ASP.NET 完全入门

《ASP.NET 完全入门》详细介绍

ASP.NET 简介

微软的.NET 技术目前正是风风火火，作为全球软件业最大的公司，微软在.NET 技术上投入了大量的人力物力，把公司未来战略重心放在了.NET 上，而从目前看来，.NET 技术无疑代表了未来 Internet 技术的方向。

ASP.NET 技术就是由微软的.NET 技术细化而来的，它跟 ASP 技术有些关系，然而又不是仅仅是一个继承，可以讲，ASP.NET 跟 ASP 想比较的话，已经是变得面目全非了，当然好得至少是一个量级以上。

ASP.NET 完全基于模块与组件，具有更好的可扩展性与可定制性，数据处理方面更是引入了许多激动人心的新技术，正是这些具有革新意义的新特性，让 ASP.NET 远远超越了 ASP，同时也提供给 web 开发人员更好的灵活性，有效缩短了 web 应用程序的开发周期。ASP.NET 与 Windows 2000 Server/Advanced Server 的完美组合，为中小型乃至企业级的 web 商业模型提供了一个更为稳定，高效，安全的运行环境。

正是基于上面的激动人心的理由，我们编写了这样一本书。

本书面向的读者

本书面向初、中、高级用户，全面系统地介绍了 ASP.NET 的特点、基础知识和具体的应用。

本书由浅入深，层层深入的讲解了 ASP.NET 技术，在本书中写作中，例子都使用了 VB.NET 的语法。所以，如果您对 VB 的语法或对 ASP 很熟悉的话，你将会发现很快就可以上手。如果你精通别的编程语言，通过对本书的阅读，也会让你跟上编程技术发展的前沿。

本书的组织结构

本书共分为 7 篇内容，从 ASP.NET 的介绍到高级应用。

- 第一篇　“概论”——本篇介绍了微软的.NET 战略、ASP.NET 的历史以及.NET 的安装和运行环境，即使你是一个菜鸟级人物，在这里你也会得心应手。

- 第二篇　“WEB Form”—— 本篇介绍了 ASP.NET 的 WEB Form 技术，对 APS.NET 的服务器控件、自定义控件和 HTML 控件技术进行了深入、详细的介绍，对应于每一个控件，都有一个例子相对应，是深入了解 ASP.NET 的基础。

- 第三篇　“数据库编程技术”——本篇详细介绍了数据库编程的基础、ADO.NET 数据库编程的基础、ADO.NET 数据库基本连接和操作、Dataset 的用法和数据绑定等技术，是制作动态页面、BBS、电子商务网站的等网站应用程序的基础，也是由初级读者向高级应用者迈进的必经之路。

- 第四篇　“应用程序”——本篇先介绍如何配置 config.web，如何编写 global.asax，再结合一个实例“会员系统”来对应用程序进行深入的讲解，让你对 ASP.NET 的基本知识在更深入一层。

- 第五篇　“WEB SERVICE”——本篇对 WEB SERVICE 进行了详细的讲解。通过例子学习 WEB SERVICE。同时，深入地讲解了数据交换和存取站点对象，本篇是微软.NET 计划的一个重点。

- 第六篇　“性能优化”—— ASP.NET 有两种用于 WEB 应用的缓冲技术：输出缓冲和数据缓冲，在本篇中将围绕的这两种缓冲技术，进行详细的讲解。

- 第七篇　“高级应用”——在高级应用一篇中，我们将介绍三个方面的内容：XML 在 ASP.NET 中的应用、三层结构及其应用、以及微软消息队列（MSMQ），这些是作为一个高级应用者所必需掌握的只是。

学习本书需要使用的工具

在本书中，有一个附带的光盘，此光盘包含全书应用到的所有例子的源代码，另外在书中的例子的前面都说明有源代码的出处。为了运行这些代码，你需要配置一个运行环境，需要一个.NET 的软件开发工具包（SDK），详细的配置请参考本书的介绍和相关的文件。

本书不要求你的机器中安装 Visual Studio 7 的工具。只要你成功安装好 ASP.NET 的调试环境即可。

作　　者

2001 年 04 月 19 日

ASP.NET 完全入门

第一篇　概论

第一章 微软.NET 战略和 ASP.NET 简介

欢迎你阅读《ASP.NET 完全入门》，通过对本书的阅读，我们相信你能够对 ASP.NET 会有更深入的了解。

ASP.NET 又叫 ASP+，但并不仅仅是 ASP 的简单升级，而是 MicroSoft 推出的新一代 Active Server Pages 脚本语言。ASP.NET 是微软发展的新型体系结构.NET 的一部分，它的全新技术架构会让每一个人的网络生活都变得更简单。

首先需要特别指出的是，ASP.NET 不仅仅只是有了一个新界面并且修复了一些缺陷的 ASP3.0 的升级版本(即不同于 ASP2.0 升级到 ASP3.0 的转变)。更为重要的是，ASP.NET 吸收了 ASP 以前版本的最大优点并参照 Java、VB 语言的开发优势加入了许多新的特色，同时也修正了以前的 ASP 版本的运行错误。

要了解 ASP.NET 的真实面目，我们首先就得了解一下微软.NET 战略。

1.1.1 微软.NET 的历史

随着网络经济的到来，微软公司希望帮助用户，能够在任何时候、任何地方、利用任何工具都可以获得网络上的信息，并享受网络通信所带来的快乐。.NET 战略就是为着实现这样的目标而设立的。

微软公开宣布，今后将着重于网络服务和网络资源共享的开发工作，并称，将会为公众提供更加丰富、有用的网络资源与服务。

微软新一代平台的正式名称叫做“新一代 Windows 服务”（NGWS），现在微软已经给这个平台注册了正式的商标——MicroSoft.Net。在.Net 环境中，微软不仅仅是平台和产品的开发者，并且还将作为架构服务提供商、应用程序提供商，开展全方位的 Internet 服务。在谈及这个平台中使用的新技术，微软透露，它将在.Net 环境中提供更多新产品和一揽子的全套服务。

MicroSoft .NET 平台的基本思想是：

侧重点从连接到互联网的单一网站或设备上，转移到计算机、设备和服务群组上，使其通力合作，提供更广泛更丰富的解决方案。用户将能够控制信息的传送方式、时间和内容。计算机、设备和服务将能够相辅相成，从而提供丰富的服务，而不是像孤岛那样，由用户提供唯一的集成。企业可以提供一种方式，允许用户将它们的产品和服务无缝地嵌入自己的电子构架中。这种思路将扩展二十世纪八十年代首先由 PC 赋予的个人权限。

MicroSoft .NET 将开创互联网的新局面，基于 HTML 的显示信息将通过可编程的基于 XML 的信息得到增强。XML 是经“万维网联盟”定义的受到广泛支持的行业标准，Web 浏览器标准也是由该组织创建的。微软公司为开发它投入了大量精力，但它并不是 MicroSoft 的专有技术。XML 提供了一种从数据的演示视图分离出实际数据的方式。这是新一代互联网的关键，提供了开启信息的方式，以便对信息进行组织、编程和编辑；可以更有效地将数据分布到不同的数字设备；允许各站点进行合作，提供一组可以相互作用的“Web 服务”。

1.1.2 微软.NET 的介绍

1.1.2.1 MicroSoft .NET 综述

MicroSoft .NET 平台包括用于创建和操作新一代服务的.NET 基础结构和工具；可以启用大量客户机的.NET User Experience；用于建立新一代高度分布式的数以百万计的.NET 积木式组件服务；以及用于启用新一代智能互联网设备的.NET 设备软件。

MicroSoft .NET 产品和服务—包括 Windows.NET，连同建立积木式服务的核心集成套件；MSNTM .NET；个人订购服务；Office.NET；Visual Studio .NET；以及用于.NET 的 bCentralTM。

.Net 环境中的突破性改进在于：

1. 使用统一的 Internet 标准（如 XML）将不同的系统对接；

2. 这是 Internet 上首个大规模的高度分布式应用服务架构；

3. 使用了一个名为“联盟”的管理程序，这个程序能全面管理平台中运行的服务程序，并且为它们提供强大的安全保护后台；

.NET 平台包括如下组件：

1. 用户数据访问技术。其中包括一个新的基于 XML 的、以浏览器为组件的混合信息架构，叫做“通用画板”；

2. 基于 Windows DNA 2000 的构建和开发工具；

3. 一系列模块化的服务，其中包括认证、信息传递、存储、搜索和软件送递功能；

4. 一系列驱动客户设备的软件；

1.1.2.2 Microsoft.NET 平台带来的重要意义

我们来看一下 MicroSoft .NET 对开发人员、IT 专业人员、以及企业应用的巨大意义。

- **对于开发人员**

MicroSoft .NET 的策略是将互联网本身作为构建新一代操作系统的基础，对互联网和操作系统的设计思想进行合理延伸。这样，开发人员必将创建出摆脱设备硬件束缚的

应用程序，以便轻松实现互联网连接。MicroSoft .NET 无疑是当今计算机技术通向计算时代的一个非常重要的里程碑。

.NET 的核心组件有：

• 一组用于创建互联网操作系统的构建块，其中包括 Passport.NET（用于用户认证）以及用于文件存储的服务、用户首选项管理、日历管理以及众多的其它任务

• 构建和管理新一代服务的基本结构和工具，包括 Visual Studio.NET、.NET 企业服务器、.NET 框架和 Windows.NET

• 能够启用新型智能互联网设备的.NET 设备软件

• .NET 用户体验

.NET 对最终用户来说非常重要，因为计算机的功能将会得到大幅度提升，同时计算机操作也会变得非常简单。特别地，用户将完全摆脱人为的硬件束缚：用户可以自由冲浪于互联网的多维时空，而不是束缚在便携式电脑的方寸空间——可通过任何桌面系统、任何便携式电脑、任何移动电话或 PDA 进行访问，并可对其进行跨应用程序的集成。

.NET 可使用户轻松进行互联网连接，并轻松完成那些在当今看来十分费时而且费力的事务，它们往往要求用户进行数据重输入并需运行几个小时才能完成。通过将多项安全数据流合并到单一的用户界面（或者甚至是可编程决策引擎），.NET 架构将用户从充斥于当今 Web 的数据竖井的束缚中解脱出来。用户可以自由访问、自由查看、自由使用他们的数据。

.NET 对开发人员来说也十分重要，因为它不但会改变开发人员的开发应用程序的方式，而且使得开发人员能创建出全新的各种应用程序。新型开发范例的核心是 Web 服务这个概念的引入。Web 服务是一种通过简单对象访问协议(SOAP)，在互联网上展露其功能性的、极为公开的服务。SOAP 是一种基于可扩展标记语言(XML)制定的协议。

在过去，开发人员通过集成本地系统服务来构建应用程序。在这种模型下，开发人员可以访问丰富的开发资源并能严格控制应用程序的行为。

如今，开发人员已在很大程度上挣脱了这种模型的束缚，致力于构建具有复杂结构的 n 层化系统，这种系统能将网络上众多的应用程序一并进行集成，大大提升了应用程序的价值。这样，开发人员便可把精力集中在充分挖掘软件独特的商业价值，而不是构建基本结构上。可喜的局面将应运而生：软件投放市场的时间大大缩短、开发人员的编程效率明显提高，最终把质量上乘的软件呈现给用户。

我们正在进入一个崭新的计算时代——一个由互联网（尤其是 Internet 核心技术 XML）实现的时代。利用 XML，能够创建出可供任何人从任何地方使用的、功能非常强大的应用程序。它极大地拓展了应用程序的功能，并实现了软件的动态提供。在这种情况下，软件已不完全指那些从光盘进行安装的程序，而是演变成了一种服务——类似于 ID 调用程序或按收看次数进行收费的电视——人们可通过通信媒体订购的服务。

n 层计算技术具有能够大幅度提高生产力、紧密耦合的特点，而 Web 概念具有面向消息、松散耦合的特点，我们将二者有机地糅合在一起，实现了上述构想。我们将这种计算风格称为 Web 服务，它的出现标志着人类已经迈入应用程序开发技术的新纪元。Web 服务是一种应用程序，它可以通过编程并使用标准的 Internet 协议，像超文本传输协议(HTTP)和 XML，将功能展示在互联网和企业内部网上。还可将 Web 服务视作 Web 上的组件编程。

从理论上讲，开发人员可通过调用 Web 应用编程接口(API)，将 Web 服务集成到应用程序中。其调用方法与调用本地服务类似，不同的是 Web API 调用可通过互联网发送给位于远程系统中的某一服务。例如，MicroSoft Passport(Passport)服务使得开发人员

能够对应用程序进行认证。通过对 Passport 服务编程，开发人员可以充分利用 Passport 的基本结构，通过运行 Passport 来维护用户数据库，以确保其正常运行、定期备份等等。

.NET 正是根据这种 Web 服务原则而创建的，微软目前正着手提供这个基本结构，以便通过.NET 平台的每一部分来实现这种新型的 Web 服务。而 Visual Studio.NET、.NET 框架、Windows.NET 和.NET 企业服务器，正是为进行基于 Web 服务模型的应用程序开发而度身定做的新一代开发工具和基本结构。.NET 构建块服务、新增的.NET 设备支持以及即将到来的.NET 用户体验，将为人们彻底攻克这一难题划上一个圆满的句号，使人们能够充分利用 Web 服务模型，如愿以偿地开发出新一代应用程序。

.NET 对 IT 专业人员的重要意义

目前，IT 专业人员能够利用与构建.NET 平台相同的技术。

.NET Enterprise Servers 和 Windows 2000 操作系统，为创建具有高度可管理性的、能迅速投入市场的应用程序提供了坚实基础。它们利用的是可扩展标记语言(XML)，因此随着 Web 体系结构的革新，在此平台上创建的程序依然很有价值。

.NET 平台的核心是，采用有效的、分门别类的方式来构建应用程序，达到其前所未有的规模。该平台上的 Web 服务模型指的是：企业应用程序的中心业务要素通常由本地管理，而支持它们的服务（如用户认证、文件存储、用户首选项管理、日历、邮件等等）却无须本地管理，可以被无缝订购。为了存储用户文件和邮件，IT 专业人员往往在服务器上安装新的独立磁盘冗余阵列（RAID 阵列），而有了.NET，他们在这一方面将会花费较少的精力，而更多地致力于怎样为公司增加效益。

该 Web 服务模型还将动态配置新软件的发布和更新。用户将以极其紧密的连接方式工作，因此更易于管理。而简化的管理又可使 IT 专业人员更能适应变幻莫测的业务需求。

开发应用程序的.NET Web 服务模型将为企业应用程序的创建开辟一条新路。通过企业内外多种服务的联合，很容易把企业内部数据和客户及合作伙伴的相关数据结合在一起，大大简化了应用程序的创建过程。这就为最终用户发掘了空前的功能涵盖性。例如，利用某公司的雇员福利程序，可以从其 HR 数据库订购信息，通过 Web 订购福利管理公司的服务、订购工资管理公司的服务。终端用户可以在简单、直观的界面下操作，而这个界面可以显示他们的累积休假时间、个人所得福利以及上次工资额。

.NET 对企业的重要意义

MicroSoft .NET 平台将从根本上改善计算机和用户之间进行交互的方式，最大限度地发挥电子商务中计算技术的重要作用。首先，让我们来分析一下当前商务计算世界的现状：

人与计算机进行交互的手段极为有限——通常使用键盘和鼠标进行输入，使用监视器监控输出。

用户信息基本上是本地信息；如果从另一台机器进行登录，则无法获取用户的个人首选项设置、数据及应用程序。

用户必须亲自处理信息，而通过设置智能选项代表用户自动进行操作，则无异于是纸上谈兵。

同一用户存放于不同应用程序和站点的数据，很难（或根本不可能）进行自动合并和关联，用户无法统一进行查看。

想在家里或在路上工作的用户，不能方便地访问办公室电脑中的应用程序和数据。这无疑成为一道阻止人们获得更高工作效率的鸿沟。

不能使用其它设备访问专为特定设备设计的数据（这些设备包括 PC、寻呼机、移动电话以及 PDA 等）；最多可以定期进行同步。

.NET 将保证完全消除当今计算技术中的所有缺陷。.NET 定能实现确保用户从任何地点、任何设备都可访问其个人数据和应用程序的宏伟蓝图。除此之外，.NET 技术还可实现多个应用程序在逻辑上的松散耦合链接和紧密耦合链接。

用户可以通过手写、语音和图象技术与其个人数据进行交互。这些数据将安全地存放在互联网上，用户通过办公室（或家庭）PC，还可以通过移动电话或寻呼机、PDA、甚至是新发明的寻呼机——移动电话——PDA——PC 联合设备访问这些数据。应用程序可进行灵活的功能调整，以适应用户所用设备的功能状况。应用程序可根据用户预定义的选项集和指令集，完全代替用户自动执行相应的操作。

上述功能将协同作用，以便大幅度地提高用户使用计算技术的生产效率。根据设计，.NET 使得用户无需在如何与计算机进行交互上劳神，从而全身心地投入到使计算机自动执行任务、实现最终目标的工作中。通过使用 XML 行业标准，可将用户数据进行跨站点和应用程序的链接，从而轻松实现当前很难实现的操作。比如：对用户在数家不同银行、信用卡公司以及计费代理商那里的数据进行集中处理；这样，用户便可依据处理后的数据支付帐单，将费用明细报告归档。

.NET 把雇员、客户和商务应用程序整和成一个协调的、能进行智能交互的整体，而各公司无疑将是这场效率和生产力革命的最大受益者。简言之，.NET 承诺为人类创造一个消除任何沟鸿的商务世界。

1.1.2.3 MicroSoft .NET 的基本模块

u 网络服务一览

通常说来，一个网络服务只是一个作为服务－－通过 Internet 标准此服务能与其它网络服务集成在一起－－发行的简单的应用程序。换句话说，它是可通过 URL 定位的自动将信息返回到需要它的客户端那里的一种资源。网络服务一个重要的特点是客户不需要知道一种服务是怎样实现的。在本节中，我将向你解释网络及网络服务如何把基于组件技术的最好的方面结合在一起的，并且介绍与网络服务通信所需的基本框架。

同组件一样，网络服务提供“黑匣子”函数，它可以被再次作用而不用关心此服务是怎样实现的。网络服务提供被称为契约的精确定义的接口，此接口描绘了所提供的服务。开发人员可以将远程服务、本地服务和定置代码组合在一起而集成应用程序。例如，某公司可以使用如下服务组建一在线商店：微软护照（原文：Passport）服务以验证用户身份，第三方个人化服务以使网页匹配每一个用户的参数，信用卡处理服务，销售税服务，对每个运输公司的包裹跟踪服务，链接公司内部库存管理程序的内部目录服务，以及少量定置代码以使他们的商店能脱颖而出。

然而，网络服务与现在的组件技术不同，它不使用需要在服务器和客户机有明确的、同类型基本构架的具体的对象模型协议，例如 DCOM、 RMI 或 IIOP 。尽管与具体组件技术紧密结合的实现在一个受控的环境中能很好地被接受，但它们在网络环境中变得不切实际。因为一个集成商业程序的参与者会发生变化，随着时间的推移，技术也在变化，所以在所有参与者间确保一个单一的、统一的体系架构就变得十分困难。网络服务

采取了另外一种途径，它使用普便存在的网络协议和数据格式，如 HTTP 和 XML，进行通信。支持这些网络标准的任何系统都支持网络服务。

而且，网络服务契约描述的是以术语报文形式提供的服务，这些服务是由网络服务生成和接受的，而不是描述服务是如何实现的。通过把重点放在报文上，网络服务模板就完全对语言、平台和对象模板一无所知。 用任何一套编程语言、对象模型和平台的完全特性集，都可实现网络服务。网络 服务可在任何平台被用任何语言所实现的应用程序使用。只要用于解释服务容量、报文序列和所期望协议的契约得到认同，那么所实现的网络服务及网络服务用户就可相互不同，而不会影响会话另一端的应用程序。

网络服务模板对最小体系架构的要求很低，以确保网络服务在使用任何技术和编程语言的平台上实现和访问。对网络服务互用性的解决可只依靠网络标准。然而，为了使应用程序更容易使用网络服务，简单地同意通过标准网络协议就可以访问网络服务是不够的。当网络服务和网络服使用者依靠标准的方式表示数据和命令、表示网络服务契约 、算出网络服务所提供的容量时，网络服务才容易使用。

XML 是定义一个标准的、可扩展的用于提供命令和典型数据的语言明显的一种选择。虽然为表示命令和典型数据可以定义使用其它技巧（比如编码为一种查询字符串）的规则，但 XML 被专门设计为描述数据的标准元语言。简单对象存取协议（SOAP）是以一种可扩展的方式使用 XML 表示数据和命令的工业标准。网络服务可选择用 SOAP 决定报文的格式。

XML 是网络服务契约的一种使能技术。服务契约语言 (SCL)是记录网络服务契约的 XML 语法。由于 SCL 是基于 XML 的，所以对开发者和开发工具来说，容易生成、解释契约。关于 SCL 细则的草案很快会出台（注意：现在的 SOAP Toolkit for Visual Studio 6.0 支持称为 SDL 的 SCL 的早期版本）。

Disco 规范为服务提供者发布网络 服务契约和相应的机制描述了一个标准方式，这将使开发者或开发工具可找到契约文献。当你读到这里时， Disco 规范的草案应出台了。

象 SOAP， SCL 和 Disco 这样的标准有助于开发者，因为它们不需要明白和实现所使用的每一个网络服务的访问方式。支持这些标准的更好的、已充分测试的、高性能的体系架构将由开发平台提供，这会大大简化整个开发过程。

- **MicroSoft .NET Framework**

MicroSoft .NET 框架的目的是使你更容易建立网络应用程序和网络服务。图 2 显示了 MicroSoft .NET 框架的体系。建立在操作系统最上层的服务，是管理运行时代码需求的 common language runtime，这些代码可以用任何现代编程语言所写。Runtime 提供了许多服务，这些服务有助于简化代码开发和应用程序的开发同时也将提高应用程序的可靠性。.NET Framework 包括一套可被开发者用于任何编程语言的类库。在此之上是许多应用程序模板，这些模板特定地为开发网络站点和网络服务提供高级组件和服务。

- **Common Language Runtime**

运行语言(runtime)调入并运行用任何运行感知编程语言所写的代码。以运行为目标的代码被称为受控（managed ）代码，受控代码代码只是意味着在内部可执行代码与运行自身间存在已定义好的合作契约。对于象生成对象、调用方法等这样的任务，被委

托给了运行语言，这使得在运行语言能为可执行代码增加额外的服务。

运行语言以交叉语言集成、自描述组件、简单配制和版本化及集成安全服务为特点。

运行语言使用一种新的能表达大部分现代编程语言语义的通用类型系统，通用类型系统定义了一套标准类型及生成新标准的规则。运行语言知道怎样生成、执行这些类型。编译器和解释器使用运行语言服务定义类型、管理对象、进行方法调用，而不是使用工具或特定于语言的方法。

类型系统的主要设计目的是使多种语言能深度集成。用一种语言所写的代码能继承用另一种语言所写的类的实现，用一种语言所写的代码抛出的异常能被用另一种语言写的代码捕获，象调试和剖析之类的操作会在完全封闭下工作，而不用考虑代码所用的语言。这就意味着编写可重用类库的开发者，不再需要为每一种编程语言或编译器生成一个版本，并且使用类库的开发者不再受到为他们使用的编程语言开发的库的限制。

自描述组件－－现在 MicroSoft .NET 框架上已成为可能－－简化了开发和配制，并提高了系统的可靠性。许多由运行语言提供的服务是由元数据及用于补充可执行代码的信息所驱动。因为所有的信息都储存在一起，只有可执行的（代码）才被称为自描述组件。

自描述组件的一个主要优点是，使用它们并不需要其它文件。类的定义不需要单独的头文件；通过检查元数据对类的定义可以从组件自身获得。跨语言或过程边界访问组件并不需要各自的 IDL 文件、类型文件或 proxy/stubs；所必需的信息已存在于元数据之中。为识别开发者请示的服务属性，并不需要展开各自的配制信息。 最主要的是，由于元数据是在编译过程中由源代码生成，并与可执行代码储存在一起，它将永远和可执行部分同步。

除了改善对单个组件的配制，Microsft .NET 框架定义了一个应用程序配制模板，以解决定置应用程序安装和 DLL 版本化（通常被称为“DLL Hell”）这一复杂过程的问题，运行语言提供了支持这个模板的服务。

Microsft .NET 框架 引入了组合体的概念。一个组合体是一组资源和类型，并包括有关这些资源和类型的元数据，也就是被作为一个单元配制的。元数据被称为组合体的名单，它包含象类型和资源表之类能被组合体外看得见的信息，这个名单也包括有关从属关系之类的信息，例如组合体建立时的版本号。开发人员可以指定版本策略，以指示运行语言是否装入系统上已安装的依赖于组合体的最新版本，装入一指定版本，或在编译时使用的版本。

某软件组件的多个拷贝总可以存在于同样的操作系统上，然而，通常说来，只有其中的一个拷贝能被操作系统注册、调入内存、执行。对系统来说，定位和调入内存的策略是全局性。.NET Framework Common Language Runtime 增加了所必须的体系架构以支持管理组件定位和调入的每个应用程序策略，这通常被称为并行配制。

组合体可以被一个应用程序私有，或被多个应用程序共享。一个组合体的多个版本可以同时配制在同一台机器上。应用程序配制信息定义了到何处去查找组合体，这样 runtime 就能为同时运行的两个不同的应用程序装入同一组合体的不同版本。这就消除了由组件版本的不兼容性引起的问题，提高了系统整体的稳定性。如果必要，如果必要，管理员可以为配制时刻的组合体增加配制信息，例如一个不同的版本策略，但是编译时提供的原始信息永远不会丢失。

因为组合体是自描述的，所以并不需要在系统上进行显式注册。应用程序的配制简单到只需将文件拷贝到目录中既可（如果为了使应用程序能够运行，必须安装未经组织过的组件的话，情况会稍微复杂一点）。配制信息保存在可被任何文本编辑器编辑的 XML 文件中。

最后，运行语言也提供完整的、普遍深入的安全服务，以确保未经授权的用户不能访问机器上的资源，并且代码不会执行未经允许的动作。这就提高了系统整体的安全性可靠性。 由于运行语言用于装入代码、生成对象、执行方法调用，所以当受控代码装入内存、执行时，运行语言能进行安全检查，强化安全策略。

Microsft .NET 框架不仅规定代码访问安全，还规定基于角色的安全。通过代码访问安全机制，开发人员能为应用程序指定完成工作所必需的权限。例如，代码或许需要写文件或访问环境变量的权力。这类信息和有关代码标志的信息一起存储在配制级上的。当代码装入内存及执行方法调用时，运行语言验证是否能给予代码所要求的权限。如果不能，将记录一条安全冲突信息。给予权限的策略，这被称为信任策略，是由系统管理员建立的，并且是建立在关于代码的证据基础之上，比如：代码是谁发布的，是从什么地方获得的，以及在组合体中找到的代码标志和它要求的权限。开发人员可以指定他们显然不需要的权限，以防止其它人恶意使用他们的代码。如果所需要的权限依赖直到运行时刻才会知道的信息，那么就可写入纲邻性的安全检查。

除了代码访问安全，运行语言还支持基于角色的安全。基于角色的安全建立同代码访问安全一样的权限模板，只是这些权限是建立在用户的身份之上，而不是建立在代码的标志之上。角色表明了用户所属的类，并且可以在开发和配制阶段定义。给予权限的策略被分配到每个预定义的角色。在运行时刻，用户的身份被确定，代码将代表这个身份运行。运行语言决定用户是哪个角色的成员，然后给予基于这个角色的权限。

在查看 Microsft .NET 框架的可编程模板前，先看一下它所提供的服务。

l 服务框架

在 Common Language Runtime 之上是服务框架，此框架提供能被任何现代编程语言调用的类。所有的类都遵循一套命名和设计方针，以大大减小开发人员的学习上的弯路。

框架包括一套开发人员希望在标准语言库中存在的基类库，例如：集合、输入/输出，字符串及数据类。另外，基类库提供访问操作系统服务如图画、网络、线程、全球化和加密的类。服务框架也包括数据访问类库，及开发工具，如调试和剖析服务，能够使用的类。本文章没有详细讨论所有的类，我将重点放在数据访问类上，因为大多数网络服务需要对数据的访问。当然，你可以在 MicroSoft .NET Framework SDK 中找到关于服务框架类库的附加信息。

l 数据访问服务

几乎所有的网络服务都需要查询和更新永久性数据，不论是以简单文件，还是以相关数据库，或是以其它的存储类型存在。为了提供对数据的访问，服务框架包括 ActiveX Data Objects+ (ADO.NET)类库。如同名子所暗示地那样，ADO.NET 由 ADO 发展而来。ADO+被设计为基于网络的可扩展的应用程序和服务提供数据访问服务。ADO.NET 为连接的指针风格的数据访问，同时也为更适合于把数据返回到客户端应用程序的无连接的数据模板提供高性能的 APIs 流，就象在以后介绍的那样。

就象其余几个部分一样，ADO.NET 定义了那些链接数据仓库、 对数据仓库发送命令及从中获取结果的类。这些类由受控数据提供者（managed data provider）实现。ADO+中链接和命令对象看上去和 ADO 中的是一样的，并且一个名为 DataReader 的新类提供了通过高性能 API 流获取结果的能力。DataReader 在功能上同前向、只读的 ADO 记录集（Recordset）是等同的，但是 DataReader 被设计用来最小化内存中生成的对象的数量，以提高性能，避免垃圾积累。在.NET Framework 中包含了针对 MicroSoft SQL

Server™的受控数据提供者以及可通过 OLE DB 访问的任何数据仓库。

ADO.NET 的一个主要创新是引入了数据集（Dataset）。一个数据集是内存中提供数据关系图的高速缓冲区。数据集对数据源一无所知，它们可以由程序或通过从数据仓库中调入数据而被生成、填充。不论数据从何处获取，数据集都是通过使用同样的程序模板而被操作的，并且它使用相同的潜在的数据缓冲区。使用.NET 平台的开发人员能够用数据集代替传统 ADO 中无连接的记录集。

受控数据提供者为数据仓库和数据集公开一名为 DataSetCommand 的接口对象。DataSetCommand 使用 ADO.NET 链接和命令以从数据仓库中填充数据集，并把在数据集中发生的变化解析到数据仓库中。

就象 DataReaders 显示了对于相关数据的有效的流访问一样，XmlReaders 显示了对 XML 数据的流访问。开发人员使用 DataNavigator 可以滚动和编辑内存中的 XML 文档。DataNavigator 在功能上和 W3C Document Object Model (DOM)是一样的，但它更有效，并提供了能很好映射关系数据表的对象模板。DataNavigator 支持 Xpath 语法以对数据流进行导航。ADO.NET 为那些希望继续使用 DOM 作为 XML 对象模板而不是使用更有效的 DataNavigator 模板的开发人员提供了一个 XMLDocument 类。

由于所有的数据都可被看作 XML，所以开发人员可以为任何数据使用转换和确认服务。ADO.NET 定义了一个消费 DataNavigator、生成一个新的 XmlReader 的通用转换体系。.NET Framework 提供了一个支持 W3C XSL Transformations (XSLT)细则的特殊转换组件。ADO.NET 同时提供了一使用 XML 简图确认 XmlReader 的确认引擎。ADO.NET 支持通过 DTDs， XSD 或 XDR 定义的简图。

l 表单应用模板

从概念上讲，在服务框架的最上面是两个应用程序模板：Windows 应用程序模板和网络应用程序模板。尽管我把重点放在把微软.NET 框架用作开发网络服务和网络应用程序的一种途径上，但框架也可用于开发较传统的基于 Windows 的应用程序（当然，这些应用程序也能使用网络服务）。

编写 Windows 客户应用程序的开发人员可使用 Win 表单应用程序模板以利用 Windows 丰富的用户接口特点，包括现在的 ActiveX 控件和 Windows 2000 的新特点，如透明的、分层的、浮动窗口。可以选择传统的 Windows 或网络外观。得知它和现在的基于 Windows 表单包的相似性以后，开发人员会发现 Win 表单可编程模板和对设计阶段的支持非常直观。

Win 表单利用了 Microsft .NET 框架 runtime 以减少基于 Windows 的客户应用程序的开销。只要应用程序和组件是用 Win 所写或被 Win 表单应用程序使用，那么它们就能被框架安全模板在客户机上安全地执行。如果以这种方式使用或执行，那么某人从 Internet 下载下来的生猛游戏就不会对配制信息和数据产生破坏，否则会自动地给用户地址薄里的每一个人发送电子邮件。

Microsft .NET 框架 装配模板简化了应用程序的配制和版本化 。应用程序可被配制为使用它们在编译和测试所用的共享组件，而不是使用恰好在客户机器上安装的随便什么版本的组件，这就提高了应用程序的可靠性，减少了应用程序所支持调用的主要因素：用户接口控件和其它共享组件版本的不兼容性。

l 网络应用程序模板

建立在 Microsft .NET 框架 上网络应用程序共享一个通用应用程序模板。在这个模型中，网络应用程序是一套起源于基 URL 的 URLs。因此它包含用于生成在浏览器中观看的网页的网络应用程序和网络服务。在本节中，我将详细介绍称为 Active Server Pages+ (ASP.NET)的网络应用程序可编程模板

如同你从名字猜到的那样，ASP.NET 是由活动服务器页面发展而来。ASP.NET 利用 common language runtime 和服务框架网络应用程序提供了一个可靠的、自动化的、可扩展的主机环境。ASP.NET 也受益于 common language runtime 集成模板，简化了应用程序的配制。另外，它提供简化应用程序开发的服务（如状态管理服务）以及高水平的编程模板（如 ASP.NET Web Forms 和 ASP.NET Web Services）。

ASP.NET 的核心是 HTTP 运行语言，一个高性能的用于处理基于低级结构的 HTTP 请求的运行语言，而基于的结构与 MicroSoft Internet Information Services (IIS)所提供的 ISAPI 结构相似。如同你在图 5 所看到的，HTTP 运行语言是在象服务器上的 IIS 或客户机上的 IE 之类的 unmanaged 主机过程中运行的受控代码。HTTP runtime 负责处理引入的所有 HTTP 请求，并对每个请求应用程序的 URL 进行解析，然后把请求分配到应用程序以进行进一步的处理。HTTP 运行语言是多线程的，并异步处理请求，因此劣质的应用程序代码阻碍不了它对新请求的处理。而且 HTTP 运行语言假定失败必会发生，因此它被控制为尽最大力量自动地从访问冲突、内存泄漏、死锁等事故中恢复过来。除非是硬件故障，运行语言的目标是 100%的可靠性。

ASP.NET 使用基于构件的 Microsft .NET 框架配制模板，因此它获得了如 XCOPY 配制、构件并行配制、基于 XML 配制等优点。ASP.NET 另一个主要优点是，它支持应用程序的实时更新。管理员不必关掉网络服务器或者甚至不用停止应用程序的运行就可以更新应用文件。应用程序文件永远不会被加锁，因此甚至在程序运行时文件就可以被覆盖。当文件更新后，系统会温和地转换到新的版本。系统检测文件变化，并用新的应用程序代码建立一个新的应用程序实例，然后将引入的请求路由到应用程序。当所有被现存的应用程序实例处理的未完成的请求处理完后，该实例就被销毁了。

在应用程序中，HTTP 请求是通过 HTTP 模块的一个管道路由的，最终到达请求处理程序。HTTP 模块和请求处理程序是一些实现特殊接口的受控类，而这些接口是由 ASP.NET 定义的。这种管道结构使得为应用程序增加服务非常方便：只需补充一个 HTTP 模块。例如，安全，状态管理及跟踪都被实现为 HTTP 模块。高级可编程模块，如网络服务和网络表单，通常被实现为请求处理程序。一个应用程序能链接与多个请求处理程序——每个处理程序一个 URL，但是所有的 HTTP 请求都通过同样的管道路由。

网络基本上是一个无状态模型，并且在 HTTP 请求间没有联系，这使得编写网络应用程序很困难，因为应用程序通常需要维护跨多个请求的状态。ASP.NET 增强了由 ASP 引入的状态管理服务，以便为网络应用程序提供三种类型的状态：应用程序、会话、用户。就象在 ASP 中一样，应用程序状态特定于一个应用程序实例，并且不会持久。会话状态是特定于一个用户与应用程序间的会话的。与 ASP 会话状态不同，ASP.NET 会话状态储存在一个独立的过程中，并且可把它配制成可以储存到一个独立的机器上。这使得会话状态当应用程序在网络群（Web farm）扩展时非常有用。用户状态类似于会话状态，但通常它不会超时，并且是永久性的。因此，用户状态对储存用户参数和其它个性化的信息是有用的。所有状态管理服务都被实现为 HTTP 模块，因此它们容易增加到应用程序管道中，或从中删除。如果除了由 ASP.NET 提供的服务外，还需要额外的状态管理服务，那么可由第三方的模块提供。

ASP.NET 同样提供高速缓冲服务，以改善性能。输出缓冲可完全节省网页翻译，段缓冲储存部分的网页。由于提供了相应的类，所以只要需要，应用程序、HTTP 模块

以及请求处理程序可以在高速缓存中储存任意数量的对象。

下面快速浏览一下建立在 ASP.NET 可编程模块之上的两个高级可编程模块：ASP.NET 网络 表单和 ASP.NET 网络 服务。

l ASP.NET 网络表单

网络表单把基于 Visual Basic®的表单的高生产性的优点带到了网络应用程序的开发中来。网络表单支持传统的将 HTML 内容与角本代码混合的 ASP 语法，但是它提出了一种将应用程序代码和用户接口内容分离的更加结构化的方法。引入的网络表单控件用于为封装通用用户接口元素提供了一种机制。这些新的特点使得开发工具在支持 VB 小应用程序的同时，也支持设计时模块，使得 WUSIWYG 工具支持网页布局。

网络表单控件负责生成用户接口，典型情况是在 HTML 表单中。ASP.NET 是提供了一套映射传统的 HTML 用户接口小部件（包括列表框，文本框和按钮）的网络表单控件和一套附加的更加复杂的网络控件（如日历和广告转板）。这些控件的一个重要特点是，它们可以被编写以适应客户端的能力；同一网页把大范围的客户端平台和表单因素作为目标。换句话说，网络表单控件能“嗅”到正在查找表单的客户，然后返回合适的用户经验－－可能是适合低级浏览器的 HTML3.2 或是适于 IE5.0 的动态 HTML。

考虑到网络是一种无状态的联接模型，网络应用程序开发人员所面临的一个很复杂的问题是，他们要对用户与基于网络的接口的交互作用作出反应。网络利用 ASP.NET 的体系架构提供了一套丰富的服务，以帮助开发人员建立交互式网页。这些服务的净作用是使基于组件的、事件驱动的可编程模块，对开发人员来说，非常象客户端的表单程序设计。用户与网页交互作用的状态管理的复杂性被 ASP.NET 网络 表单和网络表单控件隐藏起来了。对开发人员来说，提供的丰富数据绑定服务使得显示通过数据访问服务得到的数据变得非常容易。

代码与内容的分离使 ASP.NET 网页能动态地编译到受控类中，用以提高性能。每个引入的 HTTP 请求都被传递到一个新的网页实例，因此开发人员不需要关心代码中的线程安全性。

l ASP.NET 网络 服务

ASP.NET 网络 服务体系架构为用 ASP.NET 建立网络 服务提供了一高级可编程模板。虽然建立网络服务并不需要使用网络 服务平台，但是它提供许多的优点将简化开发过程，并且它使用的编程模型对用 ASP 或 VB 工作的开发人员来说是很熟悉的。使用这个可编程模型，开发人员不需要理解 HTTP、SOAP 或其它任何网络服务规范。

开发人员用 ASP.NET 生成一个扩展名为.asmx 的文件，并把此文件配制为网络应用程序的一部分，就建立起了一个网络 服务。ASMX 文件或者包含对在其它地方定义的受控类的引用，或者包含这个类的定义。这个类是由 ASP.NET 提供的 WebService 类所派生。公有的类方法在标记上 WebMethod 属性后，就会成为网络服务方法，把 HTTP 请求发送到 ASMX 文件中的 URL 后，这些方法就会被调用。你不必手工为你的网络服务建立一个契约。当被调用者请求时，ASP.NET 检查类的元数据，以自动生成 SCL 文件。

客户可通过 SOAP，HTTP GET 和 HTTP POST 提交请求。对方法和参数进行编码的约定是：对 HTTP GET，将被编码为查询字符串；对 HTTP POST，将被编码为表单数据。HTTP GET 和 HTTP POST 的机制不如 SOAP 有力，但是它们使得客户在访问

网络服务时不必支持 SOAP。

ASP.NET 网络服务模型假定了一个无状态服务结构。无状态结构通常比有状态结构更具可扩展性。每次收到一个服务请求后，就生成一个新对象，请求被转化为一个方法调用，当方法调用返回时对象被销毁。如果这些服务需要跨请求维护状态，那么它们将使用 ASP.NET 状态管理服务。基于 ASP.NET 的网络服务在网络应用程序模型中运行，因此它们得到了该模型的所有安全、配制和其它优点。

ASP.NET 网络服务还提供了一个为在 SCL 文件中描述的网络服务生成分类的受控代理工具。代理生成器把 SCL 文件中描述的消息映射成受控类中的方法。代理对应用程序代码隐藏了所有的网络和引导设备，因此使用网络服务看起来就象使用其它受控代码一样。代理将优先使用 SOAP 链接网络服务，但是它同样支持 HTTP GET 和 HTTP POST 机制。因此 HTTP GET 和 HTTP POST 同样也能被使用。

网络服务为在 Internet 上绑定应用程序提供了一个利用现存体系架构和应用程序的简单的、灵活的、基于许多标准的模型。网络应用程序很容易与当地开发的服务或已存在的服务集成在一起，而不用考虑开发平台、开发语言或使用的对象模型，以用于实现任何组成的服务或应用程序。

Microsft .NET 框架在现有开发人员技巧之上，提供了一个应用程序模板和关键技术，用于简化安全、可靠、可扩展、高可用性的网络服务的建立、部署和不断的发展。

通过上面的介绍，我们能够感觉到 MicroSoft .NET 对于我们今后的程序设计将产生巨大的影响。

1.1.3 ASP.NET 历史

我们在讲述 ASP.NET 历史之前，让我们来回顾一下 ASP。

ASP 的第一个版本是 0.9 测试版。它给 WEB 开发带来一阵暴风，它能够将代码直接嵌入 HTML，使得设计 WEB 页面变得更简单，更强大，并且通过内置的组件能够实现强大功能，最明显的就是 ActiveX Data Objects （ADO），它使得建立一个动态页面如小孩子玩游戏一样简单。

最终出场的是 Active Server Page 1.0，它做为 IIS 的附属产品免费发送。并且不久就在 Windows 平台上广泛使用。ASP 与 ADO 的结合使用开发者很容易地在一个数据库中建立和打开一个记录集。这不无疑是它如此快就被大众接受的因素，因为你现在能使用这些脚本建立和打开一个记录集，处理和输出任何数据，以任何顺序，几乎只要你能想到的，它就能完成。

1998 年，微软公司又发布了 ASP 2.0 。ASP 1.0 和 ASP 2.0 主要区别是外部的组件需要实例化。有了 ASP 2.0 和 IIS 4.0，我们就有可能建立 ASP 应用了，而且每个组件就有了自己单独的内存空间。内置的 MicroSoft Transaction Server(MTS)也使用制做组件便得简单。

微软公司接着开发了 Windows 2000 操作系统。这个 Windows 版本给我们带上了 IIS 5.0 以及 ASP 3.0。此次并不是简单对 ASP 进行补充，核心的不同实际上是把很多的事情交给了 COM 来做。在 windows 2000 中，微软结合了 MTS 与 COM 核心环境做出了 COM+，这就让主机有了一种新的方法来使用组件，同样给主机带来了更多的稳定性，成了一个可以升级的效率高的工作平台。IIS 5.0 在表面上似乎没有改什么，但是在接口上动的手术比较大。在内部，它使用 COM+ 组件服务来对组件提供一个更好的执行的环境。

有了这些，微软公司推出了 ASP.NET，ASP.NET 又叫 ASP.NET，他不是 ASP 的简单升级，而是 MicroSoft 推出的新一代 Active Server Pages。ASP.NET 是微软发展的新的体系结.NET 的一部分，其中全新的技术架构会让每个人的编程生活变得更的简单

1.1.4 小结

在本章中，我们介绍了微软.NET 的历史，以及对.NET 的构成、性能进行了一个详细的介绍，同时，我们还详细介绍了 ASP.NET 的历史。在下面的章节中，我们将按实例一步一步的讲解 ASP.NET。

第二章 .NET 的安装与运行环境

1.2.1 运行环境配置

l ASP.NET 的调试环境

操作系统：

Windows 2000 Professional，Windows 2000 Server ，Windows 2000 Advanced Server

浏览器：

IE 5.5

NGWS

l 支持哪几种语言

ASP.NET 目前能支持 3 种与语言， C# (读作 "C Sharp")， Visual Basic，and Jscrip。.

l 使你的机器持 ASP.NET，必须满足以下配置：

硬件要求：

1、 CPU: Intel Pentium II-class 300 MHz (最好 Intel Pentium III-class 600 MHz)
2、 内存: 96 MB (最好 128 MB)
3、 磁盘空间: 250 MB(完全安装) 155 MB(快速安装)
4、 显示: 800x600， 256 colors
5、 CD-ROM: required

软件要求：

1、 MicroSoft Windows 2000 + SP1
2、 MicroSoft Internet Explorer 5.5
3、 IIS5.0
4 、 其它: MDAC 2.6 Beta 2

l .NET 是运行库，还是开发平台？

微软的宏伟目标是让 MicroSoft.NET 彻底改变软件的开发方式、发行方式、使用方式等等，并且不止是针对微软一家，而是面向所有公司！2000 年 7 月份在 PDC 展会上分发的是“.NET 架构”包，“.NET 架构”是 MicroSoft.NET 计划中首先问世的一部分，它包括了两方面的组件：“.NET 通用运行库”和“.NET 类库”。最近传来好消息说这两个组件已经被打包到“.NET 架构 SDK”中，放在微软的站上免费供大家下载，有兴趣的朋友一定要去试试看哦！另外，这个 SDK 中还包括 C#、C++、JavaScript 和 VB 的命令行编译器，使用这些编译器就可以开发应用程序和组件了，从这个角度来看，.NET 架构首先是一个开发平台，因为它提供了运行库和类库，并且，下一个即将面市的就是 Visual Studio.NET，其中包括了更加全面的 SDK 和图形化的开发界面、向导、工具等等，更象一个开发平台了。但是.NET 的运行库其实已经融合到操作系统中，所以说它为运行库也是可以的。

▌什么是 NGWS?

ASP.NET 实际上是一个崭新的运行结构的一部分，这个结构提供对所有 windows 应用程序的支持。这个结构是 MicroSoft's Next Generation Web Services （ NGWS）关建部分。当你安装了这个结构，你就获得了 ASP.NET。这个结构同样支持所有其它服务器程序技术。

NGWS 结构通过对可升级分布式应用添加 [新的和增强的服务] 来扩展 COM 的结构，此种结构常用做编写可重复调用的可共同使用的软件组件，这些新的和增强的服务有：

一套统一的丰富的程序库
一个支持多语言的运行引擎
简单地应用建立，调试，以及维护
对分布式应用加强了可升级性
保护现已存在的软件和投资

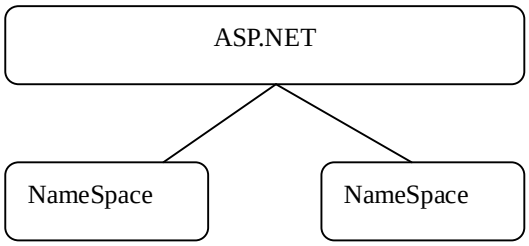
▌在 ASP.NET 引入了 namespace 的概念，那么 namespace 是什么？

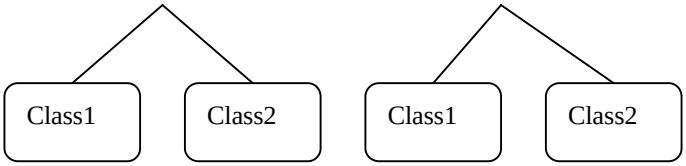
对象一直是 Windows 开发环境中，程序开发的中心。不论在 VB、VBA、VC++、VBScript 等，都是如此，不同的开发环境有不同的对象，这些对象均是各个语法所提供的的“资源”，程序开发人员可以利用这些资源，来编写所需的系统，就象我们在盖房屋的一样，建筑师使用使用同样的素材，然而盖好的房子可能不尽相同。

在过去的 SP 中，仅有 Server、Request、Response…等七个对象。而在 ASP.NET 的对象库中却分得很细。

例如在 ASP.NET 网页中要通过 SQL 语句获得数据库中的数据，必须使用“System.Data.SQL”，这是 NameSpace 名称。 在 System.Data.SQL 下，又有很多类（Class）。每个 Class 可视为一个对象，因为 Class 下有属性、方法和事件等

所以，最上层的 NameSpace 是看作是同类型对象的集合，一个 NameSpace 之下可拥有多个 Class。他们之间的关系如图：





通过此图，我们了解了 NameSpace 及 Class 的概念，二者分别是表示对象集合和对象。

- **如何应用名字空间（NameSpace）？**

```
<%@ Import Namespace="System.Globalization"%>

<%@ Import Namespace="DataEmployee" %>

<%@ Import Namespace="System.Data" %>

<%@ Import Namespace="System.Data.ADO" %>
```

以上表示在 ASP.NET 网页中使用了四个 NameSpace，接下来我们要申明变量，但此变量必须是已引用的四个 NameSpace 所属的 Class，如：

```
Dim MyConnection As ADOConnection

Dim MyCommand As ADODataSetCommand
```

说明：ADOConnection 及 ADODataSetCommand 都是 System.Data.ADO 之下的 Class。

- **ASP.NET 中的文件类型？**

ASP 的文件类型只有一种，其扩展名是.asp 文件。那么在 ASP.NET，就有很多的文件名：

文件扩展名	用途及说明
Global.asax	ASP.NET 系统环境设置文件，相当与 ASP 中的 Global.asa。
.aspx	内含 ASP 程序代码的文件，如同過去的.asp，浏览器可执行此类文件，向服务器 提出浏览請求
.asmx	制作 Web Service 的原始文件
.sdl	制作 Web Service 的 XML 格式的文件
Vb 或.cs	在非 ASP.NET 环境下，执行 Web Service 的文件
.aspc	可重覆使用在多个.aspx 的文件，此文件内可

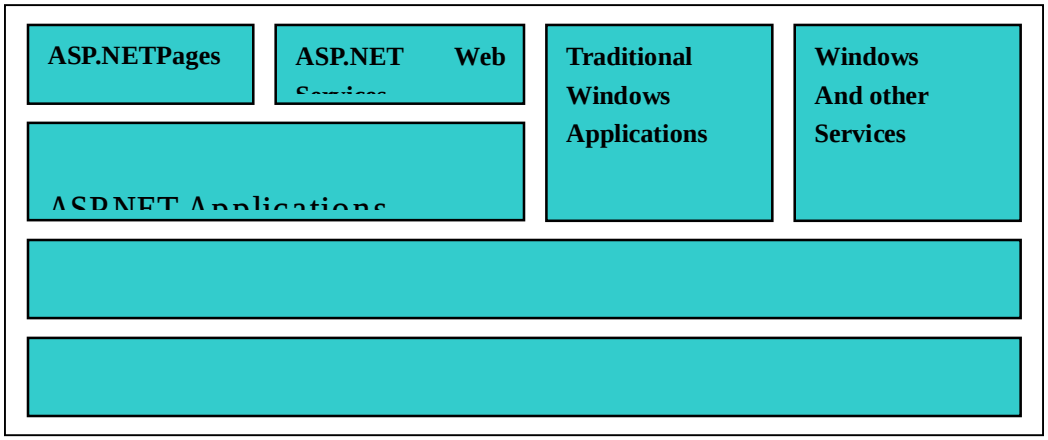
	含有控件
.ascx	内含 User Control 的文件，可内含在多個.aspx 文件中

l 什么是 NGWS Framework?

ASP 的综合性能明显的要好于以前的版本。到目前为至，ASP 是通过一个名叫 asp.dll 的 ISAPI DLL 来执行的，另外还加上一些系统文件和 ASP 用户组件。

这个新的 NGWS 结构反映了行业信息技术观点对于建立，调试以及维护各种 WEB 服务的需要的转变，这些服务包括简单的客户应用到复杂的分布式结构。上面所有的概念和策略只是 Windows Distributed Internet Applications (DIA)部分结构.

在这里我们最重要需要认识的问题是这里所说的结构（framework)不是我们所说 ASP.NET。它只是做为 windows 系统中所有应用的基础。下面的图表给我们演示了 framework 是如何支持 ASP.NET 应用的。



l ASP.NET 对于 asp 来说有什么突破呢？

u 运行机制不同

asp 属于一种解释型的编程框架，它的核心是 vbs 和 js，受这两种脚本语言的限制，决定了 asp 先天不足，它无法进行象传统编程语言那样的底层操作，所以如果你需要进行一些诸如 socket、文件等的操作时不得不借助于用其他传统编程语言如 C++、VB 、JAVA 等编写的组件，并且由于它是解释执行的，所以在运行效率上大打折扣。而 ASP.NET 呢，它是一种编译型的编程框架，它的核心是 NGWS runtime ，除了和 asp 一样可以采用 vbs 和 js 作为编程语言外，还可以用 VB 和 C# 来编写，这就决定了它功能的强大，可以进行很多低层操作而不必借助于其他编程语言。

u 执行效率

由于它是编译后运行的，所以执行效率要比 asp 高得多。

l C#编译器选项全解

可以使用 CSC.exe/?来察看可选项.

u 输出文件相关选项 ：

/out:<file>　　输出文件名(如果不指定则从第一个源文件名中取得)

/target:exe　　建立一个控制台可执行程序(这是默认选项)(可以缩略写作 /t:exe)
/target:winexe　　建立一个 windows 可执行程序(可以缩略写作 /t:winexe)
/target:library　　建立一个库(可以缩略写作 /t:library)
/target:module　　建立一个可以加到其他汇编文件的模块(可以缩略写 作 /t:module)
/win32icon:<file> 指定一个图标作为输出文件的图标
/nooutput[+|-]　　只检查代码中的错误，并不生成可执行程序
/define:<symbol file>　　定义条件编译符号(可以缩略写作 /d)
/doc:<file>　　生成 XML 文档

- 输入文件相关选项：

/recures:<wildcard>　　包括当前目录及其子目录下所有符合指定的通配符规则的文件
/main:<type>　　指定包含入口点的类型(忽略其他所有可能的入口点)(可以缩略写作 /m)
/reference:<file list>　　参考由给出的汇编文件所指定的元数据(可以缩略写作 /r)
/addmodule:<file list>　链接指定的模块到汇编文件中

- 资源相关选项 ：

/resource:<resinfo>　　嵌入特定的资源(可以缩略写作 /res)
/linkresource:<resinfo>　　链接指定的资源到汇编文件中(可以缩略写作 /linkers)

- 代码生成相关选项

/debug[+|-]　　产生调试信息
/optimize[+|-]　　提供优化(可以缩略写作 /o)
/incremental[+|-]　　进行增量编译，也就是只编译改变的部分(可以缩略写作 /incr)

- 错误和警告相关选项

/warnaserror[+|-]　　对警告与错误作相同处理
/warn:<n>　　设定警告级别(0-4)(可以缩略写作 /w)
/nowarn:<warning list>　　禁止特定的警告消息

- 语言相关选项

/checked[+|-]　　对上溢和下溢进行检查
/unsafe[+|-]　　允许"不安全"的代码

- 其他方面的选项

@<file>　　读取相应文件以获取更多选项
/help　　显示帮助文件(可以缩略写作 /?)
/nologo　　禁止编译版权信息

- 增强的选项

/baseaddress:<address>　　指定被编译库的基地址
/win32res:<file>　　通常用来指定存放版本和图标信息的 WIN32 资源文件
/bugreport:<file>　　建立"错误报告"文件
/codepage:<n>　　指定打开源文件时使用的代码页
/fullpath　　指定程序生成的完整路径

/nostdlib[+|-]　　不参考标准库(mscorlib.dll)

1.2.2 Visual Studio.NET 7.0 安装

Visual Studio.NET 7.0 的安装，机器必须满足下面的要求，

1、硬件要求：

① CPU: Intel Pentium II-class 300 MHz (最好 Intel Pentium III-class 600 MHz)
② 内存: 96 MB (最好 128 MB)
③ 磁盘空间: 250 MB(完全安装) 155 MB(快速安装)
④ 显示: 800x600, 256 colors
⑤ CD-ROM: required

2、软件要求：

① MicroSoft Windows 2000 + SP1
② MicroSoft Internet Explorer 5.5
③ IIS5.0
④ 其它: MDAC 2.6 Beta 2

ASP.NET 的安装过程很简单，只需按照简单提示安装即可。但是，如果你的机器安装了 OFFICE2000，在此建议安装 ASP.NET 之前先备份\MicroSoft Office\Office\mso9.dll 这个文件，因为安装完 ASP.NET 后，OFFICE 会提示你注册，否则的话 OFFICE2000 就会出现限制使用 50 次。此时将备份的 mso9.dll 文件覆盖掉原来的文件即可。

ASP.NET(NGWS SDK)的下载地址：

http://download.microsoft.com/download/platformsdk/Trial/1812.10full/NT5/EN-US/Setup.exe

安装微软的 VisualStudio.NET Beta1 和安装 ASP.NET 很多地方有惊人的相似，所以在此简单地提一下。

安装 beta1 版本的记得必须先安装以下内容：

1、windows2000 sp1
2、安装 IE5.5
3、必须要装有 iis,而且 iiS 要带 front page 扩展
4、front page 服务扩展的补丁 QFE

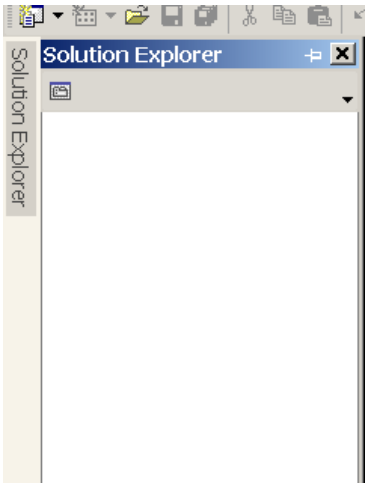
相同地，安装完 VisualStudio.NET 后同样会出现 OFFICE2000 的 50 次限制，所以可以用同样的方法，先备份 mso9.dll 文件，然后安装完后覆盖掉原来的文件。

1.2.3 运行环境 IDE

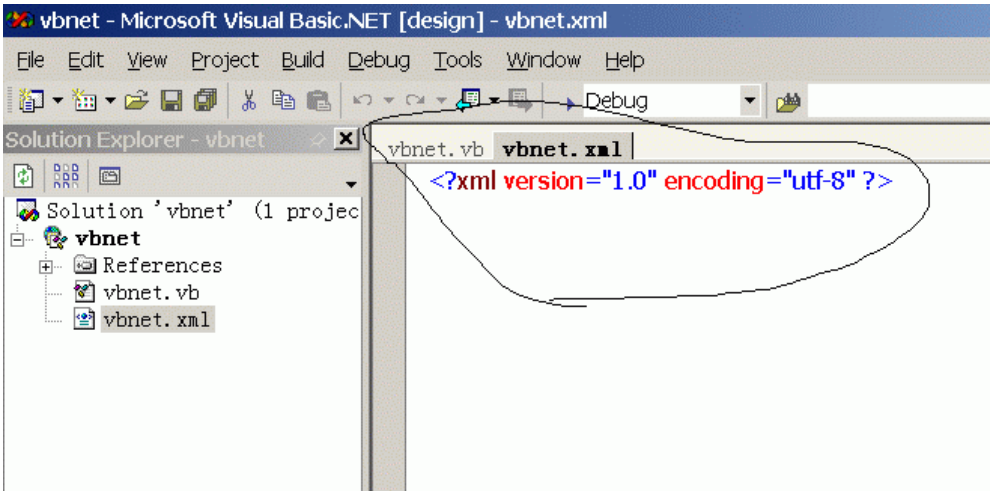
微软的 vs.net7.0 IDE 是一个非常丰富的变成环境，可以进行 C#/VC++、VB.NET、ASPX 等的编程，你甚至也可以编写 ASP 文件。

你首先看到的是 IDE，IDE 看起来很熟悉，开发 VS.NET IDE 的开发人员以前曾开发过 VB 的 IDE，它在 VB IDE 的基础上又有了新的提高。

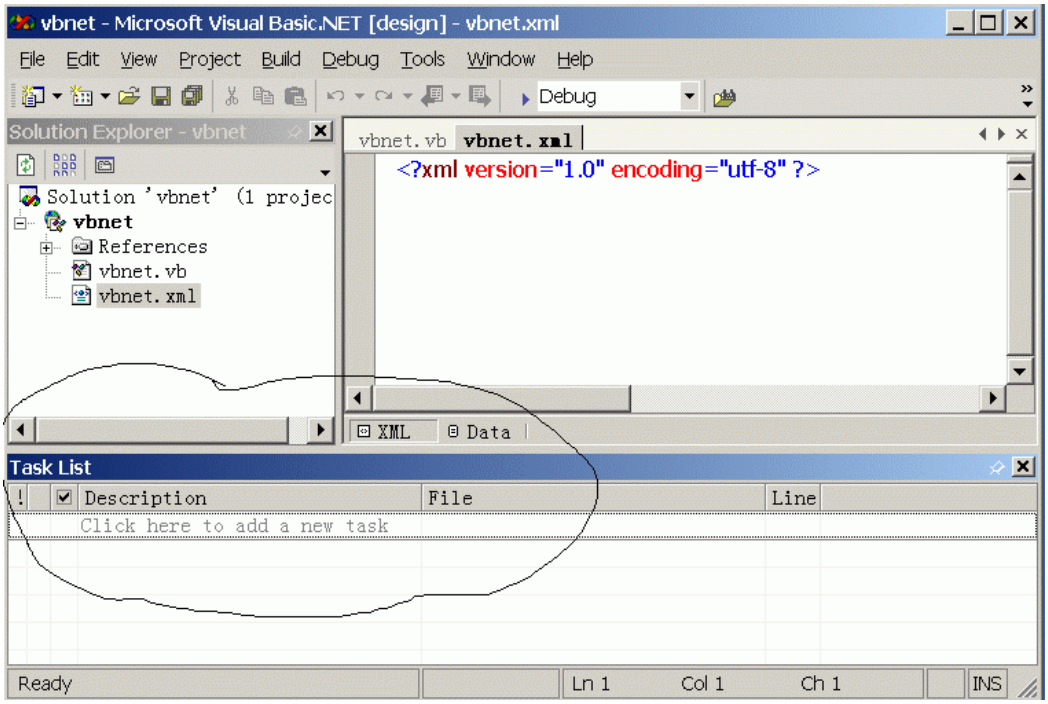
然而，IDE 的变化绝非是表面性的。所有的.NET 语言都使用同一个 IDE，其中的新工具的功能是强大而全面的，你可以把任何一个设计窗口设定为自动隐藏（就象 Windows 中的任务条一样），这样就可以使桌面显得不太凌乱，如下面所示：



主工作区是一系列的标签，也就是说 IDE 不会同时显示许多的窗体或代码模块，在打开对象的源代码时，IDE 就会在相应对象的主区内增加新按钮，如下面所示：



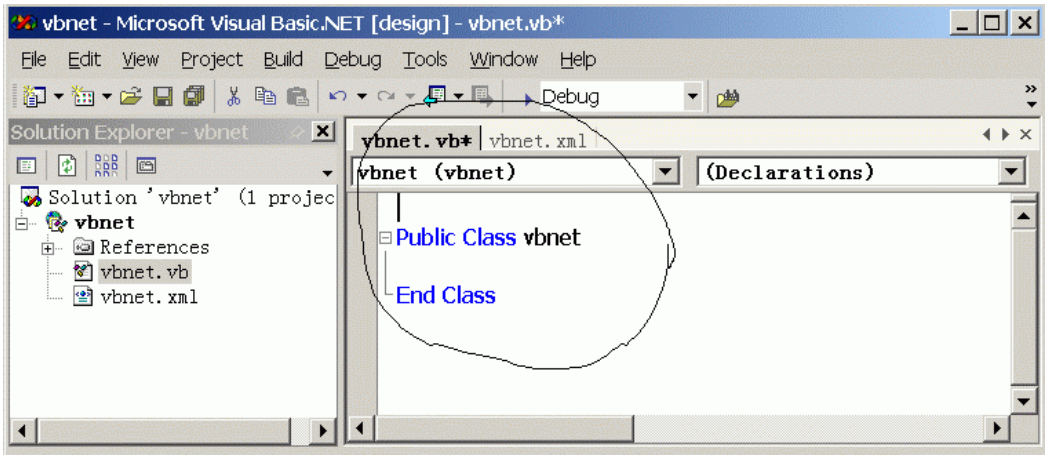
IDE 中还有一个新的被称作任务清单的窗口，其中的内容包括你和 IDE 创建的项目。例如，如果在编译一个 VB 项目时收到一个错误信息，VB 就会在任务清单中创建一个条目来解释这个问题，如下所示：



你还可以直接在任务清单中添加一个条目，或者通过在代码中建立以"TODO:"开头的注释把代码中的一个位置与任务联系起来。我非常喜欢微软添加的任务清单，它能使我节约不少的时间，并有助于我能够更好地调试自己的软件。

另一个会立刻感受到的变化是.NET IDE 中的窗体。微软抛弃了原来的窗体引擎，而采用了 Windows 风格的窗体，所有的基于 CLR 的语言都使用 Windows 的窗体引擎，与 VB6 等中的使用的窗体引擎相比，它有几个明显的优点。例如，Windows 的窗体可以自动地改变其中的组件的大小，而且可以把控制锁定在特定的位置，也就是说，我们无需借助第三方的工具来完成相应的工作了。另外，Windows 的窗体还可以使我们完成另外一些很"酷"的工作，例如创建透明的窗体。

过去，VB 隐藏了创建一个窗体所必需的全部工作。我们使用 IDE 创建一个窗体，并在 Initialize 事件处理程序中添加代码，但对于发生在这两者之间的过程则无能为力。在 VB.NET 中，窗体成了一个类，包含创建窗体的全部代码，我认为这些代码是"鸡肋"，原因是大多数的开发人员都不想去理它。如果说有一种东西一定能让你的软件出问题，那就是这些代码了。一些高级开发人员可以通过这些代码完成一些很"酷"的工作，因为它可以让你"看到"VB 创建窗体的全部情况。如果不想看，你并非必须看这些代码，新的代码编辑器可以扩展或消除一些代码区，在缺省状态下这些代码是不会显示的。代码编辑器还包括一些新的特性，例如它可以自动地对编辑的源代码进行"缩进"处理，而且可以显示源代码的行号，如下面所示：



还有有了这个 IDE 之后，我们就可以不用手工编写编译语句了，直接就可以把我们的.vb 或者.cs 文件编译成.dll 或者.exe 文件，等等。

总之，微软的.NET IDE 是一个很酷的编程环境，如果一个一个的介绍，那可得写几本书了，大家只有多用才会熟练啊。

总之，微软的.net 是一个很酷的变成环境，如果一个一个的介绍，那可的小写基本书了，大家只有多用才会熟练。

1.2.4 小结

MicroSoft .NET 的策略是将互联网本身作为构建新一代操作系统的基础，对互联网和操作系统的设计思想进行合理延伸。这样，开发人员必将创建出摆脱设备硬件束缚的应用程序，以便轻松实现互联网连接。MicroSoft .NET 无疑是当今计算机技术通向计算时代的一个非常重要的里程碑。通过上面的介绍，相信大家对.net 以及 asp.net 有了一定的了解。在下面的内容，我们将带大家进入一个崭新的 ASP.NET 世界。

第二篇 WEB 页面

第一章 WEB 页面简介

2.1.1 WEB FORM

表单，英文单词是 Form，学习过 VB 的朋友一定不会陌生。在 MS.NET 架构里，Form 是一个经常使用到的词汇。比如：编写 Windows 应用时会提到 Windows Form，编写 Web 应用时会提到 Web Form。Windows Form 可以看作一个 Windows 窗体，这和在 VB 里面一样。而 Web Form 则代表了一个一个的 Web 页面。总的看来，Form 就像是一个容纳各种控件的容器，各种控件都必须直接或者间接的和它有依存关系。Form 在这里译作“WEB 表单”似乎有些不妥。“表单”这个词，在 WEB 程序员看来，总是和 HTML 里面的“Form”相混淆。“WEB 表单”似乎翻译成“WEB 页面”更加妥当一些。

大家还记得 VB 里面的 Form 实际上就是一个对象吧，它可以有自己的属性、方法、事件等等。WEB 表单，或者说 WEB 页面，实际上是一个“对象” (Object)。MS.NET 架构里面一个比较重要的概念就是“对象”：所有的控件都是对象，甚至数据类型都成了对象；每种数据类型都有自己特有的属性和方法。我们在后面的编程中将可以体会到。

WEB FORM 的后缀名是 ASPX。当一个浏览器第一次请求一个 ASPX 文件时，WEB FORM 页面将被 CLR（common language runtime）编译器编译。此后，当再有用户访问此页面的时候，由于 ASPX 页面已经被编译过，所以，CLR 会直接执行编译过的代码。这和 ASP 的情况完全不同。ASP 只支持 VBScript 和 JavaScript 这样的解释性的脚本语言。所以 ASP 页面是解释执行的。当用户发出请求后，无论是第一次，还是第一千次，ASP 的页面都将被动态解释执行。而 asp.net 支持可编译的语言，包括 VB.NET、C#、Jscript.NET 等。所以，asp.net 是一次编译多次执行。

为了简化程序员的工作，ASPX 页面不需要手工编译，而是在页面被调用的时候，由 CLR 自行决定是否编译。一般来说，下面两种情况下，ASPX 会被重新编译：

1．ASPX 页面第一次被浏览器请求；

2．ASPX 被改写

由于 ASPX 页面可以被编译，所以 ASPX 页面具有组件一样的性能。这就使得 ASPX 页面至少比同样功能的 ASP 页面快 250%！

下面我们来看一下简单的 WEB 页面。

2.1.2 我的第一个 Page

把下面的代码拷贝到 myfirstpage.aspx 文件中，然后从浏览器访问这个文件：

```
<!--源文件：form\web 页面简介\myfirstpage.aspx-->
<form action="myfirstpage.aspx" method="post">

    <h3> 姓名: <input id="name" type=text>

    所在城市:  <select id="city" size=1>
```

```
            <option>北京</option>
            <option>上海</option>
            <option>重庆</option>
        </select>

    <input type=submit value="查询">

</form>
```

你可能觉得这个页面太简单了，用 HTML 就可以完成。是的！微软建议你将所有的文件哪怕是纯 HTML 文件都保存为 ASPX 文件后缀，这样可以加快页面的访问效率！不仅仅是在 asp.net 环境中，在 IIS5.0 以后的 ASP3.0 就已经支持这个特性了。

由于我们没有对表单提交做任何响应，所以，当你按下“查询”按钮，页面的内容没有什么改变。

下面我们将逐步使用 asp.net 的思考方式，来完成我们的页面。

2.1.3 WEB 页面处理过程

这一节我们将深入到 asp.net 内部，看看页面是怎样被处理的。

和所有的服务器端进程一样，当 ASPX 页面被客户端请求时，页面的服务器端代码被执行，执行结果被送回到浏览器端。这一点和 ASP 并没有太大的不同。

但是，asp.net 的架构为我们做了许多别的事情。比如，它会自动处理浏览器的表单提交，把各个表单域的输入值变成对象的属性，使得我们可以像访问对象属性那样来访问客户的输入。它还把客户的点击映射到不同的服务器端事件。

了解 WEB 页面的处理过程很重要。这样你可以仔细地优化你的代码，提高代码的效率。

2.1.3.1 页面的一次往返处理

用户对 Server Control 的一次操作，就可能引起页面的一次往返处理：页面被提交到服务器端，执行响应的事件处理代码，重建页面，然后返回到客户端。

正因为每个 Control 都可能引发一次页面的服务器端事件，所以，asp.net 尽量减少了控件的事件类型。很多组件都只有 OnClick 事件。特别的，asp.net 不支持服务器端的 OnMouseOver 事件。因为 OnMouseOver 事件发生得非常频繁。所以，支持服务器端的 OnMouseOver 事件是非常不现实的。

2.1.3.2 页面重建

每一次页面被请求，或者页面事件被提交到服务器，asp.net 运行环境将执行必要的代码，重建整个页面，把结果页面送到浏览器，然后抛弃页面的变量、控件的状态和属性等等页面信息。当下一次页面被处理时，asp.net 运行环境是不知道它的上一次执行情况的。在这个意义上，ASPX 页面是没有状态的。这也是 HTTP 协议的特点（为了加速页面的访问，在 asp.net 页面里面可以使用缓存机制，也就是保存页面的执行结果，下一次页面被请求时，直接送回上一次的执行结果。）。

在 ASP 中，当页面被提交到服务器端时，只有那些用户输入的值被传递到服务器。其

他的比如组件的属性、变量的值，是不会传递的。所以服务器无法了解组件的进一步的信息。

在 asp.net 中，页面对象的属性、页面控件的属性被称为“view state”（页面状态）。页面状态在 asp.net 中被受到特别关照。请看服务器端(**page1.aspx**)的代码：

```
<!--源文件：form\web 页面简介\page1.aspx-->
 <HTML>
 <BODY>
 <SCRIPT language="VB" runat="server">
   Sub ShowValues(Sender As Object, Args As EventArgs)
     divResult.innerText = "You selected '" _
 & selOpSys.value & "' for machine '" _
     & txtName.value & "'."
     End Sub
 </SCRIPT>
 <DIV id="divResult" runat="server">
</DIV>
<FORM runat="server">
机器名：
<INPUT type="text" id="txtName" runat="server">
   <P />
 操作系统：
       <select id="selOpSys" size="1" runat="server">
           <OPTION>Windows 95</OPTION>
          <OPTION>Windows 98</OPTION>
          <OPTION>Windows NT4</OPTION>
           <OPTION>Windows 2000</OPTION>
        </SELECT>
      <P />
 <INPUT type="submit" value="Submit" runat="server" onserverclick="ShowValues">
 </FORM>
   </BODY>
</HTML>
```

运行后将自动被解释成客户端代码，如下：

```
<HTML>
<BODY>
You selected 'Windows 98' for machine 'iceberg'.
<FORM name="ctrl0" method="post" action="pageone.aspx" id="ctrl0">
<INPUT type="hidden" name="__VIEWSTATE" value="a0z1741688109__x">
机器名:
<INPUT type="text" id="txtName" name="txtName" value="tizzy">
        <P />
操作系统:
<SELECT id="selOpSys" size="1" name="selOpSys">
   <OPTION value="Windows 95">Windows 95</OPTION>
```

```
      <OPTION selected value="Windows 98">Windows 98</OPTION>
          <OPTION value="Windows NT4">Windows NT4</OPTION>
    <OPTION value="Windows 2000">Windows 2000</OPTION>
      </SELECT>
 <P />
   <INPUT type="submit" value="Submit">
   </FORM>
 </BODY>
</HTML>
```

对于上面的代码，服务器端控件能在服务器端脚本中被自由运用。如果我们用传统的 ASP 代码实现上述的功能的话：

```
If Len(Request.Form("selOpSys")) > 0 Then
    StrOpSys = Request.Form("selOpSys")
    StrName = Request.Form("txtName")
    Response.Write("You selected '" & strOpSys _
    & "' for machine '" & strName & "'.")
End If
```

如果我们用 asp.net 的话，程序代码如下：

```
 If Len(selOpSys.value) > 0 Then
    Response.Write("You selected '" & selOpSys.value _
    & "' for machine '" & txtName.value & "'.")
 End If
```

通过上面例子不难看出：asp.net 页面具有组件方式的方便性和灵活性。

请注意：asp.net 通过把页面的状态封装到一个隐藏的输入域，从而可以在不同的页面之间实现传递页面的状态。

另外，asp.net 也支持应用程序一级的状态管理。这个特性在 ASP 中就已经实现。

2.1.3.3 页面处理内部过程

我们来看看页面处理的内部过程。下面的过程是依次进行的：

2.1.3.3.1 Page_load

首先，页面的状态被恢复，然后触发 Page_OnLoad 事件。在这个过程中，你可以读取或者重置页面的属性和控件的属性，根据 IsPostBack 属性判定页面是否为第一次被请求，执行数据绑定，等等。

现在我们通过一个具体的例子，来详细讲述 Page_load 事件：

我们所做的这个例子关于用户登录的。

我们先来看 page.aspx 的代码：

```
<!--源文件：form\web 页面简介\page.aspx-->
<%@ Register TagPrefix="Acme" TagName="Login" Src="page.ascx" %>
    <html>
    <title>登录演示</title>
    <script language="VB" runat="server">
```

```
    Sub Page_Load(Sender As Object, E As EventArgs)
      If (Page.IsPostBack)
      MyLabel.Text &= "用户名： " & MyLogin.UserId & "<br>"
      MyLabel.Text &= "密码：  " & MyLogin.Password & "<br>"
      End If
    End Sub
</script>
<body style="font: 10pt verdana">
 <center> <h3>登录</h3></center>
  <form runat="server">
    <Acme:Login id="MyLogin" UserId="" Password="" BackColor="beige" runat="server"/>
  </form>
  <asp:Label id="MyLabel" runat="server"/>
 </body>
</html>
```

在这个文件中，我们使用了 Page_OnLoad 事件的 IsPostBack 属性，用来显示用户登录时的用户名和密码。

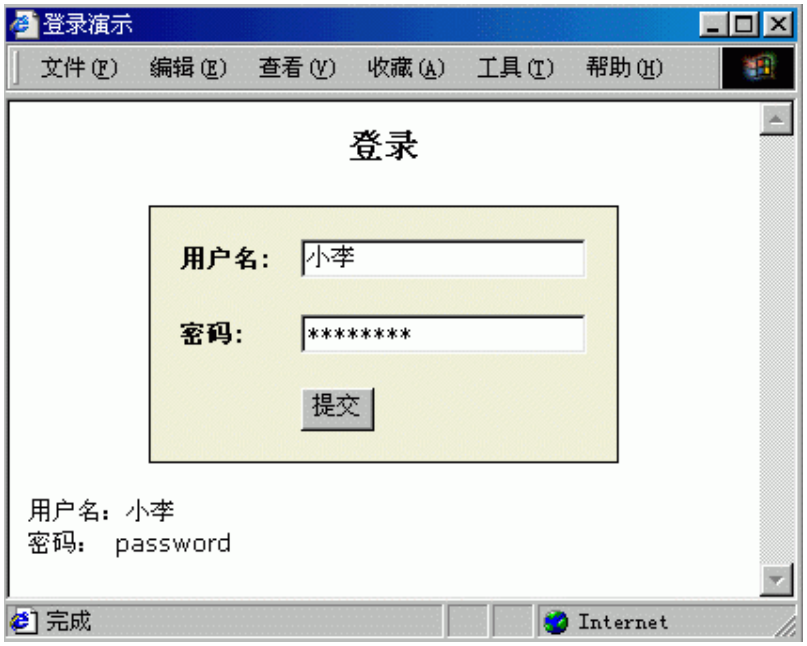
在来看一下 page.ascx 文件：

```
<!--源文件：form\web 页面简介\page.ascx-->
<script language="VB" runat="server">
Public BackColor As String = "white"
Public Property UserId As String
  Get
    Return UserName.Text
  End Get
  Set
    UserName.Text = Value
  End Set
End Property
Public Property Password As String
  Get
    Return Pass.Text
  End Get
  Set
    Pass.Text = Value
  End Set
End Property
 </script>
<center>
<table style="background-color:<%=BackColor%>;font: 10pt verdana;border-width:1;
    border-style:solid;border-color:black;" cellspacing=15>
  <tr>
    <td><b>用户名: </b></td>
    <td><ASP:TextBox id="UserName" runat="server"/></td>
```

```
    </tr>
    <tr>
      <td><b>密码: </b></td>
  <td><ASP:TextBox id="Pass" TextMode="Password" runat="server"/></td>
    </tr>
    <tr>
      <td></td>
      <td><ASP:Button Text="提交" runat="server"/></td>
    </tr>
</table>
</center>
```

在这个文件中，我们设置了控件的属性。使之能在 page.aspx 中调用

程序的运行如图：



在下一个例子中，我们将使用 Page_OnLoad 事件，来执行数据绑定：

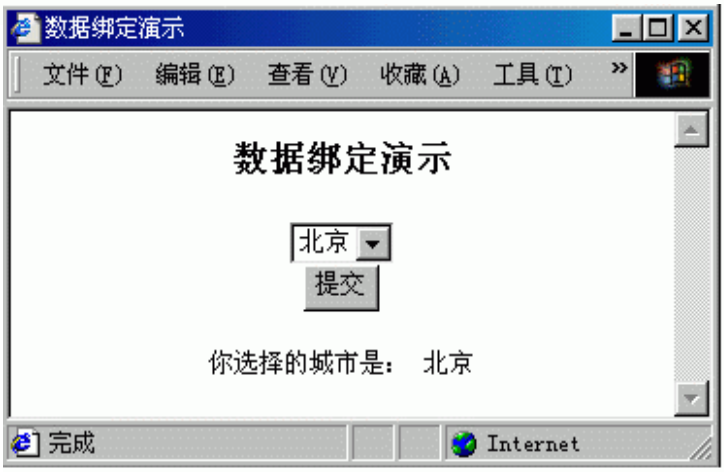
文件 **databind.aspx** 代码如下：

```
<!--源文件：form\web 页面简介\databind.aspx-->
<html>
<head>
<title>数据绑定演示</title>
  <script language="VB" runat="server">
      Sub Page_Load(sender As Object, e As EventArgs)
          If Not IsPostBack Then
              Dim values as ArrayList= new ArrayList()
              values.Add ("北京")
              values.Add ("上海")
              values.Add ("杭州")
              values.Add ("成都")
              values.Add ("重庆")
              values.Add ("西安")
              DropDown1.DataSource = values
              DropDown1.DataBind
          End If
      End Sub
   ‘定义按钮的单击事件
     Sub SubmitBtn_Click(sender As Object, e As EventArgs)
     ’结果显示
          Label1.Text = "你选择的城市是：  " + DropDown1.SelectedItem.Text
      End Sub
 </script>
```

```
</head>
<body>
<center><h3><font face="Verdana">数据绑定演示</font></h3></center>
      <form runat=server>
<center><asp:DropDownList id="DropDown1" runat="server" /></center>
<center><asp:button Text="提交" OnClick="SubmitBtn_Click" runat=server/></center>
            <p>
<center><asp:Label id=Label1 font-name="Verdana" font-size="10pt" runat="server" /></center>
    </form>
  <body>
</html>
```

程序运行效果如图：

当我们点击“提交”按钮时：



在下面的例子中，我们将用 page_load 事件来对数据库进行连接：

我们还要说明的是如果使用 SQL 语句对数据库进行操作的时候，就需要在页面中导入 System.Data 和 System.Data.SQL 名字控件，文件 **pagedata.aspx** 的代码如下：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
```

程序代码如下（pagedata.aspx）：

```
<!--源文件：form\web 页面简介\pagedata.aspx-->
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<html>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    Dim DS As DataSet
    Dim MyConnection As SQLConnection
    Dim MyCommand As SQLDataSetCommand
    ‘同数据库进行连接，采用 sql server 数据库
    MyConnection = New SQLConnection("server='iceberg';uid=sa;pwd=;database=info")
    ‘执行 SQL 操作
    MyCommand = New SQLDataSetCommand("select * from infor",MyConnection)
    DS = New DataSet()
    MyCommand.FillDataSet(ds, "infor")
    MyDataGrid.DataSource=ds.Tables("infor").DefaultView
    MyDataGrid.DataBind()
End Sub
</script>
<center>
<body>
  <h3><font face="Verdana">Page_load 事件演示</font></h3>
  <ASP:DataGrid id="MyDataGrid" runat="server"
    Width="600"
    BackColor="white"
    BorderColor="black"
    ShowFooter="false"
    CellPadding=3
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#aaaadd"
    MaintainState="false"
  />
/body>
</center>
```

</html>

在这个程序中，我们在 page_load 事件中，我们做了哪些事呢？
① 与数据库连接。在这个例子中，我们使用 SQL Server 作为后台数据库。在这个库中，我们建立了 info 数据库，在数据库中有一张 infor 表。
② 执行 SQL 操作
③ 将筛选后的数据显示出来
我们再来看看程序运行的效果：

上面就是对 Page_load 事件的介绍，相信大家通过例子能对该事件有个理解。

2.1.3.3.2 事件处理

这一阶段处理表单的事件。你可以处理特定的事件，也可以在表单需要校验的情况下，根据 IsValid 属性判定页面的输入是否有效。

Web Form 提供了一些具有验证功能的服务器控件。这些控件提供了一套简单易用并且很强大的功能能检查输入时是否有错误。而且，还能显示提示信息给用户。

对于每个控件来说，都有一特定的属性，来验证输入的值是否有效。我们来看一下对输入控件需要验证的属性：

控件	需要验证的属性
HtmlInputText	Value
HtmlTextAreaHtm	Value
HtmlSelect	Value
HtmlInputFile	Value
TextBox	Text
ListBox	SelectedItem
DropDownList	SelectedItem
RadioButtonList	SelectedItem

好了，有了上面的介绍，我们就以例子来讲解表单的有效性验证。

在下面一个简单的例子中，我们将对用户的输入验证。

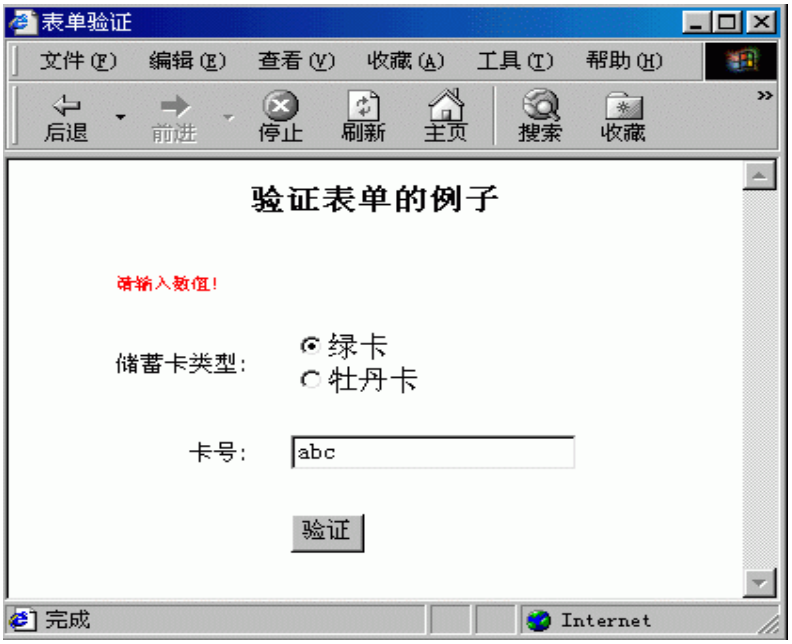
如 Validate.aspx 的内容如下：

```
<!--源文件：form\web 页面简介\validate.aspx-->
<html>
<head>
    <script language="VB" runat="server">
        Sub ValidateBtn_Click(sender As Object, e As EventArgs)
            If (Page.IsValid) Then
                lblOutput.Text = "页面有效!"
             Else
                lblOutput.Text = "在页面中不能出现空项!"
             End If
        ‘判断是否输入为数字
            if not isnumeric(TextBox1.text) then
             lbloutput.text="请输入数值!"
            End if
        End Sub
    </script>
 </head>
<body>
<center><h3><font face="Verdana">验证表单的例子</font></h3></center>
<p>
<form runat="server">
<title>表单验证</title>
<center>
 <table bgcolor="white" cellpadding=10>
    <tr valign="top">
      <td colspan=3>
        <asp:Label ID="lblOutput" Text="请填写下面的内容" ForeColor="red"
Font-Name="Verdana" Font-Size="10" runat=server /><br>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Verdana size=2>储蓄卡类型:</font>
      </td>
      <td>
        <ASP:RadioButtonList id=RadioButtonList1 RepeatLayout="Flow" runat=server>
            <asp:ListItem>绿卡</asp:ListItem>
            <asp:ListItem>牡丹卡</asp:ListItem>
        </ASP:RadioButtonList>
      </td>
```

```
        <td align=middle rowspan=1>
          <asp:RequiredFieldValidator id="RequiredFieldValidator1"
                ControlToValidate="RadioButtonList1"
                Display="Static"
                InitialValue="" Width="100%" runat=server>
                *
          </asp:RequiredFieldValidator>
        </td>
      </tr>
      <tr>
        <td align=right>
          <font face=Verdana size=2>卡号:</font>
        </td>
        <td>
          <ASP:TextBox id=TextBox1 runat=server />
        </td>
        <td>
          <asp:RequiredFieldValidator id="RequiredFieldValidator2"
                ControlToValidate="TextBox1"
                Display="Static"
                Width="100%" runat=server>
                *
          </asp:RequiredFieldValidator>
        </td>
      </tr>
        <td>
      </tr>
      <tr>
        <td></td>
        <td>
          <ASP:Button id=Button1 text="验证" OnClick="ValidateBtn_Click" runat=server />
        </td>
        <td></td>
      </tr>
      </table>
</center>
</form>
</body>
</html>
```

我们对验证按钮的 OnClick 事件进行编程，其中用到了 IsNumeric()函数，用来判断变量是否为数值型的。我们还可以用 IsData()函数对输入的日期进行判断。IsData()接受的合法日期为 100 年 1 月 1 日到 9999 年 12 月 31 日。

运行如图：



当我们在卡号一栏中输入一些字母，而不是数值时，页面上将会提示你输入数值。

让我们再举一个很有用的验证应用：

当用户在填写个人信息的时候，往往需要输入身份证号，那么我们是如何进行身份证号的验证呢？

要解决这个问题，首先，让我们先看看我国的身份证号是如何编码的。

1　　2　　3　　4　5

XX XXXX XXXXXX XX　X （这个是没有升位以前的一个身份证号码的组成方式）

1 省　2 地市 3 生日 4 顺序码 5 性别

在这个例子中，我们只对省份进行判断。

身份编码一览表：

北京	11	吉林	22	福建	35	广东	44	云南	53
天津	12	黑龙江	23	江西	36	广西	45	西藏	54
河北	13	上海	31	山东	37	海南	46	陕西	61
山西	14	江苏	32	河南	41	重庆	50	甘肃	62
内蒙古	15	浙江	33	湖北	42	四川	51	青海	63
辽宁	21	安徽	34	湖南	43	贵州	52	宁夏	64
新疆	65	台湾	71	香港	81	澳门	82	国外	91

在这个程序中，仅仅作了一个简单的判断

Validate1.aspx 的文件内容如下：

```
<!--源文件：form\web 页面简介\validate1.aspx-->
<html>
<head>
    <script language="VB" runat="server">
        Sub ValidateBtn_Click(sender As Object, e As EventArgs)
```

```
            If (Page.IsValid) Then
                lblOutput.Text = "页面有效!"
            Else
                lblOutput.Text = "在页面中不能出现空项!"
            End If
            If   not isnumeric(TextBox1.text) then
                bloutput.text="请输入数值!"
            End if
            ‘在这里我们只作了一个简单的判断。使用了 left$（）函数
            if left$(textbox1.text,2)<>"11" then
                lbloutput.text="请验证你的身份证输入"
            End if
 End Sub
    </script>
</head>
<body>
<center><h3><font face="Verdana">验证表单的例子</font></h3></center>
<p>
<form runat="server">
<title>表单验证</title>
<center>
 <table bgcolor="white" cellpadding=10>
     <tr valign="top">
       <td colspan=3>
         <asp:Label  ID="lblOutput"  Text=" 请 填 写 下 面 的 内 容 "  ForeColor="red"
Font-Name="Verdana" Font-Size="10" runat=server /><br>
       </td>
     </tr>
     <tr>
       <td align=right>
         <font face=Verdana size=2>身份证号:</font>
       </td>
       <td>
         <ASP:TextBox id=TextBox1 runat=server />
       </td>
       <td>
         <asp:RequiredFieldValidator id="RequiredFieldValidator2"
             ControlToValidate="TextBox1"
             Display="Static"
             Width="100%" runat=server>
             *
         </asp:RequiredFieldValidator>
       </td>
     </tr>
```

```
        <td>
      </tr>
      <tr>
        <td></td>
        <td>
          <ASP:Button id=Button1 text="验证" OnClick="ValidateBtn_Click" runat=server />
        </td>
        <td></td>
      </tr>
      </table>
</center>
</form>
</body>
</html>
```

在这个程序中，我们仅对北京地区的身份证号进行了验证，我们使用 Left$()函数把字符串的前两个字符取出进行比较。如果大家感兴趣的话，可以把这个程序补充完整。

程序的运行如图：

这是输入正确的情况，如输入不正确，则显示（如图）：

我们在验证的时候，有时需要进行特殊的验证。在下面的表中，列出了需要进行特殊验证时要使用的特殊控件。

控件	描述
RequiredFieldValidator	使用户在输入时，不是使这一项为空
CompareValidator	对两个控件的值进行比较
RangeValidator	对输入的值进行控制，使其值界定在一定范围内
RegularExpressionValidator	把用户输入的字符和自定义的表达式进行比较
CustomValidator	自定义验证方式
ValidationSummary	在一个页面中显示总的验证错误

现在对各个验证控件介绍：

1. RequiredFieldValidator

下面的这个例子，演示了 **RequiredFieldValidator 控件**的使用方法。

validate3.aspx 文件：

```
<!--源文件：form\web 页面简介\validate3.aspx-->
<html>
<body>
<center>
<title>验证控件演示 (1)</title>
<h3><font face="Verdana">验证控件演示 (1)</font></h3>
<form runat=server>
    姓名: <asp:TextBox id=Text1 runat="server"/>
```

```
        <asp:RequiredFieldValidator id="RequiredFieldValidator1" ControlToValidate="Text1"
Font-Name="Arial" Font-Size="11" runat="server">
            此项不能为空!
        </asp:RequiredFieldValidator>
        <p>
        <asp:Button id="Button1" runat="server" Text="验证" />
    </form>
</center>
</body>
</html>
```

当我们不在文本框中输入内容的时候，页面上将会出现不能为空的提示。

程序运行如下：

2．CompareValidator 控件

为了比较两个控件的值，此时我们需要使用 **CompareValidator** 控件。

在下面的这个例子中，我们将讲解 **CompareValidator** 控件的用法。

先看文件 **validata4.aspx：**

```
<!--源文件：form\web 页面简介\validate4.aspx-->
<%@ Page clienttarget=downlevel %>
<html>
<title>CompareValidator 控件示例</title>
<head>
    <script language="VB" runat="server">
        Sub Button1_OnSubmit(sender As Object, e As EventArgs)
            If Page.IsValid Then
                lblOutput.Text = "比较正确!"
            Else
                lblOutput.Text = "比较不正确!"
            End If
```

```
        End Sub
        Sub lstOperator_SelectedIndexChanged(sender As Object, e As EventArgs)
            comp1.Operator = lstOperator.SelectedIndex
            comp1.Validate
        End Sub
    </script>
</head>
<body>
<center>
    <h3><font face="Verdana">CompareValidator 控件示例</font></h3>
    <form runat=server>
      <table bgcolor="#eeeeee" cellpadding=10>
      <tr valign="top">
        <td>
            <h5><font face="Verdana">字符串 1:</font></h5>
            <asp:TextBox Selected id="txtComp" runat="server"></asp:TextBox>
        </td>
        <td>
            <h5><font face="Verdana">比较运算符:</font></h5>
            <asp:ListBox                                                id="lstOperator"
OnSelectedIndexChanged="lstOperator_SelectedIndexChanged" runat="server">
                    <asp:ListItem Selected Value="Equal" >=</asp:ListItem>
                    <asp:ListItem Value="NotEqual" ><></asp:ListItem>
                    <asp:ListItem Value="GreaterThan" >></asp:ListItem>
                    <asp:ListItem Value="GreaterThanEqual" >>=</asp:ListItem>
                    <asp:ListItem Value="LessThan" ><</asp:ListItem>
                    <asp:ListItem Value="LessThanEqual" >=<</asp:ListItem>
            </asp:ListBox>
        </td>
        <td>
            <h5><font face="Verdana">字符串 2:</font></h5>
            <asp:TextBox id="txtCompTo" runat="server"></asp:TextBox><p>
            <asp:Button runat=server Text=" 验 证 " ID="Button1"
onclick="Button1_OnSubmit" />
        </td>
      </tr>
      </table>
      <asp:CompareValidator id="comp1" ControlToValidate="txtComp" ControlToCompare =
"txtCompTo" Type="String" runat="server"/>
      <br>
     <asp:Label ID="lblOutput" Font-Name="verdana" Font-Size="10pt" runat="server"/>
    </form>
</center>
</body>
```

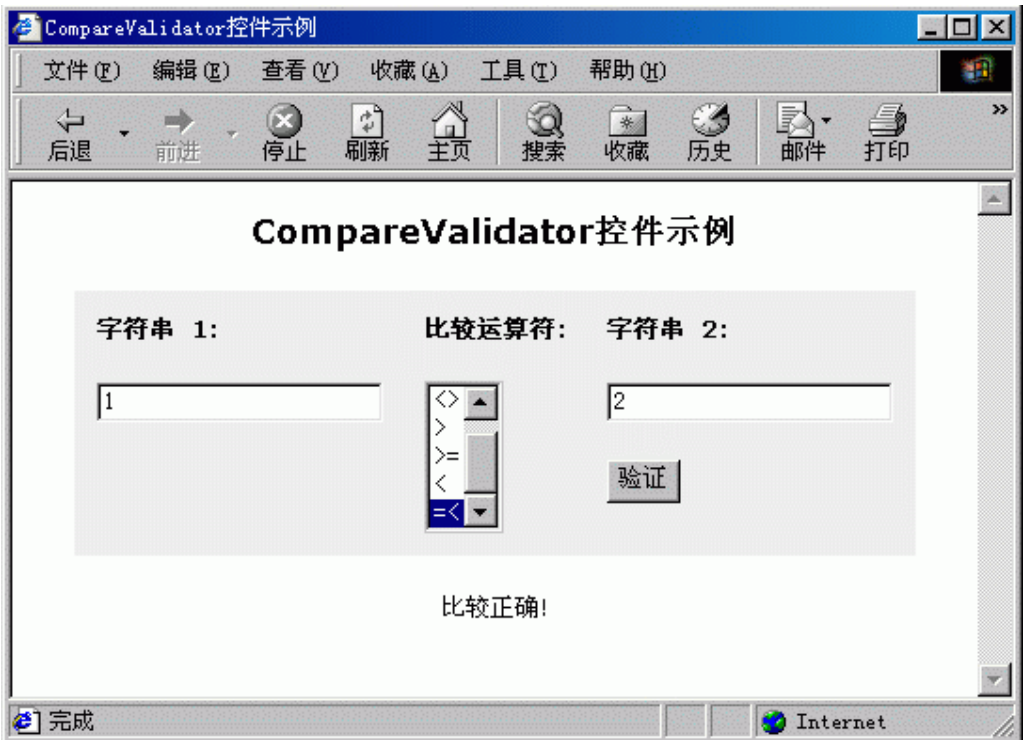
</html>

在上面的代码中，我们实现了对两个控件的值进行比较。

程序运行如下：

当我们在两个文本框中输入值，然后选定运算符后，点验证按钮后，在页面上将显示

比较结果:：



3．RangeValidator 控件

RangeValidator 控件主要界定输入的值的范围。因为有时我们要求输入的值是要有一定范围的，所以我们要使用 RangeValidator 来判断。

在下面的这个例子中，我们将来介绍 RangeValidator 控件。

请看 **validata5.aspx** 的程序内容：

```
<!--源文件：form\web 页面简介\validate5.aspx-->
<%@ Page clienttarget=downlevel %>
<html>
<center>
<title>RangeValidator 控件演示</title>
<head>
    <script language="VB" runat="server">
        Sub Button1_Click(sender As Object, e As EventArgs)
            If (Page.IsValid) Then
                lblOutput.Text = "结果正确!"
            Else
                lblOutput.Text = "结果不正确!"
```

```
            End If
        End Sub
        Sub lstOperator_SelectedIndexChanged(sender As Object, e As EventArgs)
            rangeVal.Type = lstType.SelectedIndex
            rangeVal.Validate
        End Sub
    </script>
</head>
<body>

    <h3><font face="Verdana">RangeValidator 控件演示</font></h3>
    <p>
    <form runat="server">
      <table bgcolor="#eeeeee" cellpadding=10>
      <tr valign="top">
        <td>
            <h5><font face="Verdana">输入要验证的值:</font></h5>
            <asp:TextBox Selected id="txtComp" runat="server"/>
        </td>
        <td>
            <h5><font face="Verdana">数据类型:</font></h5>
            <asp:DropDownList                                                    id="lstType"
OnSelectedIndexChanged="lstOperator_SelectedIndexChanged"    runat=server>
                <asp:ListItem Selected Value="String" >String</asp:ListItem>
                <asp:ListItem Value="Integer" >Integer</asp:ListItem>
            </asp:DropDownList>
        </td>
        <td>
            <h5><font face="Verdana">最小值:</font></h5>
            <asp:TextBox id="txtMin" runat="server" />
        </td>
        <td>
            <h5><font face="Verdana">最大值:</font></h5>
            <asp:TextBox id="txtMax" runat="server" /><p>
            <asp:Button Text="验证" ID="Button1" onclick="Button1_Click" runat="server"
/>
        </td>
      </tr>
    </table>
    <asp:RangeValidator      id="rangeVal"      Type="String"      ControlToValidate="txtComp"
MaximumControl="txtMax" MinimumControl="txtMin" runat="server"/>
    <br>
    <asp:Label id="lblOutput" Font-Name="verdana" Font-Size="10pt" runat="server" />
    </form>
```

```
</body>
</center>
</html>
```

当我们在三个文本框中分别输入要验证的值，最大值，和最小值，然后按下验证按钮，页面上将显示判断的结果。

在本例中我们只能比较 integer 和 string 的值，当然，我们也可以增加数据类型，如 double 型，float 型，date 型，currency 型等。

结果运行如下：

4. RegularExpressionValidator 控件

我们在制作网站的时候,尤其是各种电子商务网站,首先都会让用户填写一些表格来获取注册用户的各种信息,因为用户有可能输入各式各样的信息,而有些不符合要求的数据会给我们的后端 ASP 处理程序带来不必要的麻烦,甚至导致网站出现一些安全问题。因此我们在将这些信息保存到网站的数据库之前,要对这些用户所输入的信息进行数据的合法性校验,以便后面的程序可以安全顺利的执行。

使用 RegularExpressionValidator 服务器控件，可以用来检查我们输入的信息是否和我们的自定义的表达式一致。比方说用它可以检查 e-mail 地址，电话号码等合法性。

在讲述 RegularExpressionValidator 服务器控件使用之前，我们先来了解一下正则表达式（RegularExpression）的来源：

正则表达式的“祖先”可以一直上溯至对人类神经系统如何工作的早期研究。Warren McCulloch 和 Walter Pitts 这两位神经生理学家研究出一种数学方式来描述这些神经网络。1956 年，一位叫 Stephen Kleene 的美国数学家在 McCulloch 和 Pitts 早期工作的基础上，发表了一篇标题为“神经网事件的表示法”的论文，引入了正则表达式的概念。正则表达式就是用来描述他称为“正则集的代数”的表达式，因此采用“正则表达式”这个术语。随后，发现可以将这一工作应用于使用 Ken Thompson 的计算搜索算法的一些早期研究，Ken Thompson 是 Unix 的主要发明人。正则表达式的第一个实用应用程序就是 Unix 中的 qed 编辑器。如他们所说，剩下的就是众所周知的历史了。从那时起直至现在正则表达式都是基于文本的编辑器和搜索工具中的一个重要部分。

其实，正则表达式（RegularExpression）是一个正则表达式就是由普通字符（例如字符 a 到 z）以及特殊字符（称为元字符）组成的文字模式。该模式描述在查找文字主体时待匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

使用正则表达式，就可以：

1. 测试字符串的某个模式。例如，可以对一个输入字符串进行测试，看在该字符串是否存在一个电话号码模式或一个信用卡号码模式。这称为数据有效性验证。

2. 替换文本。可以在文档中使用一个正则表达式来标识特定文字，然后可以全部将其删除，或者替换为别的文字。

3. 根据模式匹配从字符串中提取一个子字符串。可以用来在文本或输入字段中查找特定文字。

例如，如果需要搜索整个 web 站点来删除某些过时的材料并替换某些 HTML 格式化标记，则可以使用正则表达式对每个文件进行测试，看在该文件中是否存在所要查找的材料或 HTML 格式化标记。用这个方法，就可以将受影响的文件范围缩小到包含要删除或更改的材料的那些文件。然后可以使用正则表达式来删除过时的材料，最后，可以再次使用正则表达式来查找并替换那些需要替换的标记。

另一个说明正则表达式非常有用的示例是一种其字符串处理能力还不为人所知的语言。VBScript 是 Visual Basic 的一个子集，具有丰富的字符串处理功能。与 C 类似的 Visual Basic Scripting Edition 则没有这一能力。正则表达式给 Visual Basic Scripting Edition 的字符串处理能力带来了明显改善。不过，可能还是在 VBScript 中使用正则表达式的效率更高，它允许在单个表达式中执行多个字符串操作。

正是由于“正则表达式”的强大功能，才使得微软慢慢将正则表达式对象移植到了视窗系统上面。在书写正则表达式的模式时使用了特殊的字符和序列。下表描述了可以使用的字符和序列，并给出了实例。

字符描述： \：将下一个字符标记为特殊字符或字面值。例如"n"与字符"n"匹配。"\n"与换行符匹配。序列"\\"与"\"匹配，"\("与"("匹配。

^ ：匹配输入的开始位置。

$ ：匹配输入的结尾。

* ：匹配前一个字符零次或几次。例如，"zo*"可以匹配"z"、"zoo"。

+ ：匹配前一个字符一次或多次。例如，"zo+"可以匹配"zoo",但不匹配"z"。

? ：匹配前一个字符零次或一次。例如，"a?ve?"可以匹配"never"中的"ve"。

.：匹配换行符以外的任何字符。

(pattern) 与模式匹配并记住匹配。匹配的子字符串可以从作为结果的 Matches 集合中使用 Item [0]...[n]取得。如果要匹配括号字符(和)，可使用"\(" 或 "\)"。

x|y：匹配 x 或 y。例如 "z|food" 可匹配 "z" 或 "food"。"(z|f)ood" 匹配 "zoo" 或 "food"。

{n}：n 为非负的整数。匹配恰好 n 次。例如，"o{2}" 不能与 "Bob 中的 "o" 匹配，但是可以与"foooood"中的前两个 o 匹配。

{n,} ：n 为非负的整数。匹配至少 n 次。例如，"o{2,}"不匹配"Bob"中的"o"，但是匹配"foooood"中所有的 o。"o{1,}"等价于"o+"。"o{0,}"等价于"o*"。

{n,m} ：m 和 n 为非负的整数。匹配至少 n 次，至多 m 次。例如，"o{1,3}" 匹配"fooooood"中前三个 o。"o{0,1}"等价于"o?"。

[xyz] ：一个字符集。与括号中字符的其中之一匹配。例如，"[abc]" 匹配"plain"中的"a"。

[^xyz] ：一个否定的字符集。匹配不在此括号中的任何字符。例如，"[^abc]" 可以匹配

"plain"中的"p".

[a-z] ：表示某个范围内的字符。与指定区间内的任何字符匹配。例如，"[a-z]"匹配"a"与"z"之间的任何一个小写字母字符。

[^m-z] ：否定的字符区间。与不在指定区间内的字符匹配。例如，"[m-z]"与不在"m"到"z"之间的任何字符匹配。

\b ：与单词的边界匹配，即单词与空格之间的位置。例如，"er\b" 与"never"中的"er"匹配，但是不匹配"verb"中的"er"。

\B ：与非单词边界匹配。"ea*r\B"与"never early"中的"ear"匹配。

\d ：与一个数字字符匹配。等价于[0-9]。

\D ：与非数字的字符匹配。等价于[^0-9]。

\f ：与分页符匹配。

\n ：与换行符字符匹配。

\r ：与回车字符匹配。

\s ：与任何白字符匹配，包括空格、制表符、分页符等。等价于"[\f\n\r\t\v]"。

\S ：与任何非空白的字符匹配。等价于"[^ \f\n\r\t\v]"。

\t ：与制表符匹配。

\v ：与垂直制表符匹配。

\w ：与任何单词字符匹配，包括下划线。等价于"[A-Za-z0-9_]"。

\W ：与任何非单词字符匹配。等价于"[^A-Za-z0-9_]"。

\num ：匹配 num 个，其中 num 为一个正整数。引用回到记住的匹配。例如，"(.)\1"匹配两个连续的相同的字符。

\n：匹配 n，其中 n 是一个八进制换码值。八进制换码值必须是 1, 2 或 3 个数字长。

例如，"\11" 和 "\011" 都与一个制表符匹配。"\0011"等价于"\001" 与 "1"。八进制换码值不得超过 256。否则，只有前两个字符被视为表达式的一部分。允许在正则表达式中使用 ASCII 码。

\xn：匹配 n，其中 n 是一个十六进制的换码值。十六进制换码值必须恰好为两个数字长。例如，"\x41"匹配"A"。"\x041"等价于"\x04" 和 "1"。允许在正则表达式中使用 ASCII 码。

RegularExpressionValidator 有两种主要的属性来进行有效性验证。ControlToValidate 包含了一个值进行验证。如取出文本框中的值。如 ControlToValidate="TextBox1" ValidationExpression 包含了一个正则表达式进行验证。

好了，有了上面的叙述，我们就举个例子来说明正则表达式。比如，我们想要对用户输入的电子邮件进行校验，那么，什么样的数据才算是一个合法的电子邮件呢？我可以这样输入：test@yesky.com，当然我也会这样输入：xxx@yyy.com.cn,但是这样的输入就是非法的：xxx@@com.cn 或者@xxx.com.cn，等等，所以我们得出一个合法的电子邮件地址至少应当满足以下几个条件：

1. 必须包含一个并且只有一个符号“@”
2. 第一个字符不得是“@”或者“.”
3. 不允许出现“@.”或者.@
4. 结尾不得是字符“@”或者“.”

所以根据以上的原则和上面表中的语法，我们很容易的就可以得到需要的模板如下："=^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)*\.[A-Za-z0-9]+$"

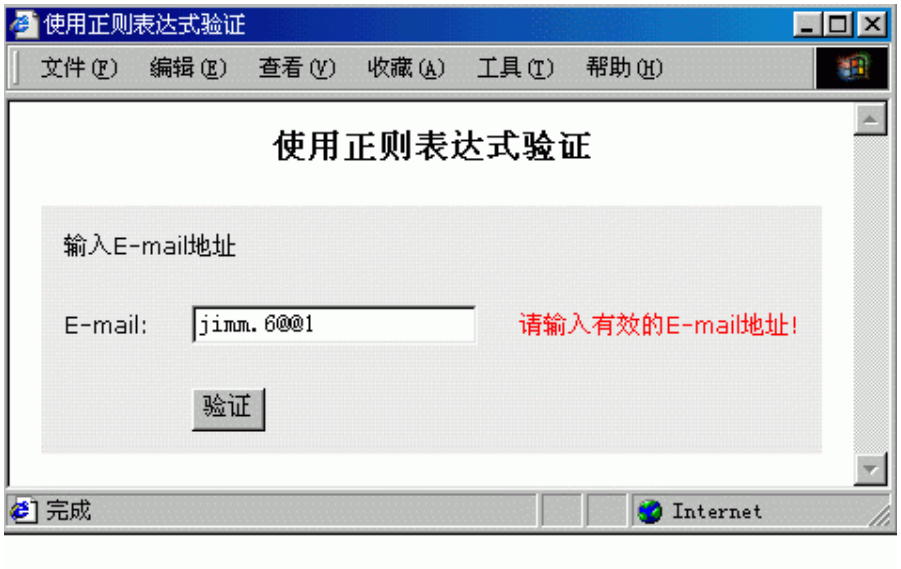
请看 **validata6.aspx** 的内容：

```
<!--源文件：form\web 页面简介\validate6.aspx-->
```

```
</head>
<body>
<center><h3><font face="Verdana">使用正则表达式验证</font></h3></center>
<p>
<form runat="server">
<center>
<title>使用正则表达式验证</title>
    <table bgcolor="#eeeeee" cellpadding=10>
    <tr valign="top">
      <td colspan=3>
        <asp:Label ID="lblOutput" Text="输入 E-mail 地址" Font-Name="Verdana"
Font-Size="10pt" runat="server"/>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Verdana size=2>E-mail:</font>
      </td>
      <td>
        <ASP:TextBox id=TextBox1 runat=server />
      </td>
      <td>
        <asp:RegularExpressionValidator id="RegularExpressionValidator1" runat="server"
            ControlToValidate="TextBox1"

ValidationExpression="^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-)[A-Za-z0-9]+)*\.[A-Za-z0-9]+
$"
            Display="Static"
            Font-Name="verdana"
            Font-Size="10pt">
              请输入有效的 E-mail 地址!
        </asp:RegularExpressionValidator>
      </td>
    </tr>
    <tr>
      <td></td>
      <td>
        <ASP:Button text="验证" OnClick="ValidateBtn_Click" runat=server />
      </td>
      <td></td>
    </tr>
    </table>
</center>
</form>
```

```
</body>
</html>
```

这样，我们只要定制不同的模板，就可以实现对不同数据的合法性校验了。所以，正则表达式对象中最重要的属性就是：“Pattern”属性,只要真正掌握了这个属性，才可以自由的运用正则表达式对象来为我们的数据校验进行服务。
程序的运行效果如图：



通过上面的介绍，我们对数据验证的方法有了一定的认识。在下面的内容中，我们还将通过更具体的实例，来对数据的有效性验证进行讲解。

2.1.3.3.3 Page_Unload

这个阶段页面已经处理完毕，需要做些清理工作。一般地，你可以在这个阶段关闭打开的文件和数据库链路，或者释放对象。
1、断开数据库连接
请看如下脚本：

```
<script language="VB" runat="server">
‘定义一个共有变量
    public Dim MyConnection As SQLConnection
 ‘定义 Page_Load 事件
     Sub Page_Load(Src As Object, E As EventArgs)
          Dim DS As DataSet

          Dim MyCommand As SQLDataSetCommand
            MyConnection = New SQLConnection("server='iceberg';uid=sa;pwd=;database=info")
MyCommand = New SQLDataSetCommand("select * from infor",MyConnection)
               Myconnection.open()
               DS = New DataSet()
               MyCommand.FillDataSet(ds, "infor")
```

```
            MyDataGrid.DataSource=ds.Tables("infor").DefaultView
            MyDataGrid.DataBind()
 End Sub
’定义 Page_UnLoad 事件
 Sub Page_UnLoad(Src As Object, E As EventArgs)
’与数据库断开连接
    MyConnection.Close()
 End Sub
```

现在我们在来看一个对文件操作的例子。

在这个例子中，我们使用的了两个事件，Page_Load 事件和 Page_Unload 事件。在 Page_Load 事件先创建一个文件，然后向这个文件中写入内容。在 Page_Unload 事件中我们将此文件关闭。

代码如下：

```
<%@ import namespace="system.io" %>
<html>
<head>
<title>ASP.NET 测试 写 文本文件</title>
</head>
<body>
<script language="vb" runat="server">
public   Dim writeFile As StreamWriter
Sub Page_Load(Sender As Object,E as EventArgs)
writeFile = File.CreateText( "c:\test.txt" )
writeFile.WriteLine( "这是一个测试文件!" )
writeFile.WriteLine( "使用了 Page_Load 事件和 Page_Unload 事件!" )
Response.Write( "test.txt 创建 并 写入 成功!" )
End Sub
Sub Page_UnLoad(Sender AS Object, E as EventArgs)
writeFile.Close
End Sub
</script>
</body>
</html>
```

这样，我们就使用了 Page_Load 事件和 Page_Unload 事件。很明显，我们定义 Page_Load 事件，是因为这个阶段页面已经处理完毕，需要做些清理工作。

上面我们分析了页面处理最重要的几个阶段。应该说明的是：页面的处理过程远比上面的复杂，因为每个控件都需要初始化。在后面的章节中，我们还将了解到更加详细的页面处理过程。

2.1.4 Web Form 事件模型

在 asp.net 中，事件是一个非常重要的概念。我们举两个例子来说明在 Web Form 中的应用。

2.1.4.1 例子一：多按钮事件

我们在一个<form></form>里面有几个按钮，多个事件的响应我们该怎么处理呢？在 asp.net 中有很好的处理机制，我们可以在一个页面中写几个方法来分别响应不同的事件。

在下面的例子中，将根据五个按钮的功能，我们定义了五个方法：AddBtn_Click(Sender As Object, E As EventArgs)、AddAllBtn_Click(Sender As Object, E As EventArgs)、RemoveBtn_Click(Sender As Object, E As EventArgs)、RemoveAllBtn_Click(Sender As Object, E As EventArgs)、result(Sender As Object,E As EventArgs)，分别用来处理全部加进、单个加进、单个取消、全部取消和提交事件。我们的 form 提交的时候，还是提交给本页面，由本页面进行处理，代码如下：

<form action="menent.aspx" runat=server>

其中，menent.aspx 就是本页面。

Menent.aspx 文件代码如下：

```
<!--源文件：form\web 页面简介\menent.aspx-->
<html>

    <script language="VB" runat="server">

        Sub AddBtn_Click(Sender As Object, E As EventArgs)

            If Not (AvailableFonts.SelectedIndex = -1)
                InstalledFonts.Items.Add(New ListItem(AvailableFonts.SelectedItem.Value))
                AvailableFonts.Items.Remove(AvailableFonts.SelectedItem.Value)
            End If
        End Sub

        Sub AddAllBtn_Click(Sender As Object, E As EventArgs)

            Do While Not (AvailableFonts.Items.Count = 0)
                InstalledFonts.Items.Add(New ListItem(AvailableFonts.Items(0).Value))
                AvailableFonts.Items.Remove(AvailableFonts.Items(0).Value)
            Loop
        End Sub

        Sub RemoveBtn_Click(Sender As Object, E As EventArgs)

            If Not (InstalledFonts.SelectedIndex = -1)
                AvailableFonts.Items.Add(New ListItem(InstalledFonts.SelectedItem.Value))
                InstalledFonts.Items.Remove(InstalledFonts.SelectedItem.Value)
            End If
        End Sub
```

```
    Sub RemoveAllBtn_Click(Sender As Object, E As EventArgs)

        Do While Not (InstalledFonts.Items.Count = 0)
            AvailableFonts.Items.Add(New ListItem(InstalledFonts.Items(0).Value))
            InstalledFonts.Items.Remove(InstalledFonts.Items(0).Value)
        Loop
    End Sub

Sub result(Sender As Object,E As EventArgs)

    dim tmpStr as String

        tmpStr="<br>"
    Do While Not (InstalledFonts.Items.Count = 0)
        tmpStr=tmpStr & InstalledFonts.items(0).value & "<br>"
        InstalledFonts.items.remove(InstalledFonts.items(0).value)
        Loop

    tmpStr=System.Web.HttpUtility.UrlEncodeToString(tmpStr,System.Text.Encoding.UTF
            8)
    Page.Navigate("result.aspx?InstalledFonts=" & tmpStr)

End Sub

</script>

<body bgcolor="#ccccff">
<center>
    <h3><font face="Verdana">.NET->不同事件的处理方法！</font></h3>
</center>
<center>
    <form action="menent.aspx" runat=server>

        <table>
            <tr>
                <td>
                    现有字体
                </td>
                <td>
                    <!-- Filler -->
                </td>
                <td>
                    选择的字体
                </td>
```

```
            </tr>
            <tr>
              <td>
                <asp:listbox id="AvailableFonts" width="100px" runat=server>
                  <asp:listitem>Roman</asp:listitem>
                  <asp:listitem>Arial Black</asp:listitem>
                  <asp:listitem>Garamond</asp:listitem>
                  <asp:listitem>Somona</asp:listitem>
                  <asp:listitem>Symbol</asp:listitem>
                </asp:listbox>
              </td>
              <td>
                <!-- Filler -->
              </td>
              <td>
                <asp:listbox id="InstalledFonts" width="100px" runat=server>
                  <asp:listitem>Times</asp:listitem>
                  <asp:listitem>Helvetica</asp:listitem>
                  <asp:listitem>Arial</asp:listitem>
                </asp:listbox>
              </td>
            </tr>
            <tr>
              <td>
                <!-- Filler -->
              </td>
              <td>
                <asp:button text="<<==" OnClick="RemoveAllBtn_Click" runat=server/>
                <asp:button text="<--" OnClick="RemoveBtn_Click" runat=server/>
                <asp:button text="-->" OnClick="AddBtn_Click" runat=server/>
                <asp:button text="==>>" OnClick="AddAllBtn_Click" runat=server/>
    <asp:label id="Message" forecolor="red" font-bold="true" runat=server/>
              </td>
              </tr>
    <tr align=center>
    <td   align=center>
                <asp:button text="提交" Onclick="result" runat=server/>
    <!-- Filler -->
              </td>
            </tr>
          </table>

      </form>
  </center>
```

```
    </body>

</html>
```

写一个页面，在提交时候接收相关信息。我们在页面进入的时候取得传送过来的数值，用：

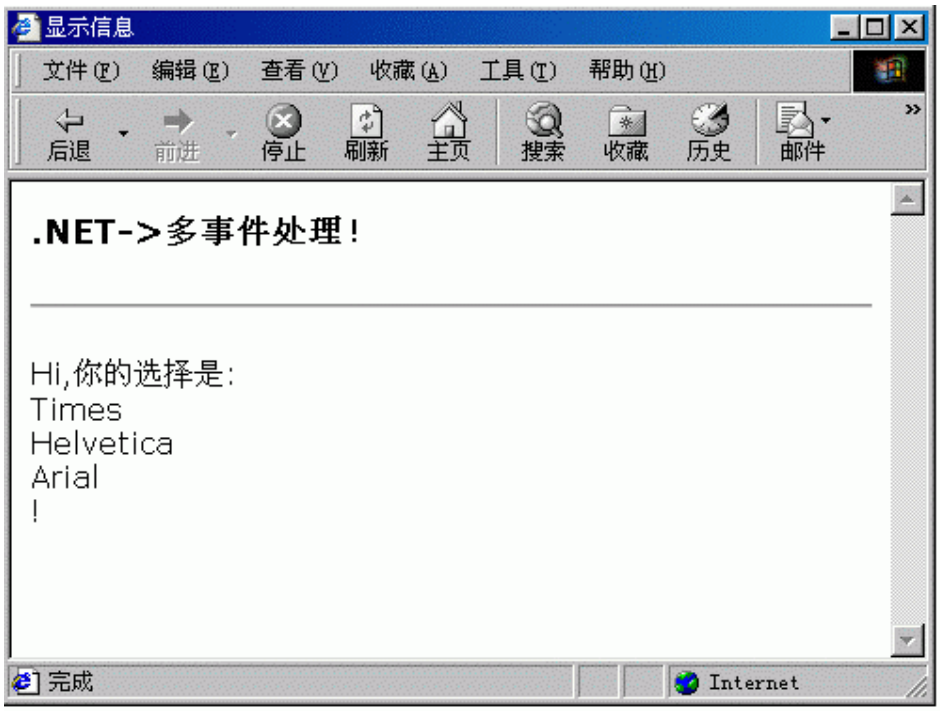
```
    Request.Params("InstalledFonts")
```

来获得，具体来看我们的文件 **result.aspx** 的代码：

```
<!--源文件：form\web 页面简介\result.aspx-->
    <html>
    <script language="VB" runat="server">
        Sub Page_Load(Sender As Object, E As EventArgs)
            If Not (Page.IsPostBack)
                NameLabel.Text = Request.Params("InstalledFonts")
            End If
        End Sub
    </script>

    <BODY >
        <h3><font face="Verdana">.NET->多事件处理！</font></h3>
        <p>
        <p>
        <hr>
        <form action="controls_NavigationTarget.aspx" runat=server>
            <font face="Verdana">
                Hi,你的选择是: <asp:label id="NameLabel" runat=server/>!
            </font>
        </form>
    </body>
</html>
```

程序运行如下：

当我们点击提交按钮的时候，将显示：



2.1.4.2 例子二：AutoPostBack

PostBack 属性在 Page_Load 事件中出现的，在一个用户请求结束后，如果页面重新 Load，则返回一个 true。这对初始化一个页面来讲是一件非常容易的事情，下面看我们的代码：

```
Sub Page_Load(Sender as Object， e as EventArgs)
   If   IsPostBack and ( TextBox2.Text = “”)
        TextBox2.Text=”Hello” & TextBox1.Text & “!!  你好啊！”
   End If
End Sub
```

如果 IsPostBack 返回一个真值并且 TextBox2.Text 为空，程序执行它下面的语句。在另外一个方面，我们设置一个标识：

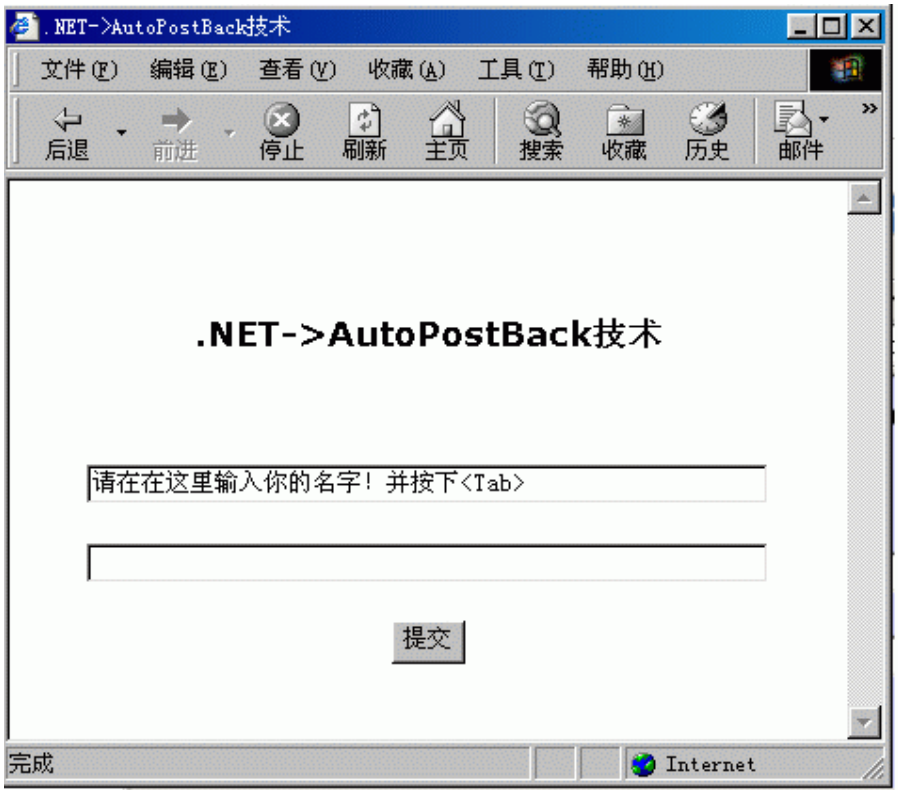
```
<asp:TextBox id="TextBox1" Text="请在在这里输入你的名字！并按下<Tab>"
      AutoPostBack="True" Columns=50 runat="server"/>
```

我们设定 AutoPostBack="True"，自动 PostBack，下面是我们的完整的代码(autopostback..aspx)：

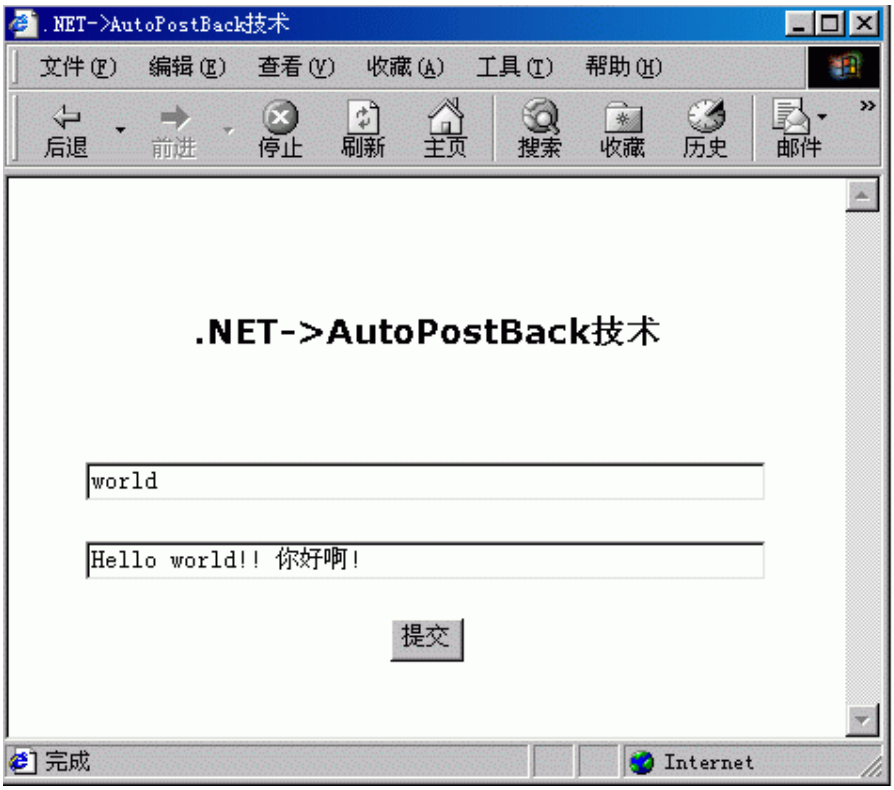
```
<!--源文件：form\web 页面简介\autopostback.aspx-->
```

```
<html>
<head>
  <script language="VB" runat="server">

        Sub Page_Load(Sender as Object，e as EventArgs)
        If   IsPostBack and ( TextBox2.Text = “”)
             TextBox2.Text=”Hello” & TextBox1.Text & “!!  你好啊！”
        End If
      End Sub

  </script>
</head>
<body>
  <center>
    <br><br><br>
      <h3><font face="Verdana">.NET->AutoPostBack 技术</font></h3>
   <br><br>
</center>
<center>
  <form runat="server">
    <p>
      <asp:TextBox id="TextBox1" Text="请在在这里输入你的名字！并按下<Tab>"
            AutoPostBack="True" Columns=50 runat="server"/>
    <p>
      <asp:TextBox id="TextBox2" Columns=50 runat="server"/>
  <p>
       <asp:Button Text="提交" Runat="server"/>
  <p>
  </form>
</center>
</body>
</html>
```

访问如下：



输入完成后，按下<Tab>键，得到如下结果：



2.1.5 小结

在这一章中，我们对 Web Form 页面进行了介绍，通过几个实例，我们分别介绍了 Server 控件，HTML Server 控件，以及 Web Form 的事件模型。在下面的章节中，我们将对本章所涉及的概念进行更深入的讲解。

第二章 服务器端控件

2.2.1 服务器端控件示例

在讲述服务器端控件的时候，我们先来讲述一个具体的例子。

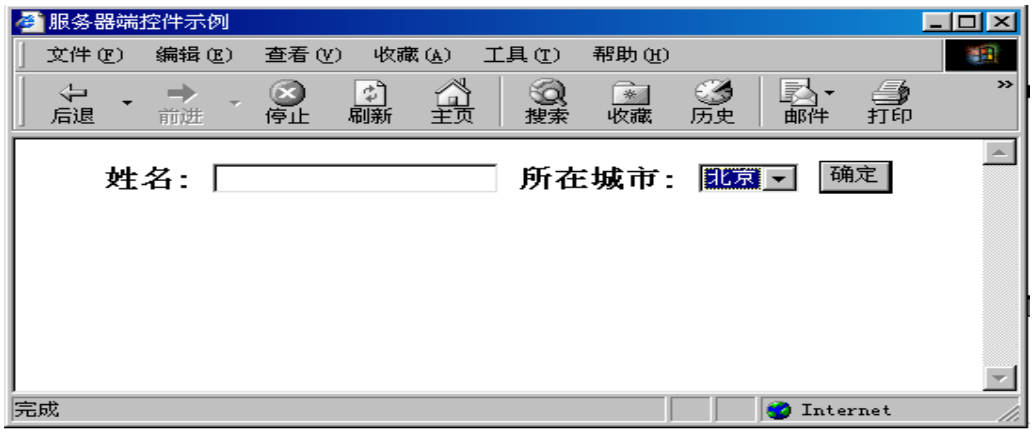
我们说过，在 asp.net 里面，一切都是对象。我们也谈到：WEB 页面本身就是一个对象。或者说，WEB 页面就是一个对象的容器。那么，这个容器可以装些什么东西呢？这一节我们学习服务器端控件，英文是 Server Control。这是 WEB 页面能够容纳的对象之一。

什么是 Control？熟悉 VB 的读者肯定再清楚不过了。简单地说，Control 就是一个可重用的组件或者对象，这个组件不但有自己的外观，还有自己的数据和方法，大部分组件还可以响应事件。通过微软的集成开发环境(Visual Studio.NET 7.0)，你可以简单地把一个 Control

拖放到一个 Form 中。

那为什么叫“Server Control”？这是因为这些 Control 是在服务器端存在的。服务器端控件也有自己的外观，在客户端浏览器中，Server Control 的外观由 HTML 代码来表现。Server Control 会在初始化时，根据客户的浏览器版本，自动生成适合浏览器的 HTML 代码。以前我们做网页或者做 ASP 程序时候，必须考虑到浏览器的不同版本对 HTML 的支持有所不同，比如 Netscape 和 IE 对 DHTML 的支持就有所不同。当时，解决浏览器版本兼容问题的最有效方法，就是在不同版本的浏览器中作测试。现在，由于 Server Control 自动适应不同的浏览器版本，也就是自动兼容不同版本的浏览器，程序员的工作量减轻了许多。下面，我们来看看如何在 WEB FORM 中嵌入 Server Control。我们的例子是从上一节继承来的：

如图：



下面是实现如图效果的代码：(**sample.aspx**)

```
<!--源文件：form\ServerControl\sample.aspx-->
<html>
    <script language="VB" runat=server>
        Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
            Message.Text = "Hi " & Name.Text & ", 你选择的城市是：" & city.SelectedItem.Text
        End Sub
    </script>
    <body>
        <center>
        <form action="form2.aspx" method="post" runat="server">
            <h3>姓名: <asp:textbox id="Name" runat="server"/>

            所在城市:  <asp:dropdownlist id="city" runat=server>
                        <asp:listitem>北京</asp:listitem>
                        <asp:listitem>上海</asp:listitem>
                        <asp:listitem>重庆</asp:listitem>
                      </asp:dropdownlist>
```

```
            <asp:button type=submit text=" 确 定 " OnClick="SubmitBtn_Click"
runat="server"/>
            <p>
            <asp:label id="Message" runat="server"/>
        </form>
        </center>
    </body>
</html>
```

请注意：上面的代码中我们使用了三种 Server Control，分别是：

1、 asp:textbox

2、asp:dropdownlist

3、asp:label

我们注意到三个控件都有相同的 RunAt 属性：**RunAt=”Sevrer”**。所有的服务器端控件都有这样的属性。这个属性标志了一个控件是在 Server 端进行处理的。

我们看下面的代码：

```
<script language="VB" runat=server>
        Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
                Message.Text = "Hi " & Name.Text & ", 你选择的城市是: " &
    city.SelectedItem.Text
        End Sub
    </script>
```

用过 VB 的朋友是不是觉得很熟悉？没错，这是用 VB 写的一个事件处理函数，void SubmitBtn_Click(Object sender , EventArgs e) ，你可能一看就明白了，void 代表该函数没有返回值，该函数带有两个参数，可是这里的 Sender 的意义是什么意思呢？它的用处又到底是什么呢？其实很简单，这个 Sender 就是这个事件的触发者。这里，Sender 就是被 Click 的 button。其中代码只有一行， 你可能注意到这行代码中的 Message、Name、city 你并没有定义，那么它们从哪里来的呢？

看下面的代码：：

```
<form action="form2.aspx" method="post" runat="server">
    <h3> Name: <asp:textbox id="Name" runat="server"/>
            Category:   <asp:dropdownlist id="city" runat=server>
                            <asp:listitem>北京</asp:listitem>
                            <asp:listitem>上海</asp:listitem>
                            <asp:listitem>重庆</asp:listitem>
                        </asp:dropdownlist>
<asp:button type=submit text="确定" OnClick="SubmitBtn_Click" runat="server"/>
<p>
            <asp:label id="Message" runat="server"/>
        </form>
```

我们发现每个服务端的控件都带有一个 ID 号。而我们在 VB.NET 代码中所引用的就是这些 ID。我们可以认为 ID 就是控件的名称。在 ASP 中我们也使用过 ID 吧。那时候，ID

属性和 Name 属性并没有什么不同：

<input id=email name=email >

在客户端，我们通过 VBSCript 代码或者 Jscript 代码，可以这样访问 Form 表单的 Input 域：

```
<SCRIPT LANGUAGE=javascript>
<!--
    document.all("email")="darkman@yesky.com";
//-->
</SCRIPT>
```

从上面的代码可以看出，在 DHTML 中，我们也是通过 ID 来访问 Form 表单的输入域的。在 ASPX 中，情况有些类似之处。差别在于：一个在客户端，一个在服务器端。

如果你和第一节的例子代码对比一下，你会发现：这个表单的写法和 html 表单完全不同了吧？首先，所有的表单项包括表单本身后面都加上了 runat=server，这句话的意思就是说这个是服务器端控制项，另外象传统表单的什么<input type=text>等的写法都变了，你仔细观察一下可以看出，原来的文本框变为<asp:textbox>，选择框变为<asp:dropdownlist>，选择框选项变为<asp:listitem>，而 submit 按钮变为<asp:button>，这个按钮对应的控制函数就刚才我提到的那个 SubmitBtn_Click 函数，它是运行在服务器端的。另外还有一个服务器端控制<asp:label id=”Message” runat=”server”/>，这个 asp:label 是传统表单所没有的，它是一个服务器端文本控制，那么就存在一个问题，如果传统的 HTML 里没有这个元素，那么 ASP.NET 是怎么接收的呢？你运行一下这个程序，然后看一下 HTML 源码，你会发现这么一行：

<input type="hidden" name="__VIEWSTATE value="…" />对，ASP.NET 就是通过这个隐藏表单的形式传递过去的。

所以，一个客户端控件，加上 runat=Server 就变成服务器端控件，服务器端控件能在服务器端脚本中被自由运用。在以后的章节中，我们还要对常用的服务器端控件进行详细介绍。

2.2.2 文本输入控件

文本输入控件目的是让用户输入文本，文本模式是一个单行的输入框，但是用户可以根据自己的需要把它改成密码输入模式或者多行输入模式。

在此我们用单行文本输入模式和密码模式来说明，在 asp.net 中，输入一个文本，可用下面的语句来表示：

```
<!--输入邮件地址-->
 <asp:TextBox id=email width=200px maxlength=60 runat=server />
```

第一句为注释，我们可以设定输入框的宽度和文本的长度，runat=server 为表示运行于服务器端，下面我们来看看输入控件的校验，一个简单的语句就可以实现我们在普通的 html 页面上的复杂的 JavaScript、VBScript 或者其他代码的验证。首先我们用户必须输入邮件地址：

```
<!--验证邮件的有效性！->不能为空-->
 <asp:RequiredFieldValidator id="emailReqVal"
      ControlToValidate="email"
```

```
        ErrorMessage="Email. "
        Display="Dynamic"
        Font-Name="Verdana" Font-Size="12"
        runat=server>
        *
    </asp:RequiredFieldValidator>
```

ControlToValidate="email"属性为针对 TextBox id=email 的文本框，Display 属性我们定义为“Dynamic”，即为当鼠标光标所在位置发生变化时属性根据条件变化。如果为空，则打印出“ * “字符出来。

在通常情况下，邮件地址总会包含一些特定的字符，我们在验证的时候，就可以要求用户输入的内容中包含我们规定的字符。

```
    <!--验证邮件的有效性！->必须包含有效字符-->
    <asp:RegularExpressionValidator id="emailRegexVal"
        ControlToValidate="email"
        Display="Static"
        ValidationExpression="^[\w-]+@[\w-]+\.(com|net|org|edu|mil)$"
        Font-Name="Arial" Font-Size="11"
        runat=server>
        不是有效邮件地址
    </asp:RegularExpressionValidator>
```

ControlToValidate="email"语句跟上面一样，
ValidationExpression="^[\w-]+@[\w-]+\.(com|net|org|edu|mil)$"表示我们在邮件里要包含的内容，如果没有包含，则打印出“不是有效邮件地址“这个提示。

对密码也是同样的道理的，主要的差别是，对于密码，通常我们要确认一次，校验两次输入的密码是否相等。下面是我们的代码：

```
    <!--输入确认密码->两个密码是否相等-->
    <asp:CompareValidator id="CompareValidator1"
        ControlToValidate="passwd2" ControlToCompare="passwd"
        Display="Static"
        Font-Name="Arial" Font-Size="11"
        runat=server>
        密码不相等
    </asp:CompareValidator>
```

ControlToValidate="passwd2" ControlToCompare="passwd"此语句即为两个密码之间的比较，不相等，打印出“密码不相等“的提示。

下面是我们的完整的代码**(textbox.aspx)**：

```
<!--源文件：form\ServerControl\textbox.aspx-->
<html>
<body>
```

```
<br><br><br>
 <center>
    <h3><font face="Verdana">.NET->文本控件</font></h3>
 </center>
    <form method=post runat=server>
    <hr width=600 size=1 noshade>
<br><br>
    <center>
    <!--标题-->
    <asp:ValidationSummary ID="valSum" runat="server"
          HeaderText="按照下面的要求填写:"
          DisplayMode="SingleParagraph"
          Font-Name="verdana"
          Font-Size="12"
          />
    <p>
    <table border=0 width=600>
    <tr>
      <td align=right>
        <font face=Arial size=2>电子邮件:</font>
      </td>
      <td>
    <!--输入邮件地址-->
     <asp:TextBox id=email width=200px maxlength=60 runat=server />
      </td>
      <td>
    <!--验证邮件的有效性！->不能为空-->
          <asp:RequiredFieldValidator id="emailReqVal"
                ControlToValidate="email"
                ErrorMessage="Email. "
                Display="Dynamic"
                Font-Name="Verdana" Font-Size="12"
                runat=server>
                *
          </asp:RequiredFieldValidator>
    <!--验证邮件的有效性！->必须包含有效字符-->
          <asp:RegularExpressionValidator id="emailRegexVal"
                ControlToValidate="email"
                Display="Static"
                ValidationExpression="^[\w-]+@[\w-]+\.(com|net|org|edu|mil)$"
                Font-Name="Arial" Font-Size="11"
                runat=server>
                不是有效邮件地址
          </asp:RegularExpressionValidator>
```

```
    </td>
  </tr>

  <tr>
    <td align=right>
      <font face=Arial size=2>密码:</font>
    </td>
    <td>
  <!--输入密码-->
      <asp:TextBox id=passwd TextMode="Password" maxlength=20 runat=server/>
    </td>
    <td>
  <!--输入密码->密码不能为空-->
        <asp:RequiredFieldValidator id="passwdReqVal"
            ControlToValidate="passwd"
            ErrorMessage="Password. "
            Display="Dynamic"
            Font-Name="Verdana" Font-Size="12"
            runat=server>
            *
        </asp:RequiredFieldValidator>
  <!--输入密码->包含其中有效字符-->
        <asp:RegularExpressionValidator id="passwdRegexBal"
            ControlToValidate="passwd"
            ValidationExpression=".*[!@#$%^&*+;:].*"
            Display="Static"
            Font-Name="Arial" Font-Size="11"
            Width="100%" runat=server>
            密码必须包含 (!@#$%^&amp;*+;:)
        </asp:RegularExpressionValidator>
    </td>
  </tr>
  <tr>
    <td align=right>
      <font face=Arial size=2>再次输入密码:</font>
    </td>
    <td>
  <!--输入确认密码->-->
      <asp:TextBox id=passwd2 TextMode="Password" maxlength=20 runat=server/>
    </td>
    <td>
  <!--输入确认密码->不能为空-->
      <asp:RequiredFieldValidator id="passwd2ReqVal"
          ControlToValidate="passwd2"
```

```
                ErrorMessage="Re-enter Password. "
                Display="Dynamic"
                Font-Name="Verdana" Font-Size="12"
                runat=server>
                *
            </asp:RequiredFieldValidator>
        <!--输入确认密码->两个密码是否相等-->
            <asp:CompareValidator id="CompareValidator1"
                ControlToValidate="passwd2" ControlToCompare="passwd"
                Display="Static"
                Font-Name="Arial" Font-Size="11"
                runat=server>
                密码不相等
            </asp:CompareValidator>
          </td>
        </tr>
    </table>
      <p>
          <input runat="server" type=submit value="提交">
        <p>
        <hr width=600 size=1 noshade>

    </form>
  </center>
</body>
</html>
```

运行结果如下：

我们不按照要求的输入，开到了下面的提示：

多行文本输入控件一般用来输入相关的内容，比如用户简短介绍、相关信息的补充等，一般情况下可以不用限制用户的输入。当然有些时候像留言板，我们不希望用户的输入内容中包含 HTML 的相关标记，这个时候我们就可以用上面的同样的方法来限制用户的输入，用法都是一样的，在此我们就不打算举个例子来说明了。

2.2.3 按钮控件

按钮控件的目的是使用户对页面的内容作出判断，当按下按钮后，页面会对用户的选择作出一定的反应，达到与用户交互的目的。

按钮控件的使用虽然很简单，但是按钮控件却是最常用的服务器控件之一，值得我们学习。对按钮控件的使用要注意它的 3 个事件和一个属性，即:

OnClick 事件，即用户按下按钮以后，即将触发的事件。我们通常在编程中，利用此事件，完成对用户选择的确认、对用户表单的提交、对用户输入数据的修改等等。

OnMouseOver 事件，当用户的光标进入按钮范围触发的事件。为了使页面有更生动的显示，我们可以利用此事件完成，诸如，当光标移入按钮范围时，使按钮发生某种显示上的改变，用以提示用户可以进行选择了。

OnMouseOut 事件，当用户光标脱离按钮范围触发的事件。同样，为使页面生动，当光标脱离按钮范围时，也可以发生某种改变，如恢复原状，用以提示用户脱离了按钮选择范围，若此时按下鼠标，将不是对按钮的操作。

Text 属性，按钮上显示的文字，用以提示用户进行何种选择。

例子：下例将显示 3 个按钮，分别演示 3 种事件的处理。

当按下第一个按钮时，根据 <asp:button id="btn1" text="OnClick 事件演示" Width=150px Onclick="btn1_Onclick" runat=server />的定义将调用 btn1_OnClick 过程，该过程的作用，即在按钮后显示 lbl1 控件的内容”OnClick 事件触发”

当移动光标到第二个按钮时，根据按钮定义 <asp:button id="btn2" text="OnMouseOver 事件演示" Width=150px OnMouseOver="this.style.backgroundColor='lightgreen'" OnMouseOut="this.style.backgroundColor='buttonface'" runat=server />，光标移动到按钮上时，按钮的背景色应该变为淡绿色。

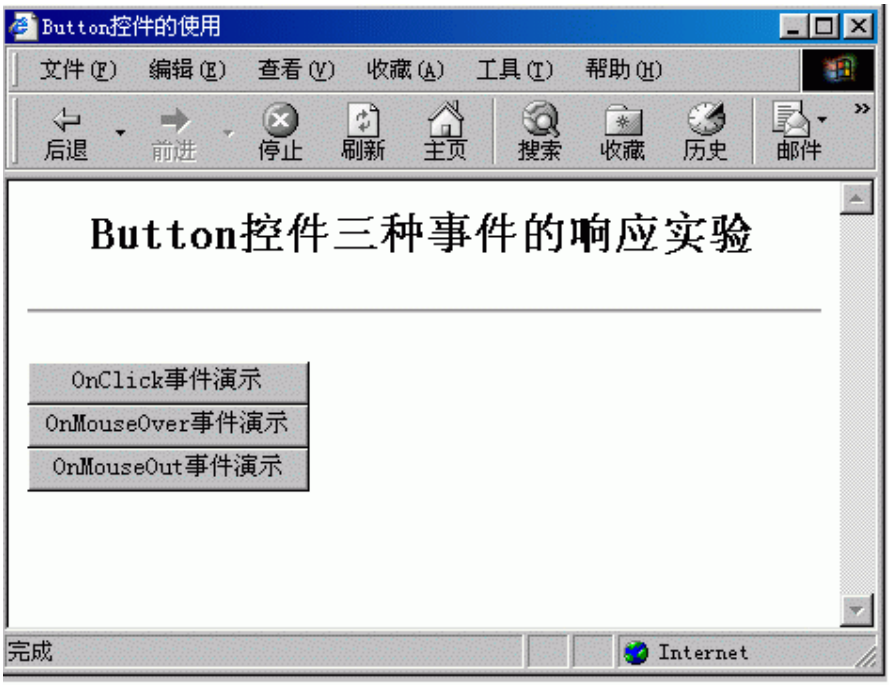
当移动光标到第三个按钮时，根据其定义<asp:button id="btn3" text="OnMouseOut 事件演示" Width=150pxOnMouseOver="this.style.fontWeight='bold'” OnMouseOut = "this.style.fontWeight = 'normal'" runat=server />，按钮的字体变为黑体，但是我们要观察的是当又把光标移开后，第三个按钮是否恢复正常的字体。

1． 源程序(**FormButton.aspx**)

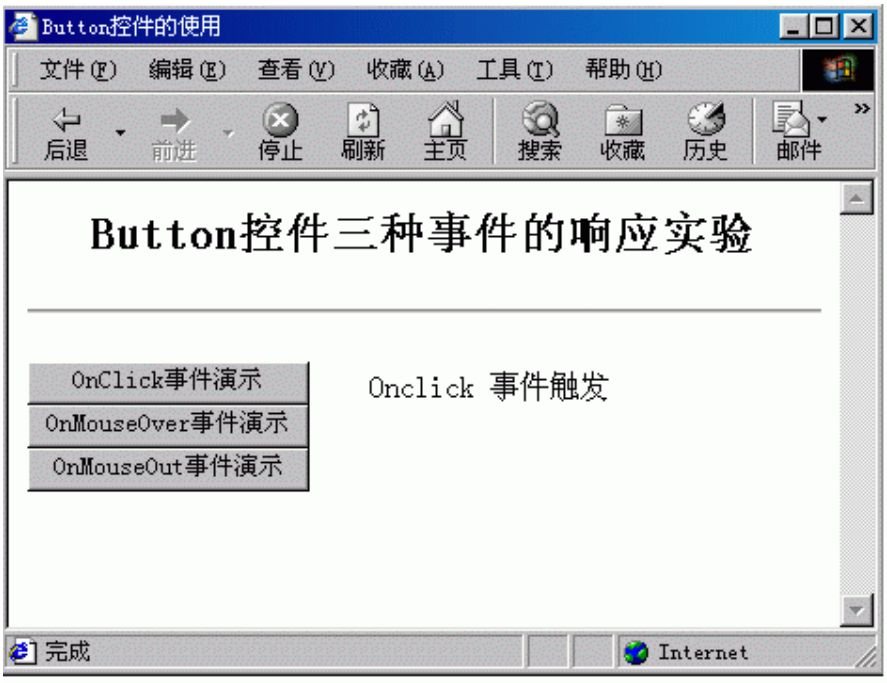
```
<!--源文件：form\ServerControl\formbutton.aspx-->
<!-- 文件名：FormButton.aspx -->
<html>
<script language="vb" runat=server>
sub btn1_OnClick(s as object,e as EventArgs)
lbl1.text="Onclick 事件触发"
end sub
```

```
</script>

 <head>
 <title>
  Button 控件的使用
 </title>
 </head>

<body bgcolor=#ccccff>
 <center>
  <h2>Button 控件三种事件的响应实验</h2>
  <hr>
 </center>
     <form runat=server>
          <asp:button id="btn1" text="OnClick 事件演示" Width=150px
     Onclick="btn1_Onclick" runat=server />
             
          <asp:label id="lbl1" runat=server/>
          <br>

          <asp:button id="btn2" text="OnMouseOver 事件演示" Width=150px
     OnMouseOver="this.style.backgroundColor='lightgreen'"
          OnMouseOut="this.style.backgroundColor='buttonface'"
          runat=server />

          <br>

          <asp:button id="btn3" text="OnMouseOut 事件演示" Width=150px
     OnMouseOver="this.style.fontWeight='bold'"
     OnMouseOut="this.style.fontWeight='normal'"
      runat=server />

     </form>

<body>
```

2. 开始时的输出画面如下：

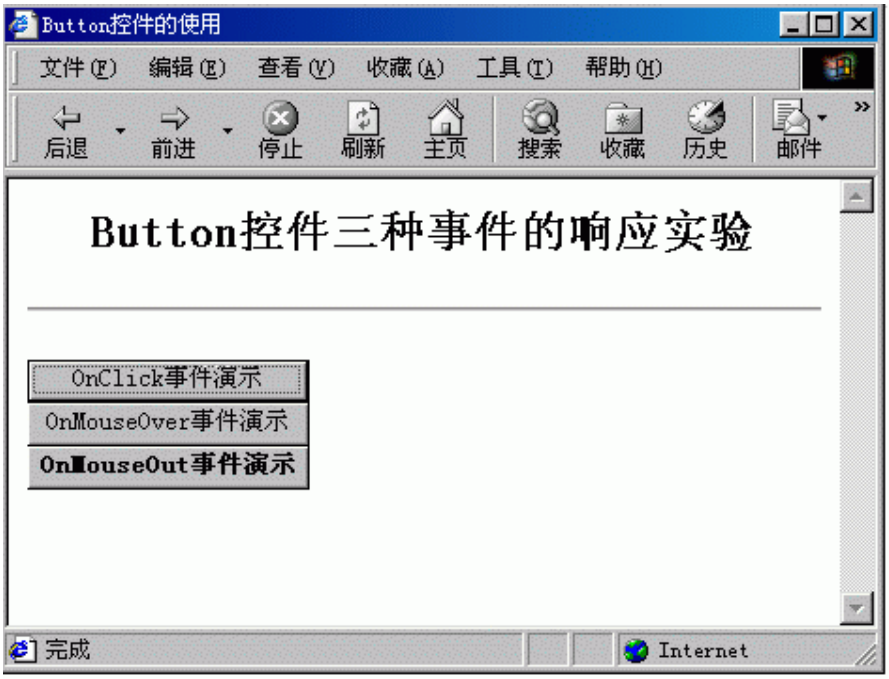


3．当按下“OnClick 事件演示”按钮后，lbl1 显示“OnClick 事件触发”：



4．当移动光标到“OnMouseOver 事件演示“按钮时，该按钮背景色变为淡绿色：

5．当光标移动到“OnMouseOut 事件演示“按钮时，该按钮的字体变为黑体，但是我们需观察的却是再移开光标后，字体是否恢复正常，结论是会的，这里只给出了移动到该按钮时的画面，移开后的画面由于和开始画面一样，就不演示了。



2.2.4 复选控件

在日常信息输入中，我们会遇到这样的情况，输入的信息只有两种可能性（例如：性别、婚否之类）,如果采用文本输入的话，一者输入繁琐，二者无法对输入信息的有效性进行控制，这时如果采用复选控件（CheckBox），就会大大减轻数据输入人员的负担，同时输入数据的规范性得到了保证。

CheckBox 的使用比较简单，主要使用 id 属性和 text 属性。Id 属性指定对复选控件实例的命名，Text 属性主要用于描述选择的条件。另外当复选控件被选择以后，通常根据其 Checked 属性是否为真来判断用户选择与否。

例如：使用复选控件

```
…
<asp:CheckBox id=”chkbox1” text=”中国人” runat=server/>
 …
 判断条件满足否：
…
If chkbox1.Checked=True
    LblTxt.text=”是中国人”
Else
     LblTxt.text=”不是中国人”
End If
…
```

2.2.5 单选控件

使用单选控件的情况跟使用复选控件的条件差不多，区别的地方在于，单选控件的选择可能性不一定是两种，只要是有限种可能性，并且只能从中选择一种结果，在原则上都可以用单选控件（RadioButton）来实现。

单选控件主要的属性跟复选控件也类似，也有 id 属性、text 属性，同样也依靠 Checked 属性来判断是否选中，但是与多个复选控件之间互不相关不同，多个单选控件之间存在着联系，要么是同一选择中的条件，要么不是。所以单选控件多了一个 GroupName 属性，它用来指明多个单选控件是否是同一条件下的选择项，GroupName 相同的多个单选控件之间只能有一个被选中。

例如：对单选控件的使用

```
…
    年龄选择：
    <asp:RadioButton id=”rbtn11” text=”20 岁以下” GroupName=”group1” runat=server /><br>
    <asp:RadioButton id=”rbtn12” text=”20-30 岁” GroupName=”group1” runat=server /><br>
    <asp:RadioButton id=”rbtn13” text=”30-40 岁” GroupName=”group1” runat=server /><br>
    <asp:RadioButton id=”rbtn14” text=”40 岁以上” GroupName=”group1” runat=server /><br>
    工作收入：
    <asp:RadioButton id=”rbtn21” text=”1000 元以下” GroupName=”group2” runat=server
/><br>
```

<asp:RadioButton id=”rbtn22” text=”1000-2000 元 ” GroupName=”group2” runat=server/>

<asp:RadioButton id=”rbtn23” text=”2000 元以上” GroupName=”group2” runat=server />

…

对选择条件的判断：

```
…
If   rbtn11.Checked = True
     LblResult1.text=”20 岁以下”
Else if rbtn12.Checked = True
     LblResult1.text=”20-30 岁”
Else if rbtn13.Checked = True
     LblResult1.text=”30-40 岁”
Else
     LblResult1.text=”40 岁以上”
End If

If rbtn21.Checked = True
     LblResult2.text=”1000 元以下”
Else if rbtn22.Checked = True
     LblResult2.text=”1000-2000 元”
Else
     LblResult.text=”2000 元以上”
End If
…
```

2.2.6 列表框

列表框（ListBox）是在一个文本框内提供多个选项供用户选择的控件，它比较类似于下拉列表，但是没有显示结果的文本框。到以后，我们会知道列表框实际上很少使用，大部分时候，我们都使用列表控件 DropDownList 来代替 ListBox 加文本框的情况，在这里对列表框的讨论，主要是为以后的应用学习一些简单的控件属性。

列表框的属性 SelectionMode，选择方式主要是决定控件是否允许多项选择。当其值为 ListSelectionMode.Single 时，表明只允许用户从列表框中选择一个选项；当值为 List.SelectionMode.Multi 时，用户可以用 Ctrl 键或者是 Shift 键结合鼠标，从列表框中选择多个选项。

属性 DataSource，说明数据的来源，可以为数组、列表、数据表。

方法 DataBind，把来自数据源的数据载入列表框的 items 集合。

例子：在这里我们将结合前面学习的按钮控件（Button）、复选控件（CheckBox）、单选控件（RadioButton）以及列表框（ListBox）给出一个实例。

首先，页面加载时，我们产生一个数组 Values，并添加 4 个关于水果的测试数据到 Valuse 数组。然后列表框从数组取得数据加载进自己的 items 集合。然后根据复选控件 chkBold 的状态决定是否用黑体字输出列表框的选择结果。根据单选控件 rbtnMulti 和 rbtnSingle 的状态

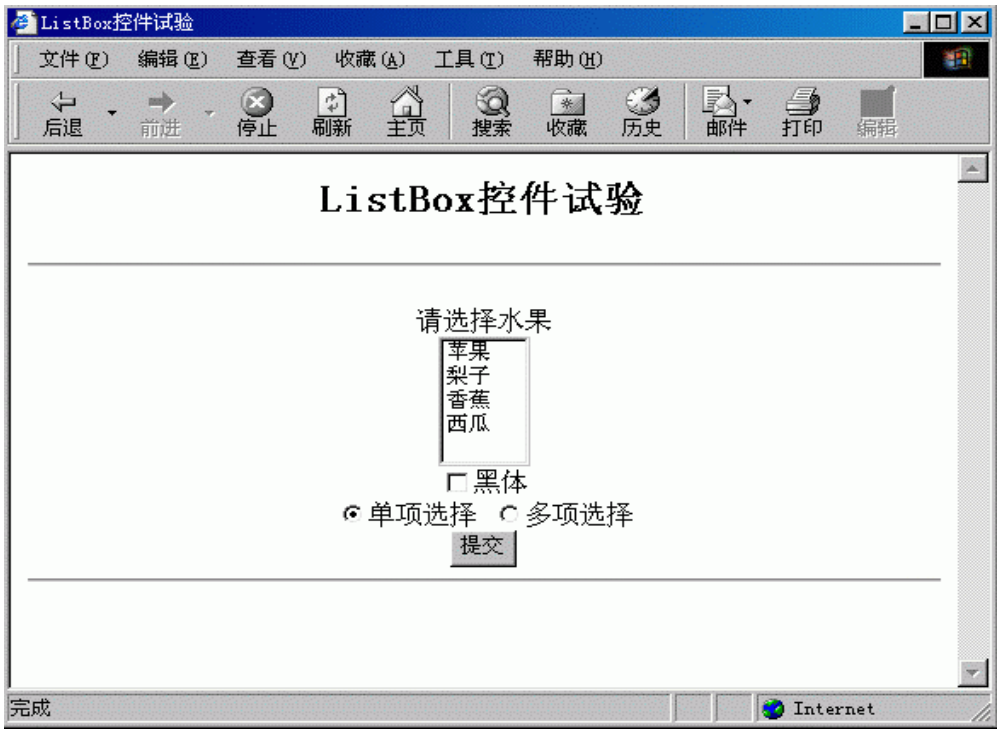
决定下一次列表框是否允许多选，最后按下按钮控件提交表单，显示选择的结果。

1．源程序(FormListBox.aspx)

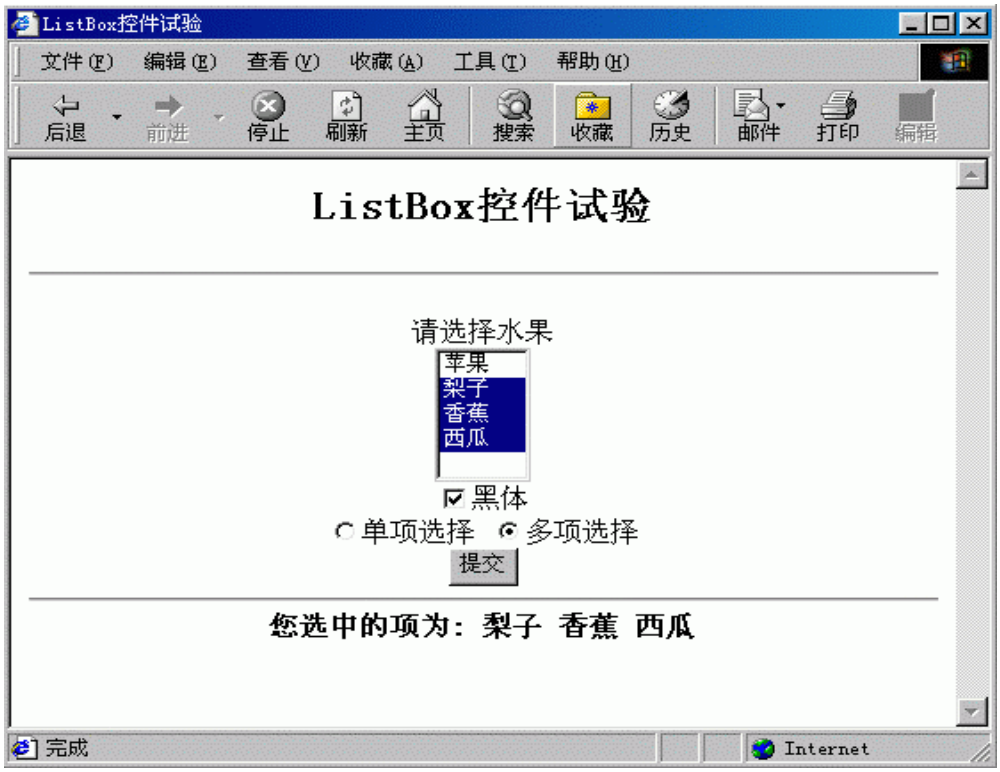
```
<!--源文件：form\ServerControl\formlistbox.aspx-->
<html>
 <head>
 <title>
 ListBox 控件试验
 </title>
 </head>

  <script language="VB" runat=server>
  Sub Page_Load()

  '如果选中黑体复选控件，把文本标签的字体设为黑体
  If chkBold.Checked
        lblTxt.font.bold=True
  Else
        lblTxt.font.bold=False
  End If

  '如果选中多选的单选控件，那么则把列表框设为允许多选
  If   rbtnMulti.Checked
    list1.SelectionMode=ListSelectionMode.Multiple
  Else
    list1.SelectionMode=ListSelectionMode.Single
  End If

  If Not IsPostBack
  '第一次请求时，为列表框设置数据
  Dim values as ArrayList=new ArrayList()

        values.add("苹果")
        values.add("梨子")
        values.add("香蕉")
        values.add("西瓜")
        list1.datasource=values
        list1.databind
  Else
  '把从列表框选中的内容赋予文本标识，如果未选择，显示"未选择"
  Dim i as int32
  Dim tmpStr as String

  ‘对列表框 list1 的 items 集合轮询，根据其 Selected 属性，判断是否被选中
```

```
   For i=0 to list1.items.count-1
   If list1.items(i).selected
   tmpStr=tmpStr & " " & list1.items(i).text
   End If
   Next

   If tmpStr is Nothing
   tmpStr="未选择"
   End If

   lblTxt.text="您选中的项为: " & tmpStr

   End If

End Sub
   </script>

    <body >
      <center>
       <h2>ListBox 控件试验</h2>
       <hr>
       <form method="POST" runat=server>
          请选择水果
          <br>
        <asp:ListBox id="list1" runat=server/>
          <br>
          <asp:CheckBox id="chkBold" text="黑体" runat=server />
          <br>
          ‘第一次设置为单项选择
          <asp:RadioButton id="rbtnSingle" Checked=True text="单项选择"
                                          groupname="group1" runat=server />
          <asp:RadioButton id="rbtnMulti" text="多项选择"
                                        groupname="group1" runat=server />
          <br>
     <asp:button text="提交" runat=server />
          <hr>
     <asp:label id="lblTxt" runat=server />
        </form>
      </center>
    </body>

</html>
```

2．开始时的画面输出，第一次缺省设置为单项选择



3．选择以黑体字输出，并且允许多项选择后的画面：



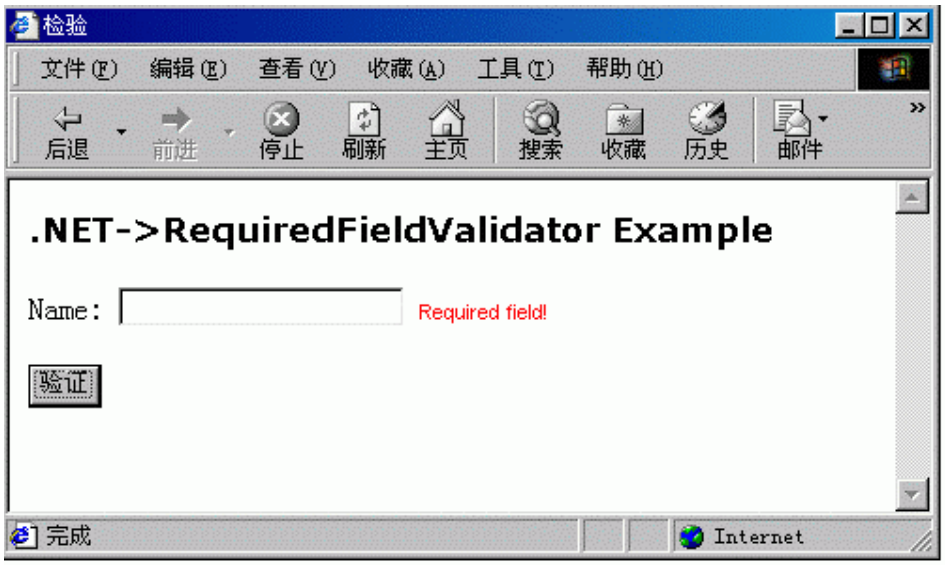
2.2.7 RequiredFieldValidator

这个 RequiredFieldValidator 服务器控件保证用户不会跳过一个入口，如果用户输入的值跟符合 RequiredFieldValidator 的要求，这个值就是有效的；否则，不会跳过这一输入步骤而往下走。

下面的例子(RequiredFieldValidator.aspx)验证了这个要求：

```
<!--源文件：form\ServerControl\requiredfieldvalidator.aspx-->
<html>
<title>检验</title>
    <h3><font face="Verdana">.NET->RequiredFieldValidator Example</font></h3>
    <form runat=server>
        Name: <asp:TextBox id=Text1 runat="server"/>
        <asp:RequiredFieldValidator id="RequiredFieldValidator1" ControlToValidate="Text1"

Font-Name="Arial" Font-Size="11" runat="server">
            Required field!
        </asp:RequiredFieldValidator>
        <p>
        <asp:Button id="Button1" runat="server" Text="验证" />
    </form>
</body>
</html>
```

运行效果如下：



2.2.8 ValidationSummary

用户的输入有时候是按照一定的顺序来的，有效性控件验证用户的输入并设置一个属性来线使用户的输入是否通过了验证。当所用得验证项都被处理之后，页面的 IsValid 属性就被设置，当有其中的一个验证没有通过时，整个页面将会不被通过验证。

当页面的 IsValid 属性为 false 时，ValidationSummary 属性将会表现出来。他获得页面上的每个确认控件并对每个错误提出修改信息。

文件 **Summary.aspx** 的内容：

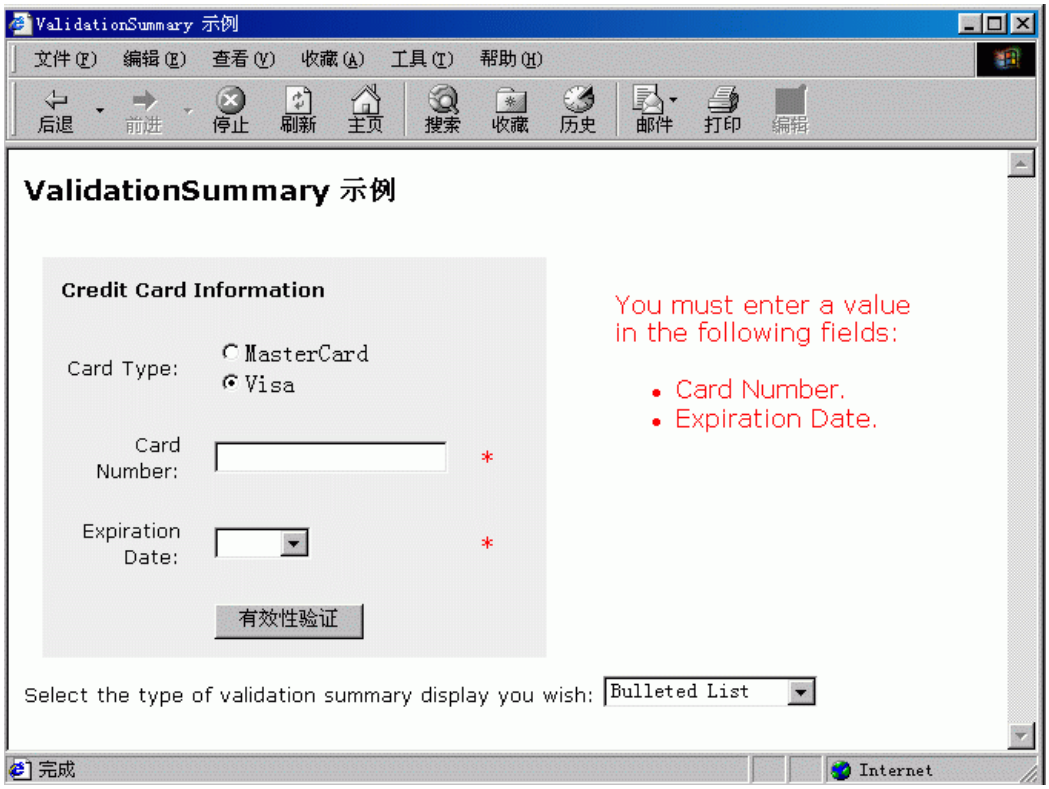
```
<!--源文件：form\ServerControl\summary.aspx-->
<%@ Page clienttarget=downlevel %>
<html>
<head>
    <script language="VB" runat="server">

        Sub ListFormat_SelectedIndexChanged(sender As Object, e As EventArgs)

            ' Change display mode of the validator summary when a new option
            ' is selected from the "ListFormat" dropdownlist
            valSum.DisplayMode = ListFormat.SelectedIndex

        End Sub

    </script>

</head>
<BODY>
<h3><font face="Verdana">ValidationSummary Sample</font></h3>
<p>

<form runat="server">
<table cellpadding=10>
    <tr>
        <td>
            <table bgcolor="#eeeeee" cellpadding=10>
            <tr>
              <td colspan=3>
              <font face=Verdana size=2><b>Credit Card Information</b></font>
              </td>
            </tr>
            <tr>
              <td align=right>
                <font face=Verdana size=2>Card Type:</font>
              </td>
```

```
    <td>
      <ASP:RadioButtonList id=RadioButtonList1 RepeatLayout="Flow"
            runat=server>
        <asp:ListItem>MasterCard</asp:ListItem>
        <asp:ListItem>Visa</asp:ListItem>
      </ASP:RadioButtonList>
    </td>
    <td align=middle rowspan=1>
      <asp:RequiredFieldValidator id="RequiredFieldValidator1"
        ControlToValidate="RadioButtonList1"
        ErrorMessage="Card Type. "
        Display="Static"
        InitialValue="" Width="100%" runat=server>
        *
      </asp:RequiredFieldValidator>
    </td>
  </tr>
  <tr>
    <td align=right>
      <font face=Verdana size=2>Card Number:</font>
    </td>
    <td>
      <ASP:TextBox id=TextBox1 runat=server />
    </td>
    <td>
      <asp:RequiredFieldValidator id="RequiredFieldValidator2"
        ControlToValidate="TextBox1"
        ErrorMessage="Card Number. "
        Display="Static"
        Width="100%" runat=server>
        *
      </asp:RequiredFieldValidator>

    </td>
  </tr>
  <tr>
    <td align=right>
      <font face=Verdana size=2>Expiration Date:</font>
    </td>
    <td>
      <ASP:DropDownList id=DropDownList1 runat=server>
        <asp:ListItem></asp:ListItem>
        <asp:ListItem >06/00</asp:ListItem>
        <asp:ListItem >07/00</asp:ListItem>
```

```
                    <asp:ListItem >08/00</asp:ListItem>
                    <asp:ListItem >09/00</asp:ListItem>
                    <asp:ListItem >10/00</asp:ListItem>
                    <asp:ListItem >11/00</asp:ListItem>
                    <asp:ListItem >01/01</asp:ListItem>
                    <asp:ListItem >02/01</asp:ListItem>
                    <asp:ListItem >03/01</asp:ListItem>
                    <asp:ListItem >04/01</asp:ListItem>
                    <asp:ListItem >05/01</asp:ListItem>
                    <asp:ListItem >06/01</asp:ListItem>
                    <asp:ListItem >07/01</asp:ListItem>
                    <asp:ListItem >08/01</asp:ListItem>
                    <asp:ListItem >09/01</asp:ListItem>
                    <asp:ListItem >10/01</asp:ListItem>
                    <asp:ListItem >11/01</asp:ListItem>
                    <asp:ListItem >12/01</asp:ListItem>
                </ASP:DropDownList>
            </td>
            <td>
                <asp:RequiredFieldValidator id="RequiredFieldValidator3"
                    ControlToValidate="DropDownList1"
                    ErrorMessage="Expiration Date. "
                    Display="Static"
                    InitialValue=""
                    Width="100%"
                    runat=server>
                    *
                </asp:RequiredFieldValidator>
            </td>
            <td>
        </tr>
        <tr>
            <td></td>
            <td>
                <ASP:Button id=Button1 text="有效性验证" runat=server />
            </td>
            <td></td>
        </tr>
        </table>
    </td>
    <td valign=top>
        <table cellpadding=20><tr><td>
        <asp:ValidationSummary ID="valSum" runat="server"
            HeaderText="You must enter a value in the following fields:"
```

```
                    Font-Name="verdana"
                    Font-Size="12"
                    />
                </td></tr></table>
            </td>
        </tr>
</table>

<font face="verdana" size="-1">Select the type of validation summary display you wish: </font>

<asp:DropDownList id="ListFormat" AutoPostBack=true

OnSelectedIndexChanged="ListFormat_SelectedIndexChanged" runat=server >
        <asp:ListItem>List</asp:ListItem>
        <asp:ListItem selected>Bulleted List</asp:ListItem>
        <asp:ListItem>Single Paragraph</asp:ListItem>
</asp:DropDownList>

</form>

</body>
</html>
```

结果如下：



2.2.9 使用 panel 控件

我们在进行用户信息录入的时候，如果使用 ASP 程序写的话，通常需要三个网页：

① 进行用户身份检查

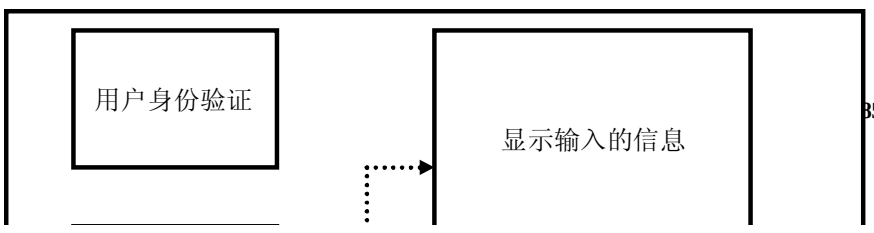
② 填写相关的内容

③ 显示你填写的内容

填写的流程如下：

这样的话，我们将分别设计 3 个网页。这样会显得很麻烦。可喜的是，我们可以使用 asp.net 中的 panel 控件，在一个页面中即可实现上述的功能。

流程如下：



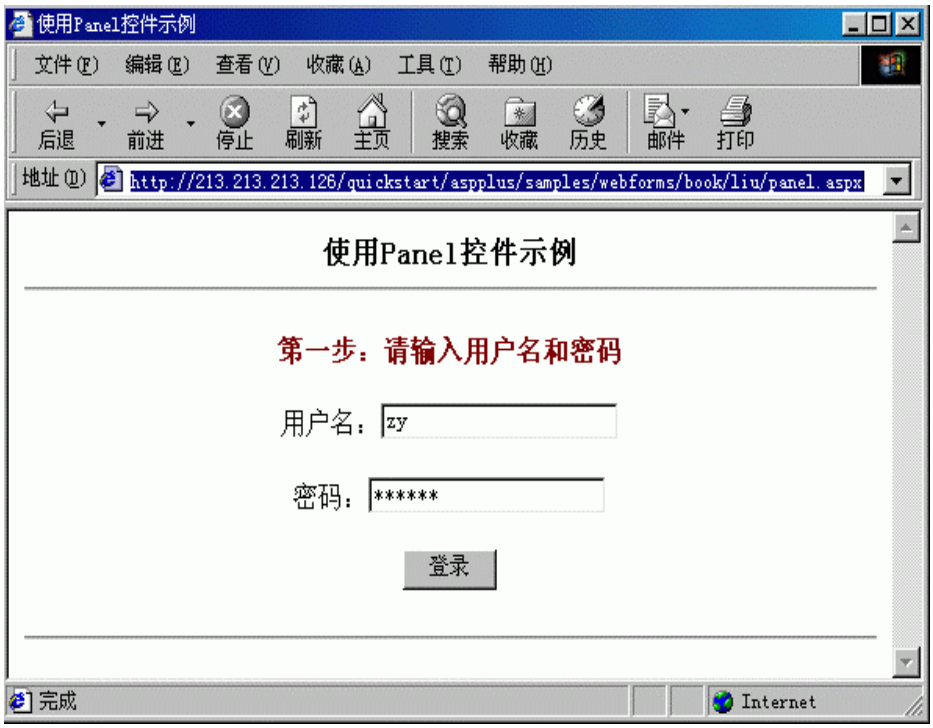
好了，有了上面的叙述，请看 **panel.aspx** 文件内容：

```
<!--源文件：form\ServerControl\panel.aspx-->
 <Html>
 <Body bgcolor="White">
 <center><H3>使用 Panel 控件示例<Hr></H3></center>
<title>使用 Panel 控件示例</title>
<script Language="VB" runat="server">
 Sub Page_Load(sender As Object, e As EventArgs)
     If Not Page.IsPostBack Then
         panel2.Visible = False
         panel3.Visible = False
     End If
 End Sub
Sub Button1_Click(sender As Object, e As EventArgs)
     panel1.Visible = False
     panel2.Visible = True
 End Sub

 Sub Button2_Click(sender As Object, e As EventArgs)
     panel2.Visible = False
     panel3.Visible = True
     Span1.InnerHtml    = "用户名: " & UserID.Text & "<BR>"
     Span1.InnerHtml &= "密码: " & Password.Text & "<BR>"
     Span1.InnerHtml &= "姓名: " & Name.Text & "<BR>"
     Span1.InnerHtml &= "电话: " & Tel.Text & "<BR>"
     Span1.InnerHtml &= "E-mail: " & mail.Text & "<BR>"
     Span1.InnerHtml &= "地址: " & Addr.Text & "<P>"

 End Sub
 Sub Button3_Click(sender As Object, e As EventArgs)
     Span1.InnerHtml &= "输入完成!"
     Button3.Visible = False
 End Sub
 </script>
```

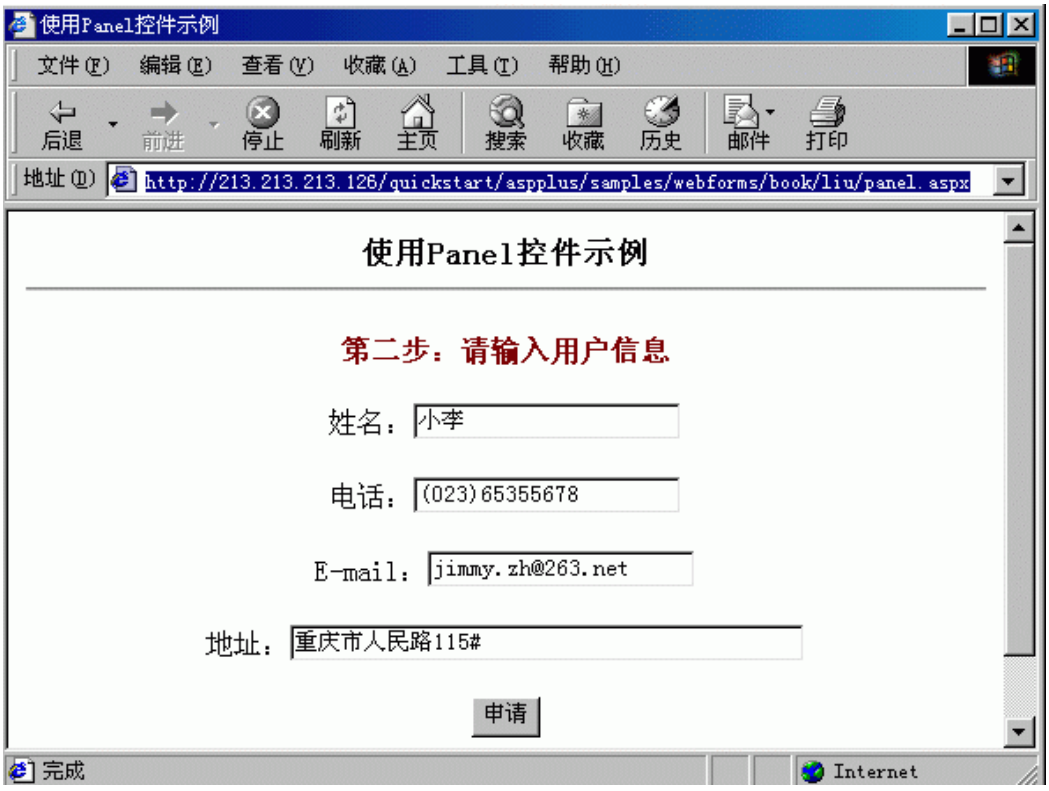
```
<Form runat="server">
<center>
<asp:Panel id="panel1" runat="server">
 <Font Color="#800000"><B>第一步：请输入用户名和密码</B></Font><Blockquote>
       用户名：<asp:TextBox id="UserID" runat="server" Text="kjwang"/><p>
         密码：<asp:TextBox id="Password" TextMode="Password"
                Text="kj6688" runat="server"/><p>
        <Input Type="Button" id="Button1" value=" 登录 "
                OnServerClick="Button1_Click" runat="server">
 </Blockquote>
 </asp:Panel>
 <asp:Panel id="panel2" runat="server">
 <Font Color="#800000"><B>第二步：请输入用户信息</B></Font><Blockquote>
        姓名：<asp:TextBox id="Name" runat="server" Text="小李"/><p>
        电话：<asp:TextBox id="Tel" runat="server" Text="(023)65355678" /><p>
        E-mail：<asp:TextBox id="mail" runat="server" Text="jimmy.zh@263.net" /><p>
        地址：<asp:TextBox id="Addr" runat="server" Text="重庆市人民路 115#" Size="40"
/><p>
        <Input Type="Button" id="Button2" value="申请"
                OnServerClick="Button2_Click" runat="server">
 </Blockquote>
 </asp:Panel>
  <asp:Panel id="panel3" runat="server">
 <Font Color="#800000"><B>第三步：请确认你的输入</B></Font><Blockquote>
     <Span id="Span1" runat="server"/>
     <Input Type="Button" id="Button3" value=" 确认 "
             OnServerClick="Button3_Click" runat="server">
 </Blockquote>
 </asp:Panel>
</center>
 </form>
 <Hr></body>
 </html>
```

请看程序的运行效果：

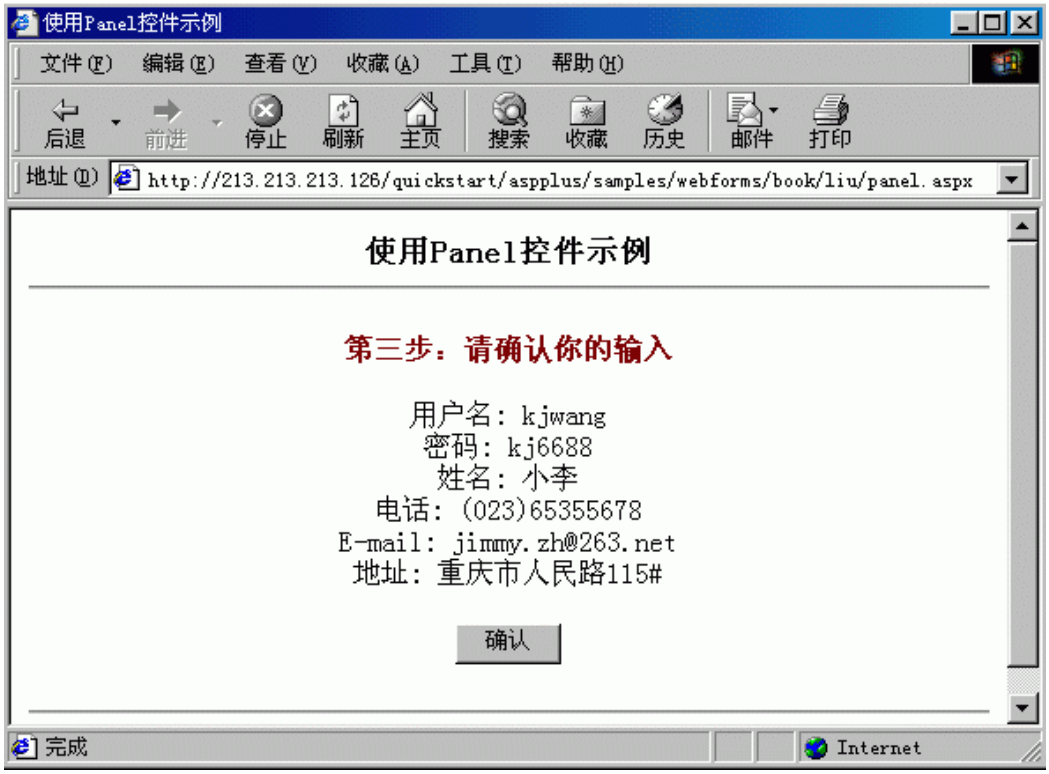
第一步：


使用Panel控件示例
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 前进 停止 刷新 主页 搜索 收藏 历史 邮件 打印
地址(D) http://213.213.213.126/quickstart/aspplus/samples/webforms/book/liu/panel.aspx
使用Panel控件示例
第一步：请输入用户名和密码
用户名：zy
密码：******
登录
完成
Internet

第二步：


使用Panel控件示例
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 前进 停止 刷新 主页 搜索 收藏 历史 邮件 打印
地址(D) http://213.213.213.126/quickstart/aspplus/samples/webforms/book/liu/panel.aspx
使用Panel控件示例
第二步：请输入用户信息
姓名：小李
电话：(023)65355678
E-mail：jimmy.zh@263.net
地址：重庆市人民路115#
申请
完成
Internet

第三步：



大家可以留意浏览器的地址栏，我们注意到地址都是相同的。我就是我们使用 pann 控件所得到的效果。

2.2.10 选择控件

选择的方式有两种：单选、多选。我们下面用具体的例子来说明这两种选择控件在 asp.net 上的实现。

对单选控件，asp.net 里面有一个专用的表示：RadioButtonList，我们看下面的代码：

```
<!--列出选择内容-->
<ASP:RadioButtonList id=ccType Font-Name="Arial" RepeatLayout="Flow"
                     runat=server>
<asp:ListItem>招行一卡通</asp:ListItem>
<asp:ListItem>牡丹卡</asp:ListItem>
</ASP:RadioButtonList>
```

我们在取值的时候，就可以用一个语句：

```
Request.QueryString("ccType")
```

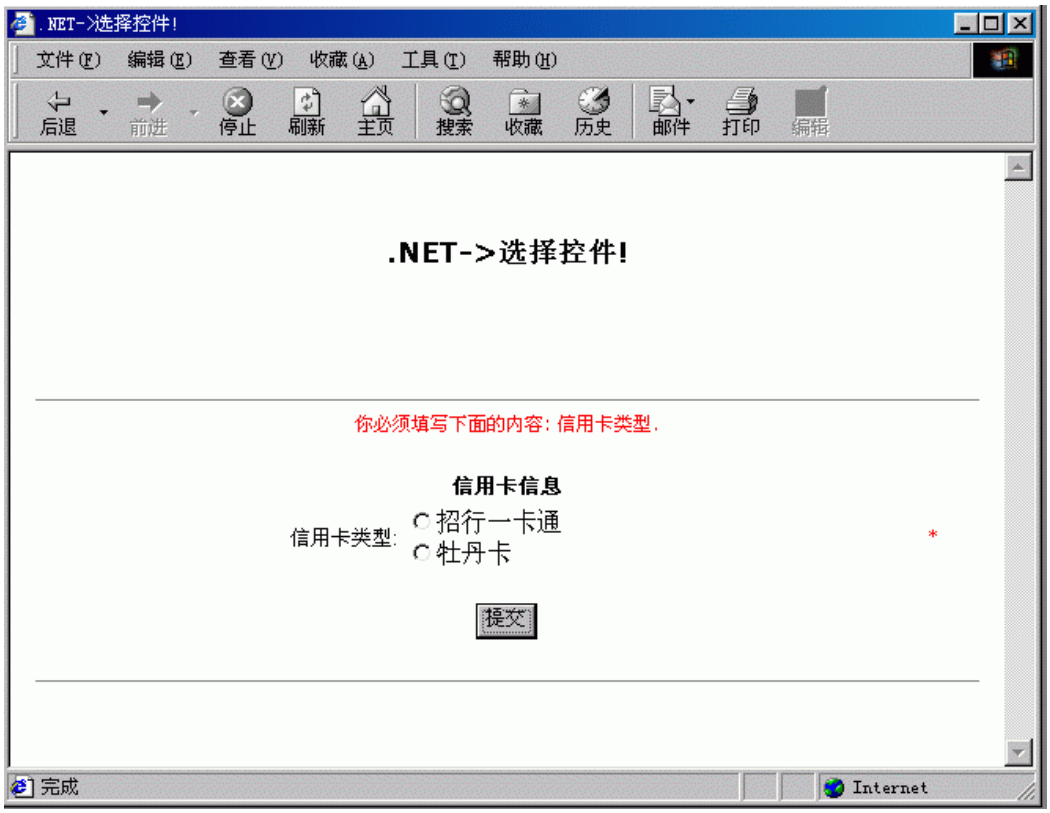
来取得这个值。下面是我们这个说明的完整代码（**RadioButtonList.aspx**）：

```
<!--源文件：form\ServerControl\RadioButtonList.aspx-->
<html>
<body >
<center>
<br><br>
    <h3><font face="Verdana">.NET->选择控件!</font></h3>
<br><br><br>
</center>
    <form method=post runat=server>
    <hr width=600 size=1 noshade>

    <center>
    <asp:ValidationSummary ID="valSum" runat="server"
        HeaderText="你必须填写下面的内容:"
        DisplayMode="SingleParagraph"
        Font-Name="verdana"
        Font-Size="12"
        />
    <p>
    <!-- 信用卡信息 -->
    <table border=0 width=600>
    <tr>
      <td colspan=3>
      <center>
      <font face=Arial size=2><b>信用卡信息</b></font>
      </center>
      </td>
    </tr>
    <tr>
      <td align=right>
        <font face=Arial size=2>信用卡类型:</font>
      </td>
      <td>
    <!--列出选择内容-->
        <ASP:RadioButtonList id=ccType Font-Name="Arial" RepeatLayout="Flow"
runat=server>
            <asp:ListItem>招行一卡通</asp:ListItem>
            <asp:ListItem>牡丹卡</asp:ListItem>
        </ASP:RadioButtonList>
      </td>
      <td>
        <asp:RequiredFieldValidator id="ccTypeReqVal"
            ControlToValidate="ccType"
```

```
            ErrorMessage="信用卡类型. "
            Display="Static"
            InitialValue=""
            Font-Name="Verdana" Font-Size="12"
            runat=server>
            *
        </asp:RequiredFieldValidator>
      </td>
    </tr>
    </table>
    <p>
    <input runat="server" type=submit value="提交">
    <p>
    <hr width=600 size=1 noshade>
</form>
</center>
</body>
</html>
```

我们的运行效果如下：

我们选择一个并提交，则会提交成功；反之，如果我们没有选择就提交，会出现如下的信息：



我们再来看看多选的情况：

```
'选择项列表
<asp:CheckBoxList id=Check1 runat="server">
    <asp:ListItem>北京</asp:ListItem>
    <asp:ListItem>深圳</asp:ListItem>
    <asp:ListItem>上海</asp:ListItem>
    <asp:ListItem>广州</asp:ListItem>
    <asp:ListItem>南宁</asp:ListItem>
    <asp:ListItem>重庆</asp:ListItem>
</asp:CheckBoxList>
```

跟我们上面的单选控件就相差在定义上，我们用 CheckBoxList 来描述我们的选择框，我们写一个方法来响应我们的“提交“事件：

```
'响应按钮事件
Sub Button1_Click(sender As Object, e As EventArgs)
    Dim s As String = "被选择的选项是:<br>"
    Dim i As Int32
    For i = 0 to Check1.Items.Count-1
```

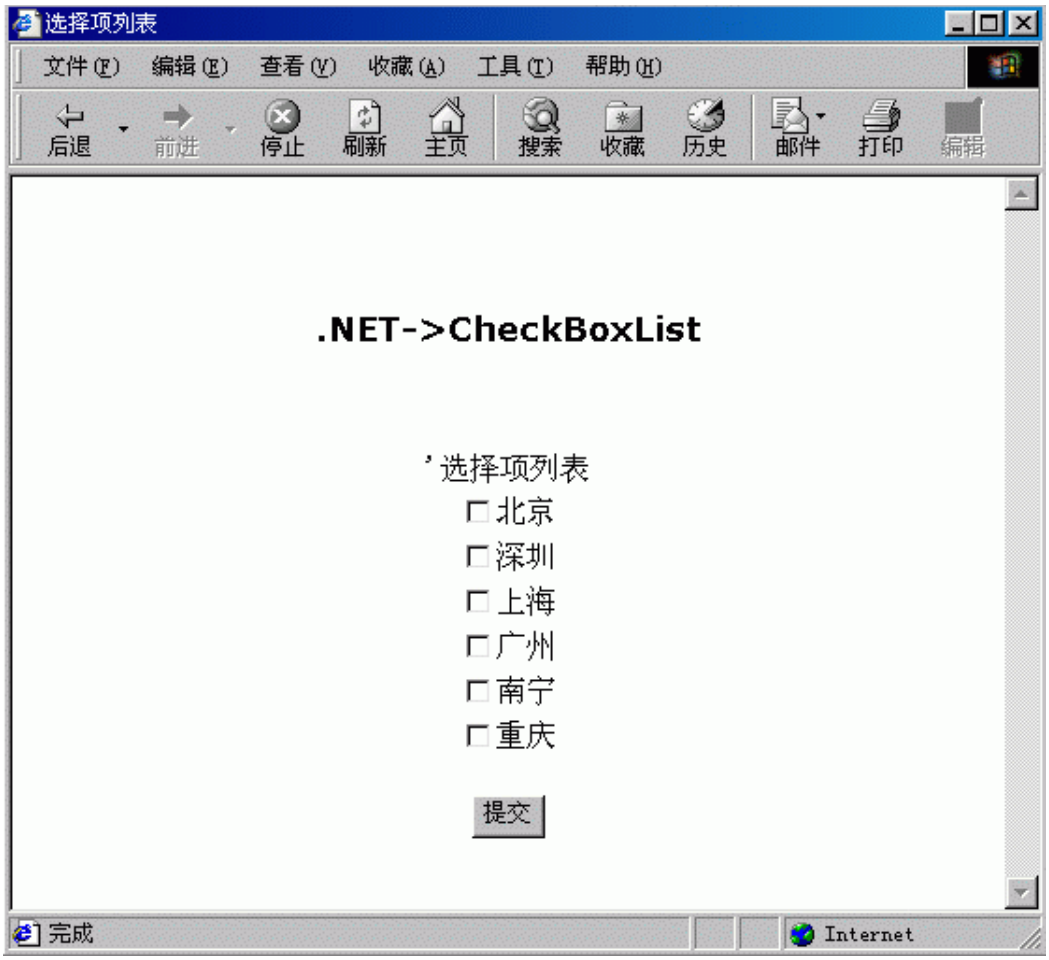
```
                If Check1.Items(i).Selected Then
                    '列出选择项
                    s = s & Check1.Items(i).Text
                    s = s & "<br>"
                End If
            Next
            '打印出选择项
            Label1.Text = s
        End Sub
```

这个方法为定义打印被选择的选项。具体的内容如下**(list.aspx)**：

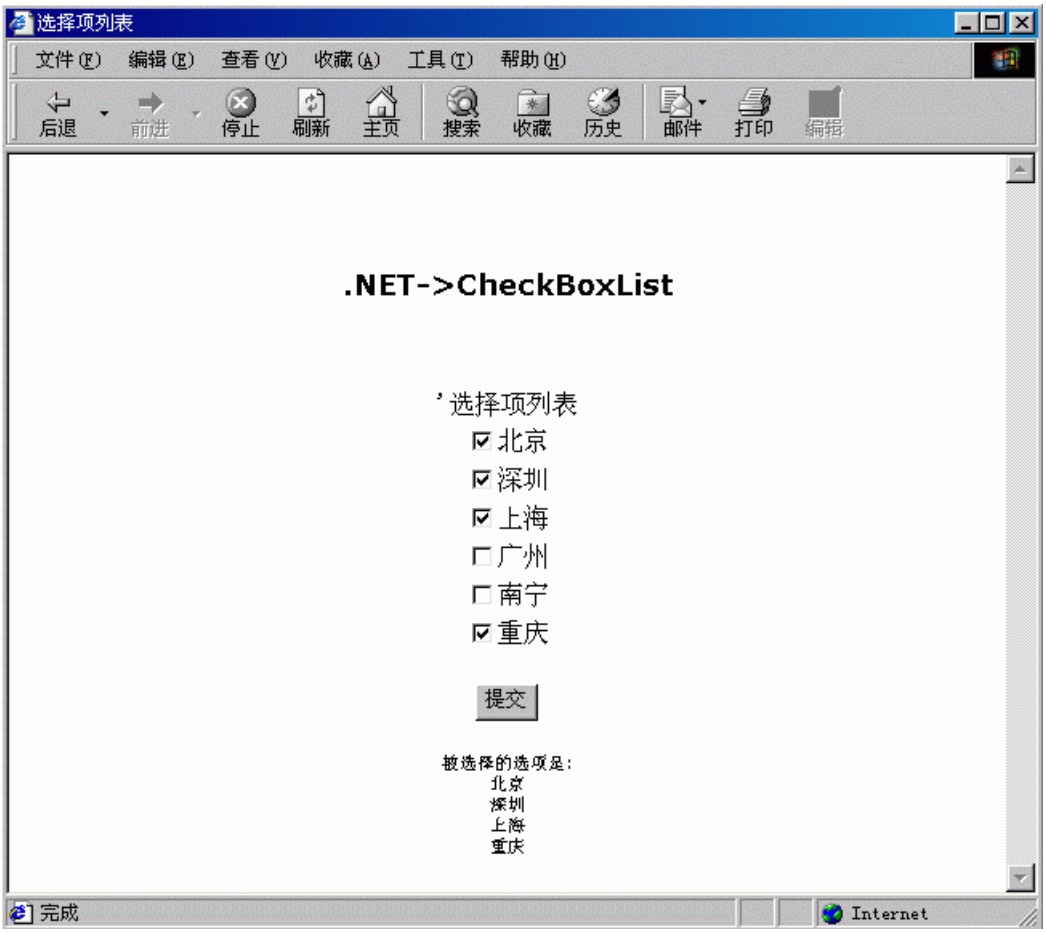
```
<!--源文件：form\ServerControl\list.aspx-->
    <html>
    <head>
    <script language="VB" runat="server">
      '响应按钮事件
      Sub Button1_Click(sender As Object, e As EventArgs)
          Dim s As String = "被选择的选项是:<br>"
          Dim i As Int32
          For i = 0 to Check1.Items.Count-1
              If Check1.Items(i).Selected Then
                  '列出选择项
                  s = s & Check1.Items(i).Text
                  s = s & "<br>"
              End If
          Next
          '打印出选择项
          Label1.Text = s
      End Sub
    </script>
</head>
<body bgcolor="#ccccff">
<br><br><br>
<center>
    <h3><font face="Verdana">.NET->CheckBoxList</font></h3>
</center>
<br><br>
<center>
    <form runat=server>
    '选择象列表
        <asp:CheckBoxList id=Check1 runat="server">
            <asp:ListItem>北京</asp:ListItem>
            <asp:ListItem>深圳</asp:ListItem>
            <asp:ListItem>上海</asp:ListItem>
            <asp:ListItem>广州</asp:ListItem>
```

```
            <asp:ListItem>南宁</asp:ListItem>
            <asp:ListItem>重庆</asp:ListItem>
        </asp:CheckBoxList>
        <p>
            <asp:Button id=Button1 Text="提交" onclick="Button1_Click" runat="server"/>
        <p>
            <asp:Label id=Label1 font-name="Verdana" font-size="8pt" runat="server"/>
    </form>
</center>
</body>
</html>
```

我们看到显示如下：



选择几个选项，并点击“提交“按钮“，看到如下的情况：



2.2.11 ImageButton 控件

ImageButton 控件

当我们在浏览网页的时候，可能会发现这样一种情况：当鼠标移到图象按钮上和当鼠标移走的时候，将会发现同一按钮上将会显示不同的两个图片。我们可以用 Image Button 控件的 onmouseout 和 onmouseover 事件来实现。

请看 **Image.aspx** 中的代码：

```
<!--源文件：form\ServerControl\image.aspx-->
<html>
<Body BgColor="White">
<center><H3>ImageButton 控件演示</H3></center>
<title>ImageButton 控件演示</title>
<script Language="VB" runat="server">
```

```
Sub Button1_Click(sender As Object, e As ImageClickEventArgs)
’定义当我们点击按钮的时候，将访问的网页
Page.Navigate( "http://www.yesky.com" )
End Sub
</script>
<Form runat="server">
<center>
<asp:ImageButton OnClick="Button1_Click"
ImageUrl="18.gif" id="Button1" runat="server"
OnMouseOut="this.src='18.gif';"
OnMouseOver="this.src='19.gif';" />
</center>
</Form>
<asp:Label id="Label1" runat="server"/>
</Body>
</Html>
```

在这段程序中，我们使用了 onmouseout 和 onmouseover 事件。

请看程序的演示效果：

当鼠标移动到按钮上的时候，将显示：

2.2.12 列表控件

在 asp.net 中，有几种方法可以应用于列表控件。我们可以在 aspx 代码中直接嵌入相关的代码，也可以在页面装入的时候加载这些列表信息。

下面是具体的应用，我们先看看在 aspx 中的列表方法：

```
<!--列表->列出内容-->
    <asp:DropDownList id=DropDown1 runat="server">
        <asp:ListItem>北京</asp:ListItem>
        <asp:ListItem>深圳</asp:ListItem>
        <asp:ListItem>上海</asp:ListItem>
        <asp:ListItem>广州</asp:ListItem>
        <asp:ListItem>南宁</asp:ListItem>
        <asp:ListItem>重庆</asp:ListItem>
    </asp:DropDownList>
```

我们在需要取出所选的数据时，直接去取 id 值，即 DropDown1；我们再定一一个方法，响应“提交“按钮的事件，就可以了。下面是完整的代码(**DropDown.aspx**)：

```
<!--源文件：form\ServerControl\dropdown.aspx-->
<html>
<head>

    <script language="VB" runat="server">
```

```
    '在点击按钮时候响应
        Sub list_Click(sender As Object, e As EventArgs)
            Label1.Text="你的选择是: " + DropDown1.SelectedItem.Text
        End Sub

    </script>

</head>
<body bgcolor="#ccccff">
<br><br><br>
<center>
    <h3><font face="Verdana">.NET->列表控件</font></h3>
</center>
<br><br>
<center>
    <form runat=server>
    <!--列表->列出内容-->
        <asp:DropDownList id=DropDown1 runat="server">
            <asp:ListItem>北京</asp:ListItem>
            <asp:ListItem>深圳</asp:ListItem>
            <asp:ListItem>上海</asp:ListItem>
            <asp:ListItem>广州</asp:ListItem>
            <asp:ListItem>南宁</asp:ListItem>
            <asp:ListItem>重庆</asp:ListItem>
        </asp:DropDownList>

        <asp:button text="提交" OnClick="list_Click" runat=server/>

        <p>

        <asp:Label id=Label1 font-name="Verdana" font-size="10pt" runat="server">

        </asp:Label>

    </form>
</center>
</body>
</html>
```

我们看看运行效果：

点击提交按钮时候看到下面的效果：

下面我们再来看看另外一个列表控件的使用，我们定义一个在页面装载的时候调用的方法：

```
'再页面装载的时候调用的方法：
    Sub Page_Load(sender As Object, e As EventArgs)
        If Not IsPostBack Then
            Dim values as ArrayList= new ArrayList()
            values.Add ("北京")
            values.Add ("深圳")
            values.Add ("上海")
            values.Add ("广州")
            values.Add ("南宁")
            values.Add ("重庆")
            '设定 DropDown1 的数据源为 values，即上面定义的信息
            DropDown1.DataSource = values
            '数据的绑定
            DropDown1.DataBind
        End If
    End Sub
```

我们在 aspx 代码中调用它：

```
<!--列出列表信息-->
<asp:DropDownList id="DropDown1" runat="server" />
```

就这样的一个简单的语句就可以了，下面是这个文件的完整的代码：

```
<html>
<head>

<script language="VB" runat="server">
'再页面装载的时候调用的方法：
    Sub Page_Load(sender As Object, e As EventArgs)

        If Not IsPostBack Then
            Dim values as ArrayList= new ArrayList()
            values.Add ("北京")
            values.Add ("深圳")
            values.Add ("上海")
            values.Add ("广州")
            values.Add ("南宁")
            values.Add ("重庆")
        '设定 DropDown1 的数据源为 values，即上面定义的信息
```

```
                DropDown1.DataSource = values
            '数据的绑定
                DropDown1.DataBind
             End If
        End Sub

    '提交按钮响应的方法
        Sub select02_Click(sender As Object, e As EventArgs)
            Label1.Text = "你的选择是: " + DropDown1.SelectedItem.Text
        End Sub

    </script>

</head>
<body BGCOLOR="#CCCCFF">

<br><br><br>
<center>
    <h3><font face="Verdana">.NET->列表控件</font></h3>
</center>
<br><br>
<center>
    <form runat=server>

    <!--列出列表信息-->
        <asp:DropDownList id="DropDown1" runat="server" />

        <asp:button Text="提交" OnClick="select02_Click" runat=server/>
        <p>
        <asp:Label id=Label1 font-name="Verdana" font-size="10pt" runat="server" />

    </form>
</center>
</body>
</html>
```

运行的结果跟上面的一样。

2.2.13 重复列表 Repeator

这种服务器控件会以给定的形式重复显示数据项目，故称之为重复列表。使用重复列表有两个要素，即数据的来源和数据的表现形式。数据来源的指定由控件的 DataSource 属性决定，并调用方法 DataBind 绑定到控件上。这里需要说明的是数据取出以后如何表现的问

题，即如何布局。重复列表的数据布局是由给定的模板来决定的，由于重复列表没有缺省的模板，所以使用重复列表时至少要定义一个最基本的模板“ItemTemplate”。

重复列表支持以下模板标识，所谓模板就是预先定义的一种表现形式，以后我们还会就这个问题专门讨论，这里就不在多说。

1） ItemTemplate 模板，数据项模板，必需的，它定义了数据项极其表现形式。

2） AlternatingItemTemplate 模板，数据项交替模板，为了使相邻的数据项能够有所区别，可以定义交替模板，它使得相邻的数据项看起来明显不同，缺省情况下，它和 ItemTemplate 模板定义一致，即缺省下相邻数据项无表示区分。

3） SeparatorTemplate 模板，分割符模板，定义数据项之间的分割符。

4） HeaderTemplate 模板，报头定义模板，定义重复列表的表头表现形式。

5） FooterTemplate 模板，表尾定义模板，定义重复列表的列表尾部的表现形式。

切记，由于缺乏内置的预定义模板和风格，在使用重复列表时，请一定记住要使用 HTML 格式定义自己的模板。

下面给出一个例子，看它是如何使用重复列表控件的。下面的例子首先在页面加载过程时把数据装载，并绑定到两个重复列表上；然后以一个 2 列的表格显示；最后把所有数据显示到一行上面，并且国家和领导人之间以 3 个中横线分隔，每一国家之间以竖划线分隔。

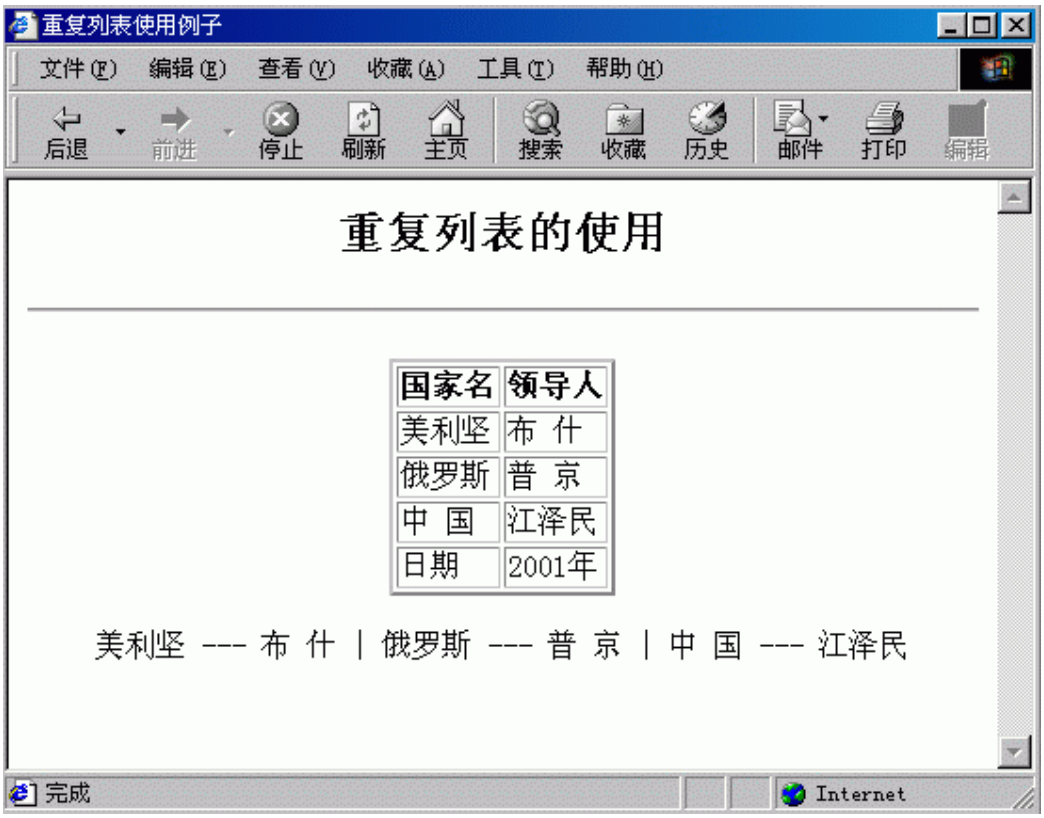
1． 源代码(**FormRepeater.aspx**)

```
<!--源文件：form\ServerControl\FormRepeater.aspx-->
<html>
 <head>
   <script language="vb" runat=server>

       Class Leader
        '定义一个类 Leader
       dim strCountry as String
       dim strName as String

       Public Sub New(country As String, name As String)
                  MyBase.New
                  strName = name
                  strCountry= country
             End Sub

             ReadOnly Property Name As String
               Get
                  Return strName
               End Get
             End Property

             ReadOnly Property Country As String
               Get
                  Return strCountry
               End Get
```

```
            End Property

        End Class

    sub Page_Load(s as object,e as eventargs)
      dim leaders as ArrayList = New ArrayList()
      if Not Page.IsPostBack
      '加载数据
          leaders.add(new leader("美利坚","布　什"))
          leaders.add(new leader("俄罗斯","普　京"))
          leaders.add(new leader("中　国","江泽民"))

          Repeater1.DataSource=leaders
          Repeater2.DataSource=leaders
          Repeater1.DataBind
          Repeater2.DataBind
      end if
    end sub
  </script>
  <title>
    重复列表使用例子
  </title>
</head>

  <center>
   <h2>重复列表的使用</h2>
   <hr>
   <br>
    '以表格形式显示国家，领导人信息
   <asp:Repeater id="Repeater1" runat=server>
        '定义表头
   <template name=HeaderTemplate>
        <table border=2>
          <tr>
            <th>
            国家名
            </th>
            <th>
            领导人
            </th>
          </tr>
      </template>

       '定义数据显示格式
```

```
            <template name=ItemTemplate>
              <tr>
                <td>
                <%# DataBinder.Eval(Container.DataItem,"Country") %>
                </td>
                <td>
                <%# DataBinder.Eval(Container.DataItem,"Name") %>
                </td>
              </tr>
            </template>

             '定义表尾
            <template name=FooterTemplate>
              <tr>
                <td>日期</td>
                <td>2001 年</td>
              </tr>
              </table>
            </template>
    </asp:Repeater>

    <br>
    <asp:Repeater id=Repeater2 runat=server>
        '国家和领导人以|分割显示
      <template name=ItemTemplate>
          <%# DataBinder.Eval(Container.DataItem,"Country") %>
          ---
          <%# DataBinder.Eval(Container.DataItem,"Name") %>
      </template>

      <template name=SeparatorTemplate>
          |
      </template>
    </asp:Repeater>
    </center>
  </body>
<html>
```

2．输出结果



2.2.14 数据列表 DataList

数据列表显示跟重复列表（Repeator）比较类似，但是它可以选择和修改数据项的内容。数据列表的数据显示和布局也如同重复列表都是通过“模板”来控制的。同样的，模板至少要定义一个“数据项模板”（ItemTemplate）来指定显示布局。数据列表支持的模板类型更多，它们如下：

1） ItemTemplate 模板，数据项模板，必需的，它定义了数据项极其表现形式。
2） AlternatingItemTemplate 模板，，数据项交替模板，为了使相邻的数据项能够有所区别，可以定义交替模板，它使得相邻的数据项看起来明显不同，缺省情况下，它和 ItemTemplate 模板定义一致，即缺省下相邻数据项无表示区分。
3） SeparatorTemplate 模板，分割符模板，定义数据项之间的分割符。
4） SelectedItemTemplate 模板，选中项模板，定义被选择的数据项的表现内容与布局形式，当未定义”SelectedItemTemplate”模板时，选中项的表现内容与形式无特殊化，由 ItemTemplate 模板定义所决定。
5） EditItemTemplate 模板，修改选项模板，定义即将被修改的数据项的显示内容与布局形式，缺省情况下，修改选项模板就是数据项模板（ItemTemplate）的定义。
6） HeaderTemplate 模板，报头定义模板，定义重复列表的表头表现形式。
7） FooterTemplate 模板，表尾定义模板，定义重复列表的列表尾部的表现形式。

数据列表还可以通过风格形式来定义模板的字体、颜色、边框。每一种模板都有它自己

的风格属性。例如，可以通过设置修改选项模板的风格属性来指定它的风格。

此外，还有一些其他属性可以导致数据列表的显示有较大的改变，下面择重说明。

RepeatLayout：显示布局格式，指定是否以表格形式显示内容。

RepeatLayout.Table 指定布局以表格形式显示。

RepeatLayout.Flow 指定布局以流格式显示，即不加边框。

RepeatDirection：显示方向，指定显示是横向显示还是纵向显示

RepeatDirection.Horizontal 指定是横向显示

RepeatDirection.Vertical 指定是纵向显示

RepeatColumns：一行显示列数，指定一行可以显示的列数，缺省情况下，系统设置为一行显示一列。这里需要注意的是，当显示方向不同时，虽然一行显示的列数不变，但显示的布局和显示内容的排列次序却有可能大不相同。

例如：有 10 个数据需要显示，RepeatColumns 设定为 4，即一行显示 4 列时

当 RepeatDirection=RepeatDirection.Horizontal 横向显示时，显示布局如下：

Item1 Item2 Item3 Item4

Item5 Item6 Item7 Item8

Item9 Item10

当 RepeatDirection=RepeatDirection.Vertical 纵向显示时，显示布局如下：

Item1 Item4 Item7 Item10

Item2 Item5 Item8

Item3 Item6 Item9

BorderWidth：当 RepeatLayout=RepeatLayout.Table 即以表格形式显示时，边框的线宽度 Unit.Pixel(x) x>=0,当 x 为 0 时无边框

GridLines: 当 RepeatLayout=RepeatLayout.Table 以表格形式显示时，在表格当中是否有网隔线分离表格各单元。

GridLines=GridLines.Both，有横向和纵向两个方向的分割线。

GirdLines=GridLines.None，无论横向还是纵向均无分割线。

例子：演示以上介绍的各属性的设置对数据列表输出的影响，并且当数据项被选中时，数据项以粉红色来反显。

1．源程序(**DataList.aspx**)

```
<!--源文件：form\ServerControl\dataList.aspx-->
<%@ Import Namespace="System.Data" %>
<html>
<script language="VB" runat="server">
'创建初始化表和载入实验数据
    Function LoadData() As ICollection
        Dim dt As DataTable
        Dim dr As DataRow
        Dim i As Integer
        '创建数据表
        dt = New DataTable
        '建立数据项结构
        dt.Columns.Add(New DataColumn("Content", GetType(String)))
```

```
        '载入 10 个实验数据
        For i = 1 To 10
            dr = dt.NewRow()
            dr(0) = "Info " & i.ToString()
            dt.Rows.Add(dr)
        Next
        '为数据表建立一个数据视图，并将其返回
        LoadData = New DataView(dt)
    End Function
    Sub Page_Load(s As Object, e As EventArgs)
        If Not IsPostBack Then
            DataList1.DataSource = LoadData()
            DataList1.DataBind
        End If
    End Sub
    Sub DataList1_ItemCommand(s As Object, e As DataListCommandEventArgs)
        Dim cmd As String = e.CommandSource.CommandName

        If cmd = "select" Then
            DataList1.SelectedIndex = e.Item.ItemIndex
        End If

        DataList1.DataSource = LoadData()
        DataList1.DataBind
    End Sub
'当刷新按钮按下后，对数据列表属性重新设置
    Sub RefreshBtn_Click(s As Object, e As EventArgs)
        If lstDirection.SelectedIndex = 0
            DataList1.RepeatDirection = RepeatDirection.Horizontal
        Else
            DataList1.RepeatDirection = RepeatDirection.Vertical
        End If

        If lstLayout.SelectedIndex = 0
            DataList1.RepeatLayout = RepeatLayout.Table
        Else
            DataList1.RepeatLayout = RepeatLayout.Flow
        End If
        If chkBorder.Checked And DataList1.RepeatLayout = RepeatLayout.Table Then
            DataList1.BorderWidth = Unit.Pixel(1)
        Else
            DataList1.BorderWidth = Unit.Pixel(0)
        End If
```

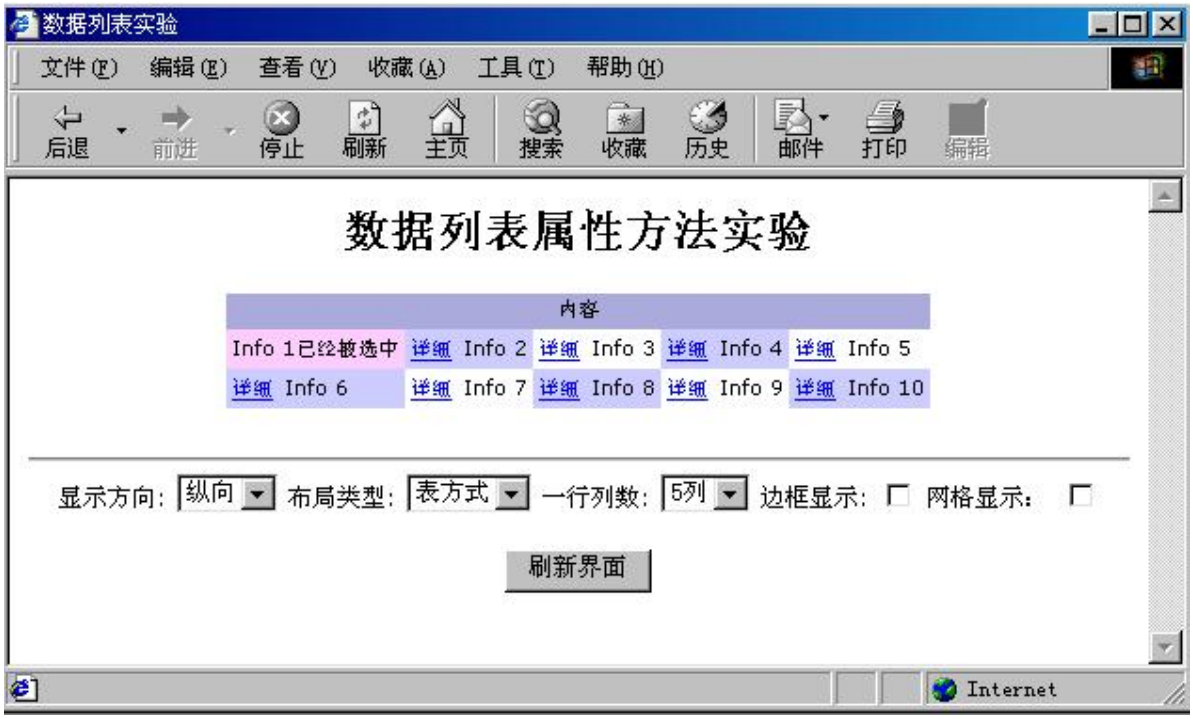
```
        If chkGridLines.Checked And DataList1.RepeatLayout = RepeatLayout.Table then
            DataList1.GridLines = GridLines.Both
        Else
          DataList1.GridLines = GridLines.None
        End If
        DataList1.RepeatColumns=lstColsPerLine.SelectedIndex + 1
    End Sub
</script>
<head>
<title>
数据列表实验
</title>
</head>
<body>
  <center>
  <h2>
  数据列表属性方法实验
  </h2>
    <form runat=server>
    <font face="Verdana" size="-1">
        <asp:DataList id="DataList1" runat="server"
            BorderColor="black"
            CellPadding="3"
            Font-Name="Verdana"
            Font-Size="8pt"
            HeaderStyle-BackColor="#aaaadd"
            AlternatingItemStyle-BackColor="#ccccff"
        SelectedItemStyle-BackColor="#ffccff"
            OnItemCommand="DataList1_ItemCommand"
            >
              <template name="HeaderTemplate">
              <h><center>内容</center></h>
              </template>
              <template name="ItemTemplate">
                <asp:LinkButton id="DetailBtn" runat="server" Text="详细"
CommandName="select" />
                <%# DataBinder.Eval(Container.DataItem, "Content") %>
              </template>
              <template name="SelectedItemTemplate">
               <%# DataBinder.Eval(Container.DataItem, "Content") %>已经被选中
              </template>
        </asp:DataList>
        <p>
        <hr>
```

```
        显示方向:
        <asp:DropDownList id=lstDirection runat="server">
            <asp:ListItem>横向</asp:ListItem>
            <asp:ListItem>纵向</asp:ListItem>
        </asp:DropDownList>
        布局类型:
        <asp:DropDownList id=lstLayout runat="server">
            <asp:ListItem>表方式</asp:ListItem>
            <asp:ListItem>流方式</asp:ListItem>
        </asp:DropDownList>
        一行列数:
        <asp:DropDownList id=lstColsPerLine runat="server">
            <asp:ListItem>1 列</asp:ListItem>
            <asp:ListItem>2 列</asp:ListItem>
            <asp:ListItem>3 列</asp:ListItem>
            <asp:ListItem>4 列</asp:ListItem>
            <asp:ListItem>5 列</asp:ListItem>
        </asp:DropDownList>
        边框显示:
        <asp:CheckBox id=chkBorder runat="server" />
        网格显示:
    <asp:CheckBox id=chkGridLines runat="server" />
        <p>

        <asp:Button id=RefreshBtn Text="刷新界面" OnClick="RefreshBtn_Click"
runat="server"/>

    </font>
    </form>
  </center>
</body>
</html>
```

2. 开始时的界面显示如下，（方向为横向，表方式，一行一列，无边框及网格）

3．当选择显示方向为横向，表方式，一行含 5 列，显示边框和网格时，界面显示如下：

4. 选择纵向显示，表方式，一行含 5 列，无边框，无网格时，界面显示如下：



5．当在步骤 4 的基础上选择了第 5 项数据项时，界面显示如下：

接下来，我们讨论一下一种比较有实际意义的应用，即对选中数据项的修改的实现。

首先是对模板 EditItemTemplate 的定义，通常做法是排列可以进行修改的内容，然后定义一个修改确认键和一个修改取消键。

然后应定义数据列表支持的三种消息处理函数即 OnEditCommand、OnUpdateCommand、OnCancelCommand(编辑事件处理、修改事件处理、撤消修改事件处理)

编辑事件处理：通常设置数据列表的 EditItemIndex 属性为选中的数据项索引，然后重载数据列表。

```
Protected Sub DataList_EditCommand(Source As Object, e As DataListCommandEventArgs)
    DataList1.EditItemIndex = CType(e.Item.ItemIndex, Integer)
    ‘重新加载并绑定数据
    BindList()
End Sub
```

取消修改事件处理：通常设置数据列表的 EditItemIndex 为-1，表示没有数据项需要修改，然后重载数据列表

```
Protected Sub DataList_CancelCommand(Source As Object, e As DataListCommandEventArgs)
    DataList1.EditItemIndex = -1
    BindList()
End Sub
```

修改事件处理：通常先修改数据源的数据，然后设置数据列表的 EditItemIndex 为-1，最后重载数据列表。

```
Sub DataList_UpdateCommand(Source As Object, e As DataListCommandEventArgs)
'修改数据源数据，应根据具体情况而变
ModifySource()
DataList.EditItemIndex=-1
BindList
End Sub
```

例子：显示一个关于书籍修改的实例。一条书籍记录包含序号、书名、价格信息。初始化数据时，我们设置序号为 1-6，书名为“书名”+序号，价格为 1.11*序号。

1． 源程序(**FormDataList01.aspx**)

```
<!--源文件：form\ServerControl\FormDataList01.aspx-->
<%@ Import Namespace="System.Data" %>
<html>
    <script language="VB" runat="server">
    dim Book As DataTable
    dim BookView As DataView
    '设置数据源，并绑定
    Sub BindList()
        DataList1.DataSource= BookView
        DataList1.DataBind
    End Sub

    Sub Page_Load(s As Object, e As EventArgs)

        Dim dr As DataRow
    '如果没有连接变量 session_book，定义数据表 Book,并载入实验数据
    if session("session_Book") = Nothing then
            Book = New DataTable()
            Book.Columns.Add(new DataColumn("num", GetType(string)))
            Book.Columns.Add(new DataColumn("name", GetType(String)))
            Book.Columns.Add(new DataColumn("price", GetType(String)))
            session("session_Book") = Book
            '载入部分测试数据
            For i = 1 To 6
                dr = Book.NewRow()
                dr(0)=i.ToString
                dr(1) = "书名  " & i.ToString
                dr(2) = ( 1.11* i).ToString
                Book.Rows.Add(dr)
            Next
```

```
    '有 session_book 变量，直接引用
        Else
            Book = session("session_Book")
        end if
    '产生数据视图，并按 num 字段排序
     BookView = New DataView(Book)
     BookView.Sort="num"
     '初次需绑定数据源
     if Not IsPostBack then
          BindList
     End If

End Sub

'编辑处理函数
Sub DataList_EditCommand(sender As Object, e As DataListCommandEventArgs)
     DataList1.EditItemIndex = e.Item.ItemIndex
     BindList
End Sub
'取消处理函数
Sub DataList_CancelCommand(sender As Object, e As DataListCommandEventArgs)
     DataList1.EditItemIndex = -1
     BindList
End Sub
'更新处理函数
Sub DataList_UpdateCommand(sender As Object, e As DataListCommandEventArgs)
     Dim lbl1 As Label = e.Item.FindControl("lblNum")
     Dim txt2 As TextBox = e.Item.FindControl("txtBook")
     Dim txt3 As TextBox = e.Item.FindControl("txtPrice")

     dim strNum as String
     dim strBook as String
     dim strPrice as String

     strNum=lbl1.text
     strBook=txt2.text
     strPrice=txt3.text
     '用先删除再插入的方式，实现数据的更新操作
     BookView.RowFilter = "num='" & strNum & "'"
     If BookView.Count > 0 Then
          BookView.Delete(0)
     End If

     BookView.RowFilter = ""
```

```
    dim dr as DataRow=Book.NewRow()
        dr(0) = strNum
        dr(1) = strBook
        dr(2) = strPrice
        Book.Rows.Add(dr)

        DataList1.EditItemIndex = -1
        BindList
    End Sub

    </script>
<head>
<title>
数据列表修改实验
</title>
</head>
<body>
<center>
    <h2>数据列表修改实验</h2>
    <hr>
    <p></p>

    <form runat=server>
    <font face="Verdana" size="-1">
        <!--编辑时显示绿色，并定义编辑、修改、取消时的处理函数-->
        <asp:DataList id="DataList1" runat="server"
            BorderColor="black"
            BorderWidth="1"
            GridLines="Both"
            CellPadding="3"
            CellSpacing="0"
            Font-Name="Verdana"
            Font-Size="8pt"
            Width="150px"
            HeaderStyle-BackColor="#aaaadd"
            AlternatingItemStyle-BackColor="Gainsboro"
            EditItemStyle-BackColor="green"
            OnEditCommand="DataList_EditCommand"
            OnUpdateCommand="DataList_UpdateCommand"
            OnCancelCommand="DataList_CancelCommand"
            >
              <template name="HeaderTemplate">
              <center><h>书籍序号</h></center>
              </template>
```

```
                <template name="ItemTemplate">
                    <asp:LinkButton id="button1" runat="server" Text="详 细 "
CommandName="edit" />
                    <%# Container.DataItem("name")  %>
                </template>
                <template name="EditItemTemplate">
                    书籍: 序号
                    <asp:Label id="lblNum" runat="server" Text='<%#
Container.DataItem("num")  %>' /><br>
                    书名:
                    <asp:TextBox id="txtBook" runat="server" Text='<%#
Container.DataItem("name")  %>' /><br>
                    价格:
                    <asp:TextBox id="txtPrice" runat="server" Text='<%#
DataBinder.Eval(Container.DataItem, "price") %>' />
                    <br>
            <center>
                    <asp:Button id="button2" runat="server" Text="修 改 "
CommandName="update" />
                    <asp:Button id="button3" runat="server" Text="撤 消 "
CommandName="cancel" />
                    </center>
                </template>
            </asp:DataList>
        </font>
        </form>
</center>
</body>
</html>
```

2. 准备对第 2 项进行修改，此时的画面如下：

3.把序号为 2 的书籍的价格改为 9.99 以后，重新进入其编辑状态后，它的输出画面如下：

2.2.15 数据表格 DataGrid

数据表格服务器端控件以表格形式显示数据内容，同时还支持数据项的选择、排序、分页和修改。缺省情况下，数据表格为数据源中每一个域绑定一个列，并且根据数据源中每一个域中数据的出现次序把数据填入数据表格中的每一个列中。数据源的域名将成为数据表格的列名，数据源的域值以文本标识形式填入数据表格中。

通过直接操作表格的 Columns 集合，可以控制数据表格各个列的次序、表现方式以及显示内容。缺省的列为 Bound 型列，它以文本标识的形式显示数据内容。此外，还有许多类型的列类型可供用户选择。

列类型的定义有两种方式：显视的用户定义列类型和自动产生的列类型（AutoGenerateColumns）。当两种列类型定义方式一起使用时，先用用户定义列类型产生列的类型定义，接着剩下的再使用自动列定义规则产生出其他的列类型定义。请注意自动定义产生的列定义不会加入 Columns 集合。

列类型介绍：

1） bound column ，列可以进行排序和填入内容。这是大多数列缺省用法。

两个重要的属性为：HeaderText 指定列的表头显示

DataField 指定对应数据源的域

2） hyperlink column，列内容以 hyperlink 控件方式表现出来。它主要用于从数据表格的一个数据项跳转到另外的一个页面，做出更详尽的解释或显示。

重要的属性有：

HeaderText 指定列表头的显示

DataNavigateUrlField 指定对应数据源的域作为跳转时的参数
DataNavigateUrlFormatString 指定跳转时的 url 格式
DataTextField 指定数据源的域作为显示列内容来源

3） button column，把一行数据的用户处理交给数据表格所定义的事件处理函数。通常用于对某一行数据进行某种操作，例如，加入一行或者是删去一行数据等等。
重要的属性有：

HeaderText 指定列表头的显示
Text 指定按钮上显示的文字
CommandName 指定产生的激活命令名

4） Template column，列内容以自定义控件组成的模板方式显示出来。通常用作用户需要自定义显示格式的时候。

5） Edit Command column，当数据表格的数据项发生编辑、修改、取消修改时，相应处理函数的入口显示。它通常结合数据表格的 EditItemIndex 属性来使用，当某行数据需要编辑、修改、取消操作时，通过它进入相应的处理函数。例如，当需要对某行数据进行修改（update）时，通过它进入修改的处理步骤中。

其他重要列属性介绍：

1） Visible 属性，控制定义的列是否出现在显示的数据列表中。

2） AllowSorting 属性，是否可以进行列排序。当 AollowSorting=true 时，可以以点击列的列表头的方式，把数据以该列次序进行排序。缺省的（即载入数据后）的排序方式，实际上是以数据在数据源中的排列次序进行排序的。

3） AllowPage 属性，是否以分页方式显示数据。当对有大量数据的数据源进行显示时，可以以例如 10 行一页的方式来显示数据，同时显示一个下页/前页的按钮，按下按钮可以以向前或向后的方式浏览整个数据源的数据。当 AllowPage=true 时，即以分页方式进行显示。可以通过设定 CurrentPageIndex 属性来直接跳转到相应的数据页。

例子：演示以上各种类型的列定义的用法

1．源程序**(FormDataGrid.aspx)**

```
<!--源文件：form\ServerControl\FormDataGrid.aspx-->
<%@ Import Namespace="System.Data" %>

<html>
<script language="VB" runat="server">
    dim Order as DataTable
    dim OrderView as DataView

    '对数据表格 1 创建数据表，并返回数据视图
    Function LoadData() As ICollection

        Dim dt As DataTable
        Dim dr As DataRow
        Dim i As Integer
```

```
    '创建数据表
    dt = New DataTable
    dt.Columns.Add(New DataColumn("Num", GetType(Integer)))
    dt.Columns.Add(New DataColumn("Name", GetType(String)))
    dt.Columns.Add(New DataColumn("DtTm", GetType(DateTime)))
    dt.Columns.Add(New DataColumn("Assembly", GetType(Boolean)))
    dt.Columns.Add(new DataColumn("Price", GetType(Double)))

    '载入数据
    For i = 1 To 6
        dr = dt.NewRow()
        dr(0) = i
        dr(1) = "书名  " + i.ToString()
        dr(2) = DateTime.Now.ToShortTimeString
        If (i Mod 2 <> 0) Then
            dr(3) = True
        Else
            dr(3) = False
        End If
        dr(4) = 1.11 * i
        '把产生的数据加入数据表中
        dt.Rows.Add(dr)
    Next


    LoadData = New DataView(dt)

End Function

'页面初始化，分别对 DataGrid1 和 DataGrid2 绑定数据源
Sub Page_Load(sender As Object, e As EventArgs)

    If Session("session_order") = Nothing Then
        Order = New DataTable()
        Order.Columns.Add(new DataColumn("Name", GetType(string)))
        Order.Columns.Add(new DataColumn("Price", GetType(string)))
        Session("session_order") = Order
    Else
        Order = Session("session_order")
    End If
    OrderView = New DataView(Order)
    DataGrid2.DataSource = OrderView
    DataGrid2.DataBind
```

```
        If Not IsPostBack Then
    DataGrid1.DataSource = LoadData()
        DataGrid1.DataBind
        End If

    End Sub

  '对 ButtonColumns 的处理函数集合
    Sub Grid_Command(sender As Object, e As DataGridCommandEventArgs)

        Dim dr As DataRow = order.NewRow()

        Dim Cell1 As TableCell = e.Item.Cells(3)
        Dim Cell2 As TableCell = e.Item.Cells(6)
        Dim name As String = Cell1.Text
        Dim price As String = Cell2.Text

        If e.CommandSource.CommandName = "Add" Then
            dr(0) = name
            dr(1) = price
            order.Rows.Add(dr)
        Else
            OrderView.RowFilter = "name='" & name & "'"
            If OrderView.Count > 0 Then
                OrderView.Delete(0)
            End If
            OrderView.RowFilter = ""
        End If
        DataGrid2.DataBind()
    End Sub

</script>
<head>
<title>
数据表格实验
</title>
</head>

<body>
<center>
  <h2>数据表格列类型实验</h2>
  <hr>
  <p></p>
```

```
    <form runat=server>
      <h3><b>图书清单</b></h3>
      <ASP:DataGrid id="DataGrid1" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
        AutoGenerateColumns="false"
        OnItemCommand="Grid_Command">
          <property name="Columns">
            <!-- 2 个 ButtonColumn 示例-->
            <asp:ButtonColumn HeaderText="操作" Text="订购" CommandName="Add" />
            <asp:ButtonColumn HeaderText="操作" Text="退订" CommandName="Remove"
/>
            <!-- HyperLinkColumn 示例  -->
        <asp:HyperLinkColumn
                HeaderText="链接"
                DataNavigateUrlField="Num"
                DataNavigateUrlFormatString="FormDataGrid01.aspx?id={0}"
                DataTextField="Num"
                Target="_new"
            />
            <!-- 2 个标准 BoundColumn 示例  -->
            <asp:BoundColumn HeaderText="书  名" DataField="Name" />
            <asp:BoundColumn HeaderText="入库时间" DataField="DtTm"/>
            <! -- 1 个 TemplateColumn 示例  ,以 CheckBox 来表示布尔型数据  -->
            <asp:TemplateColumn HeaderText="合  集">
                <template name="ItemTemplate">
                    <asp:CheckBox                ID=Chk1                Checked='<%#
DataBinder.Eval(Container.DataItem, "Assembly") %>' Enabled="false" runat="server" />
                </template>
            </asp:TemplateColumn>

            <asp:BoundColumn    HeaderText="  价      格  "    DataField="Price"
DataFormatString="{0:c}" ItemStyle-HorizontalAlign="right" />
          </property>

      </asp:DataGrid>
     <hr>
    <h3><b>订购清单</b></h3>
```

```
    <ASP:DataGrid id="DataGrid2" runat="server"
        BorderColor="black"
        BorderWidth="1"
        CellPadding="3"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
        />

  </form>
</center>
</body>
</html>
```

文件 **FormDataGrid01.aspx** 的内容：

```
<!--源文件：form\ServerControl\FormDataGrid01.aspx-->
<html>
<head>
<title>
数据表格链接测试实验
</title>
<script language="VB" runat="server">

    Dim num As String

    Sub Page_Load(sender As Object, e As EventArgs)
        num=Request.QueryString("id")
    End Sub

</script>

</head>
<body bgcolor=#ccccff>
<center>
    <h2>数据表格链接测试结果画面</h2>
    <hr>
    <p></p>

    <h4>您选择的是 第<u> <%= num %></u>本藏书</h4>

</body>
</html>
```

2．开始时画面：

3．当选择订购了第一本和第三本后的画面如下：

4．当选择退订第三本书后的画面如下：

5. 当点击连接第六项时的画面如下：

2.2.16 小结

本章主要讲述了几个服务器端的控件、它们的校验、取值方法等，从中我们可以看到 asp.net 中各种控件功能是非常强大的，如上面的例子所示，我们甚至可以用一个简单的语句就可以验证输入的合法性。对取值，我们也有简单的方法，对比于用 html 所写的代码，我们觉得用 asp.net 所写的是简单了很多。

第三章 自定义控件

asp.net 中提供的增加内嵌服务器控件的功能，使你能够多次的轻松增加你所定义的各种控件。事实上，对于表单等各种控件，可以不用更改或者稍微更改一下就可以多次使用的。在通常情况下，我们把一个用作服务器控件的 web 表单统称为用户控件，我们用一个.ascx 为后缀的文件保存起来，这样的保存使得它不被当作一个 web 表单来运行，当我们在一个.aspx 文件中使用它时，我们用 Register 方法来进行调用，假设我们有一个文件名为 saidy.ascx 的文件，我们用下面的语句来调用它：

```
<%@ Register TagPrefix="Acme" TagName="Message" Src="saidy.ascx" %>
```

上面的 TagPrefix 标记为用户控件确定个唯一的名字空间，TagName 为用户控件确定一个唯一的名称，你也可以用其它的名字代替“Message“，Src 为确定所包含的文件名称和路径。这样，我们就可以用下面的语句来调用它了：

```
<Acme:Message runat="server"/>
```

下面我们来看看具体的应用

2.3.1 小页面控件

我们建立两个简单文件来说明这个控件的使用方法：con01.aspx、con01.ascx，在 **con01.ascx** 文件里我们只有一句话：

```
<a href="http://www.yesky.com">欢迎访问天极网站</a>
```

然后我们在文件 **con01.aspx** 里面进行注册：

```
<%@ Register TagPrefix="saidy" TagName="info" Src="con01.ascx" %>
```

页面上的应用我们用这句话来表达：

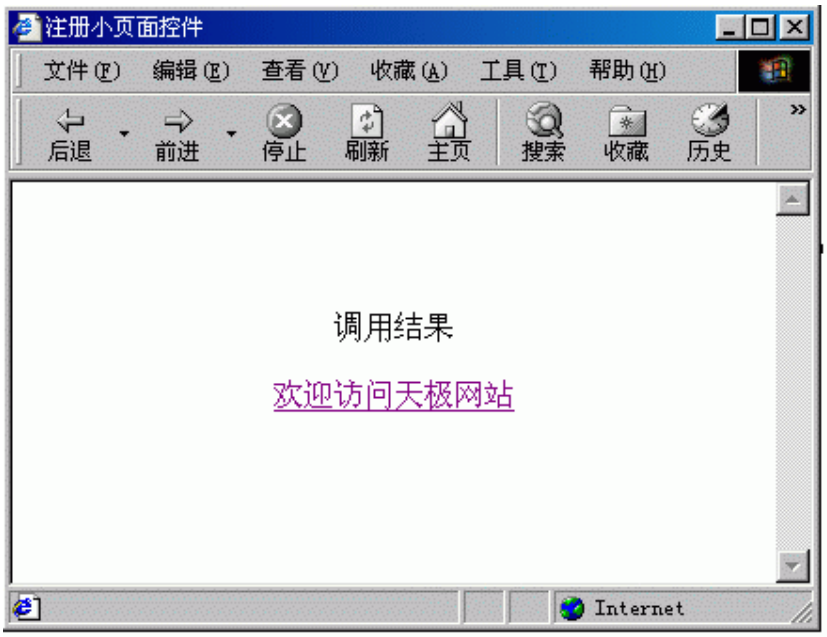
```
<saidy:info runat="server"/>
```

con01.aspx 文件的完整代码如下：

```
<!--源文件：form\CustomControl\con01.aspx-->
```

```
<!--注册小页面控件-->
<%@ Register TagPrefix="saidy" TagName="info" Src="con01.ascx" %>
<html>
<body>
<BR><BR><BR>
<CENTER>
  调用结果
<BR><BR>
  <saidy:info runat="server"/>
</CENTER>
<BR><BR>
</body>
</html>
```

下面我们访问 con01.aspx，显示如下：



2.3.2 代码和模板的分离

在编制 asp.net 程序时，我们会使用模板(Template)。那么什么是模板呢？相信大家都使用过 WORD，当我们在新建一个 WORD 文件的时候，我们可以建立模板。通过使用模板，我们就固定了文档的风格，这样就可以在模板上完善我们的内容。所以我们使用模板一个好处是：文字录入和编排界面是分开的。而且模板可以重复使用。好了，通过上面的介绍，我们对模板就有了一定的认识。 我们在编制.NET 程序时，使用模板将对主程序代码大大简化。模板的定义是使用<template>和</template>标示符的。文件保存为.ascx 文件。下面的代码是一个典型的模板的定义。

```
<template name="itemtemplate">
  <table cellpadding=10 style="font: 10pt verdana">
       <tr>
        <td valign="top">
      <b>所在系: </b><%# DataBinder.Eval(Container.DataItem, "dept") %><br>
      <b>姓名: </b><%# DataBinder.Eval(Container.DataItem, "name") %><br>
      <b>性别: </b><%# DataBinder.Eval(Container.DataItem, "sex") %><br>
      <b>年级: </b><%# DataBinder.Eval(Container.DataItem, "grade") %>
          </td>
        </tr>
</table>
 </template>
```

在这一模板中，我们使用了数据绑定控件，关于数据绑定控件，请参阅其它章节。同时我们还定义了数据的显示方式。那么在主程序中如何调用呢？请看下面的代码：

```
1.  <%@ Register TagPrefix="Acme" TagName="StuList" Src="form32.ascx" %>
2.  <html>
3.  <body style="font: 10pt verdana">
4.  <b><center><h3>模板示例</h3></center></b>
5.  <form runat="server">
6.  <Acme: StuList runat="server"/>
7.  </form>
8.  </body>
9.  </html>
```

其实，模板也属于自定义控件(User Control)，所以我们在使用时，要先注册(Register)。对主程序的第一行代码，TagPrefix 定义了一个不重复的名字空间(Name Space)。TagName 为自定义控件定义了一个名称。然后，我们就要指明使用的模板的文件名。注册完自定义控件后，我们就可以把此控件认为是服务器端控件。要使用服务器端控件，我们要做什么工作呢？对了，要使用 runat=”server”属性了。请参考第 7 行代码。

好了，现在我们就看一个完整的例子！这个例子包含了两个文件，一个主程序文件（template.aspx），另一个是用户自定义控件文件（template.ascx）。先看 template.aspx 文件。

```
<!--源文件：form\CustomControl\template.aspx-->
  <%@ Register TagPrefix="Acme" TagName="stuList" Src="zy.ascx" %>
 <html>
 <body style="font: 10pt verdana">
 <b><center><h3>模板示例</h3></center></b>
 <form runat="server">
 <Acme:stuList runat="server"/>
 </form>
</body>
```

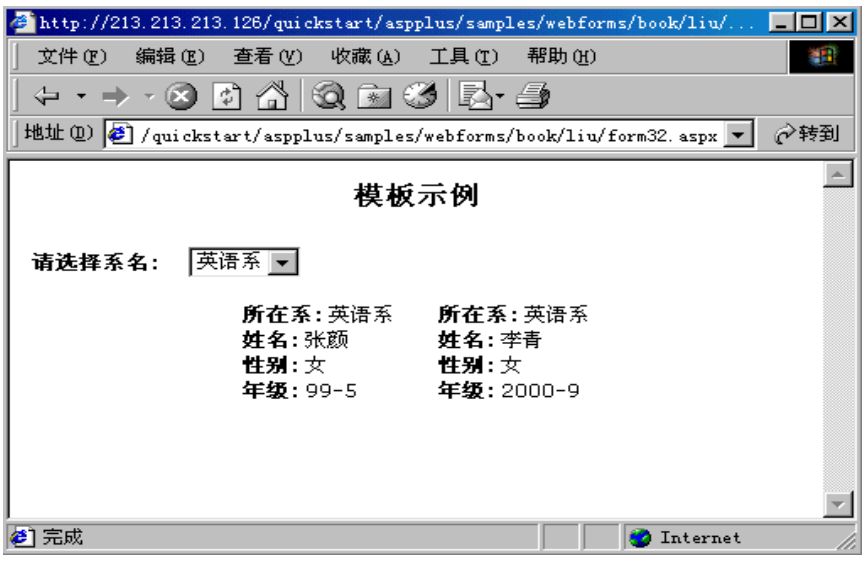
```
</html>
```

现在我们再来看 template**.ascx**：

```
<!--源文件：form\CustomControl\template.ascx-->
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<script language="VB" runat="server">
      Sub Page_Load(Src As Object, E As EventArgs)
            If Not (Page.IsPostBack)
               Dim DS As DataSet
               Dim MyConnection As SQLConnection
               Dim MyCommand As SQLDataSetCommand
    MyConnection = New SQLConnection("server='iceberg';uid=sa;pwd=;database=info")
    MyCommand = New SQLDataSetCommand("select * from infor where dept='" &
Category.SelectedItem.Value & "'", MyConnection)
               DS = New DataSet()
               MyCommand.FillDataSet(DS, "infor")
               MyDataList.DataSource = DS.Tables("infor").DefaultView
               MyDataList.DataBind()
          End If
      End Sub

      Sub Category_Select(Sender As Object, E As EventArgs)
               Dim DS As DataSet
               Dim MyConnection As SQLConnection
               Dim MyCommand As SQLDataSetCommand
        MyConnection = New SQLConnection("server='iceberg';uid=sa;pwd=;database=info")
        MyCommand = New SQLDataSetCommand("select * from infor where dept='" &
Category.SelectedItem.Value & "'", MyConnection)
               DS = New DataSet()
               MyCommand.FillDataSet(DS, "infor")
               MyDataList.DataSource = DS.Tables("infor").DefaultView
               MyDataList.DataBind()
      End Sub
</script>
<table style="font: 10pt verdana">
<center>
<tr>
   <center><td><b>请选择系名:</b></td></center>
<td style="padding-left:15">
<center>                <ASP:DropDownList AutoPostBack="true" id="Category"
OnSelectedIndexChanged="Category_Select" runat="server">
            <ASP:ListItem value="信息系">信息系</ASP:ListItem>
            <ASP:ListItem value="工程系">工程系</ASP:ListItem>
            <ASP:ListItem value="英语系">英语系</ASP:ListItem>
```

```
        </ASP:DropDownList></center>

</td>
  </tr>
</table>
<ASP:DataList id="MyDataList" BorderWidth="0" RepeatColumns="2" runat="server">
<template name="itemtemplate">
   <table cellpadding=10 style="font: 10pt verdana">
          <tr>
            <td valign="top">
              <b>所在系: </b><%# DataBinder.Eval(Container.DataItem, "dept") %><br>
              <b>姓名: </b><%# DataBinder.Eval(Container.DataItem, "name") %><br>
              <b>性别: </b><%# DataBinder.Eval(Container.DataItem, "sex") %><br>
              <b>年级: </b><%# DataBinder.Eval(Container.DataItem, "grade") %>
            </td>
          </tr>
</center>
</table>
      </template>
</ASP:DataList>
```

运行的效果图如下：



这样，一个完整的例子就做好了！实现了代码和模板的分离。试一下吧！

2.3.3 自定义控件

在 asp.net 中，除了我们应用的服务端控件之外，我们还可以创建自己的服务端控件，这样的控件叫 Pagelet。我们来介绍如何创建一个 Pagelet，这个 Pagelet 的功能是在被访问时

返回一个消息。

我们创建一个 Pagelet，用来返回一个消息在客户端的浏览器上：

Welcome.ascx：

<!--源文件：form\CustomControl\welcome.ascx-->

欢迎来到我这里啊!!!

就这么简单，当然你也可以让它复杂一点。当一个 Pagelet 被创建后，我们就可以通过下面的记录指示来调用它：

<% @ Register TagPrefix="wmessage" TagName="wname" Src="Welcome.ascx" %>

TagPrefix 为 Pagelet 指定一个唯一的名字空间，**TagName** 是 **Pagelet** 的唯一名字，当然你也可以换成其他的不是**"wname"**的名称如：TagName="saidy"。Src 属性是指指向 Pagelet 的虚拟路径。

一旦我们注册了 Pagelet，我们就可以向用普通的空件一样来应用它：

<wmessage:wname runat="server"/>

下面的例子示范了自定义的控件的应用（**welcome.aspx**）：

```
<!--源文件：form\CustomControl\welcome.aspx-->
<%@ Register TagPrefix="wmessage" TagName="wname" Src="Welcome.ascx" %>
<html>
<title>自定义的控件</title>
<h3>.NET->Pagelet</h3>
<wmessage:wname runat="server"/>
</body>
</html>
```

客户端的访问如下：

2.3.4 组合控件

1.定义
以类组合形式把已有的控件编译后形成自己定制的控件。实际上组合控件在效果上与利用内置控件形成的用户自定义控件一样，不同处在于，用户自定义控件含有一个.ascx 的纯文本控制文件，而组合控件则利用编译后的代码。
2．步骤
1.)重新定义从 Control 继承来的 CreateChildControls 方法。
2.)如果组合控件要保持于页面上，须完成 System.Web.UI.INamingContainer 接口。
3．例子:
演示一个自定义控件，当选择不同按钮时显示不同内容。
1)控件定义

```
'文件名:form\CustomControl\FormCustom.vb
Option Strict Off

Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

Namespace test
'定义类 tryVB
Public Class tryVB : Inherits Control : Implements INamingContainer
'定义属性 value,实为 TextBox 控件的 Text 属性
Public Property value As String
    Get
    Dim Ctrl As TextBox = Controls(1)
    Return Ctrl.text
    End Get

    Set
    Dim Ctrl As TextBox = Controls(1)
    Ctrl.Text = value
    End Set
End Property

Protected Overrides Sub CreateChildControls()
'重载 CreateChildControls 方法
    Me.Controls.Add(New LiteralControl("选择结果为: "))

    Dim Box As New TextBox
```

```
        Box.Text = "    "
        Me.Controls.Add(box)

End Sub

End Class

End Namespace
```

2)定义控件的编译

:批处理文件 form\CustomControl**FormCustom.bat** 的内容：
vbc /t:library /out:..\bin\testVB.dll /r:System.dll /r:System.Web.dll FormCustom.vb

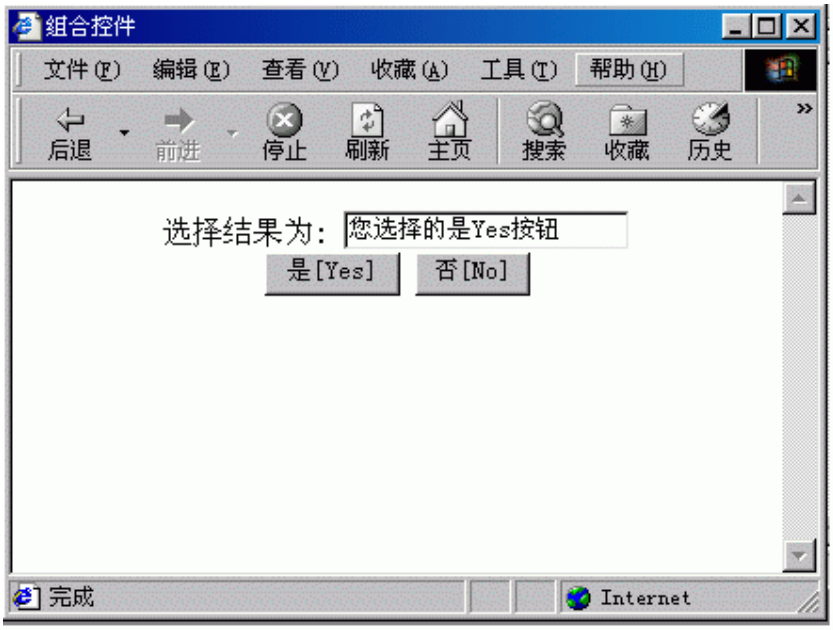
请注意把生成的 testVB.dll 放到正确的目录中，以便 asp.net 解释时能够找到相应的类。

3)自定义组合控件的使用

```
<!--源文件：form\CustomControl\formcustom.aspx-->
<%@ Register TagPrefix="test" Namespace="test" %>
<!--首先注册 test 命名空间-->
<html>
    <script language="VB" runat=server>
      Private Sub LeftBtn_Click(Sender As Object, E As EventArgs)
      '当选择左边的按钮时的显示
          CustControl.Value = "您选择的是 Yes 按钮"
      End Sub
      Private Sub RightBtn_Click(Sender As Object, E As EventArgs)
      '当选择右边的按钮时的显示
          CustControl.value = "您选择的是 No 按钮"
      End Sub

  </script>

  <body>
  <center>
      <form method="POST" action="formcustom.aspx" runat=server>
          <!--引用自定义的组合控件 tryVB-->
          <test:tryVB id="CustControl" runat=server/>
          <br>
          <!--画两个按钮供选择-->
          <asp:button text="是[Yes]" OnClick="LeftBtn_Click" runat=server/>
          <asp:button text="否[No]" OnClick="RightBtn_Click" runat=server/>
      </form>
```

```
    </center>
    </body>
</html>
```

输出结果：



2.3.5 继承控件

在学习了微软公司的.NET 平台为我们提供的大量功能强大的服务器端控件的使用方法以后，随着应用的深入，一些新的问题又出现了。首先是虽然有着大量的控制灵活的控件，但是否就真的满足了我们所有的需求？有时候，我们需要某种控件部分功能，又希望不要费太大的力气去实现，是否可以利用现有的控件来实现。再则，我们希望对某种控件进行改造，使它具有自己所希望的外形或者结果，而不是它缺省的方式运行。最后我们是否可以把自己经常用到的逻辑规则或者是应用界面作成用户控件，然后使用它就如同使用服务器控件那样方便。

其实以上三个问题，在现代面向对象设计方法中，是可以找到答案的。为最大可能的利用现有的开发成果，我们使用“继承”这一手段来节省开发的费用。光有继承不足以形成自己的应用，我们还可以利用“重载”和“多态”来形成自己的应用特点，使之区别于被继承的对象。为了使应用更加简洁和对外隐藏内部的实现、进一步实现代码重用，我们又使用了“封装”。

微软的.NET 平台是支持面向对象的设计方式的新型平台，所以支持并且鼓励用户在应用中设计和使用自己定义的控件。设计用户自己的控件就如同上面所述，有如下步骤：

1. 从 System.Web.UI.Control 类继承，并形成自己的类

为继承 Control 类，我们需引用 System、System.Web、System.Web.UI 类库，在 vb 环境下使用标识 Imports 来引入。为方便使用，我们还需定义一个命名空间以容纳多个类。在 vb 环境中使用 Namespace 空间名和 End Namespace 标识对来定义一个命名空间。定义一个类使用 Class ClassName 和 End Class 标识对。为表明类之间的继承关系，可以使用 Inherits 标识。

继承控件的类定义框架定义如下：

```
Imports System
Imports System.Web
Imports System.Web.UI

Namespace MyNamespace
    Public Class MyClass:Inherits Control
    …
    End Class
End Namespace
```

一个最简单的例子是从 Control 继承一个类，然后重载其 Render 方法。调用其 Render 方法即在页面以 h2 字体写出一行字。

```
Imports System
Imports System.Web
Imports System.Web.UI

Namespace MyNamespace
    Public Class MyClass:Inherits Control

    Protect overrides Sub Render(OutPut  as  HtmlTextWriter)
        OutPut.Write("<h2>这是一个最简单的控件继承例子！</h2>")
    End Sub
    End Class
End Namespace
```

2. 定义自己的属性和方法，包括重载一些初始化的方法。

在 vb 中，以标识 overrides 指明该方法是一个重载函数。例如上面所举的 Render 方法：Protect overrides Sub Render(Output as HtmlTextWriter)

属性定义就较为复杂一点，首先是定义内部变量，可以为 Public 或者是 Private,当为 Public 时可以被外部直接存取，这种方式面向对象方法并不提倡，为 Private 时，不能直接被外部存取，只有通过内部提供的属性定义方式来存取；然后对需要提供给外部使用的内部变量进行属性存取方式定义。在 vb 中使用 Property 属性名 As 类型和 End Property 标识对来定义，Get/End Get 标识对间定义如何通过属性取得内部变量的值，Set/End Set 标识对间定义如何设置内部变量值。

例如：描述一个人的帐号信息，大致需要设定帐号（AcctNo）、身份证号（IdNo）、余额

```
（Balance）、有效状态(Stat)
        Imports System
        Imports System.Web
        Imports System.Web.UI

        Namespace MyNamespace
        ‘定义一个枚举变量，0—正常  1—销户  2—其他状态(挂失、冻结等等)
            Public Enum Status
                Active = 0
                Deactive = 1
                Other = 2
             End Enum

            Public Class Account : Inherits Control

                Private _AcctNo As String
                Private _IdNo As String
                Private _Balance As Currency
                Private _Stat As Status

                Public Property AcctNo As String
                    Get
                        Return _AcctNo
                    End Get
                    Set
                        _AcctNo = Value
                    End Set
                End Property

                Public Property IdNo As String
                    Get
                        Return _IdNo
                    End Get
                    Set
                        _IdNo = Value
                    End Set
                End Property

                Public Property Balance As Currency
                    Get
                        Return _Balance
                    End Get
                    Set
                        _Balance = Value
```

```
            End Set
        End Property

    Public Property   Stat   As   Status
            Get
                Return _Stat
            End Get
            Set
                _Stat = Value
            End Set
        End Property
    …
        End Class

    End Namespace
```

而方法的定义就比较灵活，可以根据设计要求，提供相应的功能，例如大多数的类一般都会提供创建或者是初始化类的方法。我们仍以上面的帐号类为例，定义一个 New 方法：

```
。。。
Public Sub New(AcctNo1 As String,IdNo1 As String,Balance1 As Currency,Stat1 As Status)
MyBase.New
Me.AcctNo = AcctNo1
Me.IdNo = IdNo1
Me.Balance = Balance1
Me.Stat = Stat1
End Sub
。。。
```

3.定义自己应用界面

一个用户自定义的控件一般来说较为复杂，由至少一个以上的内置控件构成，这时就需要重载从 Control 类继承来的 CreateChildControls 方法，并在其中生成界面控件。如果用户定义的控件会在一个页面中反复使用，最好 implements System.Web.UI.INamingContainer，它会为该控件创建一个唯一的命名空间。

例如：下面的例子将创建一个控件，它由一段说明文字和一个文本输入框构成。

```
Imports   System
Imports   System.Web
Imports   System.Web.UI
Imports   System.Web.UI.WebControls

Namespace MyNamespace
Public Class Myclass :Inherits Control : Implements INamingContainer
…
```

```
Protected Overrides Sub CreateChildControls()

        Me.Controls.Add(New LiteralControl("<h3>请输入: "))

        Dim txtBox As New TextBox
        txtBox.Text = ""
        Me.Controls.Add(txtBox)

        Me.Controls.Add(New LiteralControl("</h3>"))
    End Sub
…
End Class
End Namespace
```

4. 定义自己控件的消息处理函数。

自己定义的控件含有两种类型的消息，一是包含的子控件所产生的消息，二是自定义的控件消息。

子控件产生的消息处理函数可由 AddHandler 函数来指定，其用法如下：

AddHandler 子控件.消息，AddressOf 消息处理函数

例如：自定义控件中含有一个 Button 控件，并定义其处理函数 MyBtn_Click()

```
 …
 Private Sub MyBtn_Click(Sender as Objects, E as  EventArgs)
 …
 End Sub

Protected override Sub CreateChildControls()
…
Dim MyBtn  As  New  Button
MyBtn.text=””
AddHandler  MyBtn.Click , AddressOf  MyBtn_Click
Me.Controls.Add(MyBtn)
…
End Sub
```

自定义的控件消息则需要先定义事件说明,格式如下

Public Event 消息名(Sender as Object,E as EventArgs)

例如：Public Event Change(Sender as Object,E as EventArgs)

然后定义事件发出函数,例如：

```
Protected Sub OnChange(E as EventArgs)
  RaiseEvent Change(Me,E)
End Sub
```

再然后定义引起事件发生的过程（可不写）

例如：

```
        Private Sub TextBox_Change(Sender As Object, E As EventArgs)
```

```
            OnChange(EventArgs.Empty)
        End Sub
```

最后定义何时触发事件函数，同样使用 AddHandler 函数

例如：

```
    …
    Protected override Sub CreateChildControls()
…
    Dim MyBox as New TextBox
    MyBox.Text=""
    AddHandler MyBox.TextChanged , AddressOf TextBox_Change
    Me.Controls.Add(MyBox)
…
    End Sub
    …
```

5．最后，谈一谈继承控件的使用，首先应把预先写好的继承控件编译成.DLL 文件

编译格式为：

vbc　/t:library /out:MyDll.dll /r:System.Web.dll MyVb.vb

vbc 为 vb.net 的编译器

/t:表示编译类型，library 为链接库，exe 为独立可执行文件

/out:指定输出文件名

/r:表示需要引用的 DLL 文件

MyVb.vb:指自己编写的继承控件 vb 源程序

然后，为在自己的页面中引用自己定义的控件，需在 aspx 文件头进行注册，

```
<%@ Register TagPrefix="标记前缀" Namespace="命名控件" %>
```

最后，就如同使用内置控件一样，在页面中使用自己定义的控件：

```
<命名空间名：类名 ……　runat=server />
```

下面举一个具体的例子来说明：

我们仍然以开始定义的用户帐号为例来定义一个继承控件，该类有 4 个属性分别为客户帐号、身份证号、帐户余额、帐户状态，其用户界面设定为 4 个文本框供输入属性值以供修改，另外加 2 个按钮以供确认，同时为该控件设定一个事件 Click,当按下确认键后，修改控件属性值，并且在页面中显示自定义控件的属性值，以确认事件确实生效了。

1．控件定义文件

```
'文件名：form\CustomControl\Inherit.vb
Option Strict Off

Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
```

```
Namespace MyNamespace

   Public Enum Status
          Active     = 0
          Deactive = 1
          Other       = 2
   End Enum

Public Class MyAccount:Inherits Control:Implements INamingContainer
'从 Control 类继承，并且有自己的命名空间
   Private _AcctNo As String
   Private _IdNo As String
   Private _Balance As Decimal
   Private _Stat As Status

   Public Event Click(Sender as Object,E as EventArgs)
'定义控件自身的 Click 事件

'对属性存取的定义
        Public Property AcctNo As String
            Get
                Return _AcctNo
            End Get
            Set
                _AcctNo = Value
            End Set
        End Property

        Public Property IdNo As String
            Get
                Return _IdNo
            End Get
            Set
                _IdNo = Value
            End Set
        End Property

        Public Property Balance As Decimal
            Get
                Return _Balance
            End Get
            Set
                _Balance = Value
```

```
            End Set
        End Property

        Public Property    Stat    As    Status
            Get
                Return _Stat
            End Get
            Set
                _Stat = Value
            End Set
        End Property

Public Sub New()
    MyBase.New
    Me.AcctNo = ""
    Me.IdNo = ""
    Me.Balance = "0.0"
    Me.Stat = "0"
End Sub

Protected Sub OnClick(E as EventArgs)
    RaiseEvent Click(Me,E)
End Sub



'界面定义为 4 个属性文本输入框，加一个确定和取消键
Protected Overrides Sub CreateChildControls()
    Me.Controls.Add(New LiteralControl("<h3>客户帐号："))
    dim txtAcctNo as New TextBox
    txtAcctNo.text=_AcctNo
    Me.Controls.Add(txtAcctNo)

    Me.Controls.Add(New LiteralControl("<br>身份证号："))
    dim txtIdNo    as New TextBox
    txtIdNo.text=_IdNo
    Me.Controls.Add(txtIdNo)

    Me.Controls.Add(New LiteralControl("<br>帐户余额："))
    dim txtBalance as New TextBox
    txtBalance.text=_Balance
    Me.Controls.Add(txtBalance)

    Me.Controls.Add(New LiteralControl("<br>帐户状态："))
```

```
    dim txtStat      as New TextBox
    txtStat.text=_Stat
    Me.Controls.Add(txtStat)

    Me.Controls.Add(New LiteralControl("<br><br><br>"))
    dim Btn1 as New Button
    Btn1.text="确  认"
    AddHandler Btn1.Click,AddressOf Btn1_Click
    Me.Controls.Add(Btn1)

    dim Btn2 as New Button
    Btn2.text="取  消"
    Me.Controls.Add(Btn2)

    Me.Controls.Add(New LiteralControl("</h3>"))
End Sub

Private Sub Btn1_Click(Sender as Object,E as EventArgs)
dim ctrl1 as TextBox=controls(1)
Me.AcctNo=ctrl1.text
dim ctrl2 as TextBox=controls(3)
Me.IdNo=ctrl2.text
dim ctrl3 as TextBox=controls(5)
Me.Balance=CDbl(ctrl3.text)
dim ctrl4 as TextBox=controls(7)
Me.Stat=Cint(ctrl4.text)
OnClick(E)
End Sub

End Class

End Namespace
```

2． 控件编译文件

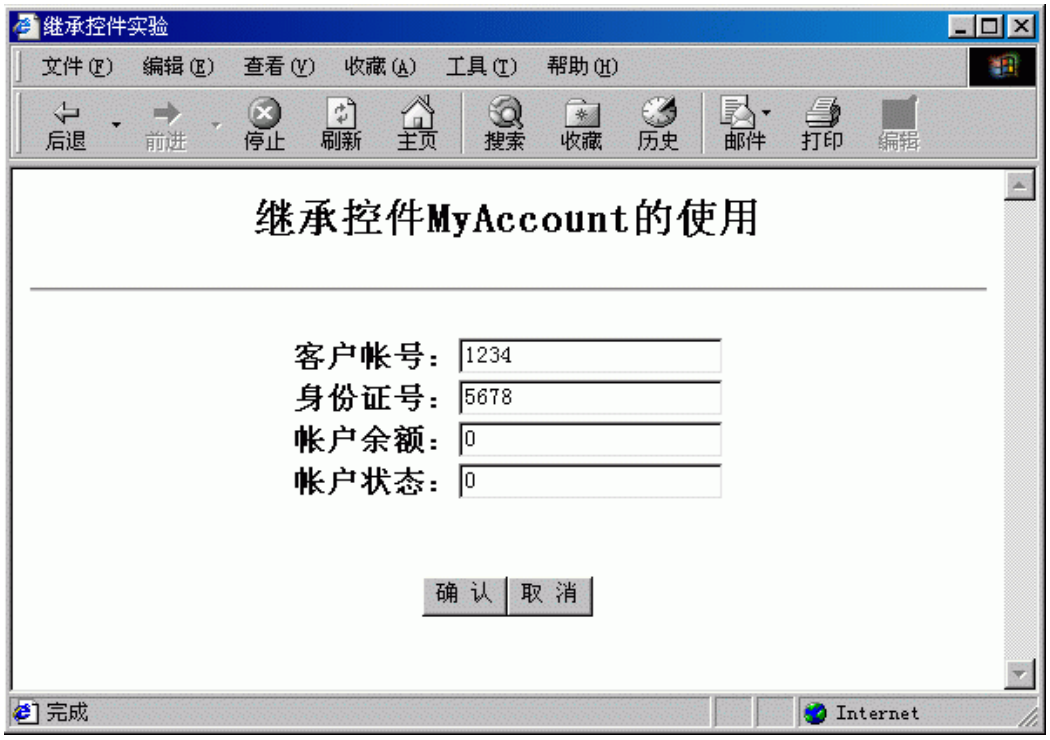
```
rem inherit.vb 的编译文件 文件名：form\CustomControl\i.bat
vbc    /t:library /out:.\bin\MyNamespaceVB.dll /r:System.Web.dll inherit.vb
```

3. 页面使用文件

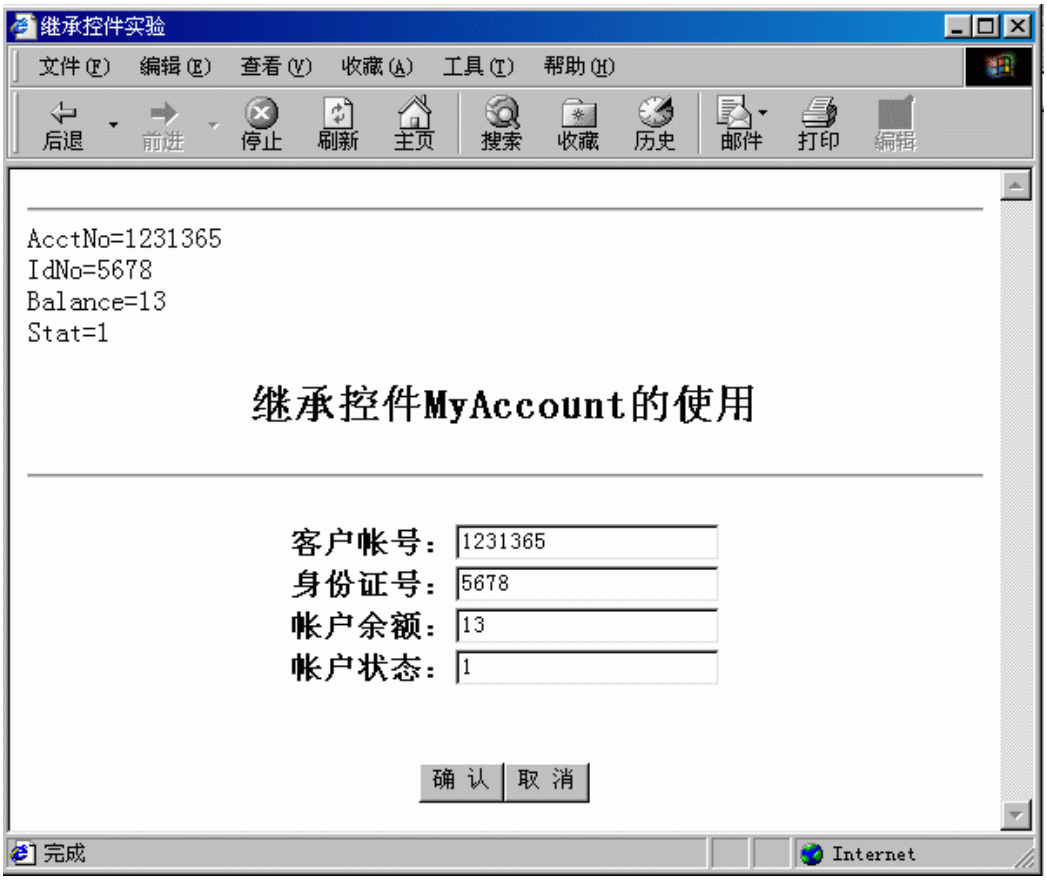
```
<!--源文件：form\CustomControl\FormInherit.aspx-->
<%@ Register TagPrefix="MyNamespace" Namespace="MyNamespace" %>
<html>
```

```
<script language="vb" runat=server>
sub acct_click(s as object, e as eventargs)
dim strTxt as string
strTxt="<hr>AcctNo="& acct1.AcctNo & "<br>"
strTxt=strTxt & "IdNo=" & acct1.IdNo & "<br>"
strTxt=strTxt & "Balance=" & acct1.Balance & "<br>"
strTxt=strTxt & "Stat=" & acct1.Stat
response.write(strTxt)
end sub
</script>
<head>
<title>
继承控件实验
</title>
</head>

<body bgcolor=#ccccff>
<center>
<h2>继承控件 MyAccount 的使用</h2>
<hr>
<br>
<form action="forminherit.aspx" method="post" runat=server>
<MyNamespace:MyAccount id="acct1" AcctNo="1234" IdNo="5678" OnClick="acct_click"
runat=server />
</form>
</center>
</body>
</html>
```

4.开始的输出画面：



5.修改后，按确认键后的效果：



2.3.6 小结

本章在前一章学习了服务器端控件的基础上，探讨了如何在利用已有控件的基础上，开发具有自己特色的自定义控件。使用自定义控件的好处在于：

1. 简化了自己程序开发的周期
2. 有助于形成具有自己特色的风格
3. 隔离了错误发生的根源，修改自己定义的控件时，不必考虑其他因素

第四章 HTML 控件

HTML 控件在服务器端是可见的，所以我们可以根据它来按照我们的意愿来编写。HTML 控件表现为一些可见的控件。

2.4.1 HtmlButton

HtmlButton server control 就象 HTML4.0 中的<button>一样，但是这与<Input type=”button”>不一样的，我们看下面的例子 button.aspx：

响应按钮事件：

```
<script language="VB" runat="server">
    Sub Button1_OnClick(sender As Object, e As EventArgs)
        Span1.InnerHtml="你点击了 Button1"
    End Sub
    Sub Button2_OnClick(sender As Object, e As EventArgs)
        Span1.InnerHtml="你点击了 Button2"
    End Sub
</script>
```

对两个 button 的描述：

button1：

```
<button id="Button1" onServerClick="Button1_OnClick" style="font: 8pt
        verdana;background-color:lightgreen;border-color:black;height=30;width:1
        00" runat="server">
    <img src="/quickstart/aspplus/images/right4.gif"> Click me!
</button>
```

button2，我们增加了鼠标事件：

```
<button id=Button2 onServerClick="Button2_OnClick"   style="font: 8pt
    verdana;background-color:lightgreen;border-color:black;height=30;width:100"
    onmouseover="this.style.backgroundColor='yellow'"
    onmouseout="this.style.backgroundColor='lightgreen'"
```

```
        runat="server">
        Click me too!
    </button>
```

点击 button2，并把鼠标移到它的上面：

第二节 HtmlForm

一个 HtmlForm Control 必须要处理 PostBack 请求，一个 Web Form 只有一个<form>标记。**Form.aspx** 中 form 的表示：

```
<form id=ServerForm runat=server>
  <button id=Button1 runat="server"
          onServerClick="Button1_OnClick">Button1</button>

    
  <span id=Span1 runat=server />
  <p>
  <button id=Button2 runat="server"
          onServerClick="Button2_OnClick">Button2</button>
    
  <span id=Span2 runat=server />
</form>
```

响应鼠标按钮事件：

```
Sub Button1_OnClick(sender As Object, e As EventArgs)
    Span1.InnerHtml = "It is Button1"
End Sub
Sub Button2_OnClick(sender As Object, e As EventArgs)
    Span2.InnerHtml = "It is Button2"
End Sub
```

看如下结果：

点击两个按钮，同时显示信息：

2.4.2 HtmlImages

我们通过一个<img src="">这个标记来显示图片：

```
<img ID="Image1" src="images/4.gif" runat="server"/>
```

我们根据 ID 号为<img >提供图片来源：

```
Sub SubmitBtn_Click(sender As Object, e As EventArgs)
    Image1.Src="images/" & Select1.Value
End Sub
```

建立一个选择控件来与用户的交互：

选择面部表情文件:

```
<select id="Select1" runat="server">
    <option Value="4.gif">4</option>
    <option Value="5.gif">5</option>
    <option Value="6.gif">6</option>
    <option Value="7.gif">7</option>
    <option Value="8.gif">8</option>
    <option Value="9.gif">9</option>
</select>
<input type="submit" runat="server" Value="提交"
        OnServerClick="SubmitBtn_Click">
```

我们运行如下：

选择相应的文件号，点击按钮，图片就显示出来。

2.4.3 TextArea

象在 HTML 中的一样，在 asp.net 中的 TextArea 也是一个多行输入框。TextArea 的宽度由他的 Cols 属性决定，长度由 Rows 属性决定。

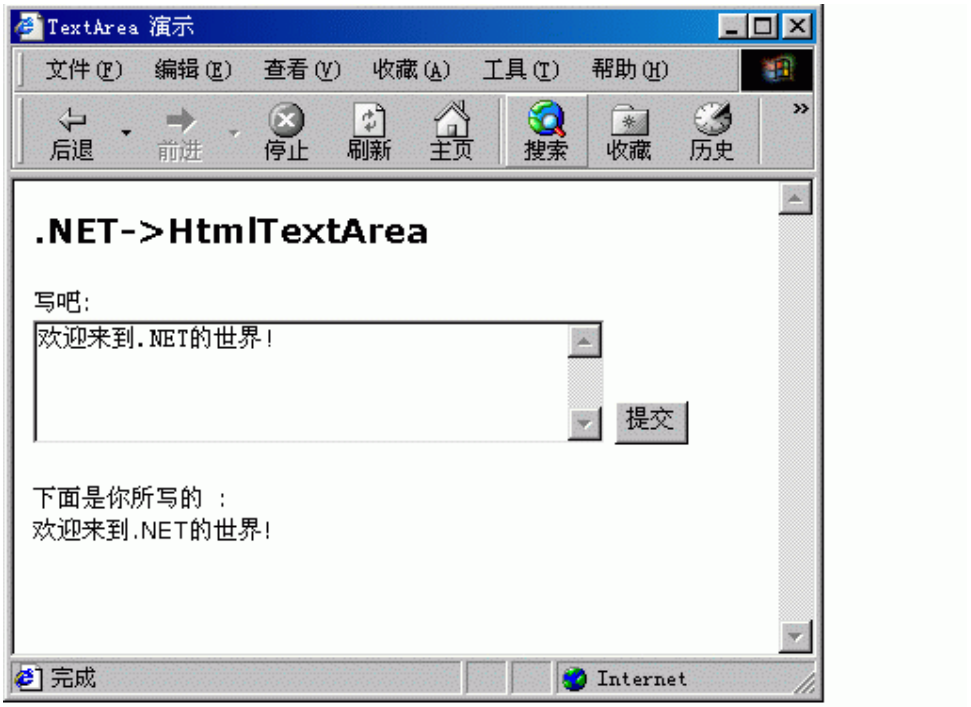
Textarea.aspx 中定义输入：

<textarea id="TextArea1" cols=40 rows=4 runat=server />

我们用 TextArea1.Value 取得输入的值。具体如下(**textarea.aspx**)：

```
<!--源文件： form\HtmlControl\textarea.aspx-->
<html>
<head>
    <script language="VB" runat="server">
        Sub SubmitBtn_Click(sender As Object, e As EventArgs)
            Span1.InnerHtml = "下面是你所写的 ：<br>" & TextArea1.Value
        End Sub
    </script>
</head>
<body bgcolor="#ccccff">
    <h3><font face="Verdana">.NET->HtmlTextArea</font></h3>
    <form runat=server>
```

```
        <font face="Verdana" size="-1">
          写吧: <br>
          <textarea id="TextArea1" cols=40 rows=4 runat=server />
          <input type=submit value="Submit" OnServerClick="SubmitBtn_Click" runat=server>

          <p>
          <span id="Span1" runat="server" />
        </font>
      </form>
</body>
</html>
```



2.4.4 InputHidden

我们可以用隐藏输入控件来处理一些我们要传送而又不想在页面上显示出来的信息，例如在电子商务网站中，我们向银行网关接口传送我们的订单信息，我们就可以用隐藏输入控件来处理。

我们下面的例子用不可见的值来取得输入值，再把不可见值显示出来。

Inputhidden.aspx 隐藏输入控件：

```
<input id="HiddenValue" type=hidden value="隐藏的字符" runat=server>
```

初始值为“隐藏的字符“，在我们第一次点击按钮时候显示出来，我们的方法：

```
        Sub SubmitBtn_Click(sender As Object, e As EventArgs)

            HiddenValue.Value = StringContents.Value
        End Sub
```

这个方法把输入值赋给不可见的控件。我们完整的代码如下**(hidden.aspx)**：

```
<!--源文件：  form\HtmlControl\hidden.aspx-->
<html>
<head>
    <script language="VB" runat="server">
        Sub Page_Load(sender As Object, e As EventArgs)
            If IsPostBack Then
                Span1.InnerHtml="隐藏值: <b>" & HiddenValue.Value & "</b>"
            End If
        End Sub
        Sub SubmitBtn_Click(sender As Object, e As EventArgs)
            HiddenValue.Value = StringContents.Value
        End Sub
    </script>
</head>
<body>
    <h3><font face="Verdana">.NET->HtmlInputHidden</font></h3>
    <form runat=server>
         <input id="HiddenValue" type=hidden value="隐藏的字符" runat=server>
         请输入: <input id="StringContents" type=text size=40 runat=server>
         <p>
         <input type=submit value="确定" OnServerClick="SubmitBtn_Click" runat=server>
         <p>
         <span id=Span1 runat=server>显示隐藏的字符</span>
    </form>
</body>
</html>
```

我们输入 InputHidden，便点击按钮，则显示出默认的隐藏值。

2.4.5 HtmlTable

HtmlTable 服务控件能让你轻松的创建你的表格的行和列，也可以按照程序的模式自动生成表格。

我们的例子展示了这个特性：

<table id="Table1" CellPadding=4 CellSpacing=0 Border="1" runat="server" />

这就是在 asp.net 中，表格的表示。做两个 Select 控件来让用户选择表格的属性：

```
<p>
  行:
  <select id="Select1" runat="server">
      <option Value="1">1</option>
      <option Value="2">2</option>
      <option Value="3">3</option>
      <option Value="4">4</option>
  </select>
  <br>
  列:
  <select id="Select2" runat="server">
      <option Value="1">1</option>
```

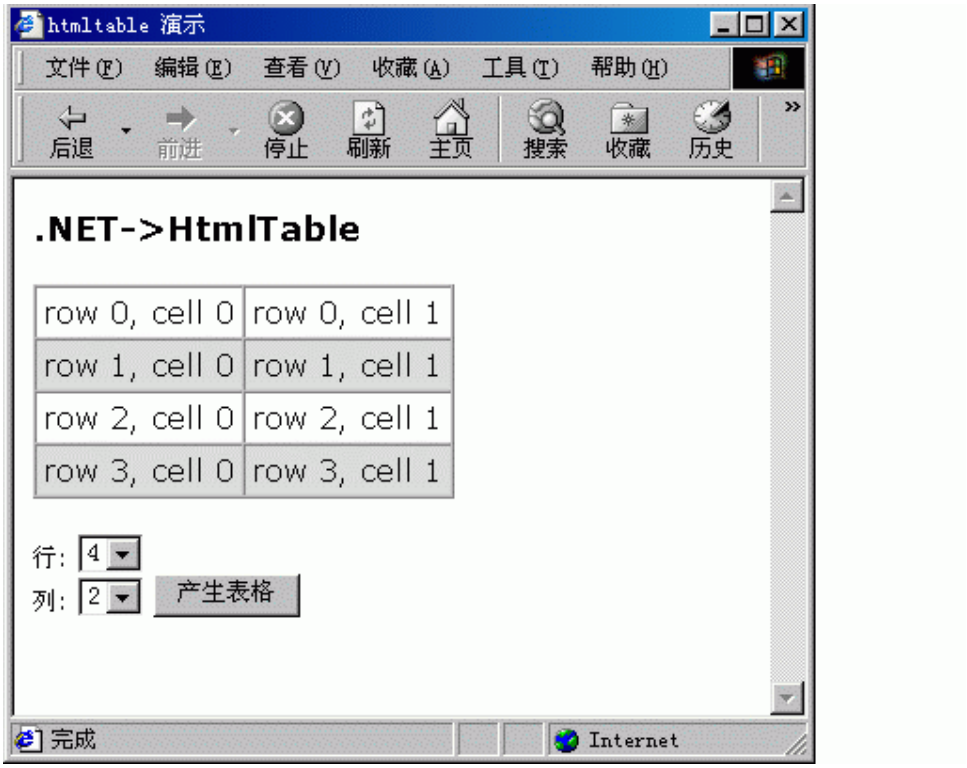
```
            <option Value="2">2</option>
            <option Value="3">3</option>
            <option Value="4">4</option>
        </select>
```

在用户提交的时候，实际上我们示对页面进行了刷新，即在 Page_Load 方法里面处理，具体如下(**htmltable.aspx**)：

```
<!--源文件：  form\HtmlControl\htmltable.aspx-->
<html>
<head>
    <script language="VB" runat="server">
        Sub Page_Load(sender As Object, e As EventArgs)
            Dim numrows As Integer
            Dim numcells As Integer
            Dim i As Integer = 0
            Dim j As Integer = 0
            Dim Row As Integer = 0
            Dim r As HtmlTableRow
            Dim c As HtmlTableCell

            ' 产生表格
            numrows = CInt(Select1.Value)
            numcells = CInt(Select2.Value)
            For j = 0 To numrows-1
                r = new HtmlTableRow()
                If (row Mod 2 <> 0) Then
                    r.BgColor = "Gainsboro"
                End If
                row += 1
                 For i = 0 To numcells-1
                    c = new HtmlTableCell()
                    c.Controls.Add(new LiteralControl("row " & j & ", cell " & i))
                    r.Cells.Add(c)
                Next i
                  Table1.Rows.Add(r)
            Next j
        End Sub
    </script>
</head>
<body>
    <h3><font face="Verdana">.NET->HtmlTable</font></h3>
    <form runat=server>
    <font face="Verdana" size="-1">
        <p>
        <table id="Table1" CellPadding=4 CellSpacing=0 Border="1" runat="server" />
```

```
            <p>
            行:
            <select id="Select1" runat="server">
                <option Value="1">1</option>
                <option Value="2">2</option>
                <option Value="3">3</option>
                <option Value="4">4</option>
            </select>
            <br>
            列:
            <select id="Select2" runat="server">
                <option Value="1">1</option>
                <option Value="2">2</option>
                <option Value="3">3</option>
                <option Value="4">4</option>
            </select>
             <input type="submit" value="产生表格" runat="server">
        </font>
        </form>
</body>
</html>
```

选择并提交，我们的表格就出来了：



2.4.6 HtmlGenericControl

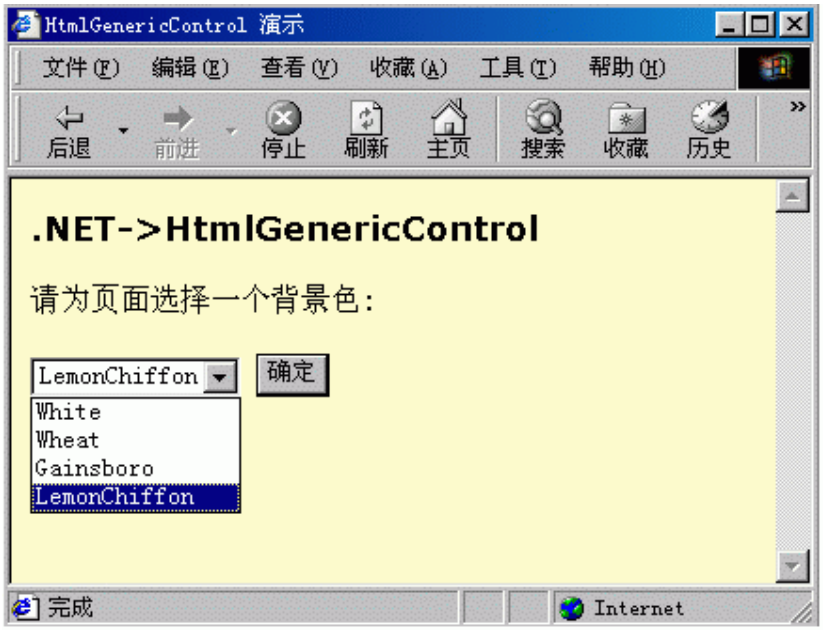
HtmlGenericControl 提供一个服务器控件，用来执行那些不直接的表现出来的未知的 Html Control 标识。

例子文件 **Gerecolor.aspx**：

```
<!--源文件：  form\HtmlControl\Gerecolor.aspx-->
<html>
<head>
    <script language="VB" runat="server">
    Sub SubmitBtn_Click(sender As Object, e As EventArgs)
        Body.Attributes("bgcolor") = ColorSelect.Value
    End Sub
    </script>
</head>
<body id=Body runat=server>
    <h3><font face="Verdana">.NET->HtmlGenericControl</font></h3>
    <form runat=server>
      <p>
      Select a background color for the page: <p>
```

```
        <select id="ColorSelect" runat="server">
            <option>White</option>
            <option>Wheat</option>
            <option>Gainsboro</option>
            <option>LemonChiffon</option>
        </select>
        <input type="submit" runat="server" Value="Apply" OnServerClick="SubmitBtn_Click">
    </form>
</body>
</html>
```

我们的运行结果如下：



选择你所要的颜色，则面背景颜色就会改变。

2.4.7 HtmlInputButton

其实我们在上面的应用的时候就大概的应用过这个控件的，现在我们专门来讲讲它。这个控件有几个功能，可以是普通的按钮来响应一般的事件；可以是 Submit 按钮；也可以是 Reset 按钮。

2.4.7.1 一般性的按钮

这个控件不是响应表单中通常的 Submit 或者 Reset 事件的，而是响应我们为他定制的的事件(**button.aspx**)。

```
<!--源文件： form\HtmlControl\button.aspx-->
```

```
<html>
<head>
    <script language="VB" runat="server">
        Sub Button1_Click(sender As Object, e As EventArgs)
            Span1.InnerHtml = "你点击了这个按钮！"
        End Sub
    </script>
</head>
    <h3><font face="Verdana">.NET->HtmlInputButton->button</font></h3>
    <form runat=server>
        <p>
        <input type=button value="Button1" onServerClick="Button1_Click" runat="server">
          
        <span id=Span1 runat=server />
    </form>
</body>
</html>
```

点击按钮，如下：

2.4.7.2 Submit 与 Reset

对这两个属性大家肯定很熟悉的了，在这里我们简单的介绍一下。在 asp.net 中，对这两个按钮，我们也用到了<input type= 这个标示，只是我们在后面加上 runat=”server“，表示

我们这个按钮在 asp.net 的框架之内，我们必须写方法来响应这个事件。如果没有 runat=”server“这个修饰，我们可以把这个控件当作普通的 html 按钮，不用写法响应事件，如：

```
<input type=reset value="重写" >
```

这是一个标准的 html 表示，我们在 asp.net 中的表示：

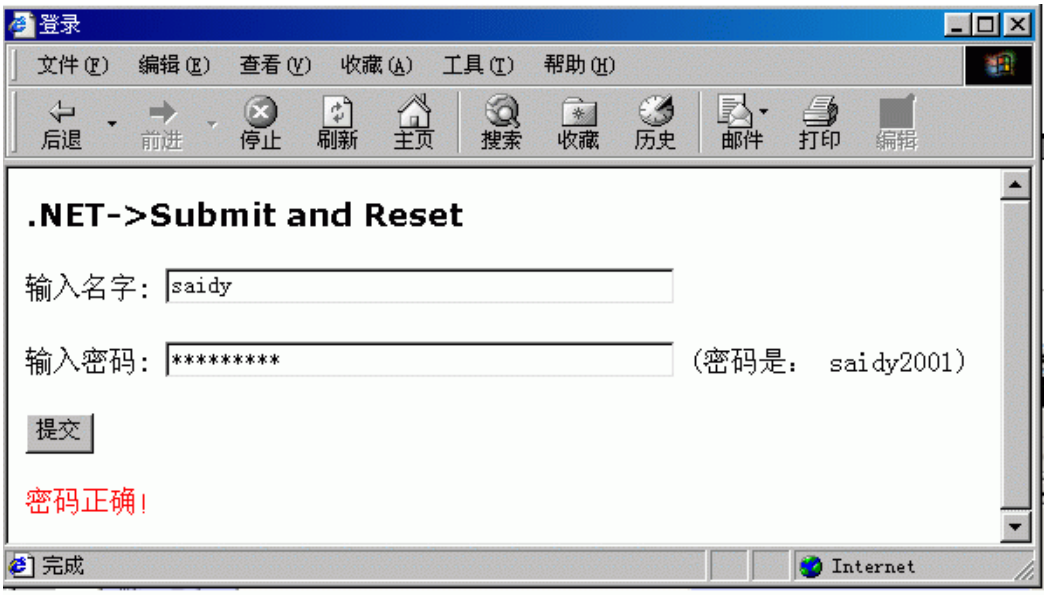
```
<input type=submit value="提交" OnServerClick="SubmitBtn_Click" runat=server>
```

我们在按钮按下时候响应事件 SubmitBtn_Click，具体如下**(button2.aspx)**：

```
<!--源文件： form\HtmlControl\button2.aspx-->
<html>
<head>
    <script language="VB" runat="server">
        Sub SubmitBtn_Click(sender As Object, e As EventArgs)
          If Password.Value = "saidy2001" Then
              Span1.InnerHtml = "密码正确！"
          Else
              Span1.InnerHtml="密码错误！"
          End If
        End Sub
        ' Sub ResetBtn_Click(sender As Object, e As EventArgs)
        '    Name.Value = ""
        '    Password.Value = ""
        'End Sub

    </script>
</head>
<BODY BGCOLOR="#CCCCFF">
    <h3><font face="Verdana">.NET->Submit and Reset</font></h3>
    <form runat=server>
        输入名字: <input id="Name" type=text size=40 runat=server>
        <p>
        输入密码: <input id="Password" type=password size=40 runat=server> (密码是：
                  saidy2001)
        <p>
        <input type=submit value="提交" OnServerClick="SubmitBtn_Click" runat=server>
        <input type=reset value="重写" OnServerClick="ResetBtn_Click" runat=server>
       <p>
        <span id="Span1" style="color:red" runat=server></span>
    </form>
</body>
</html>
```

看到如下的效果：

输入名字和提示的密码，如下：



2.4.8 小结

本章介绍了服务器端的 html 控件，虽然它们的功能都可以以简单的 html 语言来实现，但是在 asp.net 中依然提供了对它们的实现。以 html 语言书写和以服务器端控件的实现在思维方式上已经有了很大的不同，对于 html 语言而言，只是一种标识；而对服务器端 html 控件而言，却已演变成为一段程序，一个对象。两者的区别不仅仅是，一个后缀名为.html，另一个为.aspx。html 文件依赖于服务器端对标识的解释执行，html 控件却可以被编译执行，两者在效率上的差异不言而喻。

第三篇　数据库编程

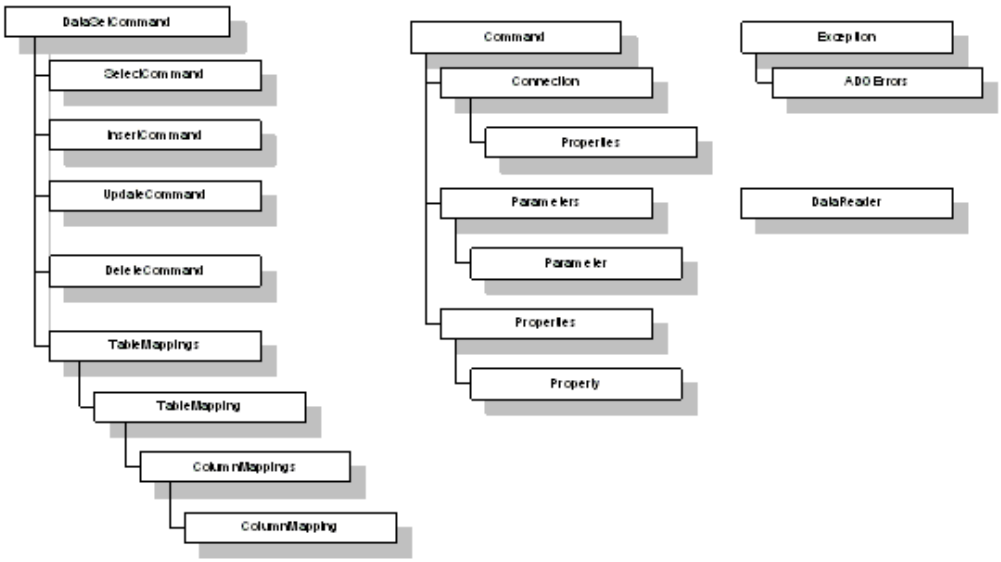
第一章　基本概念

ASP.NET 中的 ADO.NET 和 ASP 中的 ADO 相对应，它是 ADO 的改进版本。在 ADO.NET 中，通过 Managed Provider 所提供的应用程序编程接口(API)，可以轻松地访问各种数据源的数据，包括 OLEDB 所支持的和 ODBC 支持的数据库。

下面介绍 ADO.NET 中最重要的两个概念：Managed Provider 和 DataSet。

3.1.1 Managed Provider

过去，通过 ADO 的数据存取采用了两层的基于连接的编程模型。随着多层应用的需求不但增加，程序员需要一个无连接的模型。ADO.NET 就应运而生了。ADO.NET 的 Managed Provider 就是一个多层结构的无连接的一致的编程模型。



Managed Provider 提供了 DataSet 和数据中心（如 MS SQL）之间的联系。Managed Provider 包含了存取数据中心（数据库）的一系列接口。主要有三个部件：

- 连接对象 Connection、命令对象 Command、参数对象 Parameter 提供了数据源和 DataSet 之间的接口。DataSetCommand 接口定义了数据列和表映射，并最终取回一个 DataSet。
- 数据流提供了高性能的、前向的数据存取机制。通过 IdataReader，你可以轻松而高效地访问数据流。
- 更底层的对象允许你连接到数据库，然后执行数据库系统一级的特定命令。

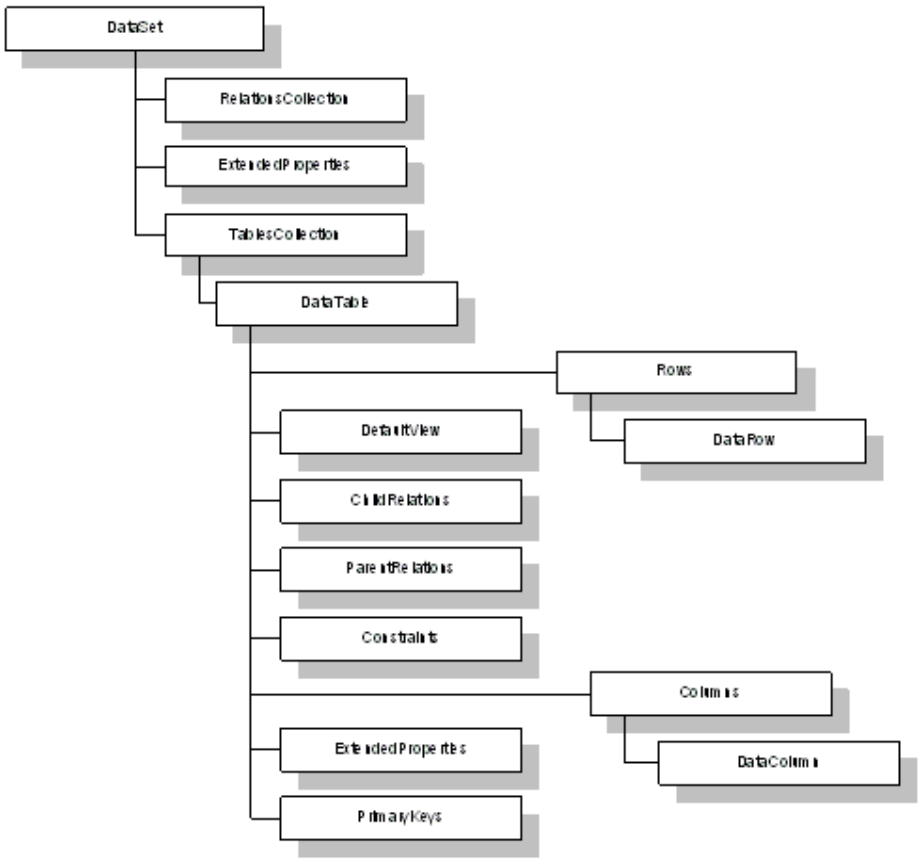
过去，数据处理主要依赖于两层结构，并且是基于连接的。连接断开，数据就不能再存取。现在，数据处理被延伸到三层以上的结构，相应地，程序员需要切换到无连接的应用模型。这样，DataSetCommand 就在 ADO.NET 中扮演了极其重要的角色。它可以取回一个 DataSet，并维护一个数据源和 DataSet 之间的“桥”，以便于数据访问和修改、保存。

DataSetCommand 自动将数据的各种操作变换到数据源相关的合适的 SQL 语句。从图上可以看出，四个 Command 对象：SelectCommand、InsertCommand、UpdateCommand、DeleteCommand 分别代替了数据库的查询、插入、更新、删除操作。

Managed Provider 利用本地的 OLEDB 通过 COM Interop 来实现数据存取。OLEDB 支持自动的和手动的事务处理。所以，Managed Provider 也提供了事务处理的能力。

3.1.2 DataSet

DataSet 是 ADO.NET 的中心概念。你可以把 DataSet 想象成内存中的数据库。正是由于 DataSet，才使得程序员在编程序时可以屏蔽数据库之间的差异，从而获得一致的编程模型：



DataSet 支持多表、表间关系、数据约束等等。这些和关系数据库的模型基本一致。

3.1.2.1 TablesCollection 对象

DataSet 里的表(Table)是用 DataTable 来表示的。DataSet 可以包含许多 DataTable，这些 DataTable 构成 TablesCollection 对象。

DataTable 定义在 System.Data 中，它代表内存中的一张表(Table)。它包含一个称为 ColumnsCollection 的对象，代表数据表的各个列的定义。DataTable 也包含一个 RowsCollection 对象，这个对象含有 DataTable 中的所有数据。

DataTable 保存有数据的状态。通过存取 DataTable 的当前状态，你可以知道数据是否被

更新或者删除。

3.1.2.2 RelationsCollection 对象

各个 DataTable 之间的关系通过 DataRelation 来表达，这些 DataRelation 形成一个集合，称为 RelationsCollection，它是 DataSet 的子对象。DataRelation 表达了数据表之间的主键-外键关系，当两个有这种关系的表之中的某一个表的记录指针移动时，另一个表的记录指针也随之移动。同时，一个有外键的表的记录更新时，如果不满足主键-外键约束，更新就会失败。

通过建立各个 DataTable 之间的 DataRelation，可以轻松实现在 ASP 中需要通过 DataShaping 才能实现的功能。

3.1.2.3 ExtendedProperties 对象

在这个对象里可以定义特定的信息，比如密码、更新时间等。

3.1.2.4 小结

本章首先介绍在 asp.net 中数据库编程的两个基本概念 Managed Provider 和 DataSet。在 asp.net 中，DataSet 屏蔽了具体数据源和应用之间差异，使得应用摆脱了具体数据的束缚。在我们今后的数据库编程中，可以把 DataSet 视为远端数据库在内存中的镜像，把繁琐的数据库操作任务交给 Managed Provider 去做。

第二章　通过 ADO.NET 访问数据库

3.2.1 ADO.NET 访问数据库的步骤

不论从语法来看，还是从风格和设计目标来看，ADO.NET 都和 ADO 有显著的不同。在 ASP 中通过 ADO 访问数据库，一般要通过以下四个步骤：

1、 创建一个到数据库的链路，即 ADO.Connection；
2、 查询一个数据集合，即执行 SQL，产生一个 Recordset；
3、 对数据集合进行需要的操作；
4、 关闭数据链路。

在 ADO.NET 里，这些步骤有很大的变化。ADO.NET 的最重要概念之一是 DataSet。DataSet 是不依赖于数据库的独立数据集合。所谓独立，就是：即使断开数据链路，或者关闭数据库，DataSet 依然是可用的。如果你在 ASP 里面使用过非连接记录集合(Connectionless Recordset)，那么 DataSet 就是这种技术的最彻底的替代品。

有了 DataSet，那么，ADO.NET 访问数据库的步骤就相应地改变了：

- 创建一个数据库链路；
- 请求一个记录集合；
- 把记录集合暂存到 DataSet；
- 如果需要，返回第 2 步；（DataSet 可以容纳多个数据集合）
- 关闭数据库链路；
- 在 DataSet 上作所需要的操作。

DataSet 在内部是用 XML 来描述数据的。由于 XML 是一种平台无关、语言无关的数据

描述语言，而且可以描述复杂数据关系的数据，比如父子关系的数据，所以 DataSet 实际上可以容纳具有复杂关系的数据，而且不再依赖于数据库链路。

3.2.2 ADO.NET 对象模型概览

3.2.2.1 ADOConnection

ADO.NET 有许多对象，我们先看看最基本的也最常用的几个。首先看看 ADOConnection。和 ADO 的 ADODB.Connection 对象相对应，ADOConnection 维护一个到数据库的链路。为了使用 ADO.NET 对象，我们需要引入两个 NameSpace：System.Data 和 System.Data.ADO，使用 ASP.NET 的 Import 指令就可以了：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>
```

和 ADO 的 Connection 对象类似，ADOConnection 对象也有 Open 和 Close 两个方法。下面的这个例子展示了如何连接到本地的 MS SQL Server 上的 Pubs 数据库。

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>
<%
  '设置连接串...
  Dim strConnString as String
  strConnString = "Provider=SQLOLEDB; Data Source=(local); " & _
                  "Initial Catalog=pubs; User ID=sa"

  '创建对象 ADOConnection
  Dim objConn as ADOConnection
  objConn = New ADOConnection

  '设置 ADOCOnnection 对象的连接串
  objConn.ConnectionString = strConnString

  objConn.Open()   '打开数据链路

  '数据库操作代码省略

  objConn.Close()      '关闭数据链路
  objConn = Nothing    '清除对象
%>
```

上面的代码和 ADO 没有什么太大的差别。应该提到的是，ADO.NET 提供了两种数据库连接方式：ADO 方式和 SQL 方式。这里我们是通过 ADO 方式连接到数据库。关于建立数据库连接的详细信息，我们在后面的篇幅中将会讲到。

3.2.2.2 ADODatasetCommand

另一个不得不提到的 ADO.NET 对象是 ADODatasetCommand，这个对象专门负责创建我们前面提到的 DataSet 对象。另一个重要的 ADO.NET 对象是 Dataview，它是 DataSet 的一个视图。还记得 DataSet 可以容纳各种各种关系的复杂数据吗？通过 Dataview，我们可以把 DataSet 的数据限制到某个特定的范围。

下面的代码展示了如何利用 ADODatasetCommand 为 DataSet 填充数据：

```
'创建 SQL 字符串
Dim strSQL as String = "SELECT * FROM authors"

'创建对象 ADODatasetCommand  和 Dataset
Dim objDSCommand as ADODatasetCommand
Dim objDataset as Dataset = New Dataset
objDSCommand = New ADODatasetCommand(strSQL, objConn)

'填充数据到 Dataset
'并将数据集合命名为  "Author Information"
objDSCommand.FillDataSet(objDataset, "Author Information")
```

3.2.3 显示 Dataset

前面我们已经把数据准备好。下面我们来看看如何显示 Dataset 中的数据。在 ASP.NET 中，显示 DataSet 的常用控件是 DataGrid，它是 ASP.NET 中的一个 HTML 控件，可以很好地表现为一个表格，表格的外观可以任意控制，甚至可以分页显示。这里我们只需要简单地使用它：

```
<asp:DataGrid id="DataGridName" runat="server"/>
```

剩下的任务就是把 Dataset 绑定到这个 DataGrid，绑定是 ASP.NET 的重要概念，我们将另文讲解。一般来说，你需要把一个 Dataview 绑定到 DataGrid，而不是直接绑定 Dataset。好在 Dataset 有一个缺省的 Dataview，下面我们就把它和 DataGrid 绑定：

```
MyFirstDataGrid.DataSource = _
        objDataset.Tables("Author Information").DefaultView
MyFirstDataGrid.DataBind()
```

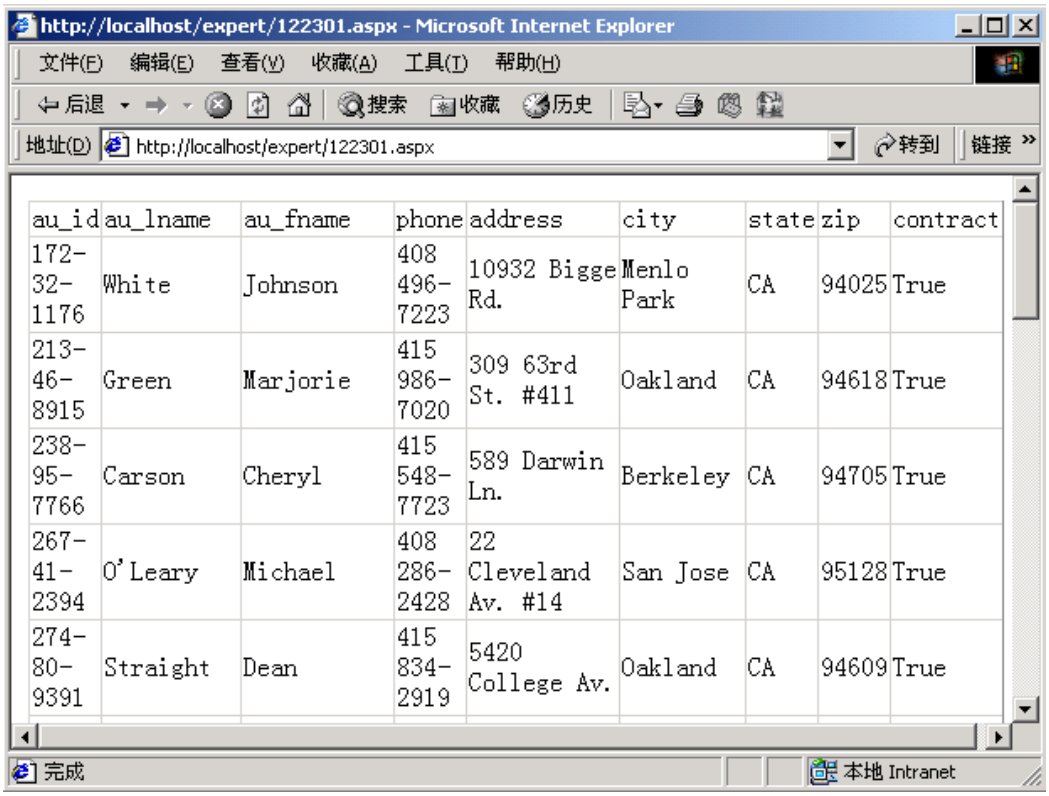
3.2.4 完整的代码

(code\122301.aspx)

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>
<%
```

```
   '设置连接串...
   Dim strConnString as String
   strConnString = "Provider=SQLOLEDB; Data Source=(local); " & _
                           "Initial Catalog=pubs; User ID=sa"

   '创建对象 ADOConnection
   Dim objConn as ADOConnection
   objConn = New ADOConnection

   '设置 ADOCOnnection 对象的连接串
   objConn.ConnectionString = strConnString

   objConn.Open()   '打开数据链路

   '创建 SQL 字符串
   Dim strSQL as String = "SELECT * FROM authors"

   '创建对象 ADODatasetCommand 和 Dataset
   Dim objDSCommand as ADODatasetCommand
   Dim objDataset as Dataset = New Dataset
   objDSCommand = New ADODatasetCommand(strSQL, objConn)

   '填充数据到 Dataset
   '并将数据集合命名为 "Author Information"
   objDSCommand.FillDataSet(objDataset, "Author Information")

   objConn.Close()      '关闭数据链路
   objConn = Nothing   '清除对象


   Authors.DataSource = _
            objDataset.Tables("Author Information").DefaultView
   Authors.DataBind()

%>

<HTML>
<BODY>
<asp:DataGrid id="Authors" runat="server"/>
</BODY>
</HTML>
```

3.2.5 运行效果



au_id	au_lname	au_fname	phone	address	city	state	zip	contract
172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	True
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	True
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	True
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	True
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	True

3.2.6 小结

本章详细介绍了如何使用 ADO.NET 方法访问数据库的步骤，并给出了一个具体的例子演示如何从服务器端取得 pubs 数据库中的 authors 表的数据到本地的 DataSet 中，然后使用 DataGrid 控件绑定到 DataSet 上，最后在客户端显示。虽然这还比较简单，但这却是最常用的技术。

第三章　ADO.NET 数据连接方法

3.3.1 数据库连接字符串

一个 Web 应用往往包括几十上百个 ASPX 文件。如果在每一个文件里都是直接构造这个数据库连接字符串，首先是觉得麻烦，其次，如果数据库发生了什么变化，比如密码变化，或者 IP 变化，难道你都要改动每一个 ASPX 文件？

在《轻松组建网上商店》第一版里，我们通过把数据库连接字符串封装到

Application(“strConn”)变量里面，在 global.asa 中初始化这个 Application 变量，从而解决了这个难题。

另外的一个解决方法就是写一个 DbOpen 函数，放到独立的一个 ASP 文件里，然后在其他的文件里包含这个 DbOpen 函数所在的文件。

这些方法在 ASP 时代非常流行。在 ASP.NET 时代，这些方法大部分依然有效，但是这里介绍的方法，却是利用了 ASP.NET 的特性。我们如果学习 ASP.NET，就一定要按照 ASP.NET 的风格来编写代码。这样会给你带来意想不到的性能提高。

和在 ASP 里面类似，ASP.NET 也有一个 Application 一级的配置文件，叫做 config.web。通过简单地配置 config.web，就可以解决数据库连接字符串问题：

(config.web)

```
<configuration>
        <appsettings>
                <add key="strConn" value="server=localhost;uid=sa;pwd=;Database=pubs"/>
        </appsettings>
</configuration>
```

在 aspx 页面里，我们可以这样获得数据库连接字符串：

```
Dim MyConnection As SQLConnection
Dim Config as HashTable

‘把 config.web 的 appsettings 全部读到临时对象中
Config = Context.GetConfig("appsettings")
‘Config 临时对象实际上是一个集合。
MyConnection = New SQLConnection(Config("MyConn"))
```

关于 config.web 的详细介绍，请阅读后面的章节。

此处要说明的是，本书为了使各个例子相对独立，没有采用上面介绍的方法。

3.3.2 两种数据库连接方式

ASP.NET 不仅带来了 ADO.NET，还带来了 SQL Managed Provider。这样在 ASP.NET 里，我们就有了两种连接数据库的方式：

1、ADO.NET Managed Provider

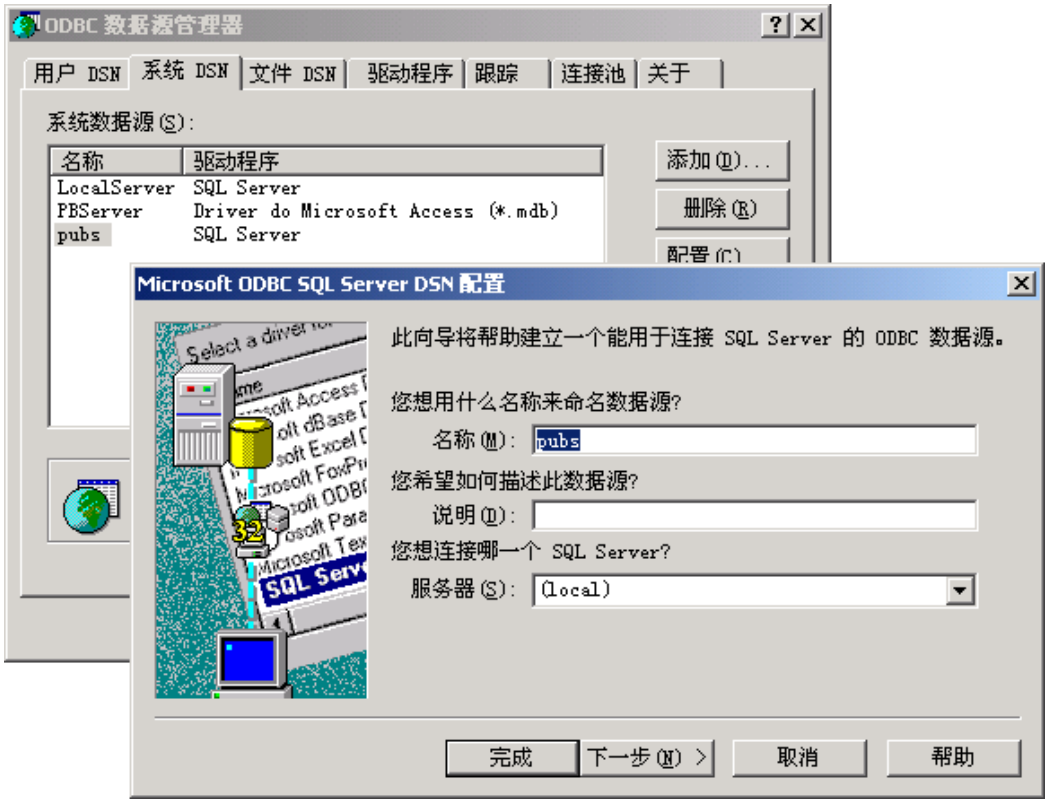
2、SQL Managed Provider

其中，方式一可以连接到任何 ODBC 或者 OLEDB 数据中心，而方式二可以连接到 MS SQL Server。仅仅就 MS SQL Server 来讲，使用方式二在性能上要优于方式一。

下面我们来看看数据库连接的各种情况。

3.3.2.1 ADO.NET Managed Provider 和 ODBC

我们要连接的数据库是 MS SQL Server 中的 pubs 数据库。首先我们创建一个 DSN：控制面板>>管理工具>>数据源(ODBC)>>添加：



下面的代码就创建了一个到 MS SQL Server 中 pubs 数据库的连接：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<script language=”VB” RunAt=”Server”>
…
   '创建对象 ADOConnection
   Dim objConn as ADOConnection=New ADOConnection(“DSN=pubs”)
   objConn.Open()   '打开数据链路
…
</script>
```

注意开始的两个 Import 语句。这是 ADO.NET 对象所在的 Namespace。

ADO.NET Managed Provider+ODBC 可以连接到各种数据源，包括：MS SQL Server、Access、Excel、mySQL、Oracle，甚至格式化的文本文件等等。

3.3.2.1.1 一个完整的例子

```
<%@ Page Language="vb" %>
<%@ Import Namespace = "System.Data" %>
<%@ Import Namespace = "System.Data.ADO" %>
<html>
```

```
<head>
<script runat=server>
    Sub Page_Load(ByVal Sender As Object, ByVal e As EventArgs)
        On Error Resume Next
        Dim cn          As ADOConnection

        cn = New ADOConnection("DSN=NWind")
        cn.Open()
        If cn.State = 1 Then
            lblReturnCode.Text = "The Connection State is: " & cn.State & " - Connection
Succeeded"
        Else
            lblReturnCode.Text = "The Connection State is: " & cn.State & " - Connection
Failed"
        End If
    End Sub
    </script>
</head>
<body>
    <asp:Label id="lblReturnCode" Runat=server />
</body>
</html>
```

3.3.2.2 ADO.NET Managed Provider 和 OLEDB

建立一个到 OLEDB 数据中心的连接，就需要精心构造数据库连接字符串。下面的代码建立了一个到 Access 数据库的连接：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<script language=”VB” RunAt=”Server”>
…
Dim cn As ADOConnection cn = New ADOConnection("provider=Microsoft.Jet.OLEDB.4.0; " &
_
  "Data Source=C:\Program Files\Microsoft Office\Office\Samples\Northwind.mdb;")
cn.Open()
…
</script>
```

下面的代码建立了到 MS SQL Server 数据库的连接：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<script language=”VB” RunAt=”Server”>
…
```

```
Dim cn As ADOConnection cn = New ADOConnection("Provider=SQLOLEDB.1;Data Source=(local);uid=sa;pwd=;Initial Catalog=pubs”)
cn.Open()
…
</script>
```

ADO.NET 目前支持下面的几个 OLEDB：

OLEDB 驱动程序	提供者
SQLOLEDB	SQL OLE DB Provider
MSDAORA	Oracle OLE DB Provider
JOLT	Jet OLE DB Provider
MSDASQL/SQLServer ODBC	SQL Server ODBC Driver via OLE DB for ODBC Provider
MSDASQL/Jet ODBC	Jet ODBC Driver via OLE DB Provider for ODBC Provider

(数 据 来 源 ： http://msdn.microsoft.com/library/default.asp?URL=/library/dotnet/cpguide/cpconaccessingdatawithado.htm)

3.3.2.2.1 一个完整的例子

```
<%@ Page Language="vb" %>
<%@ Import Namespace = "System.Data" %>
<%@ Import Namespace = "System.Data.ADO" %>
<html>
  <head>
  <script runat=server>
    Sub Page_Load(ByVal Sender As Object, ByVal e As EventArgs)
        On Error Resume Next
        Dim cn          As ADOConnection

        cn      =       New     ADOConnection("provider=Microsoft.Jet.OLEDB.4.0;      Data Source=C:\Program Files\Microsoft Office\Office\Samples\Northwind.mdb;")
        cn.Open()
        If cn.State = 1 Then
            lblReturnCode.Text = "The Connection State is: " & cn.State & " - Connection Succeeded"
        Else
            lblReturnCode.Text = "The Connection State is: " & cn.State & " - Connection Failed"
        End If
    End Sub
    </script>
```

```
  </head>
  <body>
    <asp:Label id="lblReturnCode" Runat=server />
  </body>
</html>
```

3.3.2.3 SQL Managed Provider 和 Microsoft SQL Server

通过 SQL Managed Provider 建立到 MS SQL Server 的连接很简单：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<script language=”VB” RunAt=”Server”>
…
  Dim objConn as SQLConnection = New ADOConnection("server=localhost;uid=sa;pwd=;database=pubs;")

  objConn.Open()  '打开数据链路
…
</script>
```

请注意几个地方：

1、 Import 语句的不同。在 ADO.NET Managed Provider 里面，我们 Import 的是 System.Data.ADO；而这里需要 System.Data.SQL；
2、 连接对象也不同。在 ADO.NET Managed Provider 中，所有的对象以 ADO 打头；而这里需要以 SQL 打头。

下面的表格归纳了这些不同：

	ADO.NET Managed Provider	ADO.NET SQL Managed Provider
需要引入的 Namespace	System.Data.ADO	System.Data.SQL
Connection 对象	ADOConnection	SQLConnection
Command 对象	ADODatasetCommand	SQLDatasetCommand
Dataset 对象	Dataset	Dataset
DataReader	ADODataReader	SQLDataReader
连接数据库例子	String sConnectionString = "Provider= SQLOLEDB.1; Data Source=localhost; uid=sa; pwd=; Initial Catalog=pubs"; ADOConnection con = new ADOConnection(sConnectionString); con.Open();	String sConnectionString = "server=localhost;uid=sa;pwd=;database=pubs"; SQLConnection con = new SQLConnection(sConnectionString); con.Open();
执行 SQL 语句例	ADOCommand cmd = new	SQLCommand cmd = new

子	ADOCommand("SELECT * FROM Authors", con); ADODataReader dr = new ADODataReader(); cmd.Execute(out dr);	SQLCommand(("SELECT * FROM Authors", con); SQLDataReader dr = new SQLDataReader(); cmd.Execute(out dr);

使用存储过程例子	ADOCommand cmd = new ADOCommand ("spGetAuthorByID", con);
	cmd.CommandType = CommandType.StoredProcedure;
	
	ADOParameter prmID = new ADOParameter("AuthID", ADODataType.VarChar, 11);
	
	prmID.Value = "111-11-1111";
	
	cmd.SelectCommand.Parameters.Add(prmID);
	
	ADODataReader dr;
	cmd.Execute (out dr);
		SQLCommand cmd = new SQLCommand("spGetAuthorByID", con);
		cmd.CommandType = CommandType.StoredProcedure;
		
		SQLParameter prmID = new SQLParameter("@AuthID", SQLDataType.VarChar,11);
		
		prmID.Value = "111-11-1111"
		cmd.SelectCommand.Parameters.Add(prmID);
		SQLDataReader dr;
		cmd.Execute(out dr);

3.3.2.3.1 一个完整的例子

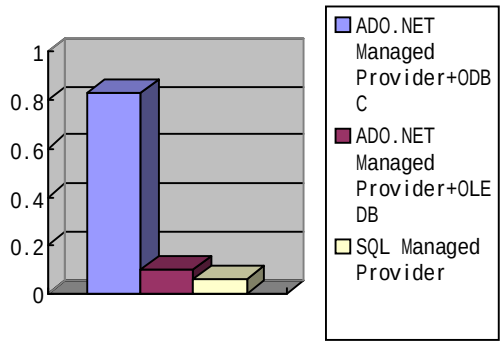
```
<%@ Page Language="vb" %>
<%@ Import Namespace = "System.Data" %>
<%@ Import Namespace = "System.Data.SQL" %>
<html>
  <head>
  <script runat=server>
    Sub Page_Load(ByVal Sender As Object, ByVal e As EventArgs)
        'On Error Resume Next
        Dim cn          As SQLConnection

        cn = New SQLConnection("server=localhost;uid=sa;pwd=;database=pubs;")
        cn.Open()
        If cn.State = 1 Then
            lblReturnCode.Text = "The Connection State is: " & cn.State & " - Connection Succeeded"
        Else
            lblReturnCode.Text = "The Connection State is: " & cn.State & " - Connection Failed"
        End If
    End Sub
```

```
    </script>
  </head>
  <body>
    <asp:Label id="lblReturnCode" Runat=server />
  </body>
</html>
```

3.3.2.4 三种方法的对比

一般来说，这三种存取数据库的方法中，SQL Managed Provider 效率最高，其次是 ADO.NET Managed Provider+OLEDB，最差的是 ADO.NET Managed Provider+ODBC。下面是在普通 PIII 微机上，对于 Access 2000 和 MS SQL Server 2000 上的测试结果：

数据库连接类型	页面显示所需时间(秒)
ADO.NET Managed Provider+ODBC	0.831195
ADO.NET Managed Provider+OLEDB	0.100144
SQL Managed Provider	0.060086



从图上可以看出，SQL Managed Provider 要优于 ADO.NET Managed Provider，而从 ODBC 和 OLEDB 的对比来看，OLEDB 要优于 ODBC。

下面列示了测试用的源程序，仅供参考。

3.3.2.5 测试程序

(122303.aspx)

```
<%@ Page EnableSessionState="False" %>

<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>
<%@ Import Namespace="System.Data.SQL" %>


<script language="VB" runat="server">
Sub Refresh(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Dim d1,d2 As DateTime

    Dim strConn
    if Page.IsValid then
        d1=Now()
```

```
        Dim iChoice As Integer=CInt(Choices.SelectedItem.Value)
        select case iChoice
            case 1
                strConn="DSN=pubs;"
                ADOBindData(strConn)
            case 2
                strConn="Provider=SQLOLEDB.1;Data    Source=(local);uid=sa;pwd=;Initial
Catalog=pubs"
                ADOBindData(strConn)
            case 3
                strConn="server=localhost;uid=sa;pwd=;Database=pubs"
                '"server=localhost;uid=sa;pwd=;database=northwind;"
                SQLBindData(strConn)
            Case Else
        end select


        d2=Now()
        result.Text = "用时(Ticks)： "&d2.Ticks-d1.Ticks
    end if
End Sub

Sub ADOBindData(strConn)
  '设置连接串...
  '创建对象 ADOConnection
  Dim objConn as ADOConnection = New ADOConnection(strConn)

  objConn.Open()   '打开数据链路

  '创建 SQL 字符串
  Dim strSQL as String = "SELECT * FROM authors"

  '创建对象 ADODatasetCommand 和 Dataset
  Dim objDSCommand as ADODatasetCommand
  Dim objDataset as Dataset = New Dataset
  objDSCommand = New ADODatasetCommand(strSQL, objConn)

  '填充数据到 Dataset
  '并将数据集合命名为 "Author Information"
  objDSCommand.FillDataSet(objDataset, "Author Information")

  objConn.Close()      '关闭数据链路
  objConn = Nothing   '清除对象
```

```
    Authors.DataSource = _
                objDataset.Tables("Author Information").DefaultView
    Authors.DataBind()
End Sub


Sub SQLBindData(strConn)
    '设置连接串...
    '创建对象 ADOConnection
    Dim objConn as SQLConnection = New SQLConnection(strConn)

    objConn.Open()    '打开数据链路

    '创建 SQL 字符串
    Dim strSQL as String = "SELECT * FROM authors"

    '创建对象 SQLDatasetCommand 和 Dataset
    Dim objDSCommand as SQLDatasetCommand
    Dim objDataset as Dataset = New Dataset
    objDSCommand = New SQLDatasetCommand(strSQL, objConn)

    '填充数据到 Dataset
    '并将数据集合命名为 "Author Information"
    objDSCommand.FillDataSet(objDataset, "Author Information")

    objConn.Close()       '关闭数据链路
    objConn = Nothing    '清除对象


    Authors.DataSource = _
                objDataset.Tables("Author Information").DefaultView
    Authors.DataBind()
End Sub

</script>

<HTML>
<BODY>
<H2>测试设置</H2>
        <Form Action="122303.aspx" Method="Post" RunAt="Server">
        <asp:RadioButtonList ID="choices" RunAt="Server">
            <asp:ListItem selected Text="ADO.NET Managed Provider+ODBC" Value=1/><br>
```

```
        <asp:ListItem Text="ADO.NET Managed Provider+OLEDB"    Value=2/><br>
        <asp:ListItem Text="SQL Managed Provider" Value=3/>
    </asp:RadioButtonList>
    <br>
    <asp:LinkButton runat="server" OnClick="Refresh" Text="开始测试"/>
    <br>
<H2>测试结果</H2>
    <asp:label id="result" RunAt="Server" Text="No result"/>
    <br>
<H2>测试数据</H2>
    <asp:DataGrid id="Authors" runat="server"/>
    </Form>
</BODY>
</HTML>
```

3.3.3 使用 DataSets

使用 DataSets 有两种方式，一是从数据库中得到，一是自己编程动态创建一个 DataSets。使用从数据库端得到的 DataSets 方式主要是为了方便用户在客户端操作修改远端的数据库管理系统中的相应信息。而使用编程创建 DataSets，是由于 DataSets 的数据事先并不知道，需要在程序运行中得到数据并填充进 DataSets。采用 DataSets 作为本地数据来源中心的好处是，应用逻辑一样的程序就与数据来源不同分开，当数据源发生变化时，就只需要修改填充 DataSets 的程序而不用修改应用程序。

3.3.3.1 从数据库得到 DataSets 的使用

使用一个从数据库获得的 DataSets 较为复杂，它的步骤大概如下：

1. 使用 SQLDataSetCommand 命令(SQL 方式)或者 ADODataSetCommand 命令(ADO 方式)从数据库管理系统中获取一个表结构及其数据填充到本地内存的 DataSet 的一个表中。

   例如：
   ·Ado 方式

```
Dim MyDsComm As New ADODataSetCommand
Dim MyComm As ADOCommand
Dim MyConn As ADOConnection

MyConn = New ADOConnection _
("Provider=SQLOLEDB.1;Initial Catalog=Northwind;" & _
"Data Source=(local);User ID=sa;")

MyComm = New ADOCommand("SELECT * FROM Customers", MyConn)
```

```
MyDsComm.SelectCommand = MyComm

·SQL 方式
Dim MyConn as SQLConnection
Dim MyComm as SQLDataSetCommand
Dim MyDs as New DataSet

MyConn=New SQLConnection("server=localhost;uid=sa;pwd=;database=pubs")
MyComm=New SQLDataSetCommand("Select * from authoers",MyConn)
MyComm.FillDataSet(Myds,"authers")
```

2. 对 DataSet 中的表对象 DataTable 的数据进行操作，包括增加、删除、修改它的 DataRow 对象

3．使用 GetChanges 方法产生一个 DataSet 修改后的对象的 DataSet 集合。

代码如下：

```
Dim changedDataSet As DataSet
changedDataSet = ds.GetChanges(DataRowState.Modified)
```

4．通过对产生的 DataSet 对象的 HasErrors 属性的监控，查看是否 DataSet 中的表有错误发生。

5．如果有错误发生，就要对 DataSet 中的各个表进行错误检查，方法一样，也是根据各个 DataTable 的 HasErrors 属性。如果表中有错误发生，那么 GetErrors 方法就会被激活，并且会返回一个含有错误的 DataRow 对象的数组。

6．当表出错时，对于每一个 DataRow 对象的 RowError 属性进行检测。

7．如果可能，处理发生的错误。

8．使用 DataSet 对象的 Merge 方法把检测无错误发生的修改后的 DataSet 合并入原先的 DataSet 中，代码如下：

```
ds.Merge(changedDataSet)
```

9．使用 DataSetCommand 对象的 update 方法，把合并后的 DataSet 对象送往数据库端进行修改，代码如下：

```
MyDataSetCommand.Update(ds)
```

10．使用 DataSet 对象的 AcceptChanges 方法对数据库修改进行确认，或者使用 RejectChanges 方法撤消对数据库的修改,代码如下：

```
Ds.AcceptChanges
```

3.3.3.2 编程实现 DataSet

1．使用 DataSet()创建器创立一个 DataSet 对象。

DataSet()可以跟一个字符串用以指明创建的 DataSet 名字

```
Dim ds1 as New DataSet
Dim ds2 as New DataSet(“MyDataSet”)
```

2．增加一个 DataTable 到 DataSet 中。

具体操作是，首先在 DataSet 对象的 Tables 集合中，增加一个表

ds.Tables.Add(New DataTable(表名))

然后，再在该表中的 Columns 集合中增加相应的列

ds.Tables(表名).Columns.Add(列名，列类型)

例如：我们在 ds 对象中建立一个订购表(Order)，它有三个字段，客户名(CUNM)、订货编号（ORNO）、订货数量（ORNM）

```
ds.Tables.Add(New DataTable(“Order”))

ds.Tables(“Order”).Columns.Add(“CUNM”,GetType(String))
ds.Tables(“Order”).Columns.Add(“ORNO”,GetType(String))
ds.Tables(“Order”).Columns.Add(“ORNM”,GetType(int32))

ds.Tables(“Order”).PrimaryKey=
                    New DataColumn(ds.Tables(“Order”).Columns(“ORNO”)
```

在上面的例子中，我们还设置了订购表的键值，这是通过对它的 PrimaryKey 的属性设置来得到的，设置键值的好处在于可以防止相同记录的输入，保证数据的唯一性。

3．设置表间的关系

由于 DataSet 对象中可以含有多个 DataTable，而事实上每一个表又不可能与其他的表没有任何的关系，这样就带来了一个问题，我们如何描述两个不同的表之间的关系？asp.net 提供了 DataRelation 对象来描述表和表之间的关系。DataRelation 对象至少需要两个参数才能确定两个表之间的关系，这是因为在两个表的关系中，至少需要一个主键列和一个外键列，才能确定两者之间的对应关系。

例如：关于客户购物有两个表，一个是客户信息表（Customer），一个是购物信息表(Order)，很显然它们两者之间存在着某种联系。经过分析，我们发现客户编号（CUNO）在两个表中都存在，它使我们能够把两个表的信息连接起来，告诉我们这样一个事实，谁订购了什么物品。因此需要建立关于客户信息表和购物信息表的一个联系，用 Asp.net 语言表达如下：

```
Dim ds as DataSet
…
ds.Relations.Add(“CustomerOrder”,ds.Tables(“Customer”).Columns(“CUNO”),
```

ds.Tables(“Order”).Columns(“CUNO”))

4．在关系表间的浏览

通过 DataRelaiton 的设置，我们可以在同一个 DataSet 中，由对一个表操作，找到可能引起的相关表的变化。例如，对于客户信息表中的对应于某个人的一条记录，我们可以在购货信息表中找到所有属于他的购货信息，演示代码如下：

```
dim orderRows() as DataRow
    orderRows=ds.Tables(“Customer”).ChildRelations(“CustomerOrder”).GetChildRows(
                                    ds.Tables(“Customer”).Rows(0))
```

5．数据约束的使用

在关系数据库中，使用数据约束的目的是为了使数据库的一致性得到保证。当数据发生改变时，数据约束被执行，用以检查对数据的修改，是否和已经定义的规则相符合，如果不符合修改将不能生效。在 asp.net 中提供了两种数据约束，ForeignKeyConstraint 和 UniqueConstraint。

ForeignKeyConstraint，外键值一致性约束，定义当表中的一条记录被删除或者是增加一条记录时，与该表相关的其他表的相应记录如何处理。例如，当一个客户被人从客户信息表中删去，那么在购物信息表中的关于他的购物信息的记录如何处理等等。

ForeignKeyConstraint 有五个可能的值如下：

- Cascade　当表中记录被删除或者更新以后，对应表中的记录相应被删除和更新
- SetNull　当表中记录被删除或者更新以后，对应表中的记录被置为 Null
- SetDefault　当表中记录被删除或者更新以后，对应表中的记录被置为缺省值
- None　当表中记录被删除或者更新以后，对应表中的记录不做任何处理
- Default　当表中记录被删除或者更新以后，ForeignKeyConstraint 采用其缺省值，通常该值为 Cascade

具体使用 ForeigKeyConstraint 时，首先应创建它，然后设置 DeleteRule 和 UpDateRule 属性，指明当删除和更新记录时，对应表的处理规则。

例子：我们对客户信息表定义一个外键定义，它定义当客户信息表中记录删除时，其关联表—购物信息表中的数据也应删除（意味着用户不存在，自然也不应该有他的购物信息），当客户信息表中记录被修改时，购物信息置为缺省的特殊值（意味着，当销售人员发生差错，记错购物用户，那么以他的名义购物的定单不应算在该用户头上，置为特殊标记，以供今后修改），代码演示如下：

```
dim fk as New ForeignKeyConstraint(ds.Tables(“Customer”).Columns(“CUNO”),
                                    ds.Tables(“Order”.Columns(“CUNO”))
‘创建外键约束为 Customer 表和 Order 表中 CUNO 字段
fk.DeleteRule=Cascade
fk.UpdateRule=SetDefault
‘删除规则为 Cascade，修改规则为 SetDefault
ds.Tables(“Customer”).Constraints.Add(fk)
‘加入 Customer 表的一致性约束集合中
```

UniqueConstraint，唯一性约束，它指定了数据表中的一个列或者几个列的集合的值的唯一性，通常被指定为唯一约束的字段都是表的键值。

例如：对于客户信息表，因为每个人的购物都必须和别人区别，这样才能保证正确地付款和发送货物，因而每一个人的客户编号都不应该相同，这时就可以使用唯一性约束来保证客户信息表中的客户编号唯一。演示代码如下：

```
dim uc as UniqueConstraint
uc=New UniqueConstraint(ds.Tables(“Customer”).Columns(“CUNO”))
‘指定唯一约束为 Customer 表中的 CUNO 字段
ds.Tables(“Customer”).Constraints.Add(uc)
‘把唯一约束加入 Customer 表的约束中
```

6．处理 DataSet 的事件

为了便于用户对 DataSet 的控制，asp.net 提供了 DataSet 的一系列可被用户处理的事件，它们包括：

- PropertyChange　当属性发生改变时
- MergeFailed　DataSet 合并失败时
- RemoveTable　删除一个表时
- RemoveRelation　删除一个关系时
- Adding the event handler to the event　增加一个事件处理函数时

例如：

```
ds.AddOnPropertyChange(new System.ComponentModel.PropertyChangeEventHandler _
(AddressOf me.DataSetPropertyChange))
‘指定当 DataSet 发生 PropertyChange 事件时的消息处理函数为 DataSetPropertyChange
ds.AddOnMergeFailed(new System.Data.MergeFailedEventHandler _
(AddressOf me.DataSetMergeFailed))
‘指定当 DataSet 发生 MergeFailed 事件时的消息处理函数为 DataSetMergeFailed

‘当 PropertyChange 发生时的处理函数
Private Sub DataSetPropertyChange _
    (ByVal sender As Object, ByVal e As System.PropertyChangeEventArgs)
    …
End Sub

‘当 MergeFailed 发生时的处理函数
Private Sub DataSetMergeFailed _
    (ByVal sender As Object, ByVal e As System.Data.MergeFaileedEventArgs)
   …
End Sub
```

3.3.3.3 使用 DataTable

DataTable 是 DataSet 中一个对象，它与数据库表的概念基本一致，简单起见，你就可以

把它认成是数据库 DataSet 中的表。

1. 创建一个 DataTable

DataTable 创建器跟使用 DataSets 创建器差不多，可以跟一个参数，用以指定表名。

例如：

```
dim MyTable as DataTable

MyTable = New DataTable("Test")
MyTable.CaseSensitive = False
MyTable.MinimumCapacity = 100
```

其中 CaseSensitive 属性指定是否区分大小写，这里指定不区分，CaseSensitive 属性是否打开对于查找、排序、过滤等操作有很大的影响。MinimumCapacity 属性指定创建时保留给数据表的最小记录空间。另外还有一个 TableName 的属性，它指定数据表的名称，例如下面两种方式创建的表是一样的。

1） dim MyTable as DataTable

```
MyTable=New DataTable(“test”)
```

2） dim MyTable as New DataTable

```
MyTable.TableName=”test”
```

2. 创建表列

一个 DataTable 又含有一个表列（Columns）的集合。表列的集合形成了表的数据结构，就如同数据库概念中，字段对应于表一样。我们可以使用 Columns 集合的 Add 方法向表中添加表列。该方法带有两个参数，一个是表列名，一个是该列的数据类型。由于我们通常在定义表列时，是使用.net 构架中的数据类型，而非数据库的数据类型，所以需要使用 GetType 方法把.net 架构的数据类型转换成数据库中的数据类型。

例如：我们建立一个客户信息表（Customer），它含有三个字段：

用户姓名（CUNM） 字符型

客户编号（CUNO） 字符型

用户序号（IDNO） 整型

```
Dim MyTable as DataTable
Dim MyColumn as DataColumn

MyTable = new DataTable("Customer")

MyColumn = MyTable.Columns.Add("CUNM",GetType("String") )
MyColumn = MyTable.Columns.Add("CUNO",GetType("String") )
```

```
MyColumn = MyTable.Columns.Add("IDNO",GetType("int32") )
```

3. 创建表达式列

asp.net 甚至允许创建一些依赖于其他表达式的表列，这样做的好处是，体现了表列之间的某种自然的联系。

要创建表达式表列，首先要指定表列的 DataType 属性，它表明了表达式运算结果的数据类型；然后设置表列的 Expression 属性为所需的表达式。

例如：一个很明显的例子是利息税，它为总金额＊税率＊0.2。在同一表中总金额为 total 列，税率为 rate 列，利息税为 tax 列。它们的关系如下：

```
Dim tax As DataColumn = New DataColumn

tax.DataType = GetType("Currency")
tax.Expression = "total *rate*0.20"
```

也可以：

```
MyTable.Columns.Add(“tax”,GetType(“Currency”),”total*rate*0.20)
```

4. 使用自增列

在一些数据库中，我们会发现有这样一种数据类型，通常称作系统序号，当我们向表中增加一条记录时，该字段会自动累加，以后我们可以通过这一唯一序号来标识每一条记录。在 asp.net 中，同样也可以实现类似的功能，这就是自增表列的使用。

定义自增表列实际上是对 DataColumn 对象的三个属性：AutoIncrement、AutoIncrementSeed、AutoIncrementStep 的使用。

AutoIncrement 属性，指定是否打开自增功能。

AutoIncrementSeed 属性，指定自增的起始值。

AutoIncrementStep 属性，指定自增的步长。

例如：

```
dim MyTable as New DataTable
dim MyColumn as DataColumn

MyColumn=MyTable.Columns.Add(“Sqno”,GetType(“int32”))
MyColumn.AutoIncrement=True
‘打开自增功能
MyColumn.AutoIncrementSeed=0
‘自增从 0 值起始
MyColumn.AutoIncrementStep=2
‘每次增长 2
```

5. 建立主键值

通常在一个表中，我们会定义一个主键，它能够唯一标识该表中的每一条记录。主键可

以为表中的一个表列，也可以为几个表列的集合。主键不能为空，而且不能重复，我们可以用 DataColumn 的两个属性 AllowNull 和 Unique 来实现（DataColumn1.AllowNull=False DataColumn1.Unique=True）。最后 DataTable 对象的 PrimaryKey 属性指定主键。

例如：

```
    dim MyColumn as DataColumn
dim MyTable as DataTable
…
MyColumn=MyTable.Columns(“CUNO”)
MyColumn.AllowNull=False
MyColumn.Unique=True
MyTable.PrimaryKey=MyColumn
```

当键值为几个表列的集合时：

```
dim MyColumn as DataColumn()

MyColumn(0)=MyTable.Columns(“col1”)
MyColumn(1)=MyTable.Columns(“col2”)
…
MyTable.PrimaryKey=MyColumn
```

3.3.3.4 数据的载入

l 向表中加入数据

当一个表结构已经创建好以后，剩下的问题就是如何把数据载入我们已经建立好的表中。通常采用的方法是，先创建一个 DataRow 对象，它类似于数据库概念中的记录，然后对 DataRow 的 Columns 集合进行赋值，最后把 DataRow 对象加入到 DataTable 的 DataRows 集合中，就相当于在表中插入一条记录。

例如：如下一个表 MyTable 中有两个列 Sqno 和 Name，Sqno 为序号，Name 设为”MyName”+序号，我们利用一个循环产生 n 条记录到 MyTable 中

```
Dim i as integer
Dim n as integer
Dim MyRow as DataRow
 …
  For i = 0 to n
    MyRow = MyTable.NewRow()
     ‘产生一条新记录
    MyRow("Sqno") = I
  ‘对 sqno 字段赋值
    MyRow("Name") = "MyName" & i.ToString()
  ‘对 name 字段赋值
    MyTable.Rows.Add(MyRow)
```

```
        ‘加入记录到表中
    Next
…
```

删除表中记录

DataTable 的 Rows 集合提供了两种方法从一个数据表中删除一条记录，它们是 Remove 方法和 Delete 方法。示例如下：

删除 MyTable 中的第三条记录：

MyTable.Rows.Remove(3)或者

MyTable.Rows(3).Delete

Delete 方法和 Remove 方法的区别不仅仅是方法的使用形式上。当调用 Remove 方法后，那么指定的 DataRow 就会从 Rows 集合中被删除。而 Delete 方法调用时，指定的 DataRow 并不真正从 Rows 集合中删除，只是作了一个删除标记，直到 DataSet 对象调用 AcceptChanges 方法的时候，才真正被删除；如果是 RejectChanges 方法被调用，那么 Delete 方法删除的 DataRow 对象将被恢复。

使用表中的数据

对于 DataTable 中的每一个 Row,它都可能有三种状态：Original、Current、Preposed。Original 状态是指当数据第一次被加入到数据表中时候的状态。Current 态指经过多次改变 Original 数据后得到的 Row。Preposed 态存在于一个相当短暂的时期，它是由 original 态过渡到 Current 态时的中间状态，一个明显的例子是对数据进行修改而尚未完成时，开始是 Original 态，完成后是 Current 态，而这之间就是 Preposed 态。

为了说明对表中数据的使用，我们来看下面一个例子,它是对 workTable 按每一个字段进行遍历，并把字段名和内容显示出来。

```
Dim CurrRows() As DataRow = workTable.Select(Nothing, Nothing, _
                            System.Data.DataViewRowState.CurrentRows)
 ‘对 workTable 数据集合选择有效的 DataRows 放入 CurrRows 数组
Dim list As String = System.String.Empty
‘清空 list 字符串
Dim RowNo As Integer
Dim ColNo As Integer

For RowNo = 0 to CurrRows.Count – 1
‘每一条记录的循环

    For ColNo = 0 To workTable.Columns.Count  –  1
     ‘一条记录中每一个字段的循环
       list = ""
       list &= workTable.Columns(colNo).ColumnName & " = " & _
           CurrRows(RowNo)(ColNo).ToString
```

```
        Console.WriteLine(list)
    Next
Next

If CurrRows.Count < 1 Then Console.WriteLine("No CurrentRows Found")
```

从上面的例子我们可以看出，对 Rows 集合使用 DataTable 的 Select 方法可以找出有效的 Rows 集合，然后根据 Rows.count 和 columns.count 可以确定有效的记录数和字段数，最后利用记录索引值和列索引值可以唯一确定数据表中的任何数据。

3.3.4 DataReader 的使用方法

3.3.4.1Read 方法

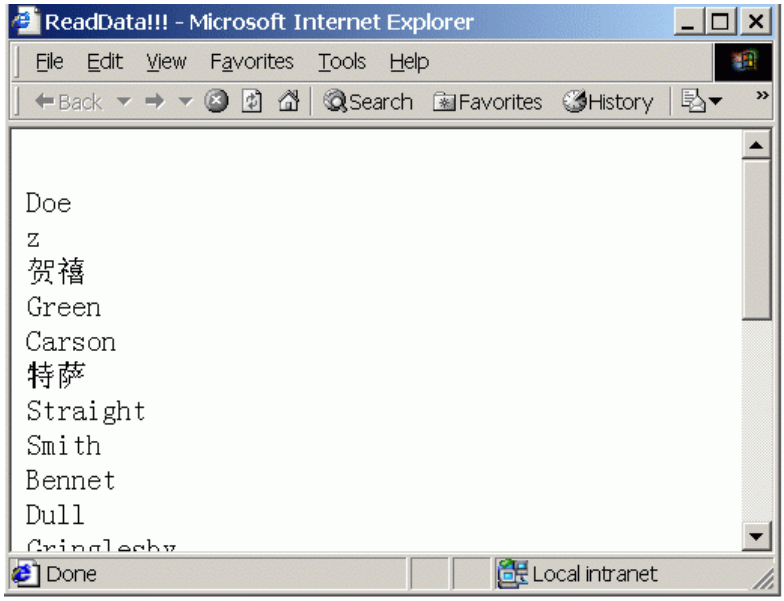
熟悉 Recordset 的读者一定在想，如何对一个 Dataset 中的每一个记录进行操作呢？这就不得不提到 ADO.NET 的 DataReader 对象了。我们来看看下面的例子：

```
<%@import namespace="system.data.SQL"%>
<script language="vb" runat="server">
Sub Page_Load(sender As Object, e As EventArgs)
    Dim sqlText as String = "select * from authors"
    Dim cnString as string = "server=localhost;uid=sa;pwd=;database=pubs;"
    Dim dbRead AS SQLDataReader
    Dim sqlCmd AS SQLCommand

    sqlCmd = New SQLCommand(sqlText,cnString)
    sqlCmd.ActiveConnection.Open()

    sqlCmd.execute(dbread)

    while dbRead.Read()
        response.write("<br>" & dbRead.Item("au_lname"))
    End while

End Sub
</script>
```

还记得 Recordset 的 MoveNext 和 Recordset 的 EOF 吗？为了对 Recordset 的每一条记录进行操作，我们需要首先调用其 MoveNext 方法移动记录，然后判断是否已经到了记录集合的末尾。

在 ADO.NET 中，一切变得更加简单了。DataReder 的 Read 方法自动移动记录到下一条，并返回移动是否成功的信息。相信使用过 Iterator 对象的读者一定很喜欢这样的操作方法吧。

另外，上面的代码中，我们并没有显式创建 SQLConnection 对象。我们把连接字符串传递给 SQLCommand，通过 sqlCmd.ActiveConnection.Open()创建了到数据库的链路。
上面代码的执行结果如下图：



3.3.4.2 更复杂的 Read

下面代码演示了利用 DataReader 生成两个下拉列表的的情况。

```
<%@import namespace="system.data.SQL"%>

<SCRIPT LANGUAGE="vb" RUNAT="server">

Sub Page_Load(myList AS Object,E as EventArgs)
If Not Page.IsPostBack()
    Dim dbRead AS SQLDataReader
    Dim dbComm AS SQLCommand
    Dim strConn AS String
    Dim SQL AS String

    strConn = "server=sql.database.com;uid=fooman;password=foopwd;"

    SQL = "Select * from Color ORDER BY Color"
    dbComm = New SQLCommand(SQL,strConn)
    dbComm.ActiveConnection.Open()
    dbComm.execute(dbRead)

    While dbRead.Read()
        ShirtColorOptions.items.add(New ListItem(dbRead.Item("Color")))
    End While
```

```
    SQL = "Select * from Size ORDER BY Size"
    dbComm = New SQLCommand(SQL,strConn)
    dbComm.ActiveConnection.Open()
    dbComm.execute(dbRead)

    While dbRead.Read()
        ShirtSizeOptions.items.add(New ListItem(dbRead.Item("Size")))
    End While
End IF

End Sub
</SCRIPT>

<FORM RUNAT="server" method="get">
    <asp:DropDownList id="shirtColorOptions" runat="server"DataTextField = "URL"/>
    <asp:DropDownList id="shirtSizeOptions" runat="server" DataTextField = "Size"/>
</FORM>
```

执行结果：

3.3.4.3 把 DataReader 绑定到 DataGrid

下面的例子是基于 NET Framework Beta2 的。

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
<%
Dim myConn As SQLConnection = new
                    SQLConnection("server=localhost;uid=sa;pwd=;database=NorthWind;")
Dim myCommand As SQLCommand = new SQLCommand("select * from cusotmers",myConn)

myConn.Open()

cusotmers.DataSource = myCommand.Execute()
cusotmers.DataBind()

myConn.Close()
%>

<HTML>
<BODY>
<asp:DataGrid id="cusotmers" runat="server"/>
</BODY>
```

```
</HTML>
```

3.3.4.4 利用 DataReader 插入记录

```
<SCRIPT LANGUAGE="vb" RUNAT="server">

Sub showDb()

<%@import namespace="system.data.SQL"%>


Dim dbRead AS SQLDataReader
Dim dbComm AS SQLCommand
Dim strConn AS String
Dim strSQL AS String
strConn= "server=my.sql.database;uid=fooname;password=foofoo;"
strSQL = "INSERT INTO myDatabase (dbValue1) VALUES('the thing')"

dbComm = New SQLCommand(strSQL, strConn)
dbComm.ActiveConnection.Open()
dbComm.execute(dbRead)


End Sub
</SCRIPT>
```

3.3.5 小结

本章对在 ADO.NET 中如何和远端数据库相连作了进一步的阐述。和数据库相连，ADO.NET 提供了三种方式，1）通过 ODBC 相连 2）通过 OLEDB 相连 3）直接和 SQL Server 相连。三种方式由于应用层次的差异，效率由低到高，独立性由高到低。对于相连数据库的数据处理，也有两种方式。一种是通过 DataSet 来隔离异构的数据源，另一种是以流方式从数据源读取（DataReader 方式）。

第四章　ADO.NET 数据库基本操作

3.4.1 插入记录

怎么向数据库插入记录？下面归纳了几种方法。

3.4.1.1 通过 SQLCommand

插入一条记录到数据库的 SQL 语句格式为：

Insert Into 表名 (字段 1，字段 2，。。。) Values (值 1，值 2，。。。)

我们知道，SQLCommand 可以执行 SQL 命令，我们把插入记录的 SQL 语句传递到 SQLCommand 的 CommandText 属性，然后执行其 ExecuteNonQuery 方法，就可以了。

```
Dim strConn As String
Dim strComm As String
Dim oConn As SQLConnection
Dim oComm As SQLCommand

strConn = "server=localhost;uid=sa;pwd=;database=northwind"
strComm = "INSERT INTO CUSTOMERS (CustomerId, CompanyName, Contactname, ContactTitle, Address) Values ('DarkMan','Sino-ASP.COM', 'Mr. Li', 'CTO','不要找我')"

oConn = new SQLConnection(strConn)
oComm = new SQLCommand(strComm, oConn)

Try
    oConn.Open()
    ‘执行命令
    oComm.ExecuteNonQuery()

Catch myException as Exception
    //出错处理
Finally
    oConn.Close()
End Try
```

3.4.1.2 通过 SQLDataSetCommand

如果需要批量插入记录，可以使用 SQLDataSetCommand。下面是一个示例：

```
Dim strConn,strComm As String

//数据库连接字符串
strConn="server=localhost;uid=sa;pwd=;database=northwind"
//查询语句
strComm="select * from region"

Dim oConn As New SQLConnection(strConn)
```

```
Dim oDSComm As New SQLDataSetCommand(strComm,oConn)
Dim oParam As SQLParameter

oDSComm.InsertCommand = new SQLCommand("Insert into Region (RegionID,
                RegionDescription) VALUES (@RegionID, @RegionDescription)", oConn)

oParam = oDSComm.InsertCommand.Parameters.Add(new SQLParameter("@RegionID",
SQLDataType.Int))
oParam.SourceVersion = DataRowVersion.Current
oParam.SourceColumn = "RegionID"

oParam = oDSComm.InsertCommand.Parameters.Add(new
                SQLParameter("@RegionDescription", SQLDataType.NChar, 50))
oParam.SourceVersion = DataRowVersion.Current
oParam.SourceColumn = "RegionDescription"

Dim oDS as New DataSet

‘取回 RegionID
oDSComm.MissingSchemaAction = MissingSchemaAction.AddWithKey
oDSComm.FillDataSet(oDS, "Region")

Dim index
Dim oRow as DataRow

for index=0 to 20
      oRow = oDS.Tables("Region").NewRow()
      oRow (0) = CStr(index)
      oRow (1) = "地区"+CStr(index)
      oDS.Tables("Region").Rows.Add(newRow)
next

Try
      oDSComm.Update(oDS,"region")
Catch oException As Exception
      //错误处理
Finally
      oConn.close
End Try
```

3.4.2 修改记录

对数据库中的数据进行修改的 sql 语句时非常简单的，如用：

UPDATE FORUM SET Notes='大家好啊!!' where [ID]=1

这个语句即可把表 FORUM 中 ID=1 的字段 Notes 的值改为“大加好啊!!”，这些大家应该很熟悉的了。但是不知大家在 .NET 中用此语句来操作数据库应用的如何?

下面就是我们具体的应用，我们创建一个 aspx 文件来具体时间一下。我们这个简单的程序具有编辑、更新、取消更新的功能。下面是我们的文件代码：

（code\database\UpDate.aspx）

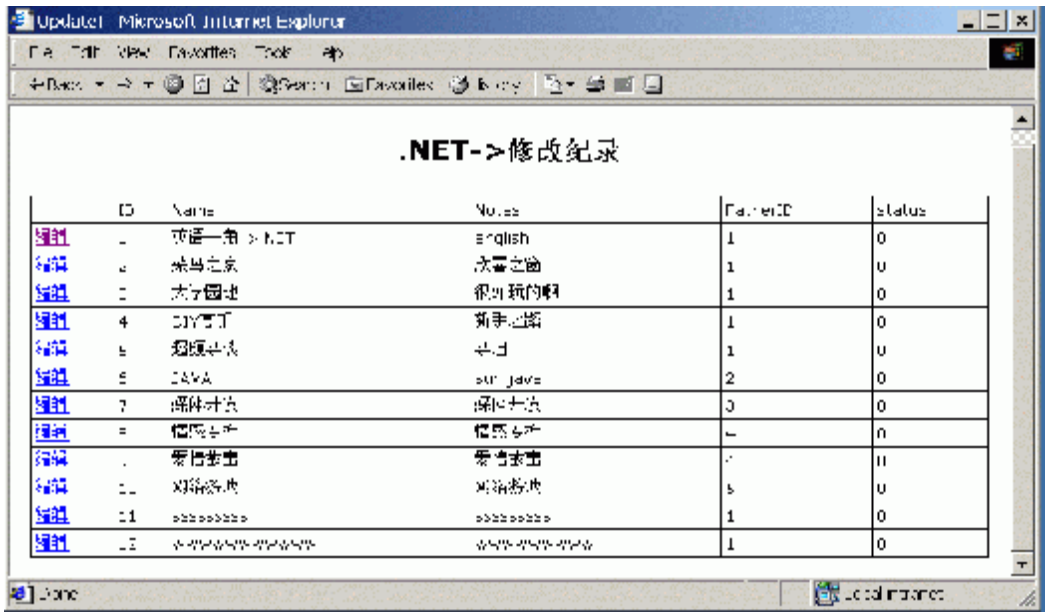
```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>

<script language="VB" runat="server">

    '建立数据连接和命令对象
    Dim UConn As SQLConnection

    '在页面装入时用此方法
    Sub Page_Load(Sender As Object, E As EventArgs)

        '建立与数据库的连接
        UConn = New SQLConnection("server=localhost;uid=NetBBS;pwd=;database=NETBBS")

        If Not (IsPostBack)
            BindGrid()
        End If
    End Sub

    '在点击 Edit 连接时用到此方法：
    Sub UDG_Edit(Sender As Object, E As DataGridCommandEventArgs)

        UDG.EditItemIndex = CInt(E.Item.ItemIndex)
        BindGrid()
    End Sub

    '在点击 Cancle 连接时用到此方法：
```

```
    Sub UDG_Cancel(Sender As Object, E As DataGridCommandEventArgs)

        UDG.EditItemIndex = -1
        BindGrid()
    End Sub

    '在点击 UpDate 连接时用到此方法：
    Sub UDG_Update(Sender As Object, E As DataGridCommandEventArgs)

    '创建数据集
        Dim DS As DataSet

        '创建命令对象
        Dim UComm As SQLCommand

        '定义修改数据的 sql 语句
        Dim UpdateCmd As String = "UPDATE FORUM SET
[ID]=@fid,[Name]=@fname,Notes=@Notes,FatherID=@FatherID,status=@status where
[ID]=@fid"

    '设置命令对象类型
        UComm = New SQLCommand(UpdateCmd, UConn)

    '获得更改的数据
        UComm.Parameters.Add(New SQLParameter("@fid", SQLDataType.VarChar, 4))
        UComm.Parameters.Add(New SQLParameter("@fname", SQLDataType.VarChar, 50))
        UComm.Parameters.Add(New SQLParameter("@Notes", SQLDataType.VarChar, 500))
        UComm.Parameters.Add(New  SQLParameter("@FatherID",  SQLDataType.VarChar,
4))
        UComm.Parameters.Add(New SQLParameter("@status", SQLDataType.VarChar, 1))

    '
        Dim Cols As String() = {"@fid","@fname","@Notes","@FatherID","@status"}

    '激活数据连接
        UComm.ActiveConnection.Open()


        Try
        '执行命令集
            UComm.ExecuteNonQuery()
            Message.InnerHtml = "修改成功！！"
```

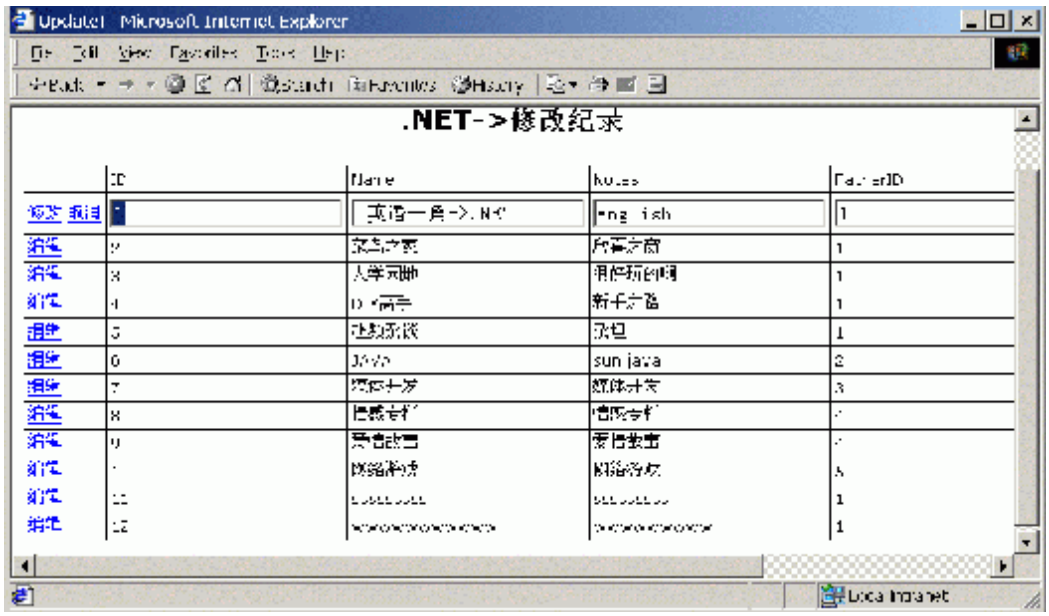
```
            '改为 Edit 连接
            UDG.EditItemIndex = -1

        '处理异常
            Catch Exp As SQLException
                If Exp.Number = 2627
                    Message.InnerHtml = "相同的记录在数据库中"
                Else
                    Message.InnerHtml = "不能更改纪录!! "
                End If
                Message.Style("color") = "red"
            End Try

        '关闭数据连接
            UComm.ActiveConnection.Close()

        '调用 BindGrid()方法
            BindGrid()

        End Sub

        '定义 BindGrid()方法
        Sub BindGrid()

            Dim DS As DataSet
            Dim UComm As SQLDataSetCommand

        '从表 forum 选出数据
            UComm = new SQLDataSetCommand("select * from forum", UConn)

        '填充数据集
            DS = new DataSet()
            UComm.FillDataSet(DS, "forum")

        '数据的绑定
            UDG.DataSource=DS.Tables("forum").DefaultView
            UDG.DataBind()
        End Sub

</script>
<title>
    Update!
</title>
<body style="font: 10pt verdana">
```

```
<BR>
<CENTER>
   <form runat="server">

      <h3><font face="Verdana">.NET->修改纪录</font></h3>

      <span id="Message" MaintainState="false" style="font: arial 11pt;" runat="server"/><p>

      <!--响应对数据库操作的模板-->
      <ASP:DataGrid id="UDG" runat="server"
         Width="800"
         BackColor="#ffffff"
         BorderColor="black"
         ShowFooter="false"
         CellPadding=3
         CellSpacing="0"
         Font-Name="Verdana"
         Font-Size="8pt"
         HeaderStyle-BackColor="#ffffff"
         OnEditCommand="UDG_Edit"
         OnCancelCommand="UDG_Cancel"
         OnUpdateCommand="UDG_Update"

      >

         <property name="Columns">
            <asp:EditCommandColumn EditText="编辑" CancelText="取消" UpdateText="修改"

ItemStyle-Wrap="false"/>
         </property>

      </ASP:DataGrid>

   </form>
  </CENTER>
</body>
</html>
```

首先出来的页面是显示从数据库选择出来的数据，带有编辑的连接，为如下的结果：



然后我们点击“编辑“连接，进入编辑页面，如下：



更改其中的数据，之后点击“修改”，发现我们的数据已经修改了：

3.4.3 删除记录

删除一个记录跟更改纪录差不多，删除一条数据库 NetBBS 中表 forum 中的击卢克用下面的语句：

```
delete from forum where [id]=11
```

这个语句把 id=11 的数据删除掉。

我们在 aspx 文件中实现这个目的，sql 语句已经没有问题了，成下的事情就是获得要删除纪录的 ID 号码。在数据库中，我们定义的表 forum 的 ID 号码为一个主键，我们用下面的语句来获得她：

```
'获得要删除的纪录的 ID 号码
SComm.Parameters.Add(New SQLParameter("@fid", SQLDataType.VarChar, 4))
SComm.Parameters("@fid").Value = SDG。DataKeys(CInt(E.Item.ItemIndex))
```

获得 ID 之后，我们就把此 ID 号传给 sql 语句来执行：

```
Try
        SComm.ExecuteNonQuery()
        Message.InnerHtml = "删除成功！" & DeleteCmd
    Catch Exp As SQLException
```

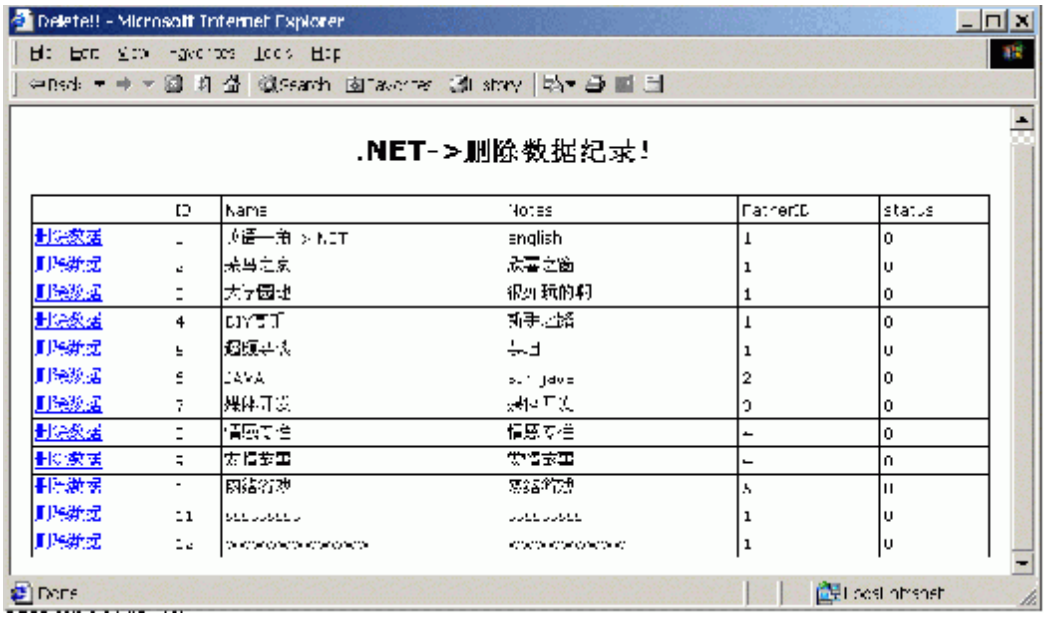
```
            Message.InnerHtml = "不能删除纪录！"
            Message.Style("color") = "red"
        End Try
```

SComm.ExecuteNonQuery()即为执行命令集的语句，成功和失败个返回不同的值。完整的代码如下（code\database\del.aspx）：

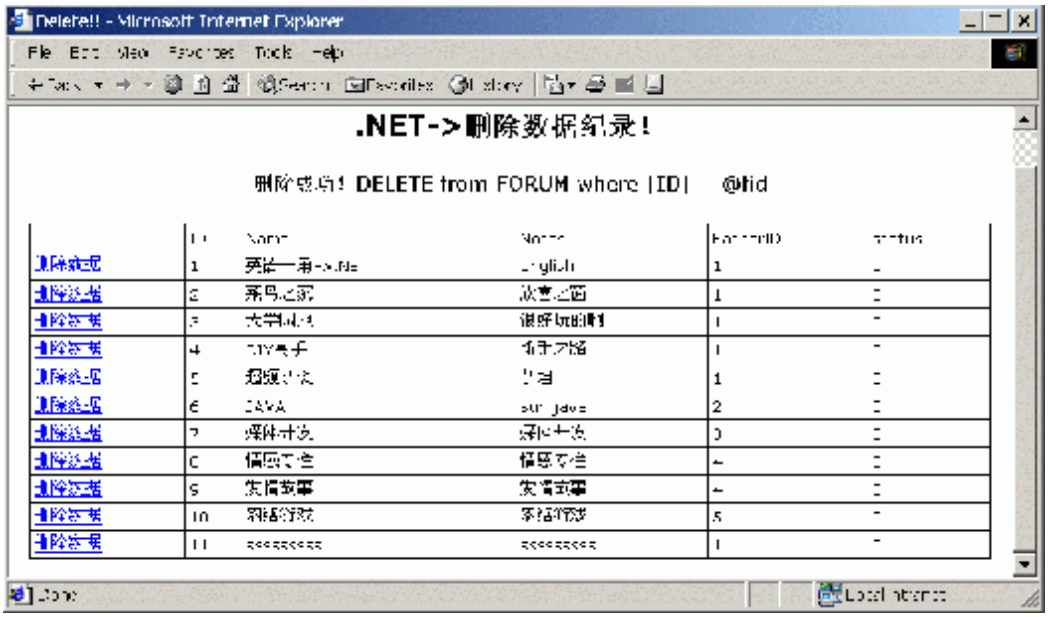
```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>

<script language="VB" runat="server">

    '建立数据连接和命令对象
    Dim SConn As SQLConnection

    '在页面装入时用此方法
    Sub Page_Load(Src As Object, E As EventArgs)

    '建立数据库的连接
        SConn = New SQLConnection("server=localhost;uid=NetBBS;pwd=;database=NETBBS")

        If Not (IsPostBack)
            BindGrid()
        End If
    End Sub

    '删除纪录的方法：
    Sub SDel(Sender As Object, E As DataGridCommandEventArgs)

        '设置命令对象
        Dim SComm As SQLCommand
        Dim DeleteCmd As String = "DELETE from FORUM where [ID] = @fid"

        SComm = New SQLCommand(DeleteCmd, SConn)

    '获得要删除的纪录的 ID 号码
        SComm.Parameters.Add(New SQLParameter("@fid", SQLDataType.VarChar, 4))
        SComm.Parameters("@fid").Value = SDG.DataKeys(CInt(E.Item.ItemIndex))

    '激活数据连接
        SComm.ActiveConnection.Open()
```

```
            Try
                SComm.ExecuteNonQuery()
                Message.InnerHtml = "删除成功！ " & DeleteCmd
            Catch Exp As SQLException
                Message.InnerHtml = "不能删除纪录！ "
                Message.Style("color") = "red"
            End Try

        '关闭数据连接!
            SComm.ActiveConnection.Close()

            BindGrid()
        End Sub

        '数据绑定方法:
        Sub BindGrid()

        '定义数据集
            Dim DS As DataSet
            Dim SComm As SQLDataSetCommand
            SComm = New SQLDataSetCommand("select * from FORUM", SConn)

        '填充数据集
            DS = new DataSet()
            SComm.FillDataSet(DS, "FORUM")

        '打包
            SDG.DataSource=DS.Tables("FORUM").DefaultView
            SDG.DataBind()
        End Sub

</script>
<title>
    Delete!!
<title>
<body style="font: 10pt verdana">


<br><br><br>
<center>
  <form runat="server">

    <h3><font face="Verdana">.NET->删除数据纪录！ </font></h3><br><br>
```

```
        <span id="Message" MaintainState="false" style="font: arial 11pt;" runat="server"/><p>

        <ASP:DataGrid id="SDG" runat="server"
          Width="800"
          BackColor="#ffffff"
          BorderColor="black"
          ShowFooter="false"
          CellPadding=3
          CellSpacing="0"
          Font-Name="Verdana"
          Font-Size="8pt"
          HeaderStyle-BackColor="#ffffff"
          DataKeyField="ID"
          OnDeleteCommand="SDel"
        >

          <property name="Columns">
              <asp:ButtonColumn Text="删除数据" CommandName="Delete"/>
          </property>

        </ASP:DataGrid>

      </form>
    </center>
    </body>
    </html>
```



点击 ID=12 的删除数据连接，结果如下：



我们可以看到该数据已经被我们删除，同时在页面上打印出删除成功的信息，并把该 sql 语句给打印出来了。

3.4.4 存储过程

在 SQL Server 中存储过程是和传统的计算机应用程序最相近的事物，并具有如下的优点：

假如你有一套复杂的 SQL 语句需要在多个 aspx 文件中执行。你可以把他们放入一个存储过程，然后执行该存储过程。这能够减少你 aspx 文件的大小。同时还能确保在每一页上执行的 SQL 语句都相同。当你执行一个 SQL 的批处理时。服务器首先必须编译在批处理中的所有语句。这不但需要时间，还要花费服务器资源。相比较而言，在存储过程第一次执行后，它就不需要重新编译了。通过使用存储过程，你可以跨过编译这一步，更快地执行 SQL 语句集合。从一个动态页中执行一个存储过程比执行一个 SQL 语句的集合更有效。

你可以对存储过程输入输出值。这意味着存储过程非常的灵活，相同的存储过程可以根据不同的输入值返回不同的信息。

当你向数据库服务器传递一个 SQL 语句集合时，必须传递其中的每一个独立语句，而当你执行存储过程时，相反的，仅仅传递一个简单的语句。通过使用存储过程，你可以减少在网络上的阻塞。

你可以配置表的权限，比如用户只能通过使用存储过程来修改表。这就能增加在你数据库中表的安全性。你可以在其他的存储过程内部执行你的存储过程。这种策略就允许你在非常小的存储过程上建立非常复杂的存储过程。这也意味着你可以为许多不同的编程任务使用相同的存储过程。

当你在动态页中添加 SQL 语句时，你必须仔细考虑能否把这些语句放置到存储过程中。上面提到的优点都是实质性的。如下一部分所示，存储过程是非常容易创建的。

3.4.4.1 创建存储过程

l 使用 CREATE PROCEDURE 创建存储过程

你可以使用 CREATE PROCEDURE 来创建一个存储过程。下面就是一个非常简单的存储过程的一个例子：

```
CREATE PROCEDURE pro_book
AS
SELECT * FROM forum
```

当你创建存储过程时，你必须给它指定一个名称。在本例子中，存储过程的名称为 pro_book。你可以给存储过程赋予任何你想要的名称，但最好你能够使该名称在一定程度上描述存储过程的功能。每一个存储过程都包括一个或多个 SQL 语句。为了指明是存储过程一部分的 SQL 语句，你只需简单地在关键词 AS 后面包含它们。在前面例子中的存储过程只包含一个 SQL 语句。当该存储过程执行时，它返回在 forum 表中所有的记录。

你可以使用 EXECUTE 语句来执行一个存储过程。比如，为了执行 pro_book 存储过程，你可以使用如下的语句：

EXECUTE pro_book

当你执行该存储过程时，所有包括在其中的 SQL 语句都会执行，在上面的例子中，会返回所有在 forum 表中的记录。

当在批处理中的第一个语句是调用存储过程时，你并不需要使用 EXECUTE 语句。你可以简单地提供存储过程的名称来执行存储过程。比如在 ISQL/W 中，可以象下面所示来执行存储过程：

pro_book

这起同样的作用。存储过程会被执行，并会返回结果。然而如果在该存储过程之前还有其他的任何语句，你就会收到错误信息（一般地，语法错误）。

当你创建和执行一个存储过程时，这仅仅是在某一个数据库的范围内完成。假设你在数据库 NetBBS 内创建了存储过程 pro_book。如果没有指明过程调用，你就不能在另一个数据库比如 NetBBS2 中调用存储过程 pro_book。假如你需要在 NetBBS2 中执行存储过程 pro_book，你必须使用如下的语句（注意下面的两个点号）：

EXECUTE NetBBS.. pro_book
一旦你已经创建了一个存储过程，你就能使用系统存储过程 sp_helptext 来观看在该存储过程的的 SQL 语句。比如，如果你输入命令 sp_helptext　pro_book，就会显示下面的结果：

text

……

CREATE PROCEDURE pro_book
AS
SELECT * FROM forum

l　注意

你可能感到奇怪的是，sp_helptext 系统过程本身就是一种存储过程类型。它是一种系统的存储过程。（系统存储过程存储在 Master 数据库中，能够被所有的数据库访问。）为了满足你的好奇心，你可以使用命令 sp_helptext sp_helptext 来观看组成 sp_helptext 本身的 SQL 语句。你在创建完存储过程后，不能对其进行修改。假如你需要修改一个存储过程。你必须首先破坏它，然后重新构建之。为了破坏一个存储过程。你可以使用 DROP PROCEDURE 语句，例如下面的语句删除 pro_book 存储过程：

DROP PROCEDURE pro_book

l　注意

你可以使用系统存储过程 sp_help 来观看在当前数据库中所有存储过程的列表。假如你不加任何修改地执行了 sp_help。该过程会显示在当前数据库中所有的存储过程、触发器和表。假如在 sp_help 后面跟上指定的存储过程，sp_help 会仅仅显示那个存储过程的信息。

另外，你也可以使用 SQL Enterprise Manager 创建存储过程，在此就不作介绍了。

l 应用

下面我们就用微软的.NET 技术来讲述怎么样通过程序来给存储过程传递参数。

首先，我们在数据库 NetBBS 中建立一个简单的存储过程，代码如下：

```
create procedure pro_book
as
select * from forum
```

这是一个最简单的存储过程了，此过程名称为 pro_book，是从表 forum 中选出所有的值。

下面我们就通过程序来调用这个存储过程。我们创建一个文件叫 StorePro.aspx，它的代码如下：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<script language="VB" runat="server">

    Sub Page_Load(Src As Object, E As EventArgs)
        '创建数据集
        Dim DS As DataSet
        '创建数据连接对象
        Dim SConn As SQLConnection
        '创建命令集对象
        Dim SComm As SQLDataSetCommand

        '位数据连接对象咐值
        Sconn=New
        SQLConnection("server=localhost;uid=NetBBS;pwd=;database=NETBBS")

        SComm = New SQLDataSetCommand("pro_book", SConn)

        '设置命令对象类型为存储过程
        SComm.SelectCommand.CommandType = CommandType.StoredProcedure

        '建立和填充数据集
        DS = new DataSet()
```

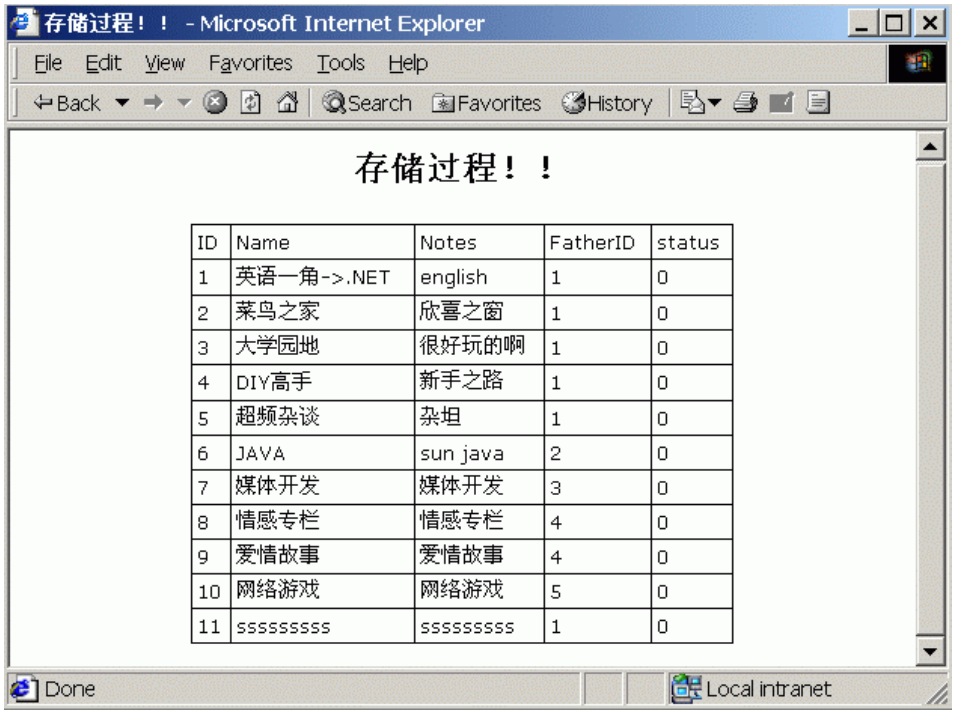
```
        SComm.FillDataSet(DS, "forum")

        SDG.DataSource=DS.Tables("forum").DefaultView
        SDG.DataBind()
    End Sub

</script>

<body>
<center>
  <h3><font face="Verdana">采用存储过程的方式从数据库中调用数据!! </font></h3>

  <ASP:DataGrid id="SDG" runat="server"
    Width="360"
    BackColor="#ccccff"
    BorderColor="black"
    ShowFooter="false"
    CellPadding=3
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#aaaadd"
    MaintainState="false"
  />
</center>
</body>
</html>
```

大家注意到在程序中有这样的语句：

```
        SComm = New SQLDataSetCommand("pro_book", SConn)

        '设置命令对象类型为存储过程
        SComm.SelectCommand.CommandType = CommandType.StoredProcedure
```

最上面的那句话为在命令中用到的存储过程和数据连接对象，其中 pro_book 即为我们在数据库 NetBBS 中创建的存储过程。中间为我们程序的注释；后面的语句为我们设置我们所创建的命令集对象为存储过程。这样，在 aspx 文件中调用存储过程的过程就完成了。

从上面我们可以看到，在 aspx 中调用一个存储过程是一件非常简单的事情。下面是程序运行的结果：



存储过程！！

ID	Name	Notes	FatherID	status
1	英语一角->.NET	english	1	0
2	菜鸟之家	欣喜之窗	1	0
3	大学园地	很好玩的啊	1	0
4	DIY高手	新手之路	1	0
5	超频杂谈	杂坦	1	0
6	JAVA	sun java	2	0
7	媒体开发	媒体开发	3	0
8	情感专栏	情感专栏	4	0
9	爱情故事	爱情故事	4	0
10	网络游戏	网络游戏	5	0
11	sssssssss	sssssssss	1	0

由于我们按照表结构的方式把数据从数据库中选取出来，所以我们看到的结果就象是一张表一样，其实我们可以定制我们的输出结果，在后面的章节中我们会详细的介绍这种方法的。

3.4.4.2 有返回值

I 从存储过程中获得值

你可以从存储过程中接受值。这些值可以直接在你的 aspx 文件中使用。同样，你可以在其他的存储过程中获得这些值。假如第一个过程调用了第二个存储过程，则第一个过程能接受有第二个过程设置的参数值。

例如，下面的存储过程输出变量@fname 的值：

```
create procedure pro_outbook
        ( @fid varchar(20),
         @fname varchar(1000) out )
    as
    select @fname=(select [name] from forum )
```

注意在本例子中关键词 OUT 的使用。该关键词紧跟在参数@fname 的定义后面。这指明该参数将会用于从该过程中输出信息。在这个简单的例子中，参数的值将会是根据 select 语句从表 forum 中选出的 name 的集合。

为了这些一个具有输出参数的存储过程，你需要在 EXECUTE 语句中使用关键词 OUT。假如你在一个批处理或者另外一个存储过程中执行该过程时，你必须首先定义一个变量用于存储从过程中传递出的值，如下面的例子所示：

```
DECLARE @pro_results VARCHAR(1000)

DECLARE @fid varchar(20)

    EXECUTE pro_outbook @fid='2',@fname=@pro_results OUT

    PRINT @pro_results
```

在该例子中的第一个语句定义了将用于存储从过程 pro_outbook 中传出的参数值的变量。该变量将和输出参数的数据类型一模一样。第二个语句执行存储过程。注意变量 @proc_results 后面必须紧跟关键词 OUT。最后变量@proc_results 的值被打印到屏幕上。

3.4.4.3 带输入参数

同样的，我们举一个例子来说明怎样用微软的.NET 技术来讲述怎么样通过程序来给存储过程传递参数，并返回程序的结果。

首先，我们在数据库 NetBBS 中建立一个带有输入参数的存储过程，代码如下：

```
create procedure pro_inputbook
        @fid varchar(4)
as
        select [id] as 'ID',[name] as '论坛名称',[notes] as '公告',[fatherid] as '父 ID',status
    as '使用状态' from forum where [ID]=@fid
```

存储过程的名字为 pro_inputbook，id 和 name 字段用“[]“括号括起来，只要是这两个字段是 SQL Server 中的关键字段。

我们创建一个 aspx 文件来调用我们的存储过程，下面是我们的文件的代码：

（code\database\StoreProWithInPara.aspx）

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<script language="VB" runat="server">
    '在页面装载时调用的方法
```

```
    Sub Page_Load(Src As Object, E As EventArgs)
    '创建数据集
    Dim DS As DataSet

    '创建数据库连接对象
    Dim IConn As SQLConnection

    '常见命令集对象
    Dim IComm As SQLDataSetCommand

    '给数据库连接对象附值
        IConn=New
    SQLConnection("server=localhost;uid=NetBBS;pwd=;database=NETBBS")

    '调用存储过程
    IComm = New SQLDataSetCommand("pro_inputbook", IConn)

    '创建命令集为存储过程!
    IComm.SelectCommand.CommandType = CommandType.StoredProcedure

    '向数据库中传递参数
        IComm.SelectCommand.Parameters.Add(New SQLParameter("@fid",
        SQLDataType.NVarChar, 15))

    '获得并传递参数
   IComm.SelectCommand.Parameters("@fid").Value = Request.QueryString("forumid")

    '填充数据集
      DS = new DataSet()
      IComm.FillDataSet(DS, "forum")

      '进行数据绑定
      IDG.DataSource=DS.Tables("forum").DefaultView
      IDG.DataBind()

  End Sub

</script>

<body style="font: 10pt verdana">
   <br><br><br>
  <center>
     有输入参数的存储过程的输出结果:  <br><br><br>
  <ASP:DataGrid id="IDG" runat="server"
```

```
        Width="650"
        BackColor="#ccccff"
        BorderColor="black"
        ShowFooter="false"
        CellPadding=3
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
        MaintainState="false"
    />
</center>

</body>
</html>
```

有输入参数的存储过程的输出结果：

ID	论坛名称	公告	父ID	使用状态
1	英语一角	english	0	0

我们给程序传递的参数为 1，通过连接中传递参数的方式来传递。我们通过

```
'获得并传递参数
IComm.SelectCommand.Parameters("@fid").Value = Request.QueryString("forumid")
```

这个语句来获得从连接中传来的参数(forumid=1)，并通过

```
'向数据库中传递参数
    IComm.SelectCommand.Parameters.Add(New SQLParameter("@fid",
    SQLDataType.NVarChar, 15))
```

这个语句来向存储过程 pro_inputbook 传递参数，存储过程在接收到参数 1 后，即把 ID=1 的数据选择出来，即为我们所显示的结果。

3.4.4.4 带输出参数

3.4.4.4.1 存储过程

首先我们创建一个带输出参数的存储过程：

```
create procedure pro_outpara
    @count int output
as
        select @count= (select count(*) from topic)
```

这是一个很简单的存储过程，它的作用就是计算出所有的文章数出来。

3.4.4.4.1 接收程序

正向我们上面说的调用存储过程一样，我们创建的命令集诗一样的：
（code\database\StoreProWithOutPara.aspx）

```
IComm = New SQLDataSetCommand("pro_outpara", IConn)
IComm.SelectCommand.CommandType = CommandType.StoredProcedure
```

创建命令集为存储过程方式，在创建数据传送的方式时，就跟上面的不一样了，

```
    IComm.SelectCommand.Parameters.Add(New SQLParameter("@title", SQLDataType.Int))  IComm.SelectCommand.Parameters("@title").Direction = ParameterDirection.Output
```

最后用 System.Math.Ceil(IComm.SelectCommand.Parameters("@count").Value) 来获得我们的参数的值，具体的程序如下：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>
<script language="VB" runat="server">

    Sub Page_Load(Src As Object, E As EventArgs)

    '创建数据连接对象
        Dim IConn As SQLConnection
    '创建命令集对象
        Dim IComm As SQLDataSetCommand
```

```
            '创建数据集
            Dim DS As DataSet

            IConn = New
                    SQLConnection("server=localhost;uid=sa;pwd=;database=NETBBS")
            IComm = New SQLDataSetCommand("pro_outpara", IConn)

            '创建命令集为存储过程!

            IComm.SelectCommand.CommandType = CommandType.StoredProcedure

        '获得并传入参数:
            IComm.SelectCommand.Parameters.Add(New SQLParameter("@count",
                    SQLDataType.Int))
            IComm.SelectCommand.Parameters("@count").Direction =
                    ParameterDirection.Output

        '填充数据集
            DS = new DataSet()
            IComm.FillDataSet(DS, "topic")

        '数据绑定
            'IDG.DataSource=DS.Tables("topic").DefaultView
            'IDG.DataBind()

            If Not Page.IsPostBack Then
                TotalPages.Text =
                    System.Math.Ceil(IComm.SelectCommand.Parameters("@count").Value
                )
            End If
        End Sub

</script>

<body bgcolor="#ccccff" style="font: 10pt verdana">
  <br><br><br>
 <center>
    .NET->有输出参数的存储过程的输出结果:  <br><br><br>


    文章总数: <asp:Label id="TotalPages"   runat="server" />
 </center>
```

```
</body>
</html>
```

运行如下：

3.4.5 表间关系

在通常情况下，不能用一个简单的 Grid 来显出我们所要的数据，在这种情况下，我们就会有很多种处理方法。我们可以用一个或者几个文件，通过传递参数来实现这个功能。

就用我们的 BBS 来说，我们想在显示贴子的页面上了解一下发言者的信息，但是我们设计数据库的时候，分别把用户的地址、教育程度、收入等分别存放在不同的表中的，但是他们都有一个 ID 是共同的，那么我们就可以根据这个 ID 分别从不同的表中选出用户的地址、教育程度、收入等信息。

又比如我们想在显示贴子的页面上获得论坛 ID，之后可以通过传递论坛的 ID 号来获得论坛信息。至于论坛 ID 的传送相对接收来说又不同的方法，可以在扁担上传送，也可以在连接上包含参数来传递。至于接收，我们可以用如下语句：

```
Request.QueryString("id")
```

来实现，在把这个 ID 传给 sql 语句，即可选出我们想要的信息。
我们写两个 aspx 页面来具体讲解如何实现这个表见关系的功能：

（code\database\Rel01.aspx）代码如下：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>

<script language="VB" runat="server">

    '在页面装入时用此方法
    Sub Page_Load(Src As Object, E As EventArgs)



        '定义数据集
        Dim DS As DataSet

        '建立数据连接和命令对象
            Dim rConn As SQLConnection

            '设置命令对象
            Dim rComm As SQLDataSetCommand

        '建立数据库的连接
            rConn = New SQLConnection("server=localhost;uid=NetBBS;pwd=;database=NETBBS")
            rComm = New SQLDataSetCommand("select UserID,ForumID AS '论坛 ID', title as '贴子标题

',contents as '贴子内容',nView as '浏览人数',nreply as '回复人数' from topic", rConn)

        '填充数据集
            DS = new DataSet()
            rComm.FillDataSet(ds, "topic")

        '打包
            rDG.DataSource=ds.Tables("topic").DefaultView
            rDG.DataBind()
        End Sub

</script>
<title>
    Relation!!!
</title>
```

```
<body style="font: 10pt verdana">

<center>
<br>
  <form runat="server">

    <h3><font face="Verdana">.NET->表间关系！</font></h3>

    <span id="Message" MaintainState="false" style="font: arial 11pt;" runat="server"/><p>

    <ASP:DataGrid id="rDG" runat="server"
      Width="800"
      BackColor="#ffffff"
      BorderColor="black"
      ShowFooter="false"
      CellPadding=3
      CellSpacing="0"
      Font-Name="Verdana"
      Font-Size="8pt"
      HeaderStyle-BackColor="#aaaadd"
      DataField="UserID"
    >

      <property name="Columns">
        <asp:HyperLinkColumn
          DataNavigateUrlField="UserID"
          DataNavigateUrlFormatString="Rel02.aspx?id={0}"
          Text="用户信息"
        />
      </property>

    </ASP:DataGrid>

  </form>
</center>
</body>
</html>
```

（code\database\Rel02.aspx）文件中的接收参数的方法跟上面我们说的一样，我们的 sql 语句相对简单，但是即使对复杂的 sql 语句，他们的方法都是一样的，下面来看看我们的第二个文件的代码：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>

<html>

<script language="VB" runat="server">

    '在页面装入时用此方法
    Sub Page_Load(Src As Object, E As EventArgs)

    '定义数据集
        Dim DS As DataSet

        '建立数据连接和命令对象
        Dim rConn As SQLConnection

        '设置命令对象
        Dim rComm As SQLDataSetCommand

        Dim SelectCmd As String = "select [ID],education, area, salary from [user]    where
[ID]=@uid"

    '建立数据库的连接
        rConn                                =                                New
SQLConnection("server=localhost;uid=NetBBS;pwd=;database=NETBBS")
        rComm = New SQLDataSetCommand(SelectCmd, rConn)

    '获得纪录的 ID 号码
        rComm.SelectCommand.Parameters.Add(New                SQLParameter("@uid",
SQLDataType.VarChar, 11))
        rComm.SelectCommand.Parameters("@uid").Value = Request.QueryString("id")

    '填充数据集
        DS = new DataSet()
        rComm.FillDataSet(ds, "user")

    '打包
        rDG.DataSource=ds.Tables("user").DefaultView
        rDG.DataBind()
```

```
        End Sub

    </script>
    <title>
        Relation!!!
    </title>

    <center>
    <body style="font: 10pt verdana">


      <form runat="server">

        <h3><font face="Verdana">.NET->表间关系！</font></h3>

        <h4><font  face="Verdana">用户： <%=Request.QueryString("id")%> 的详细信息
</font></h4>

        <ASP:DataGrid id="rDG" runat="server"
          Width="800"
          BackColor="#ffffff"
          BorderColor="black"
          ShowFooter="false"
          CellPadding=3
          CellSpacing="0"
          Font-Name="Verdana"
          Font-Size="8pt"
          HeaderStyle-BackColor="#ffffff"
        />

      </form>
    </center>
    </body>
    </html>
```

点击用户信息连接，我们会卡到下面的结果：

3.4.6 事务处理

事务控制或者事务管理，是指关系型数据库管理系统执行数据库事务的能力。事务是最基本的工作单元，事务中的 sql 语句必须按照逻辑次序执行，并且要就是成功的执行整个工作单元的操作，要么就一点也不执行。

比如有一个表 area，保存用户的家庭地址，在同一时间内，由两个用户同时对着一个表进行操作，一个用户的操作是：

select * from area

另外一个是：

update area(address) values(“深圳市福田区园岭西路”)

则第一个用户选出来的地址是原来数据苦中存在的地址，还是更新后的地址呢？结果显示，在 select 过程中出现修改，则原来的信息是无效的。

在 sql server 中，创建事务的语句是：

begin {transaction | tran}[transaction_name]

由于我们这本书不是专门介绍 sql server 的，所以在此就不涉及得太多了，大家有兴趣可以

参考其他有关书籍。

我们现在这个例子就是在用户插入数据之前有一个查询，我们应用数据库的事务处理控制方法，在查询操作结束之前，禁止数据的插入。

3.4.7 小结

本章介绍了如何以 ADO.NET 编程方式来实现数据库的常用操作。我们重点介绍了记录的增加、删除和修改，以及目前比较流行的在数据库端使用存储过程的方法。另外我们还介绍了数据库表间的关系描述以及重要应用的事物处理方法。

第五章　Dataset 的用法

Dataset 并不是 Recordset 的简单翻版。从一定的意义上来说，DataView 更类似于 Recordset。如果说 DataReader 是访问数据的最容易的方式，那么 Dataset 则是最完整的数据访问对象。通过 Dataset，你可以操作已有的数据，还可以通过程序创建 Dataset，加入 Table 到 Dataset，并建立这些 Table 之间的关系。

3.5.1 使用 Dataset 的几个步骤

第 1 步，创建到数据源的连接：

```
SQLConnection con =new SQLConnection("server=localhost;uid=sa;pwd=;database=pubs");
```

第 2 步，创建 DataSetCommand 对象，指定一个存储过程的名字或者一个 SQL 语句，指定数据链路；

```
SQLDataSetCommand cmd =new SQLDataSetCommand("SELECT * FROM Authors", con);
```

第 3 步，创建一个 Dataset 对象

```
DataSet ds = new DataSet();
```

第 4 步，调用 DataSetCommand 的 FillData 方法，为 Dataset 填充数据。注意：数据链路没有必要是打开的。如果数据链路是关闭状态，FillData 函数会打开它，并在 FillData 之后关闭数据链路。如果数据链路本来就是打开的，在 FillData 之后，数据链路依然保持打开状态。

```
int iRowCount = cmd.FillDataSet(ds, “Authors”);
```

第 5 步，操作数据。由于 FillData 返回了记录的个数，我们可以构造一个循环，来操纵 Dataset 中的数据。

```
for(int i=0; i< iRowCount; i++){
    DataRow dr = ds.Tables[0].Rows[i];
    Console.WriteLine(dr["au_lname"]);
}
```

3.5.2 小结

本章主要介绍了如何从远程数据库取得数据到本地 DataSet 中的方法步骤。

第六章　数据绑定技术

本文介绍 ASP.NET 的 Repeater，DataList，and DataGrid 服务器端控件。这些控件将数据集合表现为基于 HTML 的界面。本文还引入了利用这些控件的几个例子。

3.6.1 简介

Repeater、DataList 、DataGrid 控件是 System.Web.UI.WebControls 名空间(Namespace)里几个相关的页面组件。这些控件把绑定到它们的数据通过 HTML 表现出来，它们又被成为“列表绑定控件”(list-bound controls)。

和其他 Web 组件一样，这些组件不仅提供了一个一致的编程模型，而且封装了与浏览器版本相关的 HTML 逻辑。这种特点使得程序员可以针对这个对象模型编程，而无须考虑各种浏览器版本的差别和不一致性。

这三个控件具有把它们的相关数据“翻译”成各种外观的能力。这些外观包括表格、多列列表、或者任何的 HTML 流。同时，它们也允许你创建任意的显示效果。除此之外，它们还封装了处理提交数据、状态管理、事件激发的功能。最后，它们还提供了各种级别的标准操作，包括选择、编辑、分页、排序等等。利用这些控件，你可以轻松地完成如下的 Web 应用：报表、购物推车、产品列表、查询结果显示、导航菜单等等。

下面我们进一步讲解这些控件，其基本使用方法和如何选用它们。

3.6.2 列表绑定控件是如何工作

下面我们来看看列表绑定控件的属性和方法，从而一窥其内在工作机理。

3.6.2.1 DataSource 属性

Repeater、DataList、DataGrid 都是从 System.Collections.Icollection 继承来的，所以都带有 DataSource 属性。DataSource，最简单地讲，就是一组相同特征的对象或者一个相同对象的集合。

在 ASP.NET 框架里，有许多对象都有 DataSource 属性。包括 System.Data.DataView 和 ArrayList、HashTable 等等。

和其他传统的需要 ADO Recordset 的数据绑定控件不同，这些列表绑定控件只需要实现其 ICollection 接口，而不必一定指定其 DataSource 属性。而且，由于其 DataSource 属性允许为很多数据类型和数据结构，从而使这些对象的引用更加简单和灵活。

例子 1：下面我们以从服务器的 SQL Server 数据库 pubs 中取出作者信息，作为 三种控件 Repeater、DataList、DataGrid 的数据源为例，来说明以数据视图(DataView)作为数据源(DataSource)的方式。

设计如下：在画面的上部有一选择列表（DropDownList）。当用户从中选取一种控件来显示数据时，它会根据选择，把隐藏在下部的 3 个画板之一显示出来。

1．以数据视图作为数据源方式的源程序

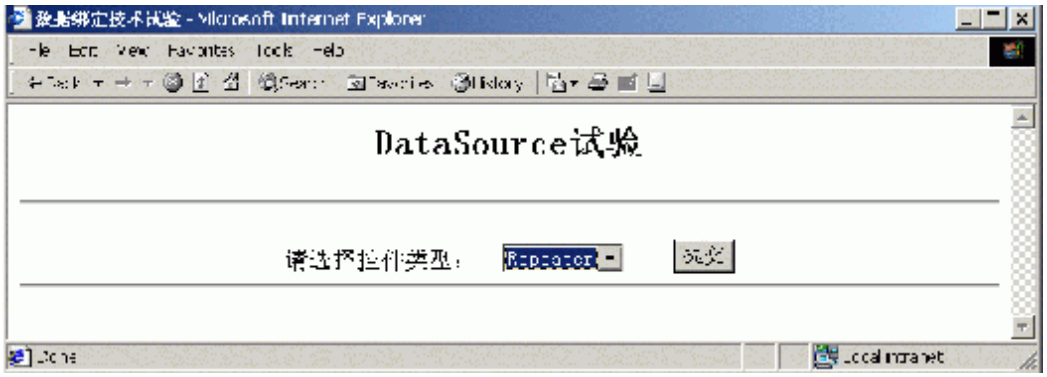
```
<!-- 文件名：code\database\FormDataSource.aspx -->

<!-- 文件名：FormDataSource.aspx -->

<%@ import namespace="system.data" %>
<%@ import namespace="system.data.sql" %>
<!--DataSet 要引用 system.data,数据库连接要用到 system.data.sql-->
<html>

<script language="vb" runat=server>

 sub Page_Load(o as object,e as eventargs)
   dim MyConnection as SQLConnection
   dim MyStr as String
   dim MyDataSetCommand as SQLDataSetCommand
   dim MyDataSet as New DataSet

   If Not IsPostBack

   MyConnection=New

SQLConnection("server=localhost;uid=sa;pwd=;database=pubs")
    '指定连接的服务器、用户、口令、数据库
   MyStr="Select au_lname,au_fname from authors"
    '要得到的数据为 author 表中的姓氏和名字
   MyDataSetCommand=New SQLDataSetCommand(Mystr,MyConnection)
   MyDataSetCommand.FillDataSet(MyDataSet,"Authors")
    '从数据库中取得数据放入内存 DataSet 对象中,并映射为 Authors 表
```

```
Session("MyDs")=MyDataSet
 '保存 DataSet 对象于连接变量 MyDs 中

Else
    MyDataSet=Session("MyDs")
    '取出 DataSet 对象
    if MyDataSet is Nothing
     Response.Write("无法取得数据")
    else
          '根据选择列表的选择,绑定数据，并显示相应的画板
          Select Case DpDnLst.SelectedItem.text
     case "Repeater"
        Response.write _
        ("<center>以<I>Repeater</I>控件显示数据</center>")
        db1.datasource=MyDataSet.tables("authors").defaultview
        db1.databind

   panel1.visible=True
   panel2.visible=False
   panel3.visible=False
     case "DataList"
   Response.write _
        ("<center>以<B>DataList</B>控件显示数据</center>")
        db2.datasource=MyDataSet.tables("authors").defaultview
        db2.databind

   panel1.visible=False
   panel2.visible=True
   panel3.visible=False


     case "DataGrid"
        Response.write _
        ("<center>以<U>DataGrid</U>控件显示数据</center>")
        db3.datasource=MyDataSet.tables("authors").defaultview
        db3.databind

   panel1.visible=False
   panel2.visible=False
   panel3.visible=True

     case else
   End Select
```

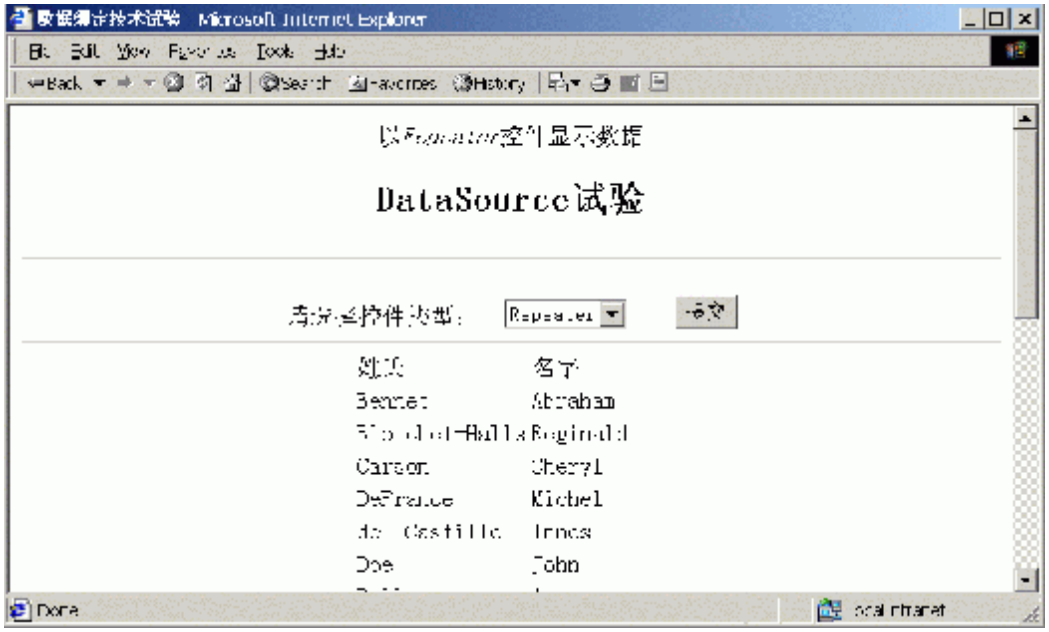
```
      end if
   End If
 end sub
</script>
<head>
<title>
数据绑定技术试验
</title>
</head>

<body bgcolor=#ffffff>
 <center>
    <h2>DataSource 试验</h2>
    <hr>

    <form runat=server>
    请选择控件类型： 
    <asp:DropDownList id="DpDnLst" runat=server>
       <asp:Listitem>Repeater</asp:Listitem>
       <asp:Listitem>DataList</asp:Listitem>
       <asp:Listitem>DataGrid</asp:Listitem>
    </asp:DropDownList>
      
    <asp:button text="提交" runat=server/>
    <hr>

    <!--定义三个画板，根据下拉列表的选择，使指定的画板可见-->

    <!-- 画板一 :定义一个 Repeater 控件 -->
    <asp:panel id="panel1" visible=false runat=server>
    <asp:repeater id="db1" runat=server>
        <!--定义 Repeater 控件显示的表头 -->
       <template name="headertemplate">
          <table>
             <tr>
               <td>
                 姓氏
               </td>
               <td>
                 名字
               </td>
             </tr>
       </template>
```

```
        <!--定义 Repeater 控件数据显示的格式 -->
       <template name="itemtemplate">
        <tr>
         <td>
          <%# databinder.eval(container.dataitem,"au_lname") %>
         </td>
         <td>
         <%# databinder.eval(container.dataitem,"au_fname") %>
         </td>
        </tr>
       </template>

        <!--定义 Repeator 控件显示的表尾 -->
       <template name="footertemplate">
         </table>
       </template>
      </asp:repeater>
     </asp:panel>

       <!-- 画板二：定义一个 DataList 控件 -->
     <asp:panel id="panel2" visible=false runat=server>
      <asp:datalist id="db2" runat=server>
         <!--定义 datalist 的显示格式为：姓氏----名字 -->
       <template name="itemtemplate">
        <%# databinder.eval(container.dataitem,"au_lname") %>
        ----
        <%# databinder.eval(container.dataitem,"au_fname") %>
        <br>
       </template>
      </asp:datalist>
     </asp:panel>

      <!-- 画板三：定义一个 DataGrid 控件 -->
     <asp:panel id="panel3" visible=false runat=server>
      <asp:datagrid id="db3" runat=server>
      </asp:datagrid>
     </asp:panel>

   </form>
  </center>
</body>
</html>
```

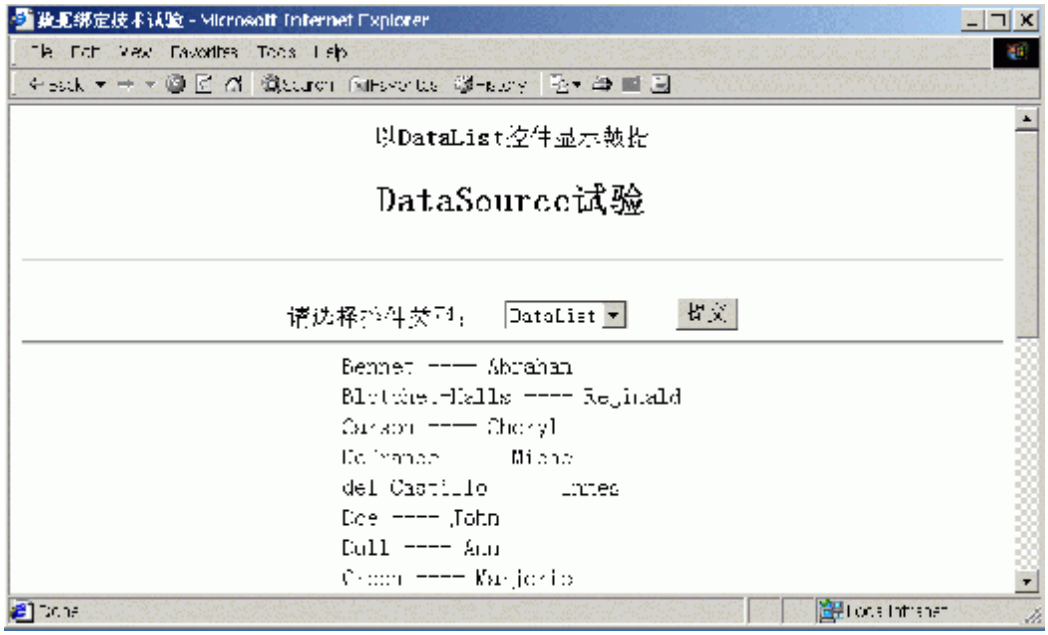
2．开始时的输出画面：



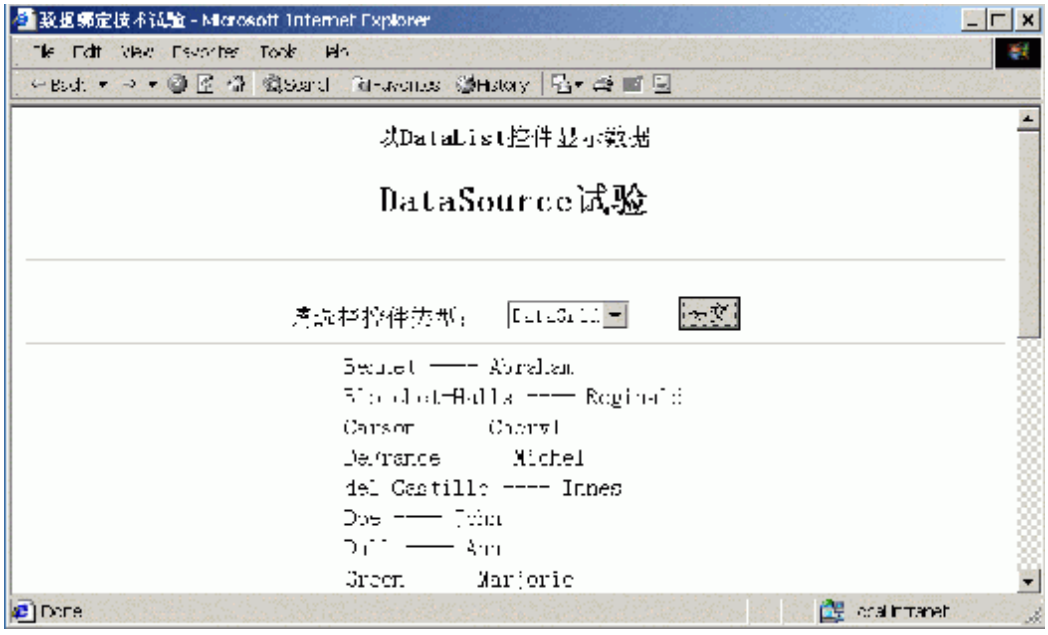
3．当选择以 Repeater 控件显示后的输出画面：



4．当选择以 DataList 控件显示后的输出画面：



5．当选择以 DataGrid 控件方式显示后的输出画面：



例子 2：下面举一个简单的例子演示用 ArrayList 作为数据源的情况，因为三种控件（Repeater 控件、DataList 控件、DataGrid 控件）关于数据源数据的取得方法是一样的，虽然最终的表现形式并不一样，为了节省篇幅，我们只以 DataGrid 控件作为输出。

为和例子 1 比较，我们也以输出上述名字为例，但由于是演示，只取了前 5 人的姓名。

1． 以数组列（ArrayList）作为数据源的源程序

```
<!-- 文件名：FormDataSource01.aspx -->

<html>

 <script language="vb" runat=server>

 '定义一个类用于保存姓名
 Public Class PName
 private first_name as String
 private last_name as String

 Public Property Fname as String
    Get
       return first_name
    End Get

    Set
      first_name=value
    End Set
 End Property

 Public Property Lname as String
    Get
       return last_name
    End Get

    Set
       last_name=value
    End Set
 End Property

 '创建实例
 Public Sub New(f as String,l as String)
     MyBase.New
           first_name=f
           last_name=l
 End Sub

 End Class

    Sub Page_Load(o as object,e as eventargs)
       If Not IsPostBack
```

```
        '第一次请求时初始化一个姓名数组，然后绑定到 datagrid 上
        dim Values as New ArrayList

            Values.add(New PName("Bennet","Abraham"))
            Values.add(New PName("Blotchet-Halls","Reginald"))
            Values.add(New PName("Carson","Cheryl"))
            Values.add(New PName("DeFrance","Michel"))
            Values.add(New PName("del Castillo","Innes"))

            dtgrd.datasource=values
            dtgrd.databind
        End If
    End Sub
</script>

<head>
  <title>
    数据绑定试验
  </title>
</head>

<body bgcolor=#ccccff>
  <center>
    <h2>数据绑定之数据源试验(ArrayList)</h2>
    <hr>
      <form runat=server>
          <asp:DataGrid id="DtGrd" runat=server/>
      </form>
  </center>
</body>
</html>
```

2．程序运行后的输出结果：

例子 3：下面的例子将演示如何使用 HashTable 作为列表控件的数据源的使用方法，它基本上和 ArrayList 的用法类似，只是在添加时要有索引如：MyHashTable.add(index,object)，index 为 hash 表的关键字，object 为具体的内容；用它作数据源时，要用它的 Values 属性而并不是其本身，如：MyDataGrid.DataSource=MyHashTable.Values。

1． 用 HashTable 作为数据源例子的源程序

```
<!-- 文件名： FormDataSoure02.aspx -->
<html>

 <script language="vb" runat=server>

'定义一个类用于保存姓名
 Public Class PName
 private first_name as String
 private last_name as String

 Public Property Fname as String
    Get
      return first_name
    End Get

    Set
```

```
    first_name=value
  End Set
End Property

Public Property Lname as String
  Get
    return last_name
  End Get

  Set
    last_name=value
  End Set
End Property

'创建实例
Public Sub New(f as String,l as String)
    MyBase.New
        first_name=f
        last_name=l
End Sub

End Class

  Sub Page_Load(o as object,e as eventargs)

  dim ht as New HashTable
      if Not IsPostBack
          'hash Table 的 Add 方法所带参数为:索引，对象
          ht.add("1",New PName("Bennet","Abraham"))
          ht.add("2",New PName("Blotchet-Halls","Reginald"))
          ht.add("3",New PName("Carson","Cheryl"))
          ht.add("4",New PName("DeFrance","Michel"))
          ht.add("5",New PName("del Castillo","Innes"))

          DtGrd.datasource=ht.values
          DtGrd.databind
           '数据绑定到 hashtable 上

      End if
  End Sub
</script>

<head>
 <title>
```

```
  数据绑定试验
  </title>
 </head>

 <body   bgcolor=#ccccff>
  <center>
   <h2>数据绑定之数据源试验(HashTable)</h2>
   <hr>
    <form runat=server>
      <asp:DataGrid id="DtGrd" runat=server/>
    </form>
  </center>
 </body>
</html>
```

2． 使用 HashTable 作为数据源的输出结果画面：

例子 4：最后我们以实现 Icolletion 接口方式来实现数据源的绑定，在下面的例子中我们用一个函数 LoadData 返回了一个 Icolletion 对象，实际上在 LoadData 函数内部，我们可以使用上面提到的几种产生数据源的方法来构造 LoadData 函数。

1． 用 Icolletion 对象来作为数据源的例子的源程序：

```
<!-- 文件名：FormDataSource03.aspx -->
<%@ Import NameSpace="System.Data" %>

<html>

  <script language="vb" runat=server>

      Function LoadData() As ICollection

          Dim dt As DataTable
          Dim dr As DataRow

          '建立数据表
          dt = New DataTable
          dt.Columns.Add(New DataColumn("姓氏", GetType(String)))
          dt.Columns.Add(New DataColumn("名字", GetType(String)))

          '载入五个人的数据
          dr = dt.NewRow()
          dr(0) = "Bennet"
          dr(1) = "Abraham"
          dt.Rows.Add(dr)

          dr = dt.NewRow()
          dr(0) = "Blotchet-Halls"
          dr(1) = "Reginald"
          dt.Rows.Add(dr)

          dr = dt.NewRow()
          dr(0) = "Carson"
          dr(1) = "Cheryl"
          dt.Rows.Add(dr)

          dr = dt.NewRow()
          dr(0) = "DeFrance"
          dr(1) = "Michel"
          dt.Rows.Add(dr)

          dr = dt.NewRow()
          dr(0) = "del Castillo"
          dr(1) = "Innes"
          dt.Rows.Add(dr)

          '返回数据表的数据视图
```

```
            LoadData = New DataView(dt)

        End Function

    Sub Page_Load(o as object,e as eventargs)

            If Not IsPostBack
                DtGrd.DataSource=LoadData
                DtGrd.DataBind
            End If
    End Sub

</script>

<head>
  <title>
    数据绑定试验
  </title>
</head>

<body bgcolor=#ccccff>
  <center>
    <h2>数据绑定之数据源试验(ICollection)</h2>
    <hr>
      <form runat=server>
        <asp:DataGrid id="DtGrd" runat=server/>
      </form>
  </center>
</body>
</html>
```

2．使用 Icollection 对象作为数据源的画面输出：

3.6.2.2 Items 集合

每一个列表绑定控件都有一个 Items 集合，集合中的每一个 Item 是 DataSource 所指定的一个对象。

下表列示的是和 DataSource 指定数据相关联的 Item 类型

Item	缺省类型的一个 Item
AlternatingItem	Items 集合中奇数编号的一个 Item
SelectedItem	当前选中的 Item
EditItem	当前编辑的 Item

下面列示的是和 DataSource 指定数据无关的 Item 类型

Header	用于表达列表表头
Footer	用于表达列表表尾
Separator	用于表达两个 Item 之间的内容。只适用于 Repeater 和 DataList
Pager	用于分页显示数据集合。适用于 DataGrid 控件。

3.6.2.3 数据绑定和 Item 集合的创建

列表绑定控件基于 ASP.NET 框架，需要你明确地进行数据绑定。这就意味着：只有当 DataBind 方法被调用时，才真正需要轮询其 DataSource 所代表的数据。

当 DataBind 方法被调用时，列表绑定控件将轮询 DataSource，创建 Items 集合，并从 DataSource 取回数据，以初始化 Items 集合。如果状态管理被激活，这些控件将自动保存所需要的信息，当用户提交数据时，不再需要你指定 DataSource 属性。

明确的 DataBind 调用使你可以准确地决定什么时候 DataSource 是需要准备好的，同时也减少了和数据库的交互，从而提高了 WEB 应用的性能。

一般的规则是：当你需要重建所有的 Items 时候，你需要调用 DataBind。大多数情况下，你只需要在页面第一次被请求的时候，调用 DataBind。在以后的页面运行中，你只需要在相应的事件中，比如引起 Items 集合变化的事件，或者和数据源关联的查询条件发生了变化，或者数据将从只读模式改变到编辑模式，这时候就需要调用 DataBind 方法。

3.6.2.4 Style 属性

通过使用对象模型的 Style 属性，你可以定义整个 DataList 或者 DataGrid 的外观。这些属性允许你指定字体、颜色、边框以及其表现风格。这些控件自身的属性，包括 ForeColor、BackColor、Font 和 BorderStyle，将影响整个控件的表现风格。

另外，对于控件包含的每个 Item，通过指定 ItemStyle、AlternatingItemStyle、HeaderStyle，也可以控制相应 Item 的外观表现。对于 DataGrid，你还可以控制到每个列的每个单元，只需要指定 HeaderStyle、FooterStyle 和 ItemStyle。

3.6.2.5 Template 模板

Style 控制列表绑定控件的可见格式，而 template 则定义了内容和每个 Item 的表现。你可以把 Template 想象成一小段 HTML 代码，通过它决定了如何把每个 Item 显示给用户。

Repeater 和 DataList 通过你指定的模板来工作，这些模板包括 ItemTemplate、AlternatingItemTemplate、HeaderTemplate。

DataGrid 控件不使用模板。但是，在此控件的 Columns 集合里使用 TemplateColumns 是可以的，而且 TemplateColumns 里的每一个 TemplateColumn 都可以包含一个模板，就象 Repeater 和 DataList 里的一样。这样你也可以定制每一个 DataGrid 的表现形式。

3.6.3 模板里的数据绑定

一个模板 Template 定义了一个 Item 所包含的控件结构。使用数据绑定表达式，这个结构里的控件属性可以绑定到和这个 Item 关联的数据属性。

Item 从逻辑上来看，是相应 Template 的父亲，可以通过“Container”来引用。每个 Container 都有 DataItem 属性，所以在构造 Template 的每个数据绑定表达式时候，Container.DataItem 常常出现。这些我们从后面的例子里也可以学习到。

数据绑定的方式大概有四种：属性绑定、集合绑定、表达式绑定以及方法绑定。

1. **属性绑定：**

是指 ASP.Net 的数据绑定可以绑定到公共的变量、页面的属性乃至其他服务器端控件的属性上。但是应该注意的是，这些属性、公用变量一定要在使用 DataBind()方法以前初始化，否则可能导致不可预知的错误。

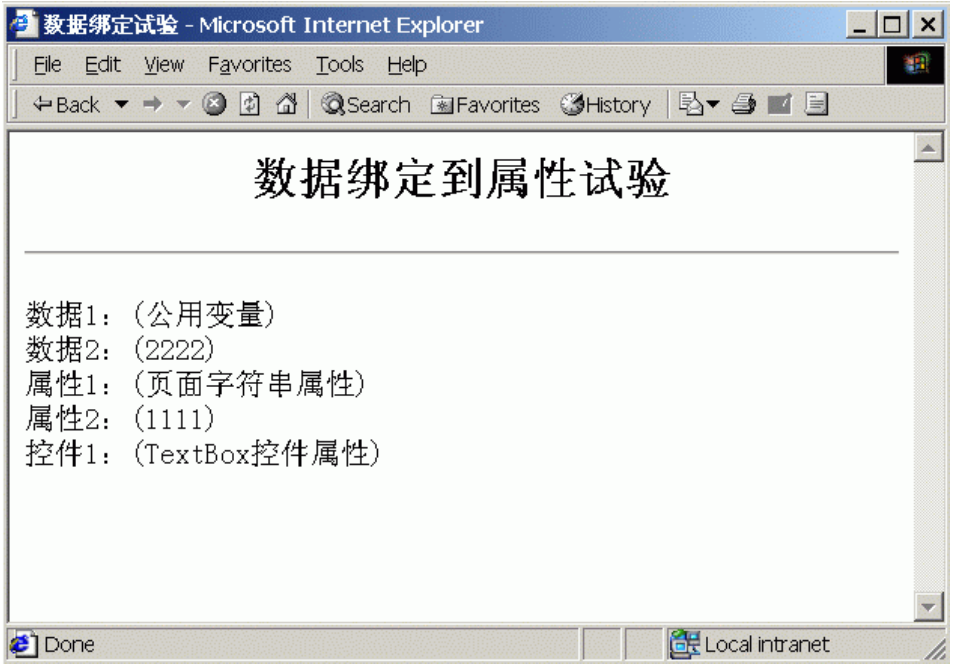
例子：在页面中定义一个字符串变量和一个整型变量，一个字符串属性和一个整型属性，以及一个不可见的 TextBox 控件，然后在页面加载的时候，调用页面的 DataBind 方法，看数据是否绑定成功。

源程序如下：

```
<!-- 文件名：code\database\bonder\FormDataBind01.aspx -->


<!-- 文件名：FormDataBind01.aspx -->

<html>

 <script language="vb" runat=server>

    Public PubVar as New String("公用变量")
    Public PubInt as Integer=2222

    Sub Page_Load(o as object,e as eventargs)
      DataBind
    End Sub

    Public ReadOnly Property PubPropStr as String
            Get
                Return "页面字符串属性"
            End Get
    End Property

    Public ReadOnly Property PubPropInt as Integer
            Get
               Return 1111
            End Get
    End Property

 </script>

 <head>
  <title>
```

```
      数据绑定试验
  </title>
 </head>

 <body bgcolor=#ffffff>
  <center>
   <h2>数据绑定到属性试验</h2>
   <asp:TextBox id="tb" text="TextBox 控件属性"
            visible=False runat=server/>
   <hr>
  </center>
    <form runat=server>
        数据 1：(<%# PubVar %>)<br>
        数据 2：(<%# PubInt %>)<br>
        属性 1：(<%# PubPropStr %>)<br>
        属性 2：(<%# PubPropInt %>)<br>
        控件 1：(<%# tb.text %>)
    </form>
 </body>
</html>
```

执行后的页面输出画面如下：



2． 集合绑定：

作为数据源的还可以是集合对象，在 asp.net 中只要是支持 Icollection 接口的集合对象都可以作为列表服务器端控件。最常见的，我们使用数组（ArrayList）、哈希表（HashTable）、

数据视图（DataView）、数据读写器（DataReader）等集合作为列表服务器端控件的数据源。

例子：以显示一个下拉列表（dropdownlist）为例，说明作为数据源的 4 种集合对象（ArrayList、DataView、HashTable、DataReader）进行数据绑定时的用法。

为了达到相同的输出效果，下拉列表的选项都为我国六个城市。

ArrayList 的用法最简单，首先生成一个 ArrayList 的对象，然后用 Add 方法把城市加入，最后绑定到 DropDownList 控件即可。代码架构如下：

```
Dim values as New ArrayList
Values.add("…")
…
DropDownList1.DataSource=values
DropDownList1.DataBind
…
```

HashTable 用法和 ArryList 的用法查不多，但是有两点值得注意。一是当使用 Add 方法向 HashTable 中添加数据时，它比 ArryList 要多出一个关键值字段，语法为 Add(keyValue,Object)；二是，设置 DataSource 属性值时，不是以 HashTable，而是以其 Values 值作为数据源，其代码框架如下：

```
Dim ht as HashTable

Ht=New HashTable
Ht.add(KeyValue,"…")
'注意 KeyValue 为键值
…
DropDownList1.DataSource=ht.values
DropDownList1.DataBind
…
```

DataView 方式绑定，首先应该得到一个数据视图（DataView），而得到数据视图的方式可以是从远端数据库中取得，或者是本地动态定义 Table，添加数据得到；然后把得到的数据视图赋予 DataSource 属性，同时指定 DataValueField 属性，指定 DataValueField 属性实际上就是指明 DropDownList 控件到底使用数据视图中的哪一个字段作为自己的数据源。下面的代码示例，使用本地定义方式来产生数据视图，当然可以使用 SQL 取数据方式，示例代码框架如下：

```
Dim dt as DataTable
Dim dr as   DataRow

Dt=New DataTable
Dt.Columns.add(New   DataColumn("…",GetTypeString(…))
'产生数据表所需要的字段
…
```

```
dr=dt.NewRow
dr(0)=…
dr(1)=…
…
dt.rows.add(dr)
‘产生一条记录加入到数据表中
…
DropDownList1.DataSource=New DataView(dt)
DropDownList1.DataValueField=…
DropDownList1.DataBind
…

‘以下为 SQL 方式
<%@ Import NameSpace=”System..Data” %>
<%@ Import NameSpace=”System..Data.SQL” %>
…
dim MyConn as SQLConnectoin
dim MyStr as String
dim MyDataSetCommand as SQLDataSetCommand
dim MyDataSet as DataSet

MyConn=new SQLConnection(“server=…;uid=…;sa=…;dataserver=…”)
‘设立连接数据库的字符串
MyStr=”Select * from …”
‘查询数据语句
MyDataSetCommand=New SQLDataSetCommand(MyStr,MyConn)
‘定义取数据命令
MyDataSetCommand.FillDataSet(MyDataSet,”…”)
‘把远地取得的 DataSet 以…名字放入内存 DataSet
…
DropDownList1.DataSource=MyDataSet.tables(…).Defaultaview
DropDownList1.DataValueField=…
DropDownList1.DataBind
…
```

最后 DataReader 方式实际上和 DataView 差不多，区别在于 DataReader 是以流方式取得数据，而 DataView 可以从内存中取得，所以 DataReader 方式在数据绑定以前必须打开连接链路，完成绑定之后再关闭链路，当数据量较大时，这种方式可能会有问题。代码框架如下：

```
dim MyConn as SQLConnection
dim Mystr as String
dim MyComm as SQLCommand
dim MyReader as SQLDataReader
```

```
MyConn=New SQLConnection("server=…;uid=…;pwd=…;database=…")
'连接数据库的字符串
MyStr="select * from …"
‘查询字符串
MyComm=New SQLCommand(Mystr,MyConn)
‘要执行的命令串
MyConn.Open
‘打开通往服务器的链路
MyComm.Execute(MyReader)
‘执行查询语句
DropDownList1.Datasource=MyReader
DropDownList1.DataValueField=…
DropDownList1.DataBind
MyConn.Close
‘绑定完毕才能执行数据链路的关闭
```

下面的代码示例中，使用了服务器上的 SQL 数据库 test 中的一个关于城市名的表 city，我们再使用前，应先在数据库服务器上建立 test 数据库，并建立一个城市名表 city,它至少含有一个 city_name 的字段，为便于比较 4 种不同的实现方法可以达到相同的效果，建议加载的试验数据为相同的城市名,整个例子的完整代码如下：

```
<!-- 文件名： FormDataBind02.aspx -->
<%@ import NameSpace="System.Data" %>
<%@ import NameSpace="System.Data.SQL" %>


<html>

 <script language="vb" runat=server>

    Sub Page_Load(o as object,e as eventargs)

        If Not IsPostBack
            '首次加载，以四种方式绑定数据源

            Dim values as ArrayList

            values=New ArrayList()
            values.add("北京")
            values.add("上海")
            values.add("天津")
            values.add("重庆")
            values.add("香港")
            values.add("澳门")
```

```
 lstArray.datasource=values
 lstArray.databind
 '控件以 ArrayList 方式绑定

 Dim dt as DataTable
 Dim dr as DataRow
 Dim i    as Integer
 Dim ar as Array

 dt=New DataTable()
 dt.Columns.add(New DataColumn("City",GetType(string)))
 '建立一个 city 字段
 For i =0 to 5
 dr=dt.NewRow()
 dr(0)=values.item(i)
 dt.rows.add(dr)
 Next
'添加六个城市的数据

lstDataView.DataSource=New DataView(dt)
lstDataView.DataValueField="City"
lstDataView.DataBind
'控件以 DataView 方式绑定

Dim ht as HashTable

ht=New HashTable()
ht.add("1","北京")
ht.add("2","上海")
ht.add("3","天津")
ht.add("4","重庆")
ht.add("5","香港")
ht.add("6","澳门")

lstHash.DataSource=ht.values
lstHash.DataBind
'控件以 HashTable 方式绑定

dim MyConn as SQLConnection
dim Mystr as String
dim MyComm as SQLCommand
dim MyReader as SQLDataReader
```

```
            MyConn=New SQLConnection("server=localhost;uid=sa;pwd=;database=test")
            '连接服务器上的 Test 数据库
            MyStr="select city_name from city"
            '从 city 表中取城市名字段(city_name)
            MyComm=New SQLCommand(Mystr,MyConn)
            MyConn.Open
            MyComm.Execute(MyReader)
            lstDR.Datasource=MyReader
            lstDR.DataValueField="city_name"
            lstDR.DataBind
            MyConn.Close
          End If
    End Sub
  </script>

  <head>
    <title>
数据绑定试验
    </title>
  </head>

  <body bgcolor=#ccccff>
    <center>
      <h2>数据绑定之集合绑定试验</h2>
      <hr>
        <form runat=server>
          <b>绑定到 ArrayList 的效果:</b>
          <asp:dropdownlist id="lstArray" runat=server />
            
          <b>绑定到 DataView 的效果:</b>
          <asp:dropdownlist id="lstDataView" runat=server />
          <br>
          <p></p>
          <b>绑定到 HashTable 的效果:</b>
          <asp:dropdownlist id="lstHash" runat=server />
            
          <b>绑定到 DataReader 的效果:</b>
          <asp:dropdownlist id="lstDR" runat=server />
        </form>
    </center>
  </body>
</html>
```

3. 绑定到表达式：除了使用固定的数据作为数据绑定的数据源以外，asp.net 还提供了具有动态表达功能的表达式数据绑定，由于它是根据数据项和常数计算而来，因而提供的数据更加灵活、方便。

例子：书价打折计算，当我们从下拉列表中选择一个折扣率后，会显示出各种书的相应价格。源程序如下：

```
<!-- 文件名：FormDataBind03.aspx -->
<%@ import namespace="System.Data" %>
<%@ import namespace="System.Data.Sql" %>

<html>

 <script language="vb" runat=server>

Public CLASS book
    private _name as string
    private _price as decimal

    public readonly property name as string
        Get
            return _name
        end Get
    end property
```

```
    public readonly property price as decimal
          Get
             return _price
          end Get
    end Property

    public sub New(n as string,p as decimal)
    MyBase.New
    _name=n
    _price=p
    end sub
  End Class

    Sub Page_Load(o as object,e as eventargs)

      if IsPostBack
          dim values as New ArrayList
               values.add(New book("红楼梦",100.0))
               values.add(New book("三国演义",90.0))
               values.add(New book("水浒",85.0))
               values.add(New book("西游记",60.0))
           lbltxt.text="各种书的价格是："
           dg1.datasource=values
           dg1.Databind
      end if
    End Sub

    Public Function GetRealPrice(price as decimal)
      dim a as decimal

         a=CDbl(rate.selecteditem.value)
         GetRealPrice=price*a
    End Function
 </script>

 <head>
   <title>
数据绑定试验
   </title>
 </head>

 <body bgcolor=#ccccff>
   <center>
```

```
<h2>数据绑定之表达式绑定</h2>
<hr>
 <form runat=server>
  请输入折扣率：
  <asp:DropDownList id="rate" runat=server>
   <asp:listitem>1.00</asp:listitem>
   <asp:listitem>0.95</asp:listitem>
   <asp:listitem>0.90</asp:listitem>
   <asp:listitem>0.80</asp:listitem>
   <asp:listitem>0.70</asp:listitem>
   <asp:listitem>0.60</asp:listitem>
   <asp:listitem>0.60</asp:listitem>
  </asp:DropDownList>
     
  <asp:button text="提交" runat=server/>
  <hr>
  <asp:label id="lbltxt" runat=server/>
  <br>
  <asp:datalist id="dg1" runat=server>
   <template name="headertemplate">
   <table>
    <tr>
    <th>
    书名
    </th>
    <th>
    价格
    </th>
    </tr>
   </template>

   <template name="itemtemplate">
   <tr>
    <td>
   <%# databinder.eval(container.dataitem,"name") %>
    </td>
    <td>
     $<%# GetRealPrice(databinder.eval(container.dataitem,"price")) %>  元
    </td>
    </tr>
   </template>

   <template name="footertemplate">
   </table>
```

```
        </template>
      </asp:datalist>

      </form>
  </center>
 </body>
</html>
```

开始时的输出画面：

当我们选择对书价进行九五折后，输出的价格如下：

当我们选择七折时的价格输出如下：

4. 方法绑定：实际上在上一个例子中我们已经见到了使用方法的数据绑定，它利用 databinder.eval 方法把指定的数据或者是表达式转换成所期望出现的数据类型。

DataBinder.Eval 含有三个参数,第一个是数据项的容器，对于常用的 DataList、DataGrid、Reapter 等控件，通常使用 Container.DataItem；第二个参数是数据项名；第三个参数是要转换成的数据类型，如果省略就认为是返回该数据项的类型。使用方法绑定的目的通常都是和模板定义相结合产生一些特殊的效果。由于方法绑定比较常见，这里就举一个简单的例子了。

例子：显示待售图书的价格

源程序如下：

```
<!-- 文件名： FormDataBind04.aspx-->

<html>

 <script language="vb" runat=server>

Public CLASS book
    private _name as string
    private _price as decimal

    public readonly property name as string
        Get
            return _name
        end Get
    end property

    public readonly property price as decimal
         Get
            return _price
         end Get
    end Property

    public sub New(n as string,p as decimal)
    MyBase.New
    _name=n
    _price=p
    end sub

  End Class

    Sub Page_Load(o as object,e as eventargs)

    if Not IsPostBack
      dim ht as New ArrayList
```

```
            ht.add(New book("红楼梦",100.0))
            ht.add(New book("三国演义",90.0))
            ht.add(New book("水浒",85.0))
            ht.add(New book("西游记",60.0))
            dl.datasource=ht
            dl.databind
    end if
    End Sub
</script>

<head>
  <title>
    数据绑定试验
  </title>
</head>

<body bgcolor=#ccccff>
  <center>
    <h2>数据绑定之 DataBinder.Eval 方法</h2>
    <hr>
      <form runat=server>
      <b>图书售价清单</b>
      <p></p>
      <asp:datalist id="dl" borderwith="1" girdlines="both" runat=server>

      <template name="itemtemplate">
      书名：<%# Databinder.eval(container.dataitem,"name") %>
        
      售价：<%# Databinder.eval(container.dataitem,"price") %>元
      </template>

      </asp:datalist>

      </form>

  </center>

</body>

</html>
```

画面输出结果如下：

3.6.3.1 Repeater 控件

正如前面讲到的，Repeater 完全是模板驱动的。对同样的 DataSource，通过应用不同的模板，你可以得到不同的外观表现。

我们来看看下面的代码：

```
<%@ Page language="C#" src="Repeater1.cs" inherits="Samples.Repeater1Page"%>
<asp:Repeater runat=server id="linksListRepeater"
  DataSource='<%# SiteLinks %>'>
  <template name="HeaderTemplate">
    <ul type="1">
  </template>
  <template name="ItemTemplate">
    <li>
      <asp:HyperLink runat=server
        Text='<%# DataBinder.Eval(Container.DataItem, "SiteName") %>'
        NavigateUrl='<%# DataBinder.Eval(Container.DataItem, "SiteURL") %>'>
      </asp:HyperLink>
    </li>
  </template>
  <template name="FooterTemplate">
    </ul>
  </template>
</asp:Repeater>
```

这个例子显示了通过(<%# … %>)实现数据绑定的语法。这些数据绑定表达式在你调用 DataBind 的时候得到执行。这里，控件的 DataSource 是这个页面的 DataLinks 属性，它是一些 URL 参考信息。

Repeater 控件是唯一允许在 Template 中使用 HTML 片断的。本例中，列表被分成三段：

- <ul type="1">代表 HeaderTemplate；
- 代表 FooterTemplate；
- 列表的中心内容，是通过<li>来表现的。对 SiteLinks 集合里的每一个对象重复这个 ItemTemplate，就产生了如图的列表内容。

你也可以在 HeaderTemplate 中使用<table>，在 FooterTemplate 中使用</Table>，在 ItemTemplate 中使用<TR>…</TR>。这样你就得到一个表格形式的列表。

你必须指定 ItemTemplate。当 HeaderTemplate 或者 FooterTemplate 没有被指定时，ItemTemplate 将被用作替代。

下面的代码是支持上面代码的：

```
namespace Samples {
    ...

    public class Repeater1Page : Page {
        protected Repeater linksListRepeater;

        public ICollection SiteLinks {
            get {
                ArrayList sites = new ArrayList();

                sites.Add(new SiteInfo("Microsoft Home",
                                       "http://www.microsoft.com"));
                sites.Add(new SiteInfo("MSDN Home",
                                       "http://msdn.microsoft.com"));
                sites.Add(new SiteInfo("MSN Homepage",
                                       "http://www.msn.com"));
                sites.Add(new SiteInfo("Hotmail",
                                       "http://www.hotmail.com"));
                return sites;
            }
        }

        protected override void OnLoad(EventArgs e) {
            base.OnLoad(e);

            if (!IsPostBack) {
```

```
                // DataBind the page the first time it is requested.
                // This recursively calls each control within the page's
                // control hierarchy.
                DataBind();
            }
        }
    }

    public sealed class SiteInfo {
        private string siteName;
        private string siteURL;

        public SiteInfo(string siteName, string siteURL) {
            this.siteName = siteName;
            this.siteURL = siteURL;
        }

        public string SiteName {
            get { return siteName; }
        }
        public string SiteURL {
            get { return siteURL; }
        }
    }
}
```

Repeater1Page 类重载了 Page 类的 OnLoad 方法。我们在页面第一次被请求时候，调用 DatBind 方法。这样，Template 里面的每一个数据绑定表达式被计算。由于 Repeater 可以保存它自身的数据和状态，所以用户提交数据时候，没有必要再次调用 DataBind 方法（也不需要指定 DataSource 了）。

页面公开了一个 ICollection 类型的 SiteLinks 属性。这个属性被用于指定为 Repeater 控件的 DataSource。前面我们已经知道 DataSource 必须是 ICollection 类型的。SiteLinks 就是一个简单的 ArrayList，里面包含一系列的站点信息。SiteLinks 属性被设置为 public 的，因为只有 public 和 protected 的属性在数据绑定表达式中才是可用的。

每一个 SiteInfo 对象有两个属性：SiteName 和 SiteURL。在 ItemTemplate 中，我们通过下面的代码来存取其属性的：

```
<asp:HyperLink runat=server
  Text='<%# DataBinder.Eval(Container.DataItem, "SiteName") %>'
  NavigateUrl='<%# DataBinder.Eval(Container.DataItem, "SiteURL") %>'>
</asp:HyperLink>
```

3.6.3.2 DataList 控件

DataList 是一个模板控件。通过指定其 Style 属性，可以控制它的表现形式。你还可以使用它的多列属性。

例子

:

```
<%@ Page language="C#" src="DataList1.cs" inherits="Samples.DataList1Page"%>
...

<asp:DataList runat=server id="peopleDataList"
  RepeatColumns="2" RepeatDirection="Vertical" RepeatMode="Table"
  Width="100%">

  <property name="AlternatingItemStyle">
    <asp:TableItemStyle BackColor="#EEEEEE"/>
  </property>
  <template name="ItemTemplate">
    <asp:Panel runat=server font-size="12pt" font-bold="true">
      <%# ((Person)Container.DataItem).Name %>
    </asp:Panel>
    <asp:Label runat=server Width="20px"
      BorderStyle="Solid" BorderWidth="1px" BorderColor="Black"
      BackColor='<%# ((Person)Container.DataItem).FavoriteColor %>'> 
    </asp:Label>
     
    <asp:Label runat=server Font-Size="10pt"
      Text='<%# GetColorName(((Person)Container.DataItem).FavoriteColor) %>'>
    </asp:Label>
  </template>
</asp:DataList>
```

通过简单地设置 RepeatColumns="2"，我们得到了一个多列的 DataList。而 RepeatDirection=”Vertical”表示：列表将从上到下、然后从左到右显示。而如果你设置成 RepeatDirection=” Horizontal”,列表将从左到右、然后从上到下显示。

本例用了 DataList 的几个 Style 属性。Width 属性使列表占用整个窗口宽度，而 AlternatingItemStyle 属性设置成灰色，使奇数行和偶数行有所区别。

下面的代码是支持这个例子的：

```
namespace Samples {
    ...

    public class DataList1Page : Page {
        protected DataList peopleDataList;
```

```
    protected string GetColorName(Color c) {
        return
          TypeDescriptor.GetConverter(typeof(Color)).ConvertToString(c);
    }

    private void LoadPeopleList() {
        // create the datasource
        Person[] people = new Person[] {
            new Person("Nikhil Kothari", Color.Green),
            new Person("Steve Millet", Color.Purple),
            new Person("Chris Anderson", Color.Blue),
            new Person("Mike Pope", Color.Orange),
            new Person("Anthony Moore", Color.Yellow),
            new Person("Jon Jung", Color.MediumAquamarine),
            new Person("Susan Warren", Color.SlateBlue),
            new Person("Izzy Gryko", Color.Red)
        };

        // set the control's datasource
        peopleDataList.DataSource = people;

        // and have it build its items using the datasource
        peopleDataList.DataBind();
    }

    protected override void OnLoad(EventArgs e) {
        base.OnLoad(e);

        if (!IsPostBack) {
            // first request for the page
            LoadPeopleList();
        }
    }
}

public sealed class Person {
    private string name;
    private Color favoriteColor;

    public Person(string name, Color favoriteColor) {
        this.name = name;
        this.favoriteColor = favoriteColor;
    }
```

```
        public Color FavoriteColor {
            get { return favoriteColor; }
        }
        public string Name {
            get { return name; }
        }
    }
}
```

例子中，控件的 DataSource 属性是程序运行时指定的，这和在 aspx 中声明这些属性形成对照。其实两种方法的效果完全一样，但是，无论你选择哪种方法，你必须调用 DataBind，以便控件可以枚举 DataSource 来创建控件的每一个项目。

本例中的 DataSource 只是一个由 Person 对象组成的简单数组。由于数组对象实现了 ICollection 接口，所以数组可以作为 DataSource。本例也显示了不同数据结构和数据类型作为 DataSource 的可行性和灵活性。

本例显示了如下概念：

- 在模板中使用丰富的 HTML 用户界面
- 使用数组作为 DataSource
- 编程指定 DataSource
- 在数据绑定时指定各种表达式

3.6.3.3 DataGrid 控件

DataGrid 控件可用于创建各种样式的表格。它还支持对项目的选择和操作。下面的几个例子使用了包含如下字段的一个表：

Title	书名
Title ID	编号
Author	作者
Price	价格
Publication date	发行日期

这个表不在数据库中，而是保存在一个名为 titlesdb.xml 的文件中。我们将逐步给出完整的代码。

首先我们来看看 titlesdb.xml 的格式：

```
<root>
<schema id="DocumentElement" targetNamespace=""
        xmlns=http://www.w3.org/1999/XMLSchema
        xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <element name="Title">
```

```
            <complexType content="elementOnly">
                <element name="title_id" type="string"></element>
                <element name="title" type="string"></element>
                <element name="au_name" type="string"></element>
                <element name="price" msdata:DataType="System.Currency"
                        minOccurs="0"
                        type="string"></element>
                <element name="pubdate" type="timeInstant"></element>
            </complexType>
            <unique name="TitleConstraint" msdata:PrimaryKey="True">
                <selector>.</selector>
                <field>title_id</field>
            </unique>
        </element>
</schema>
<DocumentElement>
        <Title>
            <title_id>BU1032</title_id>
            <title>The Busy Executive's Database Guide</title>
            <au_name>Marjorie Green</au_name>
            <price>19.99</price>
            <pubdate>1991-06-12T07:00:00</pubdate>
        </Title>
        ...
</DocumentElement>
</root>
```

在一个典型的 web 应用中，很可能你需要使用 web service 或者商业部件来保证最大可能的可扩展性和性能。下面的例子为了简化代码，我们在 global.asax 中，通过响应 application_onstart 事件，读取 xml 数据到一个 DataSet，然后缓存这个 DataSet 到一个 Application 变量。

代码如下：(global.asax)

```
public void Application_OnStart() {
    FileStream fs = null;
    DataSet ds = null;

    try {
        fs = new FileStream(Server.MapPath("TitlesDB.xml"), FileMode.Open,
                            FileAccess.Read);
        ds = new DataSet();

        // load the data in the xml file into the DataSet
```

```
            ds.ReadXml(fs);
        } finally {
            if (fs != null) {
                fs.Close();
                fs = null;
            }
        }

        // cache the dataset into application state for use in individual pages
        Application["TitlesDataSet"] = ds;
}
```

下面的代码产生了一个简单的页面：

（dg01.aspx）

```
<%@ Page language="C#" src="DataGrid.cs" inherits="Samples.DataGridPage"%>
...

<asp:DataGrid runat=server id="titlesGrid">
</asp:DataGrid>
```

(DataGrid.cs)

```
namespace Samples {
    ...

    public class DataGridPage : Page {
        protected DataGrid titlesGrid;

        public ICollection GetTitlesList() {
            // Retrieve the list of titles from the DataSet cached in
            // the application state.
            DataSet titlesDataSet = (DataSet)Application["TitlesDataSet"];

            if (titlesDataSet != null) {
                return titlesDataSet.Tables["Title"].DefaultView;
            }
            else {
                return null;
            }
        }

        private void LoadTitlesGrid() {
```

```
            // retrieve the data from the database
            ICollection titlesList = GetTitlesList();

            // set the control's datasource
            titlesGrid.DataSource = titlesList;

            // and have it build its items using the datasource
            titlesGrid.DataBind();
        }

        protected override void OnLoad(EventArgs e) {
            base.OnLoad(e);

            if (!IsPostBack) {
                // first request for the page
                LoadTitlesGrid();
            }
        }
    }
}
```

这个.cs 文件包含了页面的所有代码。在功能上，这些代码和上一节的例子非常相似。通过重载页面的 OnLoad 方法，获得数据并绑定到 DataGrid 控件，实现了数据的显示。DataBind 方法被调用时，DataGrid 控件会根据 DataSet 的 DataTable 的每一行，创建表格的每一行。当用户提交表单时，数据绑定不再被调用，控件将根据其原来的状态重新绘制每一个项目。

DataGrid 的 AutoGenerateColumns 属性缺省是 True。当 AutoGenerateColumns 为 True 时，DataGrid 将检查其数据源和其对象映射，并为每一个共有属性或者字段创建一个列。本例中，DataGrid 控件把 DataSet 中的每一个字段显示为一个列。DataGrid 的这种功能使得程序员使用很少的代码就可以使用 DataGrid 控件。

每一个自动产生的列称为一个 BoundColumn（绑定列）。绑定列根据其数据表对应列的数据类型，自动将其转化为一个字符串，显示在表格的一个单元中。

我们来看看改进后的代码：
(dg02.aspx)

```
<%@ Page language="C#" src="DataGrid.cs" inherits="Samples.DataGridPage"%>
...

<asp:DataGrid runat=server id="titlesGrid"
      AutoGenerateColumns="false">
  <property name="Columns">
```

```
      <asp:BoundColumn HeaderText="Title" DataField="title"/>
      <asp:BoundColumn HeaderText="Author" DataField="au_name"/>
      <asp:BoundColumn HeaderText="Date Published" DataField="pubdate"/>
      <asp:BoundColumn HeaderText="Price" DataField="price"/>
   </property>
</asp:DataGrid>
```

dg02.aspx 展示了用户自定义列集合的应用。由于采用了 code-behind 技术，DataGrid.cs 可以不加任何修改。

这里，DataGrid 的 AutoGenerateColumns 设置为 false，不允许控件自动创建列集合。这样，DataGrid 将应用用户定义的列集合来表现 DataSet 到一个表格中。

这样做有什么好处呢？

- 你可以控制列的顺序。表格的列将按照你给定的顺序排列。而相反地，自动产生的列将按照数据被存取的次序来创建，由于数据被存取的次序是不可指定的，他可能有别于代码中指定的顺序或者数据库中的顺序。
- 每一列的标题都可以指定。这可以通过指定其 HeaderText 属性来实现。在 dg01.aspx 中，列的标题缺省为字段名。在很多情况下，这不是你想要的。当然，你还可以使用 BoundColumn 的其他属性。
- 自动产生的列总是 BoundColumn 类型。而指定列允许使用继承了 BoundColumn 的用户控件。

Dg03.aspx 是进一步的改进版本，它显示了如何控制 DataGrid 的外观表现和各个项目的格式化控制。

(dg03.aspx)

```
<%@ Page language="C#" src="DataGrid.cs" inherits="Samples.DataGridPage"%>
...

<asp:DataGrid runat=server id="titlesGrid"
      AutoGenerateColumns="false"
      Width="80%"
      BackColor="White"
      BorderWidth="1px" BorderStyle="Solid" CellPadding="2" CellSpacing="0"
      BorderColor="Tan"
      Font-Name="Verdana" Font-Size="8pt">
   <property name="Columns">
      <asp:BoundColumn HeaderText="Title" DataField="title"/>
      <asp:BoundColumn HeaderText="Author" DataField="au_name"/>
      <asp:BoundColumn HeaderText="Date Published" DataField="pubdate"
            DataFormatString="{0:MMM yyyy}"/>
      <asp:BoundColumn HeaderText="Price" DataField="price"
        DataFormatString="{0:c}">
```

```
        <property name="ItemStyle">
          <asp:TableItemStyle HorizontalAlign="Right"/>
        </property>
      </asp:BoundColumn>
    </property>

    <property name="HeaderStyle">
      <asp:TableItemStyle BackColor="DarkRed" ForeColor="White"
        Font-Bold="true"/>
    </property>
    <property name="ItemStyle">
      <asp:TableItemStyle ForeColor="DarkSlateBlue"/>
    </property>
    <property name="AlternatingItemStyle">
      <asp:TableItemStyle BackColor="Beige"/>
    </property>
</asp:DataGrid>
```

和前面的例子一样。不同的是，这里我们对 DataGrid 的样式属性进行了控制，从而得到了更好的表格外观。和 dg02.aspx 一样，DataGrid.cs 不许要作任何的改进。

由于 DataGrid 是从 WebControl 继承来的，所以它也具有 Width、BackColor、BorderStyle、Font 等样式属性。此外，DataGrid 还具有 CellPadding 等和表格关联的特殊属性。这些属性使程序员可以完全控制 DataGrid 的样式和表现。

这里还使用了 HeaderStyle 和 AlternatingItemStyle，这是和 DataGrid 项目相关的样式属性。这些属性可以控制表格项目的样式。本例中，表格的偶数行和奇数行具有同样的前景色，但是，偶数行的背景色不同于奇数行。本例还控制了 Price 一列的样式，使文本靠右对齐。

DataGrid 还支持对表格单元的格式化控制。这是通过设置 BoundColumn 的 DataFormatString 属性来实现的。这样，表格单元的内容将被 String.Format 方法所格式化。如果你不指定 DataFormatString 属性，缺省的 ToString 方法被调用。

Dg04.aspx 显示了如何选择表格的一行：
(dg04.aspx)

```
<%@ Page language="C#" src="DataGrid4.cs" inherits="Samples.DataGrid4Page"%>
...

<asp:DataGrid runat=server id="titlesGrid"
      AutoGenerateColumns="false"
      Width="80%"
      BackColor="White"
      BorderWidth="1px" BorderStyle="Solid" CellPadding="2" CellSpacing="0"
      BorderColor="Tan"
```

```
        Font-Name="Verdana" Font-Size="8pt"
        DataKeyField="title_id"
        OnSelectedIndexChanged="OnSelectedIndexChangedTitlesGrid">
    <property name="Columns">
      <asp:ButtonColumn Text="Select" Command="Select"/>
      <asp:BoundColumn HeaderText="Title" DataField="title"/>
      <asp:BoundColumn HeaderText="Author" DataField="au_name"/>
      <asp:BoundColumn HeaderText="Date Published" DataField="pubdate"
            DataFormatString="{0:MMM yyyy}"/>
      <asp:BoundColumn HeaderText="Price" DataField="price"
        DataFormatString="{0:c}">
        <property name="ItemStyle">
          <asp:TableItemStyle HorizontalAlign="Right"/>
        </property>
      </asp:BoundColumn>
    </property>

    <property name="HeaderStyle">
      <asp:TableItemStyle BackColor="DarkRed" ForeColor="White"
        Font-Bold="true"/>
    </property>
    <property name="ItemStyle">
      <asp:TableItemStyle ForeColor="DarkSlateBlue"/>
    </property>
    <property name="AlternatingItemStyle">
      <asp:TableItemStyle BackColor="Beige"/>
    </property>
    <property name="SelectedItemStyle">
      <asp:TableItemStyle BackColor="PaleGoldenRod" Font-Bold="true"/>
    </property>
</asp:DataGrid>
...
<asp:Label runat=server id="selectionInfoLabel" Font-Name="Verdana" Font-Size="8pt"/>
```

本例中，DataGrid 的 SelectedIndexChanged 事件被处理，代码封装在下面的.cs 文件中。和前面的例子不同，我们增加了一个具有“select”命令的按钮列。这就让 DataGrid 可以为每一行产生一个选择按钮。同时，SelectedItemStyle 属性也被设置，这样可以很清楚地标志当前的选择项目。最后，DataKeyField 属性得到指定，这是为了 code-behind 代码可以使用 DataKeys 集合。

现在我们来看看 code-behind 代码：

(DataGrid4.cs)

```
namespace Samples {
    ...
```

```
public class DataGrid4Page : Page {
    protected DataGrid titlesGrid;
    protected Label selectionInfoLabel;

    public ICollection GetTitlesList() {
        // Retrieve the list of titles from the DataSet cached in
        // the application state.
        DataSet titlesDataSet = (DataSet)Application["TitlesDataSet"];

        if (titlesDataSet != null) {
            return titlesDataSet.Tables["Title"].DefaultView;
        }
        else {
            return null;
        }
    }

    private void LoadTitlesGrid() {
        // retrieve the data from the database
        ICollection titlesList = GetTitlesList();

        // set the control's datasource and reset its selection
        titlesGrid.DataSource = titlesList;
        titlesGrid.SelectedIndex = -1;

        // and have it build its items using the datasource
        titlesGrid.DataBind();

        // update the selected title info
        UpdateSelectedTitleInfo();
    }

    protected override void OnLoad(EventArgs e) {
        base.OnLoad(e);

        if (!IsPostBack) {
            // first request for the page
            LoadTitlesGrid();
        }
    }

    // Handles the OnSelectedIndexChanged event of the DataGrid
    protected void OnSelectedIndexChangedTitlesGrid(object sender,
```

```
                                                    EventArgs e) {
            UpdateSelectedTitleInfo();
        }

        private void UpdateSelectedTitleInfo() {
            // get the selected index
            int selIndex = titlesGrid.SelectedIndex;
            string selTitleID = null;
            string selectionInfo;

            if (selIndex != -1) {
                // display the key field for the selected title
                selTitleID = (string)titlesGrid.DataKeys[selIndex];
                selectionInfo = "ID of selected title: " + selTitleID;
            }
            else {
                selectionInfo = "No title is currently selected.";
            }

            selectionInfoLabel.Text = selectionInfo;
        }
    }
}
```

.cs 文件包含了 SelectedIndexChanged 事件的处理逻辑，和显示当前选择的 ID 所需要的代码。当用户点击选择按钮时，SelectedIndexChanged 事件就被激发。这时，DataGrid 的标准命令”Select”被识别，其 SelectIndex 属性相应改变，然后，OnSelectedIndexChangedTitlesGrid 得到执行。

在 OnSelectedIndexChangedTitlesGrid 函数中，我们调用了 UpdateSelectedTitleInfo 方法。此方法负责显示当前选择项目的信息，此处为简单地显示 ID 号。当然，你可以根据这个 ID 关联的行，从而显示更多的信息。

ID 是通过存取 DataKeys 集合获得的。因为在 ASPX 中指定了 DataKeyField 属性，我们可以使用这个集合。典型地，这个字段就是表的主键或者能够唯一确定一行的某个字段。根据这个字段，就可以获得当前选择项目的更具体信息。

本例显示了如何在显示数据之外，实现对数据的选择操作。DataGrid 还具有很多其他特性，比如排序、分页、编辑、列模板等等。这里就不详细展开了。

3.6.3.4 Repeater, DataList, or DataGrid?

Repeater, DataList, 和 DataGrid 控件基于同样的编程模型。同时，每个控件又为着不同

的目标而设计，所以，选择合适的控件非常重要。

从对象层次图可以看出，Repeater 是最轻最小的控件，它仅仅继承了基本控件的功能，包括 ID 属性、子控件集合等。另一方面，DataList 和 DataGrid 则继承了 WebControl 功能，包括样式和外观属性。

从对象模型看，repeater 是最简单的控件，它也是最小的数据绑定控件，它没有外观，也不表现为任何特定的用户界面。Repeater 也支持模板。但它不支持内建的样式和外观属性。如果你需要完全控制页面，用 repeater 是一个最合适的选择。

DataList 具有 repeater 的功能，并支持外观控制。它继承了 WebControl 的外观特性，并增加了一些样式属性，以控制其子控件的外观。DataList 也支持对项目的标准操作，比如选择、编辑、删除。当需要产生横向或纵向的一系列项目时，采用 DataList 是最合适的。

DataGrid 控件实现了表格样式的列和行。和 DataList 类似，它也支持外观和样式控制。除了支持对项目的选择、编辑等操作，DataGrid 还支持对整个集合的操作，包括分页、排序等等。DataGrid 和 DataList 的最大不同在于，DataGrid 不包含任何模板属性，这意味着项目或者表格的行不是模板化的。但是，通过加入 TemplateColumn 到某个列，你可以在列上使用模板。

下表概括了列表控件的主要功能：

功能	Repeater	DataList	DataGrid
模板支持	Yes (必须)	Yes (必须)	在列中应用 (可选)
表格外观	No	No	Yes
流式布局	Yes	Yes	No
列表 / 报纸样式布局	No	Yes	No
样式和外观属性	No	Yes	Yes
项目选择	No	Yes	Yes
项目编辑	No	Yes	Yes
删除	No	Yes	Yes
分页	No	No	Yes
排序	No	No	Yes

3.6.4 小结

本章主要讲述了如何把取得的 DataSet 对象和其他的数据绑定控件相结合，产生我们所期望的页面表现模式的方法。数据绑定控件的使用，不外是首先对 DataSource 指定数据的来源，然后使用 DataBind()方法把数据绑定到控件上。比较麻烦一点的是控件模板的定义，

它的使用方法比较灵活，谁也不可能把它一一列举，不过我个人认为基础在于 ItemTemplate 模板，只要掌握了它，其余的都是细节。

第四篇　应用程序

第一章 什么是应用程序

在 asp.net 中，可以这样来定义一个 Application：能够在一个 web 应用服务器的子目录或者虚拟目录上运行的所有的文件、页面、操作、模块或者能被执行的代码。比方说，在一个 web 服务器上，一个“order“应用程序将会在“/order“这个目录下被发布。

Web 服务器上的 asp.net 应用程序在一个被称作应用程序域运行空间(AppDomain)环境中被执行，以保证类的隔离（没有版本、名称上的冲突）、安全屏蔽（防止有权访问某些机器/网络的资源）、静态变量的隔离。

在一个 web 应用程序的生命周期中，asp.net 维护一个 HttpApplication 实例池。Asp.net 对一个 Http 的请求会自动分配一个来处理，这个特别的 HttpApplication 实例对管理这个在全部的生命周期里的请求是可靠的，并且在处理完成后可以被重用。

在应用程序环境下，ASP.NET 并发处理客户端的请求，所以可能存在多线程对 Application 对象的同时存取。在这种情况下，对 Application 对象的草率处理，可能会导致不可预知的错误。例如以下代码：

```
<% Application("counter") = CType(Application("counter") + 1, Int32) %>
```

原本希望对实例进行计数，但如果同时到达两个以上请求时，则有可能产生漏计。正确的方法应该是在操作以前，对 Application 对象上锁，操作完成以后，再对 Application 对象解锁。代码如下：

```
<%
Application.Lock()
Application("counter") = CType(Application("counter") + 1, Int32)
Application.UnLock()
%>
```

4.1.1 配置应用程序的步骤

4.1.1.1 设置应用程序的目录结构

一个 WEB 站点可以有多个应用程序运行，而每一个应用程序可以用唯一的 URL 来访问，所以首先应利用 IIS 开放应用程序的目录为“虚拟目录”。各个应用程序的“虚拟目录”可以不存在任何物理上的关系。例如：

应用 URL:	物理路径：
http://www.my.com	c:\inetpub\wwwroot
http://www.my.com/myapp	c:\myapp
http://www.my.com/myapp/myapp1	\\computer2\test\myapp

从“虚拟目录”上看来，http://www.my.com/myapp 和 http://www.my.com/myapp/myapp1 是乎存在某种联系，但实际情况却是，我们看到两者完全分布于不同的机器上，更不用说物理目录了。

4.1.1.2. 设置相应的配置文件

根据应用的具体需要，可以拷入相应的 global.asax 和 config.web 配置文件，并且设置相应的选项。（配置文件的设置具体见相关章节）

global.asax 主要配置 application_start、applicatoin_end、session_start、session_end 等事件。

4.1.1. 3. 把应用所涉及的各种文件放入“虚拟目录“中

把.aspx 文件、.ascx 文件以及各种资源文件分门别类放入应用目录中，把类引用所涉及的集合放入应用目录下的 bin 目录中。

4.1.2 应用程序框架

```
<%@ Application attribute="value" [attribute=value … ]%>
<%@ Import namespace="value" %>…
<%@ Assembly Name="assemblyname" %>

<script language=”vb”  runsat=server>
        …
</script>

<body>
    <form runat=server>
        …
    </form>
</body>
```

</html>

说明：

1．<%@ Application attribute="value" [attribute=value …]%>

让 ASP.NET 运行环境动态从另一个应用中动态编译出一个类来继承使用。

例如：

<%@ Application Inherits="MyApp.Object" Description="Ourapp" %>

指定应用环境从 Myapp 应用中动态编译一个 MyApp.Object 的类以供使用，它的说明为“Ourapp”。

2．<%@ Import namespace="value" %>…

显视导入一个命名空间到应用程序，这样应用程序就可以使用命名空间中定义的各种类和接口来完成特定的功能，大大加快了程序的开发速度。

例如：

<%@ Import Namespace="System.IO" %>

<%@ Import Namespace="System.NET” %>

就可以利用系统为我们提供的大量文件和网络对象，快速的开发自己的文件和网络应用程序。

3．<%@ Assembly Name="assemblyname" %>

在页面编译时产生到 assemblyname 的连接，这样就可以使用集合中类及接口。缺省情况下，应用会把应用程序目录下 bin 中的集合都动态载入。该项功能也可以在应用程序的 config.web 中配置，缺省情况下，config.web 中有如下形式：

<assembly>

<add assembly="*"/>

</assembly>

即缺省情况下，加载 bin 下的所有集合。

又如：

<%@ Assembly Name="myassembly.dll" %>

加载 bin 下 myassembly.dll 集合

4．其他

<script>、</script>对之间的代码通常是各种事件的定义，诸如页面开始时、某个按件被触发时所要做的事情。<body>、</body>和<form>、</form>之间通常是页面的界面要素，为显示给客户端的可视界面。

4.1.3 创建应用程序的典型步骤

4.1.3.1 配置 config.web

主要定义为 gb2312 字符集，以利于中文显示

```
<configuration>
<globalization requestencoding="gb2312" responseencoding="gb2312" />
</configuration>
```

4.1.3.2 配置 global.asax

主要定义应用初始化、结束，会话开始、结束，请求开始、结束等事件发生时，应用要做的事情。

```
<script language="VB" runat="server">
    Sub Application_Start(Sender As Object, E As EventArgs)
    End Sub

    Sub Application_End(Sender As Object, E As EventArgs)
    End Sub

    Sub Session_Start(Sender As Object, E As EventArgs)
    End Sub

    Sub Session_End(Sender As Object, E As EventArgs)
    End Sub

    Sub Application_BeginRequest(Sender As Object, E As EventArgs)
    End Sub

    Sub Application_EndRequest(Sender As Object, E As EventArgs)
    End Sub

</script>
```

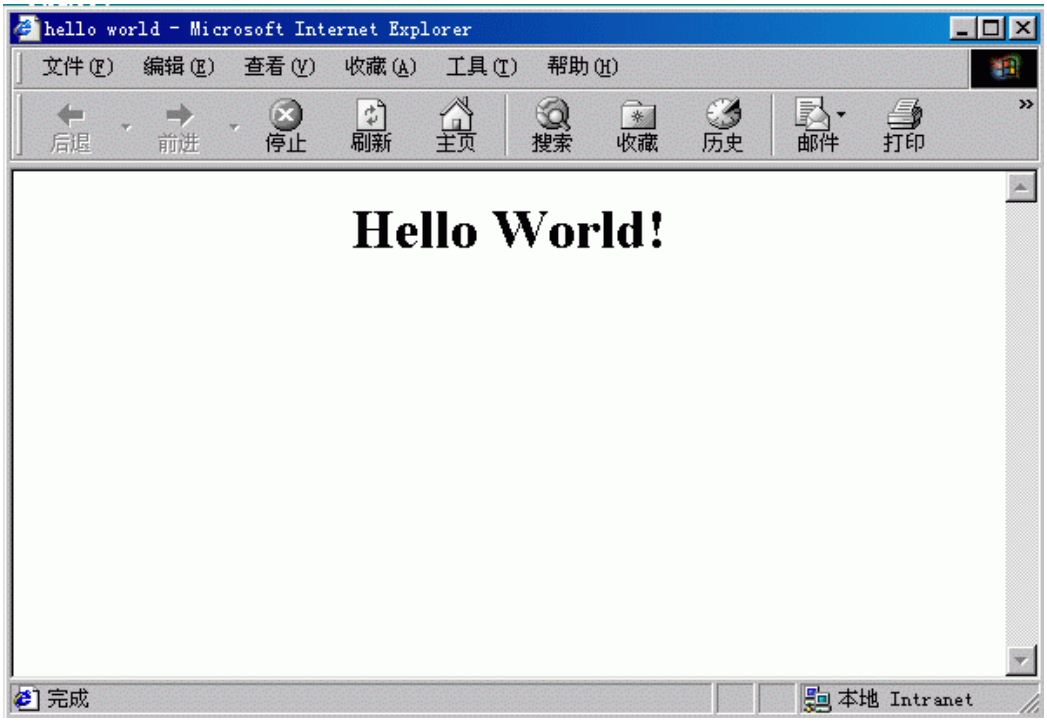
4.1.3.3 主程序

创建一个应用程序我们可以先在 web 服务器上创建一个虚拟目录或者在发布目录下创建一个新的目录。装过 Windows 2000 Advance Server 的读者会知道，安装完成后，会有一个 c:/inetpub/wwwroot 的目录，你可以通过 IIS 管理工具来创建一个新的目录或者虚拟目录。一个应用程序可能含有大量的.aspx 文件、.ascx 文件、由其他工具产生的 assembly 集合以及页面中用到的各种资源文件（声音、图片、动画等等），这里就不再一一介绍了。

下面我们就创建一个简单的 aspx 页面来说明一个 Application 的应用，它只含有一个.aspx 文件，在用户浏览时显示“hello world”，可谓最简单的 web 应用了。

```
<!-- 文件名：application/FormAppHellp.aspx -->
<%@Page Language="VB"%>
<html>

 <head>
  <title>
    hello world
  </title>
 </head>

 <body>
  <center>
   <h1><% Response.Write("Hello World!") %></h1>
  </center>
 </body>

</html>
```

下面就是我们的运行效果：



4.1.4 小结

asp.net 平台的应用，通过指定虚拟目录，使得一个主机多个地址多个应用成为可能。采用 asp.net 开发应用程序带来的好处是：程序集中可方便打包，配置的层次结构更加灵活方便，应用独立运行于自身的应用环境中更加安全可靠。

配置一个应用的过程大致为：1）指定应用目录为 IIS 的虚拟目录 2）为应用设置适当的配置权限（配置 global.asax 和 config.web 文件）3）在自己的应用目录下放置事先编好的程序。

从页面应用所支持的 Application、Import、Assemly 等标识看来，asp.net 对对象重用的支持大大加强了，ASP.NET 的“通用语言运行库”概念的提出，为实现各种开发语言的合作编程奠定了基础。

第二章 配置 Config.web

4.2.1 ASP.NET 配置简介

ASP.NET 提供了一个丰富而可行的配置系统，以帮助管理人员轻松快速的建立自己的 WEB 应用环境。ASP.NET 提供的是一个层次配置架构，可以帮助 WEB 应用、站点、机器分别配置自己的扩展配置数据。ASP.NET 的配置系统具有以下优点：

- ASP.NET 允许配置内容可以和静态内容、动态页面和商业对象放置在同一应用的目录结构下。当管理人员需要安装新的 ASP.NET 应用时，只需要将应用目录拷贝到新的机器上即可。
- ASP.NET 的配置内容以纯文本方式保存，可以以任意标准的文本编辑器、XML 解析器和脚本语言解释、修改配置内容。
- ASP.NET 提供了扩展配置内容的架构，以支持第三方开发者配置自己的内容。
- ASP.NET 配置文件的更修被系统自动监控，无须管理人员手工干预。

4.2.2 配置文件的规则

ASP.NET 的配置文件是基于 XML 格式的纯文本文件，存在于应用的各个目录下，统一命名为“config.web”。它决定了所在目录及其子目录的配置信息，并且子目录下的配置信息覆盖其父目录的配置。

WINNT\Microsoft.NET\Framework\版本号\下的 config.web 为整个机器的根配置文件，它定义了整个环境下的缺省配置。

缺省情况下，浏览器是不能够直接访问目录下的 config.web 文件。

在运行状态下，ASP.NET 会根据远程 URL 请求，把访问路径下的各个 config.web 配置文件叠加，产生一个唯一的配置集合。举例来说，一个对 URL: http://localhost\webapp\owndir\test.aspx 的访问,ASP.NET 会根据以下顺序来决定最终的配置情况：

１．.\Microsoft.NET\Framework\v.1.00\config.web (缺省配置文件)
２．.\webapp\config.web (应用的配置)
３．.\webapp\owndir\config.web (自己的配置)

4.2.3 配置文件的语法规则

１）标识

配置内容被置于 config.web 文件中的标记<configuration>和</configuration>之间。

格式：

```
<configuration>
    配置内容
    …
</configuration>
```

2)配置段句柄说明

ASP.NET 的配置文件架构并未指定任何文件格式或者是支持的配置属性。相反的，它提出了“配置段句柄申明”的概念来支持任意的用户定义配置段。

格式：

```
<configsections>
    <add name=欲定义配置段名  type=处理的句柄函数 />
</configsections>
```

３）配置段

具体定义配置的内容，供应用使用。

以下例子定义了一个“httpmodules”配置段，设置了系统 http 相关的处理模块

```
<configuration>

    <configsections>
        <add   name="httpmodules" type="System.Web.Configuration.HttpModules
ConfigurationHandler" />
    </configsections>

    <httpmodules>
        <add type="System.Web.SessionState.CookielessSessionModule" />
        <add type="System.Web.Caching.OutputCacheModule" />
        <add type="System.Web.SessionState.SessionStateModule" />
        <add type="System.Web.Security.WindowsAuthenticationModule" />
        <add type="System.Web.Security.CookieAuthenticationModule" />
        <add type="System.Web.Security.PassportAuthenticationModule" />
        <add type="System.Web.Security.CustomAuthenticationModule" />
        <add type="System.Web.Security.UrlAuthorizationModule" />
        <add type="System.Web.Security.FileAuthorizationModule" />
    </httpmodules>

</configuration>
```

4.2.4 ASP.NET 定义的标准配置段

1）httpmodule 段： 定义了应用的 http 请求的处理模块以及诸如安全、日志之类的应用方式
2）httphandlers 段： 负责映射 URLs 到 IhttpHandler 类
3）sessionstat 段： 负责配置 http 模块的会话状态
4）globalization 段： 配置应用的公用设置
5）compilation 段： 配置 ASP.NET 的编译环境
6）trace 段： 配置 ASP.NET 的跟踪服务
7）security 段： ASP.NET 的安全配置
8）iisprocessmodel 段： 在 IIS 上配置 ASP.NET 的处理模式
9）browercaps 段： 配置浏览器的兼容部件

4.2.5 一个配置读出的例子

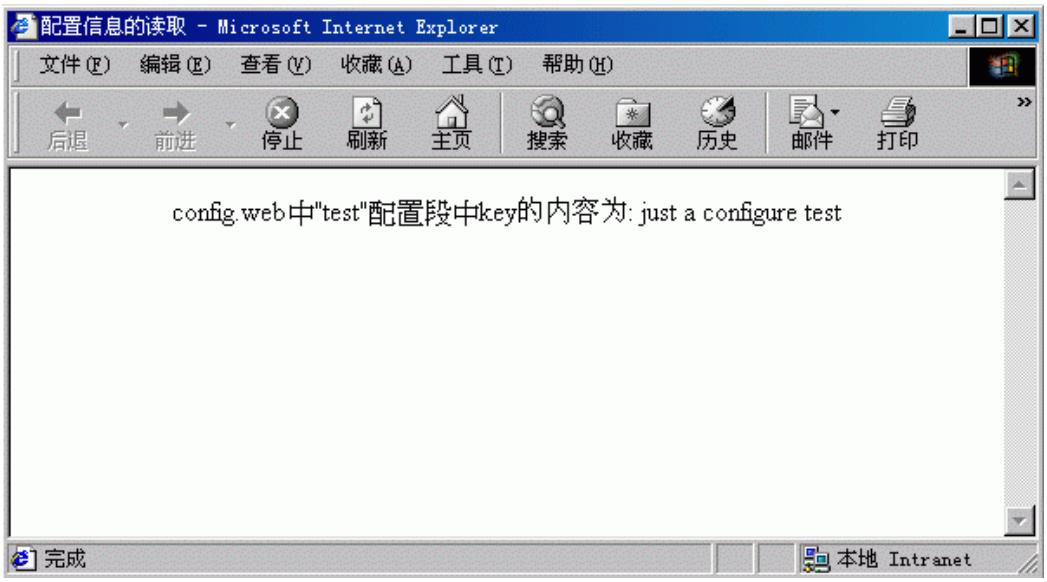
1）config.web 配置文件

```
<!--config.web 请放入 FormCfg.aspx 所在目录-->
<configuration>
<!--申明一个 test 配置段-->
    <configsections>
        <add name="test" type="System.Web.Configuration.DictionarySectionHandler" />
    </configsections>

    <test>
<!--配置一个键 key,其内容为 just a configure test-->

        <add key="key" value="just a configure test" />
    </test>

</configuration>
```

2）读出其内容

```
    <!--文件名：Application/FormCfg.aspx-->
    <html>
     <head>
    <script language="VB" runat=server>
    sub page_load(s as object ,e as eventargs)
    '取出 test 配置段的 key 键的值
            Dim CfgSection As Hashtable = Context.GetConfig("test")
            Dim Msg As String = CStr(CfgSection("key"))

        lblMsg.text=Msg
    end sub
    </script>
     <title>
    配置信息的读取
  </title>
      </head>

    <body>
    <center>
    config.web 中"test"配置段中 key 的内容为:
    <asp:label id=lblmsg runat=server />
    </center>
    </body>

    </html>
```

3）运行结果



4.2.6 Config.web 配置实例

```
<configuration>
<!--定义用户应用的公用设置，如 SQL 的 sql 连接串等等-->
<appsettings>
</appsettings>

<!--设置浏览器的兼容性部件-->
<browsercaps>
</browsercaps>

<!--编译环境设置，非调试模式-->
<compilation debugmode="false">
<!--缺省编译语言为 vb，以后可以不再在 Page 中定义脚本语言-->
<compilers defaultlanguage="vb">
<!--以 MSVSA.dll 编译.vb 为后缀的ＶB 文件-->
<compiler language="VB" extension=".vb" type="MSVSA.dll#Microsoft.VB.Compiler"/>
</compilers>

<assemblies>
<!--加入对 System.Data 的引用-->
<add assembly="System.Data" />
<!--去掉对 System.Data 的引用-->
<remove assembly="System.IO" />
<!--去掉 config.web 中包含或继承来的引用-->
<clear />
```

```
</assemblies>

</compilation>

<!--设置应用全局环境-->
<!--文件、请求、返回以 gb2312 编码，以保证浏览器正确显示中文-->
<globalization fileencoding="gb2312"   requestencoding="gb2312"
responseencoding="gb2312"/>

<!--定义用户出错的处理-->
<!--出错缺省显示 defaultredirect 指定的页面,mode 为 on 时，遵循 customerrors 配置段-->
<!--mode 为 off 时,忽略用户出错，mode 为 remoteonly 时，本地才显示真正的出错原因-->
<customerrors defaultredirect="AnErrorHasOccured.aspx?ErrNum=-1" mode="remote">
<!--当出错码为 500 时,显示 redirect 指定的页面-->
<error statuscode="500" redirect="AnErrorHasOccured.aspx?ErrNum=500"/>
</customerrors>

<!--指定目录 webapp 的访问权限-->
<location path="webapp" >
<!--非授权用户不能进入 webapp 目录-->
<security>
<authorization>
<deny users="?" />
</authorization>
</security>
</location>

<!--定义安全属性-->
<security>
<authorization>
<!--角色为 Adminstrators 和所有的用户访问其指定的资源-->
<allow roles="Adminstrators"/>
<allow users="*" />
</authorization>
</security>

</configuration>
```

4.2.7 小结

Config.web 是 aspx 区别于 asp 的一个方面，我们可以用这个文件配置我们的很多信息。

第三章 编写 global.asax

为了编写用户界面的应用程序，开发者可以把应用程序标准的逻辑和时间处理的代码加到 Web Application 里面。这些代码不产生用户界面，也不想英单个得页面的请求。事实上，这些代码处理更高水平的事件，如 **Application_Start**，**Application_End**，**Session_Start**，**Session_End**，等等。开发者通过放在 web 应用程序根目录下面的 Global.asax 来响应这些事件。

Asp.net 通过一个动态的.NET FrameWork 类自动解析和编译这个文件，这个类就是 HttpApplication 基类，在第一时间里面，在这个文件里面的应用程序的资源将会被响应。

首先，在包含有请求的应用程序名字空间中被访问之前，Global.asax 将被解析和编译成.NET Framework 的一个类。这个文件本身有拒绝被访问的配置。

下面我们来看看这个文件里面的具体内容，首先我们声明这个文件的使用语言、运行环境：

```
<script language="VB" runat=server>
'相关方法
</script>
```

然后我们就可以定义各种方法了，

```
Sub Application_Start()
    '方法的属性等
End Sub
```

如果事件处理代码需要用到名字空间，我们可以这样来引用它：

```
<%@  Import  Namespace="System.Data.SQL"%>
```

下面我们来看看这个文件的具体应用，首先我们在我们的 Web Server 上建立一个 Global.asax 文件，我们在里面加上我们的代码：

```
<script language="VB" runat=server>
  '相关方法

    Sub Application_Start()
        '方法的属性等
    End Sub

    Sub Application_Start(Sender As Object, E As EventArgs)
        Application.Lock()
        Application("counter") = CType(Application("counter") + 1, Int32)
```

```
    Application.UnLock()
  End Sub

    Sub Application_End(Sender As Object, E As EventArgs)
        ' Clean up application resources here
    End Sub

    Sub Session_Start(Sender As Object, E As EventArgs)
        Response.Write("Session 正在启动...<br>")
    End Sub

    Sub Session_End(Sender As Object, E As EventArgs)
        ' Clean up session resources here
    End Sub
</script>
```

当然，我们还要配置 Config.web，用来指定出错信息的打印页面。根据上面我们配置 Config.web 的经验，我们很容易的就可以对这个文件进行配置：

```
<configuration>
<customerrors mode="on" defaultredirect="error.htm" />
<globalization requestencoding="gb2312" responseencoding="gb2312" />
</configuration>
```

第二句话就是配置我们指定的出错页面语句。我们写两个页面来实现它，一个为出错页面，一个为实现这个功能的 aspx 页面。出错页面很简单的，就是报告程序出错时显示的信息，我们就写“在 config.web 里面配置的连接！“，是经过 aspx 页面甩出来的。

在 aspx 页面，我们用下面的语句来响应出错按钮点击事件：

```
Sub Error_Click(Sender As Object, E As EventArgs)
        ‘甩出异常！
        throw New Exception()
End Sub
```

以外我们的响应 Session 的方法用下面的语句来说明：

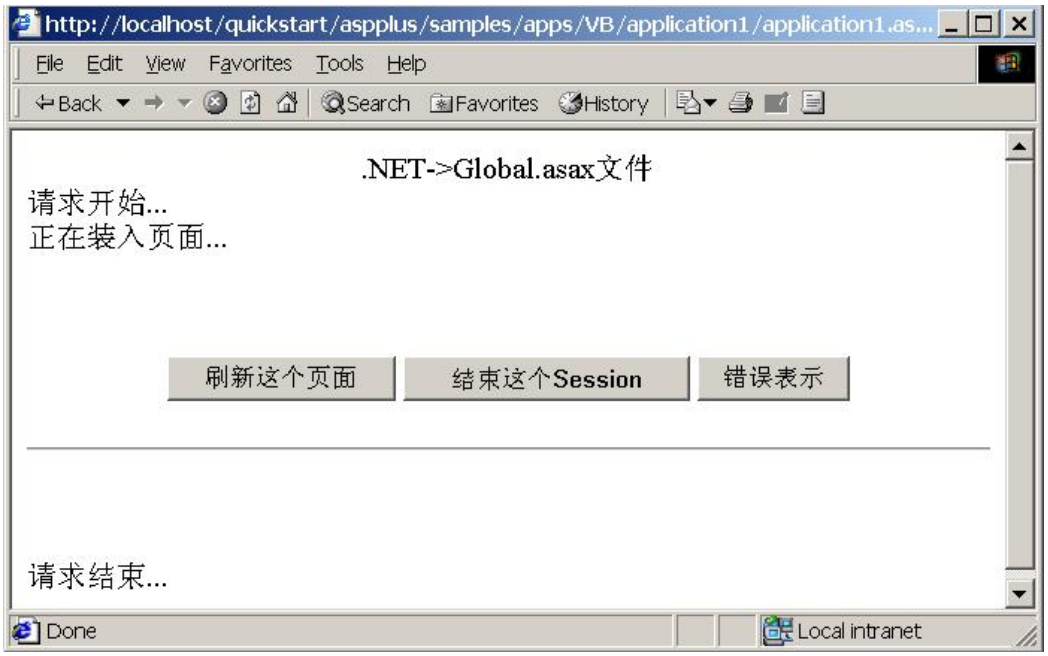
```
Sub Session_Click(Sender As Object, E As EventArgs)
    Session.Abandon()
    Response.Redirect("global.aspx")
End Sub
```

下面是完整的代码：

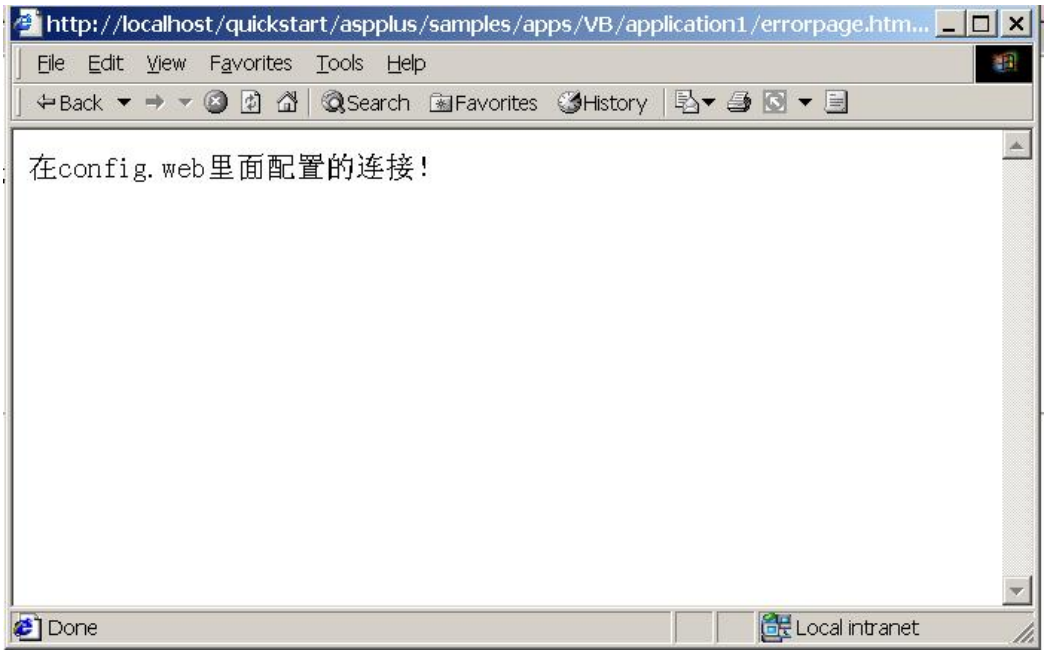
```
<html>
```

```
    <script language="VB" runat="server">
        '页面导入
        Sub Page_Load(Sender As Object, E As EventArgs)
            Response.Write("正在装入页面...<br>")
        End Sub

        'Session 事件
        Sub ssaidy(Sender As Object, E As EventArgs)
            Session.Abandon()
            Response.Redirect("global.aspx")
        End Sub

        '响应错误方法
        Sub esaidy(Sender As Object, E As EventArgs)
       '抛出异常
            throw New Exception()
        End Sub
    </script>
    <body>
     <br><br><br>
     <center>
      <form runat="server">
        <input type="submit" Value="刷新这个页面" runat="server"/>
        <input type="submit" OnServerClick="ssaidy" Value="结束这个Session"
               runat="server"/>
        <input type="submit" OnServerClick="esaidy" Value="错误表示" runat="server"/><p>
        <hr>
      </form>
      </center>
      <br><br>
    </body>
</html>
```

运行结果如下：



点击“错误表示“按钮，显示如下：



4.3.1 小结

讲述了配置文件 Global.asax 的配置问题，Global.asax 文件对一个.NET 技术构建的

WEB 站点来讲，是非常必须的，本章我们的内容就是针对它讲的。

第四章 Application 和 Session

4.4.1 Application 对象

在讲述 ASP.NET 的 Application 对象之前，我们先来回顾一下 ASP 的 Application 对象。我们知道由于变量的生命周期受限于网页，所以每当.asp 文件被解释执行完毕之后时，变量就会被释放，它的内容将不存在。而在编程过程中，我们有时又需要在页面之间传递变量的内容。例如，我们在一个登录页面中输入了用户的名字，为了使页面个性化，在后面的页面显示中，我们希望知道前面输入的用户名，以便于更好的人机交互。这就要求有一种变量传递的机制。人们最常用的保存变量的内容的方法是使用文件，但是毕竟对文件的的操作是比较麻烦的事情，有没有更简单的方法呢？其中一种比较简单的方法就是使用 Application 对象来保存我们希望传递的变量。由于在整个应用程序生存周期中，Application 对象都是有效的，所以在不同的页面中都可以对它进行存取，就像使用全局变量一样方便。在 asp.net 环境下，Application 对象来自 HttpApplictionStat 类。它可以在多个请求、连接之间共享公用信息，也可以在各个请求连接之间充当信息传递的管道。

4.4.1.1 使用 Application 对象

Application 对象重要的属性：

- **All 属性**，返回应用中保存的所有的公用对象数组
  例如：
  ```
  dim MyObjects() as object
  MyObjects=Application.All
  ```
  表示用 myobjects 取得了当前应用保存的所有对象

- **AllKeys 属性**，返回应用中保存的公用对象的名字数组（标识数组）
  例如：
  ```
  dim MyVars() as String
  MyVars=Application.AllKeys
  ```
  即取得了所有保存的公用对象的标识名字到 myvars 数组

- **Contents 属性**，返回 this 指针，主要是为了和 asp 兼容而保留

- **Count 属性**，返回当前应用中保存的公用对象的数目
  例如：
  ```
  dim VarNum as integer
  VarNum=Appliction.count
  ```

- **Item 属性**，返回当前应用中保存的公用对象集合中的指定对象，这是最常用的属性。

例如我们前面讨论的，记录变量内容的问题，就是通过 item 属性来保存的。

Appliction.Item(变量名)=要保存对象

但是通常我们都会省去 item 属性写成：

Application(变量名)=要保存对象

这里需要注意的是，Application 保存的对象为应用程序所共享，而.net 平台又是一个多用户多线程的环境，因而 Application 保存的对象在使用时，要注意避免冲突。

例如：

Application（”counter”）= Application（”counter”）+1

它使开始用户保存的数值加 1，我们可以利用它来统计页面浏览的次数。但是有一个问题发生了，那就是如果另外一个页面也使用了上述语句，那么混乱就产生了。设想一下如下情况，用户 a 对页面 a 访问，使 counter+1,然后用户 b 对页面 b 的访问，counter 又增加了 1，实际上无论对页面 a 还是页面 b，访问都只有一次，counter 却增加了 2 次，由于记数变量的相同使得我们统计页面的努力化为泡影。

• **StaticObjects 属性**，返回在应用程序文件中型如<object runat=server></object>定义的对象的集合。

下面，我们对上面学到的各种属性进行应用。

例子：

我们首先产生 6 个 Application 变量，然后分别用 item 属性和 all 属性去逐一取出各个 Application 变量的内容显示出来。注意为了避免其他公用 Application 变量的干扰，我们在页面加载时，调用了 removeall 方法，清空应用的所有公用变量。

1. 程序源代码

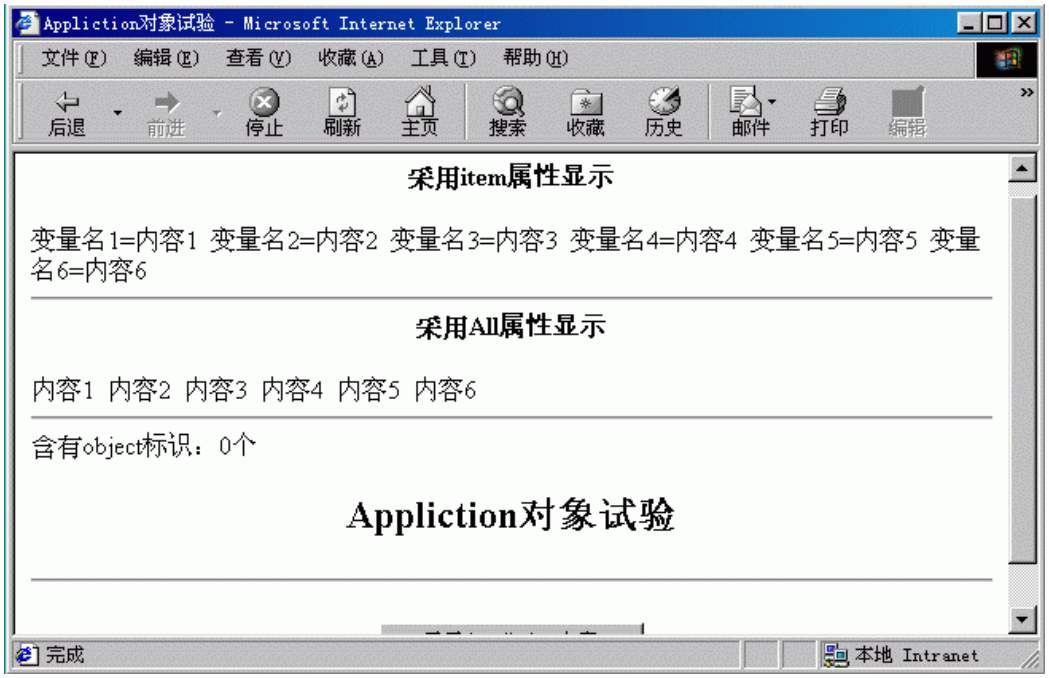
```
<!-- 文件名：Application\FormAppliction.aspx -->

<html>

 <script language="vb" runat=server>
   Sub Page_Load(o as object,e as eventargs)

     dim i as integer
     dim tStr as String
     dim sStr as String
     dim strArray() as String
     dim tObject() as Object
     dim ObCol as HttpStaticObjectsCollection

   If Not IsPostBack
         Application.removeall
    '为防止其他变量干扰，使用前清掉所有的保存变量

     '保存六个变量
         for i=1 to 6
```

```
        tStr="变量名" & i
              sStr="内容" & i
        Application(tStr)=sStr
            next
    Else
         '采用 item 属性遍历
         response.write("<center><b>采用 item 属性显示</b></center><br>")
         strArray=Application.Allkeys
         for i=1 to Application.count
         tStr= strArray(i-1) & "=" & Application.item(i-1)&"  "
         response.write(tStr)
         next

         '采用 All 属性遍历
         response.write("<hr><center><b>采用 All 属性显示</b></center><br>")
         tObject=Application.All
         for i=1 to Application.count
         tStr=tObject(i-1).ToString & "  "
         response.write(tStr)
         next

         '显示有多少个 object 定义
         ObCol=Application.StaticObjects
         response.write("<hr>含有 object 标识：" & ObCol.count & "个")

  End If
    End Sub
</script>

<head>
  <title>
  Appliction 对象试验
  </title>
</head>

<body bgcolor=#ccccff>
  <center>
    <h2>Appliction 对象试验</h2>
    <hr>
      <form runat=server>
      <asp:button text="显示 Appliction 内容" runat=server />
      </form>
  </center>
</body>
```

</html>

2．开始时，页面的输出画面

3．当按下显示按钮后，显示 appliction 内容的画面：



Application 对象重要的方法调用：

·**Add 方法**，加入一个对象到 Application 对象的 Stat 集合中

例如：

Application.Add(“string1”,”test”)

表示向 Application 的 stat 集合中加入一个名为 string1 的值为”test”的字符串，其实它的效果和

Application(“string1”)=”test”

以及 Application.item(“string1”)=”test”　　是一样的。

·**Remove 方法**,根据给出的标识从 Application 对象的 Stat 集合中删去

例如：

Application.Remove(“string1”)

表示把标识为 string1 的共享对象 string1 从 Application 对象的 Stat 集合中删去。使用它可以清除用 Add 方法添加的对象。

·**RemoveAll 方法**，把 Application 对象 Stat 集合中的所有对象清除，在我们对属性的使用举例中，我们已经见过了它的用法，但是值得小心，我们不提倡使用它，因为在编程中你并不清楚，是否有其他页面要依赖于某个 Application 的公用变量，一旦清除将造成不可预知的错误。

·**Clear 方法**，作用和 RemoveAll 方法一样。

·**Get 方法**，允许使用名字标识或者是下标，来取得 Application 对象 stat 集合中的对象元素。

例如：

dim tmp as object

tmp=Application.Get(“string1”)

或者 tmp=Application.Get(0)

表示从 Application 对象的 Stat 集合中取得标识为 string1 或者下标为 0 的对象

它等价于：

tmp=Application(“string1”)

tmp=Application(0)

或者是

tmp=Application.item(“string1”)

tmp=Application.item(0)

·**Set 方法**，修改 Application 对象 stat 集合中指定标识所对应的对象的值。

例如：

Application.Set(“string1”,”try”)

就把我们开始为 string1 变量设置的值”test”改为”try”了，它和下边的形式也是一样的：

Application(“string1”)=try

·**GetKey 方法**，根据给定的下标取得 Application 对象的 stat 集合中相应对象的标识名。例如：

dim nameStr as String
nameStr=Application.GetKey(0)
表示取得 Application 对象中 Stat 集合的第一个对象的标识名

·**Lock 方法**，锁住其他线程对 Application 对象中 stat 集合的访问权限。这个方法主要是用来防止对 Application 的变量操作过程中，其他并发程序可能造成的影响。比如在记数过程中，如果不进行上锁操作，就有可能发生脏读脏写。例如，开始从变量中取得记数值 1，如果在记数并写回到变量之间，另一页面对它发生了一次记数，并先行写回变量，那么最终写回到变量中的值为 2，而并不是实际的 3。如果采用了上锁机制，在页面读出变量到记数并写回变量的过程中，即使发生了另一次记数，由于变量被锁住，它也不可能在变量被写回以前取得成功，只有等待变量释放，从而形成两者对变量操作的串行性，避免了数据的脏读和脏写。

·**Unlock 方法**，对 Application 对象 Stat 集合锁定的解锁操作，释放资源以供其他页面使用。

下面我们就上边学到的方法做一个例子，为了强调 lock 方法和 unlock 方法，我们将单独举一个例子。例子是这样的，开始我们创建 6 个 Application 变量，赋以数值序号，页面有 3 个按钮，分别是加 1，减 1 和清除。当点击“加 1“按钮后，我们会看到变量的值都会增加 1，当点击”减 1“按钮后，变量值都减 1，当按下清除后，变量都消失了。在清除功能中，我们为了同时演示 remove 和 clear 方法，采用最后三个用 clear 清除，其他的逐一用 remove 清除。

1. 源程序

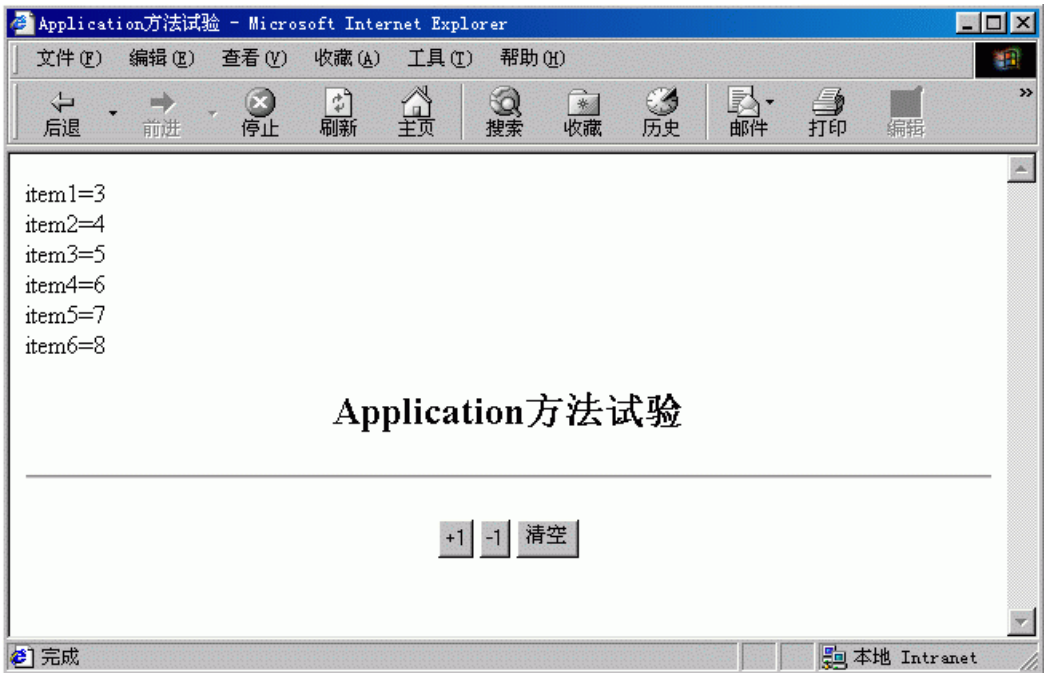
```
<!-- 文件名：Application\FormApplication01.aspx -->

<html>

 <script language="vb" runat=server>

   Sub Page_Load(o as object,e as eventargs)
   dim i as integer

    If Not IsPostBack
     for i=1 to 6
          application.add("item"&i,i)
          next
   End If

   response.clear
   for i=0 to application.count-1
```

```
  response.write(application.GetKey(i) & "=" & application.Get(i) & "<br>")
  next

 End Sub

Sub AddOne(s as object,e as eventargs)
'变量值加 1
 dim i as integer
 dim j as integer
 dim t as string

 for i=0 to Application.count-1
 j=Application.Get(i)+1
 t=Application.GetKey(i)
 Application.Set(t,j)
 next

Page_Load(s,e)
'刷新画面
End Sub

Sub SubOne(s as object,e as eventargs)
'变量减 1
 dim i as integer
 dim j as integer
 dim t as string

 for i=0 to Application.count-1
 j=Application.Get(i)-1
 t=Application.GetKey(i)
 Application.Set(t,j)
 next

 Page_Load(s,e)
 '刷新画面
End Sub

Sub Gone(s as object,e as eventargs)
'清空所有变量
 dim i as integer

 for i=0 to Application.count-3
 Application.Remove(i)
 next
```

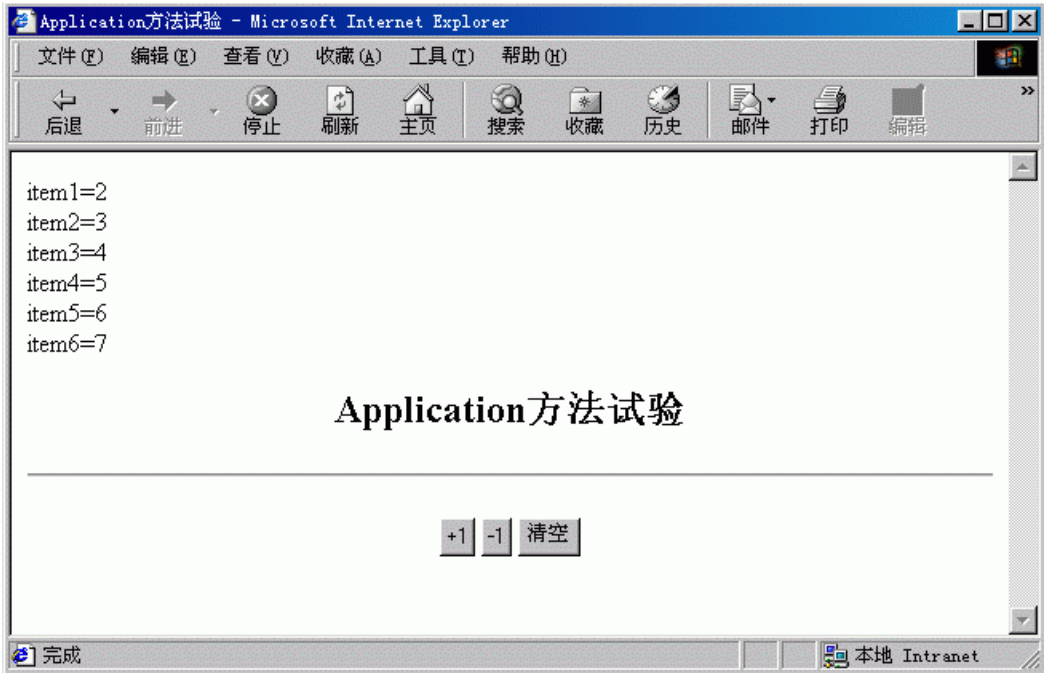
```
    '演示 remove 方法
    Application.clear
    '演示 clear 方法
    Page_Load(s,e)
    '刷新画面
  End Sub
 </script>
 <head>
  <title>
  Application 方法试验
  </title>
 </head>
 <body bgcolor=#ccccff>
  <center>
   <h2>Application 方法试验</h2>
   <hr>
    <form runat=server>
      <asp:button type="submit" text="+1" OnClick="AddOne" runat=server />
      <asp:button    type="submit" text="-1" OnClick="SubOne" runat=server />
      <asp:button    type="submit" text="清空" OnClick="Gone" runat=server />
    </form>
  </center>
 </body>
</html>
```

2．开始时，输出画面:

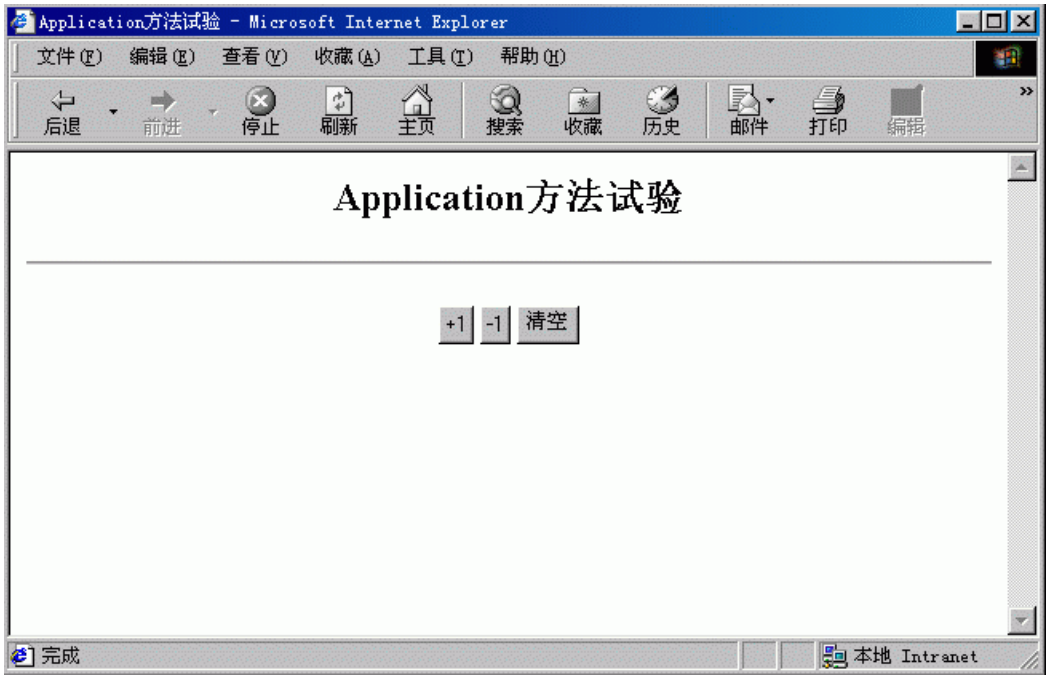
3. 当两次点击+1 后，输出的变量值：



4. 当点击”-1”后，变量的值为：



5. 当点击清空后，输出的画面



接下来，我们看一个使用 lock 和 unlock 方法制作计数器的例子：Application 对象对不同的连接者是共用的，因此适合制作计数器。

程序代码如下：

```
<!-- 文件名：application\FormAppLock.aspx -->
<html>
  <script language="vb" runat=server>
    sub Page_Load(o as object,e as eventargs)
      Application.Lock()
      Application("counter") = CType(Application("counter") + 1, Int32)
      Application.UnLock()
    end sub
</script>

<head>
  <title>
     Application 对象方法试验
  </title>
</head>

<body>
  <center>
```

```
  <h2>Application 对象 Lock 方法试验</h2>
  <hr>
你是第<%=Application("counter")%>位访问者!
 </center>
</html>
```

效果图：

关于计数器非同步更新的问题

假设有两个上网者同时启动网页（这种情况很常见），同时执行了上面的步骤，他们读出的值是相同的，但结果都是相同的。很明显，访问量少了一次。如何处理呢？我们在这里举一个例子，假设我们都在使用一个数据库管理系统，有可能访问者同时对一张表进行操作，如果一人在修改这张表的数据，而另一个人在读数据，很明显此时数据前后就不一致。通过我们需要对表进行锁定。那么，我们可以锁定对象，程序修改为：

```
Application.lock
Application("counter") = CType(Application("counter") + 1, Int32)
Application.unlock
```

这样，当某一程序执行了 Application.lock 之后，其他应用程序就暂时不能使用 Application 对象，直到 Application unlock。

4.4.1.2 Application 的事件

当 Application 对象的生命周期开始时，Application_onstart 事件会被启动，当 Application 对象的生命周期结束时 Application_onend 事件会被启动。通常我们会在 global.asax 中定义 Application_onstart 事件。这和 ASP 程序相类似，有一点差别是 Application_onstart 事件是

在 global.asa 中定义。还有就是除了以前的四个事件又 增加了 两个事件 Application_BeginRequest 事件 和 Application_EndRequest 事件。

在上一个例子中我们实现了用计数器来对页面进行统计，但是这样的程序有这样的一个问题，就是只能统计单个的页面，我们在 asp+中可以很轻松的实现对整个站点页面的统计。为了达到这个目的，我们使用将使用 Application_BeginRequest 事件和 Application_EndRequest 事件。这两个事件在站点的任意一个文件被请求的时候都会被激发，因此我们便利用这个事件实现对站点的访问统计。

我们首先来看看这个 global.asax 文件

```
<script language="VB" runat="server">
Sub Application_End(Sender As Object, E As EventArgs)
'我们捎带实现了 站点的当前在线人数
dim intOnlineNumber as integer
intOnlineNumber=cInt(Application("ONLINENUMBER"))-1
Application("ONLINENUMBER")=intOnlineNumber
End Sub

Sub Session_Start(Sender As Object, E As EventArgs)
Application.Lock
intOnlineNumber=cInt(Application("ONLINENUMBER"))+1
Application("ONLINENUMBER")=intOnlineNumber+1
Application.UnLock
End Sub
Sub Application_BeginRequest(Sender As Object, E As EventArgs)
response.write("当前访问的页面是 " + Request.FilePath + "<br>")
'既然我们可以得到 FilePAth 则我们只要把这个参数进行详细的各种各样的统计就可以了
End Sub
</script>
'好了一切完结之后，我们访问站点的任意一个 aspx 文件，都会在最上方发现一行文字：当前访问的页面是 。。。。
```

怎么样，还不赶快尝试一下？？

4.4.2 Session

4.4.2.1 Asp.net 里的 Session

对 Session 的应用，我们通过在 Global.asax 中设置，然后在 aspx 页面中调用来进行。下面我们就一个例子来说明这个问题。

4.4.2.2 一个 Session 例子：

我们在 Global.asax 中加上一个 Session_Start 方法，在里面定义响应的属性：

```
Sub Session_Start(Sender As Object, E As EventArgs)
        Session("name") = "saidy"
    Session("email") = "saidychan@sina.com"
    Session("tel") = "130000121553"
End Sub
```

赋予属性一个初始值，这样当我们在调用页面（session.aspx）装入时直接就用这些默认值：

```
<script language="VB" runat="server">
    '返回 Session 值方法：
    Function getinfo(Key As String) As String
      Return Session(Key).ToString()
    End Function
</script>
```

注意 Key 的定义，我们用这个语句获得方法的具体返回值,如“name“值：

```
<%=getinfo("name")%>
```

在另外一个文件（info.aspx）中，我们用下面的语句来获得数据：

```
Sub sc(Sender As Object, E As EventArgs)
  Session("name") = name.Value
  Session("email") = email.Value
  Session("tel") = tel.Value
  Response.Redirect(State("Referer").ToString())
End Sub
```

保存在 Session 中。

下面是我们具体的代码：

application\Global.asax 文件：

```
<script language="VB" runat="server">

Sub Session_Start(Sender As Object, E As EventArgs)

  Session("name") = "global_saidy"
  Session("email") = "global_saidychan@sina.com"
  Session("tel") = "global_130000121553"
End Sub

</script>
```

application\session.aspx 文件：

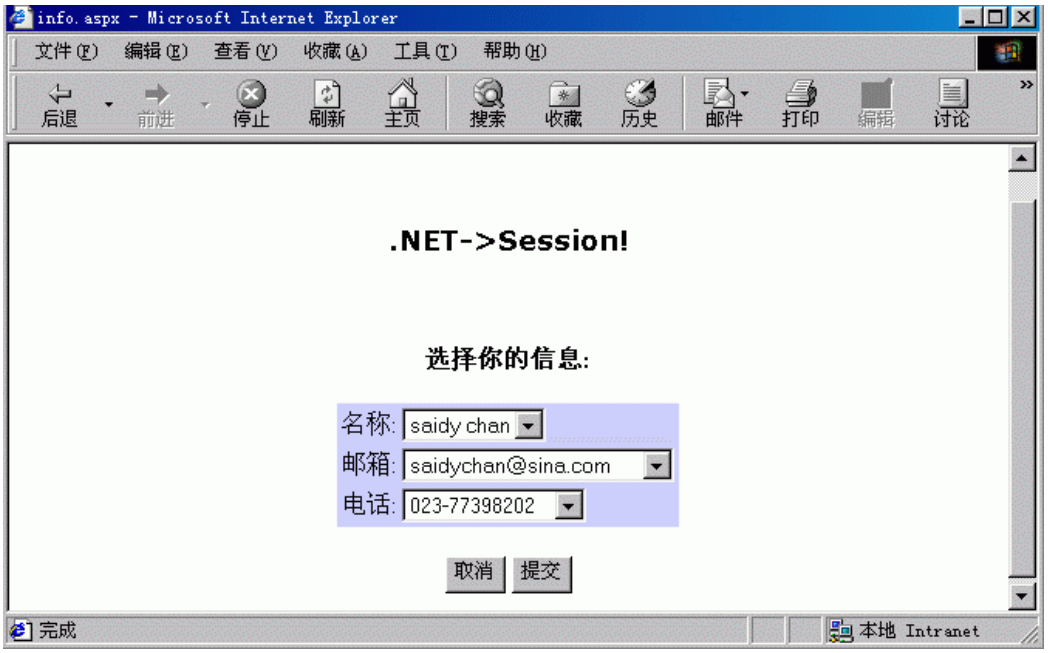
```
<html>
<script language="VB" runat="server">
    '返回 Session 值方法：
    Function getinfo(Key As String) As String
      Return Session(Key).ToString()
    End Function
</script>

<body >
<br><br><br>
<center>
  <h3><font face="Verdana">.NET->Session</font></h3>
</center>
<br><br>
  <center>
<b><a href="info.aspx">员工档案！</a></b><p>
  <div align="center">
<center>
<table border="0" width="35%" cellspacing="0" cellpadding="0">
  <tr>
    <td width="50%">名称：</td>
    <td width="50%"><%=getinfo("name")%></td>
  </tr>
  <tr>
    <td width="50%">邮箱：</td>
    <td width="50%"><%=getinfo("email")%></td>
  </tr>
  <tr>
    <td width="50%">电话：</td>
    <td width="50%"><%=getinfo("tel")%></td>
  </tr>
</table>
</center>
</div>
  </center>
  </body>
</html>
```

application\info.aspx 文件：

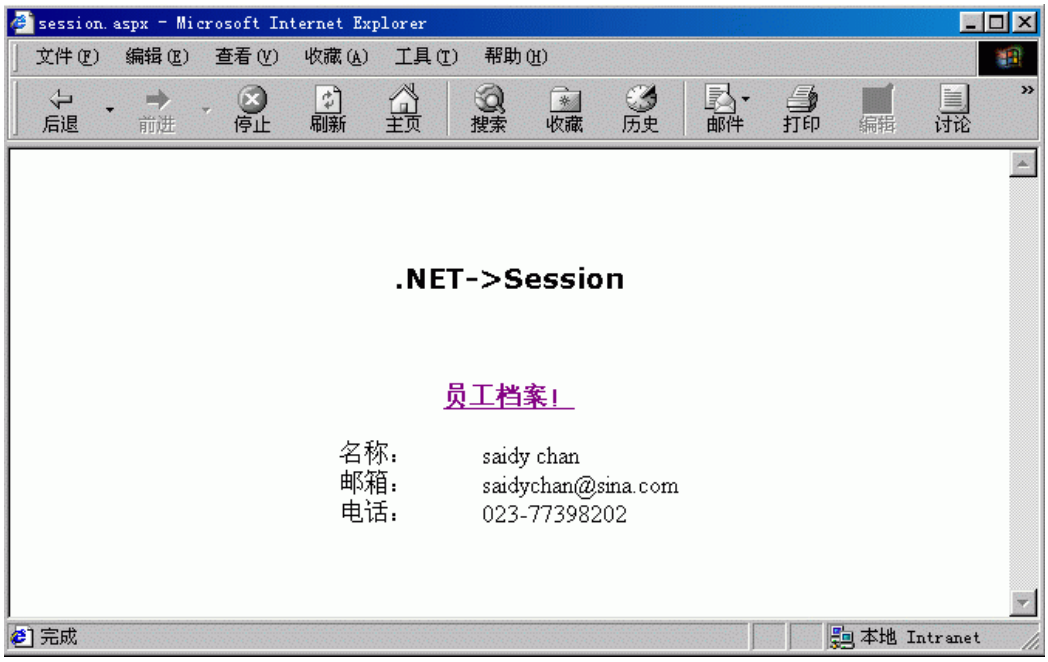
```
<html>
  <script language="VB" runat="server">
       '取得上一页的信息
       Sub Page_Load(Sender As Object, E As EventArgs)

            If Not (Page.IsPostBack)
              State("Referer") = Request.Headers("Referer")
            End If
       End Sub

       '按钮事件，把数据保存在 Session 中，并返回上一页
       Sub sc(Sender As Object, E As EventArgs)

          Session("name") = name.Value
          Session("email") = email.Value
          Session("tel") = tel.Value
          Response.Redirect(State("Referer").ToString())
       End Sub

       '按钮事件，返回上一页
       Sub cc(Sender As Object, E As EventArgs)

          Response.Redirect(State("Referer").ToString())
       End Sub
  </script>

<body >
 <br>
  <br>
  <br>
  <center>

     <form runat="server">
    <h3><font face="Verdana">.NET->Session!</font></h3>
  </center>

  <br>
    <br>
    <center>
    <b>选择你的信息: </b><p>
    <table bgcolor="#ccccff">
      <tr>
        <td>名称:</td>
        <td>
```

```
            <select id="name" runat="server">
              <option>saidy chan</option>
              <option>贺禧</option>
              <option>吕兵</option>
            </select>
          </td>
        </tr>

        <tr>
          <td>邮箱:</td>
          <td>
            <select id="email" runat="server">
              <option>saidychan@sina.com</option>
              <option>chenyx@staff.yesky.com</option>
              <option>hexi1224@sina.com</option>
            </select>
          </td>
        </tr>

        <tr>
          <td>电话:</td>
          <td>
            <select id="tel" runat="server">
              <option>023-77398202</option>
              <option>013011235488</option>
              <option>1398855566588</option>
            </select>
          </td>
        </tr>

      </table>

      <p>
      <input type="submit" OnServerClick="cc" Value="取消" runat="server"/>
      <input type="submit" OnServerClick="sc" Value="提交" runat="server"/>

      </form>

      </center>

    </body>

</html>
```

我们看到页面把我们储存在 Global.asax 文件中的数据取出来了，我们点击“员工档案连接：



点击“提交“按钮“，我们看到下面的结果：



我们看到提交的结果是 Session 中保存的数据被替代了。

4.4.3 小结

利用几个例子讲述再网上非常有用的两个属性 Application 和 Session，这两个属性非常重要，凡是做个一些网站的读者都知道的。

第五章 安全访问控制

asp.net 中讨论安全性，首先要解决 2 个问题。那就是谁有权力进入系统？他进入系统以后能进行何种操作？在解决谁能进入系统的问题中，我们通常会维护一张允许进入系统的用户的名单，当用户要求进入的时候，我们判断他是否是合法用户。这样一来，问题就转化为如何有效地判别一个用户是否是系统的有效用户，我们称之为“验证“过程。一个常见的验证过程是，当我们进入某些系统时，被要求输入用户和口令。当用户进入以后，我们只允许他访问事先指定给他的资源，这一过程称为”授权“。只有通过授权检查后，用户才能够对相应资源进行操作。在 asp.net 环境中，asp.net 和 IIS 结合在一起为用户提供验证和授权服务。

Asp.net 的应用程序还可以根据进入用户的不同标识，执行相应不同的应用代码。这种方式被称为“角色扮演“（impersonation）。

在一些应用中，对于不同的用户所能够看到和执行的功能是不尽相同的，这就要求 asp.net 提供“角色“（Role）和”用户“(User)的区分方法。

4.5.1 验证和授权(Authentication And Authorization)

asp.net 和 IIS 一起为用户提供验证服务，用户验证方式有 3 种，即基本验证方式（basic）、简要验证方式（digest）、窗口验证方式（windows）。同时 asp.net 支持微软的”护照”(passport)验证服务，它单方面提供签到服务和用户描述服务。

此外，asp.net 还提供了 cookies，帮助建立一种基于用户 Form 的验证方式，通过 cookies 用户的应用程序可以用自己的代码和逻辑实现用户定义的可信性验证。

由于 asp.net 的验证服务是建立在 IIS 的验证服务之上的，因此在设立自己的应用服务的时候有时需要在 IIS 中进行相应的设置。例如：为实现 asp.net 的基本验证服务，就必须利用 Internet Service Tool 工具把该应用设置成基本鉴别服务方式。

具体方法为：

选择 ”开始”->”程序“->”管理工具“->”Internet 信息服务器“

打开”默认 Web 站点”，选择你的应用站点，比如：此处为”777“

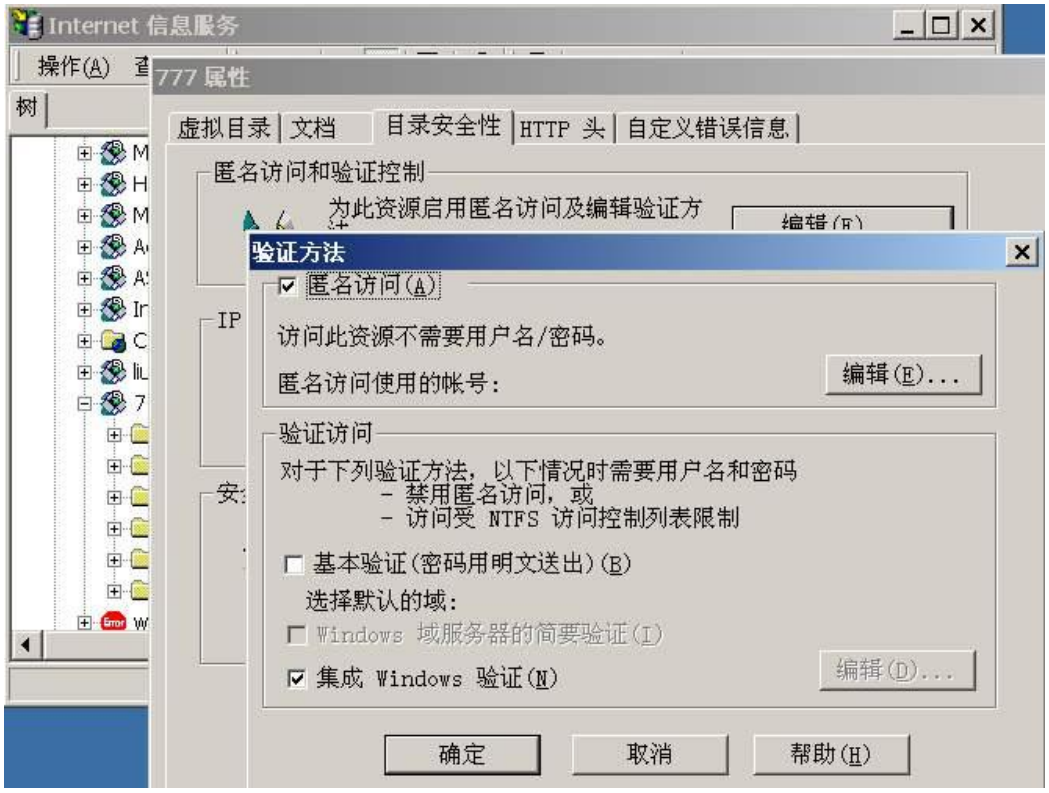
右击鼠标，选择”属性“，再选择”目录安全性“标签”

在”匿名访问和验证控制“框中，点击”编辑“按钮

然后在出现的”验证方法“窗口中，把”验证方法“框中，”基本验证“前面的选项打勾

最后确认，就完成了设置基本验证方法

操作的画面如下：



要使 Asp.Net 的验证机制生效，我们需要对应用的配置文件 config.web 进行设置。设置验证需要包含在<security>和</security>标识之间。它使用<authentication mode="…">语法形式，其中 mode 属性指定了验证采用的方式，具体的验证模式如下表格所示：

模 式	描 述
None	没有任何 Asp.Net 的验证服务被激活。
Windows	Asp.Net 和 Windows 的机制一起配合使用，可以授权限制 windiows Nt 的用户和工作组的访问。
Cookie	Asp.Net 验证服务可以管理 cookies，可以引导使一个未经授权的用户去登录网页。这种模式经常和 IIS 一起配合，可以让一个匿名用户去访问应用程序。
Passport	Asp.Net 验证服务通过使用护照（passport）服务软件包，附属在服务的外层，更加强了验证服务，这种软件包必须安装到系统中才能使用。

举例说来，使用 Cookie 验证方式对某个应用进行验证，那么在其应用的配置文件 config.web 中，应该含有一下几行：

```
<!--config.web-->
…
<configuration>
```

```
…
    <security>
    …
        <authentication mode="Cookie"/>
    ….
    </security>
    …
</configuration>
…
```

asp.net 提供了两种授权方式：基于 ACL、资源权限的授权方式和 URL 授权。基于 ACL 和资源权限的授权方式有点类似于 Unix 下的文件权限检查，不过它更加严格和完备，当用户请求某个页面时，asp.net 检查该页面的 ACL(访问控制列表)和该文件的权限，看该用户是否有权限读取该页面。有，则该页面称作“已授权“。这种授权方式主要通过系统管理员对文件的权限的设定来实现。

而 URL 授权，对于某个用户的页面请求，并不是从文件权限出发，而是根据系统的配置情况，来决定用户的请求是否是经过授权的。URL 授权的方式的实现通常是通过设置应用配置文件 config.web 中关于授权和角色的配置节来实现的。

4.5.2 基于 WINDOWS 的验证

当采用基于 windows 方式的验证以后(<authentication mode=”windows” />)，asp.net 对每一个页面请求产生一个 WindowsPrincipal 对象 User，这个对象可以被 URL 授权方式用来进行授权，或者是被用户程序用来判断是否是某种角色。

例如，判断当前用户是否是管理员，可以采用以下的代码：

```
If User.IsInRole(“Administrator”)
‘如果是管理员组成员，…
…
Else
‘如果不是管理员组成员，…
…
End If
```

可能用户希望对验证过程加入一些自己的控制代码，那么就需要对 WindowsPrincipal 对象的 WindowsAuthentication_OnAuthenticate 事件作出自己的处理，编写自己的消息处理函数。通常用户通过实现 System.Security.Principal.Iprincipal 类来完成。

4.5.3 基于 FORM 的验证

基于 FORM 的验证，实际上是允许应用程序定义自己的验证画面和可信性验证。当用户进入时，出现事先定义的画面并请求输入验证要素，输入完毕，用用户逻辑对输入验证，

通过，则进入应用，否则，返回起始输入画面。基于 FORM 的验证，通常都采用 cookies 技术来实现验证任务。启用基于 FORM 的验证，是在 config.web 中设置<authentication mode=”Cookie” />来实现的。

例如：采用基于 Form 的验证方式，拒绝匿名用户进入的配置段如下：

```
<!--config.web->
    …
    <configuration>
        <security>
            <authentication mode="Cookie"/>
            <authorization>
                <deny users="?" />
            </authorization>
        </security>
    </configuration>
    …
```

其中，<authorization>一节中，用 deny 标识表示禁止某种用户，“？”代表匿名用户，“*”代表所有用户，当然在 users 后也可以跟指定的用户，表示只拒绝指定的用户。

此外，在配置节中还有一个 cookie 标识，它规定了 Cookie 的行为，也较为重要。Cookie 标识有三个属性：

• **decryptionkey** 用于指定对 cookie 加解密的密钥。如果不指定或指定为 autogenerate，密钥将采用 Crypto API 产生的系统密钥。当指定为 autogenerate 时，产生的密钥和机器相关，因而不能跨机器和平台，除非显示给出密钥。

• **loginurl** 验证页面。当用户验证失败以后，被定向到的页面，它可以是在本地，也可以是其他机器上。

• **cookie** 指定用于验证任务的 cookie 的名字。由于在同一台机器上，可能存在多个应用，而同一应用使用一个 cookie 名，所以同一机器上的 cookie 名字不能相同。

例如，在 config.web 中有如下定义：

```
<!--config.web -->
…
<security>
    <authentication mode="Cookie">
        <cookie decryptionkey="autogenerate" loginurl="login.aspx"
                                                cookie=".ASPXCOOKIEDEMO" />
    </authentication>
</security>
…
```

从这段配置，我们可以知道 cookie 的密钥由系统产生，当验证失败以后，页面将跳转至 login.aspx，cook 名为“.ASPXCOOKIEDEMO”

在基于 FORM 的验证中，一个常用到的对象是 CookAuthentication。

CookAuthentication 有 4 个比较重要的方法：

·**RedirectFromLoginPage 方法**，它通常在验证成功以后，从用户的验证画面返回用户开始请求的页面。它带有两个参数，第一个为 cookie 要记录的用户名，第二个表明是否记录到永久性 cookie 中。

例如：

CookieAuthentication.RedirectFromLoginPage(“Chen”,True)

表示把用户“Chen”记录到硬盘上的 cookie 记录中，然后跳转到用户原先请求的页面上。

·**GetAuthCookie** 从 cookie 中取得指定用户名的值。它的参数和 RedirectFromLoginPage 方法是一样的。

例如：

CookieAuthentication.GetAuthCookie(“Chen”,Flase)

表示从当前连接的 cookie 中取出用户名为 chen 的值

• **SetAuthCookie** 把指定用户名存入 cookie 中。参数和 GetAuthCookie 方法一致

例如：

CookieAuthentication.SetAuthCookie(“Chen”,Flase)

表示把用户“Chen”记录到连接过程中存在的 cookie 中

• **SignOut** 删除当前用户的验证 Cookie，它不会理会 Cookie 到底是在内存，还是在硬盘上。

下面我将介绍一个完整的例子：

application\delault.aspx 文件：

```
<%@ Import Namespace="System.Web.Security " %>

<html>

  <script language="VB" runat=server>

    Sub Page_Load(Src As Object, E As EventArgs)
        Welcome.Text = "Hello, " + User.Identity.Name
    End Sub

    Sub Signout_Click(Src As Object, E As EventArgs)
        CookieAuthentication.SignOut()
        ‘删去 Cookies
        Response.Redirect("login.aspx")
        ‘返回验证画面
    End Sub

  </script>

  <body>
```

```
    <h3><font face="Verdana">使用 Cookie 验证方式</font></h3>

    <form runat=server>

      <h3><asp:label id="Welcome" runat=server/></h3>

      <asp:button text="Signout" OnClick="Signout_Click" runat=server/>

    </form>

  </body>

</html>
```

application\login.aspx 文件：

```
<%@ Import Namespace="System.Web.Security " %>

<html>

  <script language="VB" runat=server>

    Sub Login_Click(Src As Object, E As EventArgs)
        If (UserEmail.Value = "test@263.net" Or UserEmail.Value = "try@163.net") And
UserPass.Value = "password"
    ‘如果用户为 test@263.net 或者是 try@163.net 而且口令为 password,就认为通过验证
            CookieAuthentication.RedirectFromLoginPage(UserEmail.Value,
PersistCookie.Checked)
     ‘记录 Cookie 并且跳转到请求的页面
        Else
            Msg.Text = "非法用户或口令: 请重试"
        End If
    End Sub

  </script>

  <body>

    <form runat=server>

      <h3><font face="Verdana">登录窗口</font></h3>
```

```
        <table>
          <tr>
            <td>Email 地址:</td>
            <td><input id="UserEmail" type="text" runat=server/></td>
            <td><ASP:RequiredFieldValidator ControlToValidate="UserEmail" Display="Static"
ErrorMessage="*" runat=server/></td>
          </tr>
          <tr>
            <td>口令:</td>
            <td><input id="UserPass" type=password runat=server/></td>
            <td><ASP:RequiredFieldValidator ControlToValidate="UserPass" Display="Static"
ErrorMessage="*" runat=server/></td>
          </tr>
          <tr>
            <td>永久性 Cookie:</td>
            <td><ASP:CheckBox id=PersistCookie runat="server" /> </td>
            <td></td>
          </tr>
        </table>

        <asp:button text="Login" OnClick="Login_Click" runat=server/>

        <p>

        <asp:Label id="Msg" ForeColor="red" Font-Name="Verdana" Font-Size="10"
runat=server />

      </form>
   </body>

</html>
```

config.web 文件：

```
<configuration>

   <security>

      <authentication mode="Cookie">
      <!--采用 cookie 验证方式-->
        <cookie decryptionkey = "autogenerate" loginurl = "login.aspx" cookie =
".ASPXUSERDEMO" />
        <!--密钥系统产生，登录页面为 login.aspx,记录的 Cookie 名为.aspuserdemo-->
      </authentication>
```

```
    <authorization>
      <deny users="test@263.net" />
      <!--虽然程序中有两个用户 test@263 和 try@163.net 可以进入，但此处封掉
test@263.net,观察 deny 标识是否有效-->
      <deny users="?" />
      <!--拒绝匿名用户 anonymous-->
    </authorization>
      </security>
  <globalization requestencoding="UTF-8" responseencoding="UTF-8" />
</configuration>
```

我们还可以在 config.web 中设置用户和密码，当用户在验证窗口输入用户和密码后，将其带入 CookieAuthenticationManager.Authenticate，有它来验证是否是合法用户。

在 config.web 中设置用户和密码使用 credentials 标识，它有一个属性 passwordformat, 决定口令存储的方式。Passwordformat 可以为以下值：

- **Clear** 口令以纯文本方式存储。
- **SHA1** 口令以 SHA1 方式存储。
- **MD5** 口令以 MD5 方式存储。

例如： 在一个配置文件中，

```
<!--config.web-->
…
<configuration>
…
<security>
    <authentication>
        <credentials passwordformat=”SHA1” >
            <user name=”Li” password=”GASDFSA9823598ASDBAD”/>
            <user name=”Chen” password=”ZASDFADSFASD23483142”/>
        </credentials>
    </authentication>
</security>
…
</configuration>
```

从这个例子，我们看到 credentials 标识中的 passwordformat 为 SHA1，所以下面的用户的密码是以 SHA1 方式加密以后的格式。在<credentials>和</credentials>之间我们定义了两个用户“Chen”和”Li”，其密码不可识别。

4.5.4 授权用户和角色

URL 授权控制对资源的访问权限。它可以使一些用户和角色对资源有存取权限，也可以拒绝某些用户和角色对资源的存取。甚至它还可以决定能够存取资源的 HTTP 方法(例如：

不允许 get，允许 POST 等等）。

对于授权用户和角色的控制，asp.net 通过配置文件 config.web 中的<authorization>标识段来实现。<allow>标识表示允许对资源的访问，<deny>标识表示拒绝对资源的访问。

它们都有两个属性，users 和 roles 分别表示用户和角色。

我们来看一个实例：

```
<!--config.web->
…
<configuration>
<security>
<authorization>
    <allow users="nobody@163.net" />
    <allow roles="Admins" />
    <deny users="*" />
</authorization>
</security>
</configuration>
…
```

它表明了这样一个事实，用户“nobody@163.net“和角色 Admins 有访问本站点的权力，其他用户对本站点的访问将被拒绝。也就是用户 nobody@163.net 和角色 Admins 分别是授权用户和授权角色。

同样，我们可以定义多个用户或者角色被授权或禁止，它们之间以“，“分隔。

例如：

```
…
<allow users=”Chen,Li,Wang” />
<deny roles=”Admins,Everyone” />
…
```

表示用户“Chen”、“Li”、”Wang”是授权用户，但是角色为“Admins”或者是”Everyone”的被排除在外。

它的效果和分开写是一样的，如上例也可以写为：

```
…
<allow users=”Chen” />
<allow users=”Li” />
<allow users=”Wang” />
<deny roles=”Admins” />
<deny roles=”Everyone” />
…
```

此外，还可以决定用户的某种 HTTP 方法是否可以被允许，方法是使用 verb 属性来表明对那种 HTTP 方法操作。

例如：

```
…
        <allow verb=post users=”Chen,Li” />
        <deny verb=get roles=”everyone” />
```

…

表示允许用户“Chen”,”Li“采用 post 方法访问资源，而拒绝角色 everyone 的 get 方式对资源的访问。

符号“*“和”？“在<allow>和<deny>标识中有特殊的含义。”*“表示任何用户,”？“表示匿名用户。

例如：

```
…
<authorization>
    <allow users=”*” />
    <deny users=”?” />
</authorization>
```

表示除了匿名用户以外的所有用户都被允许访问本站点。

由前面的学习我们知道，在 asp.net 中应用是树型分层的，所以其配置文件也是呈层次结构，这也就导致了用户和角色的授权不是单一的结果，它要取决于沿树型结构上所有配置文件指定的结果的合集，而且越接近叶节点的配置越是有效。

例如：访问 http:www.my.com/MyApp/a.aspx

在 http:www.my.com 的根目录下的配置文件 config.web 有如下内容：

```
…
<security>
  <authorization>
    <allow users=”*” />
  </authorization>
</security>
…
```

而在 MyApp 目录下有配置文件 config.web 内容如下：

```
…
<security>
  <authorization>
        <allow users=”Chen” />
        <deny users=”*” />
  </authorization>
<security>
…
```

那么，授权用户的集合到底是怎样的呢？asp.net 首先取得站点根目录下的配置，即所有用户都被允许访问，然后 asp.net 进入 Myapp 的目录取得其下的配置，除用户“Chen”以外所有的用户被拒绝，最后合并两个授权集合，并且如果两者之间有冲突，以后者为准。所以最终的授权集合为用户“Chen”,其他用户被拒绝。

第六章　会员系统例子：

4.6.1　数据库：

我们建立两个表 City 和 Customer，在建立几个存储过程来操作数据库（member.sql）：

```
CREATE TABLE [dbo].[City] (
    [ID] [int] NOT NULL ,
    [Name] [nvarchar] (20) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[customer] (
    [ID] [int] IDENTITY (1, 1) NOT NULL ,
    [Name] [nvarchar] (30) NULL ,
    [sex] [bit] NOT NULL ,
    [Birth] [smalldatetime] NULL ,
    [City] [int] NULL ,
    [Zip] [nvarchar] (6) NULL ,
    [Address] [nvarchar] (50) NULL ,
    [Telephone] [nvarchar] (30) NULL ,
    [PIN] [nvarchar] (30) NULL ,
    [Password] [nvarchar] (30) NULL ,
    [EMail] [nvarchar] (30) NULL ,
    [Question] [nvarchar] (50) NULL ,
    [Answer] [nvarchar] (50) NULL
) ON [PRIMARY]
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE [GetAllCities] AS
    Select [ID],[Name] From City
GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE [GetDetailByID]
@ID Int
```

```
AS
Select PIN,Password,[Name],Birth,Sex,City,Zip,EMail,Telephone,Address,Question,Answer,[ID]
From Customer
Where [ID]=@ID


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE [GetDetailByPIN]
@PIN NVarChar(30)
AS
Select PIN,Password,[Name],Birth,Sex,City,Zip,EMail,Telephone,Address,Question,Answer,[ID]
From Customer
Where [PIN]=@PIN



GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  ON     SET ANSI_NULLS  ON
GO

CREATE PROCEDURE [IsValidUser]
@PIN NVarChar(30),
@Password NVarChar(30),
@ID Int Output
AS

Select @ID=[ID] From Customer Where PIN=@PIN and Password=@Password


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO
```

```
CREATE PROCEDURE [Modify]
@ID Int,
@PIN NVarCHar(30),
@Password NVarCHar(30),
@Name NVarCHar(30),
@Birth DateTime,
@Sex bit,
@City int,
@Zip NVarCHar(6),
@EMail NVarCHar(30),
@Telephone NVarCHar(30),
@Address NVarCHar(30),
@Question NVarCHar(50),
@Answer NVarCHar(50)
AS

Update Customer
Set
Password=@Password,[Name]=@Name,Birth=@Birth,Sex=@Sex,City=@City,Zip=@Zip,EMail
=@EMail,Telephone=@Telephone,Address=@Address,Question=@Question,Answer=@Answer
Where ID=@ID And PIN=@PIN


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  ON    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE [PromptPassword]
@PIN nvarchar(30),
@Answer nvarchar(50),
@Password nvarchar(30) output
AS
select @Password=Password From Customer Where PIN=@PIN And Answer=@Answer


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
```

```
GO

CREATE PROCEDURE [PromptQuestion]
@PIN nvarchar(30),
@Question nvarchar(50) output
AS
select @Question=Question From Customer Where PIN=@PIN



GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
GO

CREATE PROCEDURE [Register]
@ID Int OUTPUT,
@PIN NVarCHar(30),
@Password NVarCHar(30),
@Name NVarCHar(30),
@Birth DateTime,
@Sex bit,
@City int,
@Zip NVarCHar(6),
@EMail NVarCHar(30),
@Telephone NVarCHar(30),
@Address NVarCHar(30),
@Question NVarCHar(50),
@Answer NVarCHar(50)
AS

Insert Into Customer
(PIN,Password,[Name],Birth,Sex,City,Zip,EMail,Telephone,Address,Question,Answer)
Values
(@PIN,@Password,@Name,@Birth,@Sex,@City,@Zip,@EMail,@Telephone,@Address,@Ques
tion,@Answer)

SELECT @ID = @@IDENTITY


GO
SET QUOTED_IDENTIFIER  OFF    SET ANSI_NULLS  ON
```

GO

4.6.2 登录页面：

判断用户的 Pin 和密码是否正确，并且是注册和密码提示的入口：
（csbook\appsoft\member\login.aspx）

```
<%@ Page EnableSessionState="False" MaintainState="false" Debug="True"%>
<%@ Import Namespace="DarkMan" %>

<script language="VB" runat="server">
    Sub Page_Load(sender As Object, e As EventArgs)
        RegisterLink.NavigateUrl="register.aspx?ReturnUrl="+Request.Params("ReturnUrl")

    PromptPasswordLink.NavigateUrl="prompt.aspx?ReturnUrl="+Request.Params("ReturnUrl"
)
    End Sub

    Sub OnLogin(sender As Object, e As EventArgs)
        Dim theDB As New MemberDB

        Try
            Dim theID As Integer = theDB.IsValidUser(CStr(pin.Text),CStr(password1.Text))
            CookieAuthentication.SetAuthCookie(tehID, false)
            Page.Navigate("modify.aspx")
        Catch e1 As Exception
            Prompt.Text="登录失败！"
        End Try
    End Sub
</script>

<HTML>
<BODY>
<DIV Align="Center">
<H1>会员登录</H1>
<hr wifth=600>
</DIV>

<FORM RunAt="Server">
<TABLE WIDTH=600 BORDER=0 CELLSPACING=1 CELLPADDING=1 Align="Center">
    <TR>
        <TD COLSPAN="2">
```

```
        <asp:RegularExpressionValidator id="RegPin" runat="server"
              ControlToValidate="pin"
              ValidationExpression="^[\S^']{6,30}$"
              Display="Dynamic">
                 帐号中：某些特殊字符禁用！另外，必须至少 6 个字符！
        </asp:RegularExpressionValidator>
    </TD>
</TR>


<TR>
    <TD COLSPAN="2">
        <asp:RegularExpressionValidator id="RegPassword" runat="server"
              ControlToValidate="password1"
              ValidationExpression="^[\S^']{6,30}$"
              Display="Dynamic">
                 密码中：某些特殊字符禁用！另外，必须至少 6 个字符！
        </asp:RegularExpressionValidator>
    </TD>
</TR>


<TR>
    <TD>账号</TD>
    <TD><asp:textbox id="pin" RunAt="server"/>
         <asp:RequiredFieldValidator id="RFpin" runat="server"
              ControlToValidate="pin"
              ErrorMessage="*"
              ForeColor="Red">
         </asp:RequiredFieldValidator>
    </TD>
</TR>
<TR>
    <TD>密码</TD>
    <TD><asp:textbox id="password1" RunAt="server" TextMode="Password"/>
         <asp:RequiredFieldValidator id="RFpass" runat="server"
              ControlToValidate="password1"
              ErrorMessage="*"
              ForeColor="Red">
         </asp:RequiredFieldValidator>
    </TD>
</TR>
<TR>
    <TD COLSPAN="2"><asp:label id="Prompt" RunAt="Server"/>
```

```
            </TD>
        </TR>
        <TR>
            <TD COLSPAN="2">
                <input type="submit" OnServerClick="OnLogin" Value="马上登录"
runat="server"/>
                <asp:Hyperlink id="RegisterLink" runat="server"
                    Text="我要注册"
                />
                <asp:Hyperlink id="PromptPasswordLink" runat="server"
                    Text="密码提示"
                />
            </TD>
        </TR>
</TABLE>
</FORM>

</BODY>
</HTML>
```

4.6.3 注册页面：

（csbook\appsoft\member\register.aspx）：

```
    <%@ Page EnableSessionState="False" MaintainState="false" Debug="True"%>
<%@ Import Namespace="DarkMan" %>

<script language="VB" runat="server">
    Sub Page_Load(sender As Object, e As EventArgs)
        If Not Page.IsPostBack Then
            Dim result As SQLDataReaderResult
            Dim obj As New City
            result = obj.GetAll()

            city.DataSource=result.dr
            city.DataBind

            result.Close
        End If
    End Sub

    Sub OnRegister(sender As Object, e As EventArgs)
```

```
        If Page.IsValid Then
            Dim theUser As New MemberDetail
            theUser.PIN = pin.Text
            theUser.Password = password1.Text
            theUser.Name = name.Text
            theUser.Birth = DateTime.Parse(birth.Text)
            theUser.Sex = sex.SelectedItem.Value
            If city.SelectedItem<>DBNull Then
                theUser.City = city.SelectedItem.Value
            End If
            theUser.Zip = zip.Text
            theUser.EMail = email.Text
            theUser.Telephone = telephone.Text
            theUser.Address = address.Text
            theUser.Question = question.Text
            theUser.Answer = answer.Text

            Try
                Dim theDB As New MemberDB
                Dim theID As Integer = theDB.Register(theUser)

                Page.Navigate( Request.Params("ReturnUrl") )
            Catch e1 As Exception
                ErrMsg.Text="注册失败"
            End Try
        End If
    End Sub

</script>

<HTML>
<BODY>
<DIV Align="Center">
<H1>会员注册</H1>
<hr wifth=600>
</DIV>

<FORM RunAt="Server">
<TABLE WIDTH=600 BORDER=0 CELLSPACING=1 CELLPADDING=1 Align="Center">
    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegPin" runat="server"
                ControlToValidate="pin"
                ValidationExpression="^[\S^']{6,30}$"
```

```
                Display="Dynamic">
                    帐号中：某些特殊字符禁用！另外，必须至少 6 个字符！
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegName" runat="server"
                ControlToValidate="name"
                ValidationExpression="^[\S^']{6,30}$"
                Display="Dynamic">
                    姓名中：某些特殊字符禁用！另外，必须至少 6 个字符！
            </asp:RegularExpressionValidator>
        </TD>
    </TR>


    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegPassword" runat="server"
                ControlToValidate="password1"
                ValidationExpression="^[\S^']{6,30}$"
                Display="Dynamic">
                    密码中：某些特殊字符禁用！另外，必须至少 6 个字符！
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegBirth" runat="server"
                ControlToValidate="birth"
                ValidationExpression="^\d{4}-\d{1,2}-\d{1,2}$"
                Display="Dynamic">
                    日期格式：2001-7-01
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="Regzip" runat="server"
                ControlToValidate="zip"
```

```
                ValidationExpression="^\d{6}$"
                Display="Dynamic">
                    邮政编码不对
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="Regtel" runat="server"
                ControlToValidate="telephone"
                ValidationExpression="^[\d-\(\)]+$"
                Display="Dynamic">
                    电话号码不对
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegEMail" runat="server"
                ControlToValidate="email"
                ValidationExpression="^[\w-_.]*[\w-_.]@[\w].+[\w]+[\w]$"
                Display="Dynamic">
                    EMail 格式：xxx@222.com
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
             <asp:CompareValidator id="CompPassword12"
                ControlToValidate="password2"
                ControlToCompare = "password1"
                Display="Dynamic"
                Type="String" runat="server">
                 密码校验不正确！
             </asp:CompareValidator>
        </TD>
    </TR>

    <TR>
        <TD>账号</TD>
        <TD><asp:textbox id="pin" RunAt="server"/>
```

```
            <asp:RequiredFieldValidator id="RFpin" runat="server"
                    ControlToValidate="pin"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
        <TD>姓名</TD>
        <TD><asp:textbox id="name" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFname" runat="server"
                    ControlToValidate="name"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD>密码</TD>
        <TD><asp:textbox id="password1" RunAt="server" TextMode="Password"/>
            <asp:RequiredFieldValidator id="RFpass" runat="server"
                    ControlToValidate="password1"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
        <TD>密码校验</TD>
        <TD><asp:textbox id="password2" RunAt="server" TextMode="Password"/></TD>
    </TR>
    <TR>
        <TD>生日</TD>
        <TD><asp:textbox id="birth" RunAt="server"/></TD>
        <TD>性别</TD>
        <TD>
            <asp:RadioButtonList RepeatColumns="2" id="sex" runat="server">
                <asp:ListItem text="男" value="0" Selected/>
                <asp:ListItem text="女" value="1"/>
            </asp:RadioButtonList></TD>
    </TR>
    <TR>
        <TD>城市</TD>
        <TD>
            <asp:DropDownList id="city" runat="server"
                DataTextField="Name"
                DataValueField="ID"
            />
```

```
        </TD>
        <TD>邮政编码</TD>
        <TD><asp:textbox id="zip" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFzip" runat="server"
                    ControlToValidate="zip"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD>EMail</TD>
        <TD><asp:textbox id="email" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFemail" runat="server"
                    ControlToValidate="email"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
        <TD>电话</TD>
        <TD><asp:textbox id="telephone" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFtel" runat="server"
                    ControlToValidate="telephone"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="1">地址</TD>
        <TD COLSPAN="3"><asp:textbox id="address" RunAt="server" Columns="50"/>
            <asp:RequiredFieldValidator id="RFAdrress" runat="server"
                    ControlToValidate="address"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="1">密码提示问题</TD>
        <TD COLSPAN="3"><asp:textbox id="question" RunAt="server" Columns="50"/>
            <asp:RequiredFieldValidator id="RFQ" runat="server"
                    ControlToValidate="question"
                    ErrorMessage="*"
```

```
                    ForeColor="Red">
                </asp:RequiredFieldValidator>
            </TD>
        </TR>
        <TR>
            <TD COLSPAN="1">密码提示问题答案</TD>
            <TD COLSPAN="3"><asp:textbox id="answer" RunAt="server" Columns="50"/>
                <asp:RequiredFieldValidator id="RFA" runat="server"
                    ControlToValidate="answer"
                    ErrorMessage="*"
                    ForeColor="Red">
                </asp:RequiredFieldValidator>
            </TD>
        </TR>
        <TR>
            <TD COLSPAN="4">
                <input type="submit" OnServerClick="OnRegister" Value="马上注册"
runat="server"/>
                <br>
                <font color="red"><asp:label id="ErrMsg" runat="server"/></font>
            </TD>
        </TR>
</TABLE>
</FORM>

</BODY>
</HTML>
```

4.6.4 修改页面：

（csbook\appsoft\member\modify.aspx）：

```
    <%@ Page EnableSessionState="False" MaintainState="false" Debug="True"%>
<%@ Import Namespace="DarkMan" %>

<script language="VB" runat="server">
    Sub Page_Load(sender As Object, e As EventArgs)
        If Not Page.IsPostBack Then
            Dim theDB As New MemberDB
            Dim m_PIN As String=Request.Params("PIN")
            Dim m_ID As Integer=CInt( Request.Params("ID") )
            Dim theUser As MemberDetail
```

```
            If m_PIN=DBNull And m_ID=DBNull Then
                CookieAuthentication.RedirectFromLoginPage(theID, false)
            Else If m_PIN<>DBNull Then
                theUser=theDB.GetDetailByPIN(m_PIN)
                userid.Text = theUser.ID
                pin.Text=theUser.PIN
                password1.Text=theUser.Password
                name.Text=theUser.Name
                birth.Text=theUser.Birth
                zip.Text=theUser.Zip
                email.Text=theUser.EMail
                telephone.Text=theUser.Telephone
                address.Text=theUser.Address
                question.Text=theUser.Question
                answer.Text=theUser.Answer
            Else
                theUser=theDB.GetDetailByID(m_ID)
                userid.Text = theUser.ID
                pin.Text=theUser.PIN
                password1.Text=theUser.Password
                name.Text=theUser.Name
                birth.Text=theUser.Birth
                zip.Text=theUser.Zip
                email.Text=theUser.EMail
                telephone.Text=theUser.Telephone
                address.Text=theUser.Address
                question.Text=theUser.Question
                answer.Text=theUser.Answer
            End If


            Dim result As SQLDataReaderResult
            Dim obj As New City
            result = obj.GetAll()

            city.DataSource=result.dr
            city.DataBind

            result.Close

        End If
    End Sub

    Sub OnModify(sender As Object, e As EventArgs)
```

```
        If Page.IsValid Then
            Dim theUser As New MemberDetail
            theUser.ID = userid.Text
            theUser.PIN = pin.Text
            theUser.Password = password1.Text
            theUser.Name = name.Text
            theUser.Birth = DateTime.Parse(birth.Text)
            theUser.Sex = sex.SelectedItem.Value
            If city.SelectedItem<>DBNull Then
                theUser.City = city.SelectedItem.Value
            End If
            theUser.Zip = zip.Text
            theUser.EMail = email.Text
            theUser.Telephone = telephone.Text
            theUser.Address = address.Text
            theUser.Question = question.Text
            theUser.Answer = answer.Text

            Dim theDB As New MemberDB
            Dim theID As Integer = theDB.Modify(theUser)

            CookieAuthentication.RedirectFromLoginPage(theID, false)
            Page.Navigate("login.aspx")
        End If
    End Sub

</script>

<HTML>
<BODY>
<DIV Align="Center">
<H1>会员信息修改</H1>
<hr wifth=600>
</DIV>

<FORM RunAt="Server">
<TABLE WIDTH=600 BORDER=0 CELLSPACING=1 CELLPADDING=1 Align="Center">
    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegPin" runat="server"
                ControlToValidate="pin"
                ValidationExpression="^[\S^']{6,30}$"
                Display="Dynamic">
                    帐号中：某些特殊字符禁用！另外，必须至少 6 个字符！
```

```
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegName" runat="server"
                ControlToValidate="name"
                ValidationExpression="^[\S^']{6,30}$"
                Display="Dynamic">
                    姓名中：某些特殊字符禁用！另外，必须至少 6 个字符!
            </asp:RegularExpressionValidator>
        </TD>
    </TR>


    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegPassword" runat="server"
                ControlToValidate="password1"
                ValidationExpression="^[\S^']{6,30}$"
                Display="Dynamic">
                    密码中：某些特殊字符禁用！另外，必须至少 6 个字符!
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegBirth" runat="server"
                ControlToValidate="birth"
                ValidationExpression="^\d{4}-\d{1,2}-\d{1,2}$"
                Display="Dynamic">
                    日期格式：2001-7-01
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="Regzip" runat="server"
                ControlToValidate="zip"
                ValidationExpression="^\d{6}$"
                Display="Dynamic">
```

```
                    邮政编码不对
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="Regtel" runat="server"
                ControlToValidate="telephone"
                ValidationExpression="^[\d-\(\)]+$"
                Display="Dynamic">
                    电话号码不对
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegEMail" runat="server"
                ControlToValidate="email"
                ValidationExpression="^[\w-_.]*[\w-_.]@[\w].+[\w]+[\w]$"
                Display="Dynamic">
                    EMail 格式：xxx@222.com
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
            <asp:CompareValidator id="CompPassword12"
                ControlToValidate="password2"
                ControlToCompare = "password1"
                Display="Dynamic"
                Type="String" runat="server">
                 密码校验不正确！
            </asp:CompareValidator>
        </TD>
    </TR>

    <TR>
        <TD>账号</TD>
        <TD><asp:textbox id="pin" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFpin" runat="server"
                    ControlToValidate="pin"
```

```
                ErrorMessage="*"
                ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
        <TD>姓名</TD>
        <TD><asp:textbox id="name" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFname" runat="server"
                ControlToValidate="name"
                ErrorMessage="*"
                ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD>密码</TD>
        <TD><asp:textbox id="password1" RunAt="server" TextMode="Password"/>
            <asp:RequiredFieldValidator id="RFpass" runat="server"
                ControlToValidate="password1"
                ErrorMessage="*"
                ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
        <TD>密码校验</TD>
        <TD><asp:textbox id="password2" RunAt="server" TextMode="Password"/></TD>
    </TR>
    <TR>
        <TD>生日</TD>
        <TD><asp:textbox id="birth" RunAt="server"/></TD>
        <TD>性别</TD>
        <TD>
            <asp:RadioButtonList RepeatColumns="2" id="sex" runat="server">
                <asp:ListItem text="男" value="0" Selected/>
                <asp:ListItem text="女" value="1"/>
            </asp:RadioButtonList></TD>
    </TR>
    <TR>
        <TD>城市</TD>
        <TD>
            <asp:DropDownList id="city" runat="server"
                DataTextField="Name"
                DataValueField="ID"
            />
        </TD>
        <TD>邮政编码</TD>
```

```
        <TD><asp:textbox id="zip" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFzip" runat="server"
                    ControlToValidate="zip"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD>EMail</TD>
        <TD><asp:textbox id="email" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFemail" runat="server"
                    ControlToValidate="email"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
        <TD>电话</TD>
        <TD><asp:textbox id="telephone" RunAt="server"/>
            <asp:RequiredFieldValidator id="RFtel" runat="server"
                    ControlToValidate="telephone"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="1">地址</TD>
        <TD COLSPAN="3"><asp:textbox id="address" RunAt="server" Columns="50"/>
            <asp:RequiredFieldValidator id="RFAdrress" runat="server"
                    ControlToValidate="address"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="1">密码提示问题</TD>
        <TD COLSPAN="3"><asp:textbox id="question" RunAt="server" Columns="50"/>
            <asp:RequiredFieldValidator id="RFQ" runat="server"
                    ControlToValidate="question"
                    ErrorMessage="*"
                    ForeColor="Red">
            </asp:RequiredFieldValidator>
```

```
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="1">密码提示问题答案</TD>
        <TD COLSPAN="3"><asp:textbox id="answer" RunAt="server" Columns="50"/>
            <asp:RequiredFieldValidator id="RFA" runat="server"
                ControlToValidate="answer"
                ErrorMessage="*"
                ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="4">
            <input type="submit" OnServerClick="OnModify" Value="马上提交"
runat="server"/>
        </TD>
    </TR>
</TABLE>

<asp:textbox id="userid" RunAt="server" Columns="0" width="0" height="0"/>

</FORM>

</BODY>
</HTML>
```

4.6.5 密码提示页面：

当我们输入用户的 Pin 时，程序自动把用户的提问问题显示出来，如果没有这个用户，则提示没有这个用户（csbook\appsoft\member\prompt.aspx）：

```
<%@ Page EnableSessionState="False" MaintainState="false" Debug="True"%>
<%@ Import Namespace="DarkMan" %>

<script language="VB" runat="server">
    Sub OnPromptQuestion(sender As Object, e As EventArgs)
        If Page.IsValid Then
            Try
                Dim theDB As New MemberDB
                question.Text = theDB.PromptQuestion(pin.Text)
```

```
                    Dim obj As New RequiredFieldValidator()

                    obj.ControlToValidate="answer"
                    obj.ErrorMessage = "需要输入密码问题答案！"
                    Panel.Controls.Add(obj)
                Catch e1 As Exception
                    password.Text = "不存在此账号！"
                End Try
            End If
        End Sub

        Sub OnPromptPassword(sender As Object, e As EventArgs)
            m_Answer = answer.Text
            If Not Page.IsValid Or    m_Answer=DBNull Then Exit Sub

            Dim theDB As New MemberDB
            Dim m_Password As String

            Try
                m_Password = theDB.PromptPassword(pin.Text,answer.Text)
                If    m_Password <> DBNull Then
                    password.Text    =    "    你    的    密    码    是    :    "+m_Password+"<a
href="+Request.Params("ReturnUrl")+">现在返回</A>"
                Else
                    password.Text = "无法提示你的密码。可能你没有正确地回答问题！"
                End If
            Catch e1 As Exception
                password.Text = "无法提示你的密码。可能你没有正确地回答问题！"
                Exit Sub
            End Try

        End Sub

</script>

<HTML>
<BODY>
<DIV Align="Center">
<H1>会员注册</H1>
<hr wifth=600>
</DIV>

<FORM RunAt="Server">
<TABLE WIDTH=600 BORDER=0 CELLSPACING=1 CELLPADDING=1 Align="Center">
```

```
    <TR>
        <TD COLSPAN="4">
            <asp:RegularExpressionValidator id="RegPin" runat="server"
                ControlToValidate="pin"
                ValidationExpression="^[\S^']{6,30}$"
                Display="Dynamic">
                    帐号中：某些特殊字符禁用！另外，必须至少 6 个字符！
            </asp:RegularExpressionValidator>
        </TD>
    </TR>

    <TR>
        <TD>账号</TD>
        <TD><asp:textbox id="pin" RunAt="server" OnTextChanged="OnPromptQuestion"
AutoPostBack="True"/>
            <asp:RequiredFieldValidator id="RFpin" runat="server"
                ControlToValidate="pin"
                ErrorMessage="*"
                ForeColor="Red">
            </asp:RequiredFieldValidator>
        </TD>

    <TR>
        <TD COLSPAN="1">密码提示问题</TD>
        <TD COLSPAN="3"><asp:textbox id="question" RunAt="server" Columns="50"
ReadOnly/>
        </TD>
    </TR>
    <TR>
        <TD COLSPAN="1">密码提示问题答案</TD>
        <TD COLSPAN="3"><asp:textbox id="answer" RunAt="server" Columns="50"/>
            <asp:Panel id="Panel" runat="server"/>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4"><font color='red'>
                <asp:label id="password" runat="server"/></font>
        </TD>
    </TR>

    <TR>
        <TD COLSPAN="4">
                <input type="submit" OnServerClick="OnPromptPassword" Value="请马上
```

```
提示" runat="server"/>
            </TD>
        </TR>
</TABLE>
</FORM>

</BODY>
</HTML>
```

4.6.6 连接数据库组件方法：

通过这个组件，我们获得在配置文件 Config.web 中的数据库的连接串，（csbook\appsoft\member\DBConn\DBConn.vb）：

```
Imports System
Imports System.Data
Imports System.Data.SQL
Imports System.Web
Imports System.Collections

Namespace DarkMan

    Public Class DBConn
        Shared m_ConnectionString As String

        Shared ReadOnly Property ConnectionString As String

            Get
                If m_ConnectionString = "" Then

                    Dim appsetting As Hashtable = CType(HttpContext.Current.GetConfig("appsettings"), Hashtable)
                    m_ConnectionString = CStr(appsetting("DSN"))

                    If m_ConnectionString = "" Then
                        throw new Exception("iShop DSN Value not set in Config.web")
                    End if

                End If

                return m_connectionString
```

```
            End Get

        End Property

    End Class


    Public Class SQLDataReaderResult

        Public conn As SQLConnection
        Public dr As SQLDataReader

        Public Sub Close()
            If conn.State = DBObjectState.Open then
                Try
                    dr.Close()
                    conn.Close()
                Catch e As Exception
                    throw e
                End Try
            End If
        End Sub
    End Class

End Namespace
```

4.6.7 操作组件方法：

这个组件里封装了对数据的插入、更新、密码提示方法、用户有效性的判断等方法。csbook\appsoft\member\com\member.vb:

```
Imports System
Imports System.Data
Imports System.Data.SQL

Namespace DarkMan

Public Class MemberDetail
    Public ID As Integer
    Public PIN As String
    Public Password As String
```

```
    Public Name As String
    Public Birth As DateTime
    Public Sex As Boolean
    Public City As Integer
    Public Zip As String
    Public EMail As String
    Public Telephone As String
    Public Address As String
    Public Question As String
    Public Answer As String
End Class

Public Class City
    Public Shared Function GetAll() As SQLDataReaderResult
        Dim myConnection As SQLConnection = new SQLConnection(DarkMan.DBConn.ConnectionString)
        Dim myCommand As SQLCommand=New SQLCommand("GetAllCities", myConnection)

        myCommand.CommandType = CommandType.StoredProcedure

        Dim result As New SQLDataReaderResult
        result.Conn = myConnection

        Try
            myConnection.Open()
            myCommand.Execute(result.dr)
        Catch e As Exception
            throw e
        End Try

        return result
    End Function
End Class

Public Class MemberDB

    '登记信息
    Public Function Register(ByRef user As MemberDetail) As Integer
        Return RegNMod(0,user)
    End Function

    '密码提示
    Public Function PromptPassword(ByVal PIN As String,ByVal Answer As String) As String
```

```
        Dim myConnection As SQLConnection = new SQLConnection(DarkMan.DBConn.ConnectionString)
        Dim myCommand As SQLCommand= new SQLCommand("PromptPassword", myConnection)

        myCommand.CommandType = CommandType.StoredProcedure
        Dim param As SQLParameter

        Param = new SQLParameter("@PIN", SQLDataType.NVarChar,30)
        Param.Value = PIN
        myCommand.Parameters.Add(Param)

        Param = new SQLParameter("@Answer", SQLDataType.NVarChar,50)
        Param.Value = Answer
        myCommand.Parameters.Add(Param)

        Param = new SQLParameter("@Password", SQLDataType.NVarChar,30)
        Param.Direction = ParameterDirection.Output
        myCommand.Parameters.Add(Param)

        Try
            myConnection.Open()
            myCommand.Execute()
        Catch e As Exception
            throw e
        Finally
            If myConnection.State = DBObjectState.Open then
                myConnection.Close()
            End If
        End Try

        return CStr( Param.Value )

    End Function

    '密码问题提示
    Public Function PromptQuestion(ByVal PIN As String) As String
        Dim myConnection As SQLConnection = new SQLConnection(DarkMan.DBConn.ConnectionString)
        Dim myCommand As SQLCommand= new SQLCommand("PromptQuestion", myConnection)

        myCommand.CommandType = CommandType.StoredProcedure
        Dim param As SQLParameter
```

```
            Param = new SQLParameter("@PIN", SQLDataType.NVarChar,30)
            Param.Value = PIN
            myCommand.Parameters.Add(Param)

            Param = new SQLParameter("@Question", SQLDataType.NVarChar,50)
            Param.Direction = ParameterDirection.Output
            myCommand.Parameters.Add(Param)

            Try
                myConnection.Open()
                myCommand.Execute()
            Catch e As Exception
                throw e
            Finally
                If myConnection.State = DBObjectState.Open then
                    myConnection.Close()
                End If
            End Try

            return CStr( Param.Value )

        End Function

        '修改信息
        Public Function Modify(ByRef user As MemberDetail) As Integer
            Return RegNMod(1,user)
        End Function



        Public Function GetDetailByID(ByVal ID As Integer) As MemberDetail
            return GetDetail(0,ID)
        End Function

        Public Function GetDetailByPIN(ByVal PIN As String) As MemberDetail
            return GetDetail(1,,PIN)
        End Function


        '判断是否为合法用户
        Public Function IsValidUser(ByVal PIN As String,ByVal Password As String) As Integer
            Dim myConnection As SQLConnection = new
SQLConnection(DarkMan.DBConn.ConnectionString)
```

```
        Dim myCommand As SQLCommand= new SQLCommand("IsValidUser", myConnection)

        myCommand.CommandType = CommandType.StoredProcedure
        Dim param As SQLParameter

        Param = new SQLParameter("@PIN", SQLDataType.NVarChar,30)
        Param.Value = PIN
        myCommand.Parameters.Add(Param)

        Param = new SQLParameter("@Password", SQLDataType.NVarChar,30)
        Param.Value = Password
        myCommand.Parameters.Add(Param)

        Param = new SQLParameter("@ID", SQLDataType.Int)
        Param.Direction = ParameterDirection.Output
        myCommand.Parameters.Add(Param)

        Try
            myConnection.Open()
            myCommand.Execute()
        Catch e As Exception
            throw e
        Finally
            If myConnection.State = DBObjectState.Open then
                myConnection.Close()
            End If
        End Try

        return CInt( Param.Value )
    End Function


    Protected Function GetDetail(ByVal op As Integer,Optional ByVal ID As Integer=0,Optional ByVal PIN As String="") As MemberDetail
        Dim theUser As MemberDetail=New MemberDetail

        Dim myConnection As SQLConnection = new SQLConnection(DarkMan.DBConn.ConnectionString)
        Dim myCommand As SQLCommand
        Select Case op
            Case 0    'By ID
                myCommand = new SQLCommand("GetDetailByID", myConnection)
                Dim param As SQLParameter
```

```
            Param = new SQLParameter("@ID", SQLDataType.Int)
            Param.Value = ID
            myCommand.Parameters.Add(Param)
        Case 1    'By PIN
            myCommand = new SQLCommand("GetDetailByPIN", myConnection)
            Dim param As SQLParameter
            Param = new SQLParameter("@PIN", SQLDataType.Int)
            Param.Value = PIN
            myCommand.Parameters.Add(Param)
        Case Else
            Exit Function
End Select

myCommand.CommandType = CommandType.StoredProcedure

Dim dr As SQLDataReader

Try
    myConnection.Open()
    myCommand.Execute(dr)
    If dr.Read Then
        theUser.PIN = dr.GetString(0)            'dr.GetValue(0,theUser.PIN)
        theUser.Password = dr.GetString(1)       'dr.GetValue(1,theUser.Password)
        theUser.Name = dr.GetString(2)     'dr.GetValue(2,theUser.Name)
        theUser.Birth = dr.GetDateTime(3)        'dr.GetValue(3,theUser.Birth)
        theUser.Sex = dr.GetSQLBit(4).BoolValue    'dr.GetValue(4,theUser.Sex)
        theUser.City = dr.GetInt32(5) 'dr.GetValue(5,theUser.City)
        theUser.Zip = dr.GetString(6) 'dr.GetValue(6,theUser.Zip)
        theUser.EMail = dr.GetString(7)     'dr.GetValue(7,theUser.EMail)
        theUser.Telephone = dr.GetString(8)      'dr.GetValue(8,theUser.Telephone)
        theUser.Address = dr.GetString(9) 'dr.GetValue(9,theUser.Address)
        theUser.Question = dr.GetString(10)      'dr.GetValue(10,theUser.Question)
        theUser.Answer = dr.GetString(11)        'dr.GetValue(11,theUser.Answer)
        theUser.ID = dr.GetInt32(12)             'dr.GetValue(0,theUser.ID)
    End If
Catch e As Exception
    throw e
Finally
    If myConnection.State = DBObjectState.Open then
        myConnection.Close()
    End If
End Try

Return theUser
```

```
End Function


Protected Function RegNMod( ByVal op As Integer,ByRef user As MemberDetail) As Integer
        Dim myConnection As SQLConnection = new SQLConnection(DarkMan.DBConn.ConnectionString)

        Dim myCommand As SQLCommand
        Dim ParamID As New SQLParameter("@ID", SQLDataType.Int)
        Select Case op

            Case 0    'Insert
                myCommand = new SQLCommand("Register", myConnection)
                ParamID.Direction = ParameterDirection.Output
                myCommand.Parameters.Add(ParamID)
            Case 1    'Modify
                myCommand = new SQLCommand("Modify", myConnection)
                ParamID.Value = user.ID
                myCommand.Parameters.Add(ParamID)
            Case Else
                Exit Function
        End Select

        myCommand.CommandType = CommandType.StoredProcedure
        Dim param As SQLParameter

        Param = new SQLParameter("@PIN", SQLDataType.NVarChar, 30)
        Param.Value = user.PIN
        myCommand.Parameters.Add(Param)

        Param = new SQLParameter("@Password", SQLDataType.NVarChar, 30)
        Param.Value = user.Password
        myCommand.Parameters.Add(Param)

        Param = new SQLParameter("@Name", SQLDataType.NVarChar, 30)
        Param.Value = user.Name
        myCommand.Parameters.Add(Param)

        Param = new SQLParameter("@Birth", SQLDataType.SmallDateTime)
        Param.Value = user.Birth
        myCommand.Parameters.Add(Param)
```

```
Param = new SQLParameter("@Sex", SQLDataType.Bit)
Param.Value = user.Sex
myCommand.Parameters.Add(Param)

Param = new SQLParameter("@City", SQLDataType.Int)
Param.Value = user.City
myCommand.Parameters.Add(Param)

Param = new SQLParameter("@Zip", SQLDataType.NVarChar, 6)
Param.Value = user.Zip
myCommand.Parameters.Add(Param)

Param = new SQLParameter("@EMail", SQLDataType.NVarChar, 30)
Param.Value = user.EMail
myCommand.Parameters.Add(Param)

Param = new SQLParameter("@Telephone", SQLDataType.NVarChar, 30)
Param.Value = user.Telephone
myCommand.Parameters.Add(Param)

Param = new SQLParameter("@Address", SQLDataType.NVarChar, 50)
Param.Value = user.Address
myCommand.Parameters.Add(Param)

Param = new SQLParameter("@Question", SQLDataType.NVarChar, 50)
Param.Value = user.Question
myCommand.Parameters.Add(Param)

Param = new SQLParameter("@Answer", SQLDataType.NVarChar, 50)
Param.Value = user.Answer
myCommand.Parameters.Add(Param)


Try
    myConnection.Open()
    myCommand.Execute()
Catch e As Exception
    throw e
Finally
    If myConnection.State = DBObjectState.Open then
        myConnection.Close()
    End If
End Try
```

```
            If op=0 Then
                return CInt(ParamID.Value)
            Else
                return user.ID
            End If

        End Function

End Class

End Namespace
```

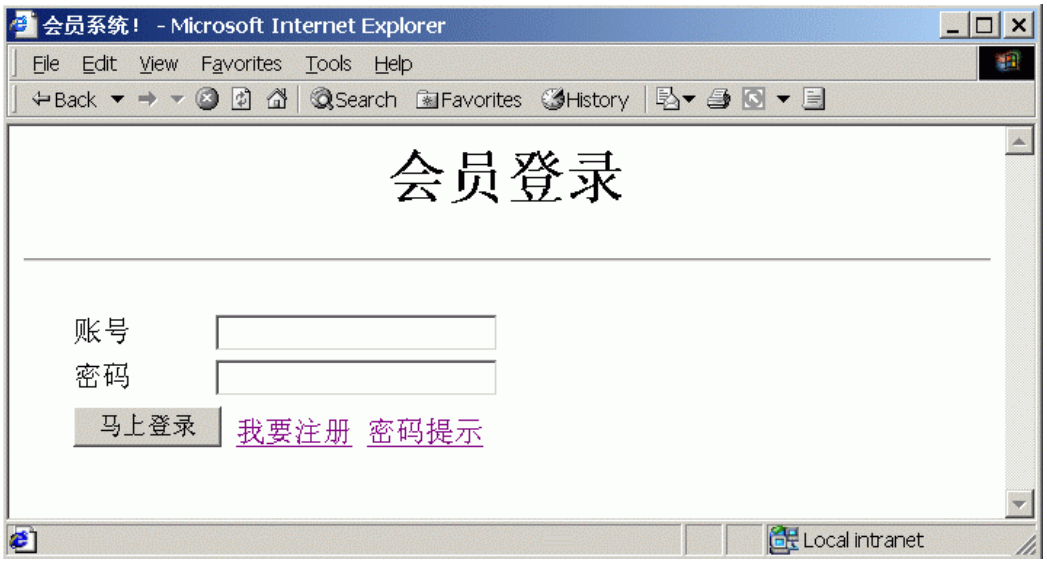
4.6.8 配置文件：

我们把数据库的连接字符串方法在配置文件 Config.web 里面：
（csbook\appsoft\member\Config.web）：

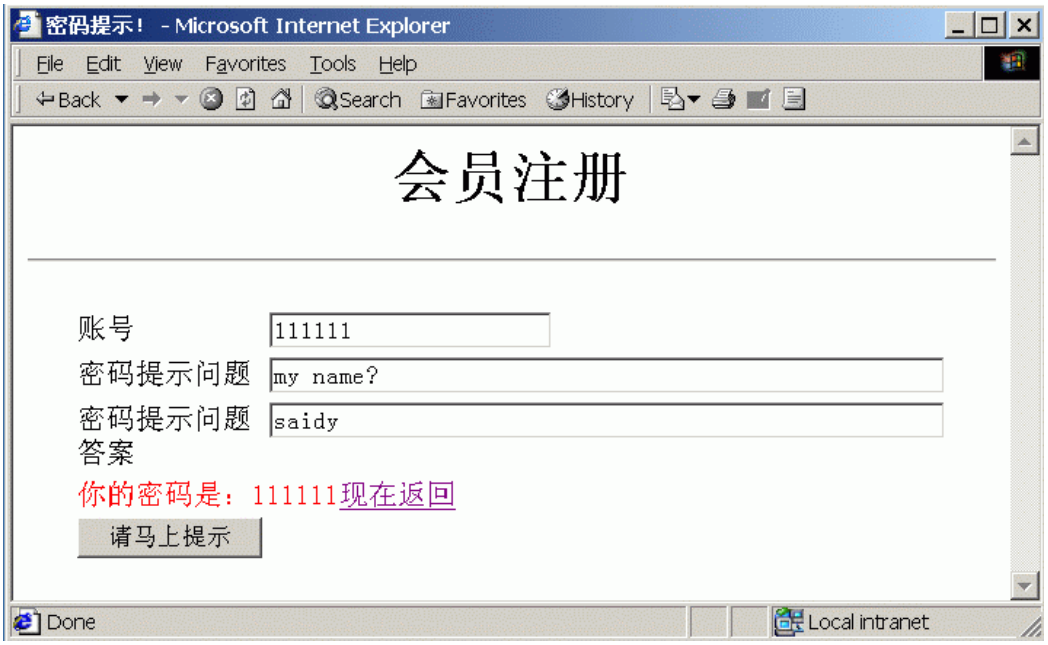
```
<configuration>
    <!--数据库连接字符串  -->
    <appsettings>
        <add key="DSN" value="server=localhost;uid=sa;pwd=;database=darkman" />
    </appsettings>

    <globalization requestencoding="gb2312" responseencoding="gb2312" />
</configuration>
```

4.6.9 运行效果：

登录页面：



密码提示页面：



4.6.10 小结

本章利用一个会员系统来讲述了一个利用.NET 的 Application 的应用。

第五章 Web Service

第一章 Web service 简介

现在 Internet 正在不断地发展着，在互联网应用刚开始的时候，我们浏览的网页只是静态的，不可交互的。而现在随着技术的日益发展，将提供给网页浏览者一个可编程的 Web 站点。这些站点将在组织、应用、服务、驱动上更加紧密的结合在一起，这些站点将通过一些应用软件直接连接到另一个 Web 站点，这些可编程的 Web 站点相比传统的 web 站点来说，将变得更加能重复使用，也更加智能化！

.net 平台给我们提供了一种运行环境，即公用语言运行环境（CLR，Common Language Runtime）。对 CLR 来说，它提供了一种内置机制来创建一个可编程的站点，、对于 Web 程序开发者和 VB 程序员来说，这将是一致、熟悉的。这种模型是可以重复使用，也可以再扩展。 它包含了开放的 Internet 标准（HTTP, XML, SOAP, SDL）。以便它能被网页浏览者访问。

ASP.NET 使用.asmx 文件来对 Web Services 的支持。.asmx 文件和.aspx 文件一样都属于文本文件。它包含在.aspx 文件之中，成为 ASP.NET 应用程序的一部分。

下面我们将举一个简单的例子来介绍.asmx 文件，我们还是从“Hello，World”这个经典的例子说起，代码如下：

```
<!-- 文件名：webservice\sisam.asmx -->
<%@ WebService Language="VB" Class="HelloWorld" %>
    Imports System.Web.Services
    Public Class HelloWorld :Inherits WebService
    Public Function <WebMethod( )> SayHelloWorld( ) As String
            Return("Hello World")
   End Function
    End Class
```

说明：

1. 编码最开始必须进行 WebService 声明，从而定义这个文件为一个 Web Service。而且，在同一行中设置好编程语言的类型。

2. 然后，引入名字空间 System.Web.Services。注意，这个名字空间属于最基本的元素，必须要包含它。

3. 接着，声明 service 中的功能模块，也就是类模块，这里的类名叫 HelloWorld。这个类来源于基类 WebService，而且应该是 public 类型。

4．最后，定义 service 的可访问方法。在表示方法的符号前面，要设置好自定义属性。对应于 C#语言，属性值就是[WebMethod]；对应于 VB，就是。如果没有设置这个属性，那么这个方法就不能从 service 中访问。一个局部应用可以使用任何的 public 类型的类，但是只有具备[WebMethod]的类才可以通过 SOAP 被远程地访问。

当对 service 的请求发生时，.asmx 文件将自动地被 ASP.NET 运行环境所编译。随后的请求就可以由缓冲的预编译类型对象执行。

为了测试编写好的代码，我们需用一个支持 ASP.NET 的 Web 服务器。假设这个 Web 服务器的名称叫做 server1，其上有一个虚拟目录 test。请跟随下面步骤开始测试：

1．将代码保存为 HelloWorld.asmx

2．放到 Web 服务器 Foo 的虚拟目录 Bar 下

3．打开 Internet Explorer5，在地址栏输入 http://server1/test/HelloWorld.asmx

这时，我们将看到关于这个 Web Service 的公用方法－也就是那些标记为 WebMethod 属性的字符，并得知调用这些方法可以使用的协议，比如 SOAP 或者 HTTP GET。

在 Internet Explorer 的地址栏中输入 http://Foo/Bar/HelloWorld.asmx?SDL 后，将产生基于服务描述语言（Service Description Language ：SDL）语法的具备相同信息的 XML 文件。这个 SDL 文件非常重要，客户端就是使用它来访问 service。

我们来看一下程序运行的效果：

从客户端进行访问：

除了允许开发者使用的创建 Web Services 的技术以外，Microsoft 的.NET 框架给客户端

提供了一套访问并使用 Web Services 的精致且高深的工具和代码。由于 Web Services 是基于如简单对象访问协议 SOAP（Simple Object Access Protocol）和 HTTP 这样的开放协议标准的，从而，我们就可以使用这种客户端技术使用非 ASP.NET 的 Web Services。当然，这也需用高水平地合成 ASP.NET Web Services 和这种客户端技术。

SDK 中有一个工具叫做 WebServiceUtil.exe，我们可以使用它来下载一个 Web Services 的 SDL 描述语言，并创建表达这个 Service 的代理类。比如，当我们输入以下命令，就可以创建一个叫做 HelloWorld.cs 的代理类：

WebServiceUtil /c:proxy /pa:http://someDomain.com/someFolder/HelloWorld.asmx?SDL

这个类看起来与前面创建的类非常相似。它包含一个方法 SayHelloWorld，该方法返回一个字符串。将这个代理类编译到一个应用程序中，然后调用这个代理类的方法，结果就是：通过 HTTP，这个代理类包装 SOAP 请求，然后接收 SOAP 编码响应，最后汇集成为一个字符串。

从客户端来看，代码是很简单的，返回的结果也很简单，就是一个字符串"Hello World"。同样为了对照方便，我们列出了使用 VB、C#以及 JScript 三种语言编写的代码：

C#

```
HelloWorld myHelloWorld = new HelloWorld();
String sReturn = myHelloWorld.SayHelloWorld();
```

VB

```
Dim myHelloWorld As New HelloWorld()
Dim sReturn As String = myHelloWorld.SayHelloWorld()
```

JScript

```
var myHelloWorld:HelloWorld = new HelloWorld();
var sReturn:String = myHelloWorld.SayHelloWorld();
```

通过上面的例程，你可能对 Web Services 有了初步的印象。下面，我们将介绍 Web Services 中涉及到的各种数据类型，也就是 Web Services 方法的输入/输出参数类型。因为 Web Services 的执行是建立在 XML 架构之上的，所以它能够支持丰富的数据类型。下表列出了使用 SOAP 协议时 Web Services 支持的数据类型：

类型	描述
基础类型	也即标准基础类型，包括： String、Int32、Byte、Boolean、Int16、Int64、Single、Double、Decimal 、 DateTime(类 似 XML 中 的 timeInstant)、DateTime(类似 XML 中的 date)、DateTime(类 似 XML 中 的 time) 以 及 XmlQualifiedName(类似 XML 中的 QName)。
枚举类型	枚举类型。例如：. "public enum color { red=1, blue=2 }"

基础，枚举数组	上面提到的类型数组。例如：string[] 和 int[]
类和结构	带有公用域或属性的类和结构，公用域和属性是串行结构的
类和结构体数组	上述类型的数组
DataSet	ADO.NET DataSet 类型。DataSets 能在类和结构体作为字段来使用。
DataSet 数组	上述类型的数组
XmlNode	XmlNode 是 XML 文档片断的内存表示，就好像一个轻量级的 XML 文档对象模型。比如说，"" 就可以存储在一个 XmlNode 类型变量中。我们可以将 XmlNodes 作为参数传递，以 SOAP 兼容方式附加到传递给 Web Services 的 XML 文档上。返回值也是同样原理。XmlNode 也可看成是类或结构中的字段。
XmlNode 数组	上述类型的数组

当通过 SOAP 或者 HTTP GET/POST 调用 Web Services 时，返回值可以是上述提到的任何一种数据类型。

参数的数据类型

使用 SOAP 协议时，"通过值"以及"通过引用"这两种输入/输出参数形式都可被支持。如果是"通过引用"的参数类型，就会产生两种方式的数据发送效果：到服务器的以及返回到客户端的。但是，当通过 HTTP GET/POST 传递输入参数给 Web Services 时，就只支持有限的数据类型了，而且还必须是"通过值"形式的参数。这些类型如下：

类型	描述
基础类型 (有限的)	支持大多数标准基础类型，包括： Int32、String、Int16、Int64、Boolean、Single、Double、Decimal、DateTime、TimeSpan、UInt16、UInt32、UInt64 和 Currency。从客户端来看，所有这些类型都转变为 string。
枚举类型	比如： "public enum color { red=1, blue=2 }"。
基础类型数组，枚举类型数组	上述类型的数组，比如 string[]和 int[]

现在我们将举一个例子，来说明上面我们介绍的数据类型：

这个例子利用 WebServiceUtil.exe 建立的 SOAP 代理来使用上面列出的数据类型。注意：因为在.asmx 文件中定义了多于一个的公用类，所以，我们必须要指定哪一个作为 WebService 类，这可以通过设置 WebService 标识的 Class 属性来实现，代码如下：

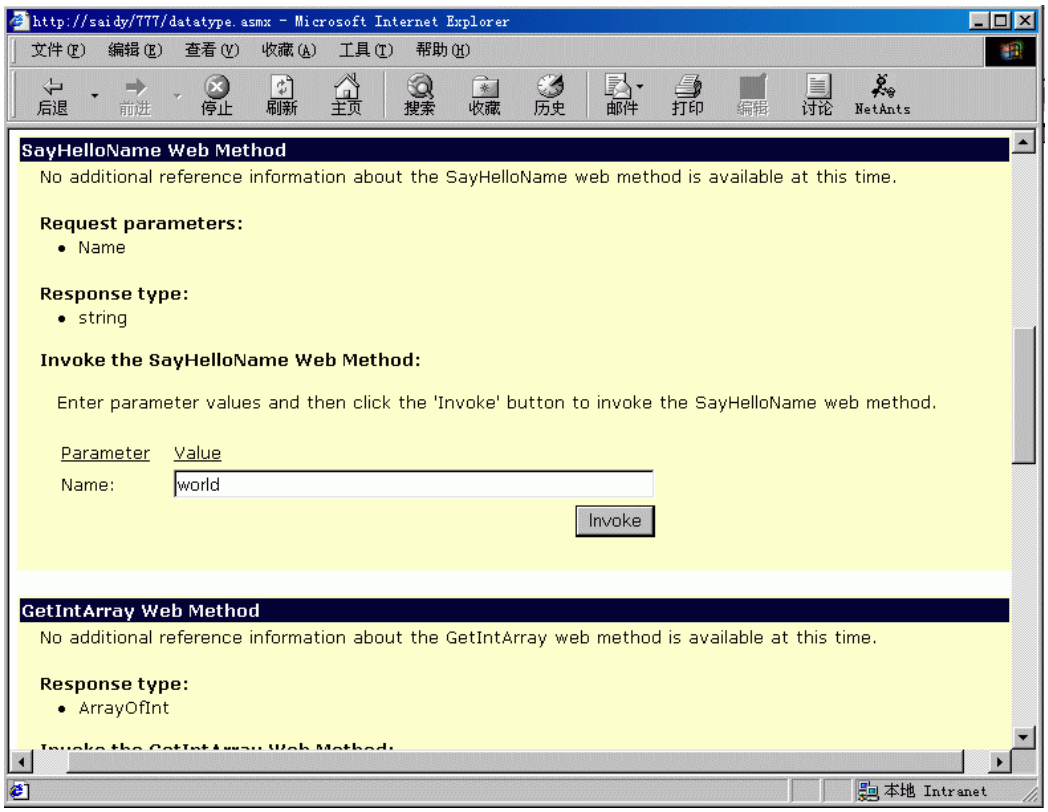
```
<%@ WebService Language="C#" Class="DataTypes" %>
```

源文件 webservice\datatype.asmx 的内容如下：

```
<%@ WebService Language="VB" Class="DataTypes" %>

Imports System
Imports System.Web.Services

Public Enum Mode

    EOn = 1
    EOff = 2

End Enum

Public Class Order

    Public OrderID As Integer
    Public Price As Double
End Class


Public Class DataTypes

  'SayHello 方法显示从 service 中返回的一个字符串信息。
    Public Function <WebMethod()> SayHello() As String

        Return "Hello World!"
    End Function


  'SayHelloName 方法返回一个字符串，并接受一个字符串参数。
    Public Function <WebMethod()> SayHelloName(Name As String) As String

        Return "Hello" & Name
    End Function


  'GetIntArray 方法显示了如何返回一个整数数组。
    Public Function <WebMethod()> GetIntArray() As Integer()

        Dim I As Integer
        Dim A(5) As Integer
```

```
        For I = 0 to 4
            A(I) = I*10
        Next
        Return A
    End Function


‘GetMode 方法返回一个枚举数值。
    Public Function <WebMethod()> GetMode() As Mode

        Return Mode.EOff
    End Function


‘GetOrder 方法返回一个类。
    Public Function <WebMethod()> GetOrder() As Order

        Dim MyOrder As New Order

        MyOrder.Price=34.5
        MyOrder.OrderID = 323232

        Return MyOrder
    End Function


‘GetOrders 方法返回定单对象数组。
    Public Function <WebMethod()> GetOrders() As Order()

        Dim MyOrder(2) As Order

        MyOrder(0) = New Order()
        MyOrder(0).Price=34.5
        MyOrder(0).OrderID = 323232
        MyOrder(1) = New Order()
        MyOrder(1).Price=99.4
        MyOrder(1).OrderID = 645645

        Return MyOrder
    End Function

End Class
```

程序运行的效果如下：



当我们单击 invoke 的时候，将显示：

```
<?xml version="1.0" ?>
<string xmlns="http://tempuri.org/">Helloworld</string>
```

对于使用客户端应用程序而言，使用 WebServiceUtil 代理生成工具配置这些数据类型是透明的。请看关于 Web Service 的一个客户端例程：

客户端访问的文件：clint.aspx，内容如下：

```
<%@ Import Namespace="DataTypesService" %>

<html>
<style>
  div
  {
    font: 8pt verdana;
    background-color:cccccc;
    border-color:black;
    border-width:1;
```

```
    border-style:solid;
    padding:10,10,10,10;
  }

</style>

<script language="VB" runat="server">

    Public Sub Page_Load(Sender As Object, E As EventArgs)

        Dim D As DataTypes = New DataTypes()
        Message1.InnerHtml = D.SayHello()
        Message1.InnerHtml = Message1.InnerHtml & D.SayHelloName("Bob")
        Message3.InnerHtml = Message3.InnerHtml & D.GetMode()

        Dim MyIntArray As Integer() = D.GetIntArray()
        Dim MyString As String = "Contents of the Array:<BR>"

        For I = 0 To MyIntArray.Length - 1
            MyString = MyString & MyIntArray(I) & "<BR>"
        Next

        Message2.InnerHtml = Message2.InnerHtml & MyString

        Dim MyOrder As Order = D.GetOrder()
        Message4.InnerHtml = Message4.InnerHtml & "<BR>OrderID: " & MyOrder.OrderID
        Message4.InnerHtml = Message4.InnerHtml & "<BR>Price: " & MyOrder.Price

        Dim MyOrders As Order() = D.GetOrders()
        Message5.InnerHtml = Message5.InnerHtml & "<BR>OrderID: " &
MyOrders(0).OrderID
        Message5.InnerHtml = Message5.InnerHtml & "<BR>Price: " & MyOrders(0).Price

    End Sub

</script>

<body style="font: 10pt verdana">
<H4>Using DataTypes with Web Services</H4>

  <h5>Methods that return a Primitive (String): </h5>
  <div id="Message1" runat="server"/>

  <h5>Methods that return an Array of Primitives (Integers): </h5>
```

```
    <div id="Message2" runat="server"/>

    <h5>Method that returns an Enum: </h5>
    <div id="Message3" runat="server"/>

    <h5>Method that returns a Class/Struct: </h5>
    <div id="Message4" runat="server"/>

    <h5>Method that returns an array of Classes/Structs: </h5>
    <div id="Message5" runat="server"/>

</body>
</html>
```

在客户端程序中，我们使用<%@ Import Namespace="DataTypesService" %>来引入 DataTypesService 这个我们自定义的名字空间。然后在程序中只是调用了 DataTypesService 中的方法。

现在我们来看如何生成名字空间：

1． Datatype.vb 中的内容：

```
Imports System.Xml.Serialization
Imports System.Web.Services.Protocols
Imports System.Web.Services

Namespace DataTypesService
  Public Class    DataType
     Inherits System.Web.Services.Protocols.SoapClientProtocol
Public Sub New()
MyBase.New
Me.Url="http://localhost/QuickStart/aspplus/samples/services/DataTypes/VB/DataTypes.asmx"
End Sub

Public Function
<System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/SayHello")>
SayHello() As String
                Dim results() As Object = Me.Invoke("SayHello", New Object(0) {})
                Return CType(results(0),String)
            End Function
            Public Function BeginSayHello(ByVal callback As System.AsyncCallback, ByVal
asyncState As Object) As System.IAsyncResult
                Return Me.BeginInvoke("SayHello", New Object(0) {}, callback, asyncState)
            End Function
            Public Function EndSayHello(ByVal asyncResult As System.IAsyncResult) As String
```

```
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),String)
    End Function
    Public Function <System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/SayHelloName")>
SayHelloName(ByVal <System.Xml.Serialization.XmlElementAttribute("Name", IsNullable:=true)> name As String) As String
        Dim results() As Object = Me.Invoke("SayHelloName", New Object() {name})
        Return CType(results(0),String)
    End Function
    Public Function BeginSayHelloName(ByVal name As String, ByVal callback As System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
        Return Me.BeginInvoke("SayHelloName", New Object() {name}, callback, asyncState)
    End Function
    Public Function EndSayHelloName(ByVal asyncResult As System.IAsyncResult) As String
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),String)
    End Function
    Public Function <System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/GetIntArray"), _
    System.Xml.Serialization.XmlArrayAttribute(IsNullable:=true, ArrayType:=System.Xml.Serialization.XmlArrayType.Soap), _
    System.Xml.Serialization.XmlArrayItemAttribute("int", IsNullable:=false)>
GetIntArray() As Integer()
        Dim results() As Object = Me.Invoke("GetIntArray", New Object(0) {})
        Return CType(results(0),Integer())
    End Function
    Public Function BeginGetIntArray(ByVal callback As System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
        Return Me.BeginInvoke("GetIntArray", New Object(0) {}, callback, asyncState)
    End Function
    Public Function EndGetIntArray(ByVal asyncResult As System.IAsyncResult) As Integer()
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),Integer())
    End Function
    Public Function <System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/GetMode")>
GetMode() As Mode
        Dim results() As Object = Me.Invoke("GetMode", New Object(0) {})
        Return CType(results(0),Mode)
    End Function
```

```
        Public Function BeginGetMode(ByVal callback As System.AsyncCallback, ByVal
asyncState As Object) As System.IAsyncResult
            Return Me.BeginInvoke("GetMode", New Object(0) {}, callback, asyncState)
        End Function
        Public Function EndGetMode(ByVal asyncResult As System.IAsyncResult) As Mode
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),Mode)
        End Function
        Public                                                          Function
<System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/GetOrder")>
GetOrder() As Order
            Dim results() As Object = Me.Invoke("GetOrder", New Object(0) {})
            Return CType(results(0),Order)
        End Function
        Public Function BeginGetOrder(ByVal callback As System.AsyncCallback, ByVal
asyncState As Object) As System.IAsyncResult
            Return Me.BeginInvoke("GetOrder", New Object(0) {}, callback, asyncState)
        End Function
        Public Function EndGetOrder(ByVal asyncResult As System.IAsyncResult) As Order
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),Order)
        End Function
        Public                                                          Function
<System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/GetOrders"),  _
        System.Xml.Serialization.XmlArrayAttribute(IsNullable:=true,
ArrayType:=System.Xml.Serialization.XmlArrayType.Soap),  _
        System.Xml.Serialization.XmlArrayItemAttribute("Order",          IsNullable:=true)>
GetOrders() As Order()
            Dim results() As Object = Me.Invoke("GetOrders", New Object(0) {})
            Return CType(results(0),Order())
        End Function
        Public Function BeginGetOrders(ByVal callback As System.AsyncCallback, ByVal
asyncState As Object) As System.IAsyncResult
            Return Me.BeginInvoke("GetOrders", New Object(0) {}, callback, asyncState)
        End Function
        Public Function EndGetOrders(ByVal asyncResult As System.IAsyncResult) As Order()
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),Order())
        End Function

    End Class
    Public          Enum          <System.Xml.Serialization.XmlRootAttribute("result",
[Namespace]:="http://tempuri.org/", IsNullable:=false)> Mode
```

```
        EOn
        EOff

    End Enum
    Public Class <System.Xml.Serialization.XmlRootAttribute("result",
[Namespace]:="http://tempuri.org/", IsNullable:=true)> Order

        Public OrderID As Integer
        Public Price As Double

    End Class
End Namespace
```

在这个 vb 文件中，我们定义了一个名字空间 DataTypesService。
请看 vb 文件的其中一段代码段：

```
Public Function <System.Web.Services.Protocols.SoapMeth_&
 odAttribute("http://tempuri.org/SayHello")> SayHello() As String
            Dim results() As Object = Me.Invoke("SayHello", New Object(0) {})
            Return CType(results(0),String)
        End Function

Public Function BeginSayHello(ByVal callback As System.AsyncCallback, ByVal asyncState As
Object) As System.IAsyncResult
            Return Me.BeginInvoke("SayHello", New Object(0) {}, callback, asyncState)
End Function

Public Function EndSayHello(ByVal asyncResult As System.IAsyncResult) As String
            Dim results() As Object = Me.EndInvoke(asyncResult)
            Return CType(results(0),String)
End Function
```

上面的代码，是触发 invoke 的单击事件。然后，调用我们在 datatype.asmx 中定义的方法。

要生成上面的名字空间，我们使用 webserviceutil.exe 来编译。

```
webserviceutil -c:proxy /pa:DataTypes.sdl /l:VB /n:DataTypesService
```

5.1.1 小结

web service 提供了在不同体系机构下构建的网站之间相互提供应用接口服务、数据的一种方案。它采用通用的 SOAP、HTTP 以及 XML，就可以把原本互不相干的站点服务形成一整套分布的、自动化和智能化的网络应用，大大减轻了程序员的开发工作量，充分地利用了已经拥有的网络资源和开发资源。在 asp.net 中，web service 文件后缀名采用.asmx，开始应使用<%@ WebService …%>申明，接着引用 System.Web.Services 命名空间，然后定义一个

公用类继承自 WebService 基类，最后实现自己的类，其中向网络开放的功能，应在其方法前面加上 WebMethod 属性。

第二章 一个简单的 Web Service 案例

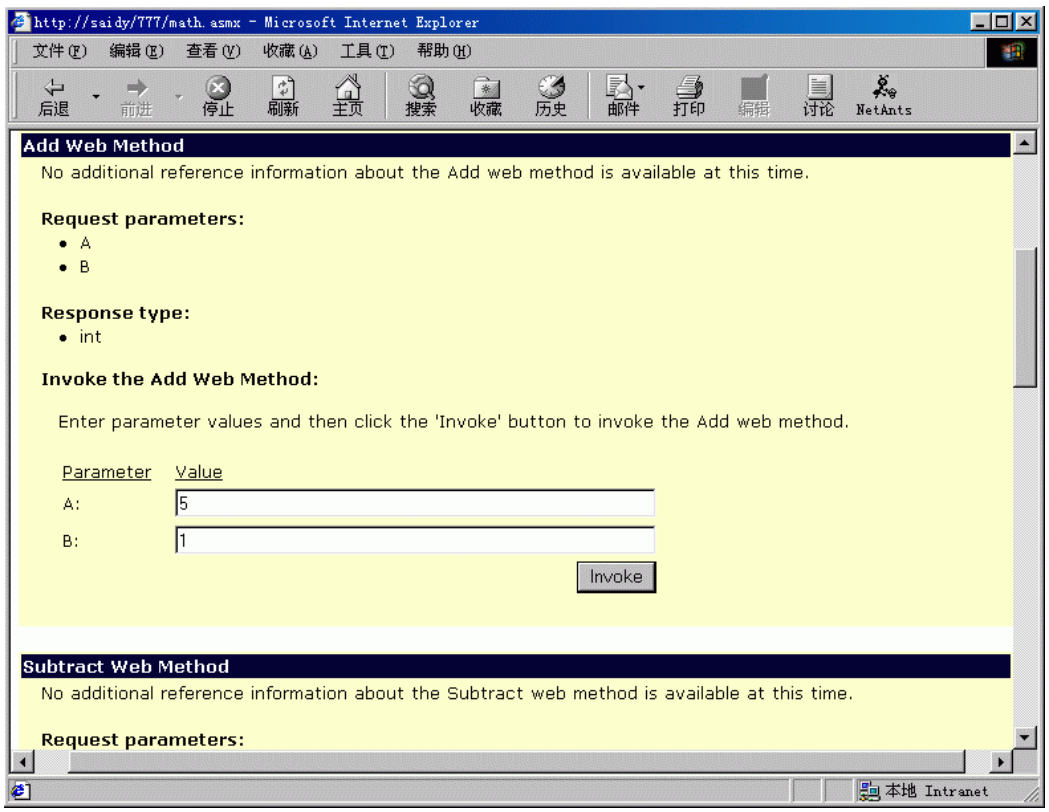
在这个例子中，我们将定义一个 mathservice 类，来对两个数字分别进行加，减，乘，除。当然这个类需要从基类 web service 中继承。请先看该程序的源代码：

源文件：webservice\math.asmx

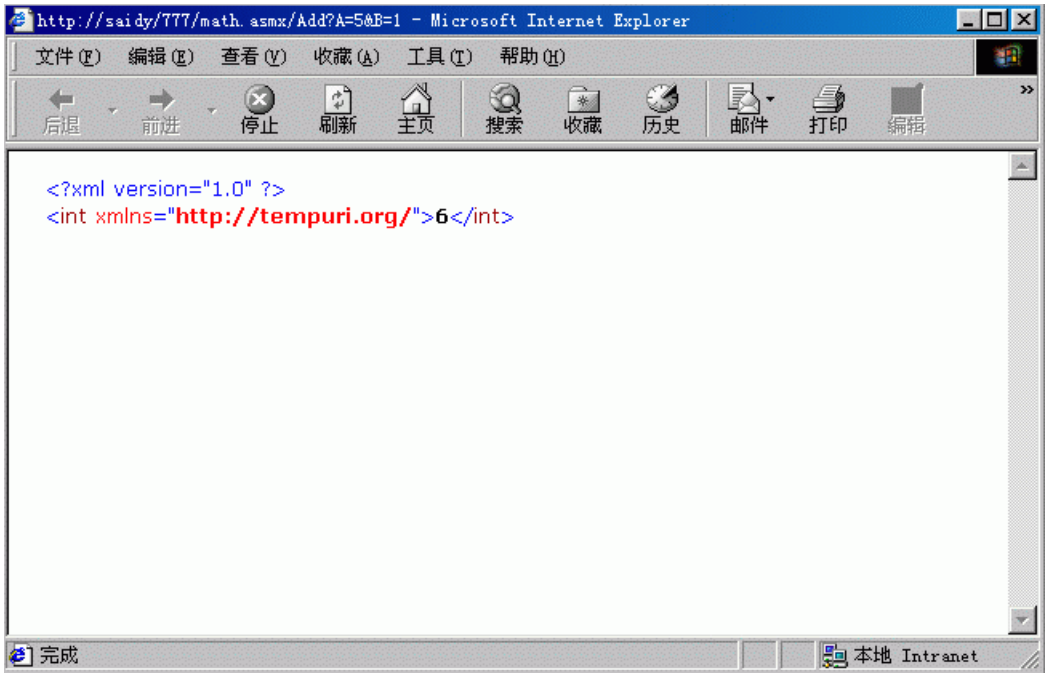
```
<%@ WebService Language="VB" Class="MathService" %>
Imports System
Imports System.Web.Services
Public Class MathService : Inherits WebService
    Public Function <WebMethod()> Add(A As Integer, B As Integer) As Integer
        Return A + B
    End Function
    Public Function <WebMethod()> Subtract(A As Integer, B As Integer) As Integer

        Return A - B
    End Function
    Public Function <WebMethod()> Multiply(A As Integer, B As Integer) As Integer
        Return A * B
    End Function
    Public Function <WebMethod()> Divide(A As Integer, B As Integer) As Integer
        If B = 0
            Return -1
        End If
        Return CInt(A / B)
    End Function
End Class
```

对于该程序，我们首先要用<%@ WebService Language="vb" Class="MathService" %>来标识。然后定义对应的方法。程序很简单，我们来看一下运行效果，假使我们使用“加”。如图：

我们要计算 5+1 的值：



通过上面的计算，结果就显示出来了。



一个代理类（proxy class）被创建出来的，我们就可以很容易的创建对象。当然，类中的方法就能够被对象调用。创建代理类，我们可以使用 WebServiceUtil.exe 来创建一个代理类。

我们还是举“加”，“减”，“乘”，“除”的例子。

我们先创建一个文件用于客户端的用户浏览：

```
<%@ Import Namespace="MathServiceSpace" %>
<html>
<script language="VB" runat="server">
   Dim Op1 As Integer = 0
   Dim Op2 As Integer = 0
   Public Sub Submit_Click(Sender As Object, E As EventArgs)
         Try
            Op1 = Int32.Parse(Operand1.Text)
            Op2 = Int32.Parse(Operand2.Text)
         Catch Exp As Exception
               ' Ignored
         End Try

         Dim Service As MathService = New MathService()
         Select (CType(sender,Control).ID)
            Case "Add" :
                  Result.Text = "<b>Result</b> = " & Service.Add(Op1, Op2).ToString()
            Case "Subtract" :
                  Result.Text = "<b>Result</b> = " & Service.Subtract(Op1, Op2).ToString()
            Case "Multiply" :
                  Result.Text = "<b>Result</b> = " & Service.Multiply(Op1, Op2).ToString()
            Case "Divide" :
                  Result.Text = "<b>Result</b> = " & Service.Divide(Op1, Op2).ToString()
         End Select
   End Sub
</script>
<body style="font: 10pt verdana">
   <h4>Using a Simple Math Service </h4>
   <form runat="server">
   <div
style="padding:15,15,15,15;background-color:beige;width:300;border-color:black;border-width:1;
border-style:solid">
      Operand 1: <br><asp:TextBox id="Operand1" Text="15" runat="server"/><br>
      Operand 2: <br><asp:TextBox id="Operand2" Text="5" runat="server"/><p>
      <input type="submit" id="Add" value="Add" OnServerClick="Submit_Click"
runat="server">
      <input type="submit" id="Subtract" value="Subtract" OnServerClick="Submit_Click"
runat="server">
      <input type="submit" id="Multiply" value="Multiply" OnServerClick="Submit_Click"
runat="server">
      <input type="submit" id="Divide" value="Divide" OnServerClick="Submit_Click"
```

```
runat="server">
      <p>
      <asp:Label id="Result" runat="server"/>
   </div>
   </form>
</body>
</html>
```

我们还需要一个 sdl 文件，当然这个文件不用手工输入，我们在浏览一个.asmx 的时候，在后缀名后直接加上？sdl 可以自动生成 sdl 文件。

然后我们在一个.vb 文件里，将定义一个名字空间，.vb 文件的内容：

```
Imports System.Xml.Serialization
Imports System.Web.Services.Protocols
Imports System.Web.Services

Namespace MathServiceSpace
      Public Class    MathService
            Inherits System.Web.Services.Protocols.SoapClientProtocol

            Public Sub New()
                  MyBase.New
                  Me.Url                                                    =
"http://localhost/QuickStart/aspplus/samples/services/MathService/VB/MathService.a"& _
"smx"
            End Sub
           Public                                                    Function
<System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/Add")>      Add(ByVal
<System.Xml.Serialization.XmlElementAttribute("A",  IsNullable:=false)>  a  As  Integer,  ByVal
<System.Xml.Serialization.XmlElementAttribute("B",  IsNullable:=false)>  b  As  Integer)  As
Integer
                  Dim results() As Object = Me.Invoke("Add", New Object() {a, b})
                  Return CType(results(0),Integer)
            End Function
            Public Function BeginAdd(ByVal a As Integer, ByVal b As Integer, ByVal callback As
System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
                  Return Me.BeginInvoke("Add", New Object() {a, b}, callback, asyncState)
            End Function
            Public Function EndAdd(ByVal asyncResult As System.IAsyncResult) As Integer
                  Dim results() As Object = Me.EndInvoke(asyncResult)
                  Return CType(results(0),Integer)
            End Function
            Public                                                    Function
<System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/Subtract")>
Subtract(ByVal  <System.Xml.Serialization.XmlElementAttribute("A",  IsNullable:=false)>  a  As
Integer,  ByVal  <System.Xml.Serialization.XmlElementAttribute("B",  IsNullable:=false)>  b  As
```

```
Integer) As Integer
        Dim results() As Object = Me.Invoke("Subtract", New Object() {a, b})
        Return CType(results(0),Integer)
    End Function
    Public Function BeginSubtract(ByVal a As Integer, ByVal b As Integer, ByVal callback
As System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
        Return Me.BeginInvoke("Subtract", New Object() {a, b}, callback, asyncState)
    End Function
    Public Function EndSubtract(ByVal asyncResult As System.IAsyncResult) As Integer
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),Integer)
    End Function
    Public                                                    Function
<System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/Multiply")>
Multiply(ByVal <System.Xml.Serialization.XmlElementAttribute("A", IsNullable:=false)> a As
Integer, ByVal <System.Xml.Serialization.XmlElementAttribute("B", IsNullable:=false)> b As
Integer) As Integer
        Dim results() As Object = Me.Invoke("Multiply", New Object() {a, b})
        Return CType(results(0),Integer)
    End Function
    Public Function BeginMultiply(ByVal a As Integer, ByVal b As Integer, ByVal callback
As System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
        Return Me.BeginInvoke("Multiply", New Object() {a, b}, callback, asyncState)
    End Function
    Public Function EndMultiply(ByVal asyncResult As System.IAsyncResult) As Integer
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),Integer)
    End Function
    Public                                                    Function
<System.Web.Services.Protocols.SoapMethodAttribute("http://tempuri.org/Divide")>
Divide(ByVal <System.Xml.Serialization.XmlElementAttribute("A", IsNullable:=false)> a As
Integer, ByVal <System.Xml.Serialization.XmlElementAttribute("B", IsNullable:=false)> b As
Integer) As Integer
        Dim results() As Object = Me.Invoke("Divide", New Object() {a, b})
        Return CType(results(0),Integer)
    End Function
    Public Function BeginDivide(ByVal a As Integer, ByVal b As Integer, ByVal callback
As System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
        Return Me.BeginInvoke("Divide", New Object() {a, b}, callback, asyncState)
    End Function
    Public Function EndDivide(ByVal asyncResult As System.IAsyncResult) As Integer
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),Integer)
    End Function
```

End Class
End Namespace
有了这四个文件，我们可以编辑一个批处理文件，执行如下的语句：

webserviceutil -c:proxy /pa:MathService.sdl /l:VB /n:MathServiceSpace

这样，我们就可以在客户端执行了。

5.2.1 小结

在这一章中，我们以提供计算加、减、乘、除的网络应用为例，详细的介绍了如何建立起一个完整的 web service 服务的步骤和注意事项，虽然这个例子和实际使用的应用环境有较大的差异，但基本方法应该是一致的。

第三章 数据交换

我们的这个例子说明了 DataSet-----一个基于 XML 技术的强大的数据分离技术，能够用 Web Service 方法返回。DataSet 能够在一个智能化的结构中存储复杂的信息和关系，这是 Web Service 的一个非常有用的方法。

通过 DataSets 的显示，你能够限制通过连接你的数据库服务器的测试。

GetTitleAuthors 方法连接一个数据库并且运行两个 SQL 语句，第一个返回颜色的列表，另外一个返回字体大小的列表。方法把两个结果用一个 DataSet 来存储，并返回一个 DataSet。

PutTitleAuthors 说明一个 Web Service 方法把 DataSet 当作一个参数并返回一个整数，这个整数就是在 DataSet 中的“Table“表的行数。虽然这个方法执行起来有点简单，但是这个方法也能够与数据库服务器把过剩的数据聪明的合并在一起。

我们来看看这个例子，首先：

```
<%@ WebService Language="VB" Class="DataService" %>
```

这句话应该包括。我们还要引入这个名字空间：

```
Imports System.Web.Services
```

第一我们两个方法，Getcolor()：

```
Public Function <WebMethod()> Getcolor() As DataSet
 Dim MyConnection As SQLConnection = New
     SQLConnection("server=localhost;uid=sa;pwd=;database=howff")
```

```
        Dim MyCommand1 As SQLDataSetCommand = New
            SQLDataSetCommand("select * from color", MyConnection)
        Dim MyCommand2 As SQLDataSetCommand = New
            SQLDataSetCommand("select * from size", MyConnection)
        '数据的填充
        Dim DS As New DataSet
        MyCommand1.FillDataSet(DS, "color")
        MyCommand2.FillDataSet(DS, "size")
        Return DS
    End Function
```

Putcolor()方法：

```
    Public Function <WebMethod()> Putcolor(DS As DataSet) As Integer
        '返回行数
        Return DS.Tables(0).Rows.Count
    End Function
```

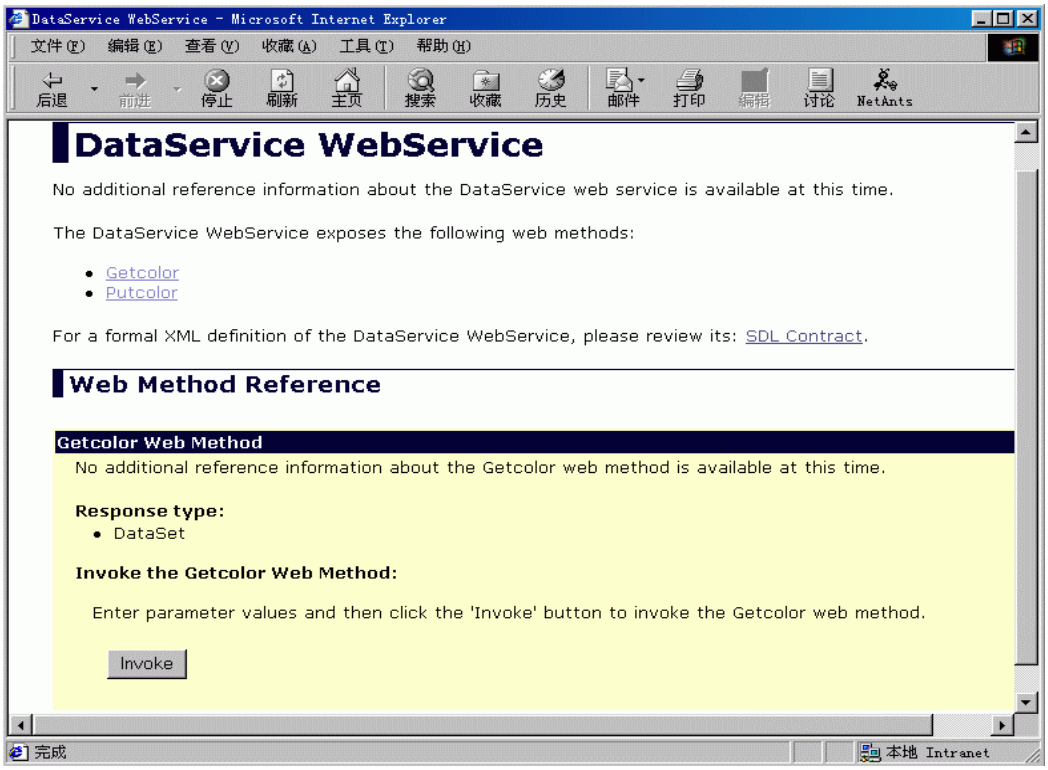
文件保存为 webservice.asmx，放在虚拟目录下，具体代码如下：

源文件：webservice\webservice.asmx

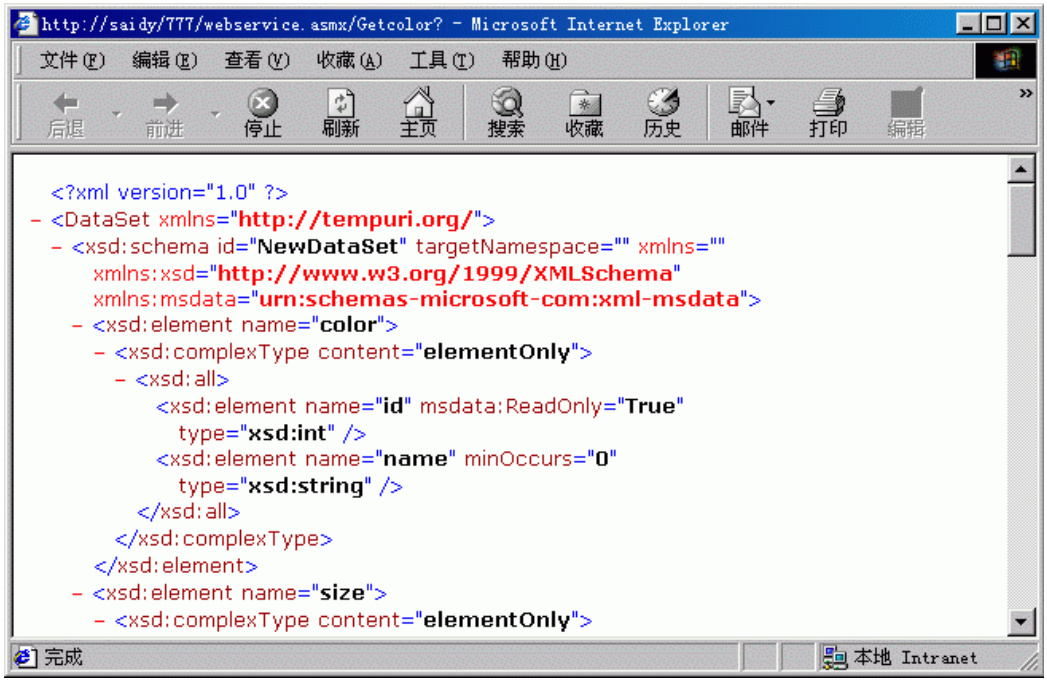
```
    <%@ WebService Language="VB" Class="DataService" %>

    Imports System
    Imports System.Data
    Imports System.Data.SQL

    '引入 System.Web.Services 名字空间
    Imports System.Web.Services

    Public Class DataService

        Public Function <WebMethod()> Getcolor() As DataSet
        '创建数据库连接和命令集
            Dim MyConnection As SQLConnection = New
                SQLConnection("server=localhost;uid=sa;pwd=;database=howff")
            Dim MyCommand1 As SQLDataSetCommand = New
                SQLDataSetCommand("select * from color", MyConnection)
            Dim MyCommand2 As SQLDataSetCommand = New
                SQLDataSetCommand("select * from size", MyConnection)
        '数据的填充
            Dim DS As New DataSet
            MyCommand1.FillDataSet(DS, "color")
            MyCommand2.FillDataSet(DS, "size")
            Return DS
```

```
    End Function
    '
    Public Function <WebMethod()> Putcolor(DS As DataSet) As Integer
        '返回行数
        Return DS.Tables(0).Rows.Count
    End Function

End Class
```

运行：



点击“Invoke“按钮，我们可以浏览到一个 xml 形式的文件。



如果我们点击了“putcolor“连接后再点击“Invoke“按钮，则得到不同的结果，有兴趣的读者不妨试一试，就象我们在上面的例子上所看到的一样。

5.3.1 小结

这一章我们学习了 web service 采用 xml 的方式在不同平台间传递数据的例子，xml 标准将是今后数据相互传输和转化的通用标准。

第四章 存取站点对象

我们的例子说明怎么样通过 Web Service 来访问 Web 站点的固有的东西如：Session 和 Application 属性，以及如何关闭 Session。

asmx 文件例子中的第一个方法，UHC 访问 Session，并在“HitCount“变量上加 1。我们定义 Session 计数器方法：

```
Public Function <WebMethod(EnableSession:=true)> UHC() As String
    If Session("HitCounter") Is Nothing
        Session("HitCounter") = 1
    Else
        Session("HitCounter") = Cint(Session("HitCounter")) + 1
    End If
  Return "你已经访问这个服务器 " & Session("HitCounter").ToString() & " 次了."
```

```
End Function
```

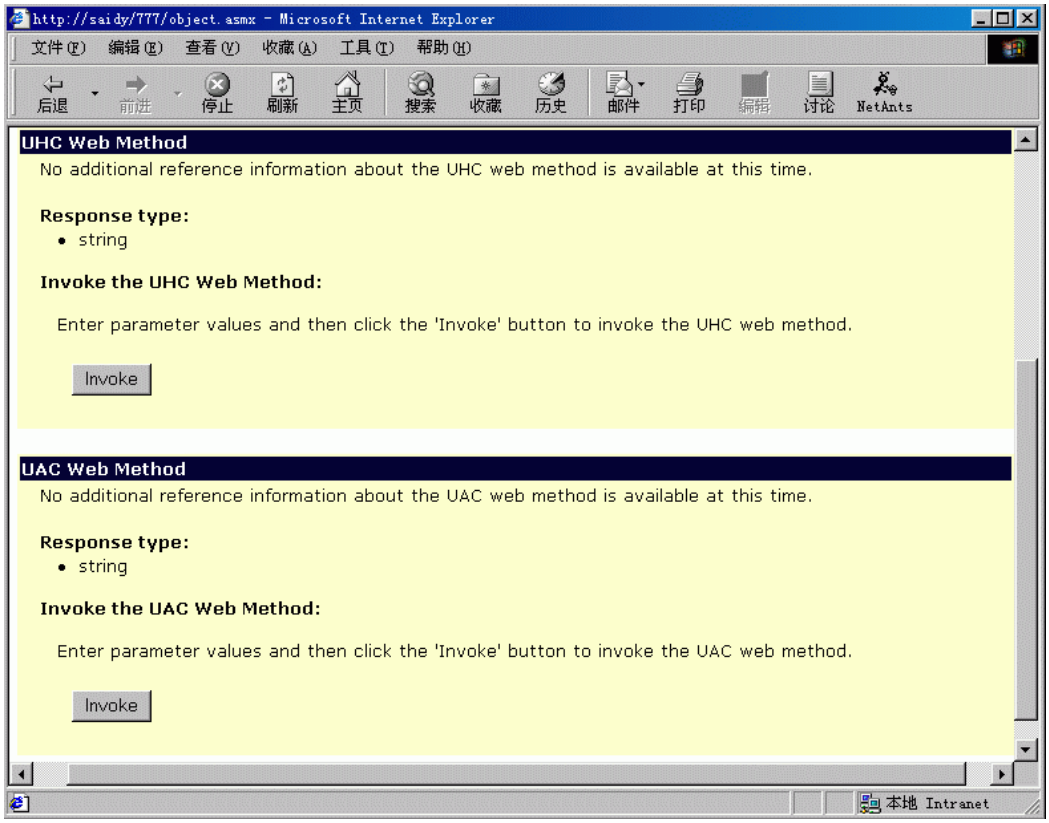
注意在方法名称的前面，我们加上了"<WebMethod(EnableSession:=true)>" 这个修饰语句。在更新计数器的方法中：

```
Public Function <WebMethod(EnableSession:=false)> UAC() As String
    If Application("HitCounter") = Nothing
        Application("HitCounter") = 1
    Else
        Application("HitCounter") = CInt(Application("HitCounter")) + 1
    End If
    Return "服务被访问了 " & Application("HitCounter").ToString() & " 次."
End Function
```

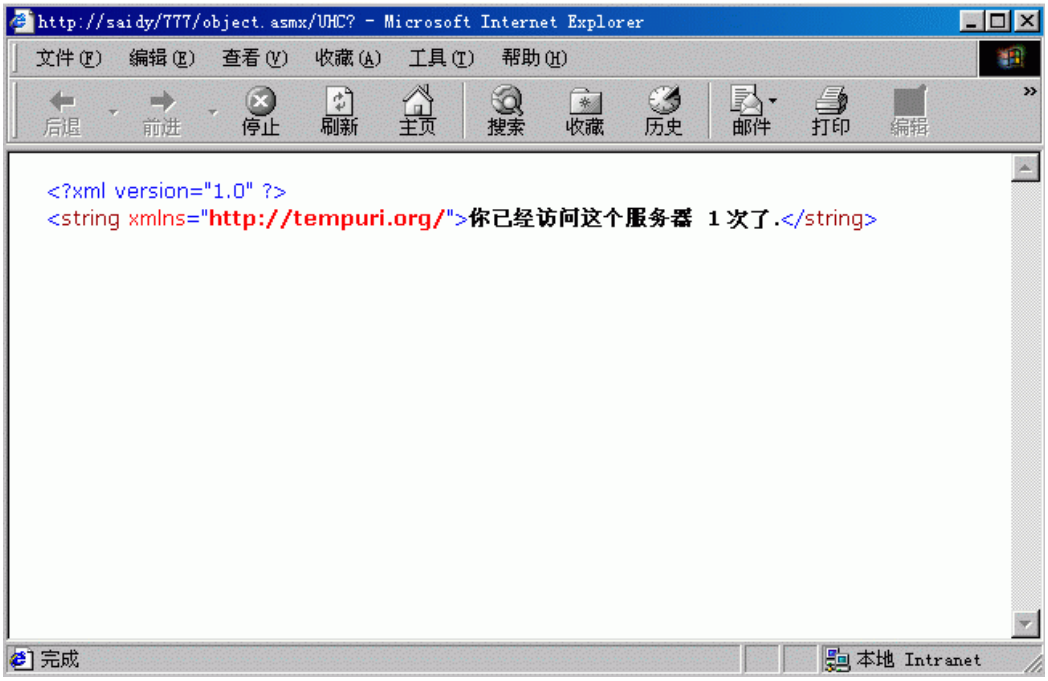
注意与上面方法的区别，我们在修饰方法名称的语句
"<WebMethod(EnableSession:=false)> "不一样。下面是我们完整的例子代码：

```
<%@ WebService Language="VB" Class="SessionService1" %>
Imports System
Imports System.Web.Services

Public Class SessionService1 : Inherits WebService
  '增加计数器的值方法
  Public Function <WebMethod(EnableSession:=true)> UHC() As String

    If Session("HitCounter") Is Nothing
        Session("HitCounter") = 1
    Else
        Session("HitCounter") = Cint(Session("HitCounter")) + 1
    End If

    Return "你已经访问这个服务器 " & Session("HitCounter").ToString() & " 次了."
  End Function

  '更新计数器值的方法
  Public Function <WebMethod(EnableSession:=false)> UAC() As String

    If Application("HitCounter") = Nothing
        Application("HitCounter") = 1
    Else
        Application("HitCounter") = CInt(Application("HitCounter")) + 1
    End If
    Return "服务被访问了 " & Application("HitCounter").ToString() & " 次."
  End Function
End Class
```

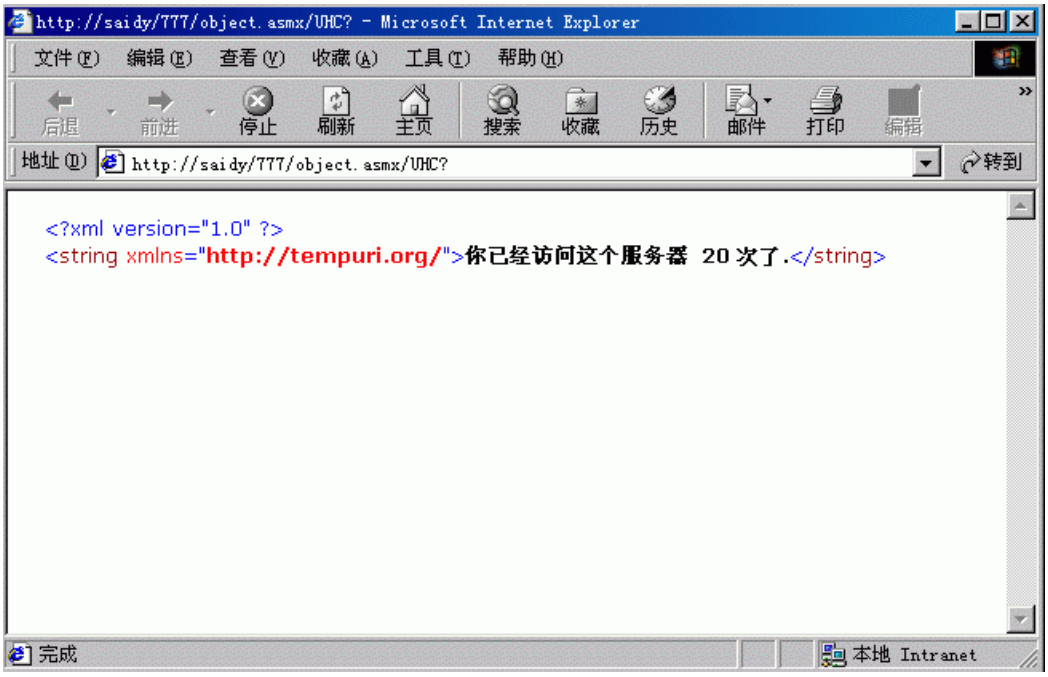
下面是我们的运行效果：



点击上面的“Invoke“按钮，有如下结果：



点击下面的“Invoke“按钮，有如下结果（在此之前我们已经点击了很多回）：



5.4.1 小结

本章通过对传统记数器在 web service 中的实现为例，介绍了 web service Application 如何使用 session 对象、application 对象的方法。

第六篇 性能优化

在计算机科学领域，广泛应用缓冲技术来提高系统的性能，它的原理是把经常存取的或者是比较重要的数据保存于内存中以减少系统的响应时间。对于 WEB 应用领域，缓冲技术主要是把 HTTP 请求的页面或数据保存于内存，以减少下次使用时重建它们的耗费。

ASP.NET 有两种用于 WEB 应用的缓冲技术：输出缓冲和数据缓冲。

输出缓冲指：把一次请求所产生的动态输出保存于内存中。

数据缓冲指：按照一定的策略把事先不确定的对象保存于内存中。

输出缓冲常用于把整个输出页面缓冲起来。对于一个存取繁忙的站点来说，把一些常用页面放入内存会带来性能上的极大提高。当一个页面被放入输出缓存，那么接下来的对该页面的请求将不再执行创建它的代码，而是从内存中直接返回该页面。

但实际上，保存整个输出页面的方法并不一定都行得通，因为有些页面的输出取决于客户端的不同请求，称之为“定制”。这时，采取的方法即找出不同中的相同，把一些并不需要经常重新创建的对象和数据识别出来，进行缓冲。一旦这些部分被识别，那么它们将被一次创建并在缓存中保持一定的时间。

选择缓存的时间是提高性能的关键。对一些部分来说，它们需要隔一定时间进行刷新，而另一些部分来说，可能仅仅只是需要保存一段时间。此种情况下，都可以设定“过期策略”来实现。一旦这些对象和数据到期，它们都将被从缓存中清除出去。当存取对象和数据的代码发现所要求的部分在内存中不存在时，将重建该对象或数据。

ASP.NET 支持文件和缓存关键字的依赖关系，它允许开发人员创建缓存依赖于一个外部文件或另一个缓存事物。利用这项技术可以更新一个缓存事物当其依赖的源文件发生改变时。

第一章 页面输出缓存

6.1.1 基本概念

页面输出缓存通过保存动态页面的输出内容，大大提高了服务器应用的能力。缺省情况下，输出缓存选项是被打开的，但并不是任意给定的输出响应都将被缓存，除非显示地指定页面应被缓存。

为使输出能够被缓存，输出响应至少应有一个有效的过期/有效策略以及公用 cache 的访问权限。当一个 GET 请求被送往页面，一个输出缓冲入口将被创建。接下来，对该页面的 GET 请求和 HEAD 的请求将直接从该缓冲入口中取出返回给用户，而对该页面的 POST 请求通常是显示地产生动态内容，却并非如同 GET 和 HEAD 请求一样从缓冲入口中取出。

输出缓存还支持带请求串的 GET 方法，把请求串作为页面识别的一部分。这就意味着带有相同键值但排列次序不同的请求串的 GET 请求，可能导致缓存中认为不存在该输出页面。

输出缓存需要知道页面缓存的过期/有效时间策略。如果一个页面在输出缓存中，而且又被指定为 60 分钟的页面过期时间，那么从它进入输出缓存开始，60 分钟后该页面将从输

出缓存中被清除。如果恰在此时，有一个对该页面的请求到达，页面的代码将被执行，页面输出又将重新进入输出缓冲。这种方式的过期策略称之为“强制过期”，页面只在一定时间内有效。

如下，我们可以用下面一条语句来显示的指出页面在输出缓冲中的保存时间。

<%@ OutputCache Duration=秒数 %>

6.1.2 实例

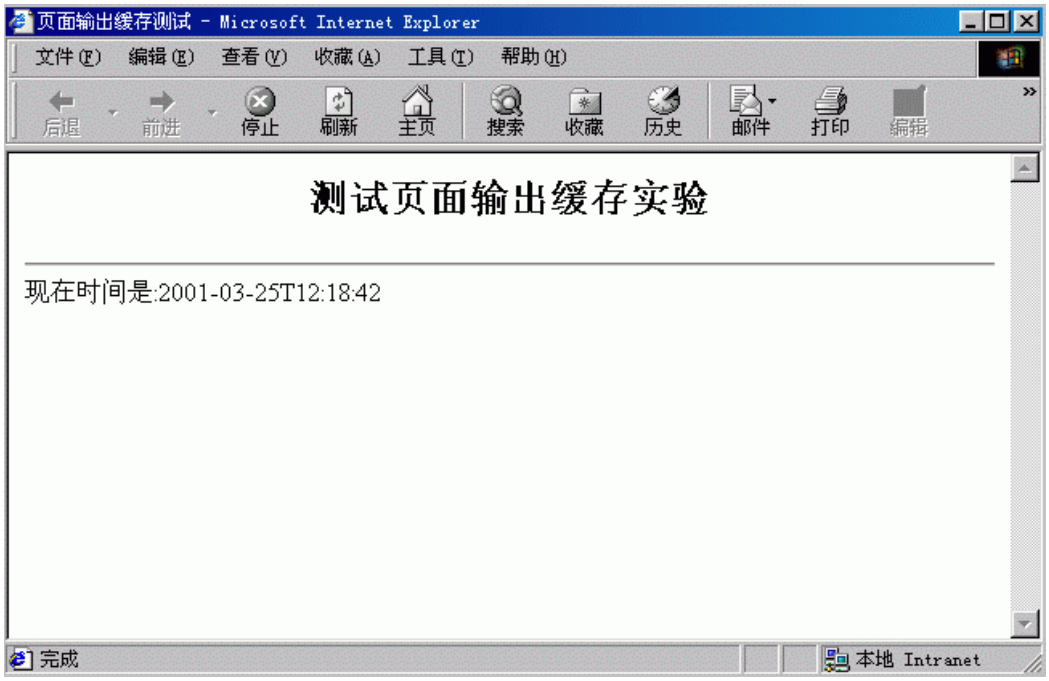
下面举一个简单的例子来证实 ASP.NET 中的页面缓存功能

在一个页加载时，我们显示它的时间，在页面过期时间（设为：10 秒）到达之前，我们把页面刷新（相当于重发 GET 请求），看一看显示的时间；然后，在过期时间到达之后，再看显示的时间。如果，第一次和第二次显示的时间相同，那么就证明了，系统存在有页面输出缓存功能，做为对比，当过期时间到达后，新的请求将导致重新执行页面代码，产生新的时间显示。

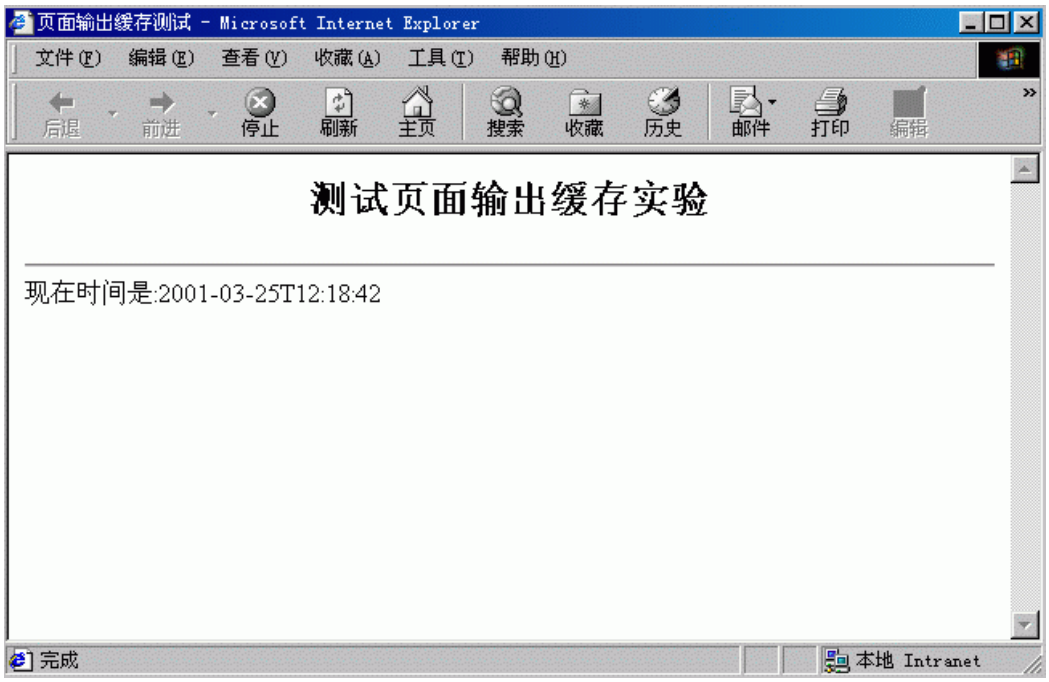
程序源程序

```
<!--文件名: performance\FormPageCache.aspx-->
<%@ OutputCache Duration="10" %>
<!--过期时间设为 10 秒-->
<html>
<head>
<title>
页面输出缓存测试
</title>
</head>
  <script language="VB" runat="server">
    Sub Page_Load(s As Object, E As EventArgs)
          lblTime.Text ="现在时间是:" &    DateTime.Now.ToString()
    End Sub
  </script>
  <body >
    <center>
    <h2><font face="Verdana">测试页面输出缓存实验</font></h2>
    <p><p><p>
    <hr>
    </center>
   <asp:label id="lblTime" runat="server"/>
  </body>
</html>
```

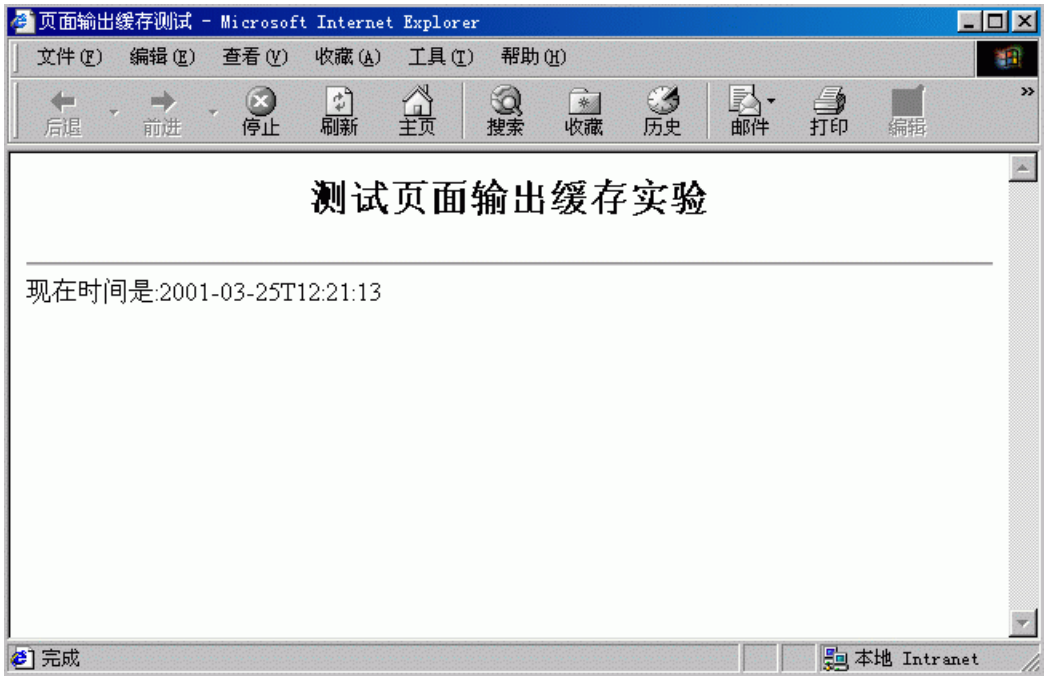
第一次输出效果：



第二次输出效果：



第三次输出效果：



实现页面缓存的另一种方法:

<%@ Import NameSpace=”System.Web.HttpCachePolicy” %>

‘指定页面输出缓存下一个 10 秒到期

Response.Cache.SetExpires(DateTime.Now.AddSeconds(10))

‘指定所有用户都有对缓存的访问权力

Response.Cache.SetCacheability(HttpCacheability.Public)

如果不希望进行页面缓存,可采用 Response.Cache.SetSlidingExpiration 方法,当其为 True 时，每次页面请求到达时，相当于页面过期时间到了，就要对页面输出重新刷新 。

‘当每次页面请求时,重置到期时间计数器，并且页面到期

Response.Cache.SetSlidingExpiration(True)

例如上面的例子可以改写成如下例子：

```
<!--文件名:performance\FormPageCache01.aspx-->
<%@ Import NameSpace="System.Web.HttpCachePolicy" %>
<html>
<head>
<title>
页面输出缓存测试 1
</title>
</head>
  <script language="VB" runat="server">
```

```
    Sub Page_Load(s As Object, E As EventArgs)
      response.cache.setexpires(DateTime.Now.addseconds(10))
      response.cache.setcacheability(Httpcacheability.Public)
      lblTime.Text ="现在时间是:" &  DateTime.Now.ToString()
    End Sub
  </script>
  <body >
    <center>
    <h2><font face="Verdana">测试页面输出缓存实验 1</font></h2>
    <p><p><p>
    <hr>
    </center>
   <asp:label id="lblTime" runat="server"/>
   </body>
</html>
```

输出的结果和上一例子大体相同，就不在重复了。

另外，在 ASP 中，对于上面的例子我们还可以写为：

```
Response.CacheControl = "Public"
Response.Expires = 10
```

从兼容性出发，ASP.NET 中依然具有相同的效果，但建议尽量使用 ASP.NET 的形式。

6.1.3 小结

本章介绍了页面输出缓存的基本概念，以及在 asp.net 中如何通过 page 命令和 response 对象在编程中实现页面输出缓存的实现方法。

第二章 页面数据缓存

6.2.1 基本概念

ASP.NET 提供了一个相当出色的缓存引擎机制，它允许页面保存和索引 HTTP 请求所要求的各种各样的对象。ASP.NET 的缓存对各个应用来说是私有的，是存储各种对象的存储器。缓存的生存周期取决于应用的生存周期，也就是说，当应用重新启动时，缓存实际上也已重建。

缓存提供了一个简单的字典接口，以便于开发者能够轻而易举放置对象到缓存中，并在以后使用。最简单的情况下，放置一个对象到缓存中，就如同对字典增加一个条目。

例如：

```
Cache(“myKey”)=MyValue
```

即是把 MyValue 放入缓存中一个叫 myKey 的缓存对象中，当需要引用 myKey 值时，可以采用下面的使用方式：

```
myValue1=Cache(“myKey”)
```

```
if myValue1 <> Null then
‘非空时的动作
…
end if
```

asp.net 提供了三种缓存替换的策略：

1.“腐烂搜索”（Scavenging）

比较类似于“最近最少使用”替换原则，当内存变得比教紧张时，缓存机制会找出最不常用和最不重要的对象，把它从内存中移出，以减轻系统压力。为控制“腐烂搜索”的具体行为，编程者必须在插入缓存对象时，指明插入它的耗费和多少时间内它必须被存取一次才能继续留在缓存中，以供替换时进行决择。

2.“到期控制”（Expiration）

编程者可以指定缓存对象的生存周期，这种指定的时间可以是绝对的也可以是相对的。例如绝对的时间（下午 6：00 到期），相对时间（该对象距最近一次存取它的时间满 10 分钟即到期）。当一个缓存对象到期后，它将从缓冲内存中移出，此时对该对象的索引将得到 null 值，除非又重新插入该对象。

3.“文件和键值依赖”

从外部文件或者是其他缓存键值是否改变，来决定本身键值是否有效。如果依赖发生改变，缓存对象将变得不可使用，并从缓存中移动出来。试想这样一种情况，应用从一个 XML 文件中读出金融信息，而该文件又被定期地修改。应用的作用是利用从该文件读出的信息构造一个图形对象以表示销售的情况。当应用读入文件时，它把数据插入缓存中，并记录下文件的依赖关系。当文件发生修改时，应用使开始产生的缓存对象无效并从内存中移出已经无用的数据，此后应用重新读入文件的数据，再把更新后的数据放入缓存，这样就完成了信息的更新，并返回给最终用户。

6.2.2 实例

例子：

从一个 XML 中读出用户信息并显示在页面上，页面提供了两个按钮，一个为增加用户，一个为刷新。我们在内存中建立了一个缓存对象“DataCache5”,它与客户信息文件”custom1.xml”建立了依赖关系。当 custom1.xml 文件未发生变化时，我们按下“刷新”按钮，可以看到信息是从缓存中读出的；当我们输入客户相应的资料，增加了一个客户以后，再按下“刷新“按钮后，我们可以看到客户信息变成了从文件读出。

1．ASPX 源程序（performance\FormDataCache.aspx）：

```
<!-- 文件名：FormDataCache.aspx -->
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<html>
<script language="VB" runat="server">
sub LoadData1
'当缓存对象 DataCache5 有效时，从缓存中读出客户信息；无效时，从文件读出信息
dim dv1 as DataView
    dv1=Cache("DataCache5")
    if dv1 = Nothing
```

```
        dim ds as DataSet
        dim fs as FileStream
        dim sr as StreamReader

        ds=New DataSet
        fs=New FileStream(Server.MapPath("custom1.xml"),FileMode.Open,FileAccess.Read)
        sr=New StreamReader(fs)
        ds.ReadXml(sr)
        fs.Close()
        dv1=new DataView(ds.Tables(0))
        Cache.Insert("DataCache5",dv1,New cachedependency(Server.MapPath("custom1.xml")))
        lblMsg.text="数据从文件中读出..."
        Else
        lblmsg.text="数据从缓存中读出..."
        end if
'绑定到 DataGrid1 对象
        DataGrid1.datasource = dv1
        DataGrid1.databind()
end sub
sub Page_Load(s as object,e as eventargs)
'加载页面时，从文件中读出客户信息
        if Not IsPostBack
        LoadData1()
        end if
end sub
sub AddBtn_Click(s as object,e as eventargs)
'增加一个客户信息到文件中
dim FS as FileStream
dim Reader as StreamReader
dim DS as DataSet
dim dr1 as DataRow
dim tw1 as TextWriter
        if Not Page.IsValid
        lblMsg.text="还有域未曾填充..."
        else
        DS=New DataSet()
        FS=New
FileStream(Server.mappath("custom1.xml"),FileMode.Open,FileAccess.Read,FileShare.ReadWrit
e)
        Reader=New StreamReader(FS)
        DS.readxml(Reader)
                FS.Close()
        dr1=DS.tables(0).newrow()
                dr1("CustName")=txtName.text
```

```
        dr1("CustIdno")=txtIdno.text
        dr1("CustCard")=txtCard.text
        DS.tables(0).rows.add(dr1)

            FS=New
FileStream(Server.MapPath("custom1.xml"),FileMode.Create,FileAccess.ReadWrite,FileShare.Re
adWrite)
        tw1=New StreamWriter(FS)
        tw1=textwriter.synchronized(tw1)
        DS.writexml(tw1)
        tw1.close()
        LoadData1()
        end if
end sub
sub RefreshBtn_Click(s as object,e as eventargs)
        LoadData1()
end sub
</script>
<head>
<title>
数据缓冲实验
</title>
</head>
<body>
<center>
<form runat=server>
<h2>XML 文件缓存测试</h2>
<ASP:DataGrid id="DataGrid1" runat="server"
            Width="600"
            BorderColor="black"
            ShowFooter="false"
            CellPadding=3
            CellSpacing="0"
            Font-Name="Verdana"
            Font-Size="8pt"
                  />
<hr>
<h3>添加一个客户信息</h3>
<table>
<tr>
<td>姓       名:</td>
<td><ASP:textbox id=txtName runat=server /></td>
<td><ASP:RequiredFieldValidator          ControlToValidate="txtName"          Display="static"
ErrorMessage="*" runat=server /> </td>
```

```
</tr>
<tr>
<td>身份证号:</td>
<td><ASP:textbox id=txtIdno runat=server /></td>
<td><ASP:RequiredFieldValidator ControlToValidate="txtIdno"
Display="static" ErrorMessage="*" runat=server /> </td>
</tr>
<tr>
<td>信用卡号:</td>
<td><ASP:textbox id=txtCard runat=server /></td>
<td><ASP:RequiredFieldValidator ControlToValidate="txtCard"
Display="static" ErrorMessage="*" runat=server /> </td>
</tr>
</table>
<p>
<asp:button text="增加" onclick="AddBtn_Click" runat=server />
<asp:button text="刷新" onclick="RefreshBtn_Click" runat=server />
<p><p><p>
<asp:label id=lblMsg runat=server />
</form>
</center>
</body>
</html>
```

2．初始运行效果

初始时，应用空间中无“DataCache”缓存对象，故文件从 custom1.xml 中读出客户信息，开始时为空，只显示了字段名。

3．当添加一个用户信息后，文件 custom1.xml 发生改变，导致“DataCache5”对象无效，

LoadData1 过程依然从文件 custom1.xml 中读取信息，并更新 DataCache5 对象。

4．由于 custom1.xml 文件未曾发生改变，所以当按下“刷新“按钮后，信息却是从缓存对象 DataCach5 中读出来。

6.2.3 小结

本章介绍了除页面输出缓存技术以外的另一种 asp.net 缓存技术，页面数据缓存。在我们的实践当中，页面数据缓存技术可能比页面输出缓存技术使用得更普遍一些。

第七篇 高级应用

第一章 XML 及其应用

XML 是标准扩展语言的简称，是未来 Web 编程的标准。在这一章中，我们将讲述 XML 在 ASP.NET 中的应用

7.1.1 制作广告条

在这个程序中，我们通过 XML 语言实现每次访问网页时，将显示不同的广告条。在本例中，我们只调用了两条广告

源文件：**advanceapp\intro.aspx**
Intro.aspx 的代码如下：

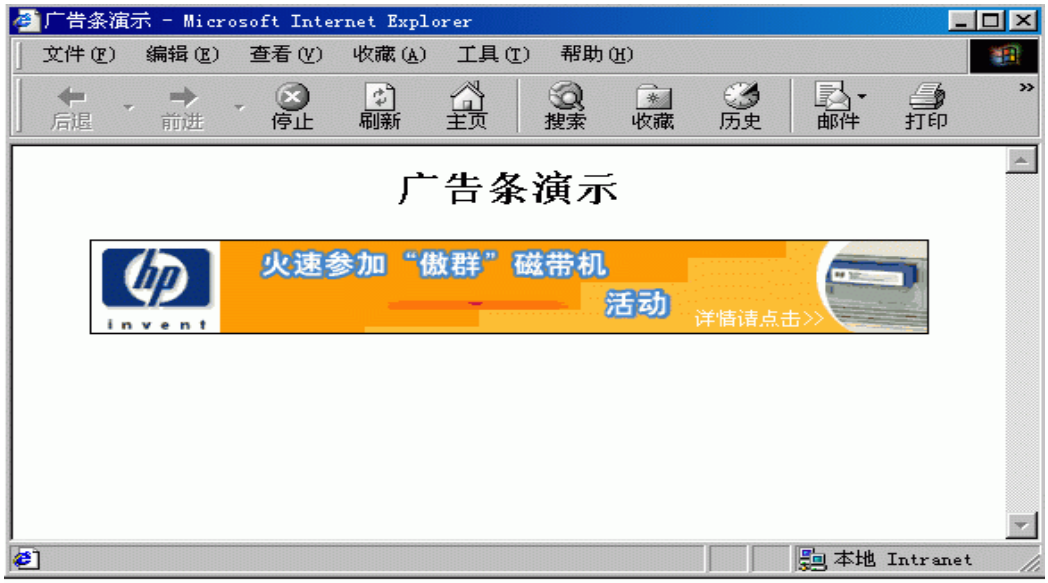
```
<html>
<center><title>广告条演示</title></center>
    <head>
        <link rel="stylesheet"href="intro.css">
    </head>
  <body>
        <center>
          <form action="intro.aspx" method="post" runat="server">
<h2>广告条演示</h2>
       <asp:adrotator AdvertisementFile="intro.xml" BorderColor="black" BorderWidth=1
runat="server"/>
      </form>
          </center>
    </body>
</html>
```

intro.xml 的 代码如下：

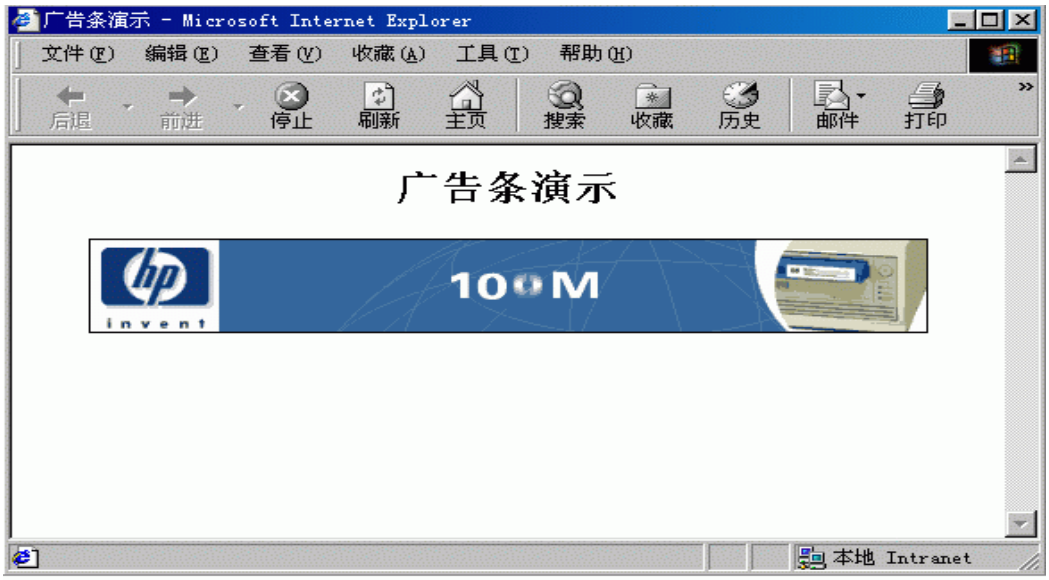
```
<Advertisements>
    <Ad>
        <ImageUrl>./hp1.gif</ImageUrl>
        <NavigateUrl>http://www.yesky.com</NavigateUrl>
        <AlternateText>欢迎访问！</AlternateText>
        <Keyword>Computers</Keyword>
        <Impressions>80</Impressions>
    </Ad>

    <Ad>
        <ImageUrl>./hp2.gif</ImageUrl>
```

```
    <NavigateUrl>http://www.yesky.com</NavigateUrl>
    <AlternateText>欢迎访问</AlternateText>
    <Keyword>Computers</Keyword>
    <Impressions>80</Impressions>
</Ad>
</Advertisements>
```

在 intro.aspx 中，我们使用了<asp:adrotator AdvertisementFile="intro.xml" BorderColor="black" BorderWidth=1 runat="server"/>这条语句来嵌入 intro.xml 文件。运行效果如图：



当我们点刷新按狃或者按 F5 键，我们将看到另一条广告条。



在本例中，我们用到了 AdRotator 服务器控件，在 xml 文件中，我们我们可以定义其属性：如表所示：

属性	描述

ImageUrl	图象文件的绝对或相对地址
NavigateUrl	当图象被点击时，可访问相应的网页
AlternateText	当鼠标移动到图片上时，将显示提示信息
Keyword	指定广告条的分类，我们可以利用此属性来对广告条进行分类
Impressions	指定图片在表格中的大小

7.1.2 XML 和 dataset 控件结合使用

数据访问是一个应用系统的核心。公用语言运行环境（Common Language Runtime）提供了管理数据访问应用程序接口（API）的方法。而这些 API 将不论它的的数据源是什么，如 SQL Server, OLEDB, XML 等，都能提取出我们所需要的数据。我们在具体编程的时候，有 3 个对象将常常用到：Connections, Commands, and DataSets。

对象	描述
Connection	与数据源进行连接。如 SQL Server 或者一个 XML 文件。
Command	对数据进行操纵。如进行删除（delete），提取（select）,更新（update）
Dataset	显示出所需的数据

为了使我们能使用 SQL 语句，我们要先导入 System.Data 和 System.Data.SQL 这两个名字空间。

语句如下：

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SQL" %>
```

下面的表格是对几个典型的 SQL 语句的说明：

SQL 语句	示例
Select（对单个表的操作）	SELECT * from Student WHERE stuname = ‘小李’;
Select（对多个表的操作）	SELECT * from Student S, Dept D WHERE S.dept= D.dept;
Insert	INSERT into Student VALUES ('小王'，21，’男’);
Delete	DELETE from Student WHERE name=’小王’;

Update	UPDATE Student SET age = 21　WHERE name=’小李’;

在执行查询之前，我们要先构件一个 SQLDataSetCommand 对象，在执行查询以后，我们需要把数据转移到 DataSet 中，我们可以使用下面的代码：

我们假设有一个 test 数据库，在这个数据库中有一个 student 表。

```
Dim myConnection As New SQLConnection("server=localhost;uid=sa;pwd=;database=test")
Dim myCommand As New SQLDataSetCommand("select * from student", myConnection)
Dim ds As New DataSet()
myCommand.FillDataSet(ds, "student")
```

可能读者不禁要问：为什么要用 XML 文件存储数据吗?为什么不使用数据库？

这是因为：对很多目的用途来说，用数据库太过浪费了。.要使用一个数据库，你必须安装和支持一个分离的服务器处理进程（a separate server process）,它常要求有安装和支持它的管理员（administrator）。你必须学习 SQL 语句, 并用 SQL 语句写查询，然后转换数据，再返回。而如果你用 XML 文件存储数据，将可减少额外的服务器的负荷。还有，你还找到了一个编辑数据的简单方法。你只要使用文本编辑器,而不必使用复杂的数据库工具。XML 文件很容易备份，和朋友共享，或下载到你的客户端。同样的，你可以方便地通过 ftp 上载新的数据到你的站点。

XML 还有一个更抽象的优点，即作为层次型的格式比关系型的更好。 它可以用一种很直接的方式来设计数据结构来符合你的需要。你不需要使用一个实体-关系编辑器，也不需要使你的图表（schema）标准化。 如果你有一个元素（element）包含了另一个元素，你可以直接在格式中表示它，而不需要使用表的关联。

注意，在很多应用中，依靠文件系统是不够充分的。如果更新很多，文件系统会因为同时写入而受到破坏。数据库则通常支持事务处理，可以应付所发生的请求而不至于损坏。对于复杂的查询统计要有反复、及时的更新，此时数据库表现都很优秀。当然，关系型数据库还有很多优点，包括丰富的查询语言，图表化工具，可伸缩性，存取控制等等。

在下面这样的案例中，正如大多数中小规模的、基于发布信息的站点一样，你可能涉及的大多数数据存取都是读，而不是写，数据虽然可能很大，但相对来说并没有经常的更新变化，你也不需要做很复杂的查询，即使你需要做，也将用一个独立的查询工具，那么成熟的 rdbms 的优点消失了，而面向对象型的数据模型的优点则可以得到体现。

最后，为你的数据库提供一个查询器外壳来进行 SQL 查询并将他们转化进入 xml stream 也是完全有可能的。

所以你可以选择这二种方式之一。XML 正变成一种非常健壮的，便于编程的工具，作为某个成熟的数据库的前端工具来进行存储和查询。(oracle 的 xsql servlet 即是这种技术的一个很好的例子。)

7.1.3 XML 语法

现在我们来介绍一下 XML Schemas：

随着 XML Schema 规范的逐渐普及，它成为 W3C 推荐标准的那一天已经不远了。我们来看一下这个新的定义文档类型的方法。

一、定义元素

文档类型定义的工作现在大都是由DTDs来完成的，现在的DTD在功能上有一些限制，随着 XML 应用的越来越广泛，这些限制逐渐阻碍了 XML 的发展。

DTD 的语法不同于 XML 的语法，因而需要文档编写者另外的学习一套符号语言。而软件也需要另外的一个解析器来对 DTD 文件进行解析。

在 DTD 中没有办法来定义能够从程序语言变量直接映射到 XML 数据的数据类型和数据格式。 没有一组被人所熟知的基本的元素以供文档编写者选择。DTDs 是 XML 从其前辈 SGML 那儿继承过来的（事实上，XML 本身就是一种 SGML 的简化版本），这个相对比较成熟的技术能够很快的让 XML 运行起来，并且能够让熟悉 SGML 的人对 XML 有更多的感觉。然而，很快的人们就发现，XML 需要一种更具表达能力的解决方案，而不仅仅是 DTD。

定义元素需要指定元素的名称、元素的内容模式、元素的属性、以及内嵌的子元素。在 XML Schemas 中，元素的内容模式是通过类型来定义的。一个服从于某个特定的 schema 的 XML 文档只能按照 schema 中定义的元素模式来编写，这同 DTD 的规则是一样的。元素可以是简单类型的，也可以使复杂类型的。在 Schemas 规范中定义了很多的简单类型，例如： string，integer 和 decimal。简单类型的元素不能在包含子元素和属性，而复杂类型的元素则能够嵌套子元素，并能够包含有属性。

我们来看看一个简单的定义元素的例子：

<element name="quantity" type="positive-integer"/>

<element name="amount" type="decimal"/>

我们知道在面向对象的观点中，有聚集和继承的概念，可以在已有的类中衍生出新类。在这儿 Schema 借用了这些观点，用户也可以通过聚集和继承来在老元素的基础上定义新的元素。聚集能够把一组已存在的元素组合成一个新的元素。继承则通过扩展已存在的元素来定义一个新的元素，并且这个新的元素能够代表被继承的那个元素。

比如，如果我们要从 decimal 类型中派生一个新的 value 元素，并让它有这样的内容模式：<value unit="Celsius">42</value>，我们可以这样的定义：

<element name="value">
<complexType base=&single;decimal&single; derivedBy=&single;extension&single;>
<attribute name=&single;unit&single; type=&single;string&single;/>
</complexType>
</element>

下面我们把 time 和 value 元素聚集到一个 measurement 元素中，形成这样的内容模式：

<measurement>
<time>2000-10-08 12:00:00 GMT<time/>
<value unit="Celsius">42</value>

那么最终我们的 schema 元素定义结果就应该是这样的：

<element name=&single;measurement&single; type=&single;measurement&single;/>
<complexType name=&single;measurement&single;>
<element name=&single;time&single; type=&single;time&single;/>
<element name=&single;value&single; type=&single;value&single;/>
</complexType>

和上面的 schema 等同 DTD 是：

```
<!ELEMENT measurement (time, value)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value (unit)>
```

显然，它所能表达的意思就少了很多。

从 Java 等面向对象语言中引入的继承的特性，还包括了可以定义抽象元素来迫使实现子类元素，或者定义一个终结元素来禁止元素再被其它元素所继承。这使得从元素到 Java 或者 C++语言的类的一一映射变得更为直观可行。

二、基数表达

XML Schema 比起 DTD 来能够更方便的表达元素基数的概念。所谓基数是指一个元素在文档中出现的次数。在 DTD 中，使用的是正则表达式来表示基数的，而正则表达式的表达能力在所有的形式化文法中是最低的，这使得你在定义文档的时候会遇到很多的困难。你只能够指定一次(1)，零次或者更多 (*)，一次或者更多 (+) 这三种基数，并用这些基数序列来构成 DTD。XML Schema 中则是使用 minOccurs 和 maxOccurs 这两个属性来定义元素基数，分别用来指定元素出现的最少次数和最多次数：

```
<element ref="optionalElement" minOccurs="0"/>
<element ref="twoOrMoreElements" minOccurs="2" maxOccurs="unbounded"/>
<element ref="exactlyOneElement" />
```

minOccurs 和 maxOccurs 的缺省值都是 1，也就是说，当没有指定这两个属性的时候，元素只允许在文档中出现一次。除了这两个属性，你还可以用 choice 和 all 这两个元素。元素 choice 只允许它的所有子元素中的一个出现在文档中。而元素 all 则最为宽松，能够让其所有的子元素在文档中以任意的顺序出现任意的次数。而这些概念在 DTD 中都是难以表达的。

```
<xsd:choice>
<element ref="EitherThis"/>
<element ref="OrThat"/>
</xsd:choice>
<xsd:all>
<element ref="Ying"/>
<element ref="Yang"/>
</xsd:all/>
```

三、名域

XML Schema 中还支持名域。一个 Schema 除了可以定义 XML 文档词汇表外，还可以通过名域来定义目标名域，和其它可能会使用到的词汇名域。例如：

<xsd:schema targetNamespace=&single;http://www.physics.com/measurements&single; xmlns:xsd=&single;http://www.w3.org/1999/XMLSchema&single; xmlns:units= &single;http://www.physics.com/units&single;>
<xsd:element name=&single;units&single; type=&single;units:Units&single;/>
<xsd:element name=&single;measurement&single; type=&single;measurement&single;/>
<complexType name=&single;measurement&single;>
<element name=&single;time&single; type=&single;time&single;/>
<element name=&single;value&single; type=&single;value&single;/>
</complexType>

这种机制为建立复杂的名域体系提供了一种模块化而又易于扩展的方法。使得名域的作用能够真正的被体现出来。

XML Schema 提供了一个丰富而更具弹性的机制来定义 XML 文档词汇表。它使用 XML 语言本身来定义关于一个 XML 文档的元信息（meta-information），这使得 XML 的协同工作能力大大的增强了。而许多对于 DTD 功能上的改进和增强，也使得它最终必定会终结 DTD，作为 XML 的一个标准出现。在 Apache Project 和 IBM alphaworks 的主页上现在已经有很多的 Schema 的工具出现了，你如果有兴趣的话，不妨去看看。

好了，有上面的介绍，我们来看如何从 xml 文件中读取数据。DataSet 控件提供了 ReadXml 方法。同时在 XML 文件中，必须存在 schema 和我们所需要的数据（Data）。
好了，有了上面的介绍，我们先看一下 data1.aspx 文件：
源文件：**advanceapp\data1.aspx**

```
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<html>
<head>
  <title>
   XML 演示
  </title>
</head>
<script language="VB" runat="server">
      Sub Page_Load(Src As Object, E As EventArgs)
            Dim DS As New DataSet
            Dim FS As FileStream
            Dim Reader As StreamReader
            FS = New FileStream(Server.MapPath("data1.xml"),FileMode.Open,FileAccess.Read)
            Reader = New StreamReader(FS)
            DS.ReadXml(Reader)
            FS.Close()
            Dim Source As DataView
            Source = new DataView(ds.Tables(0))
            MySpan.InnerHtml = Source.Table.TableName
            MyDataGrid.DataSource = Source
            MyDataGrid.DataBind()
```

```
    End Sub
</script>
<body>
<center>
  <h3><font face="Verdana">XML 演示 <span runat="server" id="MySpan"/></font></h3>
  <ASP:DataGrid id="MyDataGrid" runat="server"/>
</center>
</body>
</html>
```

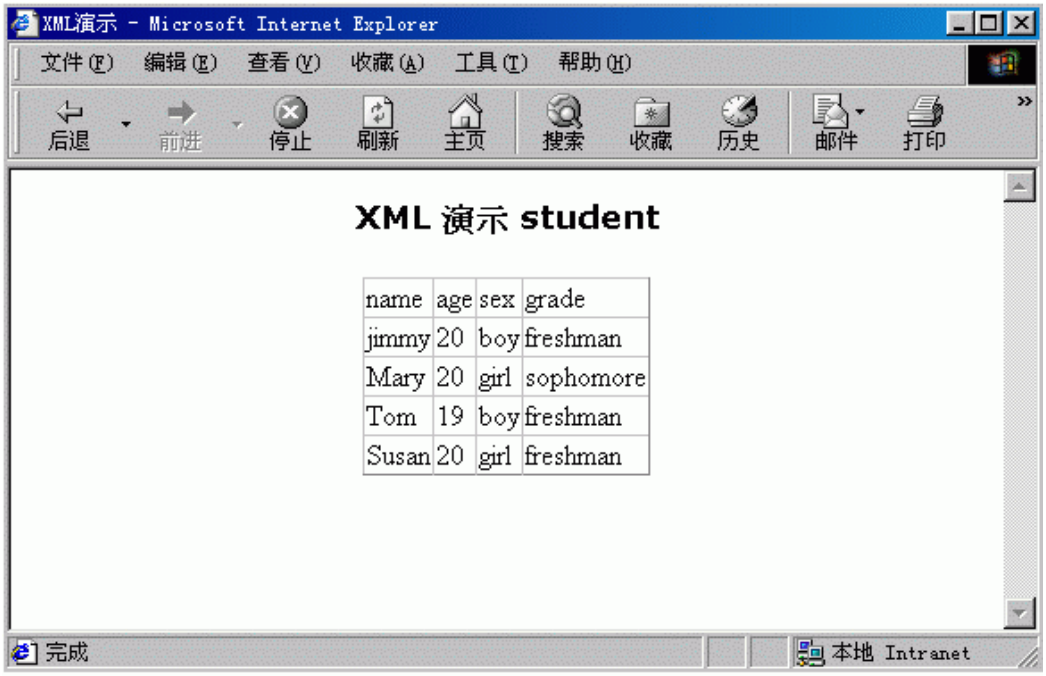
我们再来看看 xml 文件（**advanceapp\data1.aspx**）的内容：

data1.xml 的文件如下：

```
<center>
<root>
<schema id="DocumentElement" targetNamespace=""
xmlns="http://www.w3.org/1999/XMLSchema"
xmlns:xdo="urn:schemas-microsoft-com:xml-xdo" xdo:DataSetName="DocumentElement">
    <element name="student">
        <complexType content="elementOnly">
            <all>
                <element name="name" type="string"></element>
                <element name="age"    type="int"></element>
                <element name="sex"    type="string"></element>
                <element name="grade" type="string"></element>
             </all>
        </complexType>
    </element>
</schema>
<DocumentElement>
    <student>
        <name>jimmy</name>
        <age>20</age>
        <sex>boy</sex>
        <grade>freshman</grade>
    </student>
    <student>
        <name>Mary</name>
        <age>20</age>
        <sex>girl</sex>
        <grade>sophomore</grade>
            </student>
    <student>
        <name>Tom</name>
        <age>19</age>
```

```
            <sex>boy</sex>
            <grade>freshman</grade>
        </student>
        <student>
            <name>Susan</name>
            <age>20</age>
            <sex>girl</sex>
            <grade>freshman</grade>
        </student>
</DocumentElement>
</root>
</center>
```

程序运行后演示如下：



当然我们也可以把 schema 和数据分开成对立的文件。在主文件中，我们要分别使用 the ReadXmlData 和 ReadXmlSchema 方法。

如：

读取 schema.xml 的内容：

```
FS = New FileStream(Server.MapPath("schema.xml"),FileMode.Open,FileAccess.Read)
Schema = new StreamReader(FS)
DS.ReadXmlSchema(Schema)
FS.Close()
```

读取 data.xml 的内容：

```
FS = New FileStream(Server.MapPath("data.xml"),FileMode.Open,FileAccess.Read)
Reader = New StreamReader(FS)
```

```
DS.ReadXmlData(Reader)
FS.Close()
```

我们上面提到的 data1.xml 分成两部分：

第一部分：生成 schema.Xml 文件：

```
<schema id="DocumentElement" targetNamespace="" xmlns="http://www.w3.org/1999/XMLSchema"
xmlns:xdo="urn:schemas-microsoft-com:xml-xdo" xdo:DataSetName="DocumentElement">
    <element name="student">
        <complexType content="elementOnly">
            <all>
                <element name="name" type="string"></element>
                <element name="age"  type="int"></element>
                <element name="sex"  type="string"></element>
                <element name="grade" type="string"></element>

            </all>
        </complexType>
    </element>
</schema>
```

第二部分生成 data.xml：

```
<DocumentElement>
    <student>
        <name>jimmy</name>
        <age>20</age>
        <sex>boy</sex>
        <grade>freshman</grade>
    </student>
    <student>
        <name>Mary</name>
        <age>20</age>
        <sex>girl</sex>
        <grade>sophomore</grade>
            </student>
    <student>
        <name>Tom</name>
        <age>19</age>
        <sex>boy</sex>
        <grade>freshman</grade>
    </student>
    <student>
        <name>Susan</name>
        <age>20</age>
        <sex>girl</sex>
        <grade>freshman</grade>
```

```
    </student>
</DocumentElement>
```

7.1.4 小结

通过上面的介绍，使我们对 XML 在 ASP.NET 中有了一定的认识。在下面的章节中，我们将对 XML 进行更深入的讲解。

第二章 三层结构及其应用

7.2.1 概念及环境

ASP.NET 中的三层结果开发方法，其实其思想跟 Java 的一样。Java 中的三层架构为前端的 html、Jsp、Servlet，中间层为 JavaBean、EJB，后面为数据库服务器。而在 ASP.NET 中，前段为 html、asp、aspx 等，中间层为有.vb、.cs 等文件编译而成的.dll 控件，后面为数据库服务器。

在我们的三层架构中，我们的数据库层通过中间层来连接以及操作，前端给中间层传递参数，并接受中间层的参数。在我们的 ASP.NET 中，我们主要关注的是我们的中间层与前端的数据交互。

我们一般统称中间层为组件，组件可以用.vb 编译而成，也可以用.cs 文件编译而成。中间层一般为.dll 文件。微软的.NET 技术在这个方面比他的以前的任何版本都要来的简单，这也是它的一打好处之一。以前我们要注册一个.dll 文件，有是注册有是重启动，而在.net 上，我们的.dll 文件拿来就用，不用再考虑注册的问题。

在没有 Visual stutio.net 之前，我们用写成的.bat 文件来把.vb 和.cs 文件编译成.dll 文件，在.bat 文件里，我们写入编译的文件名称、相关联的名字空间、要编译成的文件名以及对应的命令名称，然后运行就行了。听起来很复杂，这也是很多初学者在编译第一个.dll 文件时所害怕的事情。但是做起来很简单的。下面我们举一个例子来说明.bat 文件的写法，假设我们有一个文件名为：saidy.vb 的文件，我们要把它编译成 saidy.dll 的文件，其中用到 System、System.Data、System.Data.SQL 名字空间，我们可以创建一个 dblink.bat 文件，内容如下：

```
vbc  /out:..\bin\saidy.dll  /t:library  /r:system.dll /r:system.data.dll  / r:system.data.sql.dll
     dblink.vb
```

这是编译.vb 程序的命令，如果是编译.cs 文件，则命令会是不一样，我们假定有一个 saidy.cs 的文件，按照上面的要求，我们编译如下：

```
cs  /out:..\bin\saidy.dll  /t:library  /r:system.dll /r:system.data.dll  / r:system.data.sql.dll
    dblink.cs
```

我们可以看出来，大部分是一样的。

当然，如果我们有微软公司的 vs.net 编程环境，则我们不用这么麻烦，我们可以象编译 vb 或者 vc 程序一样方便的编译.dll 文件。微软公司的 vs.net 是一个集大成者，把各种语言整合起来，在这个环境下都可以写出不同语言的程序。具体的应用我们会在专门的章节上介

绍的。

7.2.2 一个基于三层架构的例子

我们通过具体的例子来说明三层架构的应用，我们建一个小项目来说明这个问题。有时为了安全性，我们通常把与数据库的连接用一个动态连接库文件封装起来，这样我们就要把写数据库连接的.vb 或者.cs 文件编译成动态连接库.dll 文件。甚至我们把对数据库的相关操作页编译成.dll 文件。

下面是我们的与数据库连接以及操作的文件 dblink.vb 的主要部分，对数据库的连接：

Dim dbl As SQLConnection

对数据库的操作，我们把它写在一个方法里面，在返回相应值：

```
Function getdata() as DataView
    Dim sComm as SQLDataSetCommand
    Dim sDS as DataSet
    Dim sStr as String
    dbl = New SQLConnection("server=localhost;uid=sa;password=;database=howff")
    sStr = "select * from color"
    sComm = new SQLDataSetCommand(sStr,dbl)
    sDS = new DataSet()
    sComm.FillDataSet(sDS,"color")
    Return sDS.Table["color"].DefaultView
End Function
```

我们第六个语句就用到上面的与数据库的连接变量，我们这个函数的功能是从表“color“中选出所有的元素，并返回表结构的形式。完整的代码如下：

```
Imports System
Imports System.Data
Imports System.Data.SQL
'创建名字空间
Namespace db
'创建一个类
Public Class dblink
'建立数据库的连接
Dim dbl As SQLConnection
'方法
 Public Function getdata() As DataView
    Dim sComm As SQLDataSetCommand
    Dim sDS As DataSet
    dbl = New SQLConnection("server=localhost;uid=sa;password=;database=howff")
    Dim sStr As String
    sStr = "select * from color"
```

```
            sComm = New SQLDataSetCommand(sStr, dbl)
            '填充数据
            sDS = New DataSet()
            sComm.FillDataSet(sDS, "color")
            '返回
            Return sDS.Tables("color").DefaultView
        End Function
    End Class
End Namespace
```

我们再写一个前端掉用页面 saidy.aspx，我们首先要引入我们创建的名字空间：

```
<%@ Import Namespace="db" %>
```

在页面装入的时候，我们用此方法：

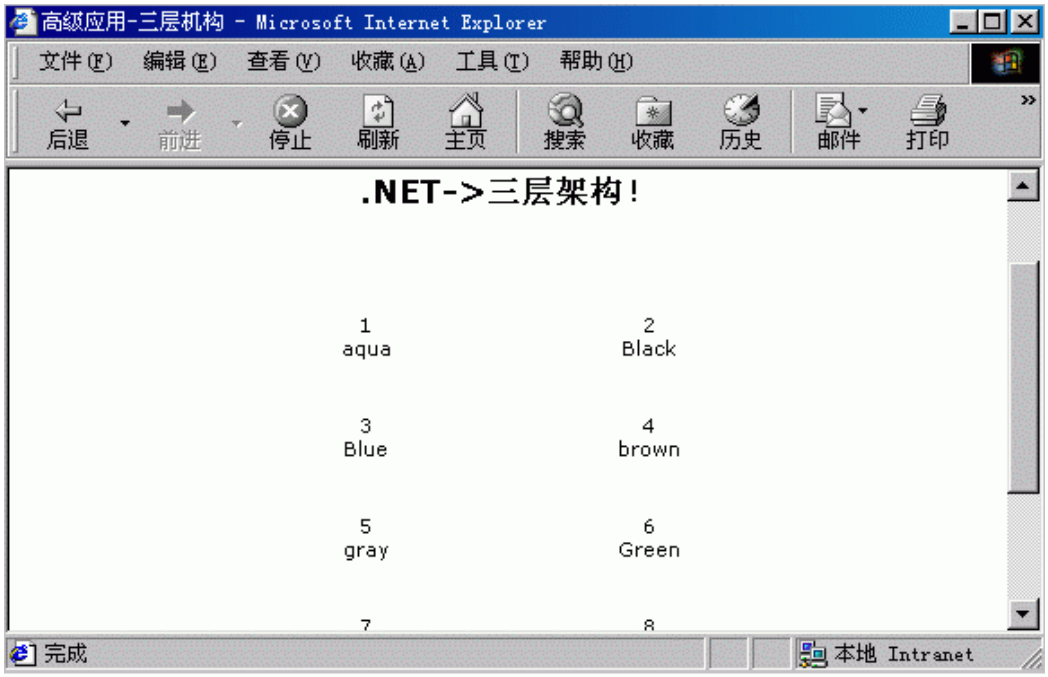
```
Sub Page_Load(Sender As Object, E As EventArgs)
    '建立一个新的对象
    Dim newdb As dblink
    newdb = new dblink()
    '数据来源
    Products.DataSource = newdb.getdata()
    '数据绑定
    Products.DataBind()
End Sub
```

下面看看我们完整的代码**(advanceapp\dblink.aspx)**：

```
<%@ Import Namespace="db" %>
<html>
<script language="VB" runat="server">
  Sub Page_Load(Sender As Object, E As EventArgs)
    '建立一个新的对象
    Dim newdb As dblink
    newdb = new dblink()
    '数据来源
    Products.DataSource = newdb.getdata()
    '数据绑定
    Products.DataBind()
  End Sub
</script>
<body style="font: 10pt verdana" bgcolor="CCCCFF">
<BR><BR><BR>
<CENTER>
  <h3>.NET->三层架构！</h3>
</CENTER>
<BR><BR>
<CENTER>
```

```
  <ASP:DataList id="Products" ShowHeader=false ShowFooter=false RepeatColumns="2"
RepeatDirection="horizontal" BorderWidth=0 runat="server">
      <template name="itemtemplate">
        <table>
          <tr>
            <td width="150" style="text-align:center; font-size:8pt; vertical-align:top;
                height:50">
              <p>
              <%# DataBinder.Eval(Container.DataItem, "id") %> <br>
              <%# DataBinder.Eval(Container.DataItem, "name", "{0:C}").ToString() %>
            </td>
          </tr>
        </table>
      </template>
  </ASP:DataList>
</CENTER>
</body>
</html>
```

我们看到，在这个页面当中，没有出现与数据库交互的语句，这样我们就很好的把数据操作封装起来了，我们的运行效果如下：



7.2.4 小结

在本章中，我们讲解一个基于三层架构的例子，这只是一个非常简单的例子，我们知道.NET 在这方面的功能是非常强大的，你可以用它来写非常复杂的组件。

第三章 使用 MSMQ

7.3.1 基本概念

MSMQ(MicroSoft Message Queue，微软消息队列)是在多个不同的应用之间实现相互通信的一种异步传输模式，相互通信的应用可以分布于同一台机器上，也可以分布于相连的网络空间中的任一位置。它的实现原理是：消息的发送者把自己想要发送的信息放入一个容器中（我们称之为 Message），然后把它保存至一个系统公用空间的消息队列(Message Queue)中；本地或者是异地的消息接收程序再从该队列中取出发给它的消息进行处理。

在消息传递机制中，有两个比较重要的概念。一个是消息，一个是队列。消息是由通信的双方所需要传递的信息，它可以是各式各样的媒体，如文本、声音、图象等等。消息最终的理解方式，为消息传递的双方事先商定，这样做的好处是，一是相当于对数据进行了简单的加密，二则采用自己定义的格式可以节省通信的传递量。消息可以含有发送和接收者的标识，这样只有指定的用户才能看到只传递给他的信息和返回是否操作成功的回执。消息也可以含有时间戳，以便于接收方对某些与时间相关的应用进行处理。消息还可以含有到期时间，它表明如果在指定时间内消息还未到达则作废，这主要应用与时间性关联较为紧密的应用。

消息队列是发送和接收消息的公用存储空间，它可以存在于内存中或者是物理文件中。消息可以以两种方式发送，即快递方式(express)和可恢复模式（recoverable）,它们的区别在于，快递方式为了消息的快速传递，把消息放置于内存中，而不放于物理磁盘上，以获取较高的处理能力；可恢复模式在传送过程的每一步骤中，都把消息写入物理磁盘中，以得到较好的故障恢复能力。消息队列可以放置在发送方、接收方所在的机器上，也可以单独放置在另外一台机器上。正是由于消息队列在放置方式上的灵活性，形成了消息传送机制的可靠性。当保存消息队列的机器发生故障而重新启动以后，以可恢复模式发送的消息可以恢复到故障发生之前的状态，而以快递方式发送的消息则丢失了。另一方面，采用消息传递机制，发送方必要再担心接收方是否启动、是否发生故障等等非必要因素，只要消息成功发送出去，就可以认为处理完成，而实际上对方可能甚至未曾开机，或者实际完成交易时可能已经是第二天了。

采用 MSMQ 带来的好处是：由于是异步通信，无论是发送方还是接收方都不用等待对方返回成功消息，就可以执行余下的代码，因而大大地提高了事物处理的能力;当信息传送过程中，信息发送机制具有一定功能的故障恢复能力；MSMQ 的消息传递机制使得消息通信的双方具有不同的物理平台成为可能。

在微软的.net 平台上利用其提供的 MSMQ 功能，可以轻松创建或者删除消息队列、发送或者接收消息、甚至于对消息队列进行管理。

在.NET 产品中，提供了一个 MSMQ 类库“System.Messaging.dll”。它提供了两个类分别对消息对象和消息队列对象进行操作。在能够使用 MSMQ 功能之前，你必须确定你的机器上安装了 MSMQ 消息队列组件，并确保服务正在运行中。在使用 ASP.NET 编程时，应在头部使用：

```
<%@ Assembly Name=”System.Messaging”%>
<%@ Import NameSpace=”System.Messsaging”%>
```

将 MSMQ 类库引入 ASP.NET 文件

1. 对消息队列的创建

```
dim MsgQue as MessageQueue
MsgQue=New MessageQueue(MsgPath)
```

其中：MsgPath 可以为本地私有队列，如“.\MyQueue”，也可以为其他机器的公有队列，如“Saidy\777$\MyQueue”,Saidy 为另一机器名。

2. 消息的发送

```
dim MsgQue as MessageQueue
MsgQue.Send(Msg)
```

其中：Msg 为任一对象。

3. 消息的接收

消息的接收又分成同步和异步方式两种，同步接收在规定时间内从消息队列中取出收到的第一条消息，当消息队列中没有消息时，程序处于等待状态；异步接收方式则是定义了一个事件处理函数，当消息队列中第一个消息到达时立即触发该函数。

1）同步方式

```
dim Msg as Message
dim Fmt As XmlMessageFormatter
 Fmt= CType(MsgQue.Formatter,XmlMessageFormatter)
Fmt.TargetTypeNames = new String(){"System.String"}
Msg=MsgQue.receive(New TimeSpan(0,0,3))
```

首先定义收到消息应转换成的格式，然后在指定时间内去接收消息

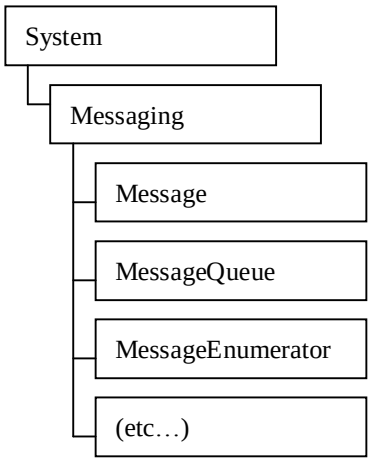
2）异步方式

```
dim Fmt As XmlMessageFormatter
 ‘定义接收消息类型
Fmt = CType(MsgQue.Formatter,XmlMessageFormatter)
Fmt.TargetTypeNames = new String(){"System.String"}

 ‘定义消息处理函数入口
AddHandler MsgQue.ReceiveCompleted, New ReceiveCompletedEventHandler
                    (AddressOf OnReceiveCompleted)

 ‘定义消息处理函数
Public Shared Sub OnReceiveCompleted(s As Object, asyncResult As
ReceiveAsyncEventArgs)
    Dim MsgQue As MessageQueue = CType(s,MessageQueue)
    Dim Msg As Message = MsgQue.EndReceive(asyncResult.AsyncResult)
‘此时 Msg.Body 即为所取消息对象
    MsgQue.BeginReceive()
```

```
        ‘重新定义异步接收方式
    End sub

        ‘启动异步接收方式
        MsgQue.BeginReceive
```

7.3.2 消息队列的命名空间体系结构

如图所示：



消息队列常用操作：

。用 create 方法创建你指定路径的消息队列，使用 delete 方法删除一个已经存在的消息队列。

。使用 exists 方法判别是否存在一个消息队列。

。使用 GetPublicQueues 方法获取消息队列网络中的一个消息队列

。使用 Peek 或者是 BeginPeek 方法查看消息队列中的消息，而不会删除它们

。使用 Receive 或者上 BeginReceive 方法从消息队列中取出一个消息，同时在消息队列中删除它。

。使用 Send 方法，送一个消息到指定的消息队列中。

7.3.3 消息队列的操作

1． 创建消息队列

。创建公共消息队列

MessageQueue.Create(“MyMachine\MyQueue”)

。创建私有消息队列

MessageQueue.Create(“MyMachine\Private$\MyPrivateQueue”)

说明：标识 Private$表示创建的是私有消息队列

2． 队列引用说明

当你创建了一个 MessageQueue 部件的一个实例以后，就应指明和哪个队列进行通信。在。Net 中有 3 种访问指定消息队列的方法：

。使用路径，消息队列的路径被机器名和队列名唯一确定，因而可以用消息队列路径来指明使用的消息队列。

。使用格式名（format name），它是由 MSMQ 在消息队列创建时或者应用程序在队列创建以后生成的唯一标识。

。使用标识名（label），它是消息队列创建时由队列管理者指定的带由描述意义的名字。它可能并不唯一。

采用路径（path）方式引用队列

消息队列类型	路径使用格式
Public queue	MachineName\QueueName
Private queue	MachineName\Private$\QueueName
Journal queue	MachineName\QueueName\Journal$
Machine journal queue	MachineName\Journal$
Machine dead letter queue	MachineName\Deadletter$
Machine transactional dead letter queue	MachineName\XactDeadletter$

。因为消息队列服务器接收到一个使用路径方式使用消息队列的操作请求时，会去解析出路径和格式名（format name）,因此它的效率上不如格式名方式使用队列。

。消息队列未连接时，只能使用格式名方式对它发送消息。

路径名的引用除了 path 属性以外，还可以由 MachineName 和 QueueName 两个属性得到。

路径引用的例子：

MessageQueue1.path=".\MyQueue"

采用格式名（format name）方式引用队列

格式名由公有私有标识串加上队列产生的 GUID，以及其他必需的标识构成。

消息队列类型	格式名的构成规则
Public queue	PUBLIC=QueueGUID
Private queue	PRIVATE=MachineGUID\QueueNumber

Journal queue	PUBLIC=QueueGUID;JOURNAL 或者 PRIVATE=MachineGUID\QueueNumber;JOURNAL
Foreign queues	DIRECT=AddressSpecification\QueueName

格式名由不由用户指定，而是在队列创建时由队列管理者自动产生。

。当你的部件作为一个 WEB sevice 或者是 WEB 调用的一部分的时候，最好采用格式名方式引用队列，因为它速度较快。

。当向一个非连接的队列发送消息时，应使用格式名方式，因为当队列不可连接时，路径解析会导致失败。

。网络拓扑结构发生变化或者消息队列重建以后，格式名会变化。

可以由消息队列对象的 FormatName 属性得到格式名。

例如：采用格式名方式引用消息队列的例子

MessageQueue1.Path = "FORMATNAME:PUBLIC=3d3dc813-c555-4fd3-8ce0-79d5b45e0d75"

采用标识方式引用消息队列

标识是消息队列创建时，由消息队列创建者指定由于描述队列的文本属性。采用标识的好处在于屏蔽了低层的具体位置，对于移植和程序修改时，应用的修改很小。它的缺点也是显然的，就是不能保证标识的唯一性，当标识有冲突时，向该标识发送消息会导致一个错误的发生。标识可以由访问消息队列对象的 Label 属性得到。

总结，我们对消息队列引用的步骤大致为：

首先，产生一个 MessageQueue 对象的实例。

然后，根据引用消息队列的方式设置不同的属性。

如果为路径方式，设置它的 path 属性

如果为格式名方式，设置其 FormatName 属性

如果为标识方式，设置它的 Label 属性

3. 删除消息队列

删除一个队列使用 Delete 方法。当删除一个队列时，队列中含有的所有消息将首先被删除，然后删除该队列，它不会把消息队列中的信息发往死信队列中。删除一个队列最主要的问题是用户是否有删除该队列的足够的权限。

使用 Delete 方法的例子如下：

MessageQueue.Delete(“MyMachine\MyQueue”)

4. 清除消息队列中的内容

有时我们需要把送入消息队列中而尚未发出的消息清除，或者定期需对消息发送日志队列进行清除，可以使用消息队列对象提供的 Purge 方法。它把指定的消息队列中的消息全部清空，并不再发送。

使用的例子如下：

MessageQueue1.Path="MyMachine\MyQueue"

MessageQueue1.Purge()

5．创建消息队列对象的实例

创建一个消息队列实例的步骤如下：

1． 创建一个 MessageQueue 类的实例

例如：

dim MyQue as New MessageQueue

2． 对 MessageQueue 类实例的 path 属性进行设置

例如：

MyQue.path=".\MyQueue"

3．设置你需要其他 MessageQueue 类的属性

6．消息队列配置属性

关于队列的属性：

path 属性：它可以决定引用队列的三种方式，路径引用、格式名引用、标识引用

category 属性：标识当前使用的队列的类型。Category 是队列所有者定义的 GUID 值。该 GUID 值可以有 GUID 生成工具产生或者是用户自定义的数字值。GUID 值不会唯一，这样才可以根据相同的 GUID 值，把多个消息队列划分为不同的类别（category）。

跟发送数据类型相关的属性

Formatter 属性：决定在一个队列中如何发送和接收消息的顺序，以及可以在一个消息中发送什么样的内容。

和队列交互相关的属性

DenyShareReceive 属性：决定同一时间内只有一个部件能够访问消息队列中的消息。

CanRead 和 CanWrite 属性：决定队列是否可以被读取或者是写入。

MaximumQueueSize 和 MaximumJournalSize 属性：以千字节为单位设置一个队列（日志队列）的消息最大容纳量。一旦接收的消息到达这个容量，新的消息将不再被接收。
一般情况下，消息队列的最大值为消息队列管理员所设置，如果这个值没有控制的话，那么缺省的消息队列最大容量将是无限制的。

UseJournalQueue 属性：:设置是否将收到的消息拷贝到日志消息队列中去。

7．消息发送

MSMQ 消息队列中定义的消息由一个主体（body）和若干属性构成。消息的主体可以由文本、二进制构成，根据需要还可以被加密。你可以在属性窗口或者是直接对消息对象的属性进行赋值。但是，在 MSMQ 中消息的大小不能够超过 4 M B 。

消息可以被送往公用、私有、日志、死信、交易队列。

。简单消息的发送

简单消息发送的步骤大致为：

1．建立与要发送消息的队列的连接

2．指定你要发送的消息的格式

3．提供要发送的数据

4．使用 Send 方法将消息发送出去

采用简单消息发送方式发送的数据类型可以是：对象、原数据类型、数据流或其他简单的数据类型。

例子：以发送一个整型数和字符串为例

首先创建一个连接：

```
Dim MessageQueue1 as new MessageQueue ("MyMachine\MyQueue")
```

或者

```
Dim MessageQueue1 as New MessageQueue
MessageQueue1.path=”MyMachine\MyQueue”)
```

由于是标准数据类型，消息格式可以不指定，使用缺省的

然后发送数值 1

```
MessageQueue1.Send(1)
```

再发送字符串“hello world”

```
MessageQueue1.Send("Hello world")
```

。发送复杂的消息

相对于简单的消息发送，发送较为复杂的消息对象能够给你带来对消息更多的控制的控制能力。对于复杂对象的发送，我们是通过创建消息的对象的实例，然后对它的属性实行相应设置来实现的。

发送复杂消息的步骤大致为：

1. 创建一个 MessageQueue 的实例，然后对它的 path 属性设置，以指明操作的消息队列。
2. 创建一个消息对象实例。
3. 设置消息的主体（body），然后修改不希望使用缺省值的属性。
4. 同样使用 send 方法把消息对象发送至相应的队列中。

例子：

```
‘创建 MessageQueue 实例，指明连接的队列
dim MessageQueue1 as New MessageQueue(“.\MyQueue”)
  ‘创建消息对象实例
dim MyMessage as New Message(“Hello world”)
  ‘设置消息队列属性
MyMessage.Label=”MyLabel”
  ‘发送消息
MessageQueue1.Send(MyMessage)
```

8．消息接收

消息的接收分为两种，即同步和异步。

同步方式，我们使用 receive 方法。当调用 receive 方法时，它会从指定的消息队列中选取出第一个适合要求的消息对象返回给用户，然后把它从消息队列中删除。如果调用 receive 方法的时候，消息队列中没有一条记录存在，那么方法将导致程序挂起，直到有消息到达消息队列中。为了不使这个等待过程太长，你可以在调用 Receive 方法时指定 time-out 值（毫秒为单位），以指定在相应时间到达后退出等待过程。在使用 Receive 方式时，还可以指定

消息队列的 DenyShareReceive 属性，防止其他用户对队列进行操作。试想这样一种情况，当消息队列中剩一条消息的时候，2 个用户同时对它调用 peek 方法，发现都有消息，于是都放心的使用 Receive 方式去获取消息，结果必然导致其中一个用户被挂起。如果，在调用 receive 方法以前使用了 DenyShareReceive 属性拒绝其他用户的对该队列的 Receive 方法，就会避免上述现象的发生。

同步接收的例子：

```
dim MyMessageQue as MessageQueue

MyMessageQue=New MessageQueue("MyMachine\MyQueue")
'指定要连接的消息队列
dim MyMessage as New Message
MyMessage=MyMessageQue.Receive(1000)
'同步收取一条消息，超时时间为 1 秒
```

Peek 方法和 Receive 方法比较类似，只是它并不把取得的消息对象从消息队列中删除。Peek 方法只取出消息队列中的第一条消息，若要取得所有的消息可以使用 GetMessages 方法或者是 GetMessagesEnumrator 方法。Peek 方法也是同步方法，所以当消息队列中没有消息的时候，它也将被挂起，直到有消息产生。同样的，Peek 也可以指定超时时间（单位毫秒），一般最常用的方式是设为 0 或无穷大。设为 0 的意义是，对消息队列搜索看是否有消息产生，并马上返回；设为无穷大的意义是，直到有消息产生才进行处理。

例如：

```
'设置为超时时间为 0
MyMessage.TimeToBeReceived=0

 '设置超时时间为无穷大
MyMessage.TimeToBeReceived=Message.Infinite

 '以同步方式使用 Peek 方法的例子
dim MyMessageQue as New MessageQueue("MyMachine\MyQueue")

'指定连接的消息队列
'对消息队列查询首条消息，若无 1 秒钟后超时返回
dim MyMessage as Message
MyMessage=MyMessageQue.Peek(1000)
```

异步接收方式，是指接收消息时，不必理会调用的消息接收方法是否成功，方法将立即返回，并继续进行程序处理。对于异步接收方式收到的消息，有两种处理方式。一种是消息接收到以后，会发出一个事件，我们可以定义一个事件处理函数，在该函数中，对接收到的消息进行处理。另一种方法是为消息接收函数定义一个回调函数，当消息接收到以后，它会去处理带来的消息。

在 ASP.NET 中，事件处理方式中，在事件处理函数中，我们使用 BeginPeek 或者是 BeginReceive 方法从消息队列中取得一条消息。如果要处理多条消息，那么就要反复调用 BeginPeek 或者 BeginReceive 方法。

在回调处理方式中，回调函数定义了一个和 BeginPeek 或者 BeginReceive 方法绑定到一

起的代理，这样即使是在消息处理当中，代理仍然可以监视是否有新的消息到来。

因为消息处理函数方式用得比较普遍，所以下面我们就重点介绍消息函数方式得使用。

在事件模型中，首先必须绑定一个消息处理函数到一个消息，它是当异步调用完成以后，你希望对它进行处理的代码入口。然后调用 BeginReceive 方法，启动异步接收模式。当接收过程完成，.net 平台会发出一个消息，表示已经从消息队列中接收到一个消息，然后调用和该消息绑定的处理函数，以处理到达的消息。

它的步骤如下：

① 对 MessageQueue 对象的 BeginReceiveCompleted 事件绑定一个消息处理函数

② BeginReceiveCompleted 处理函数中，创建一个消息对象实例和一个消息队列 对象的实例

③ 定义如何处理收到的信息，代码框架如下：

```
Public Sub MyQueue_BeginReceiveCompleted(sender as Object, args as System.Messaging.ReceiveAsyncEventArgs)
    Dim myMessage as Message
    Dim myMessageQue as MessageQueue
    MyMessageQue = CType(sender,MessageQueue)
    myMessage = MyMessageQue.EndReceive(args.AsyncResult)
End Sub
```

④ 在主程序中，使用 BeginReceive 方法启动异步接收方式。

例如：

```
MyMessageQue.BeginReceive
```

使用 peek 方法的异步接收模式和使用 receive 方法的方式差不多，就不再多述了。

9．消息确认

为确保消息被正确发送到目的消息队列，我们还可以对消息队列进行设置，让其返回消息是否正确发送到指定队列的确认消息。确认消息有两种，一种是消息队列到达指定队列后发出的确认消息，另一种是消息被对方应用从消息队列中删除。而每一种消息又存在有两种形式，确认和否定。当消息正确到达目的队列或应用时，它会发回发送成功的确认消息。如果消息无法到达指定的队列或应用，那么它会发回发送失败的确认消息。不过收到发送失败的消息，未必是对方消息队列无法到达，它有可能是在超时时间设置过小或者是无法通过对方的验证。

从对方返回的确认消息通常不会放入一般的队列中，而是放入一个称之为管理队列的特殊队列中。确认消息也和一般的消息不一样，它不包含消息的主体，仅仅通过消息的头部就可以知道确认消息的意义。

在 ASP.NET 中，确认消息发往的管理队列，由其 AdministrationQueue 属性指定的队列所决定。

你可以把不同的确认消息发往不同的管理队列中。对管理队列的操作就和操作不同的队列是一样的，可以用 peek 方法查看，也可以用 remove 移走。

那么，如何设置一个消息，要求它返回确认信息呢？

首先设置一个消息对象的 AdministrationQueue 属性，指定确认消息返回的管理队列。

接着设置返回消息的确认类型，使用消息对象的 Acknowledge 属性。

Acknowledge 属性的值可以是：

。FullReachQueue 无论发送的消息到达或不能到达目的队列，都会返回确认消息

。FullReceive　　无论发送消息能否到达对方应用程序，都会返回确认消息。
。NotAcknowledgeReachQueue 只有发送的消息未能到达目的队列，才返回确认消息。
。NotAcknowledgeReceive 只有发送的消息未能到达对方应用程序，才返回确认消息。
。None　　不返回任何确认消息
最后，发送消息，并检查管理队列，看是否有确认消息产生。

10．消息日志

日志队列可以保存你操作过的消息的备份。它的好处是，一但发现前面的操作失败，可以从日志队列中重新创建出原先的消息对象，然后再进行操作。例如，向远方发送一个消息对象，然后对方返回一个失败的确认。我们可以从失败确认消息中提取出一个和开始发送的消息相关的 ID 值，然后根据提出的 ID 值从日志队列中找到发送的消息，重新创建一个消息对象，并再次发送。在.net 中，我们使用 ReceiveByCorrelationID 或 PeekByCorrelationID 方法根据 correlation ID 值取得消息对象。

在一台机器上，都会有一个全局消息队列，它保存任何从该机器发出的消息，而不论消息发送是否成功。每个消息队列也可以有自己的消息日志队列。日志队列的使用有两种方式，一种是对消息队列对象设置 UseJournalQueue 属性，它表示对该队列收到的所有消息使用日志记录方式，而对于发出的消息不做任何记录；另一种方法是对消息对象设置 UseJournalQueue 属性，所有被发送的消息将被记录到系统日志队列中去。消息日志队列有一个最大容量，称作 quota，一旦日志队列存储容量到达该值后，以后到来的本应存储的的消息将不再被存储，同时不会发出任何的出错信息。所以作为管理人员，应该定期清理日志队列，以防止上述现象的发生。消息队列只是被动的接收端，它们不可能返回确认消息，或者发送删除的消息到死信队列中，或者是进行超时处理。

例子：

设置消息队列对象的 UseJournalQueue 属性，以记录收到的消息到日志队列中
MyMessageQueue.UseJournalQueue=True

设置消息对象的 UseJournalQueue 属性，以记录收到的消息到系统日志队列中
MyMessage.UseJournalQueue=True

7.3.4 小结

通过本章的介绍，我们对 MSMQ 有了一定的了解，并且对消息队列的操作有了一定的了解。