任务七 实现用户管理功能

学习目标

在前面章节的学习中，我们已能够运用各种控件进行界面的设计，本章开始将主要讲解如何利用 ADO.NET 技术实现对数据库的操作。学习的重点侧重理解 ADO.NET 对象模型以及能够利用 ADO.NET 的模型实现对数据库的查询与修改等相关操作。

知识目标

- 掌握 ADO.NET 数据访问技术：Connection 对象，Command 对象，DataReader 对象，DataAdaper 对象，DataSet 对象
- 代码编写规范

技能目标

- 会熟练运用 ADO.NET 技术实现对数据库的访问
- 掌握数据库连接信息存取及配置文件 Web.config

项目背景

任何网站都必须进行维护，为此后台管理功能必不可少，而后台管理功能中最基本的功能则是用户管理功能。

任务实施

本次任务将划分为三个子任务，用户登录，修改用户以及查询用户功能，下面分别详细讲解。

7.1 子任务：用户登录

【任务陈述】

在后台登录页面，输入用户名和密码，点击“登录”按钮，首先判断用户名和密码是否正确，如果错误则显示相应的提示信息，如果正确则要区分用户类型是管理员还是普通用户，如果是管理员则跳转到后台管理首页，如果是普通用户则跳转至用户个人主页，同时将用户

名存入 Session，以便显示欢迎信息。

【知识准备】

7.1.1 ADO.NET 概述

ADO.NET 是一个包含在 Microsoft .NET Framework 框架中为编程人员提供数据访问服务的一种对象模型。

ADO.NET 包含.NET Framework 数据提供程序, 用于连接各种数据源、执行查询命令以及存储、操作和更新数据，如图 7-1 所示。为了简单起见，图 7-1 中只有一个数据源，在实际情况中可能有多个数据源。

图 7-1 ADO.NET

注意：按照数据库术语，数据源是数据、访问该数据所需的信息和该数据源所处位置的特定集合，其中的数据源位置可通过数据源名称描述。例如，数据源可以是通过网络在 Microsoft SQL Server 上运行的远程数据库，也可以是本地目录中的 Microsoft Access 文件以及相应的数据库连接字符串，包括用户、密码和连接方式等。

7.1.2 ADO.NET 对象模型

ADO.NET 同 Microsoft .NET Framework 的其他组件一样，它也由一整套.NET 对象组成，它分为两部分：数据提供程序（Data Provider）和数据集（DataSet），前者用于和真实数据进行沟通，后者用于表示真实数据，它们相互协作以提供所需的功能。图 7-2 是 ADO.NET 对象模型中各对象之间关系的示意图。

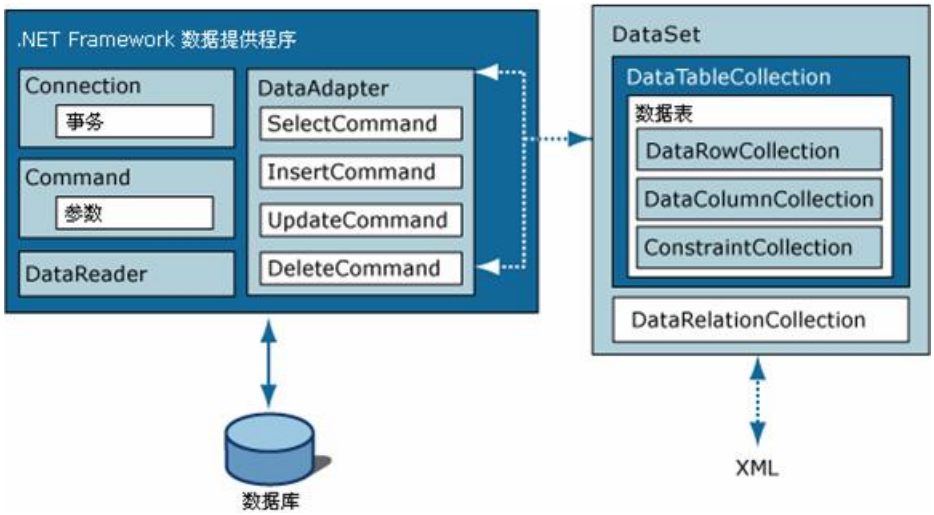

图 7-2　ADO.NET 对象模型之间的关系

ADO.NET 提供以下两种数据访问模型：

(1)连接的模型

在本模型下，能够使用数据库提供程序连接到数据库并对数据库运行 SQL 命令，命令运行过程中，数据库连接保持打开状态，命令运行结束后将关闭和数据库之间的连接。例如使用 Command 对象读取、写入和更新数据时的操作。

(2)断开连接的模型

在本模型下，能为来自数据源的数据创建内存中的缓存，创建好缓存后便可与数据源断开连接，然后对数据的查看、修改或删除等操作只需在该缓存中进行即可，当与数据源再次建立连接时便可将更改的内容合并至数据源。例如使用 DataSet 对象对数据源的操作。

7.1.3 数据提供程序

数据提供程序是一组用于访问特定数据库，执行 SQL 命令并获取值的 ADO.NET 类，简单地说，数据提供程序是连接应用程序与各种数据源之间的一座桥梁。

.NET Framework 有 4 个数据提供程序，如表 7-1 所示。

表 7-1 .NET Framework 数据提供程序及其描述

.NET Framework 数据提供程序	描述
SQL Server 数据提供程序	提供对 SQL Server 7.0 及更高版本和 Microsoft Data Engine（MSDE）的优化访问，该数据提供程序的类位于 System.Data.SqlClient 命名空间
OLE DB 数据提供程序	提供对有 OLE DB 驱动的任意数据源的访问，该数据提供程序的类位于 System.Data.OleDb 命名空间
Oracle 数据提供程序	提供对 Oracle 数据库（8i 及其后续版本）的优化访

	问，该数据提供程序的类位于 System.Data.OracleClient 命名空间
ODBC 数据提供程序	提供对有 ODBC 驱动的任意数据源的访问，该数据提供程序的类位于 System.Data.Odbc 命名空间

注意：针对其他数据库平台还有一些相应的数据提供程序。查找数据提供程序的最佳方法是访问数据库供应商的 Web 站点或开发人员论坛。

此外，不同的数据提供程序所实现的对象具有各自的属性、方法和事件。例如，以上 4 种数据提供程序中的 Microsoft SQL Server .NET Framework 数据提供程序实现以下 4 种对象：SqlConnection 对象、SqlCommand 对象、SqlDataReader 对象和 SqlDataAdapter 对象。不同类别的数据提供程序的对象名前缀会有所不同。

(1) Connection 对象

Connection 对象用来建立与特定数据源的连接。在对数据源执行任何操作（包括读取、删除、新增或更新数据）前都必须建立数据库连接，下面以对 SQL Server 数据库的操作为例（注：在本书未显式说明处均指以 SQL Server 数据库作为数据源）介绍通过编程创建 Connection 对象（注：使用 SqlConnection 时需引入 System.Data.SqlClient 命名空间）的两种方法：一种方法是创建 Connection 对象后并设置其 ConnectionString 属性，另一种方法是利用 Connection 对象的构造方法创建。这两种创建方法的语法分别如下：

方法 1：

```
SqlConnection testConnection=new SqlConnection ();
string testconnectionString = "Data Source=localhost;Initial Catalog=library;
   User ID=lib;Password=123456;Integrated Security=sspi";
testConnection.ConnectionString= testconnectionString;
```

其中字符串 testconnectionString 也可以用以下方式声明：

```
string testconnectionString = "Server=localhost;Database=library;
   User ID=lib;Password=123456; Integrated Security=true";
```

方法 2：

```
string testconnectionString = "Data Source=localhost;Initial Catalog=library;
   User ID=lib;Password=123456;Integrated Security=sspi";
SqlConnection testConnection=new SqlConnection (testconnectionString);
```

以上代码的作用是声明一个将通过 Windows 身份验证打开通往本地计算机上的 SQL Server 的数据库连接，其连接到的数据库名为“library”。

其中 ConnectionString 属性是 Connection 对象最重要的属性。它包含建立连接所需的信息，而且是以字符串的形式提供的，所以也称其为“连接字符串”。根据所使用的 Microsoft .NET Framework 数据提供程序的不同，包含在连接字符串中的值也会有所不同。表 7-2 描述了连接字符串的几种常用参数，该表只包含部分参数列表，并非所有数据提供程序都支持这些参

数。

表 7-2 连接字符串的常用参数及其描述

参数	描述	默认值
Provider	设置或返回连接的 OLE DB 数据提供程序（仅 OLE DB .NET Framework 数据提供程序）	（空）
Connection Timeout 或 Connection Timeout	在数据源终止尝试和返回错误之前，连接到服务器所需等待的时间	15s
Data Source 或 Server	要连接到的数据源的名称，对于本地计算机上的 SQL Server 数据库，其可识别的值有“localhost”、“(local)”、“127.0.0.1”或英文的“.”；对于远程计算机上的 SQL Server 数据库则只能使用计算机的 IP 地址加数据源名称的形式，例如：“192.168.1.100\数据源名称”	（空）
Initial Catalog 或 Database	连接后要打开的数据库的名称	（空）
Integrated Security 或 Truste_Connection	如果此参数的值为 false，则必须指定其中的 User ID 与 Password；如该参数值设为 true，则数据源使用当前身份验证的 Microsoft Windows 账户凭证，则 User ID 与 Password 不起作用并可以忽略不写，其可识别的值有 “true”、“false”、 “yes”、“no”以及“sspi”(强烈推荐)，sspi 等价于 true	false
User ID 或 UID	如果 Integrated Security 的值为 false，则参数 User ID 为连接到数据源时使用的登录账户	
Password 或 Pwd	如果 Integrated Security 的值为 false，则参数 Password 为连接到数据源时使用的登录账户密码	
Persist Security Info	如果此参数的值为 false，且数据源处于打开连接状态时，数据源将不返回安全敏感信息，例如密码	false

注意：所有的 ConnectionString 都遵循相同的基本格式，它们由一系列通过分号来分隔的关键字和值组成。整个字符串通过单引号或双引号来划分：

```
"Keyword=value; Keyword=value; Keyword=value"
```

关键字不区分字母大小写，但根据数据源的不同，其值可能会区分字母大小写。若要包含有分号、单引号或双引号的值，则该值必须用双引号括起来。例如，如果数据库的名称是“Tom's Data”，则 ConnectionString 必须用"Database= Tom's Data"表示这种意思。如果使用'Database= Tom's Data'则会导致错误。如果多次使用相同的关键字，则最后的那个实例是有效。例如 ConnectionString 设置为"Database= Tom's Data; Database= library"，则数据库名称设为 library。另外只能在 Connection 对象关闭时才设置 ConnectionString 属性，此时 Connection

对象即检查字符串的语法。如果发现语法错误，就会产生一个异常。

创建数据库连接后，管理连接就很简单了——只要简单地使用 Open()方法和 Close()方法。Open()方法的功能是使用 ConnectionString 所指定的属性设置打开数据库连接，Close()方法是关闭与数据库的连接。

【例 7-1】数据库连接测试。

(1)新建一个名为【TestLibrary.sln】的解决方案，然后在该方案中添加一个【ASP.NET Web 应用程序】，接着在该 Web 程序中添加一个名为【TestConnection.aspx】的页面，在页面中添加一个标签，其代码如下：

例 7-1 代码 TestConnection.aspx 页面代码

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="TestConnection.aspx.cs"
    Inherits="Web.TestConnection" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>测试数据库连接状态</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="lblInfo" runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

(2)在【TestConnection.aspx】的后台代码页的 Page_Load 事件中编写如下代码：

例 7-2 代码 TestConnection.aspx 后台代码

```
protected void Page_Load(object sender, EventArgs e)
{
    string testconnectionString = "Data Source=localhost;Initial Catalog=library;
        User ID=lib;Password=123456;Integrated Security=sspi";
    SqlConnection testConnection = new SqlConnection(testconnectionString);
    try
    {
        testConnection.Open();
        lblInfo.Text = "数据库版本为：  " + testConnection.ServerVersion.ToString();
        lblInfo.Text += "<br/>数据库连接的状态为：" + testConnection.State.ToString();
```

```
            testConnection.Close();
            lblInfo.Text += "<br/>现在数据库连接的状态为：" +
               testConnection.State.ToString();
        }
        catch (SqlException ex)
        {
            lblInfo.Text = ex.Message.ToString();
        }
        finally
        {
            testConnection.Close();
        }
    }
```

程序运行结果：

数据库版本为：10.00.1600

数据库连接的状态为：Open

现在数据库连接的状态为：Closed

注意：打开数据库连接时，将面临 2 个可能的异常。如果连接字符串缺失了必需的信息或连接已打开，会发生 InvalidOperationException 异常。其他所有问题都会产生 SqlException 异常，包括连接数据库服务器、登录或访问特定数据库的错误。

数据库连接是有限的服务器资源。也就是说，连接要尽量晚打开而要尽早释放。上面的代码示例使用了异常处理程序，它确保即使有未处理的错误发生，连接也能够在 finally 块中关闭。如果不使用这样的设计而发生了未处理的异常，连接将一直保持到垃圾回收器释放 SqlConnection 对象。

另一种方法是把数据访问代码放入到 using 块中。using 语句声明在短期内使用一个可释放的对象。using 语句一旦结束，CLR 会立刻通过调用对象的 Dispose()方法释放相应的对象，Dispose()方法与 Close()方法等效。其代码编写为例 7-3，其运行结果与代码 7-2 是一致的。

批注 [X1]: 请确认

带格式的：字体颜色：红色

删除的内容: 下

例 7-3 代码 using 块在数据库连接中的使用

```
protected void Page_Load(object sender, EventArgs e)
{
    string testconnectionString = "Data Source=localhost;Initial Catalog=library;
       User ID=lib;Password=123456;Integrated Security=sspi";
    SqlConnection testConnection = new SqlConnection(testconnectionString);
    using (testConnection)
    {
        testConnection.Open();
        lblInfo.Text = "数据库版本为：  " + testConnection.ServerVersion.ToString();
        lblInfo.Text += "<br/>数据库连接的状态为：" + testConnection.State.ToString();
```

```
        }
        lblInfo.Text += "<br/>现在数据库连接的状态为：" + testConnection.State.ToString();
    }
```

(2) Command 对象

Command 对象用来对数据源执行 SQL 命令或存储过程。在使用 Command 执行一条 SQL 语句一般分为四步，第一步是创建 Command 对象实例并设置其相应的属性，第二步打开数据库连接，第三步是调用 Command 对象相应的方法执行操作，第四步是关闭数据库连接。

创建一个 Command 对象有两种方法：一种方法是创建一个 Command 对象并设置其相应的属性（一般是设置 CommandType、CommandText 和 Connection 三个属性，其中使用 CommandType 时需引入 System.Data 命令空间），另一种方法是利用 Command 对象的构造方法。下面以创建一个表示查询的 Command 对象为例演示这两种创建方法。

方法 1：

```
SqlCommand cmd=new SqlCommand();
cmd.Connection = testConnection; // testConnection为需使用的Connection对象
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT * FROM Book";//需对数据源执行的SQL语句或存储过程
```

方法 2：

```
SqlCommand cmd = new SqlCommand("SELECT * FROM Book ", testConnection);
```

其中 CommandType 是一个枚举值，具体取值如表 7-3 所示。

表 7-3 CommandType 枚举值及其描述

值	描述
CommandType.Text	该命令将执行一条 SQL 语句，该选项为默认值
CommandType.StoredProcedure	该命令将将执行数据源中的一个存储过程，通过 CommandText 属性设置存储过程的名称
CommandType.TableDirect	该命令将查询表中的所有记录，CommandText 属性设置要从中读取记录的表的名字

Command 对象提供了 3 个方法来执行命令，表 7-4 列出了这些方法。

表 7-4 Command 方法及其描述

方法	描述
ExecuteNonQuery()	执行非 SELECT 语句，如 INSERT、UPDATE、DELETE 语句，返回值显示命令影响的行数
ExecuteScalar()	执行 SELECT 查询，并返回命令生成的记录集的第一行第一列的字段值。该方法常用来执行包含 COUNT()、SUM()等聚合函数的 SELECT 语句，用于返回单个值

ExecuteReader()	执行 SELECT 查询，并返回一个封装了只读、只进游标的 DataReader 对象

(3)DataReader 对象

DataReader 对象用来从数据源以只进、只读流的方式每次读取一条 SELECT 命令返回的记录，这种方式有时称为流水游标。使用 DataReader 是获得数据最简单的方式，不过它缺乏非连接的 DataSet 所具有的排序等功能，但绑定数据时比使用数据集方式性能要高，因为它是只读的，所以如果要对数据库中的数据进行修改就需要借助其它方法将所作的更改保存到数据库。表 7-5 列出了 DataReader 的核心方法。

表 7-5 DataReader 方法及其描述

方法	描述
Read()	将行游标前进到流的下一行（DataReader 刚创建时，行游标在第一行之前）。当还有其他行时，Reader()方法返回 true，如果已经是最后一行，则返回 false
GetValue()	返回当前行中指定序号的字段值（第一个字段的序号为 0），返回的数据类型是.NET 中和数据源类型最相似的那一个。如果使用序号访问字段不小心指定了不存在的序号，会得到 IndexOutOfRange-Exception 异常。也可使用 DataReader 的索引字段名称得到字段值(即 myDataReader.GetValue[0]与 myDataReader["UserName"]等效)
GetInt32()、GetChar()、GetDateTime()和 GetXxx()	这些方法返回当前行中指定序号的字段值，返回值的类型和方法名称中的一致。如果试图将返回结果赋给一个错误类型的变量，会得到一个 InvalidCastException 异常。还要注意这些方法不支持空数据类型，如果字段可能包含空值，那么必须在调用这些方法前对其进行检查
NextResult()	如果命令返回的 DataReader 包含多个行集，该方法将游标移动到下一个行集（刚好在第一行以前）
Close()	关闭 Reader。如果原命令执行一个带有输出参数的存储过程，该参数仅在 Reader 关闭后才可读

DataReader 对象不能被直接实例化，必须借助与相关的 Command 对象来创建其实例，例如用 SqlCommand 的实例的 ExecuteReader()方法可以创建 SqlDataReader 实例，如：

```
SqlDataReader reader= command.ExecuteReader();
```

因为 DataReader 对象读取数据时需要与数据库保持连接，所以在使用完 DataReader 对象读取完数据之后应该立即调用它的 Close()方法关闭，并且还应该关闭与之相关的 Connection 对象。在.net 类库中提供了一种方法，在关闭 DataReader 对象的同时自动关闭掉与之相关的 Connection 对象，使用这种方法是可以为 ExecuteReader()方法指定一个参数，如：

```
SqlDataReader reader =command.ExecuteReader(CommandBehavior.CloseConnection);
```

CommandBehavior 是一个枚举，上面使用了 CommandBehavior 枚举的 CloseConnection 值，它能在关闭 SqlDataReader 时关闭相应的 SqlConnection 对象。

7.1.4 配置文件 Web.config 及数据库连接字符串的设置

在项目开发过程中，只要页面与数据库有交互则该页面都要重写一次连接字符串的代码，由此不仅显得累赘，而且当该连接字符串有更改时则要每个页面都要修改一次，使得代码的可维护性大大降低。为解决这个问题，通常的做法是将数据库连接字符串保存在配置文件 Web.config 中。

Web.config 文件是一个 XML 文本文件，它用来储存 ASP.NET Web 应用程序的配置信息（如最常用的设置 ASP.NET Web 应用程序的身份验证方式），它可以出现在应用程序的每一个目录中。ASP.NET 配置文件的所有内容都嵌套在根元素<configuration>里。这个元素包含一个<system.web>元素，它用于配置各个方面的单独元素，此外还有<appSettings>元素（用于存储自定义设置）和<connectionStrings>元素（用于存储你使用的或其他 ASP.NET 特性依赖的连接到数据库的字符串）。

将数据库连接字符串设置在 Web.config 文件<connectionStrings>元素中的代码为：

```
<connectionStrings>
    <add name="ConnectionString"     connectionString="Data Source=localhost;
      Initial Catalog=library; User ID=lib;Password=123456;Integrated Security=true"
      providerName="System.Data.SqlClient" />
</connectionStrings>
```

注意：与所有的 XML 文档一样，Web.config 文件是区分大小写的。每个设置使用驼峰式命名法（骆驼式命名法就是当变量名或函数名是由一个或多个单词连结在一起，而构成的唯一识别字时，第一个单词以小写字母开始；第二个单词的首字母大写或每一个单词的首字母都采用大写字母，例如：myFirstName、myLastName）并以小写字母开始。即不能把<system.web>写为<System.Web>。

在页面后台代码中读取配置文件的数据库连接字符串代码如下：

```
string connectionString = System.Configuration.ConfigurationManager.
  ConnectionStrings["ConnectionString"].ToString();
```

7.1.5 代码编写规范

1.统一编程风格的意义

- 增加开发过程代码的强壮性、可读性、易维护性
- 减少有经验和无经验开发人员编程所需的脑力工作
- 为软件的良好维护性打下好的基础
- 在项目范围内统一代码风格
- 通过人为以及自动的方式对最终软件应用质量标准
- 使新的开发人员快速适应项目氛围
- 支持项目资源的复用：允许开发人员从一个项目区域（或子项目团队）移动到另一

个，而不需要重新适应新的子项目团队的氛围

- 一个优秀而且职业化的开发团队所必需的素质

2.变量命名规则

- 前缀（小写字母加下划线）表明变量的作用域，无前缀则表明是局部变量或函数的参数。如：
  - m_xx　　表示是类的成员变量，控件变量例外
  - g_xx　　表示是全局变量，在 C#中，也可以理解为在整个项目中都可能用到的静态变量
  - c_xx或者 XX　　表示是一个常量
- 用数据类型全称中的关键字母代表特定的数据类型（一个或多个小写字母），如下表 7-6。

表 7-6 常用数据类型缩写

数据类型	常用数据类型缩写
int	i
bool	b
string	str
char	c
float	f
double	d
object	ob
Label	lbl
TextBox	txt
Button	btn
ComboBox	cmb
Mainmenu	mnu
MenuItem	mnuItem
CheckBox	chk
DataGrid	grd
Timer	tm
Form	frm
Panel	pnl
GroupBox	gup
TreeView	tv
RadioButton	rdo
ListBox	lb
ToolBar	tlb

DateTime	dt
Connection	cn
Command	cmd
DataSet	ds
DataAdapter	da
DataView	dv
DataTable	dbTable
DataReader	dbReader
Parameter	param
DataRow	dbRow
DataColumn	dbCol

注：如果模块中只有一个类实例对象，则可以只用简写。如 Connection 对象可以用 cn 来命名。

3. 函数命名规则

- 函数名用首字母大写的英文单词组合表示（如用动词+名词的方法），其中至少有一个动词
- 应该避免的命名方式
  - 和继承来的函数名一样。即使函数的参数不一样，也尽量不要这么做，除非想要重载它
  - 只由一个动词组成，如：Save、Update。改成如：SaveValue、UpdateDataSet 则比较好
- 函数参数的命名规则
  - 函数参数应该具有自我描述性，应该能够做到见其名而知其意
  - 用匈牙利命名法命名

4. 类命名规则

- 类的命名通常以父类的简写开头。如：FrmXXX 可看出该类从 Form 中继承而来
- 类名中尽量不要出现下划线
- 类变量的命名可以参照，如：FrmXXX frmXXX = new FrmXXX()，即首字母小写即可

5. 常见语句书写规则

常用语句书写规则如表 7-7 所示：

表 7-7 常用语句书写风格

语句	提倡的风格

if	if(condition)
	{
	statements;
	}

	else
	{
	statements;
	}

for	for(initialization; condition; update)
	{
	statements;
	}

foreach	foreach(something in collection)
	{
	statements;
	}

switch	switch(…)
	{
	case ..:
	break;
	case …:
	break;
	default:
	}

while	while(..)
	{
	statements;
	}

do-while	do
	{
	statements;
	}
	while(condition);

try-catch	try
	{
	statements;
	}
	catch(Exception e)
	{
	handle exception;
	}

同一代码块内的不同逻辑块之间应空一行	{
	do statement1;
	
	do statement2;
	}

函数与函数之间至少空一行，但不超三行	

注意：代码列宽尽量控制在 110 字符左右，缩进应该是每行一个 Tab（4 个空格）。当书写的语句或表达式超出或即将超出规定的列宽，遵循以下规则进行换行：

(1)在逗号后换行

(2)在操作符前换行

(3)规则(1)优先于规则(2)

6. 注释风格

(1)文件注释

文件的头的注释要求如下：第一行和最后一行的*号所在的位置为第 70 字母位，双斜杠后有一个空格，注释的内容有文件名、版权、作者、创建日期、主要内容。编写格式如下：

```
// ************************************************************
// 文件名：BookBorrowDAL.cs
// Copyright (C) 2014-2015 图书管理系统
// 作者：熊国华
// 创建日期：2014-6-1
// 主要内容：提供获取读者已借阅但未归还的数量、借阅图书、
//           获取读者是否已借阅该书、获取已借书数量、
//           获取单本图书的借阅信息、图书归还和
//           获取已借阅但未归还的图书信息等功能
// ************************************************************
```

(2)方法注释

该类注释采用.NET 已定义好的 XML 标签来标记，在声明接口、类方法、属性、字段都应该使用该类注释，以以便代码完成后直接生成代码文档，让别人更好的了解代码的实现和接口。编写格式如下：

```
/// <summary>
/// 获取单本图书的借阅信息
/// </summary>
/// <param name="readerId">读者ID</param>
/// <param name="bookId">图书ID</param>
/// <returns>单本图书信息实体</returns>
public static BookBorrow GetSingleBookBorrowInfo(int readerId, string bookId)
{
```

```
        //实现代码
    }
```

(3)单行注释与多行注释

该类注释用于方法内的代码注释。如变量的声明、代码或代码段的解释。编写格式如下：

```
        //打开数据库连接
        conn.Open();
```

如果注释语句较多，可以分为多行书写，编写格式如下：

```
        /*
        *注释语句
        *
        */
```

【实施与测试】

1. 在项目解决方案中添加一个名为【login.aspx】的页面，其页面设计如下图 7-3 所示：

图 7-3 login.aspx 页面运行效果

例 7-4 代码 login.aspx 页面代码

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="login.aspx.cs"
      Inherits="Web.login" %>
    <%@ Register src="~/Controls/top2.ascx" tagname="usertop" tagprefix="uc1" %>
    <%@ Register src="~/Controls/bottom.ascx" tagname="bottom" tagprefix="uc2" %>
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>用户登录</title>
    <style type="text/css">
        .style1 { width: 103px; }
        .style2 { width: 356px; }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <table align="center">
        <tr>
            <td>
                <uc1:usertop ID="usertop1" runat="server" />
                <table align="center">
                    <tr>
                        <td align="center" colspan="2">
                            用户登录
                        </td>
                    </tr>
                    <tr>
                        <td class="style1" align="right">
                            <asp:Label ID="Label1" runat="server"
                              Text="用户名：">
                            </asp:Label>
                        </td>
                        <td class="style2">
                            <asp:TextBox ID="txtname" runat="server"
                              Width="180px">
                             </asp:TextBox>
                        </td>
                    </tr>
                    <tr>
                        <td class="style1" align="right">
                            <asp:Label ID="Label2" runat="server"
                              Text="密码：">
                            </asp:Label>
                        </td>
                        <td class="style2">
```

```
                        <asp:TextBox ID="txtpassword" runat="server"
                            Width="180px" TextMode="Password">
                        </asp:TextBox>
                    </td>
                </tr>
                <tr>
                    <td align="center" colspan="2">
                        <asp:Button ID="btnLogin" runat="server" Text="登录"
                            OnClick="btnLogin_Click" Width="80px" /> 
                        <asp:Button ID="btncancel" runat="server" Text="取消"
                            Width="80px" OnClick="btnCancel_Click" />
                    </td>
                </tr>
            </table>
            <uc2:bottom ID="bottom1" runat="server" />
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

2.分别编写“登录”按钮和“取消”按钮的单击事件代码如下：

例 7-5 代码 “登录”按钮和“取消”按钮的单击事件代码

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    string managername = txtname.Text.Trim();
    string managerpassword = txtpassword.Text.Trim();
    string querySql = "SELECT * FROM Admin WHERE UserName='" + managername
      + "' and UserPassword='" + managerpassword + "'";
    //获取数据库连接字符串
    string connectionString = System.Configuration.ConfigurationManager.
      ConnectionStrings["ConnectionString"].ToString();
    //声明数据库连接
    SqlConnection conn = new SqlConnection(connectionString);
    SqlCommand cmd = new SqlCommand(querySql, conn);

    //打开数据库连接
```

```
        conn.Open();
        //执行SQL语句并返回结果
        SqlDataReader reader = cmd.ExecuteReader();
        if (reader.Read())
        {
            //将用户名存入Session中
            Session["managername"] = managername;
            Boolean IsAdmin =Boolean.Parse(reader["IsAdmin"].ToString());
            if (IsAdmin)
            {
                //跳转到管理员页面
                Response.Redirect("Admin/managerindex.aspx");
            }
            else
            {
                //跳转到用户页面
                Response.Redirect("index.aspx");
            }
        }
        else
        {
            //提示出错信息
            Response.Write("<script>alert('用户名或密码错误，登录失败！')</script>");
        }
        //关闭数据库连接
        conn.Close();
    }

    protected void btnCancel_Click(object sender, EventArgs e)
    {
        txtname.Text =string.Empty;
        txtpassword.Text =string.Empty;
    }
```

7.2 子任务：修改用户信息

【任务陈述】

用户登录后可以对用户的信息进行修改操作，常用的操作就是修改密码。修改密码的操作过程是首先要求输入正确的当前密码，接着输入新密码及确认新密码，两次输入的密码必须一致且不能和当前密码相同才能进行保存。

【知识准备】

7.2.1 在客户端输出提示脚本信息

在开发过程有时经常需要在客户端弹出一些提示信息对话框，常用的方法是利用 Page.RegisterStartupScript 方法实现，该方法允许 asp.net 服务器控件在 Page 对象的 <form runat=server> 元素的结束标记之前发出客户端脚本块。方法定义如下：

```
public virtual void RegisterStartupScript(string key, string script);
```

其中参数 key 表示标识脚本块的唯一键，另一参数 script 表示要发送到客户端的脚本的内容。

【例 7-2】Web 页面经常需要提示用户的操作结果，如提示“修改成功!”。

例 7-6 代码 在客户端输出提示信息

```
protected void Page_Load(object sender, EventArgs e)
{
    Page.RegisterStartupScript("", "<script    Language='Javascript'>
      alert('修改成功');</script>");
}
```

程序运行结果：

7.2.2 内容页访问母版页中用户控件中的控件

母版页的使用是为了保证网站所有的页面具有相同的设计或布局，如页头、菜单栏、广告条和欢迎信息栏等。例如为了在欢迎信息栏中显示当前登录的用户名，就必须涉及访问母版页中的控件。

访问母版页中控件最常用的方法是在页面的 Page_Load 事件中利用内容页 page 对象的 Master 公共属性，进而使用母版页的 FindControl 方法来实现对母版页控件的访问。例如：

例如在内容页访问母版页中 ID 为 top1 用户控件可以用如下方法：

```
usertop tp = Page.Master.FindControl("usertop1") as usertop;
```

【例 7-3】内容页访问母版中用户控件中的控件

例 7-7 代码 设置母版中用户控件中的控件的属性

```
protected void Page_Load(object sender, EventArgs e)
{
    usertop tp = Page.Master.FindControl("usertop1") as usertop;
    Label lblLoginUser = tp.FindControl("lblLoginUser") as Label;
    if (Session["managername"] != null)
        lblLoginUser.Text = Session["managername"].ToString();
}
```

【实施与测试】

1、首先创建一个密码修改页面【ModifyPassword.aspx】，运行效果如图 7-4 所示：

图 7-4 ModifyPassword.aspx 页面运行效果

例 7-8 代码 ModifyPassword.aspx 页面代码

```
<%@ Page Title="" Language="C#" MasterPageFile="~/UserPage.Master"
    AutoEventWireup="true" CodeBehind="ModifyPassword.aspx.cs"
    Inherits="Web.ModifyPassword" %>
```

批注 [X2]: 下面换了一个图，因为要截全界面的效果，感觉换了差不多

删除的内容:

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
<table style="width:100%;">
        <tr>
            <td class="style2">
                <font color="maroon"size="3">当前位置：修改密码</font></td>
        </tr>
        <tr>
            <td <t>
                able border="1" cellpadding="4" width="830px">
                    <tr>
                        <td align="center">当前密码：
                            <asp:TextBox ID="txtOldPwd" runat="server"
                                TextMode="Password"></asp:TextBox>
                            <span style="color: #FF0000">*</span>
                        </td>
                    </tr>
                    <tr>
                        <td align="center">  新密码：
                            <asp:TextBox ID="txtNewPwd" runat="server"
                                TextMode="Password"></asp:TextBox>
                            <span style="color: #FF0000">*</span>
                        </td>
                    </tr>
                    <tr>
                        <td align="center">确认密码：
                            <asp:TextBox ID="txtConfirmPwd" runat="server"
                                TextMode="Password"></asp:TextBox>
                            <span style="color: #FF0000">*</span>
                        </td>
                    </tr>
                </table>
            </td>
        </tr>
        <tr>
            <td align="center">
                <asp:Button ID="btnSave" runat="server" Text="保存"
```

```
                    onclick="btnSave_Click" /> 
                <asp:Label ID="lblMessage" runat="server"
                    ForeColor="Red"></asp:Label>
            </td>
        </tr>
    </table>
</asp:Content>
```

2.编写 ModifyPassword.aspx 页面后台代码如下：

例 7-9 代码 ModifyPassword.aspx 页面后台代码

```
public partial class ModifyPassword : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        usertop tp = Page.Master.FindControl("usertop1") as usertop;
        Label lblLoginUser = tp.FindControl("lblLoginUser") as Label;
        if (Session["managername"] != null)
            lblLoginUser.Text = Session["managername"].ToString();
    }

    protected void btnSave_Click(object sender, EventArgs e)
    {
        string oldPwd = txtOldPwd.Text.Trim();
        string newPwd = txtNewPwd.Text.Trim();
        string confirmPwd = txtConfirmPwd.Text.Trim();
        string connectionString = System.Configuration.ConfigurationManager.
          ConnectionStrings["ConnectionString"].ToString();
        SqlConnection conn = new SqlConnection(connectionString);
        SqlCommand cmd;
        if (oldPwd == string.Empty)
        {
            Page.RegisterStartupScript("", "<script>alert('请输入当前密码！')</script>");
            return;
        }
        else
        {
            string selectSql = string.Format("select * from Admin
              where UserName='{0}'", Session["managername"].ToString());
```

```
            cmd = new SqlCommand(selectSql, conn);
            conn.Open();
            SqlDataReader reader =cmd.ExecuteReader();
            if (reader.Read())
            {
                string pwd = reader["UserPassword"].ToString();
                reader.Close();
                conn.Close();
                if (oldPwd != pwd)
                {
                    Page.RegisterStartupScript("", "<script>
                        alert('当前密码输入错误！')</script>");
                    return;
                }
            }
            else
            {
                reader.Close();
                conn.Close();
            }
        }
        if (newPwd == string.Empty)
        {
            Page.RegisterStartupScript("", "<script>alert('请输入新密码！')</script>");
            return;
        }
        if (newPwd != confirmPwd)
        {
            Page.RegisterStartupScript("", "<script>
              alert('两次输入的密码不一致！')</script>");
            return;
        }
        if (newPwd == oldPwd)
        {
            Page.RegisterStartupScript("", "<script>
              alert('新密码不能与旧密码相同！')</script>");
            return;
        }
```

```
            string updateSql = string.Format("update Admin Set UserPassword='{0}'
               where UserName='{1}'", newPwd, Session["managername"].ToString());
            cmd = new SqlCommand(updateSql, conn);
            conn.Open();
            int result = cmd.ExecuteNonQuery();
            conn.Close();
            if (result>0)
            {
                lblMessage.Text = "保存成功！";
                this.btnSave.Visible = false;
            }
            else
            {
                lblMessage.Text = "保存失败！";
            }
        }
    }
```

7.3 子任务：用户信息查询

【任务陈述】

管理员登录后可以对普通用户的信息进行查询操作。

【知识准备】

7.3.1 DataSet

在 ADO.NET 中，DataSet 提供了独立于数据源的关系编程模型，是断开连接数据访问的核心。DataSet 表示整个数据集，它包含两类最重要的元素：零个或多个表的集合（通过 Tables 属性提供）以及零个或多个关系的集合（通过 Relations 属性提供），关系可以把表连接到一起，图 7-5 显示了 DataSet 的基本架构。

批注 [X3]: 此句正确

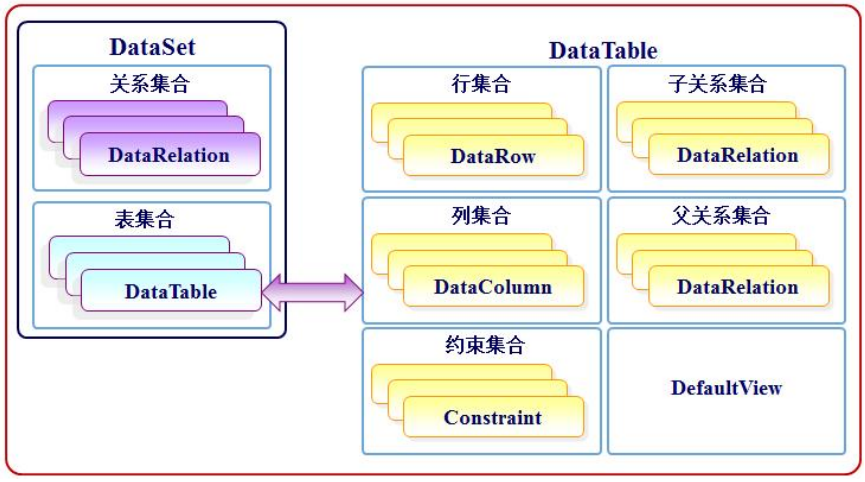

图 7-5 DataSet 基本架构

从图 7-5 可以看出，DataSet.Tables 集合里的每个项目是一个 DataTable。DataTable 又包含自己的集合—DataColumn 对象的 Columns 集合（它描述每个字段的名称和数据类型）以及 DataRow 对象的 Rows 集合（它包含每条记录的真正数据）。

DataTable 里的每条记录由一个 DataRow 对象表示。每个 DataRow 对象表示由数据源取得的表的一条记录。DataRow 是真正字段值的容器。可以通过字段名称访问它们，如 myRow["FieldName"]。注意，使用 DataSet 对象工作时，根本不会直接影响到数据源里的数据，所有变化只是作用到本地内存里的 DataSet。DataSet 从不保存任何类型的数据源连接。

创建 DataSet 对象的语法如下：

```
DataSet    ds=new DataSet();//需引入命令空间using System.Data;
```

7.3.2 DataAdapter

要将数据源中的记录提取并填入 DataSet 的 Tables 集合中，还需要使用另一个 ADO.NET

对象 DataAdapter。它是提供程序相关的对象，因此每一个提供程序都有一个 DataAdapter 类（如 SqlDataAdapter、OracleDataAdapter 等）。

DataAdapter 对象是 Connection 对象和数据集（DataSet）之间的桥梁，它含有查询和更新数据源所需的全部命令。

为了让 DataAdapter 能够编辑、删除或添加行，需要设定 DataAdapter 对象的 UpdateCommand、DeleteCommand 和 InsertCommand 属性。利用 DataAdapter 填充 DataSet 时，必须使用.NET Framework 数据提供程序的 Connection 对象连接至数据源，并设定 SelectCommand 属性（主要设置 SQL 语句），最后调用 Fill 方法即可。注意，DataAdapter 仅在调用 Fill 方法填充 DataSet 对象时才使用数据库连接，在完成填充后将释放所有的服务器资源。

DataAdapter 提供了 3 个主要的方法，如表 7-6 所示。

表 7-6 DataAdapter 方法及其描述

方法	描述
Fill()	执行 SelectCommand 中的查询后，向 DataSet 添加一个 DataTable。如果查询返回多个结果集，该方法将一次添加多个 DataTable 对象。还可以用该方法向现有的 DataTable 添加数据
FillSchema()	执行 SelectCommand 中的查询，但只获取架构信息，可以向 DataSet 添加一个 DataTable。该方法并不往 DataTable 添加任何数据。相反，它只利用列名、数据类型、主键和唯一约束等信息预配置 DataTable
Update()	检查 DataTable 中的所有变化并执行适当的 UpdateCommand、DeleteCommand 和 InsertCommand 操作为数据源执行批量更新

填充 DataSet 的示例代码如下：

```
SqlConnection conn = new SqlConnection(connectionString);
string sqlStr = "SELECT [UserName], [UserPassword],[IsAdmin] FROM [Admin]";
SqlDataAdapter da=new SqlDataAdapter(sqlStr,conn);
DataSet    ds=new DataSet();
da.Fill(ds, " Admin");
```

7.3.3 GridView 控件

GridView 控件是 ASP.NET 服务器控件中功能最强大、最实用的一个数据绑定控件，通常将 GridView 的 DataSource 设置为 DataSet 中的某个 DataTable，从而以二维表格方式显示该 DataTable 中的数据，详细的使用将在第八章中讲述。

【实施与测试】

1、首先在网站根目录下的Admin文件夹中创建一个用户查询页面【userSearch.aspx】，点击“用户查询”按钮后的运行效果如图7-6所示：

批注 [X4]: 换图也差不多

图 7-6 userSearch.aspx 用户查询页面运行效果

例 7-10 代码 userSearch.aspx 页面代码

```
    <%@ Page Title="" Language="C#" MasterPageFile="~/Admin/MasterPage.Master"
AutoEventWireup="true" CodeBehind="userSearch.aspx.cs" Inherits="Web.Admin.userSearch"
%>
    <asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
    </asp:Content>
    <asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <table style="width:100%;">
            <tr>
                <td class="style2">
                    <font color="maroon"size="3">当前位置：用户查询</font></td>
            </tr>
            <tr>
                <td>用户名称：
                    <asp:TextBox ID="txtUserName" runat="server"/>  
                    <asp:Button ID="btnSearchUser" runat="server" Text="查找用户"
                        onclick="btnSearchUser_Click" />
```

```
            </td>
        </tr>
        <tr><td><hr style="color:#5D7B9D" /></td></tr>
        <tr>
            <td>用户信息</td>
        </tr>
        <tr>
            <td>
                <asp:GridView ID="gvUserInfo" runat="server"
                  AutoGenerateColumns="False" DataKeyNames="UserName"
                  CellPadding="4" ForeColor="#333333" HorizontalAlign="Left"
                  Width="830px"   Height="65px" AllowPaging="false"
                  EmptyDataText="无用户信息！">
                    <RowStyle BackColor="#F7F6F3" ForeColor="#333333" />
                    <Columns>
                        <asp:TemplateField HeaderText="序号"
                          HeaderStyle-Width="40px">
                            <ItemTemplate>
                                <%#Container.DataItemIndex + 1 %>
                            </ItemTemplate>
                        </asp:TemplateField>
                        <asp:BoundField HeaderText="用户名称"
                          DataField="UserName" />
                        <asp:BoundField HeaderText="密       码"
                          DataField="UserPassword" />
                        <asp:BoundField HeaderText="是否管理员"
                          DataField="IsAdmin" />
                    </Columns>
                    <HeaderStyle BackColor="#5D7B9D" Font-Bold="True"
                      ForeColor="White" />
                </asp:GridView>
            </td>
        </tr>
    </table>
</asp:Content>
```

2.编写 userSearch.aspx 页面后台代码如下：

例 7-11 代码 userSearch.aspx 页面后台代码

```
public partial class userSearch : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            top tp = Page.Master.FindControl("top1") as top;
            Label lblLoginUser = tp.FindControl("lblLoginUser") as Label;
            if (Session["managername"] != null)
                lblLoginUser.Text = Session["managername"].ToString();
        }
    }

    protected void btnSearchUser_Click(object sender, EventArgs e)
    {
        string connectionString = System.Configuration.ConfigurationManager.
          ConnectionStrings["ConnectionString"].ToString();
        string userName = this.txtUserName.Text.Trim();
        string selectSql = "SELECT UserName,UserPassword,IsAdmin=CASE IsAdmin
          WHEN '1' THEN '是' ELSE '否' END "
          +"FROM Admin WHERE UserName LIKE '%" + userName + "%'";
        SqlConnection conn = new SqlConnection(connectionString);
        SqlDataAdapter da = new SqlDataAdapter(selectSql, conn);
        DataSet ds = new DataSet();
        da.Fill(ds, "Admin");
        this.gvUserInfo.DataSource = ds.Tables["Admin"].DefaultView;
        this.gvUserInfo.DataBind();
    }
}
```

项目重现

完成网上购物系统的用户管理模块

1．项目目标

完成本项目后，您能够 ：

- 实现网上购物系统用户登录功能

- 实现网上购物系统用户信息修改功能
- 实现网上购物系统用户信息查询功能

2．知识目标

完成本项目后，您应该掌握：

- 掌握利用 ADO.NET 数据访问技术（如 Connection 对象，Command 对象，DataReader 对象，DataAdaper 对象，DataSet 对象等）对数据库进行访问操作
- 掌握数据库连接信息存取及配置文件 Web.config

3．项目介绍

对网上购物系统后台用户管理功能上的实现。首先是后台登录的验证，其次是用户信息的修改（如密码的修改等）、对用户信息的查询等功能。

4．项目内容

（1）用户登录功能，即对用户名和密码的合法性进行判断，如果不合法则显示相应的提示信息，如果合法则要区分用户类型是管理员还是普通用户，如果是管理员则跳转到后台管理首页，如果是普通用户则跳转至用户个人主页。

（2）修改用户信息功能，修改用户的相关信息，如登录密码的修改等。

（3）信息查询功能，管理员对所有普通用户的信息进行查询操作。