第5章 客户关系管理系统数据库实现

5.1 项目导引——系统主要数据操作

经过努力，小陈已经搭建了和项目一模一样的用户界面，而且完成了一些简单的逻辑功能。虽然小陈很高兴，他距离成功又近了一步了，但是他乐不起来，因为他知道对于系统实现来说最困难的阶段马上就要到了。那么最困难的阶段是什么？又有多困难呢？如图 5-1 所示。

图 5-1 情节 5

5.2 项目分析

任何管理系统的应用都离不开数据的添加、修改、删除和查询操作。只有实现了这些功能，系统才真正意义上实现了数据的管理。下面具体分析一下各个模块的主要数据库操作。

1.登录模块

登录模块主要是通过验证实现用户的登录。由于用户的个人信息(用户名称和用户密码)是保存在数据库里面的，所以需要研究的主要问题是：如何通过数据库操作把要查询的用户的个人信息提取出来，然后与用户在控件里面输入的信息进行比较，根据比较的结果不同，实现不同的处理过程。

2.客户资料模块

客户资料模块主要是实现客户信息的添加、修改、查询和删除功能。由于客户的信息是以

表格的形式储存在数据库里的，所以要研究的主要问题是：如何通过数据访问技术实现对客户信息的添加、修改、查询和删除功能。

考虑到公司的需要，对公司来说这个软件的查询功能是比较重要的，因为软件的查询功能可以直接提高员工的工作效率，所以需要研究如何实现一些复杂的查询功能，例如模糊查询和多条件查询等查询功能。

为了提高软件的实用性，还需要研究数据绑定技术，提高软件的操作效率。

3.资料分析模块

资料分析模块主要是分析数据和数据之间的关系。对于公司来说，客户信息是最主要的数据，但是公司往往不需要对客户执行统一操作，往往是对某个特殊范围内的客户执行相对应操作，这就需要统计出数据的特点。为了更容易地理解显示数据特点，我们不仅需要研究如何以表格的形式显示数据特点，还要以图形（柱状图和饼状图）的方式显示数据特点。

4.用户资料模块

用户资料模块主要是实现系统用户的管理。考虑到公司的实际情况，只有少数人可以实现数据的修改和查处功能，其他的用户只能浏览和查询客户信息，所以我们需要对用户划分权限。为了实现此模块的功能，我们不仅要掌握数据的添加、修改、查询和删除操作，还要掌握用户权限的设计思想和实现技术。

5.系统维护模块、日志、工具、帮助模块

以上这些模块均没用到数据操作知识点，在后续的章节中会详细讲解。

5.3 技术准备

5.3.1 ADO.NET 简介

ADO.NET 的名称起源于 ADO（ActiveX Data Objects），这是一个广泛的类组，用于在以往的 Microsoft 技术中访问数据。之所以使用 ADO.NET 名称，是因为 Microsoft 希望表明，这是在 .NET 编程环境中优先使用的数据访问接口

ADO.NET 是一组向 .NET 程序员公开数据访问服务的类。ADO.NET 为创建分布式数据共享应用程序提供了一组丰富的组件。它提供了对关系数据、XML 和应用程序数据的访问，因此是 .NET Framework 中不可缺少的一部分。ADO.NET 支持多种开发需求，包括创建由应用程序、工具、语言或 Internet 浏览器使用的前端数据库客户端和中间层业务对象。

ADO.NET 可让用户以一致的方式存取资料来源（例如 SQL Server 与 XML），以及透过 OLE DB 和 ODBC 公开的资料来源。资料共用的消费者应用程序可使用 ADO.NET 来连接至这些资料来源，并且撷取、处理及更新其中所含的资料。

ADO.NET 可将管理的资料存取分成不连续的元件，这些元件可分开使用，也可串联使用。ADO.NET 也包含 .NET Framework 资料提供者，以用于连接资料库、执行命令和撷取结果。这些结果会直接处理、放入 ADO.NET DataSet 物件中以便利用机器操作（Ad Hoc）的方式公开给使用者、与多个来源的资料结合，或在各层之间进行传递。DataSet 物件也可以与.

NET Framework 资料提供者分开使用，以便管理应用程序本机的资料或来自 XML 的资料。

ADO. NET 具有互操作性、可维护性、可编程性、性能优化、可伸缩性等优点。与 ADO 的早期版本和其他数据访问组件相比，ADO. NET 提供了若干好处。这些好处分成以下几个类别：

(1)互操作性。

ADO. NET 应用程序可以利用 XML 的灵活性和广泛接受性。由于 XML 是用于在网络中传输数据集的格式，因此可以读取 XML 格式的任何组件都可以处理数据。实际上，接收组件根本不必是 ADO. NET 组件。传输组件可以只是将数据集传输给其目标，而不考虑接收组件的实现方式。目标组件可以是 Visual Studio 应用程序或无论用什么工具实现的其他任何应用程序。唯一的要求是接收组件能够读取 XML。作为一项工业标准，XML 正是在谨记这种互操作性的情况下设计的。

(2)可维护性。

在已部署系统的生存期中，适度的更改是可能的，但由于十分困难，所以很少尝试进行实质的结构更改。这是很遗憾的，因为在事件的自然过程中，这种实质上的更改会变得很有必要。例如，当已部署的应用程序越来越受用户欢迎时，增加的性能负荷可能需要进行结构更改。随着已部署的应用程序服务器上的性能负荷的增长，系统资源会变得不足，并且响应时间或吞吐量会受到影响。面对该问题，软件设计者可以选择将服务器的业务逻辑处理和用户界面处理划分到单独计算机上的单独层上。实际上，应用程序服务器层将替换为两层，这就缓解了系统资源的缺乏。

该问题并不是要设计三层应用程序。相反，它是要在应用程序部署以后增加层数。如果原始应用程序使用数据集以 ADO. NET 方式实现，则该转换很容易进行。请记住，当用两层替换单个层时，将安排这两层交换信息。由于这些层可以通过 XML 格式的数据集传输数据，所以通信相对较容易。

(3)可编程性。

Visual Studio 中的 ADO. NET 数据组件以不同方式封装数据访问功能，能够加快编程速度并减少犯错几率。例如，数据命令提取生成，执行 SQL 语句或存储过程的任务等。

(4)性能优化。

对于不连接的应用程序，ADO. NET 数据库提供的性能优于 ADO 不连接的记录集。当使用 COM 封送在层间传输不连接的记录集时，会因将记录集内的值转换为 COM 可识别的数据类型而导致显著的处理开销。在 ADO. NET 中，这种数据类型转换则没有必要。

(5)可伸缩性。

因为 Web 可以极大增加对数据的需求，所以可缩放性变得很关键。Internet 应用程序具有无限的潜在用户供应。尽管应用程序可以很好地为十几个用户服务，但它可能不能向成百上千个(或成千上万个)用户提供同样好的服务。使用数据库锁和数据库连接之类资源的应用程序不能很好地为大量用户服务，因为用户对这些有限资源的需求最终将超出其供应。

5.3.2　数据访问对象

1. Connection 类

对数据源中的数据进行处理之前，必须先连接到数据源，通过 Connection 类的实例来完

成。根据.NET Framework 数据提供程序的不同，Connection 类也可以分成四类，分别是：SqlConnection、OleDbConnection、OdbcConnection 和 OracleConnection。实际编程的时候应根据访问的数据源不同，选择相应的 Connection 类。

SqlConnection：SQL Server 数据库连接类，该类在 System.Data.SqlClient 命名空间下，通过它的实例可以连接到指定的 SQL Server 数据库，并且该类提供打开和关闭数据库连接功能。利用 SqlConnection 的构造方法生成该类的实例，通过实例来实现连接和关闭数据库，该方法是一个重载的构造方法，有两个重载版本。

- Sqlconnection()
- Sqlconnection(string connectionString)

例如：

```
SqlConnection conn = new SqlConnection();
或
SqlConnection conn = new SqlConnection("data source=.;database=Neusoft_CRM;
                     integrated security=true");
```

Sqlconnection 类的连接字符串中的每个配置称为“属性”或“健值”。

格式如下：

属性=属性值；

常用的属性如表 5-1 所示：

表 5-1　　连接字符串属性表

成　员	说　明
Data source 或 Server 或 Address	所要连接到 SQL Server 实例名称或网络地址，连接本机上的 SQL Server 可设为(Local)或 .
Initial Catalog 或 Database	要连接的数据库名称
Integrated security	验证，属性值可为 yes 或 no ; true 或 false ;sspi
User id 或 Uid	使用 SQL Server 登录帐户来连接 SQL Server 实例。使用这两个属性来指定用户名和密码
Password 或 pwd	

注意：

- 属性不区分大小写。
- integrated security 属性指的是登录 SQL Server 是使用哪种方式登录，使 Windows 身份验证还是混合身份验证。

例如：

```
SqlConnection conn = new SqlConnection("data source=(local);database=Neusoft_CRM;
                     Uid=user;pwd=password"); //数据库登录方式为要求用户名和密码
或
SqlConnection conn = new SqlConnection("data source=(local);database=Neusoft_CRM;
                     integrated security=true");//数据库登录方式为 windows 用户登录
```

Sqlconnection 类的成员有：

(1)SqlConnection 类的属性。

SqlConnection 类的常用属性如表 5-2 所示。

表 5-2　　SqlConnection 类的属性

属性名	说明
ConnectionString	取得或设置连接的字符串
ConnectionTimeout	取得尝试连接的等待时间默认为 15 秒，只读属
DataBase	显示 Sqlconnection 对象在连接字符串中使用的数据库名称
Data Source	Sqlconnection 所要连接的 SQL Server 实例名称，只读属性
State	当前连接状态，只读属性

(2)SqlConnection 类的方法。

- open()：打开连接。
- close()：关闭连接。

例如：

```
conn. Open();
conn. Close();
```

为了方便读者通过具体实例领会和理解数据库应用编程的相关知识，本章所有例子均对数据库 Neusoft_CRM. mdf 进行操作，表 5-3～表 5-5 列出了使用的表名、表结构和已有数据。数据库及表的名称如此定义仅仅是为了方便记忆。

表 5-3　　Client_Area 表

ClientAreaId (nvchar,20,主键)	ClientAreaName (nvchar,20)
01	吉林省
02	辽宁省
03	河北省
04	河南省
05	湖北省
06	广东省
07	浙江省

表 5-4　　Client_Info 表

ClientId (nvchar, 20,主键)	ClientName (nvchar,20)	ClientGrade (nvchar,20)	ClientArea (nvchar,20)	ClientTel (nvchar,20)	ClientEmail (nvchar,20)	ClientAdress (nvchar,20)	ClientRemark (nvchar,20)
20051018004	余震	普通用户	吉林省	0431-5455522	sxdf@wd. com	湖北省荆州市	电饭锅
20051018005	张三	普通用户	河北省	0435-5455522	ejhr@qq. com	河南商丘	电饭锅
20051018006	李四	普通用户	吉林省	0431-3455522	dfg@qq. com	吉林长春	电冰箱
20051018007	王五	VIP 用户	河北省	0431-5455522	fcgdfg@qq. com	河北石家庄	电脑
20051018008	赵六	普通用户	浙江省	0331-5495522	Dslg@qq. com	舟山	电视

表 5-5　　User_Info 表

UserId (vchar,20,主键)	UserName (vchar,20)	UserPwd (vchar,20)	UserPower (vchar,20)	UserPhoto (image)
001	1	1	管理员	
002	2	2	普通用户	
003	3	3	普通用户	

例题 5_01:通过 SqlConnection 对象连接数据库示例。

【问题分析】通过设置 SqlConnection 对象的连接字符串指定连接到哪个数据库,通过 Open()方法连接数据库。

【程序说明】

(1)运行 Visual Studio 2010,创建一个名为 Connection 的 Windows 窗体应用程序项目。

(2)在窗口中添加一个 Button 控件,设置 Button 的 Text 属性为"单击连接数据库"。

(3)双击 Button 控件,进入 Click 事件。

(4)修改 Form1.CS 代码。

代码:

```
using System.Data.SqlClient;
SqlConnection conn = new SqlConnection("data source=.;database=Neusoft_CRM;
integrated security=true");
private void button1_Click(object sender, EventArgs e)
{
    if (conn.State == ConnectionState.Closed)
    {
        conn.Open();
        MessageBox.Show("已经连接到数据库 Neusoft_CRM");
        conn.Close();
    }
}
```

【运行结果】

按<F5>,运行结果如图 5-2 所示。

图 5-2 程序运行结果

2.Command 类

使用 Connection 对象与数据源建立连接后，可使用 Command 对象来对数据源执行查询、插入、删除和更新等各种操作，操作实现的方式可以使用 SQL 语句，也可以使用存储过程。根据.NET Framework 数据提供程序的不同，Command 对象也可以分成四类，分别是：SqlCommand、OleDbCommand、OdbcCommand 和 OracleCommand。实际编程的时候应根据访问的数据源不同，选择相应的 Command 对象。下面以 SqlCommand 为例讨论 Command 类。

SqlCommand 类实例构造的三种方法如下：

- SqlCommand compubs＝new SqlCommand()
- SqlCommand compubs＝new SqlCommand(string cmdText)
- SqlCommand compubs＝new SqlCommand(string cmdText,sqlconnection connection)

例如：

```
SqlCommand cmd = new SqlCommand();
SqlCommand cmd = new SqlCommand("select * from Client_Info");
SqlCommand cmd = new SqlCommand("select * from Client_Info",conn);
```

参数说明：

cmdText：所要执行的 SQL 语句或存储过程

Connection：SqlCommand 对象所要使用的数据连接对象

SqlCommand 的常用属性如表 5-6 所示。

表 5-6　　Sqlcommand 属性

成　员	说　明
CommandType	指示或指定如何解释 CommandText 属性
CommandText	获取或设置针对数据源运行的文本命令，SQL 语句或存储过程名
Connection	执行数据命令所需要的数据库连接

例如：

```
SqlCommand cmd = new SqlCommand();
cmd.CommandType = CommandType.Text;
cmd.CommandText = "select * from Client_Info";
cmd.Connection = conn;
```

Command 类的常用方法如表 5-7 所示。

表 5-7　　Command 类的常用方法

成　员	说　明
ExecuteNonQuery()	执行 Sql 语句或存储过程指定操作，通常不返回数据集的操作，比如 Update，Insert，Delete，返回被影响的数据记录的条数。
ExecuteScalar()	执行一个查询操作，并返回查询所返回的结果集中第 1 行的 1 列的数据，忽略其他列或行。
ExecuteReader()	执行 Sql 语句或存储过程指定的查询操作，比如 Select 语句，返回只向前的数据读取器。

例如：

```
cmd.ExecuteNonQuery();
cmd.ExecuteScalar();
cmd.ExecuteReader();
```

例题 5_02：通过 Command 对象连接的数据库进行操作示例。

【问题分析】设置 Command 对象的 Connection，指定连接的数据库，设置 CommandText，指定要执行的命令。

【程序说明】

(1)运行 Visual Studio 2010，创建一个名为 Command 的 Windows 窗体应用程序项目。

(2) 添加一个 Label 控件、一个 TextBox 控件和一个 Button 控件，更改 Label 控件的 Text 值为“请输入 ClientId：”Button 控件的 Text 改为“删 除”。

(3)双击“删除”按钮进入 Click 事件，修改 Form1.cs 代码。

代码如下：

```
using System.Data.SqlClient;
namespace Command
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        SqlConnection conn = new SqlConnection("data source=.;database=Neusoft_CRM;
                                                integrated security=true");
        SqlCommand cmd = new SqlCommand();
        private void Form1_Load(object sender, EventArgs e)
        {
            if (conn.State == ConnectionState.Closed)
            {
                conn.Open();
            }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            cmd.CommandText = "delete from Client_Info where ClientId=" + textBox1.Text;
            cmd.CommandType = CommandType.Text;
            cmd.Connection = conn;
            cmd.ExecuteNonQuery();
            MessageBox.Show("已删除该记录!");
        }
    }
}
```

【运行结果】

按<F5>，运行结果如图 5-3～图 5-5 所示。

表 - dbo.Client_Info 摘要

ClientId	ClientName	ClientGrade	ClientArea
20051019006	谢元	VIP 用户	吉林省
20051019007	如果	VIP 用户	湖北省

图 5-3　执行项目之前的 Client_Info 表

表 - dbo.Client_Info 摘要

ClientId	ClientName	ClientGrade	ClientArea
20051019007	如果	VIP 用户	湖北省

图 5-4　执行项目之后的 Client_Info 表

图 5-5　项目运行结果

3．DataReader 对象及其使用

许多数据操作要求用户只是读取一串数据。DataReader 对象允许用户获得从 Command 对象的 SELECT 语句得到的结果。考虑性能的因素，从 DataReader 返回的数据都是快速的且只是“向前”的数据流。这意味着用户只能按照一定的顺序从数据流中取出数据。这对于速度来说是有好处的，但是如果用户需要操作数据，更好的办法是使用 DataSet。

例如：

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

例题 5_03：通过 DataReader 对象，对连接的数据库进行读取示例。

【问题分析】通过 SqlCommand 对象查询数据，把得到的数据放到 DataReader 对象中。

【程序说明】

(1)运行 Visual Studio 2010，创建一个名为 DataReader 的 Windows 窗体应用程序项目。

(2) 添加一个 Label 控件、一个 TextBox 控件和一个 Button 控件，更改 Label 控件的 Text 值为“请输入 ClientId：”Button 控件的 Text 改为“查 找”。

(3)双击“查找”按钮，进入 Click 事件修改 Form1.cs 代码。

代码如下：

```
using System.Data.SqlClient;
namespace DataReader
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        SqlConnection conn = new SqlConnection("data
        source=.;database=Neusoft_CRM;integrated security=true");
        SqlCommand cmd = new SqlCommand();
        private void Form1_Load(object sender, EventArgs e)
        {
            if (conn.State == ConnectionState.Closed)
```

```
            {
                conn.Open();
            }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            cmd.CommandText = "select * from Client_Info where ClientId=" + textBox1.Text;
            cmd.CommandType = CommandType.Text;
            cmd.Connection = conn;
            SqlDataReader reader = cmd.ExecuteReader();
            if (reader.HasRows)
            {
                MessageBox.Show("查找到该记录!");
            }
            else
            {
                MessageBox.Show("没有该记录!");
            }
        }
    }
}
```

【运行结果】

按<F5>,运行结果如图 5-6 所示。

图 5-6 运行结果

4.DataAdapter 对象

DataAdapter 对象是 Dataset 对象和数据源之间联系的桥梁,主要功能是从数据源中检索数据、填充 Dataset 对象中的表、把用户对 Dataset 对象做出的更改写入到数据源。在.NET Framework 中主要使用两种 DataAdapter 对象:OleDb DataAdapter 和 SqlDataAdapter 。OleDbDataAdapter 对象适用于 OLE DB 数据源,SqlDataAdapter 对象适用于 SQL server 7.0 或更高版本。下面以 SqlDataAdapter 为例来介绍 DataAdapter 对象的常用属性和方法。

(1)DataAdapter 对象的常用属性。

DataAdapter 对象的常用属性如表 5-8 所示。

表 5-8　　DataAdapter 对象常用属性

成　员	说　明
InsertCommand	用来获取 SQL 语句或存储过程，用来把新记录插入到数据源中。通常把该属性设置为某个 Command 对象的名称，该 Command 对象执行 Insert 语句或存储过程。
UpdateCommand	用来获取 SQL 语句或存储过程，用来更新数据源中的记录。通常把该属性设置为某个 Command 对象的名称，该 Command 对象执行 Update 语句或存储过程。
DeleteCommand	用来获取 SQL 语句或存储过程，用于从数据集中删除记录。通常把该属性设置为某个 Conunand 对象的名称，该 Command 对象执行 Delete 语句或存储过程。
SelectCommand	用来获取 SQL 语句或存储过程，用来选择数据源中的记录。通常把该属性设置为某个 Command 对象的名称，该 Command 对象执行 Select 语句或存储过程。

例如：

```
SqlConnection conn = new SqlConnection("data source=(local);database=db_SMM;
                                        integrated security=true");
SqlCommand cmd = new SqlCommand();
SqlDataAdapter adp = new SqlDataAdapter();
cmd.Connection = conn;
cmd.CommandText = "select * from tb_LogInfo";
adp.SelectCommand = cmd;
```

(2) DataAdapter 对象的常用方法。

• Fill 方法：该方法有多种重载格式，其主要作用是通过 DataAdapter(数据适配器)对象从数据源中提取数据以填充数据集。下面是它的一个常用格式与功能。

[格式]：public intFill(DataSet dataset,string srcTable)；

[功能]：把 DataAdapter(数据适配器)对象从数据库服务器获取的数据填充到数据集“dataset”的表“srcTable”里。(注：srcTable 是新创建的表)

例如：

```
DataSet dataset = new DataSet();  //:创建数据集对象
adp.Fill(dataset,"newTable");
```

• Update 方法：该方法用于更新数据源，也有多种重载格式。其常用格式与功能如下：

[格式 1]：public override int Update(DataSet dataSet)；

[功能]：用数据集“dataSet”更新数据源。该方法通常用于数据集中只有一个表。

例如：

```
SqlCommandBuilder builder = new SqlCommandBuilder(adp);
adp.Update(dataset);
```

注：SqlCommandBuilder 自动生成单表命令，用于将 DataSet 所做的更改与关联的 SQL Server 数据库的更改相协调。

[格式 2]：public override int Update(DataSet dataSet, string Table)；

[功能]：用数据集“dataSet”中表“Table”更新数据源。该方法通常用于数据集中有多个表。

例如：

```
adp.Update(dataset,"newTable");
```

5.DataSet 对象及其使用

(1)DataSet 对象的组成。

Dataset 对象是一个创建在内存中的集合对象，它可以包含任意数量的数据表，以及所有表的约束、索引和关系，相当于在内存中的一个小型的关系数据库。一个 Dataset 对象包括一组 DataTable 对象和 DataRelation 对象，其中每个 DataTable 对象由 DataColumn、DataRow 和 DataRelation 对象组成，这些对象的含义如表 5-9 所示。

表 5-9　　数据集对象

成　员	说　明
DataTable	代表创建在 DataSet 中的表。
DataRelation	代表两个表之间的关系。关系建立在具有相同数据类型的列上，但列不必有相同的精确度。
DataColumn	代表与列有关的信息，包括列的名称、类型和属性。
DataRow	代表 DataTable 中的记录。

除了以上对象以外，DataSet 中还有几个集合对象：包含 DataTable 对象的集合 Tables 和包含 DataRelation 对象的集合 Relations。另外 DataTable 对象还包括行的集合 Rows、列的集合 Columns 和数据关系的集合 ChildRelations 和 ParentRelations。

(2)DataSet 对象的填充。

从数据源获取数据并填充到 DataSet 对象中的方法有以下几种。

调用 DataAdapter 对象的 Fill()方法，使用 DataAdapter 对象的 SelectCommand 的结果来填充 DataSet 对象。

例如：

```
DataSet dataset = new DataSet();   (注：创建数据集对象)
adp.Fill(dataset,"newTable");
```

通过程序创建 DataRow 对象，给 DataRow 对象的各列赋值，然后把 DataRow 对象添加到 Rows 集合中。

将 XML 文档或流读入到 DataSet 对象中。

合并(复制)另一个 DataSet 对象的内容到本 DataSet 对象中。

(3)DataSet 对象的访问。

DataSet 对象包含数据表的集合 Tables，而 DataTable 对象包含数据行的集合 Rows、数据列的集合 Columns，可以直接使用这些对象访问数据集中的数据。访问数据集中某数据表

的某行某列的数据，可采用如下的格式：

[格式 1]：数据集对象名. Tables["数据表名"]. Rows[n] ["列名"]；

[功能]：访问由"数据集对象名"指定的数据集中的由"数据表名"指定数据表的第 N+1 行的由"列名"指定的列。n 代表行号，从 0 开始。

例如：

```
string s = dataset. Tables["newTable"]. Rows[n]["ClientId"]. ToString();
```

[格式 2]：数据集对象名. Tables["数据表名"]. Rows[n]. ItemsArray[k]；

[功能]：访问由"数据集对象名"指定的数据集中的由"数据表名"指定数据表的第 N+1 行的第 K+1 列。N 代表行的序号，K 代表列的序号，从 0 开始。

例如：

```
string s = dataset. Tables["newTable"]. Rows[n]. ItemArray[0]. ToString();
```

(4)向 DataSet 对象中添加行。

向 DataSet 对象中添加新行要经过三个步骤：第一步给数据集中某个数据表添加一个新行；第二步是给新行的各列赋值；第三步是把新行添加到数据表的行集合中。

例如要给例题 5_04 中产生的数据集对象 dataset 中的数据表 Student 添加一个新行，可采用以下方法：

首先执行以下语句，给数据表添加一个新行：

```
DataRow row = dataset. Tables["newTable"]. NewRow();
```

接着给数据行的各列赋值，语句如下：

```
row["ClientId"] = "新加记录的 ClientID 列";
row[1] = ""新加记录的 ClientName 列";
    ……
row[7] = "新加记录的 ClientRemark 列";
```

最后把数据行加到数据表的行集合中，语句如下：

```
dataset. Tables["newTable"]. Rows. Add(row);
```

这样就实现了向数据表 Student 添加一行的功能。

(5)从 DataSet 对象中删除行。

使用相应行的 Delete 方法就可以删除行，如要删除 newTable 表的第 1 行，可执行如下语句：

```
dataset. Tables["newTable"]. Rows[0]. Delete();
```

(6)修改 DataSet 对象中的数据。

修改 DataSet 对象中某个数据表的某行某列的数据，只需给相应列赋新的数据即可，如要

把 newTable 表的第 1 行的第 3 个(ClientGrade)字段值修改为“江苏南京”,只需执行下列语句:

```
dataset.Tables["newTable"].Rows[0][2]=" 江苏南京";
```

(7)利用 DataSet 对象更新数据源。

对 DataSet 对象的更改并没有实际写入到数据源中,要将更改传递给数据源(即使用 DataSet 对象更新数据源),还需调用 DataAdapter 对象的 Update 方法。

例如:

```
if (dataset.HasChanges() == true)
{
    adp.Update(dataset, "newTable");
    dataset.AcceptChanges();
}
```

例题 5_04:通过 DataSet 对数据库进行更新。

【问题分析】通过 SqlDataAdapter 获得数据并填充到 DataSet,对 DataSet 进行操作,最后通过 SqlDataAdapter 对象的 Update 方法更新数据库。

【程序说明】

(1)运行 Visual Studio 2010,创建一个名为 DataAdapterUpdate 的 Windows 窗体应用程序项目。

(2)添加两个 Label 控件、两个 TextBox 控件和四个 Button 控件,更改 Label 控件的 Text 值为“ClientId:”、“ClientAreaName”,Button 控件的 Text 改为“查找”、“添 加”、“修改”和“删除”。

(3)双击“添加”按钮进入 Click 事件修改 Form1.cs 代码。

代码如下:

```
using System.Data.SqlClient;
namespace DataAdapterUpdate
{
    string str = "data source=(local);database=db_SMM;integrated security=true";
    SqlConnection conn = new SqlConnection();
    DataSet dataset = new DataSet();
    SqlDataAdapter adp=new SqlDataAdapter();
    SqlCommand cmd = new SqlCommand();

    public Form1()
    {
        InitializeComponent();
    }
    private void Open_Dataset(object sender, EventArgs e)
```

```
{
        cmd.CommandText = "select * from tb_LogInfo";
        cmd.Connection = conn;
        adp.SelectCommand = cmd;
        adp.Fill(dataset, "Mytable");
}
private void Form1_Load(object sender, EventArgs e)
{

}
  private void button1_Click(object sender, EventArgs e)
{
    int k = 0;
    Open_Dataset(sender,e);
    for (int i = 0; i < dataset.Tables["Mytable"].Rows.Count; i++)
    {
        if (dataset.Tables["Mytable"].Rows[i].RowState != DataRowState.Deleted)
        {
            if (dataset.Tables["Mytable"].Rows[i][0].ToString() == textBox1.Text)
            {
                k = 1;
                MessageBox.Show("数据集中已有该记录!");
                break;
            }
        }
}
f (k == 0)
{
    DataRow row = dataset.Tables["Mytable"].NewRow();
    row["ClientAreaId"] = textBox1.Text.Trim();
    row[1] = textBox2.Text.Trim();
    dataset.Tables["Mytable"].Rows.Add(row);
    SqlCommandBuilder builder = new SqlCommandBuilder(adp);
    if (dataset.HasChanges() == true)
    //该语句要加否则更新数据时可能出错
    {
        adp.Update(dataset, "Mytable");
        dataset.AcceptChanges();
        MessageBox.Show("添加成功,并已经更新数据库!");
        textBox1.Text = "";
        textBox2.Text = "";
        }
```

```
        }
        conn.Close();
    }
}
```

(4)运行项目,单击添加按钮,结果如图 5-7 所示。

图 5-7 添加操作结果

(5)双击“查找”按钮进入 Click 事件,修改 Form1.cs 代码,代码如下:

```
private void button2_Click(object sender, EventArgs e)
{
    int k = 0;
    Open_Dataset(sender, e);
    for (int i = 0; i < dataset.Tables["Mytable"].Rows.Count; i++)
    {
        if (dataset.Tables["Mytable"].Rows[i].RowState != DataRowState.Deleted)
        {
            if (dataset.Tables["Mytable"].Rows[i][0].ToString() == textBox1.Text)
            {
                k = 1;
                textBox2.Text = dataset.Tables["Mytable"].Rows[i][1].ToString();
                break;
            }
        }
    }
    if (k == 0)
    {
        MessageBox.Show("无该记录!");
    }
    else
    {
```

```
            MessageBox.Show("查找成功!");
        }
        conn.Close();
    }
```

(6)运行,单击查找按钮。结果如图 5-8 所示。

图 5-8　查找结果

(7)双击“修改”按钮进入 Click 事件,修改 Form1.cs 代码。代码如下:

```
private void button3_Click(object sender, EventArgs e)
{
    int k = 0,m=0;
    Open_Dataset(sender, e);
    for (int i = 0; i < dataset.Tables["Mytable"].Rows.Count; i++)
    {
        if (dataset.Tables["Mytable"].Rows[i].RowState != DataRowState.Deleted)
        {
            if (dataset.Tables["Mytable"].Rows[i][0].ToString() == textBox1.Text)
            {
                k = 1;
                m = i;
                break;
            }
        }
    }
    f (k == 0)
    {
        MessageBox.Show("无该记录!");
    }
    else
    {
```

```
            dataset.Tables["Mytable"].Rows[m][1] = textBox2.Text;
            SqlCommandBuilder builder = new SqlCommandBuilder(adp);
            if (dataset.HasChanges() == true)
            {
                adp.Update(dataset, "Mytable");
                dataset.AcceptChanges();
                MessageBox.Show("修改成功,并已经更新数据库!");
                textBox1.Text = "";
                textBox2.Text = "";
            }
        }
    }
```

(8)运行,单击修改按钮。结果如图 5-9 所示。

图 5-9 修改结果

(9)双击“删除”按钮进入 Click 事件,修改 Form1.cs 代码如下:

```
private void button4_Click(object sender, EventArgs e)
{
    int k = 0;
    Open_Dataset(sender, e);//
    for (int i = 0; i < dataset.Tables["Mytable"].Rows.Count; i++)
    {
        if (dataset.Tables["Mytable"].Rows[i].RowState != DataRowState.Deleted)
        {
            if (dataset.Tables["Mytable"].Rows[i][0].ToString() == textBox1.Text)
            {
                dataset.Tables["Mytable"].Rows[i].Delete();
                k = 1;
                if (dataset.HasChanges() == true)
                {
                    SqlCommandBuilder builder = new SqlCommandBuilder(adp);
                    adp.Update(dataset, "Mytable");
                    dataset.AcceptChanges();
```

```
                    MessageBox.Show("删除成功,并已经更新数据库!");
                    textBox1.Text = "";
                    textBox2.Text = "";
                }
                break;
            }
        }
    }
    if (k == 0)
    {
        MessageBox.Show("无该记录!");
    }
    conn.Close();
}
```

(10)运行,单击“删除”按钮。结果如图 5-10 所示。

图 5-10　删除结果

5.3.3　数据绑定控件

数据绑定是指在程序运行时,窗体上的控件自动将其属性和数据源关联在一起。数据绑定技术是数据操作中使用最频繁的技术,利用数据绑定技术极大地提高项目开发的效率。这一节简单介绍与此相关的概念和用法。

Windows 窗体提供了两种类型的数据绑定:简单数据绑定与复杂数据绑定。

简单数据绑定指将一个控件的某个属性绑定到单个值。这种类型的绑定适用于只显示单个值的控件,一般将这些控件绑定到数据库中的某个记录的一个字段。例如,将 TextBox 控件或者 Label 控件的“Text”属性绑定到数据集(DataSet)中的 MyTabel 表的“名称”字段,以便导航到某个记录时,它能自动显示该记录“名称”字段的值。除了常见的绑定“Text”属性以外,也可以将单值属性的其他控件的其他任何属性绑定到某字段,如绑定 TextBox 控件的“BackColor”属性到某个表示颜色的表字段等。

复杂数据绑定指将一个控件绑定到多个值。这种类型绑定适用于显示多个值的控件,例如 DataGridView 控件、ListBox 控件和 ComboBox 控件等,一般将这些控件绑定到多个记录,即一次性地将多个记录的某些字段显示出来。

从具体实现上来讲，又分为三种方式：

第 1 种方式是在设计界面下通过鼠标拖放实现常用属性的数据绑定。具体实现步骤：

(1)将“数据源”中的数据表字段直接拖放到设计窗体上，让系统自动创建和该表字段绑定的控件。

(2)将“数据源”中的表直接拖放到设计窗体上，让系统自动创建和该表绑定的控件。

(3)将“数据源”中的字段直接拖放到窗体上已有的控件上，该控件即自动绑定到拖放的表字段。

第 2 种方式是在设计界面下设置控件的 DataBinding 属性，然后利用可视化界面实现各种属性的数据绑定。工具箱中提供的每个控件，都有一个“DataBindings”属性，用于绑定数据源。利用它实现数据绑定的步骤如下：

(1)在窗体中，选择该控件并显示属性窗口，然后展开“DataBindings”属性，此时即看到与控件对应的默认绑定属性。实际上，用鼠标拖放所绑定的属性都是自动绑定到默认的属性。

(2)单击“Advanced”属性右边的“ ... ”按钮，显示(格式设置和高级绑定)对话框，在此对话框中，选择要绑定的数据源和被绑定的属性即可。

第 3 种方式是直接编写代码实现数据绑定。如果程序员希望灵活地控制绑定的数据库表字段，也可以直接编写绑定代码。采用这种方式实现数据绑定的步骤如下：

(1)从“工具箱”中拖放一个 BindingSource 组件、一个自动生成的强类型的 DataSet 组件、一个强类型的 DataAdapter 组件到设计窗体上。如果窗体上已经有这些对象，则不需要此步骤。

(2)从“工具箱”中拖放一个被绑定的控件到设计窗体上，修改控件的“Name”属性为有意义的名称。

(3)添加绑定代码。

实际上，不论采用哪种方式，从本质上来讲，都是利用 BindingSource 组件来实现数据绑定的。下面讲解常用的数据绑定控件。

1.comboBox 控件

ComboBox 控件，即下拉文本框，由文本框和列表框两部分组成。文本框可以用来编辑或者显示当前选中的条目，在下拉列表中绑定数据是绑定指定 ComboBox 的数据源，并设置 ComboBox 的 ValueMember 和 DisplayMember 属性，分别代表值和要显示的内容。若列表框隐藏，则单击文本框旁边带有向下箭头的按钮时可弹出列表框，使用键盘或者鼠标可以在列表框中快速选择条目。

常用的基本属性有：

(1)DropDownStyle 属性：表示组合框的显示样式，它有三种选择：

- Simple：同时显示文本框和列表框，文本框可以被编辑。
- DropDown：只显示文本框，隐藏列表框，且文本框可以被编辑。
- DropDownList：只显示文本框，隐藏列表框，但文本框不可以被编辑。

(2)MaxDropDownItems 属性：设置打开列表框时所显示的最大条目数，其他多出的部分可以以滚动条的方式显示查看。

其常用的事件有：

• SelectedIndexChanged 事件：当 SelectedIndex 属性更改后触发的操作。

例题 5_05：ComboBox 控件应用。

【问题分析】通过数据配置向导添加数据源，绑定 ComboBox 控件。

【程序说明】

(1)运行 Visual Studio 2010，创建一个名为 ComboBox 的 Windows 窗体应用程序项目。

(2)添加两个 Label 控件、两个 TextBox 控件，一个 ComboBox 和两个 Button 控件，更改 Label 控件的 Text 值为“省份名称：”、“代码值：”，Button 控件的 Text 改为“添加”、“删除”。效果如图 5-11 所示。

图 5-11　Form1 设计界面

(3)添加数据源，点击菜单|数据|添加新数据源，弹出数据源配合向导窗口，如图 5-12 所示。

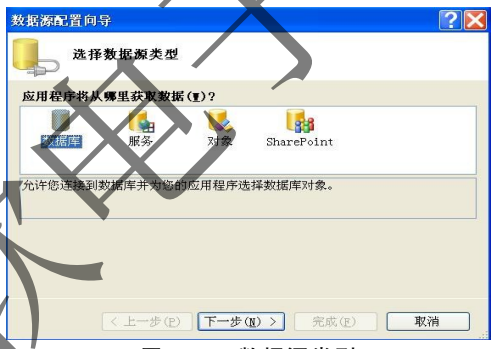

图 5-12　数据源类型

(4)选择数据库，单击下一步，弹出选择数据库模型窗口，如图 5-13 所示。

图 5-13　数据库模型

(5)选择数据集，单击下一步弹出选择数据连接窗口，如图 5-14 所示。

图 5-14　选择数据连接

(6)单击新建连接,弹出添加连接窗口,如图 5-15 进行设置。

图 5-15　添加连接

(7)单击"确定",如图 5-16 所示。

图 5-16　生成连接字符串

(8)单击"下一步",如图 5-17 所示。

图 5-17　连接字符串保存

(9)单击“下一步”,如图 5-18 所示。

图 5-18　选择数据集中包含的内容

(10)选择表 Client_Area 单击完成。

(11)点击 Combobox 控件右上角黑色箭头,弹出 Combobox 任务窗口,设置如图 5-19 所示。

图 5-19　ComboBox 任务窗口

(12)双击窗体 Form1,在 Form1.cs 中添加如下代码:

```
using System.Data.SqlClient;
namespace comboBox
{
    public partial class Form1 : Form
    {
```

```
        string str = "data source=(local);database=Neusoft_CRM;integrated security=true";
        SqlConnection conn = new SqlConnection();
        SqlCommand cmd = new SqlCommand();
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.client_AreaTableAdapter.Fill(this.neusoft_CRMDataSet.Client_Area);
            //通过 combobox 任务窗口添加的代码
            comboBox1.SelectedIndex = 0;
            comboBox1_SelectedIndexChanged(sender, e);
        }
    }
}
```

(13)双击“添加”按钮,在 Click 事件中添加如下代码:

```
private void button1_Click(object sender, EventArgs e)
{
            conn.ConnectionString = str;
            if (conn.State == ConnectionState.Closed)
                conn.Open();
            cmd.Connection = conn;
            cmd.CommandType = CommandType.Text;
            cmd.CommandText = "insert into Client_Area values(" + "'" +
                textBox2.Text + "','" + textBox1.Text + "')";
            cmd.ExecuteNonQuery();
            textBox1.Text = "";
            textBox2.Text = "";
            conn.Close();
  }
```

(14)双击“删除”按钮,在 Click 事件中添加如下代码:

```
private void button2_Click(object sender, EventArgs e)
{
        conn.ConnectionString = str;
        if (conn.State == ConnectionState.Closed)
            conn.Open();
        cmd.Connection = conn;
```

```
        cmd.CommandType = CommandType.Text;
        cmd.CommandText = "delete from Client_Area where num=" + textBox2.Text;
        cmd.ExecuteNonQuery();
        textBox1.Text = "";
        textBox2.Text = "";
        conn.Close();
}
```

(15)双击 ComboBox 控件,在 SelectedIndexChanged 事件中添加如下代码:

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
        textBox1.Text = comboBox1.Text;
        textBox2.Text = comboBox1.SelectedValue.ToString();
}
```

【运行结果】

运行,在 ComboBox 控件下选择一项。结果如图 5-20 所示。

图 5-20　选择显示界面

添加数据并点击“添加“按钮。结果如图 5-21 所示。

添加数据并点击“删除”按钮。结果如图 5-22 所示。

图 5-21　添加结果

图 5-22　删除结果

2.BindingSource

数据绑定的目的是为了简化编程,然而实际数据有多种形式,包括数据库、对象和 Web 服务等都是实际数据。由于不同的数据处理方式不同,如果把这个工作全部交给程序员来完成,设计工作量会很大。BindingSource 组件就是为简化设计而提供的。

BindingSource 组件的作用之一就是实现统一的数据绑定形式,即让 BindingSource 组件绑定实际数据,让控件绑定 BindingSource 组件,而与数据的所有进一步交互(包括导航、排序、筛选和更新)全部都是通过 BindingSource 组件来完成。

实际上,在前面介绍的各种绑定技术中,都是先将 BindingSource 组件与实际数据绑定,然后将控件绑定到 BindingSource 组件。

对于大多数数据库应用,在设计时通过简单的拖放操作就可以完成大部分功能。但是,通过编写代码实现数据绑定,可以使程序控制更灵活。直接通过编写代码完成数据绑定的操作叫做运行时数据绑定。

当然,在项目设计中,是不可能把设计时绑定和运行时绑定截然分开的。多数情况下,都会将两者配合使用。

运行时实现数据绑定的常用方法为:

(1) 从工具箱中拖放一个 BindingSource 组件,一个自动生成的强类型的 DataSet 组件,一个强类型的 DataAdapter 组件到设计窗体上。如果窗体上已经有这些对象,则不需要此步骤。

(2) 从工具箱中拖放一个被绑定的控件到设计窗体上,修改控件的【Name】属性为有意义的名称。

(3) 添加绑定代码。

在代码方式下,大部分控件都是通过 DataBindings 属性的 Add 方法实现绑定的。下面通过具体例子列举一些常用控件的运行时绑定方法。

例题 5_06:直接编写代码完成数据绑定的应用。

【问题分析】通过控件的 DataBindings 实现控件某个属性与数据的绑定。

【程序说明】

(1)运行 Visual Studio 2010,创建一个名为 BindingSource 的 Windows 窗体应用程序项目。

(2)添加数据源步骤参考例 5_05(选择表 Client_Info),然后选择菜单的“生成”|“生成解决方案”,以便能在工具箱中看到自动生成的强类型的组件。

(3)从工具箱中向设计窗体拖放一个 DataGridView 控件,一个 DataSet 组件,一个 TableAdapter 组件、一个 BindingSource 组件和一个 BindingNavigator 组件,修改对应的“Name”属性。然后再向窗体拖放八个 Label 控件、八个 TextBox 和两个 Button 控件,修改对应的“Name”属性和“Text”属性,设计效果如图 5-23 所示:

图 5-23 设计界面

(4)添加 Form1 窗体的 Load 事件代码：

```
private void Form1_Load(object sender, EventArgs e)
{
    client_InfoTableAdapter1.Fill(neusoft_CRMDataSet1.Client_Info);
}
```

(5)双击“绑定”按钮，添加 Click 事件代码：

```
private void button1_Click(object sender, EventArgs e)
{
    bindingSource2.DataSource = neusoft_CRMDataSet1;
    bindingSource2.DataMember = "Client_Info";//绑定导航条
    bindingNavigator1.BindingSource = bindingSource2;
    textBox1.DataBindings.Add("Text", bindingSource2, "ClientId");
    textBox2.DataBindings.Add("Text", bindingSource2, "ClientName");
    textBox3.DataBindings.Add("Text", bindingSource2, "ClientGrade");
    textBox4.DataBindings.Add("Text", bindingSource2, "ClientArea");
    textBox5.DataBindings.Add("Text", bindingSource2, "ClientTel");
    textBox6.DataBindings.Add("Text", bindingSource2, "ClientEmail");
    textBox7.DataBindings.Add("Text", bindingSource2, "ClientAdress");
    textBox8.DataBindings.Add("Text", bindingSource2, "ClientRemark");
    dataGridView1.DataSource = bindingSource2;
}
```

(6) 双击“取消绑定”按钮，添加 Click 事件代码：

```
private void button2_Click(object sender, EventArgs e)
{
    //绑定导航条
    bindingNavigator1.BindingSource = null;
    textBox1.DataBindings.Clear();
    textBox2.DataBindings.Clear();
    textBox3.DataBindings.Clear();
    textBox8.DataBindings.Clear();
    textBox4.DataBindings.Clear();
    textBox5.DataBindings.Clear();
    textBox6.DataBindings.Clear();
    textBox7.DataBindings.Clear();
    textBox1.Text = "";
    textBox2.Text = "";
    textBox3.Text = "";
    textBox4.Text = "";
    textBox5.Text = "";
```

```
        textBox6.Text = "";
        textBox7.Text = "";
        textBox8.Text = "";
        dataGridView1.DataSource = null;
        bindingSource2.DataSource = null;
        bindingSource2.DataMember = null;
    }
```

【运行结果】

运行,单击“绑定”和“取消绑定”按钮,效果如图 5-24 和图 5-25 所示。

图 5-24 绑定后显示效果

图 5-25 取消绑定后显示效果

3.BindingNavigator

BindingNavigator 控件提供了一个 ToolStrip 导航条,该导航条默认提供第一条记录、最后一条记录、上一条记录和下一条记录按钮,以及添加和删除记录的按钮。这些按钮与 BindingSource 组件的 MoveFirst 方法等,即自动实现了利用 BindingSource 组件以编程方式实现的功能。将 BindingNavigator 控件和 BindingSource 组件配合使用,可以轻松实现对数据表

的添加、删除以及移动当前记录位置等功能。

BindingNavigator 控件常用的属性如下：

(1)“BindingSource”属性：指定所要绑定的 BindingSource 对象。

(2)“Dock”属性：确定 BindingNavigator 控件在窗体中的停靠位置。

下面通过一个例子说明 BindingNavigator 控件的基本用法，注意这个例子中仅仅使用了 BindingNavigator 控件提供的默认属性，没有处理当单击 BindingNavigator 导航条上面的按钮出现异常的情况，例如连续单击导航条的“+”出现的异常等。在对应的实验指导中，再通过具体步骤练习解决这个问题的办法。

例题 5_07：BindingNavigator 控件的应用。

【问题分析】DataGridView 控件和 BindingNavigator 对象绑定的数据源根据 ListBox 的选择进行变化。

【程序说明】

(1)运行 Visual Studio 2010，创建一个名为 BindingNavigator 的 Windows 窗体应用程序项目。

(2)添加数据源，步骤参考例题 5-05(选择表：全选)，然后选择菜单的“生成”|“生成解决方案”，以便能在工具箱中看到自动生成的强类型的组件。

(3)从工具箱中向设计窗体拖放一个 DataSet 组件，一个 BindSource 组件，一个 DataGridView 控件，一个 BindingNavigator 控件，两个 GroupBox 控件，一个 ListBox 控件和三个 Button 控件。设计界面效果如图 5-26 所示。

图 5-26　设计界面

(4)在代码中添加命名空间的引用。

```
using System. Data. SqlClient;
```

(5)在构造函数上方添加类一级的对象声明。

```
SqlDataAdapter adapter;
DataTable selectedTable;
```

(6)双击窗体，添加 Form1_Load 对应的事件代码如下：

```
private void Form1_Load(object sender, EventArgs e)
```

```
{
    bindingNavigator1.BindingSource = bindingSource1;
    //将 MyDatabase.mdf 中的表名称添加到 dataTableListBox 中
    for (int i = 0; i < neusoft_CRMDataSet11.Tables.Count; i++)
    {
        listBox1.Items.Add(neusoft_CRMDataSet11.Tables[i].TableName);
    }
    //listBox1.SelectedIndex = 0;
    //不允许用户直接在最下面的行添加新行
    dataGridView1.AllowUserToAddRows = false;
    //不允许用户直接按 Delete 键删除行
    dataGridView1.AllowUserToDeleteRows = false;
}
```

(7)双击 ListBox 控件,进入默认的 listBox1_SelectedIndexChanged 事件,添加代码如下:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    int index = listBox1.SelectedIndex;
    selectedTable = neusoft_CRMDataSet11.Tables[index];
    string queryString = "select * from " + selectedTable.TableName;
    adapter = new SqlDataAdapter(queryString, Properties.Settings.Default.Neusoft_CRMConnectionString);
    SqlCommandBuilder builer = new SqlCommandBuilder(adapter);
    adapter.InsertCommand = builer.GetInsertCommand();
    adapter.DeleteCommand = builer.GetDeleteCommand();
    adapter.UpdateCommand = builer.GetUpdateCommand();
    adapter.Fill(selectedTable);
    bindingSource1.DataSource = selectedTable;
    dataGridView1.DataSource = bindingSource1;
}
```

(8)双击“添加新记录”按钮,进入 button1_Click 事件,添加代码如下:

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        bindingSource1.AddNew();
    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message);
    }
}
```

(9)双击“删除所有记录”按钮,进入 button3_Click 事件,添加代码如下:

```
private void button3_Click(object sender, EventArgs e)
{
    if (dataGridView1.SelectedRows.Count == 0)
    {
    MessageBox.Show(  "请先单击最左边的空白列选择要删除的行,可以按住<Ctrl>同时选中多行");
    }
    else
    {
        if (MessageBox.Show( "确实要删除选定的行吗?", "小心",
            MessageBoxButtons.YesNo, MessageBoxIcon.Warning) ==
            DialogResult.Yes)
        {
            for (int i = dataGridView1.SelectedRows.Count - 1; i >= 0; i--)
            {
            bindingSource1.RemoveAt(dataGridView1.SelectedRows[i].Index);
            }
        }
    }
}
```

(10)双击“保存修改”按钮,进入 button2_Click 事件,添加代码如下:

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
      this.Validate();
      bindingSource1.EndEdit();
      adapter.Update(neusoft_CRMDataSet11.Tables[listBox1.SelectedIndex]);
       MessageBox.Show("保存成功");
    }
    catch (Exception err)
    {
        MessageBox.Show(err.Message, "保存失败");
     }
}
```

【运行结果】

按<F5>,运行结果如图 5-27 所示。

图 5-27 选择 Client_Info 表，显示表数据

点击“添加新记录”按钮，输入数据，并点击 BindingNavigator 控件上的添加()，结果如图 5-28 所示。

图 5-28 添加记录

点击“保存修改”和“删除所有记录”按钮，在 DataGridView 控件中查看结果。

4.DataGridView

在前面的例子中，虽然多处用到 DataGridView 控件，但也仅仅是利用它简单地显示或编辑数据，并没有涉及该控件的其他用法。实际上，DataGridView 控件是一个非常复杂的控件，利用该控件，除了可以显示、编辑数据之外，还可以利用它进行灵活的样式控制和数据校验处理。

将 DataGridView 控件从“工具箱”中拖放到设计窗体上后，如果不做其他处理，则该控件默认具有如下功能：

(1)将该控件绑定到数据源时，数据源列的名称自动作为该控件的列标题，而且上下移动滚动条时，列标题位置固定不变。

(2)支持自动排序。用鼠标单击某个列标题，则对应的列就会自动按升序或降序排序(单击升序，再单击降序)。字母顺序区分大小写。

(3)单击 DataGridView 左上角的矩形可以选择整个表，单击每行左边的矩形块可以选择整行。

(4)支持自动调整大小功能。在标题之间的列分隔符上双击，该分隔符左边的列会自动按照单元格的内容展开或收缩。

(5)当用户单击单元格时，默认选中整个单元格。可以设置 DataGridView 的“EditMode”属性来更改 DatagridView 控件的默认模式。

(6)在编辑模式中，用户可以更改单元格的值，并可以按＜Enter＞键提交更改，或按

＜Esc＞键将单元格恢复为原始值。

(7)如果用户滚动至网格的结尾,将会看到用于添加新纪录的行。用户单击此行时,会向 DataGridView 控件添加使用默认值的新行,用户按＜Esc＞键时,此新行将消失。

由于 DataGridView 提供的属性、方法、事件非常多,限于篇幅,无法将其全部罗列出。这里仅举一个例子说明常用的数据处理技术。

例题 5_08:演示 DataGridView 常用功能。

【问题分析】通过设置 BindingNavigator 控件的 BindingSource 属性, DataGridView 控件的 DataSource 属性来设置绑定和取消绑定。

【程序说明】

(1)运行 Visual Studio 2010,创建一个名为 DataGridView 的 Windows 窗体应用程序项目。

(2)添加数据源,步骤参考例题 5-05(选择表 Client_Info)。然后选择菜单的“生成”|“生成解决方案”,以便能在工具箱中看到自动生成的强类型的组件。

(3)从工具箱中向设计窗体拖放一个 BindingSource 组件、一个 BindingNavigator 组件、一个 DataGridView 控件、一个 DataSet 组件和一个 TableAdapter 组件,不改变自动生成的对象名称。

(4)向窗体中添加一个 Button 按钮,修改“Name”属性为“buttonBindingDataSource”,“Text”属性为“绑定/取消绑定到数据源”。然后双击该按钮,添加 Click 事件代码如下:

```
private void buttonBindingDataSource_Click(object sender, EventArgs e)
{
    if (dataGridView1.DataSource != null)
    {
        bindingNavigator1.BindingSource = null;
        dataGridView1.DataSource = null;
    }
    else
    {
        client_InfoTableAdapter1.Fill(neusoft_CRMDataSet1.Client_Info);
        bindingSource1.DataSource = neusoft_CRMDataSet1.Client_Info;
        bindingNavigator1.BindingSource = bindingSource1;
        dataGridView1.DataSource = bindingSource1;
    }
}
```

(5)按＜F5＞键运行程序,单击 buttonBindingDataSource 按钮,DataGridView 中应该显示 Client_Info 表的数据,导航条中也应该有相应的信息;再次单击该按钮,DataGridView 和导航条均不可用。结果如图 5-29 所示。

图 5-29 GridView 控件绑定数据

(6)向窗体中添加一个 Button 按钮,修改“Name“属性为“buttonShowExpectedField”,“Text”属性为“显示全部/部分字段”。然后双击该按钮,添加 Click 事件代码如下:

```
private void buttonShowExpectedField_Click(object sender, EventArgs e)
{
    dataGridView1.Columns["ClientRemark"].Visible
    = ! dataGridView1.Columns["ClientRemark"].Visible;
}
```

(7)按<F5>键运行程序,多次单击 buttonShowExpectedField 按钮,观察照片字段的显示变化情况。

(8)向窗体中添加一个 Button 按钮,修改“Name”属性为“buttonExchangeColumn”,“Text”属性为“交换两列”。然后双击该按钮,添加 Click 事件代码如下:

```
private void buttonExchangeColumn_Click(object sender, EventArgs e)
{
    int columnIndex = dataGridView1.Columns["ClientId"].DisplayIndex;
    dataGridView1.Columns["ClientId"].DisplayIndex =
    dataGridView1.Columns["ClientName"].DisplayIndex;
    dataGridView1.Columns["ClientName"].DisplayIndex = columnIndex;
}
```

按<F5>,运行结果如图 5-30 所示。

图 5-30 交换后的 GridView 控件内容

5.4　项目实施

本项目实现需要的数据库操作比较多，由于篇幅有限，本章只讲解重要的数据库操作，具体代码参考随书光盘。

5.4.1　登录窗体

1.登录窗体概述

登录窗体主要实现了用户的登录功能，用户必须通过输入正确的用户名称、用户密码和验证码才能成功登录。因为用户姓名和用户密码是存放在数据库里的，所以此窗体的主要数据库操作是判断用户名和用户密码是否正确。运行结果如图 5-31 所示。

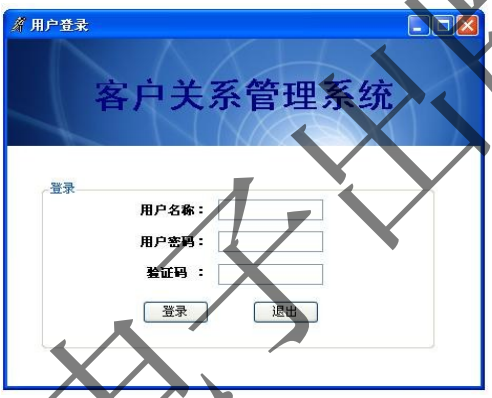

图 5-31　用户登录界面

2.登录窗体实现过程

登录窗体具体实现步骤如下：

(1)新建一个 Windows 窗体，命名为 frmLogin.cs，具体窗体设计参照第 4 章内容。

(2)此窗体需要声明的变量如下：

```
dbOperate myUserLogin = new dbOperate();//实例化数据库操作对象
public static string trueName = "";//记录用户名
public static string truePwd = "";//记录用户密码
public static string truePower = "";//记录用户权限
public  string trueNum; //记录个人日志编号
string checkCode = String.Empty;//记录验证码
```

关键代码解析：

dbOperate myUserLogin = new dbOperate();

功能：从公共类 dbOperate 实例化 myUserLogin 对象，就可以通过 myUserLogin 对象实现类 dbOperate 的属性和方法。

(3)在 frmLogin 窗体的 Load 事件中关键代码如下：

```
private void frmLogin_Load(object sender, EventArgs e)
{
    myUserLogin.dbCon();//数据库连接
    this.label2.Visible = false; //设置 label2 不可见
    this.pictureBox1.Visible = false;
    this.linkLabel1.Visible = false;
    this.textBox1.GotFocus += new EventHandler(textBox1_GotFocus);
}
```

关键代码解析：

myUserLogin.dbCon();

功能：通过调用 myUserLogin 的 dbCon 方法，实现数据库的连接。

this.textBox1.GotFocus += new EventHandler(textBox1_GotFocus);

功能：当 textBox1 获得焦点时，显示验证码。

(4)“登录”按钮的 Click 事件中关键代码如下：

```
private void butnLogin_Click(object sender, EventArgs e)
{
    //声明 SQL 语句，语句主要实现查询操作
    string sqlUserSelect = "select * from User_Info where UserName='" +
    tBoxName.Text.Trim() + "' and UserPwd='" + tBoxPassword.Text.Trim() + "'";
    //执行 SQL 语句查看返回记录集
    SqlDataReader myDtReader = myUserLogin.dbRead(sqlUserSelect);
    //如果 myDtReader.HasRows 为真说明输入的姓名和密码是正确的，如果为假证明输入的信息有误
    if (myDtReader.HasRows)
    {
        //判断是否输入验证码
        if (this.textBox1.Text.ToUpper().Trim() == "")
        {
            MessageBox.Show("请输入验证码", "提示");
            checkCode = String.Empty;
            this.label2.Text = GenerateCheckCode();
            this.pictureBox1.Image = this.CreateCheckCodeImage(this.label2.Text);
        }
        //判断验证码是否输入正确
        if (this.textBox1.Text.ToUpper().Trim() != checkCode)
        {
            MessageBox.Show("验证码输入错误", "提示");
            checkCode = String.Empty;
            textBox1.Text = "";
            this.label2.Text = GenerateCheckCode();
```

```
                this.pictureBox1.Image = this.CreateCheckCodeImage(this.label2.Text);
                return;
            }
            while (myDtReader.Read())
            {
                trueNum = myDtReader.GetString(0);//读取用户 ID
                trueName = tBoxName.Text.ToString();//读取用户姓名
                truePwd = tBoxPassword.Text.ToString();//读取用户密码
                truePower = myDtReader.GetString(3).ToString();//读取用户权限
                MessageBox.Show("" + truePower + "欢迎使用客户关系管理系统");
                frmMain myFrmMain = new frmMain();//实例化主窗体
                frmMain.noteNum = trueNum ; //记录用户 ID
                this.Hide();
                myFrmMain.Show();//显示主窗体
            }
            myDtReader.Close();//关闭记录集对象
        }
        //如果输入的用户信息有误,弹出消息对话框
        else
        {
            MessageBox.Show("用户名或密码错误","提示", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
            tBoxName.Text = "";
            tBoxPassword.Text = "";
            tBoxName.Focus();
        }
        myDtReader.Close();
    }
```

关键代码解析：

'" + tBoxName.Text.Trim() + "'

功能：用 tBoxName 控件中输入的信息绑定数据表的 UserName 字段。

while (myDtReader.Read())

功能：选中查询的记录。

myDtReader.GetString(0)

功能：读取查询记录中索引为 0 的字段的值，参照数据库设计该字段为字符串类型。

(5)参照第 8 章内容设计验证码功能。

5.4.2　系统主窗体

1.系统窗体概述

主窗体是个 MDI 窗体，通过 MDI 窗体可以同时打开多个子窗体，界面清晰不易混乱。运

行结果如图 5-32 所示。

图 5-32 客户关系管理系统界面

2.系统主窗体的实现

系统主窗体具体实现步骤如下：

(1)新建一个 Windows 窗体，命名为 frmMain.cs，具体窗体设计参照第 4 章内容。

(2)此窗体需要声明的变量如下：

```
public static string noteNum;//记录日志文件编号
```

(3)在 frmMain 窗体的 Load 事件中关键代码如下：

```
private void frmMain_Load(object sender, EventArgs e)
{
    tSSTime.Text = "||登录时间:" + DateTime.Now.ToLongDateString() + " " +
    DateTime.Now.ToLongTimeString();
    //如果是“普通用户”,不显示以下菜单信息,实现用户权限的限制功能
    if (frmLogin.truePower.ToString()=="普通用户")
    {
        客户资料管理 ToolStripMenuItem.Visible =false;
        客户区域管理 ToolStripMenuItem.Visible = false;
        客户信息管理 ToolStripMenuItem.Visible = false;
        系统维护 ToolStripMenuItem.Visible = false;
    }
}
```

关键代码解析：

frmLogin.truePower.ToString();

功能：通过调用在登录窗体里定义的静态变量“public static string truePower”来获得用户权限值。静态变量通过类名调用。

(4)frmMain 窗体的客户资料查询菜单的 Click 事件中关键代码如下：

```
private void 客户资料查询 ToolStripMenuItem_Click(object sender, EventArgs e)
{
        //从 frmClientManage 窗体类里实例化 myClientManage 对象
        frmClientManage myClientManage = new frmClientManage();
        myClientManage.MdiParent = this;
        myClientManage.butnAdd.Visible    = false;
        myClientManage.butnDelete.Visible = false;
        myClientManage.butnUpdate.Visible = false;
        myClientManage.butnSure.Visible = false;
        myClientManage.butnReset.Visible = false;
        myClientManage.Show();
}
```

关键代码解析：

myClientManage.MdiParent = this;

功能：设置 myClientManage 为子窗体。

(5)其他菜单实现代码参考步骤(4)。

5.4.3　客户资料管理窗体

1.客户资料管理窗体概述

客户资料管理模块用来实现客户信息的添加、修改、删除和查找功能。该窗体中使用了公共类 dbOperate 中相关的方法，对客户信息进行验证。运行结果如图 5-33 所示。

图 5-33　客户资料管理界面

2.客户资料管理窗体具体实现步骤如下：

(1)新建一个 Windows 窗体，命名为 frmClientManage.cs，具体窗体设计参照第 4 章内容。

(2)此窗体需要声明的变量如下：

```
dbOperate myClientManage = new dbOperate();  //从公共类实例化对象 myClientManage
DataSet myds = new DataSet();//实例化数据集对象
int mark = 0;//定义操作标记
```

(3)在 frmClientManage 窗体的 Load 事件中关键代码如下：

```
private void frmClientManage_Load(object sender, EventArgs e)
{
    myClientManage.dbCon();//连接数据库
    dvClientInfoBound();//窗体刷新数据
    groupBox1.Visible = false;
}
```

方法：dvClientInfoBound()。
功能：实现窗体刷新数据。
实现代码如下：

```
public void dvClientInfoBound()
{
    string mySql = "SELECT  ClientId as 用户编号, ClientName as 用户名称,
    ClientGrade as 用户级别 ClientArea as 用户区域, ClientTel as 用户电话,
    ClientEmail as 用户邮箱, ClientAdress as 用户地址, ClientRemark as 备注 FROM Client_Info ";
    myds = myClientManage.Sqldataset(mySql); //返回 mySql 语句执行的记录集
    if (myds.Tables[0].Rows.Count > 0)
        dvClientInfo.DataSource = myds.Tables[0].DefaultView;
}
```

关键代码解析：
myds.Tables[0].Rows.Count > 0
功能：记录集 myds 中索引为 0 的表中记录的数量大于零。
dvClientInfo.DataSource = myds.Tables[0].DefaultView
功能：dvClientInfo 控件的 DataSource 属性为 myds 数据集中索引为 0 的表。
(4)“添加”按钮的 Click 事件中关键代码如下：

```
private void butnAdd_Click(object sender, EventArgs e)
{
    clear();//调用 clear()方法
    mark = 1;//设置 mark=1,表示执行添加操作
    cBoxAreaBound();//调用 cBoxAreaBound()方法
    string clientId = getId();//调用 getId()方法
    tBoxClientId.Text = clientId.ToString();
```

```
        tBoxClientId. Enabled = false;
        butnDelete. Enabled = false;
        butnUpdate. Enabled = false;
        groupBox1. Visible = true;//显示客户资料信息
    }
```

方法:dvClientInfoBound()。
功能:实现窗体刷新数据。
实现代码如下:

```
    public void dvClientInfoBound()
    {
        string mySql = "SELECT  ClientId as 用户编号, ClientName as 用户姓名,
        ClientGrade as 用户级别, ClientArea as 用户区域, ClientTel as 用户电话,
        ClientEmail as 用户邮箱, ClientAdress as 用户地址, ClientRemark as 备注 FROM Client_Info ";
        myds = myClientManage. Sqldataset(mySql);  //返回查询记录集
        if (myds. Tables[0]. Rows. Count > 0)
            dvClientInfo. DataSource = myds. Tables[0]. DefaultView; //绑定记录集中的数据
    }
```

方法:getId()。
功能:生成添加用户的 ID。
实现代码如下:

```
    public string  getId()
    {
        string year = "";
        string month = "";
        string day = "";
        string mydate = "";
        string lastClientId = "";
        DateTime mydt = DateTime. Now; //获取系统时间
        year = DateTime. Now. Year. ToString();//获取年
        month = DateTime. Now. Month. ToString();//获取月
        day = DateTime. Now. Day. ToString();//获取日
        //判断,如果连接状态为关闭,就打开连接,否则不执行任何操作
        if (myClientManage. myCon. State == ConnectionState. Closed)
         { myClientManage. dbCon(); }
        //定义 SQL 语句返回最后一条记录
        string sqlItemsCount = "SELECT TOP (1) ClientId FROM  Client_Info ORDER BY ClientId
DESC";
        SqlDataReader mydr = myClientManage. dbRead(sqlItemsCount); //执行 SQL 语句
        if (mydr. HasRows)
```

```
    {
        while (mydr.Read())
        {
            //获取最后一条记录索引为 0 的字段的值
            astClientId = mydr.GetString(0).ToString();
        }
    }
    int lastId = Convert.ToInt32(lastClientId.Substring(8, 3))+1;//字符串截取
    string strLastIdFormat = string.Format("{0:d3}", lastId);//
    mydate = year + month + day + strLastIdFormat;
    return mydate;
}
```

关键代码解析：

SELECT TOP (1) ClientId FROM Client_Info ORDER BY ClientId DESC

功能：按倒序查找第一条记录。

LastClientId.Substring(8, 3)

功能：从索引为 8 的字符开始查找，查找长度为 3。

(5)“修改”按钮的 Click 事件中关键代码如下：

```
private void butnUpdate_Click(object sender, EventArgs e)
{
    if (tBoxClientId.Text == "")
    {
        MessageBox.Show("请选择记录!");
    }
    else
    {
        mark = 2;// 设置 mark=1,表示执行修改操作
        cBoxAreaBound();
        butnAdd.Enabled = false;
        butnDelete.Enabled = false;
        tBoxClientId.Enabled = false;
        groupBox1.Visible = true;
    }
}
```

(6)“删除”按钮的 Click 事件中关键代码如下：

```
private void butnDelete_Click(object sender, EventArgs e)
{
    if (tBoxClientId.Text == "") // 判断是否选择删除记录
    {
```

```
            MessageBox.Show("请选择记录");
        }
        else
        {
            string sqldelete = "";
            if (myClientManage.myCon.State == ConnectionState.Closed)
            { myClientManage.dbCon(); }
            if (MessageBox.Show("是否删除该记录","提",
                MessageBoxButtons.OKCancel) == DialogResult.OK)
            {
                sqldelete = "delete from Client_Info   where ClientID='" + tBoxClientId.Text.Trim() + "'";
                myClientManage.SqlExecute(sqldelete); //执行 SQL 语句,实现删除功能
                MessageBox.Show("删除成功!");
                dvClientInfoBound();//调用函数
                clear();
                groupBox1.Visible = false;
            }
        }
    }
```

(7)“确定”按钮的 Click 事件中关键代码如下：

```
private void butnSure_Click(object sender, EventArgs e)
{
    if (myClientManage.myCon.State == ConnectionState.Closed)
    { myClientManage.dbCon(); }
    string sqlstring = "";
    if (mark == 1) //如果 mark == 1,表示执行添加操作
    {
        butnDelete.Enabled = true;
        butnUpdate.Enabled = true;
        if (tBoxClientId.Text == "")
        {
            MessageBox.Show("客户名称不能为空!", "提示", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
        }
        else
        {
            // 验证输入格式
            if (! myClientManage.validatePhone(tBoxClientTel.Text.Trim()))
            {
                errorCEmail.Clear();
                errorCPhone.SetError(tBoxClientTel, "电话号码格式不正确");
```

```
            }
            else if (! myClientManage.validateEmail(tBoxClientEmail.Text.Trim()))
            {
                errorCPhone.Clear();
                errorCEmail.SetError(tBoxClientEmail, "E-mail 地址输入格式不正确");
            }
            else
            {
                errorCPhone.Clear();
                errorCEmail.Clear();
                sqlstring = "insert into
                Client_Info(ClientId,ClientName,ClientGrade,ClientArea,ClientTel,
                ClientEmail," + "ClientAdress,ClientRemark) values('" +
                tBoxClientId.Text.Trim()+ "','" + tBoxClientName.Text.Trim() + "','" +
                cBoxGrade.Text.Trim() + "','" + cBoxArea.Text.Trim()+ "','" +
                tBoxClientTel.Text.Trim() + "','" + tBoxClientEmail.Text.Trim() + "','" +
                rBoxClientAdress.Text.Trim()+ "','" + rBoxClientRemark.Text.Trim() + "')";
                myClientManage.SqlExecute(sqlstring);
                MessageBox.Show("添加成功!");
                dvClientInfoBound();
              }
        }
    }
    else if (mark == 2)// mark == 1,表示执行添加操作
    {
        butnAdd.Enabled = true;
        butnDelete.Enabled = true;
        if (tBoxClientId.Text == "")
        {
            MessageBox.Show("客户名称不能为空!", "提示", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
        }
        else
        {
            if (! myClientManage.validatePhone(tBoxClientTel.Text.Trim()))
            {
                errorCEmail.Clear();
                errorCPhone.SetError(tBoxClientTel, "电话号码格式不正确");
            }
            else if (! myClientManage.validateEmail(tBoxClientEmail.Text.Trim()))
            {
                errorCPhone.Clear();
```

```
                errorCEmail.SetError(tBoxClientEmail, "E-mail 地址输入格式不正确");
            }
            else
            {
                sqlstring = "update Client_Info set ClientName='" +
                tBoxClientName.Text.Trim()+ "',ClientGrade='" + cBoxGrade.Text.Trim()
                + "',ClientArea='" + cBoxArea.Text.Trim() + "',ClientTel='" +
                tBoxClientTel.Text.Trim() + "',ClientEmail='" +
                tBoxClientEmail.Text.Trim() + "',ClientAdress='"+
                rBoxClientAdress.Text.Trim() + "',ClientRemark='" +
                rBoxClientRemark.Text.Trim() + "'where ClientID='"+
                tBoxClientId.Text.Trim() + "'";
                myClientManage.SqlExecute(sqlstring);
                MessageBox.Show("修改成功!");
                dvClientInfoBound();
            }
        }
    }
}
```

(8)工具栏“查找”按钮的 Click 事件中关键代码如下：

```
private void toolStripButton1_Click(object sender, EventArgs e)//查询
{
    groupBox1.Visible = true;
    //如果查询条件为“"客户名称”,定义相对应的模糊查询 SQL 语句,然后执行 SQL 语句并且把执行
    结果显示在 dvClientInfo 控件中。
    if (toolStripComboBox1.Text.Trim() == "客户名称")//条件查询——客户名称
    {
        string mySql = "SELECT  ClientId as 用户编号, ClientName as 用户名称,
        ClientGrade as 用户级别, ClientArea as 用户区域, ClientTel as 用户电话,
        ClientEmail as 用户邮箱, ClientAdress as 用户地址, ClientRemark as 备注
        FROM    Client_Info WHERE (ClientName LIKE '%[" +
        toolStripTextBox1.Text + "]%')";
        myds = myClientManage.Sqldataset(mySql);//填充数据集
        if (myds.Tables[0].Rows.Count > 0)//判断是否存在记录
            dvClientInfo.DataSource = myds.Tables[0].DefaultView;
        else
            MessageBox.Show("您输入的客户名称不对,没有相关记录");
    }
    //如果查询条件为“"客户级别”,定义相对应的模糊查询 SQL 语句,然后执行 SQL 语句并且把执行
    结果显示在 dvClientInfo 控件中。
    if (toolStripComboBox1.Text.Trim() == "客户级别")
```

```
    {
        string mySql = "SELECT    ClientId as 用户编号, ClientName as 用户名称,
        ClientGrade as 用户级别, ClientArea as 用户区域, ClientTel as 用户电话,
        ClientEmail as 用户邮箱, ClientAdress as 用户地址, ClientRemark as 备注
        FROM      Client_Info WHERE (ClientGrade LIKE '%[" + toolStripTextBox1.Text + "]%')";
        myds = myClientManage.Sqldataset(mySql);
        if (myds.Tables[0].Rows.Count > 0)
            dvClientInfo.DataSource = myds.Tables[0].DefaultView;
        else
            MessageBox.Show("您输入的客户级别不对,没有相关记录");
    }
    //如果查询条件为""客户区域",定义相对应的模糊查询 SQL 语句,然后执行 SQL 语句并且把执行
    结果显示在 dvClientInfo 控件中。
    if (toolStripComboBox1.Text.Trim() == "客户区域")
    {
        string mySql = "SELECT    ClientId as 用户编号, ClientName as 用户名称,
        ClientGrade as 用户级别, ClientArea as 用户区域, ClientTel as 用户电话,
        ClientEmail as 用户邮箱, ClientAdress as 用户地址, ClientRemark as 备注
        FROM      Client_Info WHERE (ClientArea LIKE '%[" + toolStripTextBox1.Text + "]%')";
        myds = myClientManage.Sqldataset(mySql);
        if (myds.Tables[0].Rows.Count > 0)
            dvClientInfo.DataSource = myds.Tables[0].DefaultView;
        else
            MessageBox.Show("您输入的客户级别不对,没有相关记录");
    }
}
```

(9)工具栏“全部”按钮的 Click 事件中关键代码如下：

```
private void toolStripButton2_Click(object sender, EventArgs e)/
{
    //定义查询全部信息的 SQL 语句
    string mySql = "SELECT    ClientId as 用户编号, ClientName as 用户名称,
    ClientGrade as 用户级别, ClientArea as 用户区域, ClientTel as 用户电话,
    ClientEmail as 用户邮箱, ClientAdress as 用户地址, ClientRemark as 备注
    FROM      Client_Info ";
    myds = myClientManage.Sqldataset(mySql);
    if (myds.Tables[0].Rows.Count > 0)
        dvClientInfo.DataSource = myds.Tables[0].DefaultView; //显示数据
    else
        MessageBox.Show("无记录");
}
```

(10)“查询”按钮的 Click 事件中的代码主要是分支语句，比较繁琐，具体代码参照随书

案例。

5.4.4　客户资料查询

客户资料查询窗体和客户资料管理窗体是同一个窗体，在这里不做具体介绍。

5.4.5　客户区域管理

客户区域管理模块是用 LINQ 技术实现的，具体实现步骤参照第 9 章内容。

5.4.6　资料分析

资料分析模块是用水晶报表技术实现的，具体实现步骤参照第 6 章内容.

5.4.7　用户资料

用户资料模块与客户资料管理模块知识点重复，具体实现步骤参照客户资料管理模块。

5.4.8　系统维护

1.系统维护模块概述

系统维护模块主要是对系统执行备份与还原。运行结果如图 5-34 和图 5-35 所示。

图 5-34　备份界面

图 5-35　还原界面

2.系统维护模块的实现

系统维护模块实现步骤如下：

(1)添加 2 个 Windows 窗体，分别命名为 frmDataRevert.cs 和 frmDataStore.cs，具体窗体设计参照第 4 章内容。

(2)“数据备份”按钮的 Click 事件中关键代码如下：

```
private void btnDStore_Click(object sender, EventArgs e)
{
    try
    {
        if (File.Exists(txtDSPath.Text.Trim() + ".bak"))
        {
            MessageBox.Show("该文件已经存在!", "提示", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
            txtDSPath.Text = "";
            txtDSPath.Focus();
        }
        else
        {
            if (txtDSPath.Text == "")
            { MessageBox.Show("请选择路径!"); }
            else
            {
                //执行数据备份的 SQL 语句
                myDataStore.dbRead("backup database Neusoft_CRM to disk='" +
                txtDSPath.Text.Trim() + ".bak'");
                MessageBox.Show("数据备份成功!", "提示",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "提示", MessageBoxButtons.OK, MessageBoxIcon.
Information);
    }
}
```

(3)“数据还原”按钮的 Click 事件中关键代码与“数据备份”代码类似，只是执行的 SQL 语句不通，这里不详细介绍。

5.4.9 日志

日志模块是用文件操作技术实现的，具体实现步骤参照第 7 章内容.

5.4.10 工具和帮助

工具和帮助模块比较简单，具体实现参照随书案例。

5.5　技术拓展

5.5.1　数据库基本知识

所谓数据库(Database，DB)，其实就是存放在计算机的外存储器中的相关数据的集合，可以形象地看作是数据的“仓库”，它是通过文件或类似于文件的数据单位组织起来的。数据库只是数据的集合，建立数据库的目的是为了使用数据库，为了对数据库中的数据进行存取，必须使用数据库管理系统(Database Management System，DBMS)。数据库管理系统是对数据进行管理的软件，使一个数据库系统的核心，数据库的一切操作，包括数据库的建立和数据的检索、修改、删除等操作，都是通过数据库管理系统来实现的。数据库管理系统只提供数据的管理功能，为了实现某种具体操作，必须要有相应得数据库应用程序。正是由于数据库应用程序的不同，才使数据库应用丰富多彩。

所谓数据库系统是指实际可运行的，按照数据库方式存储、维护和向应用系统提供数据或信息支持的计算机系统。是在计算机系统中引进数据库后的系统构成，一个完整的数据库系统还应包括数据库管理员。

因此一个完整的数据库系统由数据库、数据库管理系统、数据库应用程序、计算机软件和硬件系统及数据库管理员组成。

1.数据模型及关系数据库

数据库中的数据按照一定的数据模型组织，数据模型是把现实世界转换为计算机能够处理的数据世界的桥梁。目前常用的数据模型有三种：层次模型、网状模型和关系模型。最常用的是关系模型。

关系模型中，数据被组织成若干张二维表的结构，每一张二维表称为一个关系或表。表中的一行称为一个元组，在计算机中存放称为记录，表中的一列称为属性。

表的性质：

(1)表中的每一列均不可再分。

(2)表中的每一列数据的数据类型是相同的。

(3)表中两列不能取相同的名字。

(4)表中不允许有完全相同的两行，即任二行记录必须能够区分。

在关系数据库管理系统中，要建立一个数据库通常经过以下几个步骤：一是建立数据库文件，数据库文件通常用来容纳各种表；二是建立表结构，即建立表的各个字段的字段名，字段类型，字段宽度等；三是向表中录入实际的数据。

在 C# 中使用 ADO.NET 访问数据库。

2.存储过程

存储过程是指将常用的或复杂的数据库操作，预先用 SQL 语句写好并用一个指定的名称存储起来，以后需要完成与已定义好的存储过程的功能相同的数据库操作时，只需调用存储过程的名称即可。和数据库中的表一样，存储过程也被保存在数据库中。

存储过程具有以下优点：

一是存储过程编辑器事先对存储过程进行了语法检查处理，避免了因 SQL 语句语法不正确引起运行时出现异常的问题；二是只在保存存储过程时数据库服务器才进行编译，以后每次执行存储过程都不需要再重新编译。而一般的 SQL 语句每执行一次就需要数据库引擎重新编译一次，所以使用存储过程可提高数据库执行的效率；三是可以在定义存储过程的时候直接检查运行结果是否正确，可视化的设计界面提高了开发效率；四是避免了查询字符串中包含单引号等特殊符号可能会出现的问题；五是一个项目中可能会多处用到相同的 sql 语句，使用存储过程便于重用；六是修改灵活方便，当需要修改完成的功能时，只需要修改定义的存储过程即可，而不必单独修改每一个引用。

对于不能直接使用表的情况，或者具有复杂查询功能以及事务处理等情况，使用存储过程比较合适。

即使只有一条查询语句，也一样可以使用存储过程。

下面介绍存储过程的基本知识。

(1)存储过程的定义。

定义存储过程的常用形式为：

CREATE PROCEDURE [拥有者.]存储过程名

[(参数＃1,……,参数＃1024)]

AS 程序行

其中存储过程名不能超过 128 个字。每个存储过程中最多可以设定 1024 个参数。参数的常用形式为：

@参数名 数据类型 [＝默认值] [OUTPUT]

每个参数名都以“@”符号开头，每一个存储过程的参数只能在该存储过程内使用，参数的类型为 SQL Server 所支持的数据类型。

[＝默认值]指在调用存储过程时事先给一个默认的值，可以为每个参数设定一个默认值。[OUTPUT]指该参数既可用于输入又可用于输出，也就是说，如果所指定的参数值是需要输入的参数，同时也需要在结果中输出的，则该项必须为 OUTPUT。

(2) 存储过程内常用的基本语句。

①声明局部变量。

局部变量只有在声明变量的存储过程内才有效。每个局部变量都必须用 DECLARE 语句定义。每个变量的定义中都必须包含变量名和相应的数据类型。变量名以“@”符号开头。例如：

```
DECLARE   @studentName nvarchar(20)
```

②存储过程中常用的 SQL 语句。

a. IF 语句的一般形式：

```
IF 条件
  BEGIN
    SQL 语句系列
  END
```

```
ELSE
  BEGIN
    SQL 语句系列
  END
```

b. WHILE 语句一般形式：

```
WHILE 条件
    BEGIN
      SQL 语句系列
      [BREAK]
      SQL 语句系列
      [CONTINUE]
      SQL 语句系列
    END
```

c. SET 语句用于对一个局部变量赋值。例如：

```
SET @studentName='张三'
```

(3)编辑和调用存储过程。

在数据库中新建存储过程的语句如下：

```
CREATE PROCEDURE delete_1
    @ClientId nvchar(20)
AS
BEGIN
    delete from Client_Info where ClientId=@ClientId;
END
GO
```

例题 5_09：修改例题 5_02 中代码。

例题 5_02 中代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    cmd.CommandText = "delete from Client_Info where ClientId=" + textBox1.Text;
    cmd.CommandType = CommandType.Text;
    cmd.Connection = conn;
    cmd.ExecuteNonQuery();
    MessageBox.Show("已删除该记录!");
}
```

代码修改为：

```
private void button1_Click(object sender, EventArgs e)
{
```

```
        cmd.CommandText = "delete_select_1";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = conn;
        cmd.ExecuteNonQuery();
        MessageBox.Show("已删除该记录!");
    }
```

5.5.2 异常处理

1.异常的基本概念

简单的说,异常处理是一种处理.NET 程序在运行过程中产生的错误的机制。如果没有异常处理机制,可能会导致应用程序因发生错误而提前退出。

例题 5_10:除法运算。

【问题分析】通过 try…catch 来捕获程序异常。

【程序说明】

(1)运行 Visual Studio 2010,创建一个名为 Divide 的 windows 窗体应用程序项目。

(2)添加三个 Label 控件、三个 TextBox 控件和一个 Button 控件,更改 Label 控件和 Button 控件的 Text 值设置结果如图 5-36 所示。

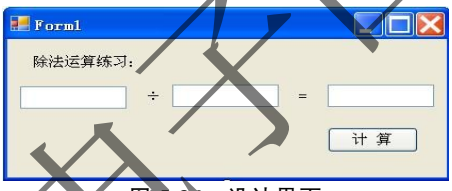

图 5-36 设计界面

(3)双击“计算”按钮进入 Click 事件,修改 Form1.cs 代码如下:

```
private void button1_Click(object sender, EventArgs e)
{
    double x, y, z;
    x = double.Parse(textBox1.Text);
    y = double.Parse(textBox2.Text);
    z = x / y;
    string str = string.Format("{0:F2}",z);
    textBox3.Text = str;
}
```

当输入如图 5-37 所示不符合要求的数据时,会出现如图 5-38 所示异常。

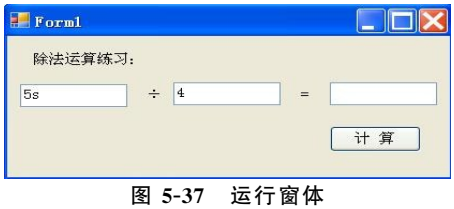

图 5-37 运行窗体

图 5-38　异常报错界面

解决这个问题的一种方法，就是对输入数据进行合法性检查。修改代码如下：

```
private void button1_Click(object sender, EventArgs e)
{
    double x, y, z;
    try
    {
        x = double.Parse(textBox1.Text);
        y = double.Parse(textBox2.Text);
        z = x / y;
        string str = string.Format("{0:F2}", z);
        textBox3.Text = str;
    }
    catch (FormatException e1)
    {
        MessageBox.Show(e1.Message);
        textBox1.Text = "";
        textBox2.Text = "";
        textBox1.Focus();
    }
}
```

【运行结果】

按＜F5＞，运行结果如图 5-39 所示。

图 5-39　进行异常处理后的运行结果

2.C＃和.NET 中的异常

为了解决上述问题，.NET 提出了一种解决方案——异常。程序设计人员可以定义一个异

常，在出现不期望发生的事件的情况下抛出该异常。下面是 C# 和. NET 中，关于异常和处理的几点重要说明：

在. NET 中，所有的异常都是对象。. NET 中的异常共有一个基类，就是 System. Exception 类。在程序执行过程中，所有的方法都可以通过使用 C# 中的关键字 throw，在发生不期望事件的时候抛出相应的异常。在外部调用方法中，可以使用 try…cath 结构获取和处理抛出的异常。

那些可能会抛出需要处理的异常的代码将放在 try 语段中，被称为试图捕获异常。那些对所出现的异常进行处理的代码会被放在紧邻 try 语段之后的 catch 语段中，被称为捕获异常。可以使用关键字 catch 加异常名，用来指定在 catch 语段中将处理的特定的异常类。此外，对一个 try 语段可以同时有多个 catch 语段用来捕获其抛出的异常，每个 catch 语段处理的一种特定的异常类。

在 try 语段或者 try…catch 结构后还可以跟随 finally 语段，该语段的作用是用来放置那些必须被执行的代码，而不会受到是否产生异常的影响。

在程序执行过程中，如果没有异常发生，那么 try 语段内发生异常的语句后的代码将不会被执行，程序将执行有效的 catch 语段或者 finally 语段内的代码。

因为异常在. NET 中是以类和对象的形式实现的，所以它也遵循继承的原则。也就是说，如果有一个 catch 语段是用来处理基类异常，那么它就自动地可以处理所有该基类的子类异常。如果在捕获父类异常之后，又尝试捕获该父类异常的子类异常，那么将会引起编译器报错。

Finally 语段是可选的，异常的处理可以通过使用 try…catch、try…catch…finally 或者 try…finally 3 种结构中的任意一种。

3. 多异常的捕获

有的时候，try 语段的代码在执行时可能会抛出多个不同的异常，这就需要有一系列的 catch 语段用来捕获这些异常。

例如：

```
private void button1_Click(object sender, EventArgs e)
{
    double x, y, z;
    try
    {
        x = double.Parse(textBox1.Text);
        y = double.Parse(textBox2.Text);
        z = x / y;
        string str = string.Format("{0:F2}", z);
        textBox3.Text = str;
    }
    catch (FormatException e1)
    {
        MessageBox.Show(e1.Message);
        textBox1.Text = "";
        textBox2.Text = "";
        textBox1.Focus();
    }
```

```
        catch (Exception e2)
        {
            MessageBox.Show(e2.Message);
            textBox1.Text = "";
            textBox2.Text = "";
            textBox1.Focus();
        }
    }
```

第 1 个用来捕获 FormatException 异常，第 2 个捕获 Exception 异常，其中 FormatException 异常是由于输入数据格式不正确产生的异常。

最后一个 catch 语段设计成可以捕获除了 FormatException 异常外的其他异常。

5.6　本章小结

本章主要讲解了客户关系管理系统数据库操作的实现部分。在技术准备里首先介绍了 ADO.NET 技术，告诉大家什么是 ADO.NET 以及 ADO.NET 的优点；然后重点讲解了数据访问对象知识点，告诉大家 ADO.NET 技术是如何操作数据库的；最后讲了几个数据绑定控件，给大家提供一个简单的、不需要写太多代码的数据绑定技术。

在项目实施里重点讲解了数据库的添加、修改、删除和查询等基本操作。

在技术拓展里首先介绍了数据库的基本知识，然后讲解了异常处理的基本概念和简单应用。

5.7　强化练习

1.填空题

(1)ADO.NET 是一组向 .NET 程序员公开数据访问服务的_____。

(2)ADO.NET 具有_____、_____、_____、性能优化、可伸缩性等优点。

(3)使用_____对象与数据源建立连接后，可使用 Command 对象来对数据源执行查询、插入、删除、更新等各种操作。

(4)DataReader 对象允许用户获得从 Command 对象的_____语句得到的结果。

(5)_____对象是 DataSet 对象和数据源之间联系的桥梁，主要功能是从数据源中检索数据。

(6)Windows 窗体提供了两种类型的数据绑定：_____与_____。

2.简答题

(1)Connection 类的主要属性和及其作用是什么？

(2)DataAdapter 类的主要功能是什么？