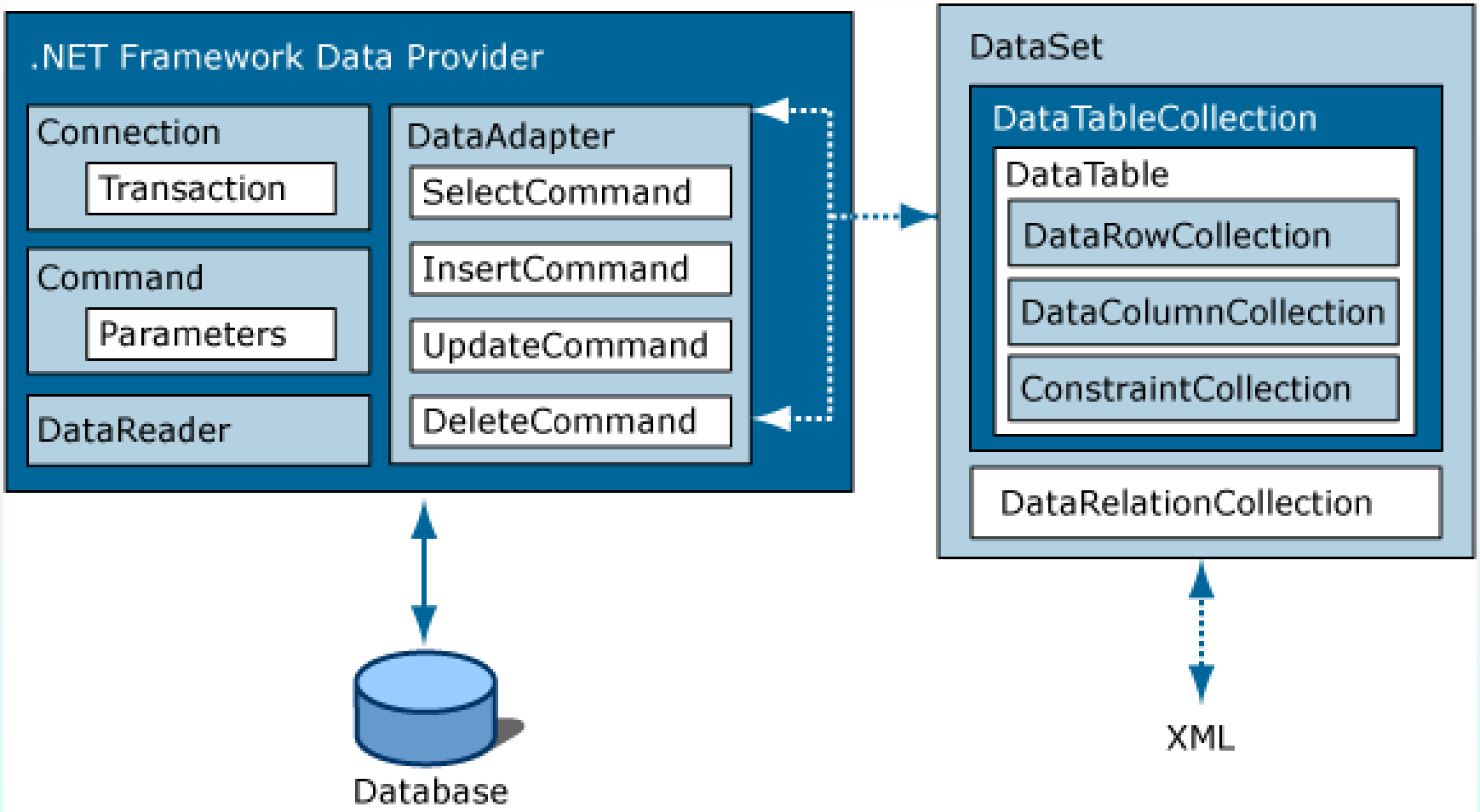
第 7 章

访问Web数据库

目　　录

7.1 ADO.NET体系结构

ADO.NET核心组件结构



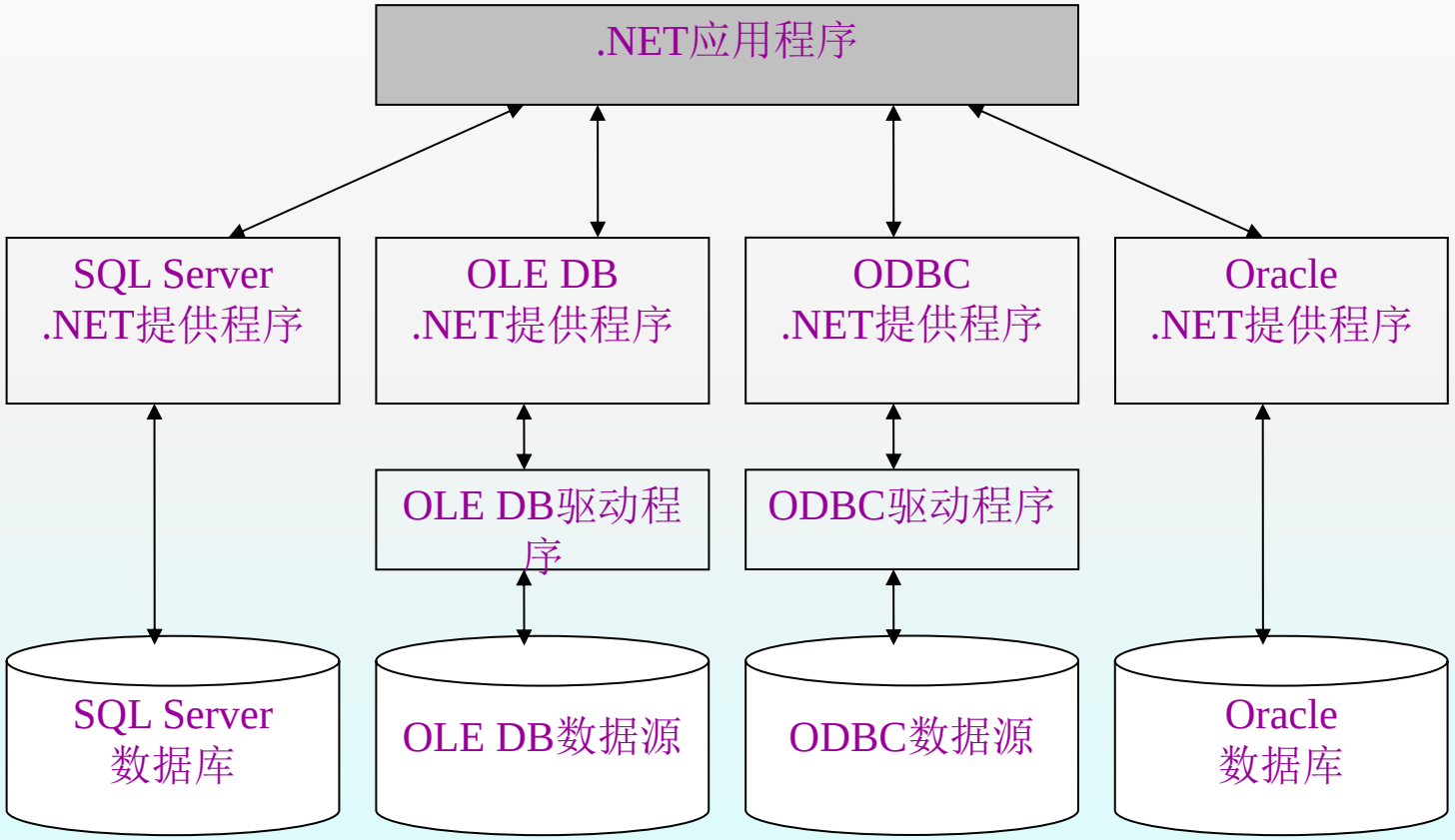
7.1.1 ADO.NET数据提供程序

- 什么是ADO.NET数据提供程序

是应用程序与数据源之间的一座桥梁，包含一组用于访问特定数据库，执行SQL语句并获取值的.NET类。

- 核心类及其作用
  - **DbConnection类：建立与数据源的连接**
  - **DbCommand类：执行SQL命令及存储过程**
  - **DbDataReader类：提供对Select语句查询结果的快速、只读、只进的访问方法**
  - **DataAdapter类：数据源与DataSet之间的桥梁**

数据提供程序模型结构

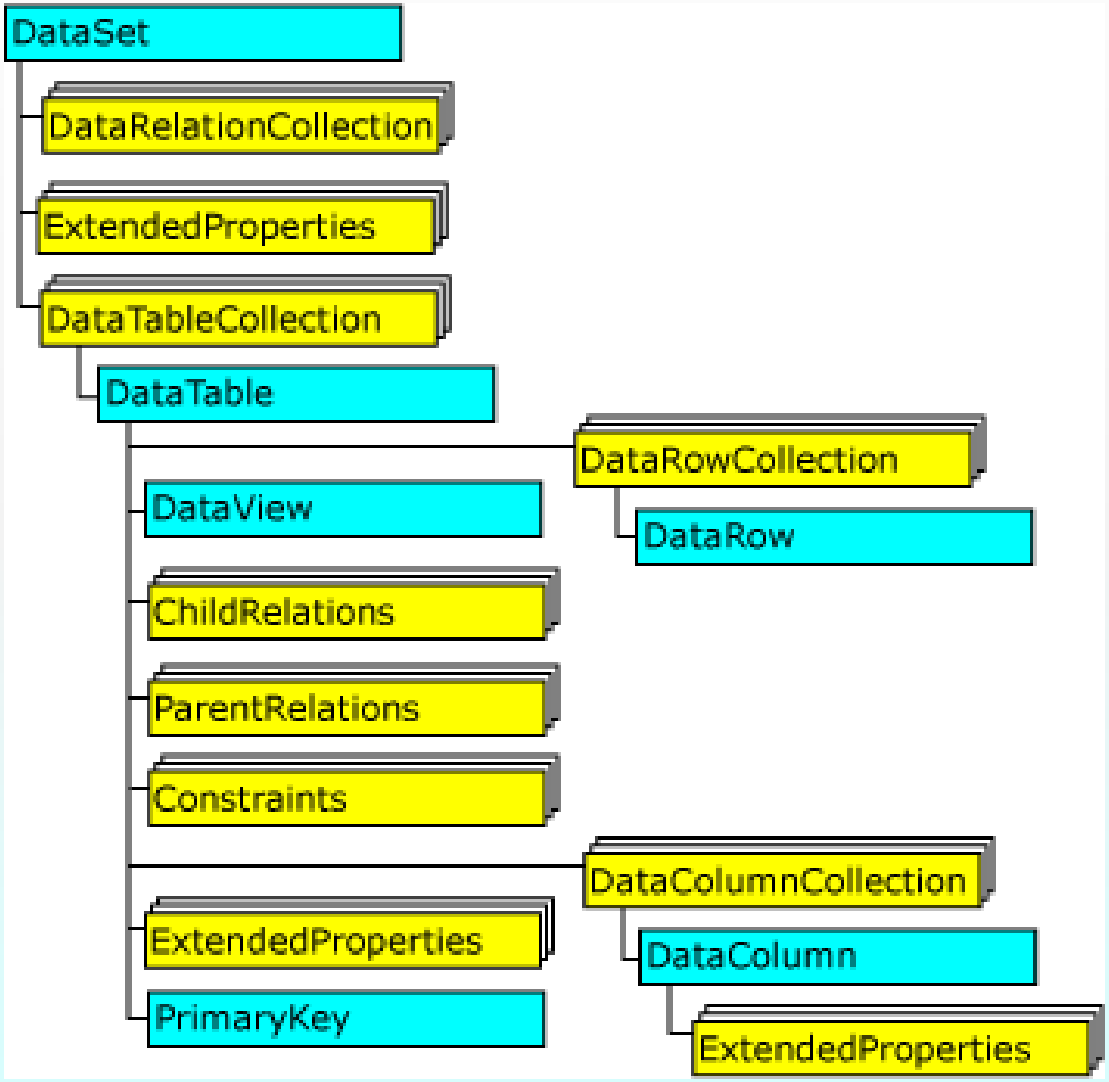

.NET应用程序
SQL Server
.NET提供程序
OLE DB
.NET提供程序
ODBC
.NET提供程序
Oracle
.NET提供程序
OLE DB驱动程序
ODBC驱动程序
SQL Server
数据库
OLE DB数据源
ODBC数据源
Oracle
数据库

7.1.2 ADO.NET数据集

- 什么是DataSet

是数据驻留在内存中的表示形式。

不管数据源是什么，它都可提供一致的关系编程模型。



➢ DataSet中的核心对象

- DataTableCollection：数据表的集合
- DataRelationCollection：关联的集合
- DataTable：数据表，或称内存表
- DataView：数据视图，DataTable中数据的不同表现形式
- DataRow：数据行，关系表中的一行数据
- DataColumn：数据列，关系中的一个属性
- PrimaryKey：主键

7.1.3 ADO.NET类的组织

命名空间	描述
System.Data	包含关键的容器类，这些类为关系、表、视图、数据集、行和约束建立模型，另外还包含了基于连接的数据对象要实现的关键接口的定义
System.Data.Common	包含由各种 .NET Framework 数据提供程序共享的抽象类，具体的数据提供程序继承这些类，创建自己的版本。
System.Data.OleDb	包含用于连接OLE DB数据提供程序的类，主要有OleDbCommand、OleDbConnection、OleDbDataReader及OleDbDataAdapter等。
System.Data.SqlClient	包含用于连接微软SQL Server数据库所需的类，包括SqlDbCommand、SqlDbConnection、SqlDbDataReader及SqlDbDataAdapter等，这些类使用经过优化的SQL Server的TDS接口。
System.Data.OracleClient	包含连接Oracle所需的类，包括OracleCommand、OracleConnection、OracleDataReader及OracleDataAdapter等，这些类使用经过优化的Oracle的OCI接口。
System.Data.Odbc	包含连接大部分ODBC驱动所需的类，包括OdbcCommand、OdbcConnection、OdbcDataReader及OdbcDataAdapter等

7.2 使用基于连接的对象访问数据库

7.2.1 访问数据库的一般方法

访问数据库的一般步骤

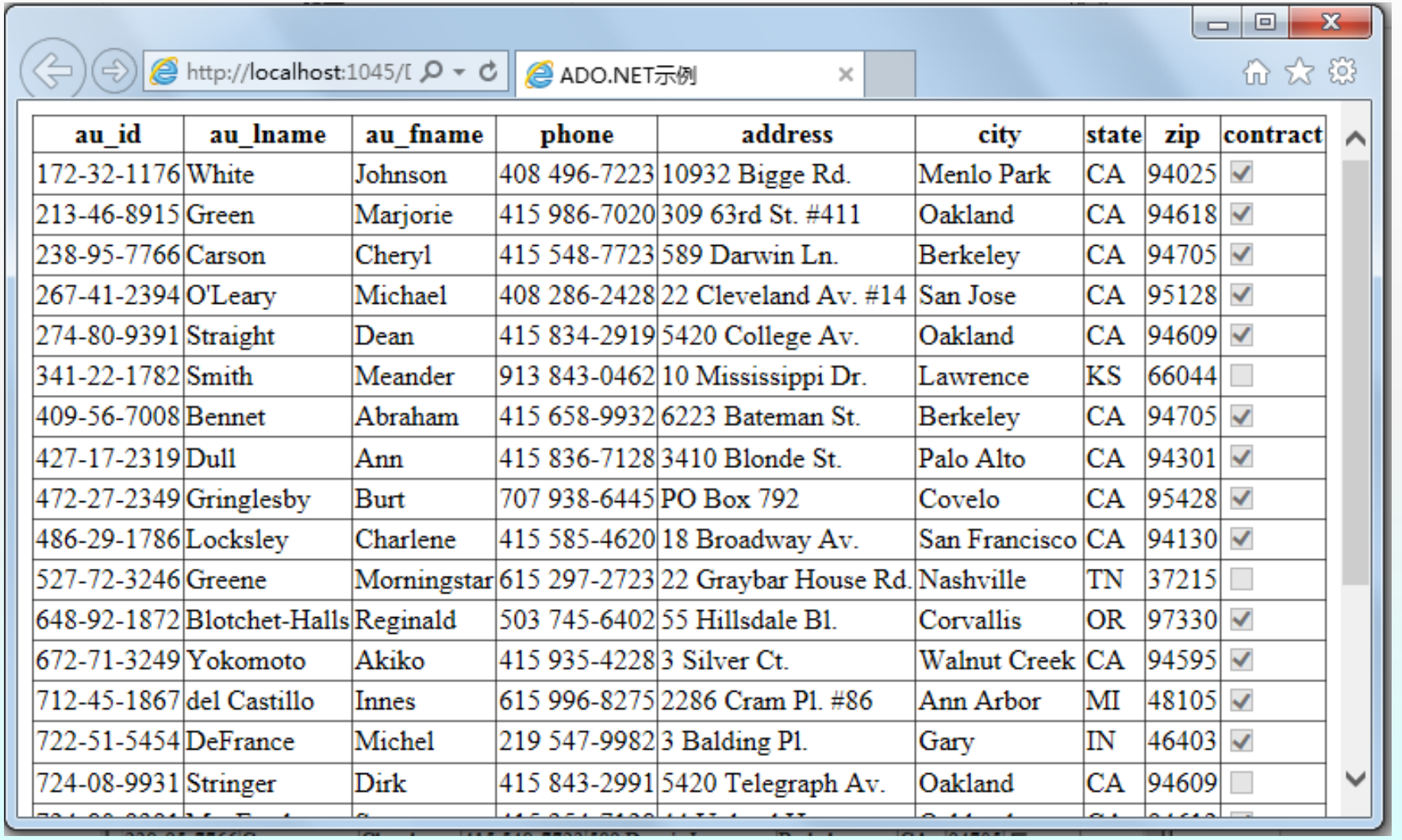
查询Authors表中的所有记录并在页面上列表显示(页面)

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="exam9_1.aspx.cs" Inherits="dbprj.exam9_1" %>
<html>
<head runat="server">
    <title>ADO.NET示例</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:GridView ID="GridView1" runat="server"></asp:GridView>
    </form>
</body>
</html>
```

查询Authors表中的所有记录并在页面上列表显示(后台)

```
public partial class exam7_1 : System.Web.UI.Page  {
      protected void Page_Load(object sender, EventArgs e)   {
         string connstr = @"Data Source=.\sqlexpress;Initial
                                   Catalog=pubs; Integrated Security=True";
         SqlConnection conn = new SqlConnection(connstr);
         SqlCommand cmd = new SqlCommand();
         cmd.Connection = conn;
         cmd.CommandText = "select * from authors";
         try  {
            conn.Open();
            SqlDataReader dr = cmd.ExecuteReader();
            GridView1.DataSource = dr;
            GridView1.DataBind();
            dr.Close();
         }
         catch{ }
         finally {   conn.Close();       }
      }
   }
```

运行结果



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	☑
213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	☑
238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	☑
267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	☑
274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	☑
341-22-1782	Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS	66044	☐
409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705	☑
427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	☑
472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	95428	☑
486-29-1786	Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA	94130	☑
527-72-3246	Greene	Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	37215	☐
648-92-1872	Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR	97330	☑
672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	94595	☑
712-45-1867	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI	48105	☑
722-51-5454	DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN	46403	☑
724-08-9931	Stringer	Dirk	415 843-2991	5420 Telegraph Av.	Oakland	CA	94609	☐

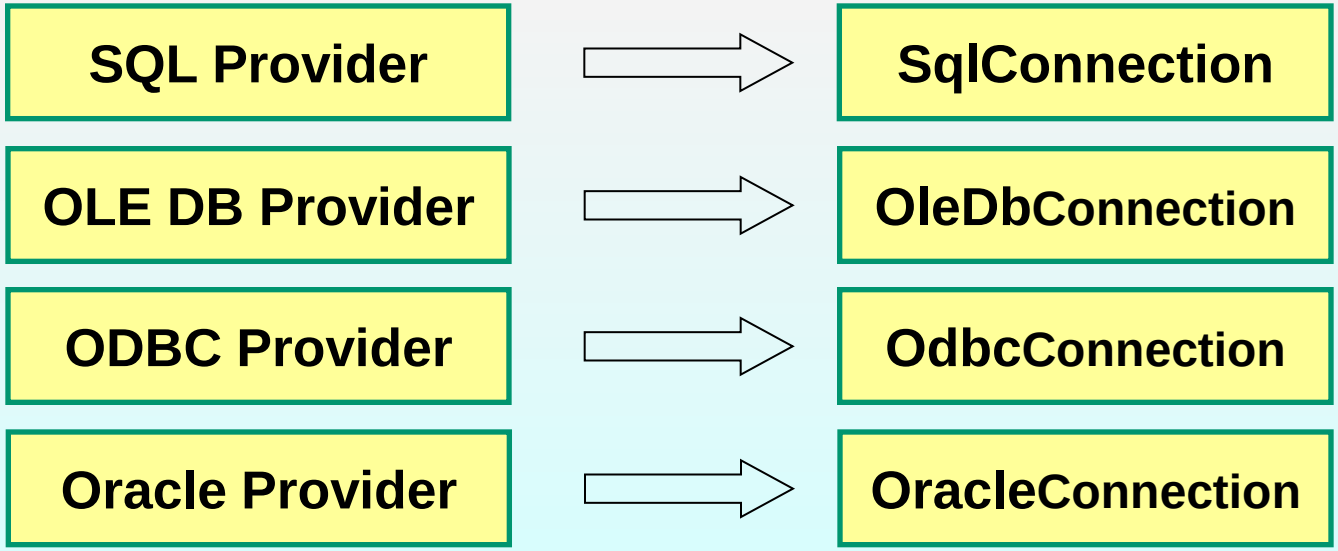
程序说明：

- 由于要操作SQL Server数据库，所以使用了SQL Server .NET提供程序。也可以使用OLE DB .NET提供程序等。
- 本例使用SqlConnection、SqlCommand及SqlDataReader等类，位于SqlClient命名空间下，需要事先导入。
- 在创建连接对象时，需要数据库的位置以及登录验证信息等，这些信息通过连接字符串指定。
- 在连接对象上调用open方法打开连接，使用完成后调用其close方法关闭连接
- 使用SqlCommand对象向数据库发送Select语句，本例使用SqlDataReader对象操作返回的结果集。
- 本例使用富数据控件GridView显示查询结果

7.2.2 使用Connection对象

➢ Connection对象的基本说明

- ✓ ADO.NET使用Connection对象建立到数据源的连接
- ✓ 针对不同的数据提供程序要使用不同的连接对象
- ✓ 使用连接字符串指定连接参数
- ✓ 连接对象使用完成后一定要及时释放
- ✓ 为提供效率，通常会使用连接池来缓存和共享连接



➢ 连接字符串

服务器位置

名称：通常使用DataSource、Server、Address等

值：通常填服务器的IP地址或主机名，或localhost

数据库名称

名称：通常使用Initial Catalog，或者DataBase

值：填连接到的数据库实例名

身份验证方式

集成身份验证：Integrated Security=true

SQL Server验证：使用用户名和口令验证

使用集成身份验证的连接字符串：

```
Server=localhost; Initial Catalog=Pubs; Integrated Security=true
```

使用SQL Server身份验证的连接字符串：

```
Server=localhost; Initial Catalog=Pubs; UID=dbuser; Pwd=dbpwd
```

➢ 连接到数据库

使用SqlClient连接SQL Server数据库：

```
using System.Data.SqlClient;          // 导入SqlClient命名空间
……
string constr = “Data Source=(local); Initial Catalog=Northwind;
                                              Integrated Security=true”
SqlConnection conn = new SqlConnection( constr );    // 建立连接对象
```

使用OracleClient 连接 Oracle数据库：

```
using System.Data.OracleClient;      // 导入 OracleClient 命名空间
……
string constr = string constr = @" Server=orcl; UID=scott; pwd=tiger"
OracleConnection conn = new OracleConnection(connstr);
```

示例 使用OLE DB连接ACCESS数据库：

```
using System.Data.OleDb;      // 导入OleDb命名空间
……
string constr = “Provider=Microsoft.Jet.OLEDB.4.0;
                        Data Source=D:\dbprj\pubs.mdb”
OleDbConnection conn = new OleDbConnection(connstr);
```

示例 使用OLE DB连接SQL Server数据库：

```
using System.Data.OleDb;
……
string constr =  “Provider=SQLOLEDB; Data Source=.\sqlexpress;
                         Integrated Security=SSPI;Initial Catalog=pubs”
OleDbConnection conn = new OleDbConnection(connstr);
```

- 打开连接

  conn.Open( );

- 关闭连接

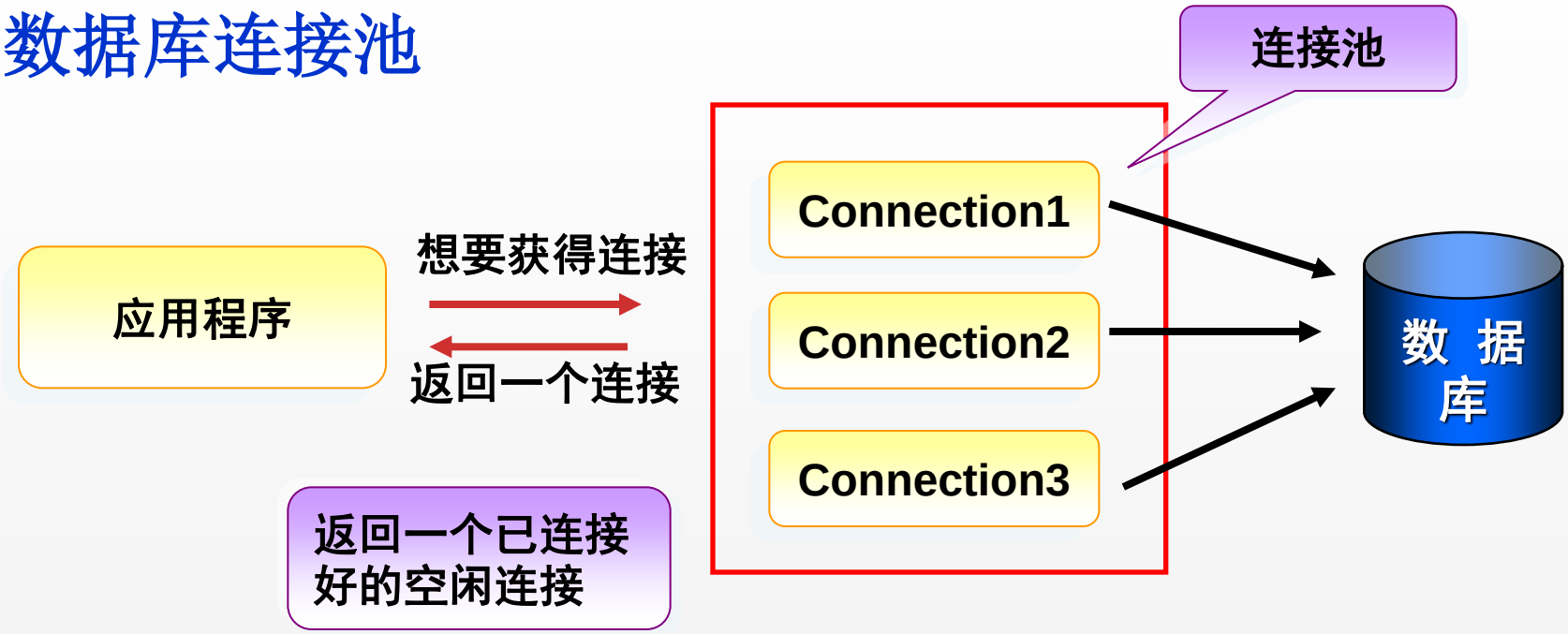
  conn.Close( );

  conn.Dispose( );

➢ 处理异常

```
SqlConnection conn = new SqlConnection(connstr);
try{
    conn.Open();      // 打开到数据库的连接
      ……            // 操作数据库
}
catch (Exception ex) {
      ……            // 处理异常
}
finally {
      conn.Dispose();   // 释放连接
}
```

➢ 数据库连接池



自动创建

ADO.NET通常根据连接字符串来自动创建连接池，多个连接若连接字符串相同则使用同一个连接池，若连接字符串不同，则创建一个新的连接池。

透明访问

连接池对开发者完全透明，数据访问代码不需要做任何更改。调用**Open()**方法实际上是从池中取出一个连接；调用**Close()**方法实际上是将连接归还到池中。

➢ 保存连接字符串

保存到Web.config文件的<connectionStrings>节

```
<connectionStrings>
    <add name="connstr" connectionString="Data Source=.\sqlexpress;
                        Initial Catalog=pubs; Integrated Security=True"/>
</connectionStrings>
```

保存到Web.config文件的<appSettings>节

```
<appSettings>
   <add key="connstr" value="Data Source=.\sqlexpress;Catalog=Pubs;
                                    Integrated Security=True"/>
</appSettings>
```

➢ 获取连接字符串

示例 从<connectionStrings>节获取

```
using System.Configuration;
string connstr =
ConfigurationManager.ConnectionStrings["connstr"].ConnectionString;
```

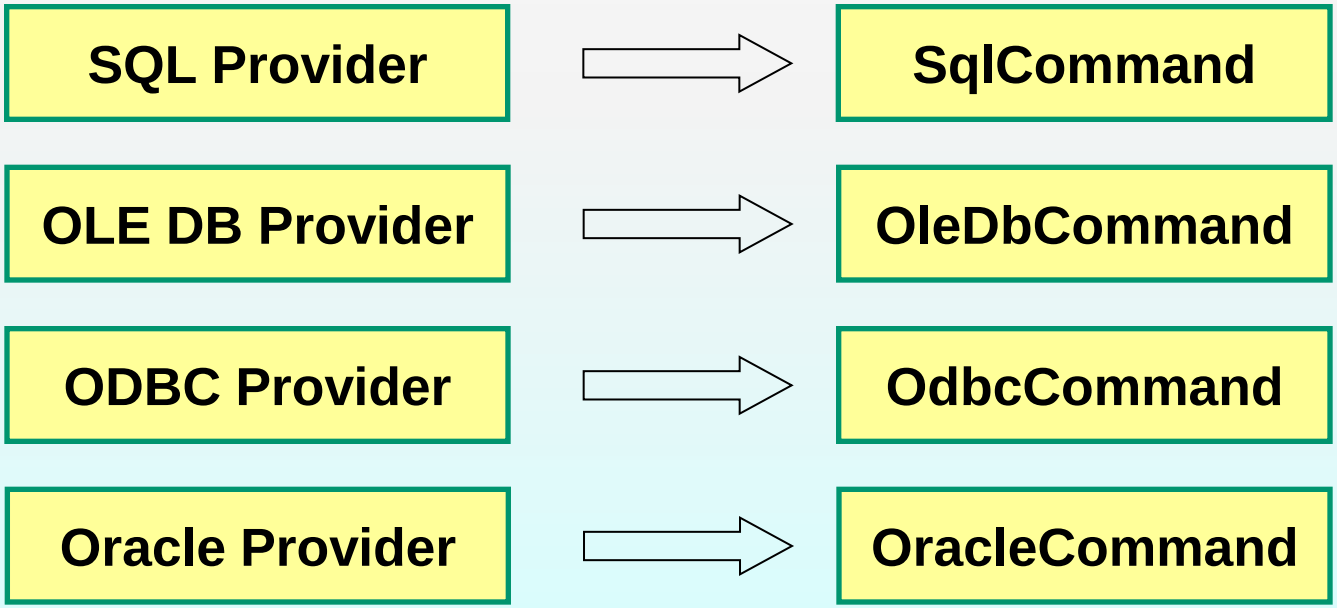
示例 从<AppSettings>节获取

```
using System.Configuration;
string connstr = ConfigurationManager. AppSettings["connstr"];
```

7.2.3 使用Command对象

➢ Command对象概述

- ✓ 使用Command对象来执行SQL命令并返回结果
- ✓ 针对不同的数据提供程序要使用不同的Command对象



➢ Command对象的常用属性

- ✓ CommandText：获取或设置针对数据源运行的文本命令。通常指定为一条SQL语句，或者一个存储过程的名称。
- ✓ CommandType：指定CommandText的类型，其取值为System.Data.CommandType枚举。
- ✓ Connection：获取或设置命令对象依赖的数据库连接。
- ✓ Parameters：获取命令参数的集合，当执行参数化查询时该属性非常重要。

➢ Command对象的常用方法

ExecuteReader	执行查询命令或存储过程 返回由DataReader封装的记录集
ExecuteNonQuery	执行一个非查询的SQL语句或存储过程 返回受影响的记录行数
ExecuteScalar	执行一个查询命令或存储过程 返回一个标量值（即单值）

➢ 使用ExecuteReader( )方法

例 在NorthWind数据库中，根据顾客号查询该顾客的订单信息

```
protected void btnqry_Click(object sender, EventArgs e)  {
    string connstr =
        ConfigurationManager.ConnectionStrings["connstr"].ConnectionString;
    string sql = "select orderid, orderdate, shipaddress from orders "
        + " where customerid = ' " + tbxuid.Text + "'";
    using (SqlConnection conn = new SqlConnection(connstr))  {
        SqlCommand cmd = conn.CreateCommand();
        cmd.CommandText = sql;
        conn.Open();
        SqlDataReader dr = cmd.ExecuteReader();
        GridView1.DataSource = dr;
        GridView1.DataBind();
        dr.Close();
    }
}
```

例 使用StringBuilder对象辅助拼接SQL

```
StringBuilder sb = new StringBuilder();
sb.Append ("select orderid, orderdate, shipaddress from orders
                                                    where customerid = '");
sb.Append(tbxuid.Text);
sb.Append("'");
cmd.CommandText = sb.ToString();
```

例 使用参数化查询

```
cmd.CommandText = "select orderid, orderdate, shipaddress from orders
                                                    where customerid = @custid";
SqlParameter parameter = new SqlParameter( "@custid",
                                                    SqlDbType.VarChar);
parameter.Value = tbxuid.Text;
cmd.Parameters.Add(parameter);
SqlDataReader dr = cmd.ExecuteReader();
```

例 查询指定用户指定时间段内的所有订单

```
cmd.CommandText = "select orderid, orderdate, shipaddress from orders where "
            + " customerid = @custid and orderdate between @fromdate and @todate";
cmd.Parameters.Add ("@custid", SqlDbType.VarChar);
cmd.Parameters.Add ("@fromdate", SqlDbType.DateTime);
cmd.Parameters.Add ("@todate", SqlDbType.DateTime);
cmd.Parameters ["@custid"]. Value = tbxuid.Text;
cmd.Parameters ["@startdate"]. Value = "1996-01-01";
cmd.Parameters ["@enddate"]. Value = "1997-01-01";
```

说明：参数占位符的语法与数据提供程序的关系

数据提供程序	参数占位符语法
System.Data.SqlClient	以 *@parametername* 格式使用命名参数
System.Data.OracleClient	以 *:parmname*（或*parmname*）格式使用命名参数
System.Data.OleDb	使用由问号 (?) 表示的位置参数标记
System.Data.Odbc	使用由问号 (?) 表示的位置参数标记

例 使用OleDb方式连接SQL Server数据库，查询顾客指定时间段的订单

```
using (OleDbConnection conn = new OleDbConnection(connstr)) {
    OleDbCommand cmd = conn.CreateCommand();
    cmd.CommandText = "select orderid, orderdate, shipaddress from orders "
                          + " where customerid = ? and orderdate between ? and ?";
    cmd.Parameters.AddWithValue ("@custid", tbxuid.Text);
    cmd.Parameters.AddWithValue ("@fromdate", "1996-01-01");
    cmd.Parameters.AddWithValue ("@todate", "1997-01-01");
    conn.Open();
    OleDbDataReader dr = cmd.ExecuteReader();
    GridView1.DataSource = dr;
    GridView1.DataBind();
    dr.Close();
}
```

➢ 使用ExecuteScalar( )方法

例 查询Employees表，统计所有员工的数量

```
using (SqlConnection conn = new SqlConnection(connstr))  {
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "select count(*) from employees";
    conn.Open();
    int num = (int)cmd.ExecuteScalar();
    lblmsg.Text = string.Format("员工总数：  <b>{0}</b>", num);
}
```

➢ 使用ExecuteNonQuery( )方法

例 对Employees表执行增删改操作

```
using (SqlConnection conn = new SqlConnection(connstr))  {
    conn.Open();
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "Insert into employees(LastName, FirstName) "
                                                + " values ('Tom', 'Cat')";
    int num =  cmd.ExecuteNonQuery();
    lblmsg.Text += string.Format("<br /> 共插入记录：<b> {0} 条</b>",num);
    cmd.CommandText = "Update employees set LastName = 'Jerry' "
                                                  + " where EmployeeID = 9";
    num = cmd.ExecuteNonQuery();
    lblmsg.Text += string.Format("<br /> 共修改记录：<b> {0} 条</b>",num);
    cmd.CommandText = "Delete from employees where EmployeeID > 9";
    num = cmd.ExecuteNonQuery();
    lblmsg.Text += string.Format("<br /> 共删除记录：<b> {0} 条</b>",num);
}
```

➢ 执行存储过程

提高性能

存储过程在数据库中进行了编译和优化，执行效率提高了很多

使用存储过程减少与数据库交互的次数，大幅提高效率

简化设计

对于非常复杂的业务处理（例如销售统计），若将处理逻辑封装在存储过程中，则应用程序中只需简单的调用存储过程就可以完成所有工作，有效的降低了程序的复杂度。

易于维护

将复杂的数据处理逻辑从应用程序中分离出来，你可以对存储过程进行优化，只要接口不变，就不需要更改或重新编译应用程序

易于分工

大型项目中明确的人员分工异常重要，使用存储过程，可以将复杂的数据处理逻辑分工给数据库开发人员，任务更加明确。

例 在NorthWind数据库中查询指定类别下所有产品的销售量

```
string connstr =
            ConfigurationManager.ConnectionStrings ["nwconnstr"]. ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))  {
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "SalesByCategory";
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameter parameter = new SqlParameter ("@CategoryName",
                                                SqlDbType.VarChar);
    parameter.Value = tbxcatname.Text;
    cmd.Parameters.Add(parameter);
    conn.Open();
    SqlDataReader dr = cmd.ExecuteReader();
    GridView1.DataSource = dr;
    GridView1.DataBind();
    dr.Close();
}
```

例 向Employee表中插入一条记录，并返回该员工的ID号

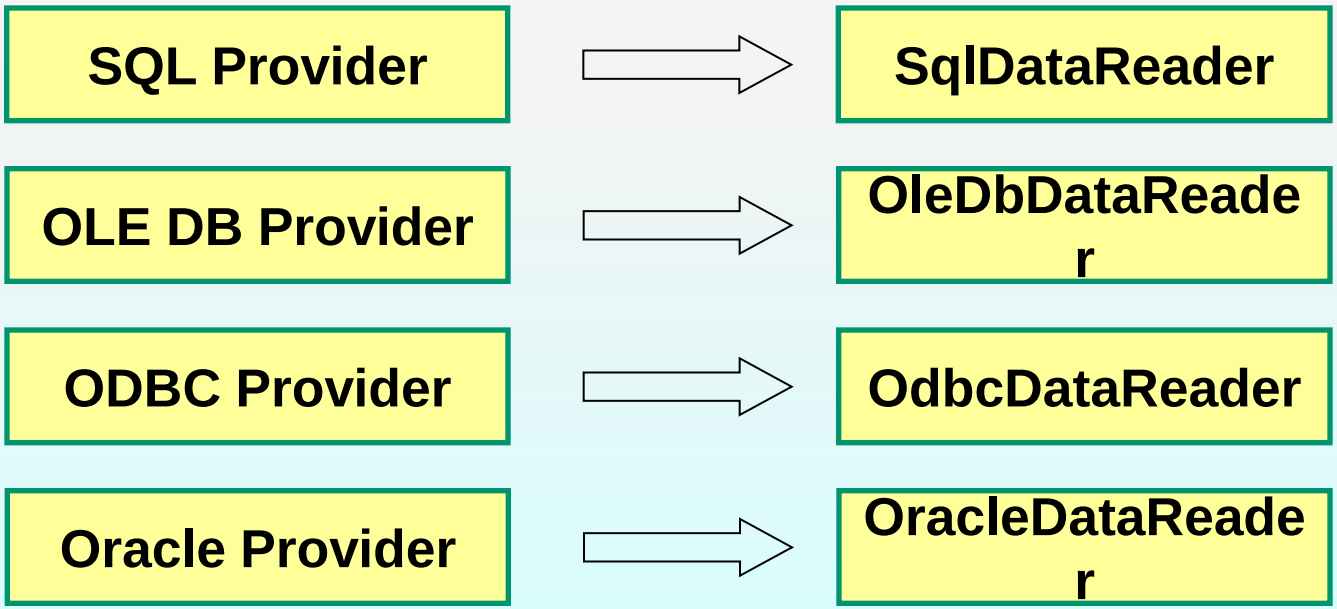
```
CREATE PROCEDURE InsertEmployee
        @LastName varchar(20),
        @FirstName varchar(10),
        @EmployeeID int output
AS
   Insert into Employees (LastName, FirstName) Values (@LastName, @FirstName);
   set @EmployeeID = @@IDENTITY;
```

```
string connstr =
            ConfigurationManager.ConnectionStrings ["nwconnstr"]. ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))
{
   SqlCommand cmd = conn.CreateCommand();
   cmd.CommandText = "InsertEmployee";
   cmd.CommandType = CommandType.StoredProcedure;
   cmd.Parameters.Add ("@LastName", SqlDbType.VarChar, 20);
   cmd.Parameters.Add ("@FirstName", SqlDbType.VarChar, 10);
   cmd.Parameters ["@LastName"]. Value = tbxlname.Text;
   cmd.Parameters ["@FirstName"]. Value = tbxfname.Text;
   cmd.Parameters.Add ("@EmployeeID", SqlDbType.Int);
   cmd.Parameters ["@EmployeeID"]. Direction = ParameterDirection.Output;
   conn.Open();
   int n = cmd.ExecuteNonQuery();
   lblmsg.Text += string.Format ("Inserted {0} Records <br />", n);
   int empid = (int) cmd.Parameters ["@EmployeeID"]. Value;
   lblmsg.Text += "New ID : " + empid.ToString();
}
```

7.2.4 使用DataReader对象

➢ DataReader 概述

- ✓ DataReader以只读、只进的方式，每次读取一条记录进行处理，占用内存资源极小，操作效率极高。
- ✓ 针对不同的数据提供程序要使用不同的DataReader对象



➢ DataReader对象常用属性和方法

成员名称	成员描述
HasRows 属性	指示该DataReader中是否包含数据
FieldCount 属性	获取当前行中的列数
Read( )方法	将游标移动到记录集的下一行
GetValue ( ) 方法	获取当前行中指定序号的字段的值，系统将根据数据源中该字段的数据类型匹配一个最相近的.NET数据类型返回
GetXxx ( ) 方法	获取当前行中指定序号的字段的值，但明确指定了返回值的类型，所以返回类型与方法名称中指定的类型一致。 例如：GetInt32( )、GetChar( )、GetDateTime( )等
Close ( ) 方法	关闭DataReader对象

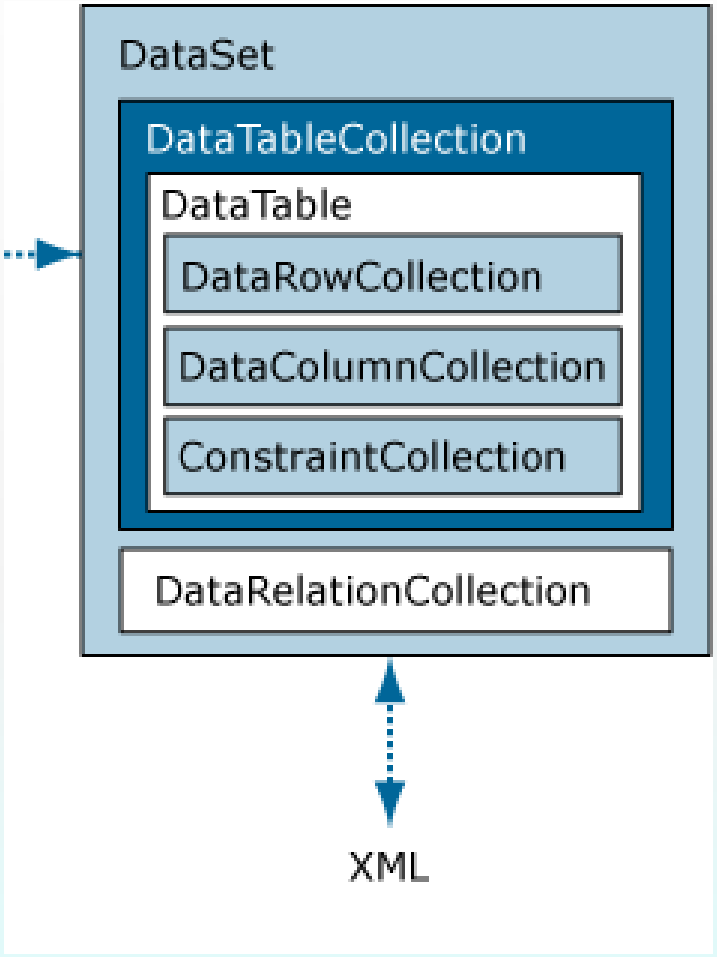
例 查询并显示Categories表中所有类别的ID和名称

```
string connstr =
            ConfigurationManager.ConnectionStrings["nwconnstr"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))  {
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "Select CategoryID, CategoryName from Categories";
    conn.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    StringBuilder sb = new StringBuilder("");
    sb.Append("<table><tr><td>CategoryID</td><td>CategoryName</td></tr>");
    while (reader.Read())  {
       sb.Append("<tr><td>");   sb.Append(reader[0]);   sb.Append("</td><td>");
       sb.Append( reader ["CategoryName"]);              sb.Append("</td></tr>");
    }
    sb.Append("</table>");
    reader.Close();
    lblmsg.Text = sb.ToString();
 }
```

7.3 使用基于内容的对象访问数据库

➢DataSet概述：

- DataSet架构使用非连接的特性，可以将批量数据读入内存，然后进行离线的处理，最后还可以将处理结果批量写回数据库中。
- 通常使用DataAdapter实现DataSet与数据库的交互
- DataSet中主要包含两种元素，一是表的集合，二是表间关系的集合。前者代表数据，后者代表约束。

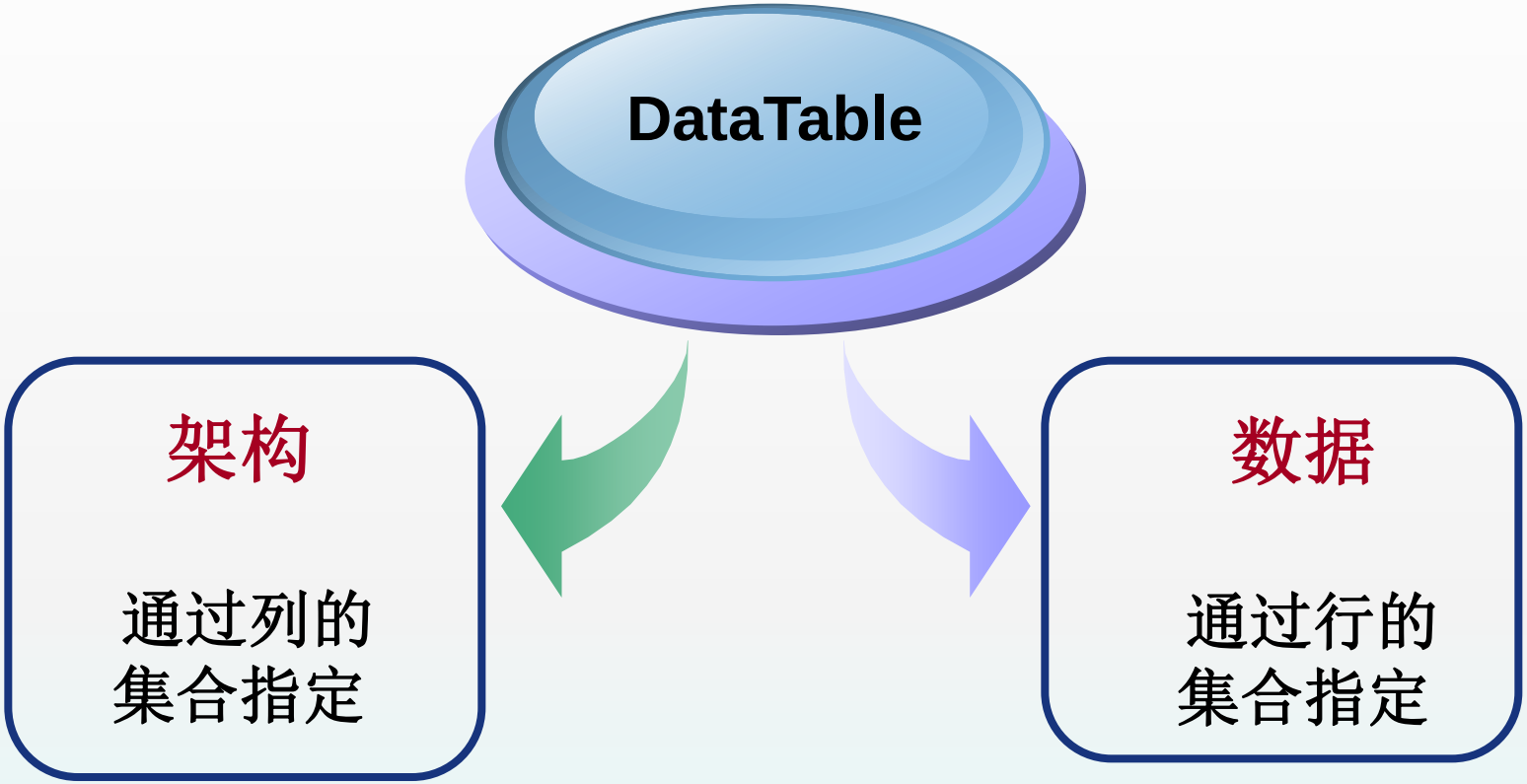


7.3.1 使用DataTable对象

➢ DataTable概述

- ✓ DataTable是DataSet架构的核心，代表内存中的表。
- ✓ 依靠它可以离线处理数据，前、后移动游标定位记录，快速检索记录，对记录进行排序，按条件过滤记录等。
- ✓ DataTable既可以包含在DataSet中，也可以游离于DataSet之外而独立存在

➢ 创建 DataTable对象



例 以编程方式创建DataTable架构

```
DataTable dt = new DataTable();
DataColumn col = dt.Columns.Add ("EmployeeID", typeof(Int32));
col.Unique = true;
dt.Columns.Add ("LastName", typeof(String));
dt.Columns.Add ("FirstName", typeof(String));
```

例 向DataTable中添加数据

```
DataRow row = dt.NewRow();      // 创建一个新行
// 为行中的各列赋值
row[0] = 1;
Row ["LastName"] = "Tom";
Row ["FirstName"] = "Cat";
dt.Rows.Add (row);                    // 将该行数据添加到表格中
```

例 使用load( )方法装载数据

```
DataTable dt = new DataTable();
cmd.CommandText = "Select EmployeeID, LastName, FirstName from Employees";
SqlDataReader reader = cmd.ExecuteReader();
dt.Load(reader);
```

说明：

- 通常都是从数据库中检索数据并填充到DataTable中，最直接的方式是调用DataTable对象的Load( )方法，并传入一个DataReader对象，这样系统会自动从DataReader中不断获取数据并装载到DataTable中。
- 装载数据时，由于可以连接到数据库，自动获取表的架构信息，所以不需要专门编写代码定义数据表结构，只要创建空的DataTable对象，即可开始装载或填充数据。

➢ 遍历 DataTable数据

- ✓ DataTable的Rows属性返回行的集合，其中每个元素就是一条记录。
- ✓ 通常使用Foreach循环遍历行集中的每一个DataRow。
- ✓ DataRow是字段值的容器，可以指定字段序号或者名称来访问各字段值

例 遍历并显示DataTable中的数据

```
StringBuilder sb = new StringBuilder("<table>");
sb.Append("<tr><td>FirstName</td><td>LastName</td></tr>");
foreach (DataRow row in dt.Rows)
{
    sb.Append("<tr><td>");
    sb.Append (row ["FirstName"]. ToString());
    sb.Append("</td><td>");
    sb.Append (row ["LastName"]. ToString());
    sb.Append("</td></tr>");
}
sb.Append("</table>");
lblmsg.Text = sb.ToString();
```

➢ 检索 DataTable数据

- ✓ Select( )方法：返回满足条件的行集。
- ✓ Find( )方法：按主键检索并返回特定行。

例 使用Select方法在内存表中检索数据

```
DataRow[ ] matchrows = dt.Select ("CategoryID = 2");
StringBuilder sb = new StringBuilder("<ul>");
foreach (DataRow row in matchrows)
{
    sb.Append("<li>");
    sb.Append (row ["ProductName"]. ToString());
    sb.Append("</li>");
}
sb.Append("</ul>");
lblmsg.Text = sb.ToString();
```

例 使用Find方法在内存表中检索数据

```
// 为数据表定义主键列
DataColumn[ ] columns = new DataColumn[1];
columns[0] = dt.Columns ["ProductID"];
dt.PrimaryKey = columns;
// 根据主键进行检索（查询商品号为5的行）
DataRow findrow = dt.Rows.Find(5);
// 输出检索结果
if (findrow != null)   lblmsg.Text = findrow ["ProductName"].ToString();
```

7.3.2 使用DataView对象

➢ DataView概述

- ✓ DataView为DataTable对象定义数据视图。
- ✓ DataView使DataTable能够支持自定义过滤和数据排序。
- ✓ 每个DataTable都有一个默认的DataView与之关联，使用DefaultView属性可以引用该DataView
- ✓ 可以在同一个表上创建多个不同的DataView对象

➢ 利用DataView进行数据排序：Sort属性

例 对Products表中的数据进行排序显示

```
DataTable dt = new DataTable();
string connstr =
        ConfigurationManager.ConnectionStrings ["nwconnstr"]. ConnectionString;
using (SqlConnection conn = new SqlConnection(connstr))  {
   SqlCommand cmd = conn.CreateCommand();
   cmd.CommandText = "Select ProductID, ProductName, UnitPrice from Products ";
   conn.Open();
   SqlDataReader reader = cmd.ExecuteReader();
   dt.Load(reader);
}
GridView1.DataSource = dt.DefaultView;
GridView1.DataBind();
DataView dv = new DataView(dt);
dv.Sort = "UnitPrice";
GridView2.DataSource = dv;
GridView2.DataBind();
```

➢ 利用DataView进行数据过滤：RowFilter属性

例 从Products表中选出满足条件的数据并显示

```
dt.DefaultView.RowFilter = "UnitPrice > 40";
GridView1.DataSource = dt.DefaultView;
GridView1.DataBind();
DataView dv = new DataView(dt);
dv.RowFilter = "ProductName Like 'M%'";
GridView2.DataSource = dv;
GridView2.DataBind();
```

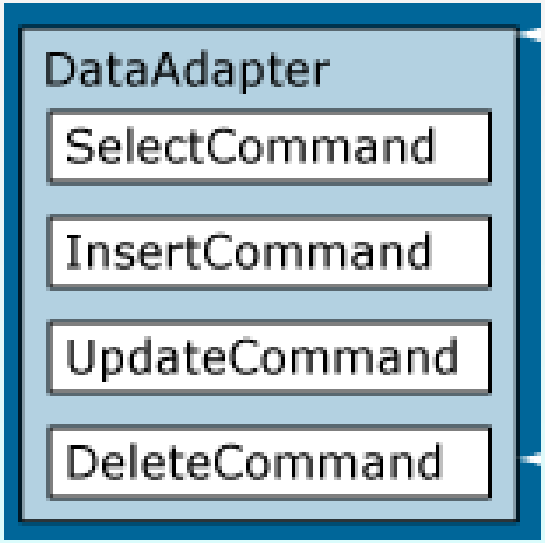
7.3.3 使用DataAdapter对象

➢ DataAdapter的用途

- ✓ 作为数据源与DataSet之间的桥梁，起着承上启下的作用
- ✓ 可以将数据源中的数据填充到DataSet
- ✓ 也可以将DataSet中所作的数据更改保存到数据源中

➢ DataAdapter的构成

- ✓ SelectCommand
- ✓ InsertCommand
- ✓ UpdateCommand
- ✓ DeleteCommand



➢ 使用DataAdapter填充数据

- ✓ 调用Fill( )方法
- ✓ 本质上是执行了SelectCommand对象中的命令

例 使用DataAdapter对象填充数据表

```
string connstr =
        ConfigurationManager.ConnectionStrings ["nwconnstr"]. ConnectionString;
  SqlConnection conn = new SqlConnection(connstr);
  string sql = "Select EmployeeID, LastName, FirstName from Employees";
  // 创建DataAdapter对象，需传入要执行的查询语句及连接对象
  SqlDataAdapter da = new SqlDataAdapter(sql, conn);
  // 创建DataSet对象，并向其中的employees表中填充数据
  DataSet ds = new DataSet();
  da.Fill(ds, "employees");
```

➢ 使用DataAdapter更新数据

- ✓ **调用Update( )方法**
- ✓ **先分析DataTable所作的更改，然后使用InsertCommand、UpdateCommand或DeleteCommand对象来处理更改。**

例 使用DataAdapter对象填充数据表

```
using (SqlConnection connection = new SqlConnection(connectionString))    {
   SqlDataAdapter dataAdpater = new SqlDataAdapter(
            "SELECT CategoryID, CategoryName FROM Categories", connection);
   dataAdpater.UpdateCommand = new SqlCommand(
         "UPDATE Categories SET CategoryName = @CategoryName "
         + "WHERE CategoryID = @CategoryID", connection);
      dataAdpater.UpdateCommand.Parameters.Add(
         "@CategoryName", SqlDbType.NVarChar, 15, "CategoryName");
      SqlParameter parameter = dataAdpater.UpdateCommand.Parameters.Add(
        "@CategoryID", SqlDbType.Int);
      DataTable categoryTable = new DataTable();
      dataAdpater.Fill(categoryTable);
      DataRow categoryRow = categoryTable.Rows[0];
      categoryRow ["CategoryName"] = "New Beverages";
      dataAdpater.Update(categoryTable);
   }
```

例 使用DbCommandBuilder对象辅助更新数据

```
SqlDataAdapter adapter = new SqlDataAdapter();
adapter.SelectCommand = new SqlCommand(sql, connection);
SqlCommandBuilder builder = new SqlCommandBuilder(adapter);
connection.Open();
DataSet dataSet = new DataSet();
adapter.Fill(dataSet, tableName);
// 这里可以写更新DataSet的代码
builder.GetUpdateCommand();          // 自动构造更新数据库的SQL
adapter.Update(dataSet, tableName);  // 将更改写回数据库
```

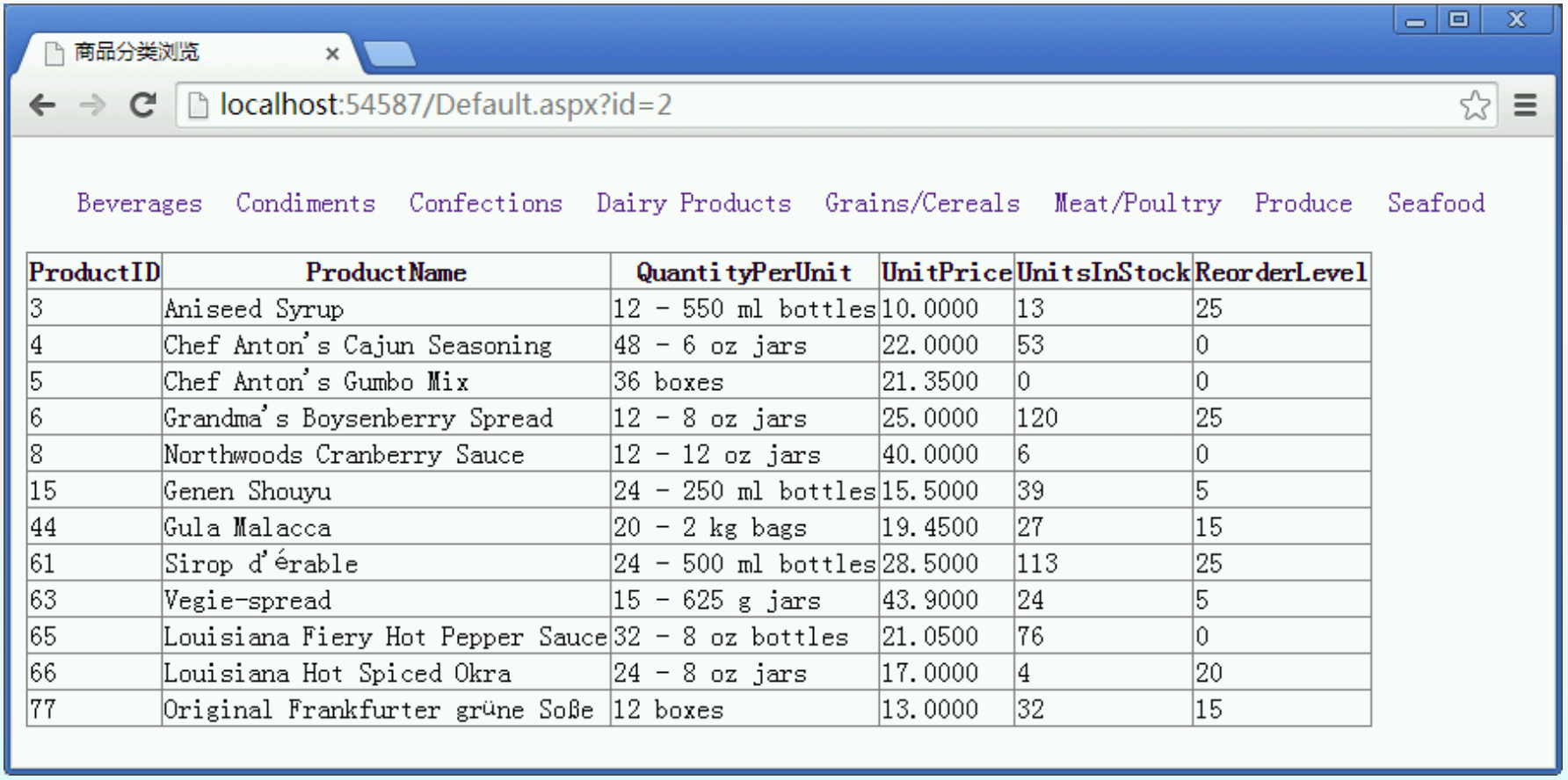
7.4 实例：分类浏览商品信息的页面

7.4.1 设计说明

以NorthWind数据库为例，商品类别信息存放在Category表中，商品信息存放在Product表中。

页面加载时，首先连接数据库，查询商品类别信息，然后遍历类别列表，根据每个列表项生成一个查询该类别商品的超链接，由这些超链接组成导航菜单；当点击某个超链接时，根据传入的类别号查询商品信息，并在GridView中显示出来。

7.4.2 程序实现



ProductID	ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	ReorderLevel
3	Aniseed Syrup	12 - 550 ml bottles	10.0000	13	25
4	Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.0000	53	0
5	Chef Anton's Gumbo Mix	36 boxes	21.3500	0	0
6	Grandma's Boysenberry Spread	12 - 8 oz jars	25.0000	120	25
8	Northwoods Cranberry Sauce	12 - 12 oz jars	40.0000	6	0
15	Genen Shouyu	24 - 250 ml bottles	15.5000	39	5
44	Gula Malacca	20 - 2 kg bags	19.4500	27	15
61	Sirop d'érable	24 - 500 ml bottles	28.5000	113	25
63	Vegie-spread	15 - 625 g jars	43.9000	24	5
65	Louisiana Fiery Hot Pepper Sauce	32 - 8 oz bottles	21.0500	76	0
66	Louisiana Hot Spiced Okra	24 - 8 oz jars	17.0000	4	20
77	Original Frankfurter grüne Soße	12 boxes	13.0000	32	15

作业5

1、访问SQL Server数据库时需要用到哪些对象，各自有何作用？

2、数据库连接字符串有哪几个部分组成？如何用web.config文件保存？如何使用web.config文件中的数据库连接字符串？

3、对数据表执行添加、修改、查询和删除操作时，分别使用Command对象的什么方法？举例说明。