现)。这些 UI 元素包括圆角矩形、标注、进度指示器、弹出窗口。
- AJAX 支持页面的局部更新。只更新网页的局部可以极大地缩短用户的等待时间。这带来了较高的 UI 性能，使基于 Web 的应用程序更加贴近桌面应用程序。
- 大多数流行的浏览器都支持 AJAX。除了 Internet Explorer, Mozilla Firefox 和 Apple Safari 也支持 AJAX。虽然我们还要在 UI 的丰富性和客户端的兼容性之间进行权衡，但事实上，大多数现代浏览器都具备 AJAX 所依赖的功能，进而消减了开发者对这方面的考虑。
- AJAX 引入了许多新功能。尽管标准的 ASP.NET 控件和页面呈现模型为 Web 开发提供了良好的灵活性和可扩展性，但 AJAX 引入了一个新概念——扩展程序控件(extender control)。扩展程序控件能够在运行时附加到现有的服务器端控件上(如 TextBox、ListBox 和 DropDownList)，并为这些控件提供新的外观和客户端行为。有些扩展程序控件(如 AutoComplete)甚至能够使用从 Web 服务获取的数据来填充列表框等控件。
- AJAX 改善了 ASP.NET 的“Forms 身份验证”、配置文件和个性化服务。ASP.NET 对身份验证和个性化的支持为 Web 开发者提供了诸多便利，而 AJAX 进一步增强了该功能。

如今，在网上冲浪时，可能会遇到许多 AJAX 编程风格的实例。下面给出其中的几个：

- Colorado Geographic: <http://www.coloradogeographic.com/>
- Cyber Homes: <http://www.cyberhomes.com/>
- Component Art: <http://www.componentart.com/>

23.3.2 现实中的 AJAX

从 20 世纪 90 年代到 21 世纪的最初几年，Web 应用程序几乎都遵循 20 世纪 70 年代的主机与终端架构。然而，与大型计算机为若干终端服务不同，Web 应用程序是由 Web 服务器(或 Web 场)和连接到它的浏览器构成，这些浏览器更为强大，支持非常复杂的呈现功能。直到近些年，Web 应用程序才采用一种从 HTTP 表单获取输入并以 HTML 网页形式表现输出的方式。为理解标准 Web 应用程序的实质，要先理解 HTTP 无连接的、无状态的性质。这也就导致了传统 Web 应用程序只能够表现应用程序状态的快照。

正如本章所介绍的，Microsoft 通过 ASP.NET 框架支持标准的 AJAX 习惯和模式。不过，AJAX 不是一种具体的技术，而是一种支持带外 HTTP 请求的 Web 编程风格。

我们经常会见到网站通过 AJAX 编程实现新的界面特性和样式。这些网站包括 Microsoft.com、Google.com 和 Yahoo.com。在浏览这些网站时，我们经常会发现一些特殊功能。例如，在不进行显式回发的前提下自动更新页面。再如，模式对话框能够在被主动关闭之前一直显示，以此来引起用户的注意。这些功能都可以通过 AJAX 风格的编程模式和支持 AJAX 的 ASP.NET 扩展(如 AJAX 服务器端控件和扩展程序)来实现。

如果读者熟悉 Microsoft 环境下的 Web 开发，则可能会对 AJAX 的价值产生怀疑，也可能思考是否能够通过像 DHTML 这种成熟的、具体的技术来实现相同功能。

23.3.3 AJAX 展望

面向 Internet Explorer 的、经验丰富的 Web 开发者一般都熟悉“动态 HTML”(DHTML)。DHTML 是一种能够在 Web 客户端环境(浏览器)中实现 Windows 桌面风格 UI 的技术。DHTML 建立了一个良好的开端，而 AJAX 将更贴近桌面的特性带到了 Web 应用程序中。

AJAX 提供的功能要远多于 DHTML。使用 DHTML，我们主要通过 JavaScript 修改 HTML 元素的样式，但也仅限于此了。DHTML 适合实现当鼠标悬停时打开一个菜单这样的 UI 特性。AJAX 通过 JavaScript 将客户端 UI 延伸到服务器的带外调用。由于 AJAX 基于带外服务器请求(而不只是依赖于大量客户端脚本代码)，相比 DHTML，AJAX 要具有更多潜在的发展空间。

23.4 ASP.NET 对 AJAX 的服务器端支持

ASP.NET 对 AJAX 的大部分支持位于一个服务器端控件集中，这些控件负责向浏览器呈现 AJAX 风格的输出。第 3 章介绍过，ASP.NET 应用程序的整个页面呈现过程被打散到粒度较小的单元中。每单元的呈现工作都由派生自 System.Web.UI.Control 的类来处理。服务器端控件的全部工作是将 HTML 元素呈现到输出流中，以便能够在浏览器中正确显示。例如，ListBox 控件会呈现<select/>标签，TextBox 控件会呈现<input type="text"/>标签。类似地，ASP.NET AJAX 服务器端控件会相应地向浏览器呈现 AJAX 风格的脚本和 HTML。

ASP.NET 对 AJAX 的支持由这些服务器端控件和实现 AJAX 行为的客户端脚本构成。下面几节将介绍最常见的几种面向 AJAX 的 ASP.NET 服务器控件：ScriptManager、ScriptManagerProxy、UpdatePanel、UpdateProgress 和 Timer。

23.4.1 ScriptManager 控件

ScriptManager 控件用于管理页面的脚本资源。该控件主要针对页面，负责注册 AJAX Library

脚本，以便客户端脚本能够利用扩展的类型系统。ScriptManager 还用于实现页面的局部呈现，并支持本地化和自定义用户脚本。ScriptManager 能够辅助对服务器进行带外回调。ASP.NET 网站中需要使用 AJAX 功能的页面必须包含 ScriptManager 控件的实例。

23.4.2 ScriptManagerProxy 控件

在服务器呈现网页中的脚本时，往往需要进行一些特殊的处理。通常，页面通过 ScriptManager 控件来组织页面层的脚本。对于内容页和用户控件这样的嵌套组件，如果宿主中已经存在 ScriptManager，则要通过 ScriptManagerProxy 来管理页面引用的脚本和服务。

使用母版页经常会遇到这种情况。ScriptManager 控件一般由母版页承载。然而，如果给定的内容页面也包含 ScriptManager 的实例，ASP.NET 则会抛出一个异常。如果内容页要访问母版页所包含的 ScriptManager，这时该怎么办呢？答案是在内容页中添加 ScriptManagerProxy 控件，而通过这个代理来间接地使用 ScriptManager 控件。当然，正如之前所谈到的，这种规则也同样适用于用户控件。

23.4.3 UpdatePanel 控件

UpdatePanel 控件能够实现页面的局部更新，它通过几种指定的服务器端控件和事件来触发这一过程。UpdatePanel 控件(在一般的 HTTP 回发过程中)只刷新页面的指定部分，而不刷新整个页面。

23.4.4 UpdateProgress 控件

在 UpdatePanel 控件进行局部更新时，UpdateProgress 控件能够反映相关状态信息。UpdateProgress 控件能够向用户即时反馈耗时的操作正在进行。

23.4.5 Timer 控件

Timer 控件能够以设定的间隔进行回发。虽然 Timer 控件触发的是一般的回发(回发整个页面)，但它与 UpdatePanel 控件结合便可以实现周期性的页面局部更新。

23.5 AJAX 客户端支持

ASP.NET AJAX 对客户端的支持是基于一组 JavaScript 库实现的。ASP.NET AJAX 的脚本模块包含以下几个层次。

- **浏览器兼容性层** 针对常见的浏览器负责管理兼容性。ASP.NET 本身在服务器端提供了浏览器的功能信息，而这个层次的目的是在客户端处理兼容性问题(支持的浏览器包括 Internet Explorer、Mozilla Firefox 和 Apple Safari)。
- **核心服务层** 通过引入 AJAX 编程中所要用到的类、命名空间、事件处理、数据类型和对象序列化扩展了一般的 JavaScript 环境。
- **针对客户端的 ASP.NET** AJAX 基类库包括不同的组件，比如用于字符串管理组件和用于扩展错误处理的组件。
- **网络层** 用于管理客户端与基于 Web 的服务和应用程序间的通信。网络层还能够处理异步远程方法调用。

受社区支持的 Control Toolkit 是基于 ASP.NET AJAX 的重要工具。虽然之前提到的功能为 ASP.NET AJAX 提供了坚实的基础架构，但假使没有丰富的工具集，AJAX 也就黯然失色了。

23.5.1 ASP.NET AJAX Control Toolkit 简介


ASP.NET AJAX Control Toolkit 是一套封装 AJAX 功能的组件(和相关示例)。通过示例，我们可以了解使用其中的控件和扩展程序能够实现怎样的用户体验。Control Toolkit 还为创建自定义的控件和扩展程序提供了功能强大的软件开发包。完整的 ASP.NET AJAX Control Toolkit 可以从 ASP.NET AJAX 网站上下载。

ASP.NET AJAX Control Toolkit 是独立下载的，而不会随 Microsoft Visual Studio 2010 的安装而自动安装。在本书截稿前，其最新版本是 4.1，发布于 2010 年 4 月中旬。详细内容，可以访问 <http://www.asp.net/ajaxlibrary/>。读者可以下载二进制代码和源代码。如果对源代码不感兴趣，则在项目中引用 AjaxControlToolkit.dll 程序集即可。

如果想自行生成这套工具集，请遵循以下步骤：

1. 下载该工具集的源代码。
2. 解压文件，并在 Visual Studio 中打开 AjaxControlToolkit 解决方案文件。
3. 生成 AjaxControlToolKit 项目。编译程序会在 AjaxControlToolkit\bin 目录下生成名为 AjaxControlToolkit.dll 的文件。
4. 左击 Visual Studio 的“工具箱”，从菜单中选择“选择项目”。找到 AjaxControlToolkit\bin 目录下的 AjaxControlToolkit.dll 文件，单击“确定”引用此 DLL。这样便可以在 Visual Studio 中引用 AJAX 控件，以便我们能够将其从工具箱拖放到应用程序的页面中。^①

① 译者注：Visual Studio 会自动为网站添加必要的引用。

 **提示** 我们可以通过社区获得许多支持 AJAX 的服务器端控件和客户端脚本。虽然这些不是官方发布的 Microsoft AJAX 内容，但这些内容包括 ASP.NET AJAX 社区支持的控件(如前所述)，以及对客户端声明式语法(XML 脚本)的支持等。要了解相关信息，请访问 <http://www.asp.net/ajax/>。该网站包含 ASP.NET AJAX Control Toolkit 的下载链接，以及有关 ASP.NET AJAX 的网站链接、视频教程和其他相关内容的下载。

23.5.2 AJAX Control Toolkit 中的组件

我们可以从社区获得许多额外的扩展程序和控件。AJAX Control Toolkit 可以从 <http://www.asp.net/ajax/> 获得下载链接。本章将介绍此工具集中的几个控件。表 23.1 列出了这个工具集中的控件和扩展程序。^①

表 23.1 ASP.NET Control Toolkit

组 件	说 明
Accordion	此扩展程序能够显示一组面板，每次显示其中的一个。其行为与使用多个 CollapsiblePanel 控件类似，只不过每次只显示一个面板。该扩展程序由一组 AccordionPane 控件组成
AlwaysVisibleControl	此扩展程序能够将控件固定在页面上，在后面的内容滚动时，此控件的位置保持不变
Animation	此扩展程序为向页面内容添加动画提供了一个简洁的接口
AsyncFileUpload	此控件用于在后台以异步方式上传文件
AutoComplete	此扩展程序能够与指定的 Web 服务进行通信，根据当前文本框输入的内容获取并列出可能的数据项
Calendar	此扩展程序能够以自定义的方式为 TextBox 控件提供客户端的日期选取功能
CascadingDropDown	此扩展程序针对 DropDownList 控件，能够自动填充一组相关的 DropDownList 控件
CollapsiblePanel	此扩展程序针对 Panel 控件，能够为页面提供可折叠的区域
ColorPicker*	此扩展程序针对 TextBox 控件，能够提供客户端的颜色选取功能。在用户单击文本框后，该扩展程序会显示一个用于选取颜色的调色板
ComboBox*	此控件模拟“Windows 窗体”应用程序中的同名控件，将 DropDownList 和 TextBox 结合，既能够选取某一项，也可以直接输入文本
ConfirmButton	此扩展程序针对 Button 控件(和派生自 Button 的控件)，用于向用户显示消息。可以在需要用户确认某些操作时使用它(例如，如果重定向到其他页面会使用户丢失数据，则应向用户进行确认)

① 译者注：所有带星号(*)的扩展程序/控件是本书截稿之前最新 AJAX Control Toolkit 版本引入的，原书中不存在。

续表

组 件	说 明
DragPanel	此扩展程序针对 Panel 控件，允许用户在页面上拖拽 Panel 控件
DropDown	此扩展程序实现了 Microsoft SharePoint 中的下拉菜单
DropShadow	此扩展程序针对 Panel 控件，能够为其添加阴影
DynamicPopulate	此扩展程序能够显示 Web 服务或页面方法调用返回的 HTML 字符串
FilteredTextBox	此扩展程序能够强制用户只在文本框中输入有效字符
HoverMenu	此扩展程序针对所有与弹出窗口关联的 WebControl，能够显示额外的内容。当鼠标指针悬停在目标控件上后，弹出窗口便会被激活
HTMLEditor*	此控件允许用户在客户端编辑和预览 HTML 文档。该控件会在自身的工具栏中显示用于编辑的按钮，并能够显示生成的 HTML
ListSearch	此扩展程序能够根据用户的输入在目标 ListBox 或 DropDownList 中搜索项目
MaskedEdit	此扩展程序针对 TextBox 控件，能够通过遮罩(mask)来约束 TextBox 中文本的格式
ModalPopup	此扩展程序能够模拟标准 Windows 模式对话框的行为。使用 ModalPopup，页面能够显示一个让人引起注意的弹出窗口，直到用户主动将其关闭
MultiHandleSlider*	此扩展程序针对 TextBox 和 Label，能够显示多个滑块，允许用户以图形方式在指定的范围内选取一个或多个值
MutuallyExclusiveCheckBox	此扩展程序针对 CheckBox 控件，能够根据键对多个 CheckBox 进行分组。如果多个 CheckBox 控件存在多个相同的键，此扩展程序能够确保只有其中的一个 CheckBox 被选中
NoBot	此控件能够提供 CAPTCHA ^① 式的自动程序/垃圾信息监测与防范，而不需要与用户进行交互。这种不进行人机交互的方式要远比进行人机交互的方式方便，而且对用户是完全透明的
NumericUpDown	此扩展程序针对 TextBox 控件，能够使其具有类似标准 Windows 中带有 Spin 按钮的 Edit 控件的行为。这个扩展程序会添加“向上”和“向下”按钮来调整 TextBox 中的值
PagingBulletedList	这个扩展程序针对 BulletedList 控件，它能够在客户端上实现根据类别进行分页
PasswordStrength	此扩展程序针对 TextBox 控件，能够在用户输入密码时提供帮助。一般的 TextBox 只能够隐藏实际的文本，而 PasswordStrength 扩展程序能够以可视方式显示密码的强度
Popup	这个扩展程序针对所有控件，能够通过弹出窗口来显示相关内容

① 译者注：CAPTCHA 代表“全自动区分计算机和人类的测试”(Completely Automated Public Turing test to tell Computers and Humans Apart)。

续表

组 件	说 明
Rating	此控件能够显示等级刻度，允许最终用户通过代表其意图的图标(最常见的是五角星)来对事物进行分级
ReorderList	此 ASP.NET AJAX 控件能够显示一个带项目标号的、支持数据绑定的列表，允许用户调整其中项目的顺序
Resizable	此扩展程序适用于任何网页元素。在 Resizable 与某个元素关联后，用户便可以调整该控件的大小。Resizable 会在这个控件的右下角添加一个手柄
RoundedCorners	RoundedCorners 扩展程序适用于任何网页元素，能够使矩形框呈现圆角
Seadragon	此控件能够为图片添加启用缩放、全屏和拖动功能的按钮
Slider	此扩展程序针对 TextBox 控件，能够为其添加滑块，用户可以通过它来修改 TextBox 中的值
SlideShow	此扩展程序控件能够使用户通过按钮来切换图片，并以幻灯片的形式自动播放图片
TabContainer 和 TabPanel	这组服务器控件能够通过选项卡式的面板来管理页面上的内容
TextBoxWatermark	此扩展程序针对 TextBox 控件，能够在该控件为空的时候显示提示文本。如果 TextBox 包含文本，那么该控件则表现一般行为
ToggleButton	此扩展程序允许通过自定义的图片来反映 CheckBox 的状态
UpdatePanelAnimation	此扩展程序为向 UpdatePanel 添加动画提供了简洁的接口
ValidatorCallout	此扩展程序针对验证控件(如 RequiredFieldValidator 和 RangeValidator)，能够针对包含错误数据的 UI 元素显示弹出式窗口，以引起用户的注意

23.6 AJAX 入门

下面的练习将使您进一步认识 AJAX。我们将构建一个非常简单的“Web 窗体”应用程序，通过服务器端控件 UpdatePanel 在后台更新页面的内容。在这个练习中，我们将创建一个带有标签的页面，标签能够显示页面加载的日期和时间。一个标签放置在 UpdatePanel 之外，另一个标签在 UpdatePanel 之内。通过比较两个标签所显示的日期和时间，可以直观地体现页面的局部更新机制。

➤ 实现简单局部页面更新

- 单击“文件”|“新建”|“项目”，创建一个“ASP.NET 空 Web 应用程序”，将其命名为 AJAXORama。Visual Studio 会创建一个支持 AJAX 的网站。在这个网站中添加一个名为 Default.aspx 的页面，确保其已被打开。
- 从“工具箱”拖放一个 ScriptManager 控件到刚刚添加的页面上。(该控件在“工具箱”的“AJAX Extensions”选项卡中。)ScriptManager 必须出现在所有其他 AJAX 控件之前。根据惯例，此控件通常放置在 Visual Studio 创建的 DIV 之外。在页面上添加脚本管理器控件后，此时“源”视图中的<body>元素如下所示：

```
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <div>

        </div>
    </form>
</body>
```

3. 拖放一个 Label 控件到 Default.aspx 页面上。在“属性”窗口中，将这个 Label 的名称设置为 LabelDateTimeOfPageLoad。再拖放一个 Button 到表单上，将其文本设置为“Click Me”。打开代码旁置文件(Default.aspx.cs)，更新 Page_Load 处理程序，使标签显示当前的日期和时间：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Default: System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.LabelDateTimeOfPageLoad.Text = DateTime.Now.ToString();
    }
}
```

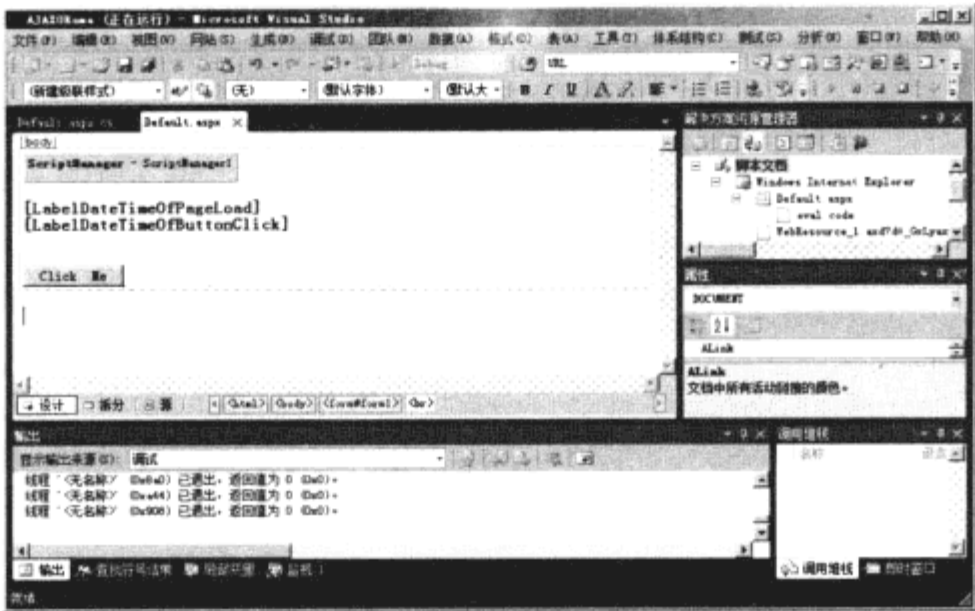
4. 运行这个页面，通过单击按钮来触发几次回发。我们会发现，页面上的标签会被更新为每次单击按钮时的日期和时间。
5. 在页面上添加一个 UpdatePanel 控件。(该控件和 ScriptManager 控件都位于 Visual Studio “工具箱”的“AJAX Extensions”选项卡中。)然后，从“工具箱”拖放一个 Label 到 UpdatePanel 的内容区域，并将这个标签命名为 LabelDateTimeOfButtonClick。
6. 为 Page_Load 方法添加代码，使刚添加的标签显示当前日期和时间：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class Default: System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        this.LabelDateTimeOfPageLoad.Text = DateTime.Now.ToString();
    }
}
```

```
        this.LabelDateTimeOfButtonClick.Text = DateTime.Now.ToString();  
    }  
}
```

下图展示了 Visual Studio “设计器”中的 UpdatePanel、Button 和 Label(这几个控件之间插有换行符，以使页面可读性更好)：



- 7. 运行这个页面，重复单击按钮来触发几次回发。每次回发，标签都会显示回发时的日期和时间(两者显示的时间相同)。虽然第二个标签位于 UpdatePanel 的内部，但导致回发的操作发生在 UpdatePanel 之外。

下图展示了 Button 与 UpdatePanel 关联之前的页面效果：



- 8. 回到 Visual Studio，从页面中删除按钮和 LabelDateTimeOfButtonClick 标签，并拖放一个新的按钮到 UpdatePanel1 控件中。再向 UpdatePanel1 添加一个 Label 控件，并将其命名为 LabelDateTimeOfButtonPress。此时，Visual Studio 会生成如下代码：

```
<%@ Page Language="C#" AutoEventWireup="true"  
CodeBehind="Default.aspx.cs" Inherits="AJAXORama.Default"%>  
  
<!DOCTYPE html PUBLIC  
"-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head runat="server">  
    <title></title>  
</head>  
<body>  
    <form id="form1" runat="server">
```



```
<asp:ScriptManager
  ID="ScriptManager1" runat="server"/><br/>
<asp:Label ID="LabelDateTimeOfPageLoad"
  runat="server"></asp:Label><br/>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">

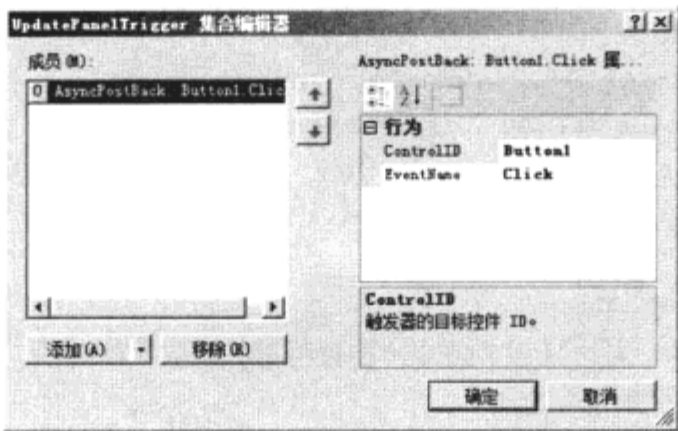
  <ContentTemplate>
    <asp:Label ID="LabelDateTimeOfButtonPress"
      runat="server">
    </asp:Label><br/>
    <asp:Button ID="Button1"
      runat="server" Text="Click Me"/>
  </ContentTemplate>
</asp:UpdatePanel>
</form>
</body>
</html>
```

新添加的 Button 和第二个 Label 都应嵌在 UpdatePanel 内部。

9. 运行这个页面，单击按钮以触发回发。此时，我们会发现，只有 UpdatePanel 中显示日期和时间的标签被更新。这就是所谓的局部页面更新——页面操作(如按钮的单击)实际只导致页面的一部分被更新。局部页面更新有时被称为“回调”(callback)，而不是“回发”(postback)。下图展示了这个页面，其中的 Button 与 UpdatePanel 存在关联。



10. 添加一个 UpdatePanel 触发器。由于按钮和第二个标签都与一个 UpdatePanel 关联，因而在按钮触发回发后，只有第二个标签会被更新。如果只能通过内部的控件来触发对 UpdatePanel 的更新，这样未免显得过于局限。正如这里将要展示的，UpdatePanel 能够通过一组触发器来触发局部页面的呈现。为演示这种特性，我们先将按钮移到 UpdatePanel 之外(这样，按钮便会触发完整的、常规的回发)——最简单的方法是在表单上拖动按钮，确保其位于 UpdatePanel 之外。
- 由于按钮处于 UpdatePanel 之外，因此按钮触发的回发不再只更新第二个标签，因而也不会引起第 9 步展示的局部页面更新行为。
11. 更新 UpdatePanel 的 Triggers 集合，使其中包含按钮的 Click 事件。在“设计器”中，选择 UpdatePanel。在“属性”窗口中，选择“Triggers”。
12. 添加一个触发器，将 ControlID 设置为按钮的 ID，将 EventName 设置为 Click(如下图所示)。(每个属性都可以通过下拉列表提供的选项来方便地进行选择。)



运行这个页面。单击按钮会触发回发，而且最关键的是这仍然是局部页面回发。第一个标签仍显示页面最初加载的日期和时间，而第二个标签展示的是按钮被单击的日期和时间。这很说明问题！

异步回调

正如刚刚介绍的，标准的网页要求浏览器来触发回发。在多数情况下，(用 ASP.NET 的术语来说)回发由 Button 控件触发。不过，大多数 ASP.NET 控件其实都能触发回发。例如，如果希望在用户选中 DropDownList 中的某一项时进行回发，则只需要将其 AutoPostBack 属性设置为 true。这样，该控件便会在选定项发生变化时进行常规的回发。完整的回发有时由事件引发(如选定项发生变化)。然而，在大多数情况下，产生回发会分散用户的注意力，并且导致页面性能下降。这是因为标准的回发会刷新整个页面。

ASP.NET AJAX 通过运行客户端页面中的 JavaScript 来实现异步回发(asynchronous postback)。XMLHttpRequest 对象^①能够将数据投递到服务器——在后台执行完整的回发。服务器以 XML、JSON 或 HTML 的形式返回数据，并且只需要刷新局部页面。运行在页面中的 JavaScript 能够根据异步回发的结果，将“文档对象模型”(DOM)中的原始 HTML 替换为新的。

如果从事过客户端脚本编程，则不难想象这有多少工作要做。进行异步回发并更新页面通常需要许多 JavaScript 代码。

之前练习中使用的 UpdatePanel 控件隐藏了所有客户端代码和服务端端的处理。同样也是源于 ASP.NET 中服务器端控件架构的出色设计，UpdatePanel 采用的也是人们所熟知的服务器端控件模型。

23.7 定 时 器

除了根据控件生成的事件来产生局部页面更新外，AJAX 还包含一个定时器(Timer)，能够以固定的间隔引发事件。Timer 控件和其他 AJAX 标准控件都位于“工具箱”中。为自动

^① 译者注：这是浏览器(客户端)提供的对象。在其他浏览器中，实现相同功能的对象可能具有不同的名称，但 ASP.NET AJAX 已实现了这种兼容性，不需要开发者介入。

生成对服务器的回发，只需要将 Timer 拖放到页面上即可。

Timer 可以实现留言板或聊天室之类的功能——用户可以发送消息，这些消息显示在页面顶部，就像对话一样。如果要更新网络相机的图像或其他频繁更新的内容，都可以使用自动回发功能。

Timer 的使用非常容易——只需要将它拖放到包含 ScriptManager 的页面上。在默认情况下，Timer 每分钟(60000 毫秒)产生一次回发。Timer 默认是启用的，在页面加载后便开始计时。

下面的练习将演示如何使用 Timer 来写一个简单的聊天页面，使它显示多个登录用户发送的消息。在用户输入消息后会立即更新对话。但如果消息在发出后不刷新页面，用户是无法看到所发消息的(除非主动刷新页面)。这个页面将通过 Timer 来自动更新对话。首先，整个页面会被刷新。然后，聊天页面通过 UpdatePanel 来更新消息记录(频繁变化的信息)。

➤ 通过 Timer 来实现聊天页面

1. 如果 AJAXORama 应用程序未被打开，则打开它。首先要建立一个能够供多个会话访问的聊天消息列表。为添加全局应用程序类，在“解决方案资源管理器”的项目节点上右击，选择“添加”|“新建项”。选择“全局应用程序类”，然后单击“确定”。Visual Studio 会在网站中分别添加 Global.asax 和 Global.asax.cs 文件。
2. 更新 Global.asax.cs 中的 Application_Start 方法，以创建存储消息的列表，然后将这个列表添加到应用程序缓存中。

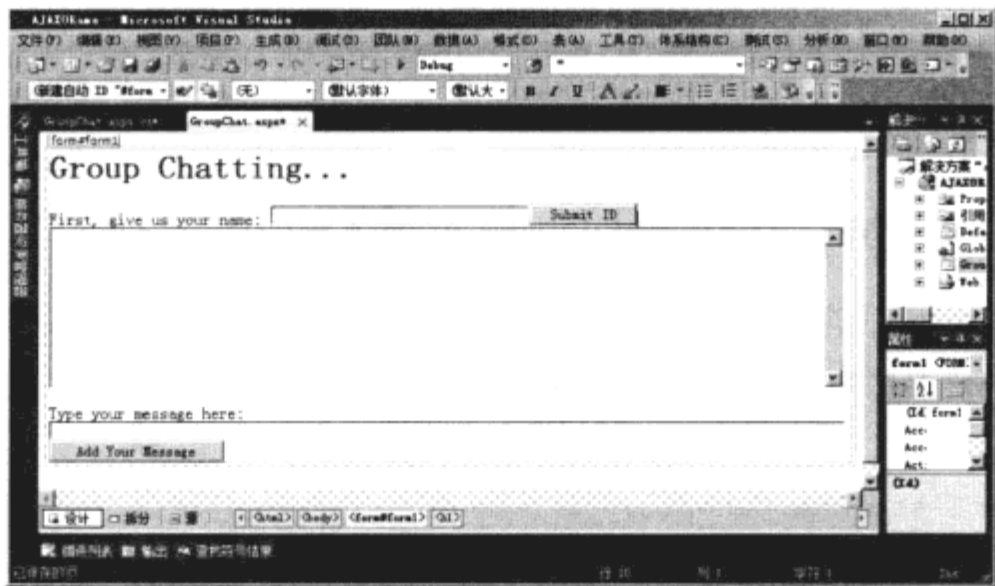
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.SessionState;

namespace AJAXORama
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            // Code that runs on application startup
            List<string> messages = new List<string>();
            HttpContext.Current.Cache["Messages"] = messages;
        }
        // 其他自动生成的代码...
    }
}
```

3. 在网站中添加一个页面，将其命名为 GroupChat.aspx。该页面应包含一个文本框来显示累积的消息，用户可以通过另一个文本框来添加消息。
4. 当消息传入后，应显示发送消息的人。这个页面会要求用户输入自己的身份，然后才

允许其添加消息。首先，添加一个 `ScriptManager`，并随后输入文本“Group Chatting...”。将这段文本标注为 `<h1>` 标题，以便使其单独显示为一行。然后，输入文本“First, give us your name:”。从“工具箱”拖放一个 `TextBox` 控件到页面上。将这个文本框的 ID 设置为 `TextBoxUserID`。在页面上添加一个按钮，以使用户能够提交其姓名。将这个按钮的文本设置为“Submit ID”，将其 ID 设置为 `ButtonSubmitID`。

5. 在页面上再添加一个 `TextBox`。这个文本框用于显示消息，因而应将其放大(800 个像素宽，150 个像素高足以)。将这个文本框的 `TextMode` 属性设置为 `MultiLine`，将 `ReadOnly` 属性设置为“True”，并将其 ID 设置为 `TextBoxConversation`。
6. 为页面添加最后一个 `TextBox`。这个文本框用于输入用户的消息。将其 ID 设置为 `TextBoxMessage`。
7. 为页面添加一个 `Button` 控件来提交当前消息。将其文本设置为“Add Your Message”，将其 ID 设置为 `ButtonAddYourMessage`。下图展示了这些控件的布局。



8. 打开代码旁置文件 `GroupChat.aspx.cs`。添加一个用于从会话状态获取用户名的方法。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace AJAXORama
{
    public partial class GroupChat : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected string GetUserID()
        {
            string strUserID =
                (string)Session["UserID"];
        }
    }
}
```

```

        return strUserID;
    }
}

```

9. 添加一个用于管理 UI 的方法，仅当用户提交身份信息后才允许其输入消息。如果用户未提交身份信息(即相应的会话变量不存在)，则禁用聊天 UI 元素，而只启用用户身份 UI 元素。如果用户提交了其身份，则启用聊天 UI 元素，并禁用用户身份 UI 元素：

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace AJAXORama
{
    public partial class GroupChat : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // 这里是其他代码...
            void ManageUI()
            {
                if (GetUserID() == null)
                {
                    // if this is the first request, then get the user's ID
                    TextBoxMessage.Enabled = false;
                    TextBoxConversation.Enabled = false;
                    ButtonAddYourMessage.Enabled = false;

                    ButtonSubmitID.Enabled = true;
                    TextBoxUserID.Enabled = true;
                }
                else
                {
                    // if this is the first request, then get the user's ID
                    TextBoxMessage.Enabled = true;
                    TextBoxConversation.Enabled = true;
                    ButtonAddYourMessage.Enabled = true;

                    ButtonSubmitID.Enabled = false;
                    TextBoxUserID.Enabled = false;
                }
            }
        }
    }
}

```

10. 为触发存储用户身份信息的按钮(ButtonSubmitID)添加一个 Click 事件的处理程序。这个事件处理程序要将用户的身份信息存储在会话状态中，并调用 ManageUI 来启用或

禁用相关控件：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace AJAXORama
{
    public partial class GroupChat : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // 这里是其他代码...
        }
        protected void ButtonSubmitID_Click(object sender, EventArgs e)
        {
            Session["UserID"] = TextBoxUserID.Text;
            ManageUI();
        }
    }
}
```

11. 为页面添加一个刷新对话记录的方法。该方法要查询应用程序缓存中的消息列表，并根据按时间倒序排列(最新消息在最上端)的消息列表生成一个字符串。然后，该方法应将该字符串赋给 TextBoxConversation 的 Text 属性(也就是显示对话记录的文本框的 Text 属性)：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace AJAXORama
{
    public partial class GroupChat : System.Web.UI.Page
    {
        // 这里是其他代码...
        void RefreshConversation()
        {
            List<string> messages = (List<string>)Cache["Messages"];
            if (messages != null)
            {
                string strConversation = "";
                int nMessages = messages.Count;

                for (int i = nMessages - 1; i >= 0; i--)
                {
                    strConversation += messages[i] + "<br>";
                }
            }
        }
    }
}
```



```

        {
            string s;

            s = messages[i];
            strConversation += s;
            strConversation += "\r\n";
        }

        TextBoxConversation.Text =
            strConversation;
    }
}

```

12. 双击表单底部名为 **ButtonAddYourMessage** 的按钮,为其添加一个 Click 事件处理程序,以便提交并添加用户的消息。此方法要从消息文本框 **TextBoxMessage** 来获取文本,在消息前插入用户的 ID,并将结果追加到应用程序缓存中的消息列表中。然后,该方法应调用 **RefreshConversation** 方法,以确保新的消息能够显示在对话记录文本框中:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace AJAXORama
{
    public partial class GroupChat : System.Web.UI.Page
    {
        // 这里是其他代码...
        protected void ButtonAddYourMessage_Click(object sender, EventArgs e)
        {
            // Add the message to the conversation...
            if (this.TextBoxMessage.Text.Length > 0)
            {
                List<string> messages = (List<string>)Cache["Messages"];
                if (messages != null)
                {
                    TextBoxConversation.Text = "";

                    string strUserID = GetUserID();

                    if (strUserID != null)
                    {
                        messages.Add(strUserID +
                                    ": " +
                                    TextBoxMessage.Text);
                        RefreshConversation();
                    }
                }
            }
        }
    }
}

```

```

        TextBoxMessage.Text = "";
    }
}
}
}
}

```

13. 更新 Page_Load 方法，使其调用 ManageUI 和 RefreshConversation:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

using System.Xml.Linq;
using System.Collections.Generic;

public partial class GroupChat : System.Web.UI.Page
{
    // 这里是其他代码...
    protected void Page_Load(object sender, EventArgs e)
    {
        ManageUI();
        RefreshConversation();
    }
}

```

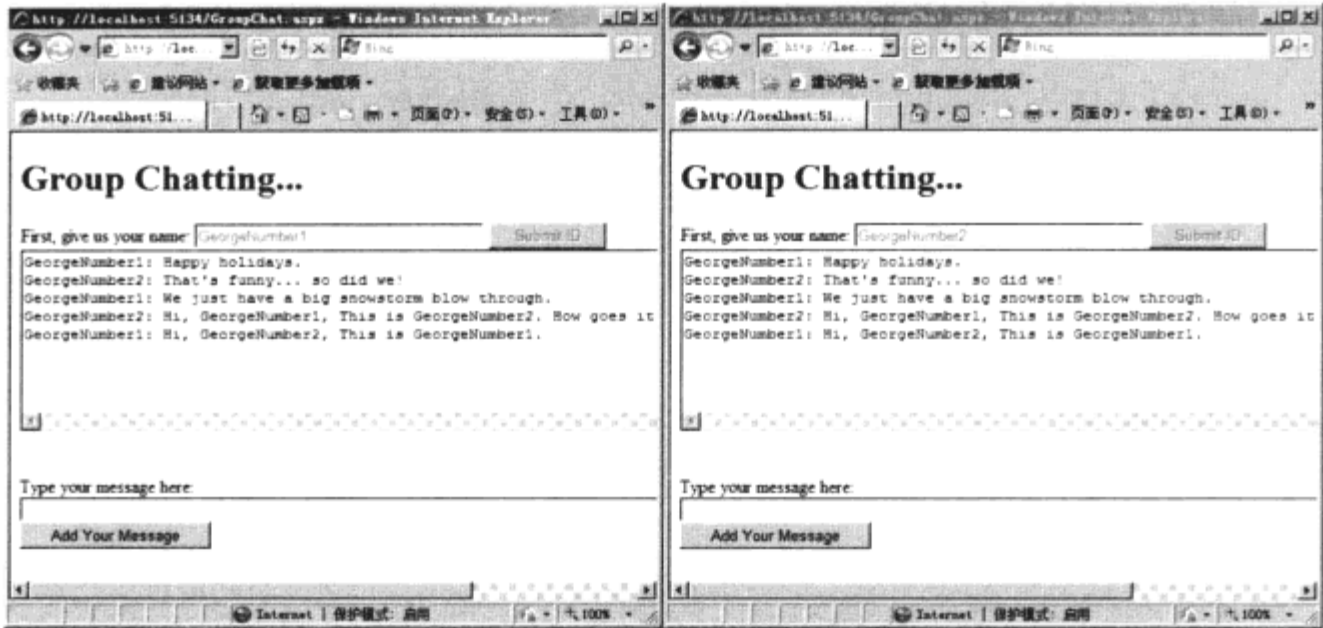
14. 运行页面，看看它的效果如何。用户在表明自己的身份后，便可以输入消息(这些消息会最终显示在对话文本框中)。尝试通过两个不同的浏览器实例来浏览这个页面。发现问题了吗？在一个用户发送消息后，他能够立即在自己的对话记录中看到这条消息。然而，对话中的另一个用户在提交自己的消息之前却无法看到其他任何新消息。为解决这个问题，我们要在页面中添加 AJAX Timer 控件。

15. 从“工具箱”中分别拖放一个 ScriptManager 和一个 Timer 控件到页面上。虽然 AJAX Timer 能够自动进行回发，但默认间隔是 60000 毫秒(1 分钟)。将 Timer 的 Interval 属性设置为一个更为合理的值(如 10000 毫秒，也就是 10 秒)。

再次用两个浏览器实例运行这个页面，看看会发生什么。此时，页面应每隔 10 秒自动回发一次。然而，这里仍存在一个问题。如果仔细观察则会发现，即使用户名没有任何变化，整个页面也会被刷新。在聊天过程中，用户其实只关心对话记录文本框是否被更新。这个问题可以通过 UpdatePanel 来解决。

16. 从“工具箱”中拖放一个 UpdatePanel 到页面上。将 UpdatePanel 放置到可以包容对话记录文本框的位置。将对话记录文本框移动到 UpdatePanel 中。为 UpdatePanel 添加一个针对 Timer.Tick 事件的触发器。再次运行这个聊天页面。此时，只有对话记录文本

框会在计时器激发时被更新。下图展示了使用 UpdatePanel 后的页面。



如果需要向服务器定期进行回发，则可以使用 ASP.NET AJAX Timer 控件。它尤其适合与 UpdatePanel 结合来实现局部页面更新。

23.8 进度的更新

UI 编程的惯例之一是使用户了解程序当前的执行进度。对于“Windows 窗体”应用程序，我们可以使用 BackgroundWorker 组件，并通过 Progress 控件来显示最新的进度。而 Web 编程则要采用不同的策略。ASP.NET AJAX 包含一个相关的组件——UpdateProgress 控件。

UpdateProgress 控件能够在异步回发执行期间显示。在 UpdatePanel 控件触发异步回发后，页面上的所有 UpdateProgress 控件就会变为可见。^①

下面的练习演示了 UpdateProgress 控件的使用。

➤ 使用 UpdateProgress 控件

1. 在 AJAXORama 网站中添加一个新页面，将其命名为 UseUpdateProgressControl.aspx。
2. 从“工具箱”拖放一个 ScriptManager 控件到页面上。
3. 拖放一个 UpdatePanel 控件到页面上，将其 ID 设置为 UpdatePanelForProgress，以便稍后在程序中使用它。在其内部添加一行文本：“This is from the update panel”。在这个 UpdatePanel 中添加一个 Button 控件，用它来触发一个运行时间较长的操作。将这个按钮的 ID 设置为 ButtonLongOperation，将其文本设置为“Activate Long Operation”。
4. 为这个按钮添加一个 Click 事件处理程序。模拟耗时操作最简单的方法是暂停几秒线

^① 译者注：UpdateProgress 控件只能反映当前是否处于更新状态，而不能反映真正意义上的进度。如果要想实现百分比形式的进度条，则必须引入其他机制。

程的执行(如下所示)。有了耗时的操作, 我们便可以测试 UpdateProgress 控件, 并在请求处理期间观察该控件的变化。

```
public partial class UseUpdateProgressControl : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

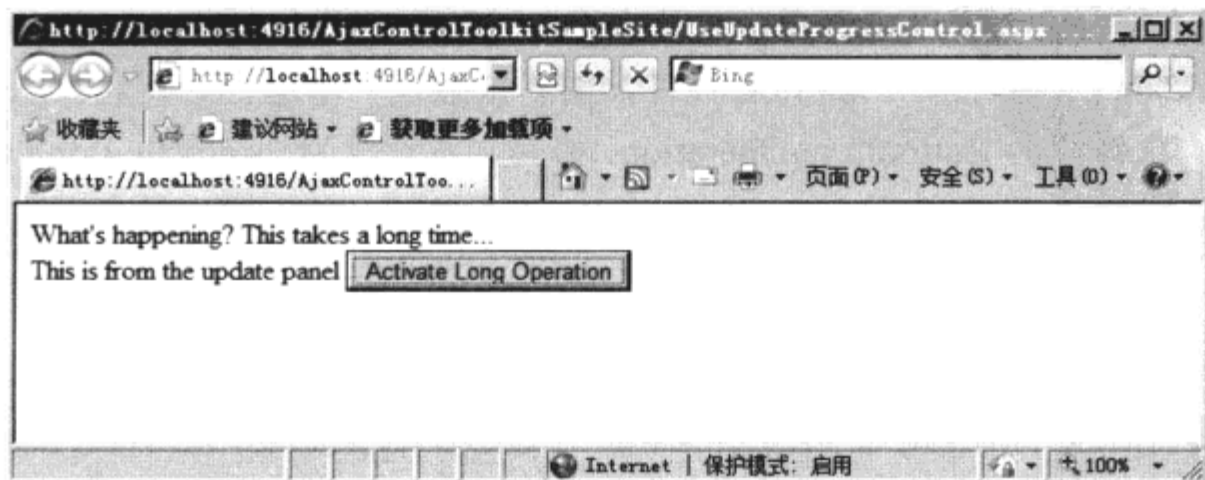
    }

    protected void ButtonLongOperation_Click(object sender, EventArgs e)
    {
        // Put thread to sleep for five seconds
        System.Threading.Thread.Sleep(5000);
    }
}
```

5. 在页面上添加一个 UpdateProgress 控件。UpdateProgress 控件必须与某个 UpdatePanel 相关联。为此, 要将这个 UpdateProgress 控件的 AssociatedUpdatePanelID 属性设置为刚刚添加的面板 UpdatePanelForProgress(使用下拉列表便可以选择这个 ID)。此外, 将 DisplayAfter 的值设置为 100(这意味着, 进度控件在刷新开始 100 毫秒后显示进度)。
6. 为 UpdateProgress 控件添加 ProgressTemplate 模板来容纳更新过程中显示的内容。在 ProgressTemplate 模板中添加一个 Label 控件, 以方便我们观察页面的变化。

```
<asp:UpdateProgress ID="UpdateProgress1"
    runat="server"
    AssociatedUpdatePanelID="UpdatePanelForProgress"
    DisplayAfter="100">
    <ProgressTemplate>
        <asp:Label ID="Label1" runat="server"
            Text="What's happening? This takes a long time...">
        </asp:Label>
    </ProgressTemplate>
</asp:UpdateProgress>
```

7. 运行这个页面, 看看其效果。在单击触发耗时操作的按钮后, UpdateProgress 控件会自动显示其内容。下图展示了被激活的 UpdateProgress 控件。



8. 如果不能撤销耗时的操作，那么异步进度显示功能也显得不完整。撤销耗时操作是可以实现的，为此只需要在页面中添加几行 JavaScript 代码。不过这段代码需要手动添加，因为没有这方面的设计时支持。编写一段客户端脚本块，并将其置于页面的顶部(<head>标签内)。这段脚本应获取 `Sys.WebForms.PageRequestManager` 的实例。在 ASP.NET AJAX 服务器端控件向客户端生成脚本时，会自动包含 `PageRequestManager` 类。`PageRequestManager` 类有一个名为 `get_isInAsyncPostBack()` 的方法，我们可以用它来判断页面当前是否正在执行异步回调。如果页面正在进行异步回调，则可以使用 `PageRequestManager` 的 `abortPostBack()` 方法来撤销请求。在 `ProgressTemplate` 中添加一个 `Button` 控件，设置其 `OnClientClick` 属性，以便调用新添加的 `abortAsyncPostBack` 函数。除了在 `OnClientClick` 属性中调用撤销方法外，还应紧随对撤销方法的调用后添加“`return false;`”(如下所示)。(插入“`return false;`”能够防止浏览器在调用此函数后进行回发。)

```
<%@ Page Language="C#"
    AutoEventWireup="true"
    CodeFile="UseUpdateProgressControl.aspx.cs"
    Inherits="UseUpdateProgressControl"%>

<!DOCTYPE html PUBLIC
    "...">
<head runat="server">
    <title></title>

    <script type="text/javascript">
        function abortAsyncPostBack()
        {
            var obj =
                Sys.WebForms.PageRequestManager.getInstance();
            if (obj.get_isInAsyncPostBack())
            {
                obj.abortPostBack();
            }
        }
    </script>

</head>
<body>
    <form id="form1" runat="server">
        <div>

            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>

        </div>
        <asp:UpdateProgress ID="UpdateProgress1"
            runat="server"
            AssociatedUpdatePanelID="UpdatePanelForProgress"
            DisplayAfter="100">
            <ProgressTemplate>
```

```
<asp:Label ID="Label1" runat="server"
    Text="What's happening? This takes a long time...">
</asp:Label>
<asp:Button ID="Cancel" runat="server"
    OnClientClick="abortAsyncPostBack(); return false;"
    Text="Cancel"/>
</ProgressTemplate>
</asp:UpdateProgress>
<asp:UpdatePanel ID="UpdatePanelForProgress" runat="server">
    <ContentTemplate>
        This is from the update panel
        <asp:Button ID="ButtonLongOperation"
            runat="server"
            OnClick="ButtonLongOperation_Click"
            Text="Activate Long Operation"/>
    </ContentTemplate>
</asp:UpdatePanel>

</form>
</body>
</html>
```

撤销注意事项

正如刚刚所看到的，撤销异步回发完全是客户端行为。在客户端撤销耗时操作相当于断开客户端与服务器的连接。一旦客户端与服务器间的连接被断开，客户端将无法收到来自服务器的响应。

此外，虽然客户端撤销了操作，但服务器可能无法得知客户端的撤销行为。因此，应谨慎处理耗时的操作，避免由于阻塞而造成失去对程序的控制。虽然 Microsoft Internet 信息服务(IIS)6 和 IIS 7 能够通过最终刷新应用程序池来清理失控的线程，但最好通过良好的编程实践来确保耗时操作能够被合理地、优雅地终止。

ASP.NET AJAX 为管理页面的局部更新和引入其他事件(如定时)提供了强大的基础设施。下一节将介绍 ASP.NET AJAX 扩展程序控件。

23.9 扩展程序控件

UpdatePanel 为更新页面的局部提供了一种方法。这非常有用，但 AJAX 的强大功能并不止于此。扩展程序(extender)控件架构是另一种常见功能。

扩展程序控件旨在扩展现有控件的功能。ScriptManager 和 Timer 这样的控件只会在页面中生成脚本代码，而扩展程序控件一般用于管理最终生成的页面中的标记(HTML)。

下面几小节将讨论几种 ASP.NET AJAX 扩展程序。首先，让我们来认识一下 AutoComplete 扩展程序。

23.9.1 AutoComplete 扩展程序

AutoComplete(自动补全)扩展程序要与标准的 ASP.NET TextBox 控件进行关联。在最终用户向 TextBox 输入文本的过程中，AutoComplete 扩展程序会通过 Web 服务来查询，并根据 Web 服务调用的结果向用户提示候选项。下面的练习借用了第 15 章中的一个组件——包含名人名言的 QuotesCollection 类。

➤ 使用 AutoComplete 扩展程序。

1. 在 AJAXORama 项目中添加一个新页面。由于该页面是为了演示 AutoComplete 扩展程序，因而将这个页面命名为 UseAutocompleteExtender。
2. 在这个页面中添加一个 ScriptManager 控件的实例。
3. 从第 15 章复制 QuotesCollection 类到当前项目中。回忆一下，这个类派生自 System.Data.Table 类，包含一些名言及出处。为添加这个组件到 AJAXORama 项目，右键单击项目节点，选择“添加”|“现有项”，找到第 15 章 UseDataCaching 项目中的 QuotesCollection.cs 文件并添加之。
4. 添加一个根据名字获取名言的方法。该方法应以字符串参数的形式接受作者的名字。为获取指定的名言，可以使用 System.Data.DataView 类，该类能够对内存中的表进行查询。该方法应以字符串列表的形式返回名言。根据所查询的名字，结果可能是空、有一个元素或有多个元素。我们稍后会用到这个方法。

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Collections.Generic;

///<summary>
/// Summary description for QuotesCollection
///</summary>
public class QuotesCollection : DataTable
{
    public QuotesCollection()
    {
    }

    public void Synthesize()
    {
        this.TableName = "Quotations";
    }
}
```

```

        DataRow dr;

        Columns.Add(new DataColumn("Quote", typeof(string)));
        Columns.Add(new DataColumn("OriginatorLastName", typeof(string)));
        Columns.Add(new DataColumn("OriginatorFirstName", typeof(string)));

        dr = this.NewRow();
        dr[0] = "Imagination is more important than knowledge.";
        dr[1] = "Einstein";

        dr[2] = "Albert";
        Rows.Add(dr);

        // 这里添加了其他名言...
    }

    public string[]
    GetQuotesByLastName(string strLastName)
    {
        List<string> list = new List<string>();

        DataView dvQuotes = new DataView(this);
        string strFilter = String.Format("OriginatorLastName = '{0}'", strLastName);
        dvQuotes.RowFilter = strFilter;

        foreach (DataRowView drv in dvQuotes)
        {
            string strQuote =
                drv["Quote"].ToString();

            list.Add(strQuote);
        }

        return list.ToArray();
    }
}

```

5. 在项目中添加一个名为 **QuotesManager** 的类来管理缓存。第 15 章演示缓存的示例在 **Page_Load** 事件执行期间存储和获取 **QuotesCollection**。这里，由于 **QuotesCollection** 要在 **Web** 服务中使用，因而需要选择其他时机进行缓存。为此，我们添加一个名为 **GetQuotesFromCache** 的公共静态方法来从缓存获取 **QuotesCollection** 实例。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

///<summary>
/// Summary description for QuotesManager
///</summary>
public class QuotesManager

```

```

{
    public QuotesManager()
    {
    }

    public static QuotesCollection GetQuotesFromCache()
    {
        QuotesCollection quotes;
        quotes = (QuotesCollection)HttpContext.Current.Cache["quotes"];

        if (quotes == null)
        {
            quotes = new QuotesCollection();
            quotes.Synthesize();
        }
        return quotes;
    }
}

```

6. 为应用程序添加一个 XML Web 服务。右键单击项目节点，在项目中添加一个“Web 服务” (ASMX 文件)，将这个服务命名为 QuoteService。我们可以移除 WebService 和 WebServiceBinding 特性，但应确保该 XML Web 服务类至少具有 [System.Web.Script.Services.ScriptService] 特性 (Visual Studio 已经添加了该特性，将注释去掉即可)。通过这种方式，该特性就可以供 AutoComplete Extender 稍后使用了。AutoComplete Extender 使用这个 Web 服务来填充其下拉列表框。
7. 添加一个用于获取作者名字的方法——这个方法将填充下拉列表框。该方法的第一个参数代表文本框中已输入的文本，第二个参数用于限定返回字符串的最大数目。使用 QuoteManager 的静态方法 GetQuotesFromCache 来从缓存中获取 QuotesCollection。通过 QuotesCollection 的 Rows 属性来获取其中的行。最后，对这些行进行遍历，如果作者名字以传入的字符串为前缀，则将这个名字添加到字符串列表中。“公共语言运行库” (CLR) 中的 String 类型包含一个名为 StartsWith 的方法，可以用来判断字符串是否具有指定的前缀。注意，为使用泛型集合和数据，还必须通过 using 语句来添加相应的引用。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

using System.Data;

[System.Web.Script.Services.ScriptService]
public class QuoteService : WebService
{
    [WebMethod]

```



```

public string[]
GetQuoteOriginatorLastNames(string prefixText,int count)
{
    List<string> list = new List<string>();

    QuotesCollection quotes =
        QuotesManager.GetQuotesFromCache();

    prefixText = prefixText.ToLower();

    foreach (DataRow dr in quotes.Rows)
    {
        string strName =
            dr["OriginatorLastName"].ToString();

        if (strName.ToLower().StartsWith(prefixText))
        {
            if (!list.Contains(strName))
            {
                list.Add(strName);
            }
        }
    }

    return list.GetRange(0,
        System.Math.Min(count, list.Count)).ToArray();
}
}

```

8. 拖放一个 TextBox 到 UseAutocompleteExtender 页面上, 以使用户输入要查询的作者姓名。将这个 TextBox 的 ID 设置为 TextBoxOriginatorLastName。
9. 从“工具箱”拖放一个 AutoCompleteExtender 到页面上。^① 将其 ID 设置为 AutoCompleteExtenderForOriginatorLastName。使 AutoComplete 的 TargetControlID 属性指向用于输入作者姓名的 TextBox(即 TextBoxOriginatorLastName)。将 AutoCompleteExtender 扩展程序的 MinimumPrefix 属性设置为 1, 将 ServiceMethod 设置为 GetQuoteOriginatorLastNames, 将 ServicePath 设置为 quoteservice.asmx。这样, AutoCompleteExtender 扩展程序便会从 TextBoxOriginatorLastName 文本框获取文本, 并将其传给 XML Web 服务方法 GetQuoteOriginatorLastNames。

```

<CCI:AutoCompleteExtender
    ID="AutoCompleteExtenderForOriginatorLastName"
    TargetControlID="TextBoxOriginatorLastName"
    MinimumPrefixLength="1"
    ServiceMethod="GetQuoteOriginatorLastNames"

```

① 译者注: 对于第三方控件, 标签的前缀是在程序中注册时定义的(如在 ASPX 文件中使用<%@ Register%>标签), 这里使用的是默认前缀 asp, 而原书使用的是 ccl。

```

        ServicePath="quoteservice.asmx"
        runat="server">
</CCI:AutoCompleteExtender>

```

10. 在页面中添加一个用于显示名言的 `TextBox`，将其命名为 `TextBoxQuotes`。
11. 修改 `Page_Load` 方法，使它根据文本框中输入的名字来查找名言。为此，要获取 `QuotesCollection` 的实例，然后调用该实例的 `GetQuotesByLastName` 方法。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text;

public partial class UseAutocompleteExtender : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        QuotesCollection quotes =
            QuotesManager.GetQuotesFromCache();
        string[] quotesArray =
            quotes.GetQuotesByLastName(TextBoxOriginatorLastName.Text);

        if (quotesArray != null && quotesArray.Length > 0)
        {
            StringBuilder str = new StringBuilder();
            foreach (string s in quotesArray)
            {
                str.AppendFormat("{0}\r\n", s);
            }
            this.TextBoxQuotes.Text = str.ToString();
        }
        else
        {
            this.TextBoxQuotes.Text = "No quotes match your request.";
        }
    }
}

```

12. 为使页面能够更有效地进行更新，在页面中添加一个 `UpdatePanel` 控件。将显示名言的 `TextBox` 移到 `UpdatePanel` 中。这样，只有显示名言的文本框会被更新，而不会更新整个页面。在用于输入作者名字的 `TextBox` 后面添加一个按钮，将其 ID 设置为 `ButtonFindQuotes`。
13. 为 `UpdatePanel` 添加两个 `asynchPostBack` 触发器。第一个触发器与 `TextBoxOriginatorLastName` 文本框的 `TextChanged` 事件关联，另一个触发器与 `ButtonFindQuotes` 按钮的 `Click` 事件关联。

下图展示了使用 AutoCompleteExtender 的页面。



14. 运行这个页面。在文本框中输入作者名字的过程中，页面会显示一个下拉列表，其中包含了基于 QuotesCollection 中内容的候选项。

AutoComplete 扩展程序展示了 ASP.NET AJAX 的强大功能。Internet Explorer 很早就内建了自动补全功能。该浏览器会记录近期使用的网址。例如，假设在线购买过一张机票，那么当用户要再次到那个网站购票时，地址栏便会进行提示。当用户在地址栏中输入地址的前几个字母后，Internet Explorer 的自动补全功能会显示一个下拉列表，允许用户从中进行选择。

ASP.NET 中的 AutoComplete 扩展程序与之类似。不过，该控件的差别在于，用户看到的候选项是由网站生成的，而不是历史记录。当然，网站可以通过跟踪用户的配置文件标识并存储用户对页面上特定字段的输入，从而来模拟这种历史记录功能。对于 AutoComplete 扩展程序，生成自动补全候选项的过程完全由 Web 服务器承担，这为实现用户友好的网站提供了全新的手段和灵活性。

23.9.2 一种类似模式对话框的组件

AJAX 还提供了一个功能，能够使 Web 应用程序看起来更像桌面应用程序——ModalPopup 扩展程序。在过去，用户在网站中导航的过程一般只是在网站的层次结构中穿梭。在用户输入数据的时候只能使用验证控件来为用户提供数据质量的反馈，标准的网页无法在用户输入信息时引起用户的注意。

传统的桌面应用程序在需要从最终用户获取重要信息时，通常使用模式对话框(modal dialog box)来引起用户注意。模式对话框非常简单和优雅，适用于这样一种情景：最终用户必须输入某些数据，并单击“确定”或“取消”才能继续进行后续操作。在解除对话框后，用户会看到对话框显示之前的界面。这样便不会由于要求用户在网站的层次结构之间穿梭而

引入不确定的或不相关的过程。

下面的练习将介绍如何使用弹出窗口扩展程序控件。我们将创建一个包含标准内容的页面，然后在用户提交表单之前显示一个类似模式对话框的弹出窗口。

➤ 使用 ModalPopup 扩展程序

1. 在 AJAXORama 项目中添加一个页面来承载 ModalPopup 扩展程序，将其命名为 UseModalPopupExtender。
2. 与其他使用 AJAX 控件的练习一样，从“工具箱”拖放一个 ScriptManager 到页面上。
3. 为页面添加一个标题(示例页面的标题为“ASP.NET Code of Conduct”)。为了使标题更加醒目，用<h1>和</h1>标签对其进行修饰。这里只需将<div>标签改为<h1>标签即可。
4. 从“工具箱”拖放一个 Panel(面板)控件来容纳页面的主要内容。
5. 在 Panel 中添加一个 Button 来提交用户数据，将其 ID 设置为 ButtonSubmit，将其文本设置为 Submit。稍后会用到这个控件。
6. 在面板中添加一些内容。这里添加几个复选框，在页面被提交之前会对这些复选框进行检查，并通过模式对话框提示相关信息。

```
<h1>ASP.NET Code Of Conduct</h1>
```

```
<asp:Panel ID="Panel1" runat="server"
    Style="z-index: 1; left: 10px; top: 70px;
    position: absolute; height: 213px; width: 724px;
    margin-bottom: 0px;">
    <asp:Label ID="Label1" runat="server"
        Text="Name of Developer:"></asp:Label>
        &nbsp;<asp:TextBox ID="TextBox1"
            runat="server"></asp:TextBox>

    <br/>
    <br/>
    <br/>
    As an ASP.NET developer, I promise to
    <br/>
    <input type="checkbox" name="Check" id="Checkbox1"/>
    <label for="Check1">Use Forms Authentication</label>
    <br/>
    <input type="checkbox" name="Check" id="Checkbox2"/>
    <label for="Check2">Separate UI From Code</label>
    <br/>
    <input type="checkbox" name="Check" id="Checkbox3"/>
    <label for="Check3">Take Advantage of Custom Controls</label>
    <br/>
```

- 再向页面添加一个 **Panel** 控件，用它来表示弹出窗口。为使弹出窗口更醒目，将这个 **Panel** 的背景设置为淡黄色。将其 ID 设置为 **PanelModalPopup**。
- 为将作为模式弹出窗口的 **Panel** 添加一些内容。它至少应具有 **OK(确定)**和 **Cancel(取消)**按钮。将 **OK** 和 **Cancel** 按钮的 ID 分别设置为 **ButtonOK** 和 **ButtonCancel**。我们稍后会用到这两个按钮。

9. 为当前 ASPX 文件添加一个脚本块。由于没有工具支持，这段代码需要人工输入。分别编写处理 OK 和 Cancel 按钮的单击事件的函数。对于本练习的示例代码，OK 按钮的处理程序会检查复选框是否被选中，并弹出一个对话框，显示哪些复选框被选中了。Cancel 按钮的处理程序只是通过警告对话框来提示 Cancel 按钮被单击了。

446

```
optionsChosen = "Options chosen: ";

if ($get('Checkbox1').checked)
{
    optionsChosen =
        optionsChosen.toString() +
        "Use Forms Authentication ";
}

if ($get('Checkbox2').checked)
{
    optionsChosen =
        optionsChosen.toString() +
        "Separate UI From Code ";
}

if ($get('Checkbox3').checked)
{
    optionsChosen =
        optionsChosen.toString() +
        "Take Advantage of Custom Controls ";
}

if ($get('Checkbox4').checked)
{
    optionsChosen =
        optionsChosen.toString() +
        "Give AJAX a try ";
}

alert(optionsChosen);
}

function onCancel() {
    alert("Cancel was pressed");
}
</script>
```

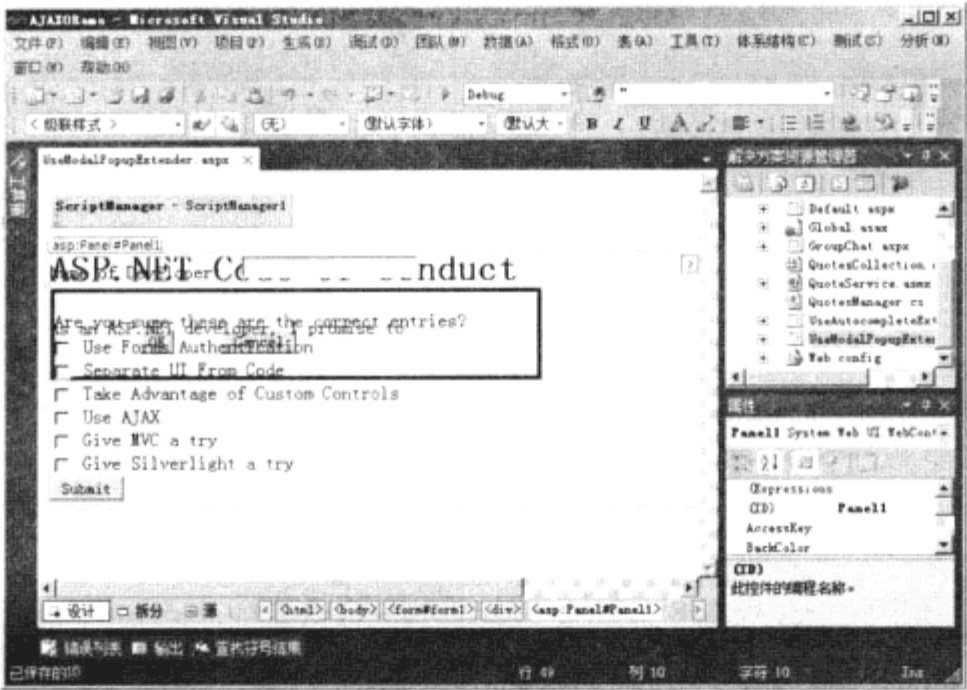
10. 从“工具箱”拖放一个 ModalPopup 扩展程序到页面上。
11. 为页面添加以下代码, 以便设置 ModalPopup 扩展程序的各属性。这段代码将 OkControlID 属性设置为 ButtonOK, 将 CancelControlID 属性设置为 ButtonCancel, 将 OnCancelScript 属性设置为 onCancel()(之前为 Cancel 按钮编写的客户端处理程序), 将 OnOkScript 属性设置为 onOk()(之前为 OK 按钮编写的客户端处理程序), 并将 TargetControlID 设置为 ButtonSubmit:

```
<asp:ModalPopupExtender
    ID="ModalPopupExtender1"
    runat="server"
    OkControlID="ButtonOK"
    CancelControlID="ButtonCancel"
    OnCancelScript="onCancel()"
```

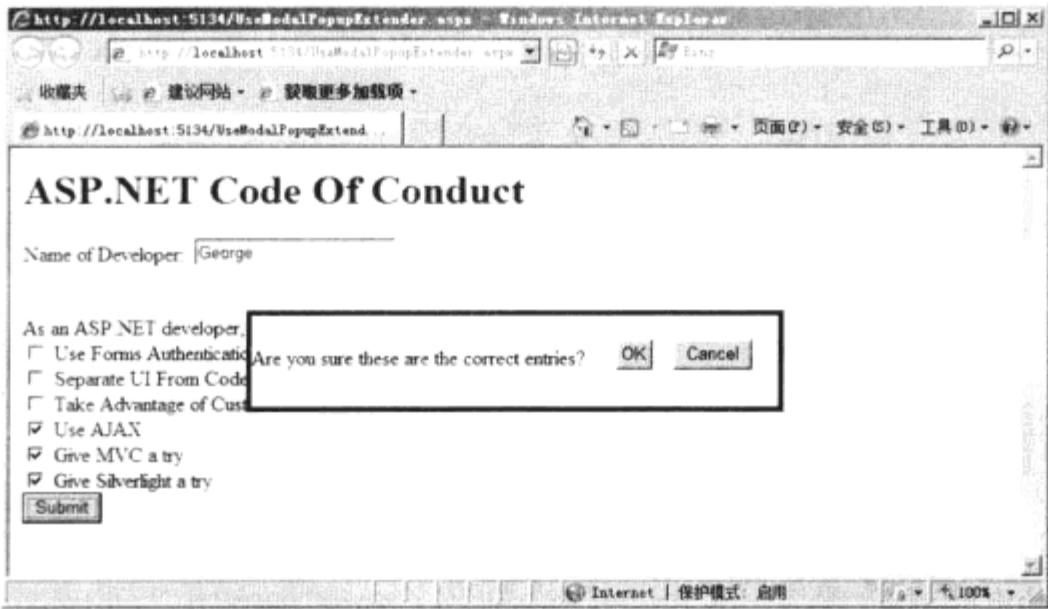


```
OnOkScript="onOk()"
TargetControlID="ButtonSubmit"
PopupControlID="PanelModalPopup"
DynamicServicePath="" Enabled="True">
</asp:ModalPopupExtender>
```

下图展示了添加 ModalPopup 后页面在 Visual Studio 2010 中的效果。



12. 运行这个页面。在单击 Submit 按钮后，作为模式弹出窗口的 Panel 便会被激活。(还记得吗？ModalPopup 通过 TargetControlID 属性与 Submit 按钮建立了关联。)在单击 OK 或 Cancel 关闭弹出窗口后，客户端代码会执行。下图展示了基于 ModalPopup 扩展程序模拟模式弹出窗口的效果。



23.10 快速参考

目 标	操 作
使网站支持 AJAX	通过 Visual Studio 2010 的模板生成的网站默认会启用 AJAX。不过，必须在使用 AJAX 服务器端控件之前为页面添加 ScriptManager

续表

目 标	操 作
实现局部页面更新	在 ASP.NET 项目中，从“工具箱”中拖放 UpdatePanel 控件到目标页面上。这样，只有置于 UpdatePanel 内部的控件会被更新，而页面的其余部分会保持原状
为 UpdatePanel 控件添加触发器 (即通过与 UpdatePanel 控件无关的控件和事件来)	修改 UpdatePanel 的触发器集合，使其包含新的控件和事件。为此，可以在“属性”窗口中选择 Triggers 属性，并通过专用的对话框来添加必要的触发器
使页面按一定间隔进行回发	使用 ASP.NET AJAX 中的 Timer 控件。该控件会按一定的间隔触发回发
通过 AJAX 来使网页的 UI 与众不同	在安装 Visual Studio 2010 后，可以创建支持 AJAX 的网站。可以使用 ASP.NET AJAX Control Toolkit 中特有的服务器端控件，从中选择必要的控件。大部分 AJAX 服务器端控件都可以完全通过服务器端的编程进行控制。然而，有些控件需要编写少量客户端 JavaScript 代码

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 24 章

Silverlight 与 ASP.NET

学习目标

- 理解 Microsoft Silverlight 的重要性
- 创建 Silverlight 应用程序
- 为网页添加 Silverlight 内容
- 理解 Silverlight 编程思想
- 使用 Silverlight 布局
- 使用 Silverlight 动画

第 21 章介绍了如何通过基于 XAML 的浏览器应用程序(XBAP)将 Windows Presentation Foundation(WPF)内容添加到网站中。使用 XBAP 是为客户端浏览器提供内容的方式之一。WPF XBAP 应用程序不仅能够生成可在浏览器中转变为控件的 HTML 标签,还支持更丰富的内容,其中包括复杂的布局模式、丰富的色彩,甚至 2D 和 3D 的图形和变换。

虽然 XBAP 已经在帮助网站生成富内容上前进了一大步,但它也存在一些弊端。XBAP 的主要弊端之一是它只受 Microsoft 操作系统支持。提供 XBAP 的网站无法在 Macintosh 客户端上运行,因为这种客户端不支持 Microsoft .NET Framework 3.0(或更高版本)运行库。另一个弊端是,将传统的 Web 内容与 Windows Presentation Foundation(WPF)内容融合是一件比较复杂的事。添加 XBAP 内容,一种方式是单独创建一个客户端可以访问的页面,另一种方式是在其他 HTML 中添加<iframe />标签来容纳 XBAP 内容。

为克服这些障碍,Microsoft 引入了 Silverlight 来创建“富 Internet 应用程序”(RIA)。RIA 中包含的内容看起来像桌面应用程序。跨平台是 Silverlight 的主要优势之一。在 Windows 中,它可以运行在 Internet Explorer 或 Firefox 浏览器上,而在 Macintosh 中,它可以运行在 Safari 浏览器上。这两个平台占据了大部分市场份额,也使得 Silverlight 能够将 WPF 风格的内容带给更多用户。^①

为使您对 Silverlight 有初步认识,本章将介绍以下几方面内容:

- 在网站中包含 Silverlight 内容的几个理由
- 如何创建 Silverlight 应用程序

① 译者注:如今, Silverlight 已不仅像桌面和 Web 应用程序,还可以用于开发基于 Windows Phone 的移动客户端应用程序,三者之间的移植变得非常容易。

- 如何开发基于 Silverlight 的内容
- 如何将 Silverlight 内容与网站中的页面融合

首先, 让我们看看为什么会出现 Silverlight。

24.1 Web 应用程序的发展

本书读到这里, 您已经了解了一般 Web 应用程序的工作方式——浏览器向服务器发送 HTTP 请求, 服务器通过某些内容进行响应。大多数情况下响应由 HTML 组成, 这些 HTML 表示了用户界面(UI)和在浏览器中显示的数据。正如您所看到的, Microsoft ASP.NET 的响应一般通过 Page 类和服务器端控件树生成。Page 可以包含若干服务器端控件, 由服务器端控件向响应流生成 HTML。最终, 浏览器将响应流变成浏览器显示的可视元素。

最早的网页一般只包含图片、带有格式的文本、用于在网站中导航的 HTTP 链接, 而没有特别之处。比如, 那时没有通过动态 HTML(DHTML)生成的上下文菜单, 没有可交互的图形, 也没有弹出窗口(使用 AJAX)。随着 Web 技术的发展, 这些类似桌面的用户界面特性出现在越来越多的网站中。事实上, 其中许多特性的演变都可以在一个名为 The Wayback Machine 的网站中找到(可访问 <http://www.archive.org>)。这个网站展示了某个公司网站的演变历程——可以追溯到 1996 或 1997 年。

有些 Web 技术(如 DHTML 和 AJAX)已经将这种 UI 和交互性带到了支持脚本的客户端中。例如, 早期的弹出菜单(在鼠标悬停在某个菜单项上时显示的菜单)一般是通过 DHTML 实现的。在使用时开发者一般仍需要自行编码。MSDN 发布过一个 ASP.NET 服务器端控件集, 通过 ASP.NET 简化了开发者的工作。

对于更丰富的内容, 许多开发者使用 Flash 实现(它最初是 Macromedia 的产品, 现已归属 Adobe)。直到现在, Flash 仍是 Web 开发者实现丰富图形和动画的主要工具。针对 Windows 的 Flash 播放器是 ActiveX 控件, 在其他操作系统中则是浏览器插件。在 20 世纪 90 年代末到 21 世纪的最初几年, 大多数动画和图形内容都采用 Flash 制作。Adobe 最新提供了一种名为 Flex 的工具来制作富内容。

既然有了这么多 Flash 和 Flex 内容, 有人可能会问: 微软为什么要在 RIA 领域冒险去开发一种竞争对手已涉足的 Web 工具呢?

最令人信服的答案可能是, Silverlight 是利用现有的产品来提供上述技术所提供的功能(尤其针对 Microsoft 开发者)。笔者个人认为, 作为一名网站开发者, 从优雅的、简洁的.NET 编程模型转型到脚本编程模型是一件很困难的事。脚本编程会带来一系列的问题。例如, 编写脚本经常会由于类型不安全而出现错误。由于脚本没有预先编译为本地代码, 而是边解释边执行的, 因而其性能较低。此外, 开发脚本的工具也远不如开发.NET 代码的工具成熟。

下面让我们从概念上认识一下 Silverlight。

24.2 何为 Silverlight

Silverlight 是 Microsoft 在 RIA 领域的尝试。RIA 的思想已经被广泛接受。通过客户端脚本、AJAX 或浏览器插件, 越来越多与桌面应用程序类似的功能被加入到基于浏览器的应用程序中。对于 Windows 平台上的 Internet Explorer 和 Firefox, 以及 Macintosh 平台上的 Safari, Silverlight 带来了丰富的客户端功能。

在最近几年里, Silverlight 已历经了多个版本, 从最初的 1.0 到 2, 再到现在的 4。Silverlight 每一个版本都是以前一个版本为基础构建的, 但 1.0 和 2 之间的差异较大。^①

Silverlight 1.0 只不过是一个“可扩展应用程序标记语言”(XAML)呈现引擎。虽然它允许用户与浏览器中的内容进行交互, 并支持事件, 但都是通过脚本来实现的。Silverlight 实际上是 .NET 公共语言运行库(CLR)与 WPF 呈现技术的一个子集, 运行在客户端。如果采用 Silverlight 2、3 或 4 进行开发, 程序将被即时(JIT)编译为“中间语言”(IL), 并像桌面 .NET 应用程序那样运行。Silverlight 提供了许多 .NET 特性, 但并不是全部, 因为有的功能并不适用于 Silverlight, 或者在基于浏览器的平台中实现没有意义。

Silverlight 所具有的基本功能如下:

- 可以通过 Silverlight 在网页中构建丰富的、可交互的内容。虽然页面的 HTML 还是由浏览器来解释, 但 Silverlight 内容由一种轻型的 CLR 来运行。Silverlight 的 .NET 编程模型仍沿用了传统的 .NET 事件处理程序、.NET 集合, 以及 .NET 控件与事件模型。
- Silverlight 包含丰富的控件库, 并在不断被扩展。从开发角度讲, 这些控件非常类似常见的 Windows 控件和 ASP.NET 控件, 其中包括 Button、ListBox、RadioButton、Label 和 TextBox 控件。这些控件与传统的 Windows 和 ASP.NET 控件具有许多相同的属性和事件。例如, Silverlight 中的 Button 控件同样支持 Foreground 和 Background 属性, 也支持 Click 事件。Silverlight 中的 ListBox 也支持可绑定的集合和 SelectionChanged 事件。
- Silverlight 支持丰富的图形。在过去, 使浏览器显示图片需要在设计工具中进行绘制, 然后以 JPG 或 PNG 文件的形式发送给客户端。Silverlight 提供了一套绘图 API, 可以用来在客户端计算机上呈现图形。由于 Silverlight 允许以编程方式访问图形, 因而图形变成了动态内容, 能够在运行时变化并响应用户的操作。
- Silverlight 包含支持媒体服务的控件, 开发者可以非常方便地在网页中嵌入视频和音频。
- Silverlight 能够良好地与 HTML 文档对象模型结合。我们可以在 Silverlight 代码中访问

① 译者注: 在本书截稿前, Silverlight 的最新版本是 4, 而原书是 3。由于 Silverlight 总体上是向后兼容的, 因而这种版本差异不会影响本章所要介绍的内容。有关不同版本间功能上的差异, 可以参考以下页面的“Features Matrix”一节:
<http://www.silverlight.net/getstarted/overview.aspx>。

HTML 元素, 也可以通过 JavaScript 代码访问 Silverlight 代码。

- Silverlight 针对 .NET 开发者。Silverlight 实际上是一种下载到客户端的轻型 .NET 运行库, 它以 ActiveX 控件(针对基于 Windows 的计算机)或 Safari 浏览器插件(对于 Mac 计算机)的形式存在。虽然有一些其他的 RIA 工具, 但它们都采用特殊的语法来支持动态内容。 .NET 开发者之所以能够进行 Silverlight 开发是因为可以通过 .NET 编程模型来编写 Silverlight 应用程序。
- Silverlight 拥有强大的工具支持。Microsoft Visual Studio 2010 全面支持 Silverlight 开发, 从编程到调试。设计人员可以使用 Microsoft Expression Blend 来单独开发应用程序的可视界面, 而且这种界面与应用程序的逻辑无关。Microsoft Expression Design 是一种高级矢量图形工具, Microsoft Expression Encoder 是一种多媒体工具。

Silverlight 的许多架构和设计思想来源于 WPF, 因而在许多方面都与 WPF 一致。例如, Silverlight 和 WPF 的可视 UI 元素都通过 XAML 来描述——XAML 是一种基于 XML 的标准, 用于表达对象模型(如 Silverlight 可视树的对象模型)。此外, 两者的编程逻辑都通过 .NET 语言来表达。

下面让我们看看如何创建 Silverlight 应用程序。

24.3 创建 Silverlight 应用程序

在深入 Silverlight 的细节之前, 先考虑一下如何通过 Visual Studio 将 Silverlight 与其他项目类型融合。使用 Visual Studio 创建 Silverlight 应用程序与创建其他类型的应用程序的方法相同。Visual Studio 包含创建 Silverlight 内容的模板, 并提供了两种选择:

- 通过简单 HTML 测试页面来运行 Silverlight 内容
- 通过完整 ASP.NET 环境下的 ASP.NET 页面来运行 Silverlight 内容

不论选择哪一种方式, Silverlight 组件的开发都遵循 .NET 编程模型。首先开发出 Silverlight 部分, 然后再用 HTML 页面或 ASP.NET 项目来加以练习。事实上, 在调试时, Visual Studio 会启动 Web 开发服务器和我们所选择的浏览器——就像一般 Web 应用程序的开发一样。如果在 Silverlight 代码中设置断点, Visual Studio 也允许我们跟踪代码的执行。

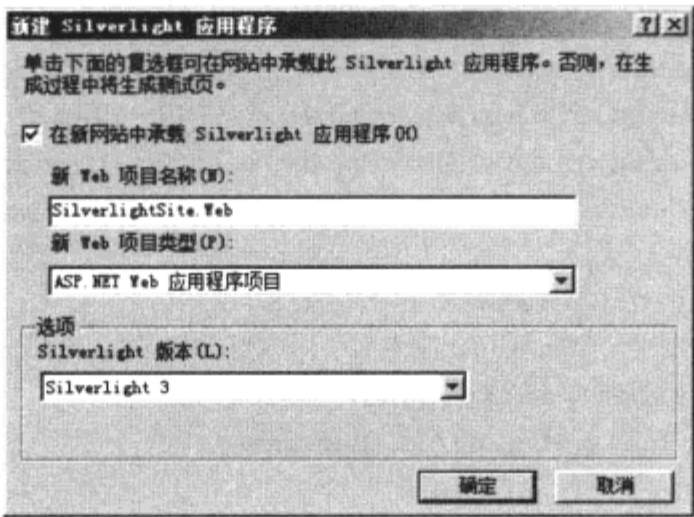
作为一个开端, 下面的练习将演示创建一个 Silverlight 应用程序所涉及的必要步骤。后面的练习会演示更多 Silverlight 的功能。

➤ 创建 Silverlight 应用程序

1. 首先, 打开 Visual Studio。在主菜单中选择“文件”|“新建”|“项目”。在“已安装的模板”中选择“Silverlight”, 然后选择“Silverlight 应用程序”模板。将这个项目名称命名为 SilverlightSite。



2. 为测试 Silverlight 应用程序，Visual Studio 会询问是创建单个 HTML 页面，还是创建完整的 ASP.NET 网站。如果选择创建网站，则需输入网站项目的名称和 Web 项目类型。该对话框还提示我们选择所要使用的 Silverlight 的版本。^①默认选项是创建一个完整的网站(如下图所示)。这里保持默认设置，单击“确定”。



3. Visual Studio 会生成包含两个项目的解决方案。第一个项目包含的是 Silverlight 内容，另一个是“Web 应用程序”项目。下图展示了创建这个 Silverlight 应用程序后的“解决方案资源管理器”。

解决方案中的第一个节点代表包含 Silverlight 内容的项目。与一般的项目一样，该项目中的“Properties”文件夹中有一个名为 AssemblyInfo.cs 的文件，其中包含程序集的信息。Silverlight 项目也有“引用”文件夹，其中包含对其他程序集的引用信息。这个项目默认引用的程序集实际上是针对 Silverlight 的 .NET 系统程序集。此外，该项目节点下还有 MainPage.xaml 和 App.xaml 节点。后面的 24.4 节将深入介绍这两个文件。

^① 译者注：如果要开发 Silverlight 4，可以安装 Microsoft Silverlight 4 Tools for Visual Studio 2010，其中包含所有必备程序和文件。在这种情况下，“Silverlight 版本”下拉列表中会包含针对 Silverlight 的不同版本的选项。



解决方案中的第二个项目是一个 ASP.NET 网站，用于承载 Silverlight 内容。这个网站与一般的 ASP.NET 网站非常类似。其中常规的“Properties”和“引用”文件夹。不过，其中也包含一些特殊的文件夹和文件。例如，有两个名为 SilverlightSiteTestPage 的文件——一个是 HTML 版本，一个是 ASPX 版本。这两个文件演示了在网站中承载 Silverlight 内容的不同方式。详细内容，请阅读后面的 24.7 节。

4. 打开 MainPage.xaml 文件，输入以下标签(加粗的部分)，使页面呈现一个 Button。这段代码会使 Silverlight 的显示区域出现一个 Button 控件，在 Content 属性中输入的文本都会被 Button 显示出来(效果可以稍后在应用程序运行后看到)。

```
<UserControl x:Class="SilverlightSite.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Hello World!"></Button>
    </Grid>
</UserControl>
```

5. 按快捷键 Ctrl+F5，生成并运行程序。由于在创建解决方案时选择了 ASP.NET 项目来作为测试项目，因而 Visual Studio 会启动 Web 开发服务器，并打开 ASP.NET 测试页面。由于没有为按钮定义 Height 和 Width 属性，因此这个按钮充满了整个 Silverlight 显示区域(见下图)。



6. 为使 Silverlight 内容具有交互性，下面我们为按钮添加一个处理程序。在添加处理程序之前，最好对这个控件命名，因为 Visual Studio 会在生成的处理程序名称中使用控件的名称。这样方便后续对这个处理程序的跟踪。为<Button>标签添加 Name 特性，并在之后添加 Click 处理程序。在输入单词 Click 之后，Visual Studio 会为我们生成相应的事件处理程序^①。此时的代码如下所示：

```
<UserControl x:Class="SilverlightSite.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Hello World!"
            x:Name="theButton"
            Click="theButton_Click">
        </Button>
    </Grid>
</UserControl>
```

7. 下面我们为按钮的 Click 处理程序添加一些代码。打开 MainPage.xaml.cs 文件并找到新建的处理程序。^②如果是 Visual Studio 自动生成的处理程序，则其名称应该是 theButton_Click。这个处理程序接受两个参数(与其他.NET 处理程序一样)。第一个参数是发送者(sender)——对于按钮，这个参数是被单击的按钮的引用。第二个参数是 RoutedEventArgs。该参数类似.NET 中的 EventArgs，其中包含有关事件的信息。不过，由于 Silverlight 会管理自己的消息路由模式(路由事件，routed event)，因而事件信息参数的类型有所改进。

修改 Click 处理程序，更改按钮的文字内容，放大字体，并将前景颜色更改为红色(详见下面代码被加粗的部分)。

```
private void theButton_Click(object sender,
    RoutedEventArgs e)
{
    Button button = sender as Button;
    button.Content =
    "The button was clicked";
    button.FontSize = 22;
}
```

① 译者注：请注意 Visual Studio 的提示：

```
<Button Content="Hello World!"
    x:Name="theButton" <新建事件处理程序>
    Click="theButton_Click"></Button>
```

② 译者注：除了手动打开代码旁置文件进行查找外，也可以在“属性”窗口的“事件”选项卡里双击目标事件。另外一种快速定位事件处理程序的方法是在“XAML”视图中右键单击目标事件名称，然后从菜单中选择“导航到事件处理程序”——这一操作默认是没有快捷键的，但可以在 Visual Studio 的“选项”窗口中(选择“环境”|“键盘”节点)为代表该操作的 EditorContextMenus.XAMLEditor.NavigateToEventHandler 分配一个。

```
button.Foreground =
    new SolidColorBrush(Colors.Red);
}
```

8. 运行这个程序, 测试其能否正常工作。Visual Studio 会打开一个浏览器并显示 ASP.NET 测试页面。在单击按钮后, 按钮中的文本内容会发生变化, 字体也会变大, 而前景色会变为红色。

24.4 架 构

直观地认识 Silverlight 后, 下面让我们了解一下其内部机制。Silverlight 组件的架构非常简单。它由 Application 对象(由 Application.xaml 和 Application.xaml.cs 文件定义)的案例和可视组件(MainPage.xaml 和 MainPage.xaml.cs 文件定义)构成。Silverlight 显式地区分表示层代码和逻辑(行为)代码——这与 ASP.NET 如出一辙。ASP.NET 的代码旁置模型涉及 ASPX/C#或 ASPX/Visual Basic 文件对。Silverlight 采用 XAML/C#或 XAML/Visual Basic 文件对的形式。XAML 文件通常用于定义用户界面, 而 C#或 Visual Basic 文件用于定义行为。

Application 对象是 Silverlight 组件的全局访问点, 能够存储组件范围内的状态, 并具有可订阅的应用程序启动事件和关闭事件。如果打开 Application.xaml.cs 文件, 则会看到 Visual Studio 已经在注释中包含了订阅这些事件的存根。

Application 对象总是可用的(这与 ASP.NET 中的 HttpApplication 一样)。除支持全局事件外, Application 对象还能够维护全局状态。

Application 存在的另一个作用是设置应用程序的 RootVisual 属性。RootVisual 属性用于向 Silverlight 呈现引擎提供要显示的内容。RootVisual 属性的类型派生自 UserControl, 指向可视内容所在的位置。深入研究一下 UserControl 便可以理解 XAML 的工作原理。

24.5 XAML

XAML 代表“可扩展应用程序标记语言”(eXtensible Application Markup Language)。XAML 是一种 XML, 主要用于以声明方式描述可视树。虽然它最初是为支持 WPF 而开发的, 但也可以用在其他方面——最特别的是它用在了 Windows Workflow Foundation 中。Silverlight 的表示层通过 XAML 来描述。

Silverlight 通过 XAML 来生成可视树。不妨回顾一下上个练习的 MainPage 类:

```
<UserControl x:Class="SilverlightSite.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Hello World!"
```



```
x:Name="theButton"
Click="theButton_Click"></Button>
</Grid>
</UserControl>
```

这段 XAML 描述了一个 UserControl 类的实例，这个实例与一个名为 SilverlightSite.MainPage 的类关联，将 DesignWidth 属性初始化为 400，并将 DesignHeight 属性初始化为 300。它还描述了一个 Grid 类的实例，名称为 LayoutRoot，并将 Background(背景)设置为白色。这段代码最后描述了一个 Button 类的实例，其名称为 theButton，Content 属性为“Hello World!”，并通过 theButton_Click 方法订阅了 Click 事件(该方法是 SilverlightSite.MainPage 类的成员)。在学习过 ASP.NET 之后，便不会对此感到陌生——这也是 ASP.NET 中 Page 类构造其控件树的方式。这里的主要差别在于 ASPX 与 XAML 语法上的差异。

24.5.1 可视树的构造

Silverlight 中的 XAML 用于构造 Silverlight 组件的可视树。该步骤在 MainPage 类的构造函数中进行。注意，Visual Studio 生成的代码调用了 InitializeComponent 方法。InitializeComponent 方法会对 XAML 进行解析，以创建组成可视树的对象，并根据 XAML 初始化这些对象的属性。例如，上面那段 XAML 代码和下面这段代码所构造的可视树相同：

```
Page ConstructVisualTreeProgrammatically()
{
    PagePage = new page()
    Grid grid = new Grid();
    Button button = new Button();
    button.Click += this.theButton_Click;
    button.Content = "Hello World!";
    grid.Children.Add(button);

    return grid;
}
```

24.5.2 XAML 与命名空间

上一个练习的 XAML 代码的最上端定义了 4 个 XAML 命名空间，这里将介绍前两个(因为这两个最为重要)。默认的命名空间被指定为“http://schemas.microsoft.com/winfx/2006/xaml/presentation”。这个命名空间包含 Silverlight 的常见功能和控件标准。例如，指定这个命名空间后，便可以直接使用<Button />标签，而无需添加任何前缀。第二个命名空间“x”被指定为“http://schemas.microsoft.com/winfx/2006/xaml”。这个命名空间定义了 Silverlight 的关键字，例如 Name 和 Key——前者用于指定 XAML 元素名称，以便在编程时使用；后者用于指定 Silverlight 资源字典中的键/值对。

如果必要，可以添加其他命名空间。例如，如果希望在 XAML 中使用 SilverlightSite 程序集中定义的类型，则可以添加一个自定义的命名空间。假设这个 Silverlight 组件程序集中

有一个名为 `CustomButton` 的类型，并希望将其添加到界面上，则可以在 XAML 中像下面这样添加命名空间和使用其中的组件(加粗的代码)。

```
<UserControl x:Class="SilverlightSite.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

    xmlns:my="clr-namespace:SilverlightSite"

    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <my:CustomButton
            Content="Hello World!"
            x:Name="theButton"
            Click="theButton_Click"></my:CustomButton>

    </Grid>
</UserControl>
```

24.6 Silverlight 应用程序的编译

为在 Visual Studio 中编译 Silverlight 应用程序，可从主菜单中选择“生成”|“生成解决方案”，或者按 F6 键。在生成 SilverlightSite 解决方案后，Visual Studio 会将 Silverlight 组件编译为常规的 .NET 程序集(虽然引用的是 Silverlight 系统程序集，而不是一般的 .NET 程序集)。此外，Visual Studio 还会创建 XAP(读作|zæp|)文件，其中包含希望在客户端浏览器中使用的 Silverlight 组件和相关资源(图片、图形、字体等)。

XAP 文件是以 ZIP 格式压缩的文件。所以，我们可以通过任何标准 ZIP 压缩工具打开并查看其中的内容。对于使用 Silverlight 组件的页面而言，对 Silverlight 内容进行压缩可以缩短页面的加载时间。

最后，Visual Studio 会将 XAP 文件复制到 ASP.NET 项目中的 ClientBin 目录。

24.7 在网页中添加 Silverlight 内容

添加 Silverlight 内容有两种方式。第一种是使用传统的 `<object>` 标签，将其嵌在 HTML 中。第二种是通过 JavaScript 脚本来动态加载 Silverlight 内容。^①

① 译者注：除了这两种方法外，原书还介绍了通过 `asp:Silverlight` 控件来添加 Silverlight 内容的方法。然而，从 Silverlight 3.0 开始，出于某些原因，这个控件已从 Silverlight SDK 中被移除。Visual Studio 2010 RTM 生成的代码也不会包含这个控件，而是在 HTML 和 ASPX 文件中统一采用 `<object>` 标签(两者的标记几乎是一样的)。因此，译稿中没有包含有关该控件的内容。虽然不能使用这个控件，但可以在 ASP.NET 自定义控件或用户控件中对其进行包装。

首先，让我们看看如何使用<object>标签进行添加。

24.7.1 使用<object>标签

第一种添加 Silverlight 内容的方法是使用标准的<object>标签(这是 20 世纪 90 年代末为在页面中加载 ActiveX 控件而引入的)。事实上，Visual Studio 会创建一个通过<object>标签加载 Silverlight 内容的 HTML 和 ASPX 测试页面。

```
<!-- 这里包含 Silverlight 在运行时显示的错误-->
<!-- 这里包含调试信息。在调试完成后，应将其移除或隐藏-->
<body>
  <form id="form1" runat="server" style="height: 100%">
    <div id="silverlightControlHost">
      <object data="data:application/x-silverlight-2,"
        type="application/x-silverlight-2"
        width="100%" height="100%">
        <param name="source" value="ClientBin/SilverlightSite.xap"/>
        <param name="onError" value="onSilverlightError"/>
        <param name="background" value="white"/>
        <param name="minRuntimeVersion" value="3.0.40818.0"/>
        <param name="autoUpgrade" value="true"/>
        <a href="http://go.microsoft.com/fwlink/?LinkId=149156&v=3.0.40818.0"
          style="text-decoration: none">
          
        </a>
      </object>
      <iframe id="_sl_historyFrame"
        style="visibility: hidden; height:0px; width:0px;
        border: 0px"></iframe>
    </div>
  </form>
</body>
```

对于 Windows 客户端，Silverlight 是以 ActiveX 控件的形式实现的，而对于 Macintosh 客户端，Silverlight 是以 Safari 浏览器插件的形式实现的。代码中的<object>标签会引导服务器检查客户端计算机上 Silverlight 的实现，然后从服务器获取相应的 XAP 文件，并承载它。注意参数是如何传入 Silverlight 中的。这段代码中的几个参数是预定义的，开发者可以定义其他参数并在 Silverlight 加载时传入其中。

24.7.2 使用 JavaScript 函数

Visual Studio 提供了一个 JavaScript 库，其中包含一个辅助函数——Silverlight.createObjectEx。该函数位于 Silverlight.js 文件中，在生成测试项目时 Visual

Studio 会自动添加该文件。为添加 Silverlight 内容，可以在 HTML 页面中调用 `Silverlight.createObjectEx`，并传入 XAP 文件的路径、事件和其他与 Silverlight 内容有关的参数。这种方式可以实现 Silverlight 内容的动态调用(如通过 JavaScript 事件)。

24.8 控件与事件

如果熟悉 ASP.NET 中的控件，那么对 Silverlight 控件也不会陌生。Silverlight 几乎包含所有与 ASP.NET 服务器端控件(如 Button、ListBox、RadioButton、TextBox、Label 等)相同的控件。

ASP.NET 服务器端控件的任务是生成最终供浏览器解释的标签，而 Silverlight 控件直接由客户端的 Silverlight 引擎呈现。ASP.NET 控件的事件通常由服务器端的事件处理程序进行处理，而 Silverlight 控件的事件则在客户端由 Silverlight 组件进行处理。

24.8.1 路由事件

Silverlight 控件的事件与一般的 .NET 事件非常相似。所有事件处理程序都以 `object` 作为第一个参数(名称总是 `sender`)。紧随 `sender` 参数之后是事件参数——标准事件参数的变体。为管理事件，Silverlight 使用了路由事件(routed event)。一般的 .NET 事件能够与触发事件的控件直接关联(例如 Button 控件暴露的 Click 事件)。路由事件与一般的事件类似，但路由事件可以在 Silverlight 可视树的任意位置关联事件和处理程序。也就是说，Silverlight 不要求处理程序与控件直接关联。这种机制在许多场景下都能用到。例如，假设要为版式面板(layout panel，如 Grid)中的所有元素捕获鼠标左键被按下的事件。此时，我们不用为面板中的每个元素都设置一个处理程序，而只需要将处理程序与这个版式面板关联，在此监听鼠标左键被按下的事件即可。

24.8.2 Silverlight 控件与类成员

在 ASPX 文件中包含 ID 的 ASP.NET 标签在代码旁置类中都有对应的成员，具有名称的 XAML 标签在代码旁置类中也有对应的成员。例如，下面这段 XAML 会在 Silverlight 应用程序的代码旁置类中生成两个成员变量，其类型分别为 Grid 和 Button，名称分别为 `LayoutRoot` 和 `theButton`。

```
<UserControl x:Class="SilverlightSite.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">
    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Hello World!"
```



```
x:Name="theButton"
Click="theButton_Click"></Button>
</Grid>
</UserControl>
```

Expression Blend

虽然可以通过 Visual Studio 的设计器来查看当前内容的效果，但 Visual Studio 更适合用来编写代码和调试。为进行更复杂的设计工作，Microsoft 提供了 Expression Blend。我们可以考虑使用 Expression Blend 来设计和修改应用程序的可视元素。本章无法深入讲解 Expression Blend，但值得一提的是，Visual Studio 和 Expression Blend 具有相同的解决方案和项目文件格式。可以通过 Visual Studio 来创建 Silverlight 应用程序，并通过 Expression Blend 打开，反之亦然。

刚刚提到，Expression Blend 的具体内容不在本书的讨论范围内，但学习这个工具还是很有价值的。Expression Blend 对 Silverlight 组件提供的可视支持与 Visual Studio 所提供的不同。有关 Expression Blend 的更多内容，可以访问 Microsoft Expression，网址：http://www.microsoft.com/expression/products/Blend_Overview.aspx。

24.9 Silverlight 的布局方式

正如本章之前所提到的，MainPage(派生自 UserControl)是 Silverlight 内容的主要显示区域。UserControl 是一种承载单个界面内容的 ContentControl。从之前的 SilverlightSite 应用程序不难看出，虽然可以在页面上放置任何控件(如 TextBlock 或 Button)，但显示空间毕竟有限。

Silverlight 并没有强制规定使用某种特定的布局模式(如绝对 x,y 定位方式)，而是通过版式面板支持了多种布局方式。在面板中添加控件后，每个控件在用户界面上的位置便会自动调整。Silverlight 默认提供的版式面板包括 Canvas(画布)、Grid(网格)和 StackPanel(堆叠面板)。

Canvas 版式面板可以放置在 MainPage 中，这种面板会根据绝对(x,y)坐标对 UI 元素进行定位。Grid 版式面板能够以行和列的方式对元素进行布局。StackPanel 能够以横向或纵向对元素进行布局。下面的练习演示了 Silverlight 的布局方式。

► 利用 Silverlight 布局

1. 创建一个“Silverlight 应用程序”，将其命名为 SilverlightLayout。通过 Visual Studio 创建一个 ASP.NET 测试项目。
2. 打开 Visual Studio 生成的 MainPage.xaml 文件，将 UserControl 的宽度设置为 600。
3. Visual Studio 默认采用网格进行布局。下面我们打开网格线，并添加两个网格列。Grid.ShowGridLines(显示网格线)属性是 Boolean(布尔)类型。然而，网格列的定义稍微

复杂些, 要使用这样的语法:

```
<UserControl x:Class="SilverlightLayout.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"

  d:DesignHeight="300" d:DesignWidth="600">
  <Grid x:Name="LayoutRoot" Background="White"
    ShowGridLines="True">
    <Grid.ColumnDefinitions>
      <ColumnDefinition></ColumnDefinition>
      <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
  </Grid>
</UserControl>
```

4. 在网格中添加一个 Canvas 控件, 将其命名为 theCanvas, 以便稍后在程序中引用。将其置于 0 列中。下面我们为这个 Canvas 添加一些内容。这里演示如何绘制图形, 并使用附加的 Canvas.Left 和 Canvas.Right 属性来在 Canvas 中对其进行定位。注意, 这些图形设置了 Stroke 和 StrokeThickness 属性, 分别用于定义图形边缘的颜色和粗细。Fill 属性用于设置填充图形的颜色。最后, 将每个图形的不透明度(Opacity 属性)设置为 60%。我们可以通过这个属性实现鼠标的某种交互效果。

```
<UserControl x:Class="SilverlightLayout.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d"

  DesignWidth="600"> DesignHeight="300"
  <Grid x:Name="LayoutRoot" Background="White"
    ShowGridLines="True">
    <Grid.ColumnDefinitions>
      <ColumnDefinition></ColumnDefinition>
      <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>

    <Canvas Grid.Column="0" x:Name="theCanvas">
      <Rectangle Width="100" Height="100"
        Fill="LightBlue" Stroke="Black"
        StrokeThickness="2"
        Canvas.Top="75" Canvas.Left="150"
        Opacity=".6">
      </Rectangle>
```

```

        <Ellipse Width="150" Height="100"
            Fill="Green" Stroke="Yellow"
            StrokeThickness="3"
            Canvas.Left="50"
            Canvas.Top="175"
            Opacity=".6">
        </Ellipse>
        <Rectangle Width="150" Height="150"
            Fill="Red"
            Stroke="Maroon"
            StrokeThickness="3"
            Canvas.Left="30"
            Canvas.Top="45"
            Opacity=".6">
        </Rectangle>
    </Canvas>
</Grid>
</UserControl>

```

5. 下面我们为 Canvas 上的图形添加鼠标交互效果。首先，添加 MouseEnter 和 MouseLeave 事件的处理程序，使得当鼠标指针悬停于 Canvas 中的图形上时，提高图形的不透明度。为此，打开 MainPage.xaml.cs 文件，实现 MouseEnter 和 MouseLeave 事件的处理程序。由于事件处理程序将与图形本身关联，因而这些事件的发送者(sender)将会是图形。为访问图形的 Opacity 属性，要将第一个参数(sender)的类型转换为 Shape。MouseEnter 处理程序应将 Opacity 设置为 1(即 100%)，而 MouseLeave 处理程序则将 Opacity 设置为 .6(60%)。实现 MouseEnter 和 MouseLeave 处理程序之后，在 MainPage 构造函数中，订阅 Canvas 上所有元素的 MouseEnter 和 MouseLeave 事件。代码如下所示：

```

public partial class MainPage : UserControl
{
    private void theCanvas_MouseEnter(object sender, MouseEventArgs e)
    {
        Shape shape = sender as Shape;
        if (shape != null)
        {
            shape.Opacity = 1;
        }
    }
    private void theCanvas_MouseLeave(object sender, MouseEventArgs e)
    {
        Shape s = sender as Shape;
        if (s != null)
        {
            s.Opacity = .6;
        }
    }

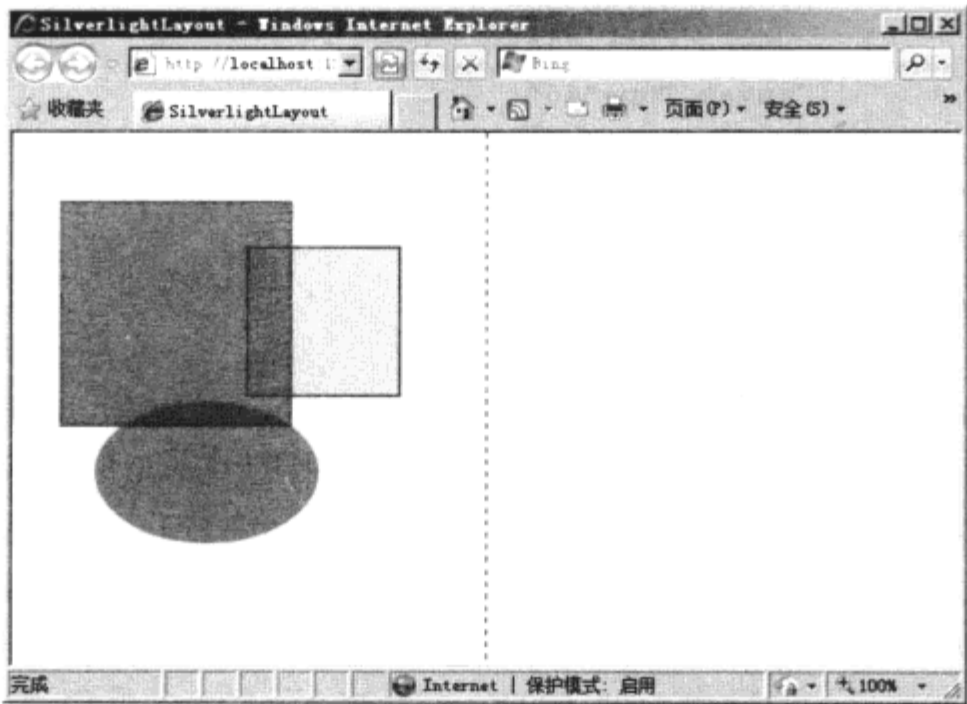
    public MainPage()
    {
        InitializeComponent();
    }
}

```

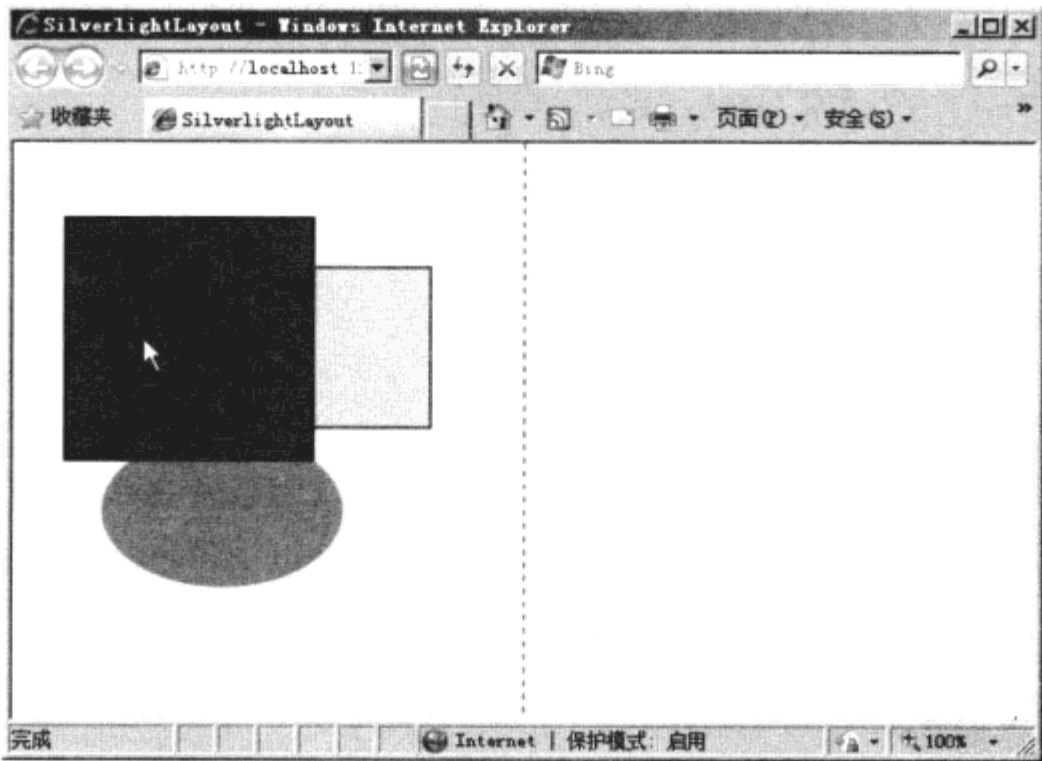


```
foreach (Shape s in theCanvas.Children)
{
    s.MouseEnter += this.theCanvas_MouseEnter;
    s.MouseLeave += this.theCanvas_MouseLeave;
}
}
```

6. 运行这个应用程序。最初，所有图形的边缘都可见，因为此时的不可见度都是 60%(如下图所示)。



7. 然后，当鼠标悬停在某个图形上后，该图像应变为不透明(如下图所示)。



对于这张图，当指针悬停在最上层的矩形中时，这个矩形区域会将另一个矩形和椭圆遮挡住。

8. 在网格中添加一个 StackPanel 控件，将其置于第二列(列 1)。在 StackPanel 中添加一个按钮、一个 ListBox 和另一个 StackPanel。为这个 ListBox 填充几个项目。将第二个

StackPanel 的 Orientation 属性设置为 Horizontal，使其横向布局。在第二个 StackPanel 中再添加两个 ListBox。

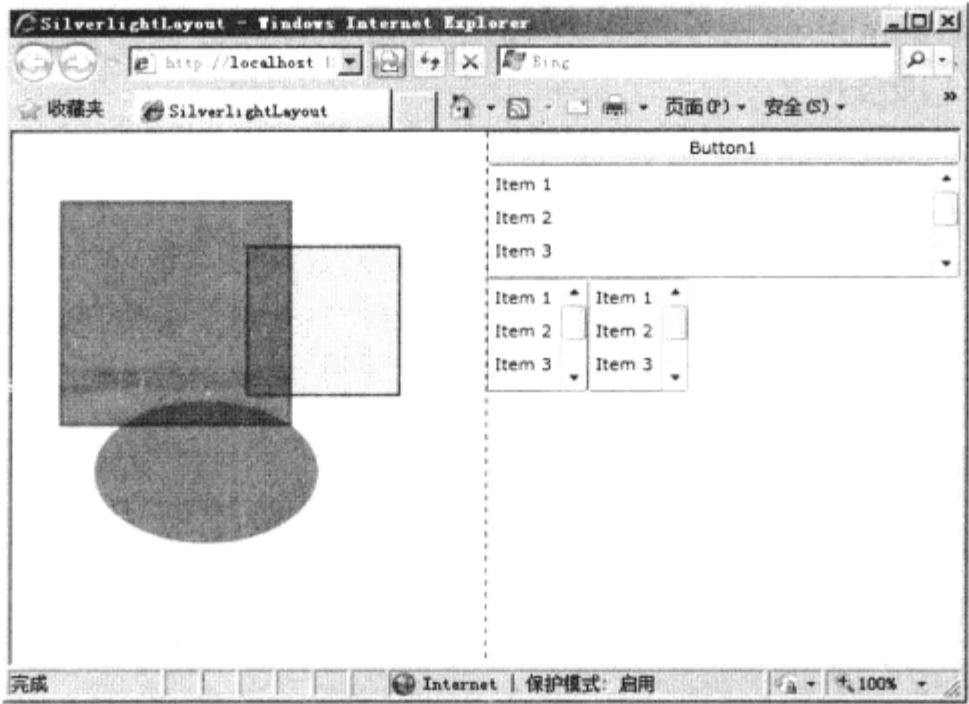
```
<UserControl x:Class="SilverlightLayout.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"

DesignHeight="600" DesignWidth="600">
    <Grid x:Name="LayoutRoot" Background="White"
        ShowGridLines="True">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <!--这里是 Canvas 控件... -->

        <StackPanel Grid.Column="1">
            <Button Content="Button1"></Button>
            <ListBox Height="75">
                <ListBoxItem Content="Item 1"></ListBoxItem>
                <ListBoxItem Content="Item 2"></ListBoxItem>
                <ListBoxItem Content="Item 3"></ListBoxItem>
                <ListBoxItem Content="Item 4"></ListBoxItem>
                <ListBoxItem Content="Item 5"></ListBoxItem>
            </ListBox>
            <StackPanel Orientation="Horizontal">
                <ListBox Height="75">
                    <ListBoxItem Content="Item 1"></ListBoxItem>
                    <ListBoxItem Content="Item 2"></ListBoxItem>
                    <ListBoxItem Content="Item 3"></ListBoxItem>
                    <ListBoxItem Content="Item 4"></ListBoxItem>
                    <ListBoxItem Content="Item 5"></ListBoxItem>
                </ListBox>
                <ListBox Height="75">
                    <ListBoxItem Content="Item 1"></ListBoxItem>
                    <ListBoxItem Content="Item 2"></ListBoxItem>
                    <ListBoxItem Content="Item 3"></ListBoxItem>
                    <ListBoxItem Content="Item 4"></ListBoxItem>
                    <ListBoxItem Content="Item 5"></ListBoxItem>
                </ListBox>
            </StackPanel>
        </StackPanel>
    </Grid>
</UserControl>
```

此时界面的效果如下图所示。



通过这个示例可知，Silverlight 的版式面板可以任意嵌套。除了官方控件集中的这三个面板外，还可以通过从 Panel 类进行派生来编写自定义的面板。

24.10 Silverlight 与 HTML 的结合

Silverlight 能够与传统的 Web 内容(即 HTML)结合。Silverlight 组件能够访问“文档对象模型”(DOM)，而网页中的 JavaScript 也能够访问 Silverlight 组件。

为在 Silverlight 中访问 HTML 元素，需要使用 System.Windows.Browser.HtmlDocument 类。HtmlDocument 对象随当前网页的 HTML 文档对象模型一起加载。在程序中，我们可以通过 System.Windows.Browser.HtmlPage 的静态成员 Document 来访问 HtmlDocument 对象。如果要获取 HTML 文档中指定的元素，可以调用 HtmlDocument.GetElementByName 或 HtmlDocument.GetElementById 方法。在获得 HTML 元素后，便可以以编程方式访问并使用标签中的数据。此外，我们还可以将托管的事件与 HTML 元素(如按钮)进行关联，并在托管代码中处理这些事件，而不必使用 JavaScript。

例如，下面这段 HTML 代码中包含几个带 id 特性的标签：

```
<input id="input" type="text"/>
<input id="submit" type="button" value="Submit"/>
<div id="output"/>
```

对 DOM 的访问非常简单。在 Silverlight 组件代码中，只需要为每个要访问的 DOM 中的标签声明一个 HtmlElement 变量即可。HtmlElement 类在 System.Windows.Browser 命名空间中定义。使用 HtmlDocument 类(通过 HtmlPage 类获取)便可以访问标签。GetProperty、SetProperty 和 AttachEvent 是 HtmlElement 类中最常用的 3 个方法。下面的代码演示了如何通过 HtmlDocument 类来访问具有标识的标签。它也展示了如何获取和设置标签属性，以及如何附加事件(例如，为按钮添加单击事件的处理程序)。相关的 HTML DOM 辅助类都位于 System.Windows.Browser 命名空间中。


```
namespace UseDOM
{
    public partial class MainPage : UserControl
    {
        HtmlElement _input;
        HtmlElement _output;
        HtmlElement _submit;

        public MainPage()
        {
            InitializeComponent();

            HtmlDocument document = HtmlPage.Document;
            _input = document.GetElementById("input");
            _output = document.GetElementById("output");
            _submit = document.GetElementById("submit");

            _submit.AttachEvent("onclick", OnSubmit);
        }

        void OnSubmit(object sender, HtmlEventArgs ea)
        {
            string input = (string)_input.GetProperty("value");
            string output = "you typed: " + input;
            _output.SetProperty("innerHTML", output);
        }
    }
}
```

为在 JavaScript 中访问托管类型，需要为这些类型添加[ScriptableType]特性，并为每个在脚本中访问的成员添加[ScriptableMember]特性。System.Windows.HtmlPage 类包含两个成员，分别用于注册要在脚本中使用的托管类型：RegisterCreatableType 注册的类型允许在脚本中创建该类型的实例，而 RegisterScriptableObject 注册的类型只允许在脚本中访问该类型现有的实例。要想在脚本代码中创建类型的实例或查找现有的实例，可以使用 Silverlight 插件的功能。在获得实例后，便可以像普通 JavaScript 对象一样来使用它。

24.11 动 画

在显示区域中添加动画是 Silverlight 最引人注目的功能之一。Silverlight 为可视树中的 UI 元素添加动画提供了统一的方式。

为理解 Silverlight 动画的工作方式，需要理解 Silverlight 中的“依赖项属性”(dependency property)的概念。许多 Silverlight 类都以依赖项属性的形式暴露其属性。拿 Button 来说，Silverlight 并没有以类成员的形式存储 Height 和 Width 这样的属性，而是将这些属性实际的数据存储在由 Silverlight 管理的后端存储中。这里所谓后端存储是指提供属性数据存储空间的特殊的底层类或基础设施。

每个类都有自己的空间来存储数据成员和类实例的关键数据。虽然这种存储数据成员的方式可能有些让人困惑，但这种机制带来了两个优势。首先，这样存储数据能够减少许多实例数据，有助于性能的改善(例如，降低垃圾回收的负担)。另一个优势在于，如果值(出于某种原因)发生了变化，Silverlight 就能够触发事件(如通知界面重新绘制自身)。这样便减少了为处理值的变化而要编写的事件处理程序的数目。

动画的运用最能凸显依赖项属性的优势。所有 Silverlight 动画一般都以这种方式工作。

1. 选择要变化的属性(只能通过依赖项属性添加动画)。
2. 选择一种动画驱动程序(不同的驱动程序针对不同数据类型，这些数据类型包括双浮点数、整数、布尔值、颜色等)。
3. 将动画驱动程序连接到该属性。
4. 创建一个容纳动画的演示图板(storyboard)，并添加动画。
5. 启动演示图板。

在演示图板启动后，它会使用动画驱动程序来修改目标属性。由于目标属性是依赖项属性，因而变化会立即显现。例如，如果为按钮的 Width 属性添加动画，则会立即看到按钮宽度的变化。

我们可以配置 Silverlight 动画的不同特性，如：

- 动画的持续时间
- 是否自动撤销动画
- 动画的加速度曲线
- 动画重复的次数或所持续的时间

最后，Silverlight 也提供了更为复杂的动画。例如，通过关键帧动画和样条(spline)动画来实现非线性动画效果。

下面的练习将演示 Silverlight 动画的使用。我们将对一组图片添加动画，使其长和宽增大，同时使其变为不透明，最后恢复其原始状态。这个练习将演示几方面内容，其中包括如何在代码旁置类中使用动画，如何对 UI 元素进行可视变换，以及如何将资源以二进制的形式包装在 Silverlight XAP 容器中。

➤ 对图片的 RenderTransform 和 Opacity 属性添加动画

1. 在 Visual Studio 中，创建一个“Silverlight 应用程序”，将其命名为 SilverlightAnimations。
2. 为名为 LayoutRoot 的 Grid 添加 5 个 ColumnDefinition 和 10 个 RowDefinition。下面给出了所要添加的 XAML 代码。除此以外，这个练习所要编写的其他代码都在代码旁置类中。

```

<UserControl x:Class="SilverlightAnimations.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"

    DesignWidth="600" DesignHeight="300"

    <Grid x:Name="LayoutRoot"
        Background="White">
        <Grid.RowDefinitions>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>
    </Grid>
</UserControl>

```

3. 下面我们添加一些国旗的图片来作为应用动画的对象。配套资源中本章的示例包含 50 个国旗的 JPG 图片。在“解决方案资源管理器”中，右键单击“SilverlightAnimations”项目节点，选择“添加”|“文件夹”。将这个文件夹命名为 Images。右键单击这个文件夹，选择“添加”|“现有项”。在配套资源中找到第 24 章对应的解决方案(目录是按章节组织的)，选择 Images 文件夹下的 50 个国旗图片。这 50 个国旗图片的格式均为 JPEG。将这些图片添加到项目中就可以加载它们了。生成项目时，一般将采用资源(resource)的形式来包含它们。这些图片最终会被传输到客户端计算机。
4. 添加一个管理这些国旗的类。注意，这些国旗图片的文件名采用的形式为<国家名称缩写>.jpg。这个类只是一个列表，包含国名的缩写。右键单击“SilverlightAnimations”项目，选择“添加”|“新建项”，然后选择“类”(以便通过 Visual Studio 生成这个类基本的代码)。将这个类命名为 States。单击“添加”后，Visual Studio 会在编辑器中打开该类的代码。
5. 使这个 States 类派生自泛型类 List，将类型参数指定为 string。然后，添加国家名称的缩写。(不妨从配套资源的示例中复制过来，不必人工输入)


```
public class States: List<string>
{
public States()
{
    Add("AL");
    Add("AK");
    Add("AZ");
    Add("AR");
    Add("CA");
    Add("CO");
    // 其他国名.....
    Add("WA");
    Add("WV");
    Add("WI");
    Add("WY");
}
}
```

6. 打开 MainPage.xaml.cs, 为构造函数 MainPage 编写加载图片的代码, 为每个单元格添加包含国家名称的 TextBlock。创建 States 类的实例, 以便获取国家名称的缩写。对 LayoutRoot 中的行和列进行遍历。为网格中的每个单元格创建一个显示国名缩写的 TextBlock。从 States 类的实例获取国名缩写, 并将其赋给 TextBlock 的 Text 属性。此外, 我们还可以设置 TextBlock 的其他属性(如 FontSize 和 FontFamily)。将每个 TextBlock 添加到 LayoutRoot 的子控件集合中。通过静态方法 Grid.SetRow 和 Grid.SetColumn 来设置 TextBlock 在网格中的位置。

随后, 创建 Image 类的实例。为这个 Image 实例添加 ScaleTransform。我们稍后将对图片添加动画。根据当前的国名缩写创建国旗图片(在 Images 文件夹中)的 URI。最后, 将 Image 的 Opacity 设置为 0.4, 将 Image 对象添加到 LayoutRoot 中, 并将其放到网格中的当前位置。

```
public MainPage()
{
    InitializeComponent();

    States states = new States();
    int stateNumber = 0;

    for (int column = 0; column < 5; column++)
    {
        for (int row = 0; row < 10; row++)
        {
            // get the state abbrev
            string stateAbbrev = states[stateNumber];

            TextBlock theTextBlock = new TextBlock();
            theTextBlock.FontSize = 22;
            theTextBlock.Text = stateAbbrev;
            theTextBlock.TextAlignment = TextAlignment.Center;
```

```
theTextBlock.VerticalAlignment = VerticalAlignment.Center;
LayoutRoot.Children.Add(theTextBlock);
Grid.SetRow(theTextBlock, row);
Grid.SetColumn(theTextBlock, column);

// Add an image control to the grid
Image theImage = new Image();
ScaleTransform st = new ScaleTransform();
st.ScaleX = 1; st.ScaleY = 1;
theImage.RenderTransform = st;
Uri uri =
new Uri("Images/" + stateAbbrev + ".jpg", UriKind.Relative);

theImage.Source = new BitmapImage(uri);
theImage.Margin = new Thickness(10);
theImage.Opacity = .4;
LayoutRoot.Children.Add(theImage);

// then position it
Grid.SetRow(theImage, row);
Grid.SetColumn(theImage, column);

stateNumber++;
    }
}
}
```

7. 运行这个程序。其效果应如下图所示。



8. 编写一个方法为国旗添加缩放动画。分别为 ScaleTransform 的 ScaleX 和 ScaleY 属性以及 Image 的 Opacity 属性创建 DoubleAnimation。将两个针对 ScaleTransform 的 DoubleAnimation.From 属性都设置为 1，将 DoubleAnimation.To 属性都设置为 5。将针对 Opacity 的 DoubleAnimation.From 设置为 0.4，将 DoubleAnimation.To 设置为 1。将这 3 个 DoubleAnimation 的 AutoReverse 属性都设置为 true，以便在动画播放完毕后恢

复播放动画之前的状态。创建一个 Storyboard 来容纳动画。通过 StoryBoard 的 SetTarget 和 SetTargetProperty 方法将这 3 个动画和目标元素相关联。最后，将动画添加到 Storyboard。

为在程序执行期间获取错误信息，可将这段代码置于 try/catch 块中。

```
public partial class MainPage : UserControl
{
    void AnimateImage(Image image)
    {
        try
        {
            DoubleAnimation scaleXAnimation = new DoubleAnimation();
            scaleXAnimation.AutoReverse = true;
            DoubleAnimation scaleYAnimation = new DoubleAnimation();
            scaleYAnimation.AutoReverse = true;
            DoubleAnimation opacityAnimation = new DoubleAnimation();
            opacityAnimation.AutoReverse = true;

            scaleXAnimation.From = 1;
            scaleXAnimation.To = 5;
            scaleYAnimation.From = 1;
            scaleYAnimation.To = 5;
            opacityAnimation.From = .4;
            opacityAnimation.To = 1;

            Storyboard sb = new Storyboard();
            Storyboard.SetTarget(scaleXAnimation, image.RenderTransform);
            Storyboard.SetTargetProperty(scaleXAnimation,
                new PropertyPath("ScaleX"));

            Storyboard.SetTarget(scaleYAnimation, image.RenderTransform);
            Storyboard.SetTargetProperty(scaleYAnimation,
                new PropertyPath("ScaleY"));

            Storyboard.SetTarget(opacityAnimation, image);
            Storyboard.SetTargetProperty(opacityAnimation,
                new PropertyPath("Opacity"));

            sb.Children.Add(scaleXAnimation);
            sb.Children.Add(scaleYAnimation);
            sb.Children.Add(opacityAnimation);

            sb.Begin();
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.WriteLine(ex.Message);
        }
    }
}
```


9. 为 MainPage 类添加 OnMouseLeftButtonDown 事件处理程序来为图片应用动画。该处理程序将直接与图片关联，因而发送者(sender)就是图片本身。将发送者转换为 Image(图片)类型，调用 Animate 方法，并传入图片：

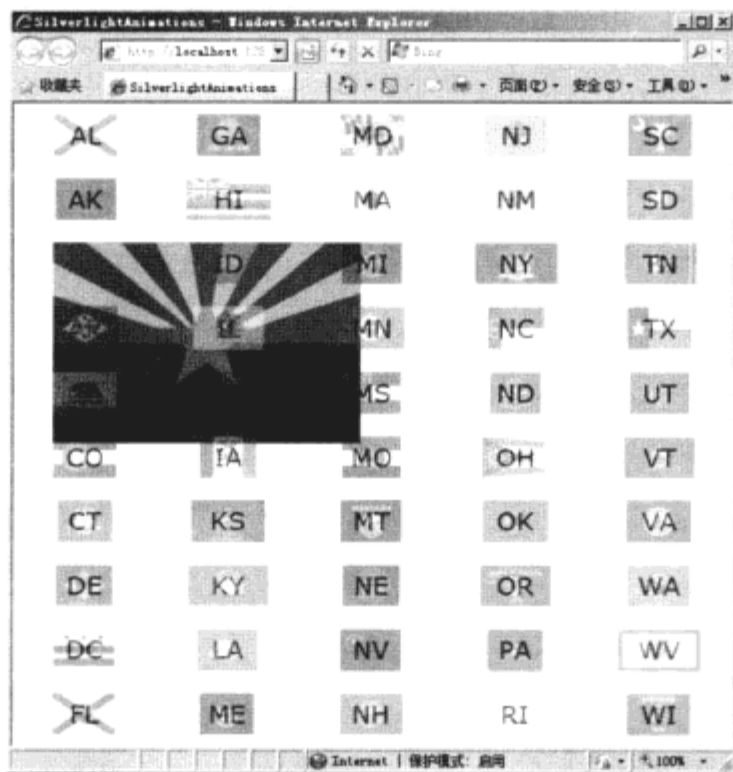
```
public partial class MainPage : UserControl
{
    void OnMouseLeftButtonDown(object sender, MouseButtonEventArgs ea)
    {
        Image image = sender as Image;
        AnimateImage(image);
    }
}
```

10. 最后，用这个处理程序订阅所有 Image 的 MouseLeftButtonDown 事件：

```
public MainPage()
{
    // 其他初始化代码
    for (int column = 0; column < 5; column++)
    {
        for (int row = 0; row < 10; row++)
        {
            // 其他代码
            Image theImage = new Image();

            theImage.MouseLeftButtonDown += OnMouseLeftButtonDown;
            // 其他代码
        }
    }
}
```

11. 再次运行程序。这次所有国旗都具有了动画效果。单击某个国旗后，其长和宽会以动画形式变为原来的 5 倍，同时透明度降低，然后又恢复到原始状态(如下图所示)。





24.12 WCF 服务与 Silverlight

不难发现，Silverlight 代表了一种全新的 Web 编程模型。传统的 ASP.NET 编程所管理的是为客户端生成 HTML 的 ASP.NET 控件。ASP.NET 具有管理会话状态和视图状态(针对 HTTP 协议上的 HTML)的完整基础设施。

Silverlight 内容一般不与网站的其他部分保持连接状态。Silverlight 能够通过“文档对象模型”与客户端的 HTML 结合，进而可以在 Silverlight 内容被发送到客户端之前传入参数。不过，Silverlight 并未包含直接连接网站其他部分的支持。为与网站的其他部分进行通信，Silverlight 通常要使用网站提供的 Windows Communication Foundation(WCF)服务。

下面的练习将演示 Silverlight 如何调用 WCF 服务。我们将为 Silverlight 网站添加一个启用 Silverlight 的 WCF 服务，通过 Web 服务的形式暴露一个商品集合。Silverlight 控件会对这个商品集合进行查询，并在用户从列表框中选择某个商品后从中获取详细信息。这个练习将演示如何创建 WCF 服务，如何通过 WCF 服务暴露数据，以及如何对 ListBox 和 TextBlock 进行数据绑定。

► 通过 WCF 与 ASP.NET 网站交互

1. 创建一个“Silverlight 应用程序”，将其命名为 SilverlightAndWCF。通过 Visual Studio 为这个 Silverlight 应用程序创建一个用于测试的 ASP.NET Web 项目。
2. 在 Web 项目中添加一个“启用了 Silverlight 的 WCF 服务”，将其命名为 ProductsService。
3. 在 Web 项目中创建一个名为 ProductInfo 的类，我们用它来保存每个产品的信息。在“解决方案资源管理器”中右键单击 SilverlightAndWCF 项目，选择“添加”|“新建项”。选择“类”模板，将这个类命名为 ProductInfo。该类应包含商品的名称、描述和价格。由于 Silverlight 客户端要通过数据绑定来使用这些成员，因而它们应以属性的形式暴露出来。为这个类添加 DataContract 和 DataContractFormat 特性，以便这个类能够通过 WCF 服务暴露出来。为每个属性成员添加 DataMember 特性，以便这些成员能够在代理(为在客户端使用而自动生成的类)中以属性的形式出现。为使用这些特性，需要添加 System.Runtime.Serialization 和 System.ServiceModel 命名空间。请不要忘记，所有这些任务都要在 Web 项目中完成。

```
[DataContract]
[DataContractFormat]
public class ProductInfo
{
    string product;
    [DataMember]
    public string Product
    {
        get { return product; }
```

```

        set { product = value; }
    }

    string description;

    [DataMember]
    public string Description
    {
        get { return description; }
        set { description = value; }
    }

    double price;

    [DataMember]
    public double Price
    {
        get { return price; }
        set { price = value; }
    }
}

```

4. 创建一个 ProductInfo 的集合(派生自泛型类 List)。这个类可以通过 Visual Studio 创建，也可以直接在 ProductInfo.cs 文件中添加。^①通过构造函数向集合添加一些 ProductInfo 对象。这些产品可以是任意的(由于笔者是一个吉他爱好者，因而这里添加了一些吉他产品)。

```

public class Products : List<ProductInfo>
{
    public Products()
    {
        ProductInfo productInfo = new ProductInfo();

        productInfo = new ProductInfo();
        productInfo.Product = "Solidbody";
        productInfo.Description = "Flame maple top" +
            "mahogany body. Rosewood fingerboard. " +
            "One piece mahogany neck. Two humbucking " +
            "pickups. With case.";
        productInfo.Price = 2500.00;
        Add(productInfo);
        //后面还有其他商品
    }
}

```

5. 打开 ProductsService.svc.cs(还是在 Web 项目中)。创建一个 Products 类的静态实例。为这个服务添加两个方法：一个用于获取整个 ProductInfo 的列表，另一个用于查询指

① 译者注：在配套资源中，刚刚添加的这两个类都位于 SilverlightAndWCF.Web\ProductsService.svc.cs 文件中。

定的商品。ServiceContract 特性通常包含命名空间。这里为了简明略去了这个命名空间。^①

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements
    (RequirementsMode =
        AspNetCompatibilityRequirementsMode.Allowed)]
public class ProductsService
{
    static Products products = new Products();

    [OperationContract]
    public Products GetProducts()
    {
        return ProductsService.products;
    }

    [OperationContract]
    ProductInfo GetProduct(string key)
    {
        return ProductsService.products.Find(
            delegate(ProductInfo productInfo)
            {
                if (productInfo.Product == key)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        );
    }
}
```

6. 编辑 MainPage.xaml 文件中的 XAML 来进行布局。在 LayoutRoot 中添加 4 个行定义和 3 个列定义。使前三行的大小适应其内容(即将 RowDefinition 的 Height 属性设置为 Auto)，使第一列自动调整大小(将 ColumnDefinition 的 Width 属性设置为 Auto)。

```
<Grid x:Name="LayoutRoot" Background="White">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
```

① 临时译注：这里删掉最后一句，因为 System.ServiceModel.Activation 命名空间是默认添加的。

```

        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>

```

```
</Grid>
```

7. 下面为网格添加内容。

- a. 在第一行第一列(0 行 0 列)添加一个 TextBlock, 用作 Details(详细信息)区域的标题。该控件应跨越两列。
- b. 在第一行第三列添加一个 TextBlock, 用作列表框的标题。
- c. 在第一列添加 3 个用作标签的 TextBlock。第二行的 TextBlock 的 Text 属性设置为“Product:”, 第三行的 TextBlock 的 Text 属性设置为“Price:”, 第四行的 TextBlock 的 Text 属性设置为“Description:”。
- d. 在第二列添加 3 个用于显示商品信息的 TextBlock。通过扩展标记 Binding 将第二行的 TextBlock 的 Text 属性绑定到 ProductInfo 的 Product 属性, 即通过下面这种语法来设置 Text 属性: Text="{Binding Product}”。类似地, 依次将下面两个 TextBlock 的 Text 属性绑定到 ProductInfo 的 Price 和 Description 属性。
- e. 最后, 在第二行第三列添加一个 ListBox, 使其跨越 3 行。将这个列表框命名为 theListBox。为其 SelectionChanged 事件添加一个处理程序。这个处理程序可以在输入标签中的 SelectionChanged 时, 通过 Visual Studio 的提示来创建。将 DisplayMemberPath 属性设置为 Product。这样, 当 ProductInfo 集合绑定到这个 ListBox 后, ListBox 便会显示 Product 属性。

```

<Grid x:Name="LayoutRoot" Background="White">
<!-- 这里是网格行和列的定义 -->
<TextBlock Grid.Row="0"
    Grid.Column="0"
    Grid.ColumnSpan="2"
    FontSize="24"
    Text="Details:" />

<TextBlock Grid.Row="0"
    Grid.Column="2"
    FontSize="24"
    Text="Select Product:" />

<TextBlock Grid.Row="1"
    Grid.Column="0"
    FontSize="18"
    Text="Product:" />

<TextBlock Grid.Row="2"
    Grid.Column="0"

```

```
FontSize="18"
Text="Price:" />

<TextBlock Grid.Row="3"
    Grid.Column="0"
    FontSize="18"
    Text="Description:" />

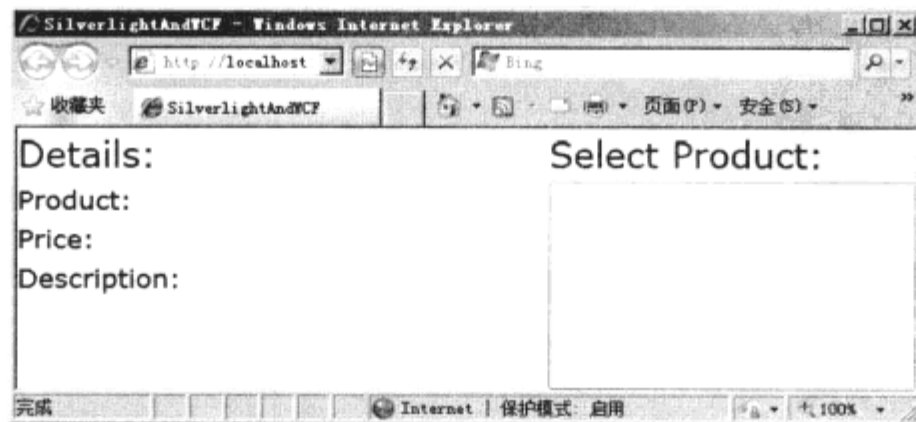
<TextBlock Grid.Row="1"
    Grid.Column="1"
    FontSize="14"
    Text="{Binding Product}"
    Margin="5"/>

<TextBlock Grid.Row="2"
    Grid.Column="1"
    FontSize="14"
    Text="{Binding Price}"
    Margin="5"/>

<TextBlock Grid.Row="3"
    Grid.Column="1"
    FontSize="14"
    Margin="5"
    TextWrapping="Wrap"
    Text="{Binding Description}"/>

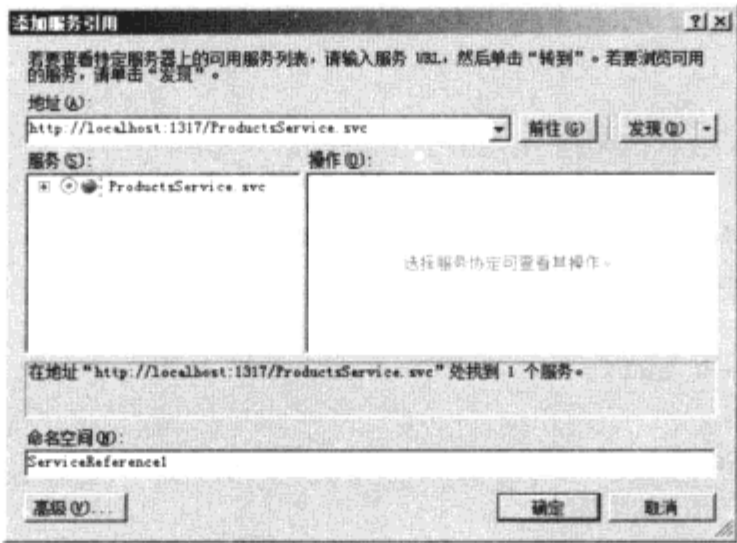
<ListBox x:Name="theListBox"
    Grid.Row="1"
    Grid.Column="2"
    Grid.RowSpan="3"
    DisplayMemberPath="Product"
    SelectionChanged="theListBox_SelectionChanged">
</ListBox>
</Grid>
```

8. 运行这个程序。其效果应如下图所示。



9. 为 SilverlightAndWCF 项目添加 WCF 服务的引用。在“解决方案资源管理器”中右键单击“SilverlightAndWCF”项目节点，选择“添加服务引用”。Visual Studio 会显示下图所示对话框。我们可以通过该对话框在 Silverlight 项目中获取有关服务的信息。

单击右上角的“发现”按钮。Visual Studio 会找到 ProductsService 服务。



- 10. 单击“ProductsService.svc”节点边的加号便可以看到该服务的详细内容。接受默认的命名空间(ServiceRefernce 1)，单击“确定”。Visual Studio 会在 Silverlight 项目中生成一个代理。
- 11. 在 MainPage 的代码旁置类中通过 using 语句添加对该服务的引用。在 MainPage 类中，以成员形式声明一个 ProductsServiceClient 类的变量，并创建其实例。这样，MainPage 便可以与网站进行交互。我们可以通过这个代理来调用服务，进而获取所有商品或查询指定的商品。

```
using SilverlightAndWCF.ServiceReference1;

public partial class Page : UserControl
{
    ProductsServiceClient productsService =
        new ProductsServiceClient();

    public MainPage()
    {
        InitializeComponent();
    }
}
```

支持 Silverlight 的 WCF 代理会以异步方式工作。GetProducts 和 GetProduct 方法会在单独的线程中执行。我们需要为该服务的客户端添加一个事件处理程序，以便在调用完成后获取结果。编写一个名为 GetProductsCompleted 的方法，我们通过它来获取商品集合。第一个参数的类型为 object(发送者)，第二个参数的类型为 GetProductsCompletedEventArgs(在 Visual Studio 生成的代理中定义)。GetProductsCompletedEventArgs 具有一个 Result 属性，代表商品集合。GetProductsCompleted 应将这个集合赋给 ListBox 的 ItemsSource 属性。由于 ListBox 的 DisplayMemberPath 属性被设置为“Product”，因而 ListBox 会显示集合中每个 ProductInfo 的 Product 属性。

此外，再添加一个名为 GetProductCompleted 的方法来获取单个商品的查找结果。第一个参数的类型为 object(发送者)，第二个参数的类型为 GetProductCompletedEventArgs(同样由 Visual Studio 生成)。GetProductCompletedEventArgs 的 Result 为返回的结果

(ProductInfo)。将这个结果赋给 LayoutRoot 的 DataContext 属性。由于 Grid 中的 3 个 TextBox 控件被分别绑定到了 Product、Price 和 Description 属性，因而对应的数据会自动显示。

```
public partial class MainPage : UserControl
{
    void GetProductscompleted(object sender,
        GetProductsCompletedEventArgs ea)
    {
        if (ea.Error == null)
        {
            this.theListBox.ItemsSource = ea.Result;
        }
        else
        {
            System.Diagnostics.Debug.WriteLine(ea.Error.InnerException);
            this.theListBox.Items.Add("Gibson Les Paul Standard");
        }
    }

    void GetProductCompleted(object sender,
        GetProductCompletedEventArgs ea)
    {
        ProductInfo pi = ea.Result as ProductInfo;
        if (pi != null)
        {
            this.LayoutRoot.DataContext = pi;
        }
    }
}
```

12. 在构造函数 MainPage 中将 GetProductCompleted 和 GetProductsCompleted 处理程序与代理 ProductsService 进行关联，并调用 ProductsService.GetProducts 方法来获取商品集合。

实现 ListBox 的 SelectionChanged 处理程序。获取 ListBox.SelectedItem 属性的值，并在调用代理的 GetProduct 方法时将其传入(即传入当前选定项的键)。在服务完成其工作后，结果会通过 GetProductCompleted 方法返回。

```
public partial class MainPage : UserControl
{
    public MainPage()
    {
        InitializeComponent();

        productsService.GetProductsCompleted +=
            GetProductscompleted;

        productsService.GetProductCompleted +=
            this.GetProductCompleted;

        productsService.GetProductsAsync(this);
    }
}
```

```
}

// 这里是异步处理程序

private void theListBox_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
{
    string key =
        (theListBox.SelectedItem as ProductInfo).Product;

    productsService.GetProductAsync(key, this);
}
}
```

13. 运行这个程序。当浏览器打开后，我们可以在右侧看到商品列表。在选定某个商品后，其详细信息会显示在左侧。



24.13 快速参考

目 标	操 作
生成 Silverlight 应用程序	从主菜单中选择“文件” “新建” “项目”。通过“Silverlight 应用程序”模板创建项目
在 Silverlight 组件中管理布局	选择一种版式面板作为 UserControl 的根节点。Grid 能够按行和列对可视元素进行布局。StackPanel 能够从上到下(垂直)或从左到右(水平)对可视元素进行布局。Canvas 允许以绝对(x,y)坐标来定位可视元素
修改页面上控件的属性	在“设计”模式下，选择目标控件，在“属性”窗口中进行编辑
在 Silverlight 组件中使用 HTML 文档对象模型	遵循以下步骤： 1. 通过 ID 特性为目标 HTML 元素命名 2. 通过 HtmlPage.Document 属性获取 HtmlElement。调用其 GetElementById 方法来获取这个 HTML 元素 3. 通过 HtmlElement 的 GetProperty 和 SetProperty 方法来修改元素属性 4. 通过 HtmlElement 的 AttachEvent 方法将托管的事件处理程序与 HTML 元素的事件相关联

续表

目 标	操 作
在 JavaScript 中访问托管代码	用 ScriptableType 特性来标记要脚本化的类型，用 ScriptableMember 特性来标记要脚本化的成员。然后，使用 HtmlPage.RegisterCreatableType 来注册允许在脚本中实例化的对象，通过 HtmlPage.RegisterScriptableObject 来注册在脚本中使用的现有对象。然后在脚本中利用 Silverlight 插件来访问方法和属性
以编程方式访问 ASP.NET 网站	为网站添加启用针对 Silverlight 的 WCF 服务。在 Silverlight 项目中可以通过 Visual Studio 创建的代理方便地访问该服务



第 VI 部分

服务与部署

- ▶ 第 25 章 Windows Communication Foundation
- ▶ 第 26 章 部署

还在为学习 .NET技术发愁吗？350元等于什么？答案就是等于Microsqft .NET技术！等于Microsoft .NET 290 GB的 从入门到精通 实战 视频教程！！ 需要的请联系QQ：2569343569

第 25 章

Windows Communication Foundation

学习目标

- 理解 Windows Communication Foundation(WCF)的作用
- 理解 WCF 架构
- 实现基于 WCF 的服务器
- 实现使用 WCF 服务器的客户端

25.1 分布式计算的复兴

Windows Communication Foundation(WCF)发布于 2006 年,是 Microsoft.NETFramework3(和更高版本)中的三大技术之一——另外两大技术分别是 WindowsWorkflow Foundation(WWF)和 Windows Presentation Foundation(WPF)。这些技术都针对各自的领域重新定义了相应的编程模型。Windows Workflow Foundation 统一了业务 workflow 模型。Windows Presentation Foundation 重新定义了基于 Windows 的桌面应用程序和 Web 应用程序 UI(使用 Microsoft Silverlight)的开发。^①Windows Communication Foundation 统一了 Microsoft 环境中分布式应用程序的编程模型。.NET 3.5 的主要目标就是融合这 3 个独立的编程模型。

为理解分布式计算解决方案的概况,不妨回顾一下早期计算机的连接方式。过去只能使用传统的 RS232 串行连接或调制解调器。经过了若干年,针对 Microsoft 环境中的分布式计算出现了多种协议。例如,Microsoft Windows NT 操作系统支持过一种名为“远程过程调用”(Remote Procedure Call, RPC)的机制,而后被包装成了“分布式组件对象模型”(Distributed Component Object Model, DCOM)。Windows 操作系统还支持套接字编程。在临近 21 世纪之交,Microsoft 发布了“Microsoft 消息队列”(Microsoft Message Queuing, MSMQ)来支持无连接的、排队式的分布式应用程序。当 DCOM 濒临淘汰时,Microsoft 引入了 .NET Remoting。(DCOM 被淘汰的主要原因是它要求周期性地与客户端对象进行通信,通过这种方式来确保连接的持续性,但这同时也降低了可伸缩性,编程模型复杂,需要复杂的配置,安全架构不适应 Internet)最后,为对分布式编程提供更广泛的支持,Microsoft 在 ASP.NET 中通过 ASMX 引入了 XML Web 服务框架(正如在第 23 章中所演示的)。

^① 译者注: Silverlight 已经从桌面开发延伸到了 Web 和移动开发。Windows Phone 7 也将 Silverlight 作为主要开发技术之一。这不仅简化了移动应用程序的开发,而且使应用程序的界面更加丰富和绚丽。

25.2 种类繁多的通信 API

之前提到的每种过时的技术都有各自的优点——尤其是从当时的历史背景来看。不过，由于为实现分布式计算而出现了如此之多的技术，因而导致了种类繁多的应用程序编程接口 (API)。一般而言，采用哪种技术在开发的早期就必须确定下来。早期的分布式技术一般要求应用程序采用特定的传输协议。如果架构决策失误或后期希望采用新的技术，那么即便不是不可能，也是很难实现的。即使能够实现这种转变，那么也有很大可能会在升级开发应用程序、最终用户的认可程度和部署等一系列问题上遇到麻烦。

依赖于这些早期技术还会带来许多编程和配置方面的问题。早期的连接技术与许多通信过程本身的辅助因素有关，而与数据通信无直接联系。例如，早期的分布式系统往往要在设计的早期及分布式系统的实现阶段决定数据格式。为利用“DCOM 远程过程调用”，应用程序需要采用 DCOM 连接协议和数据格式。这迫使管理员开放 135 端口 (DCOM 对象发现端口)，进而造成巨大的安全隐患。使用 .NET Framework，我们可以选择传输协议和数据格式：在默认情况下，可以采用 HTTP 或更为底层的 TCP 来作为连接协议，然后使用 SOAP 或 .NET 二进制格式作为数据格式。然而，即便提供了这种选择余地，使用传统 .NET 远程通信机制的应用程序一般仍要采用某种特定的连接协议和数据格式。对连接协议和数据格式重新进行全新包装并不容易，轻则大量代码需要修改，重则整个应用程序架构需要重新设计。

除了将连接协议和数据格式与分布式系统的实现紧密耦合，在连接两台计算机时还会出现更多问题。此时，事务、安全、可靠性、序列化等问题接踵而至——这些功能不可避免地要嵌在应用程序的代码中 (而不能在以后需要时再添加)。此外，早期的通信技术不能适应目前流行的面向服务架构 (SOA)。对于 SOA，互操作性 (interoperability) 是关键，而这在过去很难实现。

25.3 针对连接型系统的 WCF

WCF 旨在统一之前种类繁多的 Windows 通信 API，为应用程序提供统一的编程模型。与此同时，WCF 还致力于解除分布式系统的通信过程与应用程序本身的耦合。WCF 不仅是提供一种好的建议，它还具有约束性。使用后便会发现，通信的参与者、传输协议和实现之间是被强制分离的。此外，由于 Microsoft 考虑了现有应用程序的需要，WCF 允许从之前的通信技术 (.NET Remoting 或 XML Web 服务) 部分或完全迁移到基于 WCF 的计算。

SOA 正在成为影响现代软件设计的重要因素。SOA 是一种架构上的思想，由端点 (endpoint) 暴露公开的接口，而通过松耦合的端点来构建大型分布式系统。WCF 遵循了标准的 SOA 原则，如在自治的服务之间划设清晰的边界、使服务建立协定和遵循策略 (而非基于类)、使服务以业务流程为核心 (而不是服务本身)，以及使业务模型易于变动。WCF 兼顾了高性能和高互操作性。

WCF 面向通信层，也就是在可分布的应用程序与使应用程序成为分布式的方法之间引入了

一个层次。WCF 是一个独立的层次，通过提供管理安全性、可靠性、并发性、事务、负荷控制(对调用者或方法的吞吐量进行控制)、序列化、错误处理和实例管理等方面的支持，使得分布式应用程序的实现和配置变得更为容易。

WCF 能够通过 SOAP(目前诸多 Web 服务所采用的标准)与 XML Web 服务进行通信。而通过配置和扩展，WCF 也能够采用非 SOAP 格式的消息进行通信，如采用自定义的 XML 或“真正简单的联合发布系统”(RSS)。

WCF 能够感知通信双方是否都是基于 WCF 的，以便采用最优的数据编码方式。消息的结构并未改变(都采用二进制)。WCF 还包含分布式系统经常用到的其他服务，如内建的消息队列。

25.4 WCF 的组成元素

WCF 由几种独立的元素构成：端点、信道、消息和行为。之前的通信技术往往将这几个概念混为一谈，直到 WCF 的出现才真正将它们按不同的实体区分开来。下面将逐一介绍每种元素。

25.4.1 端点

端点(endpoint)定义了 WCF 通信的组织者和接收者。Microsoft 巧妙地通过一个缩写定义了端点——ABC。ABC 分别代表“地址”(Address)、“绑定”(Binding)和“协定”(Contract)，端点就是由这 3 种信息来描述的。

25.4.1.1 地址

地址用于标识端点的网络位置。WCF 端点地址的形式与传输消息的协议有关。WCF 支持完全限定地址和相对地址。例如，下面这行就是一个完全限定的 Internet 协议地址：

```
http://someserver/someapp/mathservice.svc/calculator
```

WCF 也支持通过基地址(base address)和相对地址(relative address)进行相对寻址。基地址与服务关联，WCF 能够根据与基地址相应的地址找到服务。例如，完整的端点地址可以由基地址 `http://someserver/someapp/mathservice.svc` 和相对地址 `calculator` 构成。

25.4.1.2 绑定

WCF 绑定用于指定消息的传输方式。WCF 绑定并不规约关联紧密的传输协议和数据格式(如 DCOM)绑定本身只是一组信息，其中至少包含协议(protocol)、传输(transport)和编码器(encoder)。

25.4.1.3 协定

定义端点的最后一个元素是协定。协定是客户端与服务之间关于服务对客户端所能履行职

责的一种承诺，规定了在服务调用期间将要交换的信息。

WCF 通过带有 `ServiceContract` 特性的 .NET 接口来描述服务的协定。WCF 协定接口中的方法要使用 `OperationContract` 特性进行修饰。WCF 接口也接受数据结构，但这些结构中的数据成员要以带有 `DataMember` 特性的属性的形式暴露出来。

25.4.2 信道

WCF 信道(channel)是消息的传输系统。WCF 定义了协议信道(protocol channel)和传输信道(transport channel)。协议信道允许在独立于协议的前提下添加安全和事务这样的服务。传输信道用于管理端点之间字节的物理传输。WCF 支持的传输信道包括 MSMQ、HTTP、P2P(Point-to-Point)、TCP 和“命名管道”(Named Pipe)等。WCF 通过工厂模式使信道以一致的方式创建。

25.4.3 行为

在 WCF 中，服务的协定定义了服务所能做的事，而服务协定的实现则决定了服务协定的具体功能。不过，分布式系统的特点之一，是其往往要用到某种插件式的功能，并且这些功能没必要与协定的实现耦合。举例来说，为了保护 Web 服务，可能需要对客户端进行身份验证和授权，但这通常不是服务协定的内容。WCF 通过“行为”(behavior)实现了这种插件式的功能。行为是对 SOA 策略的高层抽象，能够根据具体场景进行定制。

行为由特性进行配置，其中最主要的两个是 `ServiceBehavior` 和 `OperationBehavior`。`ServiceBehavior` 和 `OperationBehavior` 特性能够控制服务执行过程中的以下几个方面：

- 身份模拟(impersonation)
- 并发和同步支持
- 事务行为
- 地址筛选和标头处理
- 序列化行为
- 配置行为
- 会话生命周期
- 元数据变换
- 实例生命周期

通过特性来配置服务器的执行非常方便，只需为服务或操作的实现添加适当的特性进行修饰，并设置必要的属性值即可。例如，为要求操作的主调者支持身份模拟，应添加

OperationBehavior 特性, 并将 Impersonation 属性设置为 ImpersonationOption.Require。

25.4.4 消息

实际的消息(message)是 WCF 中最后的元素。WCF 消息根据 SOAP 消息而设计, 由封套(envelope)、标头(header)、主体(body)和寻址信息(addressing information)构成。当然, 消息还包含所要交换的信息。WCF 支持 3 种消息交换模式: 单向(one-way)、请求-响应(request-response)和全双工(duplex)。对于单向模式, 消息只从发送者到接收者。在请求-响应模式下, 消息从发送者到接收者, 接收者应向发送者发送回应信息。发送者在请求后会进行等待, 直到接收者发送响应。对于全双工方式, 在服务执行期间, 服务能够对客户端进行回调。默认的消息交换模式为请求-响应模式。

25.5 WCF 与 ASP.NET

虽然 WCF 应用程序可以由手动编写的服务器程序承载, 但 ASP.NET 已非常合适作为宿主。开发者可以编写自己的“Windows 服务”来作为宿主, 也可以利用现有的“Windows 服务”——Internet 信息服务(IIS)。WCF 能够以两种模式与 ASP.NET 共存于同一台计算机——并行模式(side-by-side mode)和 ASP.NET 兼容模式(ASP.NET compatibility mode)。下面我们来分别了解这两种模式。

25.5.1 并行模式

在并行模式下运行时, 由 IIS 承载的 WCF 服务可以与由 ASPX 文件和 ASMX 文件(还可能有 ASCX 和 ASHX 文件)组成的 ASP.NET 应用程序位于同一处。此外, ASP.NET 文件和 WCF 服务还可以运行于同一个应用程序域(AppDomain)。对于这种模式, ASP.NET 提供了公共的基础服务(例如, 为 WCF 和 ASP.NET HTTP 运行库提供 AppDomain 管理和动态编译)。在默认情况下, WCF 以并行模式与 ASP.NET 在一起运行。

在并行模式下, ASP.NET 运行库只管理 ASP.NET 请求。对 WCF 服务的请求会直接定向到基于 WCF 的服务。虽然 ASP.NET 运行库不参与这种请求的处理, 但这种模式有几个特点。

首先, ASP.NET 与 WCF 服务可以共享 AppDomain 状态, 其中包括静态变量和全局事件。虽然 WCF 与 ASP.NET 共享同一 AppDomain, 但 WCF 是独立运行的——无法使用 ASP.NET 的某些功能。在 WCF 服务中, 最主要的限制可能是没有 HttpContext 这样的对象(尽管 WCF 与 ASP.NET 的运行时管线具有类似的架构)。从架构角度上讲, WCF 能够通过多种不同的协议进行通信, 而不仅限于 HTTP, 因此针对 HTTP 的上下文在许多情况下是没有意义的。

其次, 身份验证和授权会稍显复杂。例如, 如果 ASP.NET 与 WCF 服务采用不同的身份验证机制, 那么即便客户端通过 ASP.NET 的“Forms 身份验证”进行了身份验证, 也可能会因为要调用服务而被要求重新进行身份验证。在这种情况下, 可能要支持两种身份验证和

授权机制。

尽管 WCF 应用程序不会影响 ASP.NET 应用程序，但 WCF 应用程序仍能够访问 ASP.NET 基础设施中的某些部分，如应用程序数据缓存。本章的示例将会介绍一种在 WCF 应用程序中访问 ASP.NET 缓存的方法。

25.5.2 ASP.NET 兼容模式

WCF 旨在统一不同传输协议和宿主环境下的编程模型。然而，在许多情况下，如此灵活且统一的编程模型并不必要。应用程序可能需要用到 ASP.NET 运行库提供的某些服务。在这种情况下，可以使 WCF 运行在 ASP.NET 兼容模式下。在这种模式下，WCF 应用程序可作为 ASP.NET 应用程序中的一员，调用 ASP.NET 所提供的各种功能和服务。

采用 ASP.NET 兼容模式运行的 WCF 服务能够访问 ASP.NET 管线，并历经整个 ASP.NET HTTP 请求生命周期^①。对于这种模式，WCF 包含 IHttpHandler 的实现，它对 WCF 服务进行了包装，以便穿越 ASP.NET HTTP 管线。事实上，在 ASP.NET 兼容模式下运行的 WCF 服务类似于标准的 ASP.NET Web 服务(即 ASMX 文件)。

在 ASP.NET 兼容模式下运行的 WCF 应用程序可以使用当前 HttpContext 对象的所有内容——会话状态、Server 对象、Response 对象和 Request 对象。此外，这种模式下的 WCF 应用程序允许将 Windows 访问控制列表(ACL)与服务的.svc 文件关联来实现某种安全机制。ASP.NET URL 身份验证也对 ASP.NET 兼容模式的 WCF 应用程序可用。由于不用像在并行模式中那样监听服务的请求，因而 ASP.NET 应用程序形式的 WCF 应用程序能够获得良好的可扩展性——在请求的整个生命周期中，请求都由 ASP.NET 管理。

ASP.NET 兼容模式可以针对整个应用程序(通过修改应用程序的 web.config 文件)，也可以针对特定的 WCF 服务的实现。

25.6 编写 WCF 服务

下面的练习将演示 WCF 的工作方式。在第 23 章中，我们通过名为 QuoteService 的 XML Web 服务来为调用该服务的客户端提供名言文本。而这里，我们将通过基于 WCF 的网站来实现相同的服务，而并不使用基于 ASMX 的 Web 服务。通过这个练习，我们可以了解到编写基于 WCF 的服务和客户端需要做些什么，并发现基于 WCF 的服务与基于 ASMX 的服务之间存在的差异(存在于多个方面)。

► 以 WCF 服务的形式实现 QuoteService

1. 创建一个支持 WCF 的 Web 应用程序项目。这个练习将实现一个在任何客户端都能够

^① 译者注：请求生命周期是指请求在 ASP.NET 管线中所经历的所有状态，而这些状态由不同事件表现出来。

访问的 WCF 应用程序，以此来展示开发过程中的细节。启动 Microsoft Visual Studio 2010。单击“文件”|“新建”|“项目”，单击“WCF”节点，选择“WCF 服务应用程序”。将这个网站命名为 WCFQuotesService。下图展示了此时的“新建项目”对话框。



- 查看 Visual Studio 创建的文件。Visual Studio 会生成以下几个文件：IService1.cs、Service1.svc 和 Service1.svc.cs 文件。这几个文件分别针对 WCF 协定(采用.NET 接口类型的形式)和实现协定的类。
- 修改 Visual Studio 生成的文件。将服务的代码文件从 IService1.cs 重命名为 IQuotesService.cs，将 Service1.svc 重命名为 QuotesService.svc。在对 SVC 文件重命名后，Visual Studio 会自动重命名对应的 C#文件。
- 将接口的名称由 IService1 改为 IQuotesService，将实现服务的类的名称由 Service1 修改为 QuotesService。为此，可以利用 Visual Studio 的重构功能。选中要修改的标识符，在文本编辑器中右击，从“重构”子菜单中选择“重命名”。Visual Studio 会确保该标识符在整个项目中的所有位置保持一致。
- 从第 15 章复制 QuotesCollection 类到当前项目(也就是说将 QuotesCollection.cs 文件添加到 WCFQuotesService 项目中)。QuotesCollection.cs 文件位于第 15 章的 UseDataCaching 项目。在“解决方案资源管理器”的项目节点上右击，选择“添加”|“现有项”。导航至第 15 章的 UseDataCaching 项目(可以使用本书配套资源中的示例)。选择 QuotesCollection.cs 文件，并单击“添加”。QuotesCollection.cs 文件将会被复制到当前的 WCF 项目中。
- 借用 Web 服务示例中的 QuotesCollection.xml 和 QuotesCollection.xsd。在“WCFQuotesService”项目的 App_Data 节点上右击，选择“添加”|“现有项”。找到在第 15 章创建的 UseDataCaching 项目，选择 XML 和 XSD 文件。

7. 至此，数据和数据管理部分已就绪。下面我们将服务暴露出来。在这个步骤中，我们要确定服务的协定。首先，创建一个用于传输名言文本的结构。为添加数据和操作协定，打开 IQuotesService.cs 文件，删除 Visual Studio 作为示例而生成的 CompositeType 类。然后，在该位置添加以下 Quote 结构的代码。Quote 结构包含三个成员，分别用于表示名言文本、作者的姓和作者的名。用 DataMember 特性对它们进行修饰，以便将它们以属性的形式暴露出来。

```
[DataContract]
public struct Quote
{
    private String _strQuote;

    [DataMember]
    public String StrQuote
    {
        get { return _strQuote; }
        set { _strQuote = value; }
    }

    private String _strOriginatorLastName;

    [DataMember]
    public String StrOriginatorLastName
    {
        get { return _strOriginatorLastName; }
        set { _strOriginatorLastName = value; }
    }

    private String _strOriginatorFirstName;

    [DataMember]
    public String StrOriginatorFirstName
    {
        get { return _strOriginatorFirstName; }
        set { _strOriginatorFirstName = value; }
    }

    public Quote(String strQuote,
                String strOriginatorLastName,
                String strOriginatorFirstName)
    {
        _strQuote = strQuote;
        _strOriginatorLastName = strOriginatorLastName;
        _strOriginatorFirstName = strOriginatorFirstName;
    }
}
```

8. 为服务建立一个服务协定。在 IQuotesService.cs 文件中，修改接口的定义，使其包含获取和添加单个名言，以及获取所有名言的方法。

```

[ServiceContract]
public interface IQuotesService
{
    [OperationContract]
    Quote GetAQuote();

    [OperationContract]
    void AddQuote(Quote quote);

    [OperationContract]
    DataSet GetAllQuotes();
}

```

```
Using System.Data; // must be added to identify Dataset
```

9. 实现服务协定。打开 QuotesService.svc.cs 文件，添加必要的实现。首先，添加一个方法，将名言加载到内存中的集合，并将这个集合添加到 ASP.NET 缓存。虽然这个应用程序是一个 ASP.NET 应用程序，但 ASP.NET 会在管线的前端处理 WCF 方法的调用。也就是说，这个应用程序不能像一般的 ASP.NET 应用程序那样使用当前的 HttpContext 对象。不过，我们仍可以在 WCF 上下文中通过 HttpRuntime 对象访问缓存。HttpRuntime.AppDomainAppPath 属性包含应用程序的路径，可以用于建立针对名言的 XML 文件的缓存依赖项。为此，应通过 using 关键字引用 System.Web、System.Web.Caching 和 System.Data 这 3 个命名空间。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Web;
using System.Web.Caching;
using System.Data;

namespace WCFQuotesService
{
    // 注意：使用“重构”菜单上的“重命名”命令，可以同时更改代码、svc 和配置文件中的类名“Service1”。
    public class QuotesService : IQuotesService
    {
        QuotesCollection LoadQuotes()
        {
            QuotesCollection quotesCollection;
            quotesCollection =
                (QuotesCollection)
                HttpRuntime.Cache["quotesCollection"];
            if (quotesCollection == null)
            {
                quotesCollection = new QuotesCollection();
            }
        }
    }
}

```



```

        string strAppPath;
        strAppPath = HttpRuntime.AppDomainAppPath;
        string strFilePathXml =
            string.Format("{0}\\App_Data\\QuotesCollection.xml", strAppPath);
        string strFilePathSchema =
            string.Format("{0}\\App_Data\\QuotesCollection.xsd", strAppPath);
        quotesCollection.ReadXmlSchema(strFilePathSchema);
        quotesCollection.ReadXml(strFilePathXml);
        CacheDependency cacheDependency =
            new CacheDependency(strFilePathXml);
        HttpRuntime.Cache.Insert("quotesCollection",
            quotesCollection,
            cacheDependency,
            Cache.NoAbsoluteExpiration,
            Cache.NoSlidingExpiration,
            CacheItemPriority.Default,
            null);
    }
    return quotesCollection;
}

// 下面还有其他代码
}

```

10. 在 QuotesService 类中实现 GetAQuote 方法。调用 LoadQuotes 来获取 QuotesCollection 对象。生成一个介于 0 和集合中名言总数之间的随机数，以此来在集合中选择一条名言。创建一个 Quote 结构的实例，在填充相关字段后将其返回。

```

public class QuotesService : IQuotesService
{
    // 这里是 LoadQuotes 方法

    public Quote GetAQuote()
    {
        QuotesCollection quotesCollection = this.LoadQuotes();
        int nNumQuotes = quotesCollection.Rows.Count;

        Random random = new Random();
        int nQuote = random.Next(nNumQuotes);
        DataRow dataRow = quotesCollection.Rows[nQuote];
        Quote quote = new Quote((string)dataRow["Quote"],
                                (string)dataRow["OriginatorLastName"],
                                (string)dataRow["OriginatorFirstName"]);

        return quote;
    }

    // 下面还有其他代码
}

```

11. 实现 AddAQuote 方法。首先，调用 LoadQuotes 获取 QuotesCollection。在集合

QuotesCollection 中新建一条记录，使用来自客户端的信息(即 Quote 参数)来填充它。通过 HttpRuntime.AppDomainAppPath 属性来构造 QuotesCollection.xml 文件的路径，并通过 QuotesCollection 的 WriteXml 方法来保存 XML 文件。QuotesCollection 之所以有 WriteXml 方法是因为它派生自 System.Data.DataTable 类。由于该文件与文件依赖项关联，因此缓存会被标记为失效。下次使用时，新的名言集合会被加载。

```
public class QuotesService : IQuotesService
{
    // 这里是 LoadQuotes 方法
    // 这里是 GetAQuote 方法

    public void AddQuote(Quote quote)
    {
        QuotesCollection quotesCollection = this.LoadQuotes();

        DataRow dr = quotesCollection.NewRow();
        dr[0] = quote.StrQuote;
        dr[1] = quote.StrOriginatorLastName;
        dr[2] = quote.StrOriginatorFirstName;
        quotesCollection.Rows.Add(dr);

        string strAppPath;
        strAppPath = HttpRuntime.AppDomainAppPath;

        String strFilePathXml =
            String.Format("{0}\\App_Data\\QuotesCollection.xml", strAppPath);
        String strFilePathSchema =
            String.Format("{0}\\App_Data\\QuotesCollection.xsd", strAppPath);

        quotesCollection.WriteXmlSchema(strFilePathSchema);
        quotesCollection.WriteXml(strFilePathXml);
    }
}
```

12. 实现 GetAllQuotes 方法。加载名言，创建一个 DataSet 对象，将名言集合添加到这个 DataSet 中，然后将其返回。

```
Public class QuotesService : IQuotesService
{
    // 这里是 LoadQuotes 方法
    // 这里是 GetAQuote 方法
    // 这里是 AddQuote 方法

    public DataSet GetAllQuotes()
    {
        QuotesCollection quotesCollection = LoadQuotes();
        DataSet dataSet = new DataSet();
        dataSet.Tables.Add(quotesCollection);
        return dataSet;
    }
}
```

```
}  
}
```

13. 查看 web.config 文件。ASP.NET 4 在很大程度上简化了 WCF Web 服务的配置，而 ASP.NET 的早期版本(如 3.5)要求手动添加有关端点的信息。虽然配置得到了简化，但我们仍可以修改绑定信息并添加服务行为(如管理服务的安全性)。在这个实例中，Visual Studio 提供的默认设置完全可用，不需要修改。

这个练习演示了如何创建由 ASP.NET 承载的、能够在任意位置(当然是 Internet 服务可用的地区)调用的 WCF 服务。在许多情况下，编写 WCF 服务与编写传统的 ASP.NET Web 服务类似。不过，由于这里演示的这种 WCF 服务运行于 ASP.NET 并行模式下，因而无法使用 HttpContext 对象(该对象对于一般 ASP.NET 应用程序都可用)。然而，这往往不是障碍。通过 HttpRuntime 对象，我们仍可以使用许多 ASP.NET 运行库的主要服务(如缓存)。此外，WCF 还支持 ASP.NET 兼容模式。

25.7 WCF 客户端的构建

如果没有客户端去调用，WCF 服务也就失去意义了。本节将演示如何构建一个使用 Quotes 服务的客户端应用程序。在这个练习中，你将会发现，通过 Visual Studio 可以非常方便地引用服务。此外，这个练习还将演示如何以同步和异步方式调用 WCF 服务。

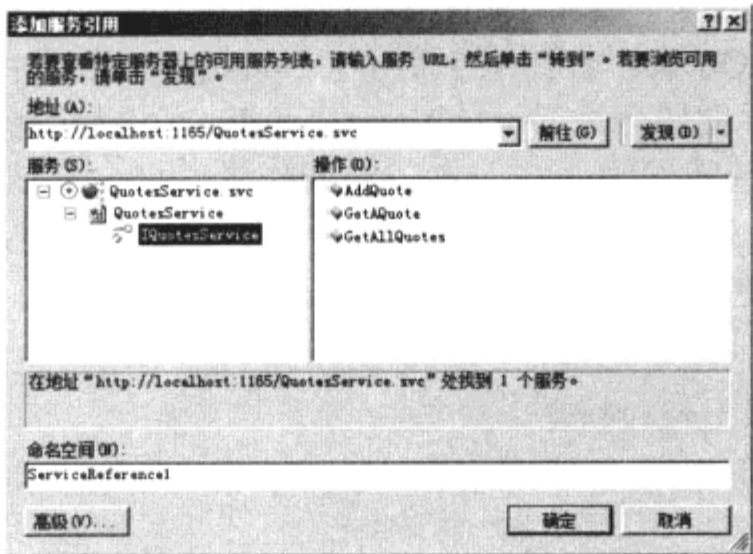
➤ 构建 QuotesService 客户端

1. 启动 Visual Studio 2010。打开 WCFQuotesService 解决方案，在这个解决方案中添加一个“控制台应用程序”，将其命名为 ConsumeQuotesService(如下图所示)。



2. 在“解决方案资源管理器”中右键单击“ConsumeQuotesService”项目节点，选择“添加服务引用”，添加对 WCFQuotesService 的引用。在“添加服务引用”对话框中，单击“发现”按钮。在左侧窗格中选择 QuotesService.svc 文件，并展开与之关联的树。稍等片刻，该对话框会显示该服务的服务协定。展开 QuotesService 树便会看到

IQuotesService 协定。注意，该对话框默认的命名空间为 ServiceReference1。此时的对话框如下图所示。先不要单击“确定”按钮。



- 3. 单击“高级”按钮。选择“生成异步操作”选项，以便使 Visual Studio 生成异步的代理方法。
- 4. 单击“确定”来添加服务引用。Visual Studio 会在 ConsumeQuotesService 项目中生成一个名为 ServiceReferences 的目录，并将有关服务的信息生成为 XML、XSD 和 WSDL 文件。此外，我们还会得到用于调用服务的代理类(在默认情况下，该代理类位于 Reference.cs 文件中)。
- 5. 尝试调用 GetAQuote 操作。针对 WCF 服务生成的代理方法在使用时可能会稍显复杂，但这样做相对高效，而且比我们自己手动逐一建立要好得多。首先，创建 QuotesServiceClient 类(Visual Studio 在添加服务引用时生成的)的实例。调用 QuotesServiceClient 的 GetAQuote 方法，其结果会以 ServiceReference1.Quote 结构的形式返回。将这个结果输出到控制台。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote
            ServiceReference1.QuotesServiceClient quotesServiceClient =
                new ServiceReference1.QuotesServiceClient();

            ServiceReference1.Quote quote = quotesServiceClient.GetAQuote();

            Console.WriteLine("Getting a single quote: " + quote.StrQuote);
            Console.WriteLine();
        }
    }
}
```

```

    }
}
}

```

6. 下面我们调用 `AddAQuote`。该方法的调用与 `GetAQuote` 的调用类似。不过，`AddAQuote` 方法需要传入参数。创建一个 `Quote`(Visual Studio 在添加服务引用时生成的)的实例。找一个简练的名言(或随便输入些文本)，连同作者姓名一起赋给 `Quote` 对象。我们可以使用同一个 `QuotesServiceClient` 实例来调用 `AddAQuote`，传入 `Quote` 对象。稍后在调用 `GetAllQuotes` 时便会发现刚刚添加的这个名言

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote
            ...

            // Now add a quote
            ServiceReference1.Quote newQuote = new ServiceReference1.Quote();
            newQuote.StrQuote = "But to me nothing - the negative, the empty" +
                "- is exceedingly powerful.";
            newQuote.StrOriginatorFirstName = "Alan";
            newQuote.StrOriginatorLastName = "Watts";

            quotesServiceClient.AddQuote(newQuote);

            Console.WriteLine("Added a quote");
            Console.WriteLine();
        }
    }
}

```

7. 下面我们来调用 `GetAllQuotes`。具体方式与之前相同。通过 `QuotesServiceClient` 调用 `GetAllQuotes`。`GetAllQuotes` 会返回一个包含所有名言的 `DataSet` 对象，因此我们需要声明一个 `DataSet` 变量。在 `DataSet` 对象返回后，将其内容打印到控制台。注意，为使编译器能够识别 `DataSet` 类，需要添加 `System.Data` 命名空间。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;

namespace ConsumeQuotesService
{

```

```

class Program
{
    static void Main(string[] args)
    {
        // Get a single random quote
        ...

        // Now add a quote
        ...

        // Now get all the quotes
        DataSet dataSet = quotesServiceClient.GetAllQuotes();
        DataTable tableQuotes = dataSet.Tables[0];

        foreach (DataRow dr in tableQuotes.Rows)
        {
            System.Console.WriteLine(dr[0] + " " +
                                      dr[1] + " " + dr[2]);
        }
    }
}

```

8. 准备以异步方式调用 `GetAQuote`。由于前面添加服务引用时的设置，Visual Studio 生成的代理会支持异步方法调用。为以异步方式调用 `GetAQuote`，需要实现一个回调方法，在执行完毕后，WCF 会调用此方法。在 `Program` 类中添加一个名为 `GetAQuoteCallback` 的静态方法。该方法的返回值为 `void`，接受一个 `IAsyncResult` 类型的参数。当 WCF 回调此方法时，会通过 `IAsyncResult` 参数返回调用该方法的类的对象——`QuotesServiceClient` 对象。声明一个 `ServiceReference1.QuotesServiceClient` 类型的变量，将 `IAsyncResult` 参数 `AsyncState` 的属性赋给 `ServiceReference1.QuotesServiceClient` 变量。然后，声明一个 `Quote` 类型的变量，调用 `QuotesServiceClient.EndGetAQuote` 方法并传入 `AsyncResult` 参数来获取名言。最后，将这个名言输出到控制台。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;

namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote
            ...

            // Now add a quote

```



```

...

// Now get all the quotes
...
}

static void GetAQuoteCallback(IAsyncResult asyncResult)
{
    ServiceReference1.QuotesServiceClient quotesServiceClient =
        (ServiceReference1.QuotesServiceClient)asyncResult.AsyncState;

    ServiceReference1.Quote quote =
        quotesServiceClient.EndGetAQuote(asyncResult);

    Console.WriteLine(quote.StrQuote);
}
}
}

```

9. 以异步方式调用 GetAQuote。过程非常简单，只需要在 Program 类的 Main 方法中调用 QuotesServiceClient 的 BeginGetAQuote 方法。将刚刚编写的回调函数作为第一个参数传入，将 QuotesServiceClient 对象作为第二个参数传入。另外，调用 System.Console.ReadLine 来暂停主线程的执行，以便等待异步调用执行完毕。

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;

namespace ConsumeQuotesService
{
    class Program
    {
        static void Main(string[] args)
        {
            // Get a single random quote
            ...

            // Now add a quote
            ...

            // Now get all the quotes
            ...

            // Now call GetAQuote asynchronously
            System.Console.WriteLine(
                "Now fetching a quote asynchronously");
            Console.WriteLine();

            quotesServiceClient.BeginGetAQuote(GetAQuoteCallback,

```

```
quotesServiceClient);

Console.WriteLine("Press enter to exit...");
Console.ReadLine();
}

static void GetAQuoteCallback(IAsyncResult asyncResult)
{
    // 简明起见，这里将实现略去
}
}
```

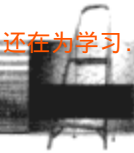
10. 运行这个调用 WCFQuotesService 的控制台应用程序。^①此程序的输出如下图所示。



25.8 快速参考

目 标	操 作
创建基于 WCF 的网站	在 Visual Studio 中，选择“文件” “新建” “网站”，从可用的模板中选择“WCF 服务”。该操作会使 Visual Studio 生成启用 WCF 的网站，并创建默认的协定和实现，这些内容可以进行修改

^① 译者注：这要求当前控制台应用程序是“启动项目”。在“解决方案资源管理器”中，右键单击该项目节点，然后选择“设为启动项目”。



续表

目 标	操 作
创建服务协定	服务协定采用 .NET 接口的形式进行定义，并要求使用 [ServiceContract] 特性对整个接口进行修饰。要以服务的形成暴露的每个成员，均应使用 [OperationContract] 特性进行修饰。可能存在某些通过接口成员传递的数据结构。为使这些数据结构对外可见，需要使用 [DataContract] 特性来进行修饰
实现服务协定	创建一个类，使其派生自定义了服务协定的接口，然后实现相应的成员
以 ASP.NET 应用程序的形式暴露 WCF 服务	确保在 web.config 文件中对服务协定和实现添加声明
创建调用 WCF 服务的客户端	在 Visual Studio 的“解决方案资源管理器”中，使用项目上下文菜单中的“添加服务引用”菜单项来发现和获取服务的元数据。此外，也可以使用“ServiceModel 元数据实用工具”(该工具的程序集的名称为 Svcutil.exe)
对服务的本地执行(如安全性、实例的生命周期和线程)进行定制	通过 ServiceBehaviorAttribute 和 OperationBehaviorAttribute 特性可以控制服务执行，主要包括以下几方面：实例的生命周期、并发和同步支持、配置行为、事务行为、序列化行为、元数据变换、会话生命周期、地址筛选与标头处理，以及身份模拟
在标准的 WCF 应用程序(不在 ASP.NET 兼容模式下运行)中访问 ASP.NET 应用程序缓存	使用 HttpRuntime.Cache 属性



第 26 章

部 署


学习目标

- 了解 Microsoft Visual Studio 对 Web 应用程序部署的支持
- 通过 Visual Studio 创建部署包

ASP.NET 旨在解决 Web 开发中的常见问题。本书的前 25 章关注的是如何使用 ASP.NET 的各种功能，主要介绍了 ASP.NET 的以下几方面：

- 呈现模型，它将整个页面的呈现工作分派到各个服务器端控件完成
- 对数据绑定的支持，它简化了集合的呈现
- 登录控件，它能够满足常见的登录任务
- 会话状态，它能够实现对用户跟踪的管理
- 导航与站点地图的支持
- XML Web 服务及 Windows Communication Foundation(WCF)服务的支持
- 通过母版页和主题使应用程序具有一致的观感
- 通过 Microsoft Silverlight 对富内容的支持
- AJAX 风格的编程支持

在构建应用程序(如为优化公司的流程或扩大市场而构建了丰富的功能)之后，我们需要有效地部署和管理它。这就是本章要讨论的话题——Visual Studio 的不同开发模型对部署策略的影响。此外，本章还将介绍 Web 安装项目的创建。

 **注意** 执行配套资源上针对本章的示例代码要求 IIS 的支持。有关运行本章示例代码的注意事项，请阅读本书“前言”部分“示例代码”小节。

26.1 Visual Studio 网站

Visual Studio 的“文件”|“新建”菜单中有两个选项，一个可以新建 ASP.NET 项目，另一个用于新建网站。本书大部分练习使用的都是 ASP.NET 项目模板(而这也是本章讨论的重点)。Visual Studio 的网站模板可以用来直接针对不同平台(如文件系统、FTP 站点或 Internet

信息服务)进行开发和部署。在讨论部署之前,我们先来了解一下网站模板。

26.1.1 HTTP 网站

在 Visual Studio 中,单击“文件”|“新建”|“网站”,然后在“新建网站”对话框的“Web 位置”组合框中选择“HTTP”,这样创建的网站所采用的是 Visual Studio 中非常早期的 ASP.NET 项目开发模型(比 Visual Studio 2005 还要早的版本)。对于 HTTP 网站模型,Visual Studio 会在 IIS 中创建一个虚拟目录,在开发期间通过 IIS 来监听请求。在这种模式下,解决方案文件(.sln)位于 Visual Studio 项目设置目录下的某个指定目录中,而网站的源文件位于 IIS 的虚拟目录(即\inetpub\wwwroot)。

虽然大多数企业不会采用这种开发模型,但在某些情况下仍可以使用(如个人开发者可以采用这种方式)。如果希望尽量在真实的 IIS 环境下进行开发和调试,便可以使用这种开发模型。在这种情况下,由于网站运行在生产环境下,因而可以对完整的请求路径进行测试(而不仅是 Visual Studio 集成的 Web 服务器所提供的路径)。

26.1.2 FTP 网站

Visual Studio 的“新建网站”模板还提供了创建 FTP 网站的支持。为创建 FTP 网站,应在“新建网站”对话框的“Web 位置”组合框中选择“FTP”。该选项是在 2005 年为通过 FTP 服务器远程管理网站而引入的。例如,如果网站是由远程托管服务来承载的,就可以采用这种方式。FTP 网站提供了一种合理的方式来将文件从开发环境复制到宿主环境。

对于这种网站,Visual Studio 会连接指定的 FTP 服务器并管理其中的内容,因而开发者应对其中的文件和目录有“读”和“写”访问权限。

26.1.3 文件系统网站

“文件系统”网站是对开发者最为友好的一种选择。为创建“文件系统”网站,应单击“文件”|“新建”|“网站”,然后在“新建网站”对话框的“Web 位置”组合框中选择“文件系统”。在开发期间,“文件系统”网站依赖于 Visual Studio 集成的 Web 开发服务器。通过 Visual Studio,我们可以在当前计算机文件系统的任意位置或其他计算机的共享文件夹中创建这种网站。

如果无法访问 IIS,或者没有系统管理员权限,则可以创建基于文件系统的网站项目。这种网站可以完全独立于 IIS 运行。最常见的场景是直接在文件系统上开发和测试网站,然后在需要发布网站时,创建一个 IIS 虚拟目录,并将该虚拟目录指向文件系统中网站所在的物理目录。

在开发 ASP.NET Web 应用程序时，除选择正确的项目类型外，还需要确定是否对 Web 应用程序进行“预编译”。在默认情况下，Visual Studio 不对 Web 应用程序进行预编译。不过，在通过 Visual Studio 完成网站的开发后，也可以为增进性能而对其进行预编译。下面一节就将介绍这方面的内容。

26.2 预编译

在主菜单中选择“生成”|“生成解决方案”时，早期的几个 Visual Studio 版本会自动生成 ASP.NET 应用程序。所有源代码(.vb 和 .cs 文件)都会被编译成一个与项目同名的程序集。这个预编译的程序集位于项目的 bin 目录下，成为要部署的文件之一。ASP.NET 依然支持对应用程序进行预编译。不过在预编译网站时，现在我们有两种选择——使用虚拟路径(对于由 IIS 承载的网站)和使用物理路径(对于文件系统上的网站)。这两种预编译方式分别面向性能和部署，读者应对此有清晰的认识。对网站进行预编译要使用命令行工具。

26.2.1 针对性能的预编译

这种预编译方式也被称为“原地预编译”(precompiling in place)，可以增进现有网站的性能。未经预编译的网站在首次被访问时，ASP.NET 会进行编译，而预编译后的网站可以避免这一过程，这是预编译网站的主要益处。如果网站的代码更新频繁，预编译可以带来略微的性能提升。

为在原地对基于 IIS 的网站进行预编译，需要打开 Visual Studio 的命令行窗口。导航到 .NET 目录(可能位于 Windows\Microsoft.Net\Framework\<版本号>)。在这个目录下有一个名为 aspnet_compiler 的程序。执行 aspnet_compiler，并提供 IIS 中定义的网站名称和 -v 选项。例如，如果 IIS 中有一个名为 MySite 的虚拟目录，则输入以下命令行：

```
aspnet_compiler -v MySite
```

预编译的网站会被置于当前 .NET 安装目录的“Temporary ASP.NET Files”目录下。

如果所要预编译的网站是文件系统网站，并不涉及 IIS 虚拟目录，则可以通过 -p 命令行参数来指定物理路径。该选项会将预编译后的程序集置于目标网站的 bin 目录下。

26.2.2 针对部署的预编译

为部署而进行编译是在对网站进行编译后，将结果输出到某个特定的目录。最终可以将这个目录的内容复制到目标计算机上，也可以用在安装项目中(稍后会介绍安装方面的内容)。在这种情况下，编译器会编译所有原本会在运行时编译的 ASP.NET 源代码，并生成程序集。这些代码包括页面代码、App_Code 目录中的源代码和资源文件。

为进行针对部署的预编译，同样要打开 Visual Studio 的命令提示窗口。通过命令行导航到 .NET 目录。这里要在运行 `aspnet_compiler` 命令执行程序之前，指定虚拟路径或物理路径。在指定目标目录后，指定输入目录。例如，下面的命令会对虚拟目录 `MySite` 中的代码进行编译，并将编译后的结果输出到 `C:\MySiteTarget`：

```
aspnet_compiler -v MySite c:\MySiteTarget
```

如果在这条命令的末尾添加 `-u` 选项，那么编译器只会编译源代码，而不会编译页面代码（这些代码会在运行时以即时方式编译）。

26.3 Visual Studio 2010 的部署支持

许多开发者在开发结束之前并不关心 ASP.NET 应用程序的部署。然而，这样做在部署或重新部署应用程序时便会遇到问题——尤其是在利用云计算而无法直接访问服务器时。Visual Studio 只提供了部分部署支持。直到现在也是如此。

Visual Studio 包含了一些新功能，可以使网站的部署更易于管理，其中包括：

- Web 打包
- 针对部署转换 `web.config` 文件（例如，将开发时的连接字符串替换为生产时所要使用的）
- 数据库的部署/重新部署
- 针对 Web 应用程序的“一键发布”（One-Click Publish）

Visual Studio 2010 引入的这些新的部署特性解决了 ASP.NET 应用程序开发中之前一直被忽视的问题——将代码放置到用户可以浏览的位置。在此之前，部署 ASP.NET 应用程序的最佳方式一般是通过创建安装包来将网站分发到 Web 场中。如果只是复制文件，那么通过安装包完全可以满足要求。然而，安装包并不负责其他任务。例如，在重新部署时，数据库架构的变动就是一大难题。Web 安装项目也不会考虑 ASP.NET 应用程序中 `web.config` 文件的 `debug` 和 `release` 版本的差异。ASP.NET 新的部署打包功能则解决了这些问题。通过这种部署打包功能，我们可以创建 ZIP 文件或文件夹，其中包含将项目部署到 Web 服务器上所必需的所有内容和支持文件。部署包中具体包含以下各项：

- 内容（如 Web 窗体页面、用户控件和 HTML 文件）
- Microsoft SQL Server 数据库架构和数据（如果需要的话）
- IIS 设置（如错误页面设置和应用程序池信息）
- 为支持项目所必需的其他内容（如要安装在全局程序集缓存中的组件、安全证书、注册表设置等）

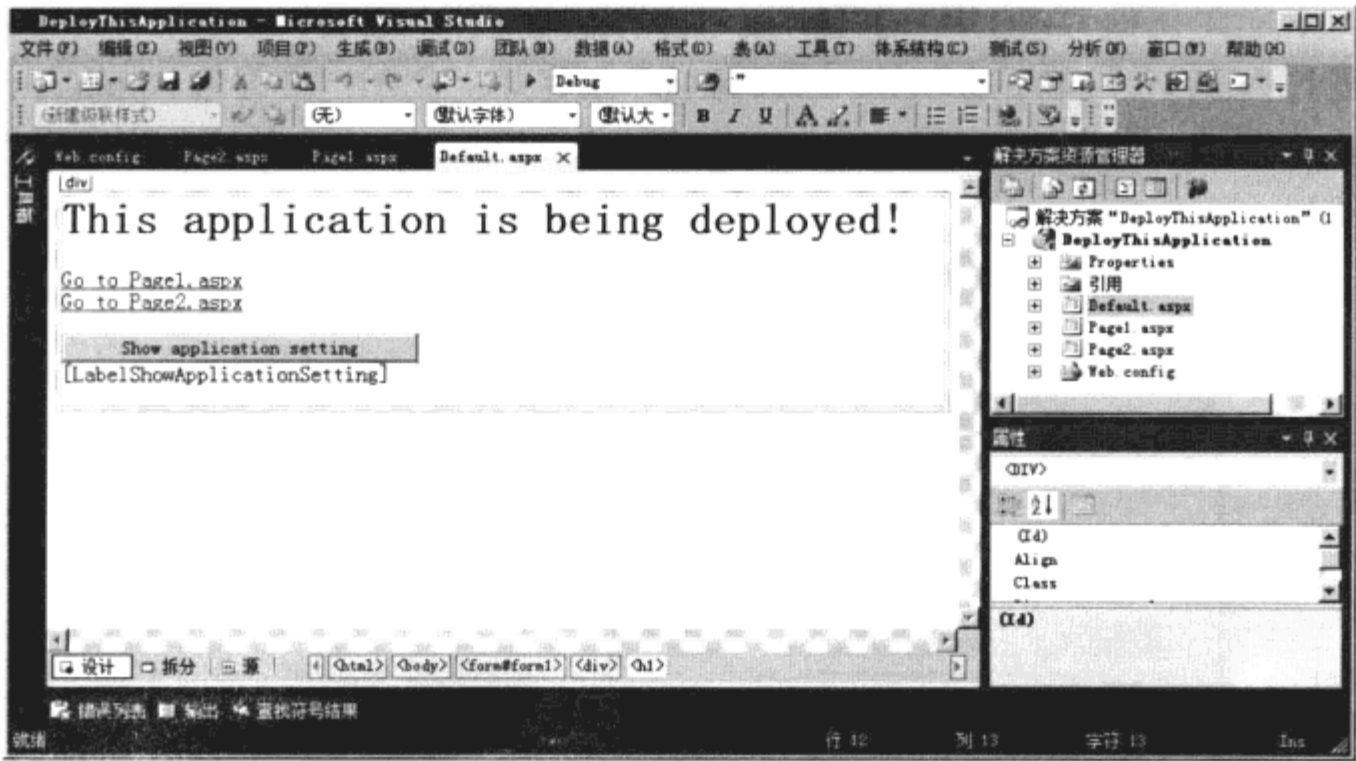
在创建部署包后，我们可以将其复制到 Web 服务器并手动安装（使用 IIS 管理器），或者也

可以使用命令行程序或部署 API 来进行安装(用以实现完全自动化的部署)。

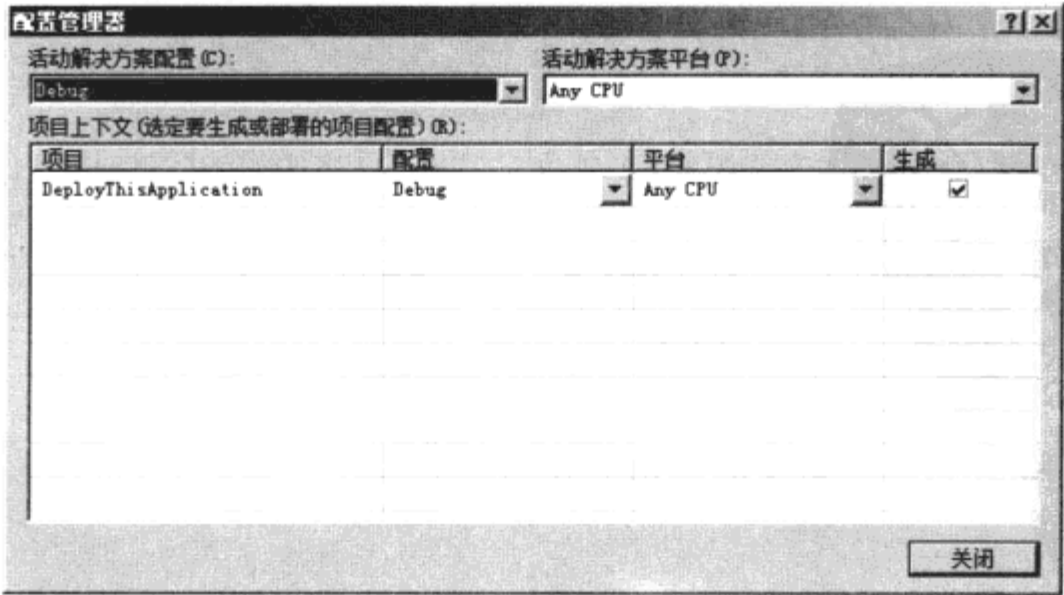
理解这些功能最好的方式是创建一个 ASP.NET 网站，然后部署它。

➤ 为部署而对 ASP.NET 项目打包

- 1. 创建一个“空 ASP.NET Web 应用程序”项目，将其命名为 DeployThisApplication。
- 2. 为这个网站添加一些内容，以便在部署后查看。为简明起见，可以只添加一个 Default.aspx 页面，并在其中添加一个名为 ButtonShowApplicationSetting 的 Button 和一个名为 LabelShowApplicationSetting 的 Label。这些控件使用于演示 Web.config 转换是如何进行的。本书的示例中还包含 Page1.aspx 和 Page2.aspx 页面(见下图)。

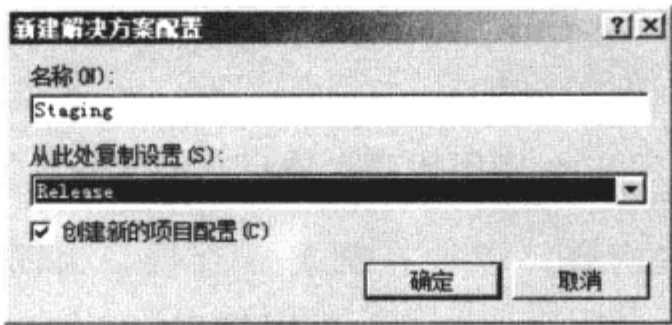


- 3. Visual Studio 创建的应用程序有两个解决方案配置：Debug 和 Release。这里我们另外创建一个名为 Staging 的配置，其中包含程序在 Web 服务器上运行所需要的设置。在“解决方案资源管理器”的解决方案节点上右击，选择“配置管理器”。Visual Studio 会显示下图所示的对话框。

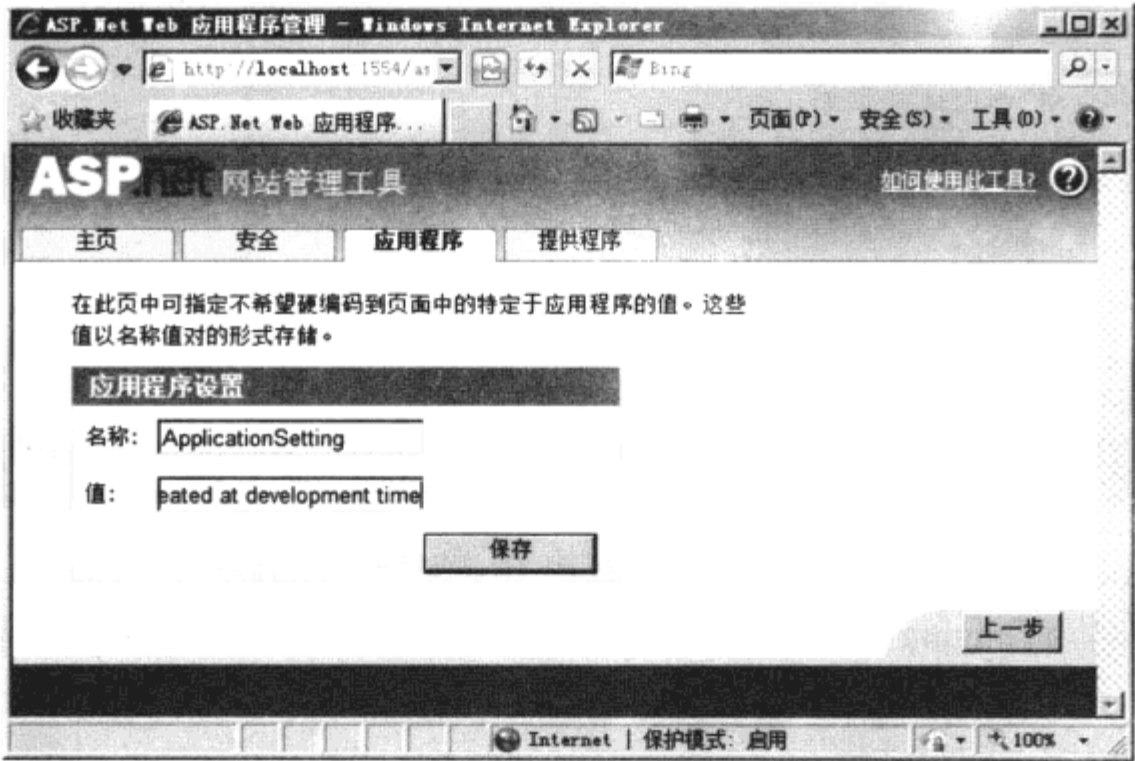


- 4. 在“活动解决方案配置”组合框中选择“<新建...>”。此时，Visual Studio 会打开“新

建解决方案配置”对话框(如下图所示)。将这个配置命名为 Staging，并选择从 Release 配置复制设置。单击“确定”来保存配置。



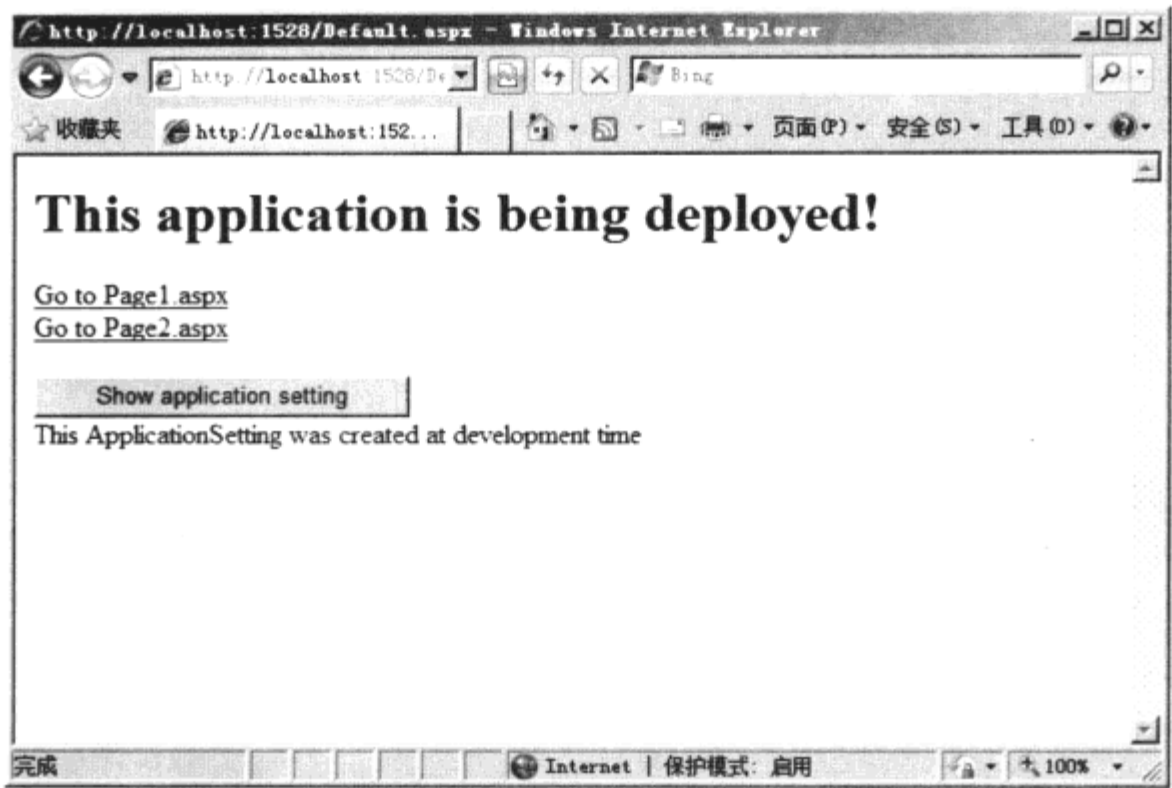
5. Staging 配置已经创建好了。下面我们针对 web.config 添加一个转换，以便处理开发和部署之间配置的变化。为此，首先为 web.config 文件添加一个应用程序设置。使用“ASP.NET 网站管理工具”工具，在项目的配置文件中添加一个应用程序设置(该工具可通过“项目”菜单下的“ASP.NET 配置”项打开)。添加一个键为 ApplicationSetting，值为 This ApplicationSetting was created at development time 的应用程序设置(如下图所示)。



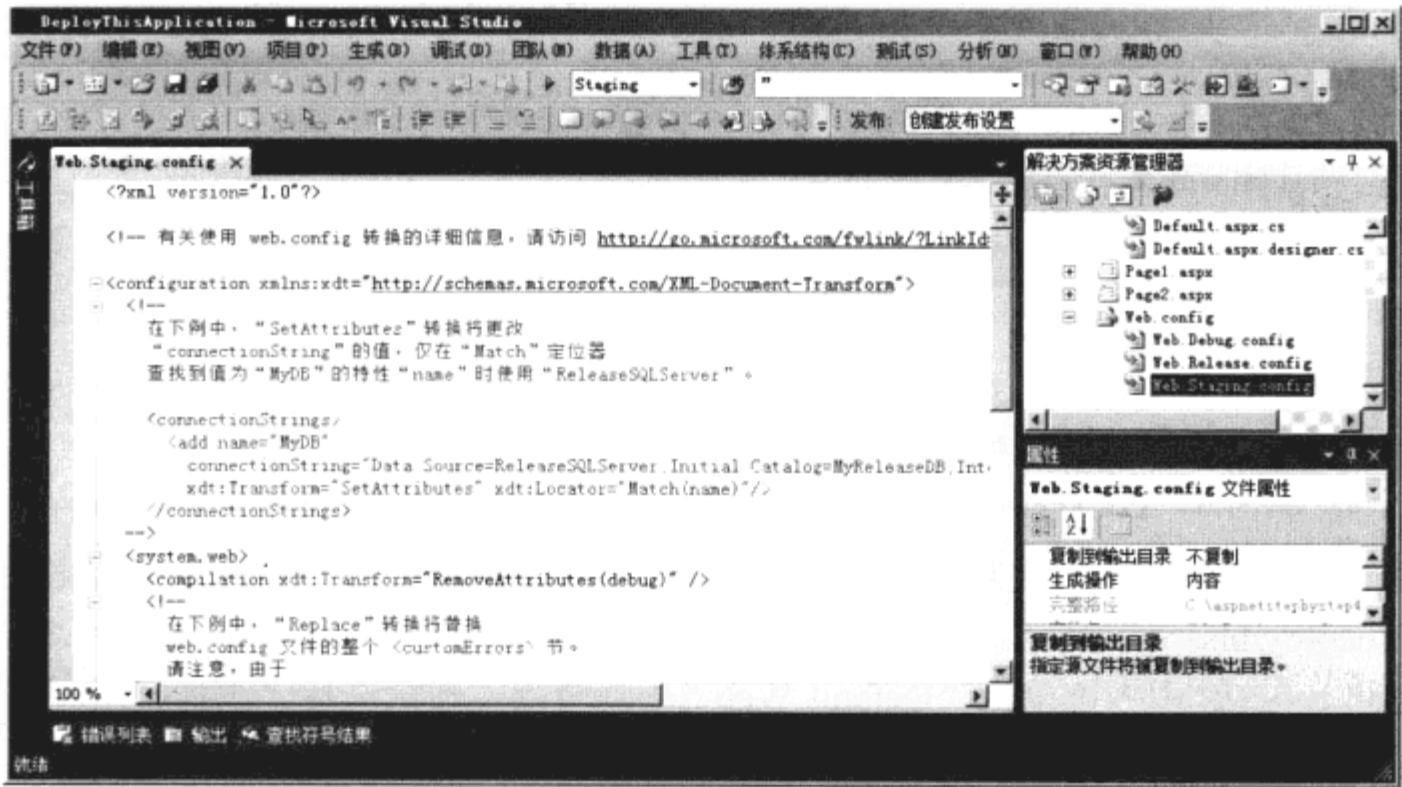
6. 在“设计器”中双击按钮，为其添加一个事件处理程序。使事件处理程序查找应用程序设置，并将其值显示在标签中(如下所示)。

```
protected void ButtonShowApplicationSetting_Click(object sender, EventArgs e)
{
    string applicationSetting =
        ConfigurationManager.AppSettings["ApplicationSetting"];
    this.LabelShowApplicationSetting.Text = applicationSetting;
}
```

运行这个页面并单击按钮后的效果应如下图所示。



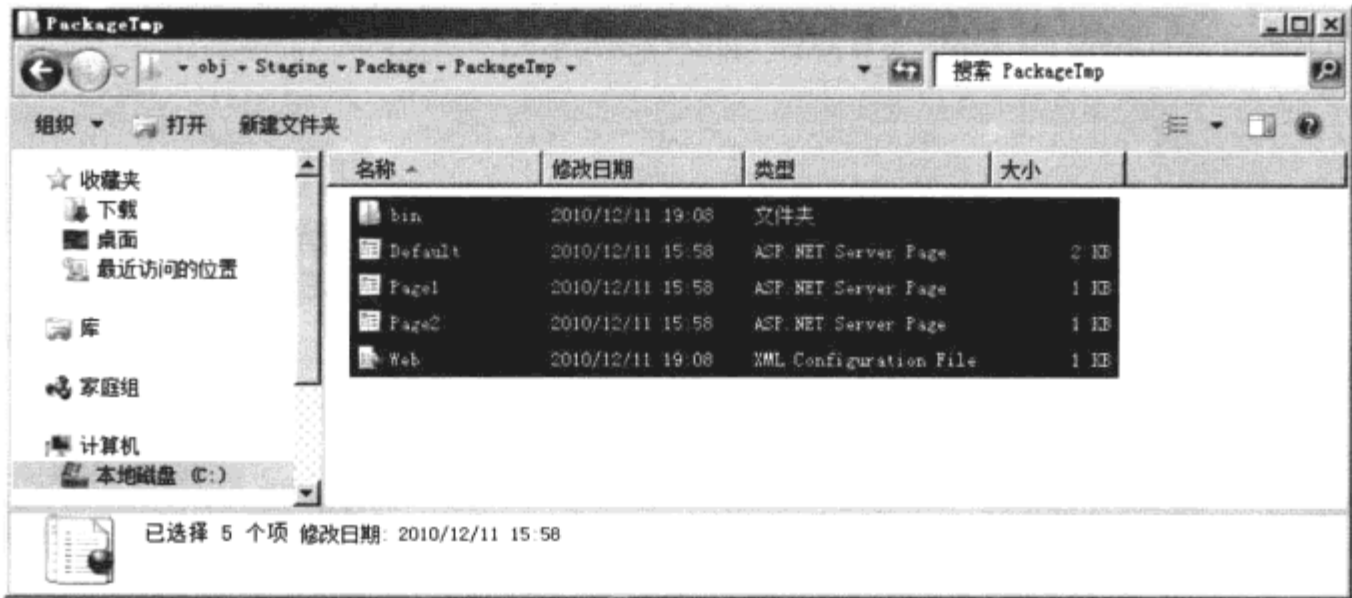
7. 现在我们创建一个在部署阶段使用的配置文件。在“解决方案资源管理器”中右键单击“Web.config”节点，然后选择“添加配置转换”。Visual Studio 会自动创建 Web.Staging.config。它在“解决方案资源管理器”中与 Web.Debug.config 和 Web.Release.config 位于同一节点下(如下图所示)。



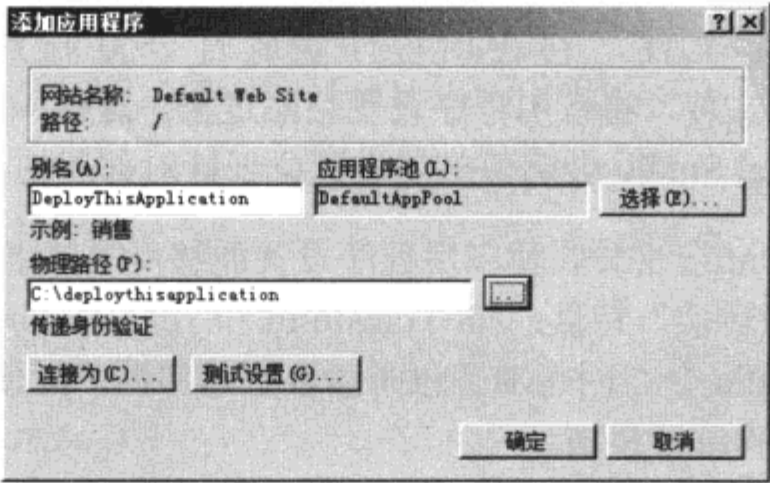
8. 打开主 web.config 文件，将应用程序设置部分复制到剪贴板。然后，打开 Web.Staging.config 文件，将应用程序设置粘贴过来。这样，应用程序的开发和部署设置就不一样了(Visual Studio 会确保部署设置位于针对部署的 web.config 文件中)。
9. 修改 Web.Staging.config 文件，使应用程序设置能够反映其来自于部署配置。此外，添加两个“XML 文档转换”特性：xdt:Transform 和 xdt:Locator，以便它能够针对部署正确替换。Visual Studio 会在创建部署包时读取这些特性，并使用 Web.Staging.config 文件中的应用程序设置进行替换。

```
<appSettings>
  <add key="ApplicationSetting"
    value="This ApplicationSetting was created at DEPLOYMENT time"
    xdt:Transform="Replace"
    xdt:Locator="Match(key)"
  />
</appSettings>
<system.web>
</system.web>
```

- 10. 下面我们将通过 Visual Studio 创建部署包。确保活动的解决方案配置为 Staging。首先，在“解决方案资源管理器”中右键单击项目节点，选择“打包/发布设置”。此时会打开“打包/发布设置”窗口。清除“以 zip 文件格式创建部署包”复选框。这样会使 Visual Studio 将待部署的文件输出到目录结构中，以便我们稍后查看。单击工具栏中的“保存”按钮来保存当前设置。
- 11. 右键单击“解决方案资源管理器”中的项目节点，选择“生成部署包”。此时，Visual Studio 会创建一个目录结构，并将要部署的文件全部置于该目录下。这个目录结构位于网站项目的 obj\Staging\Package\PackageTmp 目录下(如下图所示)。



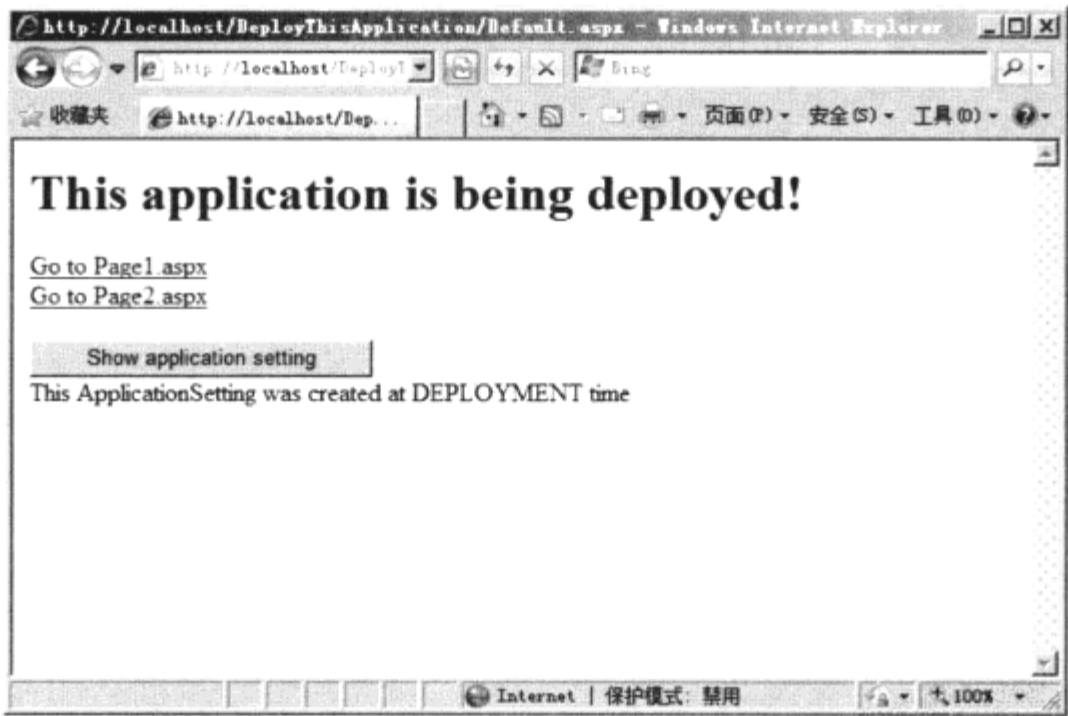
- 12. 将部署包目录的内容复制到 IIS 虚拟目录对应的物理目录下。这里假设该物理目录为 C:\deploythisapplication。通过 IIS 在“Default Web Site”下新建一个“Web 应用程序”。为此，在 IIS 中右键单击“Default Web Site”节点，然后选择“添加应用程序”（详见第 2 章）。指定包含网站内容的目录(如下图所示)。



13. 打开新网站的“内容视图”。此时，部署的内容已对 IIS 可见(如下图所示)。



14. 右键单击“Default.aspx”页面，选择“浏览”。单击页面上的按钮。如下图所示，页面所获取的应用程序设置是针对部署的版本。



26.4 快速参考

目 标	操 作
开发位于本地的网站，而不使用 IIS	创建“文件系统”网站
开发通过 IIS 运行的网站	创建“HTTP”网站
所开发的网站需要将文件复制到 FTP 服务器	创建“FTP”网站
为性能或部署而对网站进行预编译	使用 aspnet_compiler 命令行工具进行预编译，或通过 Visual Studio 进行发布

续表

目 标	操 作
发布 Web 应用程序	在 Visual Studio 的主菜单中选择“生成” “发布”。Visual Studio 会将内容文件推送到指定的目录下(该目录可以是 IIS 虚拟目录)
为 Web 应用程序创建安装程序	在解决方案中添加一个“Web 安装项目”。将必要的文件添加到该项目中，然后生成安装程序
创建针对部署的配置	在“解决方案资源管理器”中右键单击解决方案节点，选择“配置管理器”。在“活动解决方案配置”组合框中选择“<新建...>”。为新配置命名，然后可以选择根据现有配置进行创建
创建可转换的 web.config 文件	创建新的配置后，在“解决方案资源管理器”的 web.config 文件上右击，选择“添加配置转换”
创建部署包	在“解决方案资源管理器”中右键单击项目节点，选择“生成部署包”

```
[General Information]
name=ASP.NET 4从入门到精通
author=(美)谢菲尔德著
bookcontentsStr=http://nsst.5read.com/300-06/diskQJS/QJS1659/03/BookContents.dat;
dxid=000008112249
pages=514
pagesatt=0
pagesbok=1
pagescov=2
pagesflow=15
pagesleg=1
press=清华大学出版社
publishDate=2011.06
ssid=12786085
url=http://book1.duxiu.com/readDetail.jsp?dxNumber=000008112249&d=D94C1AD2D498085081F75295FD8DED72
```