第三篇　数据库编程

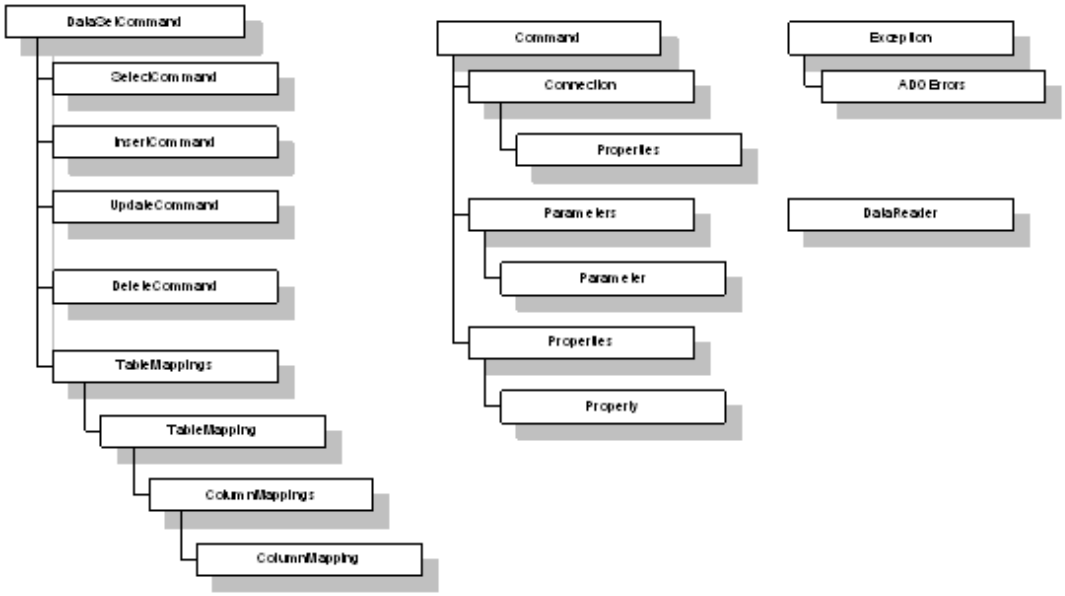
第一章　　基本概念

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。

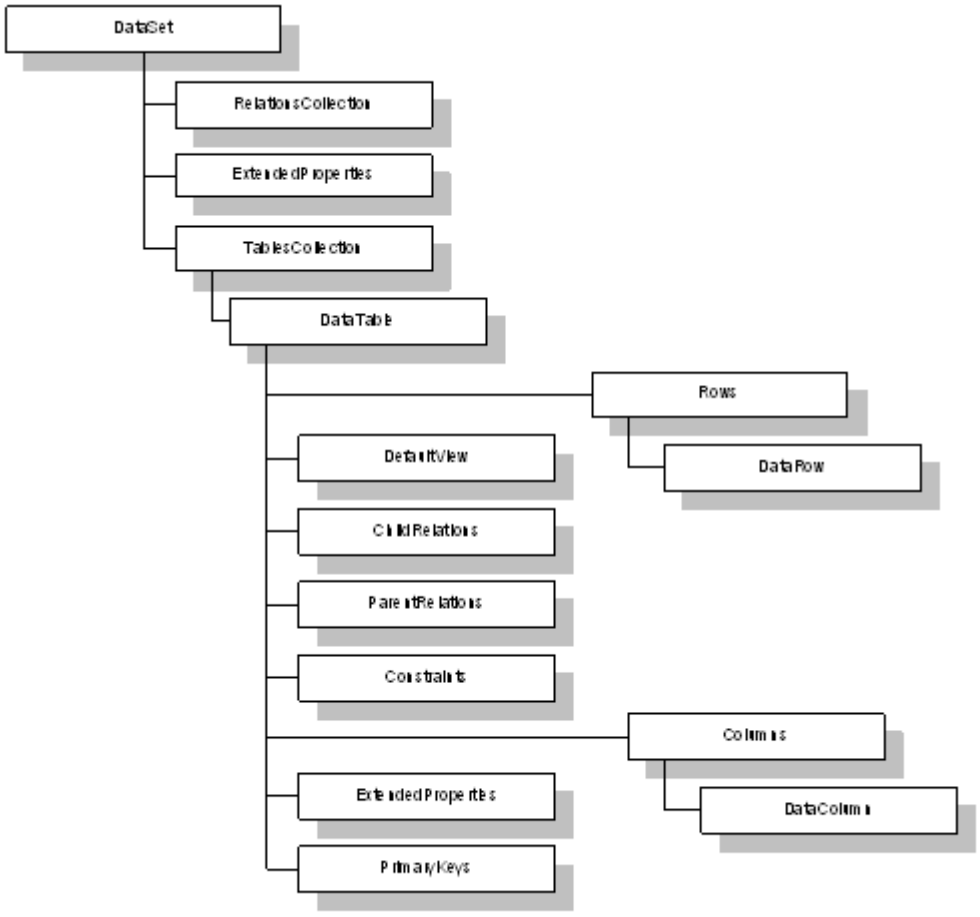
过去，数据处理主要依赖于两层结构，并且是基于连接的。连接断开，数据就不能再存取。现在，数据处理被延伸到三层以上的结构，相应地，程序员需要切换到无连接的应用模型。这样，DataSetCommand 就在 ADO.NET 中扮演了极其重要的角色。它可以取回一个

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()
```

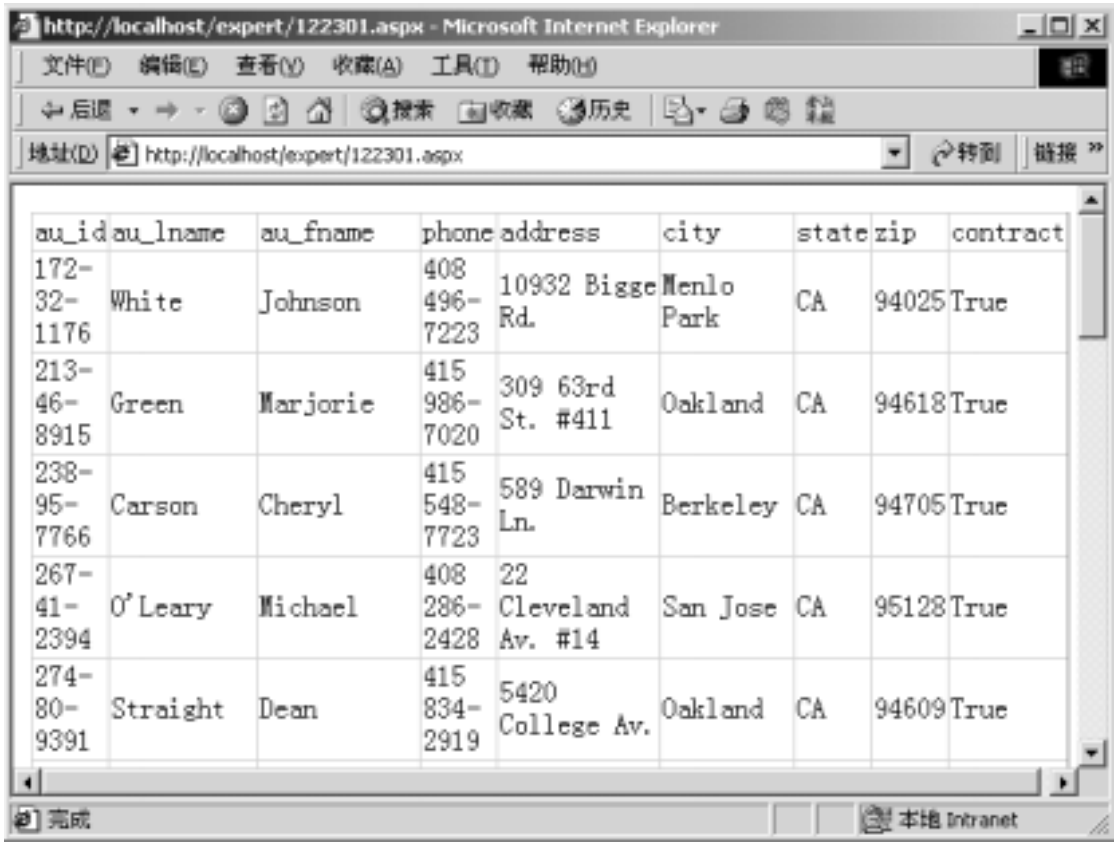
3.2.4 完整的代码

(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

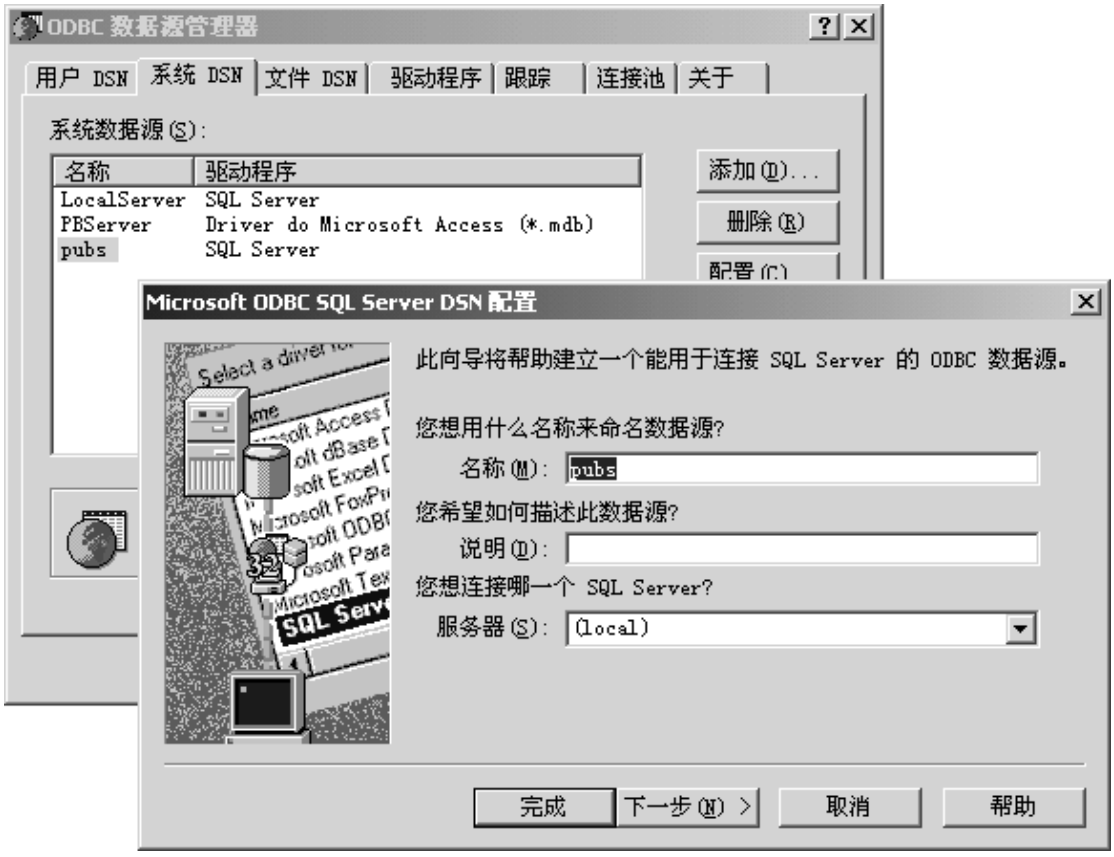
2、SQL Managed Provider

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 语句例子	ADOCommand cmd = new ADOCommand("SELECT * FROM Authors", con); ADODataReader dr = new ADODataReader(); cmd.Execute(out dr);	SQLCommand cmd = new 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

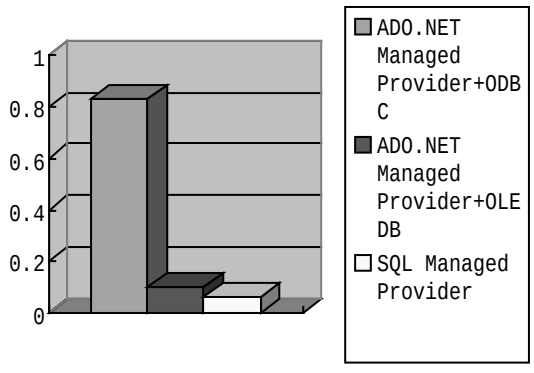
```
<%@ Page Language="vb" %>
<%@ Import Namespace = "System.Data" %>
<%@ Import Namespace = "System.Data.SQL" %>
<html>
  <head>
  <script runat=server>
    Sub Page_Load(ByVal Sender As Object, ByVal e As EventArgs)
        'On Error Resume Next
        Dim cn          As SQLConnection
```

```
        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

- **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。

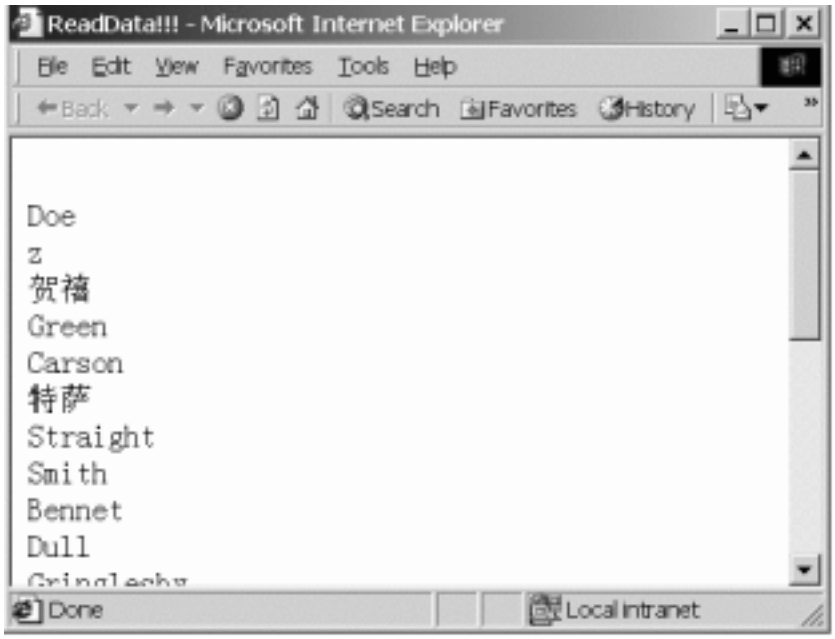
3.3.4 DataReader 的使用方法

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：

	ID	Name	Notes	FatherID	status
编辑	1	英语一角->.NET	english	1	0
编辑	2	菜鸟之家	欣喜之窗	1	0
编辑	3	大学园地	很好玩的啊	1	0
编辑	4	DIY高手	新手之路	1	0
编辑	5	超频杂谈	杂坦	1	0
编辑	6	JAVA	sun java	2	0
编辑	7	媒体开发	媒体开发	3	0
编辑	8	情感专栏	情感专栏	4	0
编辑	9	爱情故事	爱情故事	4	0
编辑	10	网络游戏	网络游戏	5	0
编辑	11	sssssssss	sssssssss	1	0
编辑	12	wwwwwwwww	wwwwwwww	1	0

然后我们点击“ 编辑 “ 连接，进入编辑页面，如下：

	ID	Name	Notes	FatherID
修改 取消	1	英语一角->.NET	english	1
编辑	2	菜鸟之家	欣喜之窗	1
编辑	3	大学园地	很好玩的啊	1
编辑	4	DIY高手	新手之路	1
编辑	5	超频杂谈	杂坦	1
编辑	6	JAVA	sun java	2
编辑	7	媒体开发	媒体开发	3
编辑	8	情感专栏	情感专栏	4
编辑	9	爱情故事	爱情故事	4
编辑	10	网络游戏	网络游戏	5
编辑	11	sssssssss	sssssssss	1
编辑	12	wwwwwwwww	wwwwwwww	1

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

- **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

- **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

- **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。

- **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：

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

- **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。

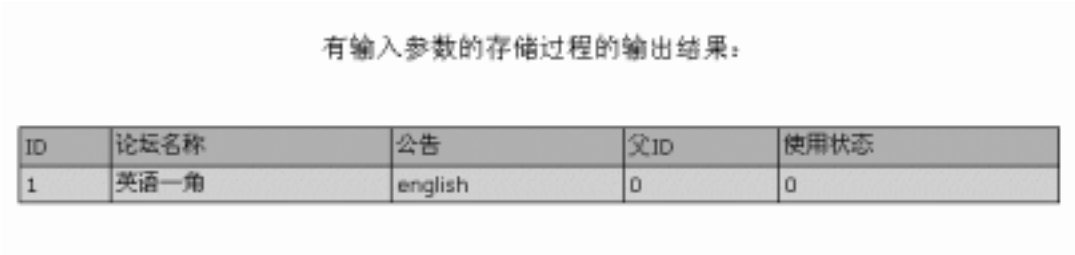
我们创建一个 aspx 文件来调用我们的存储过程，下面是我们的文件的代码：

（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>
```



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。

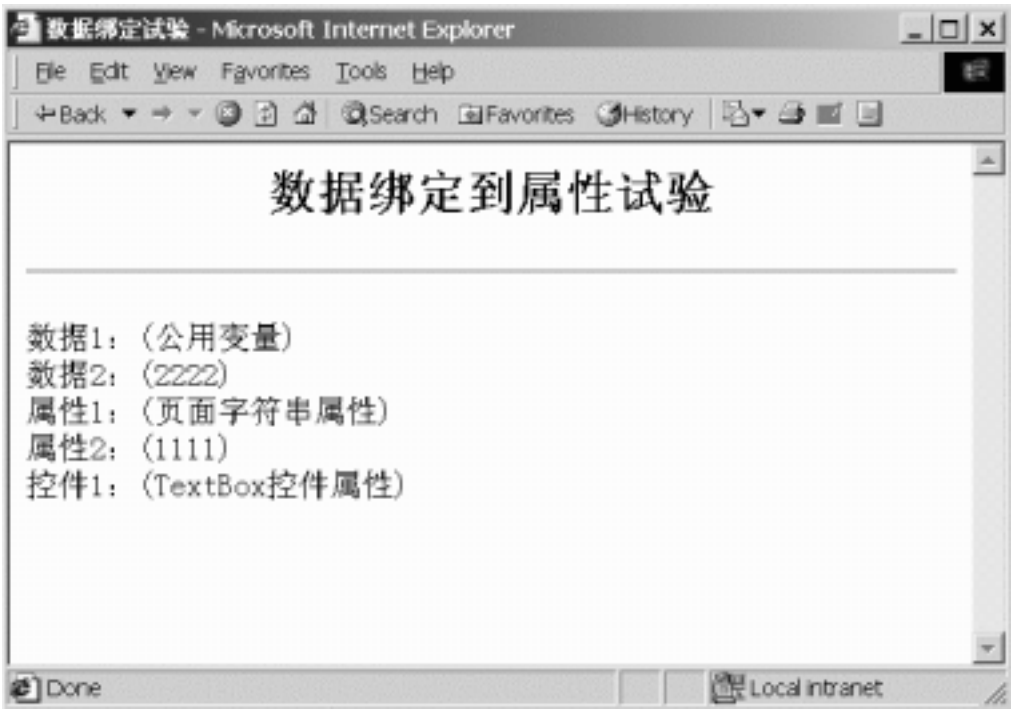
源程序如下：

```
<!-- 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。