第 7 章 网络点餐管理系统项目开发

7.1 案例说明

网络点餐管理系统是一个基于 C#语言开发，使用 SQL Server 数据库、采用 C/S 架构的完整的开发项目，该项目综合了全书各章知识点，包括基础语法、控件编程和界面设计、文件操作、数据库操作、数据报表以及安装包制作等内容，是学习本课程内容后的一次实战演练，也是提高开发技能的一个综合性实训项目。

7.2 开发背景

随着科技的发展使得人们的在生活和工作上处理各种事务更加便捷，如使用电脑或手机实现网购快递、点餐外送、二维码支付、行程预定…等等，这一切都体现了科技的进步带给人们在的衣、食、住、行等各个方面的便捷性，而软件的开发和应用使这些便捷性得到强有力的支撑。现在各行业基本上都有相应的信息管理系统，而餐饮行业中，大多数顾客还是按照传统方式等候服务员的点餐服务，经营者还是手工收集顾客点餐信息进行结账…。如果能为餐饮行业设计一套网络点餐系统，让顾客能够直接在餐台的终端下单；厨师能够通过在厨房终端显示的点餐单为顾客制作菜品；收银员能够在服务台的终端进行结账等，这种智能化的系统将让顾客既能体验科技感又能感受到消费的便捷性，餐厅经营者为此将大大降低人力物力等运营成本，同时又能提高服务效率。本项目正是基于当前餐饮行业的发展状况和业务模式，同时为提高顾客消费便捷性，以及提高餐饮行业经营者的服务效率和降低成本等因素而开发。

课程设计参考 需求分析

通过社会调研以及对市面同类型软件的分析得知，顾客对点餐系统界面的最大要求是简洁明了，操作便捷和人性化，无须专门去学习操作技能；根据顾客行为习惯，顾客通常是根据菜品分类、菜品图片和价格来决定是否消费。因此，为适应用户的操作习惯并参考普通餐厅的纸质点菜单，本项目的顾客点餐界面将首先分类展示菜品的图片和价格，顾客可以随时切换菜品来查看菜品信息，然后通过单击菜品图片来实现点餐功能。

对于前台界面显示的所有信息，后台系统都需要预先录入，包括菜品分类、菜品信息、菜品图片和价格等相关的基础数据，因此，后台系统必须能够提供对这些基础数据进行维护的界面。同时，为了能够在后台系统显示当前餐台的消费情况，需要对每张餐台的信息进行收集和管理，使经营者可以对当前餐台的就餐状态一目了然，最后自动结账并打印消费清单。此外，为帮助餐厅经营者了解营业情况，后台系统需要根据时间段对营业额和利润进行统计并形成数据报表。

由此可知，需要开发前台系统，提供给顾客使用，以便采集顾客消费信息；同样，需要开发提供给餐厅经营者使用的后台系统，用于处理顾客消费信息，并对基础数据进行维护。由于本书是介绍 C# WinForm 程序设计的课程，因此本项目采用 Client/Server(又称客户端/服务端、或是前端/后端，简称 C/S)架构进行开发。客户端用于采集用户操作数据提交到数

据库服务器，服务端对数据库服务器的数据进行处理和维护。

而当我们以后学习了后续课程“ASP.NET 动态网页设计”后，可以将本项目改为采用基于浏览器的 Browser/Server(简称 B/S，即网站)架构进行开发。B/S 架构的优点在于，客户端无需独立安装，只要安装了浏览器即可操作，所以又称“瘦客户端”。

基于.NET 语言开发 WinForm 应用程序，数据库一般采用与微软公司的产品无缝对接的 Access 或 SQL Server。Access 是小型桌面数据库，是 Office 应用程序中的套件之一，一般用于没有服务端的桌面应用程序(也叫单机版应用程序)，而网络应用程序基本都采用 SQL Server 数据库。SQL Server 是大型网络数据库，其安全、稳定、可靠。

基于以上分析，本项目采用 Client/Server 架构、使用 C#语言，选择 SQL Server 数据库进行开发，前台客户端用于采集用户操作数据，后台服务端用于对数据进行处理和维护，两者作为独立的项目分别进行开发。

7.4 项目概况

1. 功能结构图

网络点餐管理系统的功能结构图如图 7.1 所示。

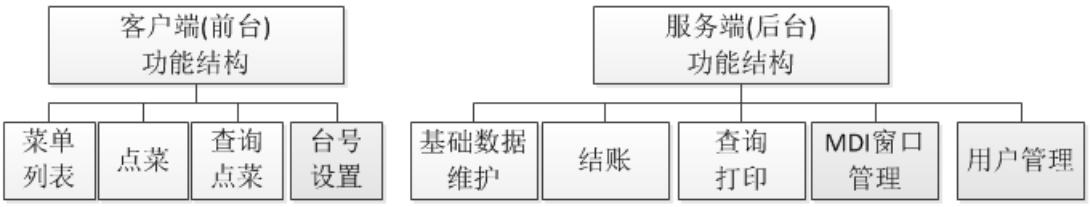
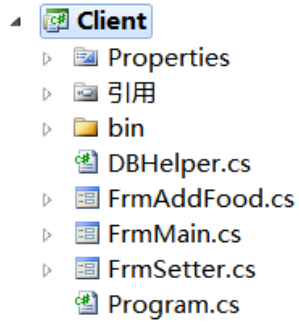


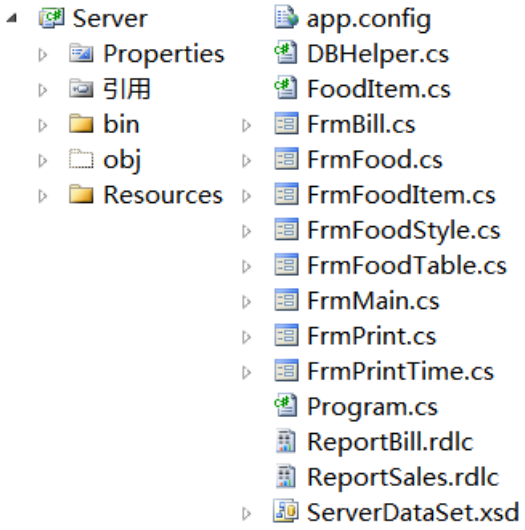
图 7.1　客户端和服务器端功能结构图

2. 项目文件清单

为对网络点餐管理系统有个大致了解，先给出项目文件清单，并对文件清单中各个文件的用途作出简要的说明。客户端项目文件清单如图 7.2(a)所示，服务器端项目文件清单如图 7.2(b)所示。



(a) Client 项目文件清单



(b) Server 项目文件清单

图 7.2 客户端和服务器端项目文件清单

(1) 客户端项目 Client 文件清单说明。

- DBHelper.cs：公共类文件，在 Client 项目实现对数据库所有操作。
- FrmMain.cs：主窗体文件，提供顾客点餐界面，实现点餐操作。
- FrmAddFood.cs：具体点餐窗体文件，实现将菜品项添加到点餐单。
- FrmSetter.cs：设置窗体文件，用于设置餐台编号，同时限制顾客擅自退出程序。

(2) 服务端项目 Server 文件清单说明。

- DBHelper.cs：公共类文件，在 Server 项目实现对数据库所有操作。
- FoodItem.cs：菜品实体类文件，用于保存 FoodItem 表的数据。
- FrmMain.cs：主窗体文件，提供操作菜单和工具栏，实现系统其他操作的接口。
- FrmFoodStyle.cs：菜品分类数据维护窗体文件，实现菜品分类信息的维护。
- FrmFood.cs：菜品数据维护窗体文件，实现对菜品信息的增、删、改、查操作。
- FrmFoodItem.cs：菜品修改和添加窗体文件，实现菜品项信息的添加和修改。
- FrmFoodTable.cs：餐台数据维护窗体文件，实现为餐台编号和命名。
- FrmBill.cs：结账窗体文件，实现顾客结账和查询账单、打印小票功能。
- FrmPrint.cs：报表统一打印窗体文件。
- FrmPrintTime.cs：打印条件设置窗体文件。
- ReportBill.rdlc：消费小票报表文件。
- ReportSales.rdlc：营业额报表文件。
- ServerDataSet.xsd：项目数据集设计文件。

(3) 系统整体操作

系统的整体操作流程如图 7.3 所示。

- 客户端：客户端启动后，顾客可以浏览菜品，确认就餐时，执行“开单点菜”操作，然后选择感兴趣的菜品并确认数量，进入就餐状态，最后执行“结账”操作，结束就餐过程。
- 服务端：服务器启动后，管理员可以编辑已有菜品数据，或者增加新菜品数据，当接收到顾客结账信息时，统计消费金额并打印消费清单。

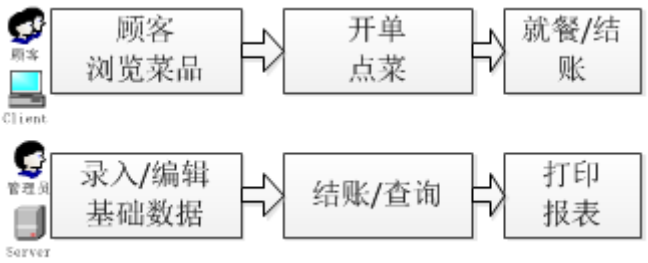


图 7.3　操作示意图

对于本项目的学习，建议读者下载本项目示例代码，熟悉操作流程后，根据操作逻辑，再学习代码的具体实现。

7.5 数据库设计

网络点餐管理系统使用的数据库的是在第 6 章“6.1.3 网络点餐管理系统数据库设计”节中介绍的 Restaurant 数据库，Restaurant 数据库的设计过程已在该节有详细的介绍，请读者务先必熟悉 Restaurant 数据库中各个表的具体结构和表之间的关系。

需要指出的是，数据库设计的规范程度将关系到代码编写的复杂程度，数据库设计范式等级越高，关系就越多，数据表也将越多，代码编写越复杂；反之，数据库范式等级越低，数据冗余度越高，代码相对简单，两者是一对矛盾体，需要根据实际经验取得平衡，这就要求读者在平时多学，多练，在实践中总结经验。

课程设计参考-主要功能模块

7.6.1 公共类设计

由于对数据表的基本操作是查询、添加、修改和删除数据，只要给定了 SQL 语句，这些操作都可以通过 ADO.NET 类库中的 SqlCommand 对象的 Execute 开头的相关方法来实现，这些方法区别在于是否返回查询结果，同时由于在类中不需要保存具体操作数据，因此将这些对数据库的基本操作封装为静态类 DBHelper。静态类的所有的属性和方法都必须是静态的，即都需要使用 static 关键字修饰。另外一种可以更便捷取得查询结果的方式是使用数据集，因此 DBHelper 类也提供了取得数据集的方法，总之，DBHelper 类封装了项目中对数据库的所有的操作功能。

Client 项目和 Server 项目都分别包含了 DBHelper 类文件，除了 DBHelper 类在两个项目中的命名空间不一样，其他内容完全一样。该类虽然可以作为单独项目编译为 DLL 文件，然后在 Client 和 Server 项目分别添加对该文件的引用，而无须将 DBHelper 类的源代码分别包含在项目中，但由于本书没有介绍 DLL 文件的制作内容，也为了方便独立介绍，所以采用类文件的方式分开使用，即 Client 项目和 Server 项目都包含内容一样的 DBHelper 类文件。以下是 DBHelper 类文件的完整代码。

程序代码 – DBHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
namespace Client
{
   public static class DBHelper
   {
      //1.连接字符串字段和私有连接对象字段
      public static String ConnString = "Data Source=(local);Initial Catalog=Restaurant;
      Integrated Security=true";
      static SqlConnection _conn = null;//私有字段

      //2.连接对象属性
      public static SqlConnection Conn//公共只读属性，取得连接对象
      {
```

```
        get
        {
            try
            {
                //如果连接对象没有创建，或已关闭，则创建并打开
                if (_conn == null || _conn.State == ConnectionState.Closed)
                {
                    _conn = new SqlConnection(ConnString);
                    _conn.Open();
                }
                return _conn;
            }
            catch (Exception ex)
            {
                System.Windows.Forms.MessageBox.Show("连接发生错误，原因如下：\r\n"
                + ex.Message);return null;
            }
        }
    }
```

//3.关闭连接对象

```
    static void Close()
    {
        if (_conn != null)
        {
            _conn.Close();
            _conn = null;
        }
    }
```

//4.返回SqlDataReader对象，仅用于执行Select查询语句

```
    public static SqlDataReader GetDataReader(string Sql)
    {
        if (Conn == null) return null;
        SqlCommand comm = new SqlCommand(Sql, Conn);
        return comm.ExecuteReader(System.Data.CommandBehavior.CloseConnection);
    }
```

//5.用于执行Update/Insert/Delete语句，并返回操作所影响的记录数

```
    public static int ExecSql(string Sql)
    {
        if (Conn == null) return -1;//返回-1表示当前连接对象没有创建
        SqlCommand comm = new SqlCommand(Sql, Conn);
        int num = comm.ExecuteNonQuery();
        Close();
        return num;//返回因添加、删除、修改操作而受影响记录数
    }
```

```
    //6.返回单值查询结果
    public static object GetOne(string sql)
    {
        if (Conn == null) return null;
        SqlCommand comm = new SqlCommand(sql, Conn);
        object obj = comm.ExecuteScalar();//返回单值查询结果
        Close();
        return obj;//返回object类型，使用时需要转换为实际类型
    }
    //7.取得数据集
    public static DataSet GetDataSet(string Sql, string TableName)
    {
        if (Conn == null) return null;
        SqlDataAdapter DA = new SqlDataAdapter(Sql, Conn);//接收已打开的连接对象
        DataSet ds = new DataSet();
        DA.Fill(ds, TableName);
        Close();//必须关闭连接对象
        return ds;
    }
  }//DBHelper类结束
}//命名空间结束
```

代码说明

(1) 注释1：创建连接字符串connString，定义连接对象属性对应的私有字段_conn，由于是静态类，所有的属性和方法都是静态的，可以直接通过类名使用其方法和属性。

(2) 注释2：将连接对象设计为只读属性Conn，而不是设计为方法返回连接对象(当然也可以设计为方法)，主要为了方便使用，当取得的连接对象属性时，如果该属性为null，自动创建该对象；如果无法连接数据库，则提示用户，同时将该属性重新设置为null。其他方法使用到该属性时，需要判断该值是否为null，再决定是否执行其他操作。由于MessageBox类位于System.Windows.Forms命名空间，创建DBHelper类时并没有默认引用该命名空间，因此这里需要包含命名空间的全名称使用，也可以在using声明区引用该命名空间后直接使用。

(3) 注释 3：自定义方法 Close，用于关闭已经创建的连接对象。由于在注释 7 的位置使用了已打开的连接对象 Conn 来填充数据集，必须调用该方法关闭连接对象。

(4) 注释 4：自定义方法 GetDataReader，实现通过指定的 Select 查询语句，返回数据读取器对象 SqlDataReader，从而得到查询结果。ExecuteReader 有两个重载的方法，其中之一为可带枚举类型 System.Data.CommandBehavior 的参数，CloseConnection 枚举值表示在类外部关闭 SqlDataReader 对象时，自动关闭连接对象，从而不需要在类外部调用实现关闭连接对象的方法。

(5) 注释 5：自定义方法 ExecSql，实现执行指定的 SQL 语句，包括 Insert、Update 和 Delete 语句，返回操作所受影响的记录数。

(6) 注释 6：自定义方法 GetOne，实现执行指定的查询语句，返回包含单值结果。一般用于执行 SQL 语句中的计数、求和、求最大值或最小值以及求平均值等函数。

(7) 注释 7：自定义方法 GetDs，实现根据 Select 语句，返回数据集对象。注意，这里创建数据适配器对象时，必须手动关闭连接对象，而且数据集指定了表名，更新数据集时也

必须指定数据集中的表名。

(8) 在所有的自定义方法中都必须判断当前连接对象是否可用。

7.6.2 客户端设计

Client 项目主要包含分类显示菜品的主窗体 FrmMain 和具体点菜窗体 FrmAddFood。为使用系统更实用，为管理员增加了需要权限设置餐台编号和退出系统功能的窗体 FrmSetter，使得普通顾客用户不能随意关闭该系统。

7.6.2.1 主窗体 FrmMain 设计

主窗体主要用于分类显示菜品，以供顾客选择。由于菜品的类别在是后台动态添加的，可能会经常更新，因此，前台主界面需要根据后台菜品的类别，动态创建用于显示菜品类别的操作按钮，单击不同的按钮将显示不同类别的菜品列表，为了美观，所有的按钮都采用平面样式，实际效果如图 7.4 所示。

图 7.4 客户端主界面运行效果

界面布局

为 Client 项目添加窗体，保存为 FrmMain。在窗体中添加 1 个 Panel 控件使其停靠在底部；在 Panel 中添加 1 个标签和 3 个按钮，标签用于显示餐台编号；再向窗体添加 2 个 FlowLayoutPanel 控件，1 个停靠在窗体顶部，并为其添加 1 个按钮，作为固定显示“推荐菜品”的操作按钮；另 1 个 FlowLayoutPanel 控件填充窗体剩余的空间，作为动态创建的标签的容器，这些标签用于显示菜品信息。所有的按钮设置为平面样式和“红底白字”的外观，界面布局和控件文本、控件名称如图 7.5 所示。

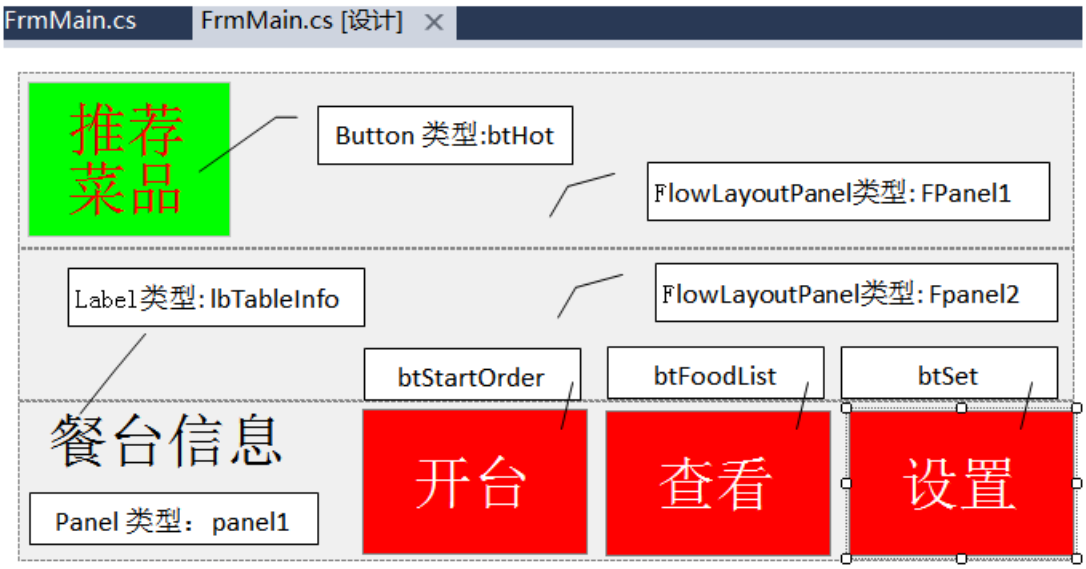


图 7.5　主窗体界面布局

程序代码

```
//省略自动添加的using部分
//1.添加命名空间引用
using System.IO;
using System.Data.SqlClient;
namespace Client
{
    public partial class FrmMain : Form
    {
        public FrmMain()  {  InitializeComponent(); }//构造方法

        //2.定义字段
        public string TableID = "";//当前餐台编号
        public int OrderID = -1;//当前餐台订单ID
        DataSet ds = null;//保存菜品信息的数据集

        //3.自定义方法：动态创建菜品类别按钮
        void ShowMenuStyle()
        {
            string sql = "Select * From FoodStyle";
            //调用静态类的方法，得到查询结果
            SqlDataReader dr = DBHelper.GetDataReader(sql);
            while (dr.Read())
            {
                Button bt = new Button();
                bt.Name = "bt" + dr["StyleID"];//设置动态按钮的名称
                bt.Text = dr["StyleName"].ToString();//设置按钮文本
```

```
            bt.ForeColor = Color.White;

            //设置动态按钮的字体、边界、尺寸、平面外观与存在的按钮btHot一样
            bt.Font = btHot.Font;
            bt.Margin = btHot.Margin;
            bt.Size = btHot.Size;
            bt.FlatStyle = btHot.FlatStyle;
            bt.FlatAppearance.BorderSize = btHot.FlatAppearance.BorderSize;
            bt.FlatAppearance.BorderColor = btHot.FlatAppearance.BorderColor;
            bt.FlatAppearance.MouseDownBackColor =
                    btHot.FlatAppearance.MouseDownBackColor;
            bt.FlatAppearance.MouseOverBackColor =
                    btHot.FlatAppearance.MouseOverBackColor;
            bt.Click += new EventHandler(bt_Click);
            FPanel1.Controls.Add(bt);//添加到流式布局面板
        }
        dr.Close();//关闭dr的同时，将自动关闭了数据库连接对象。
    }
    //4.菜品分类中动态按钮事件处理过程
    void bt_Click(object sender, EventArgs e)
    {
        Button bt = sender as Button;//转换为Buton类型以取得相关属性
        int StyleID = int.Parse(bt.Name.Replace("bt", "").ToString());//取得StyleID
        ShowFoodMenu(StyleID);//显示指定StyleID的菜品列表
}
//5.自定义方法：根据菜品类别编号StyleID，动态创建标签控件，显示菜品列表
void ShowFoodMenu(int StyleID = 0)
{
        Font ft = new System.Drawing.Font("黑体", 12, FontStyle.Bold);//创建标签字体
        FPanel12.SuspendLayout();//挂起控件，添加控件完毕一起显示
        FPanel12.Controls.Clear();//清空上一次显示的菜品列表信息标签

        //建立筛选条件，准备在数据集中筛选菜品
        string where="";
        if (StyleID > 0) where = " styleID=" + StyleID;//菜品类别
        else where = " IsHot=1";//推荐菜品

        DataRow[] dRows = ds.Tables[0].Select(where);//取得满足条件的数据行
        foreach (DataRow dr in dRows)
        {
            Label    lb = new Label();
            lb.Name = "ID" + dr["FoodID"];
            lb.Text = dr["FoodName"] + "\r\n" + "￥" + dr["price"];
            lb.Size = new Size(180, 220);
```

```
            //设置lb的click事件属性
            lb.Click += new EventHandler(lb_Click);

            //只有图片文件存在当前程序目录下，才显示
            if (File.Exists(Application.StartupPath + "\\Images\\" + dr["ImagePath"]))
            lb.Image = Image.FromFile(Application.StartupPath + "\\Images\\" +
            dr["ImagePath"]);

            //设置标签的外观属性
            lb.ImageAlign = ContentAlignment.TopCenter;
            lb.TextAlign = ContentAlignment.BottomCenter;
            lb.Cursor = Cursors.Hand; //显示手状光标
            lb.Margin = new System.Windows.Forms.Padding(6);//设置上、下、左、右边距
            lb.Font = ft;//设置字体
            lb.ForeColor = Color.White;
            FPanel12.Controls.Add(lb);//添加到容器控件
        }
        FPanel12.ResumeLayout();//子控件添加完毕确认显示
}
```

//6.菜品信息标签的事件组处理过程

```
void lb_Click(object sender, EventArgs e)
{
        if (OrderID <= 0)
        {
            MessageBox.Show("请先开台，再点菜！", "提示");
            return;
        }

        Label lb = sender as Label;//强制转换
        FrmAddFood frm = new FrmAddFood();
        frm.FoodName_Price = lb.Text;//菜品名称和价格
        frm.Img = lb.Image;//菜品图片

        //取得具体菜品的点菜数量
        if (frm.ShowDialog() == DialogResult.OK)
        {
            SaveFood(lb, frm.Count);//取得菜品数量并保存
        }
}
```

//7.自定义方法：根据菜品名称、价格、数量和订单ID，保存到订单明细表

```
void SaveFood(Label lb, int Count)
{
        string FoodID = lb.Name.Replace("ID", "");//去掉ID前缀
```

```
        int pos = lb.Text.IndexOf('￥');//查找子串
        string Price = lb.Text.Substring(pos+1);//取价格部分的子串

        string sql = "Insert Into OrderDetail(OrderID,FoodID,Price,Count) Values(";
        sql += OrderID + "," + FoodID + "," + Price + "," + Count;
        sql += ")";
        DBHelper.ExecSql(sql);
}
```

//8.自定义方法：初始化控件设置背景前景色，读取菜品信息

```
void Init()
{
        TableID =    Program.TableInfo.Split('|')[0];//从餐台信息字符串中拆分出餐台ID
        lbTableInfo.Text = Program.TableInfo;//显示餐台信息
        this.BackColor = Color.Black;
        lbTableInfo.ForeColor = Color.White;

        ds = DBHelper.GetDataSet("Select * From FoodItem", "FoodItem");//填充数据集
        if (ds == null)//如果无法连接数据库，则返回null，无法继续操作，退出本程序
        {
            MessageBox.Show("数据库连接失败，即将退出", "提示");
            Application.Exit();//退出应用程序，关闭所有打开的窗体(假如有)
        }
}
```

//9.窗体Load事件处理过程

```
private void Form1_Load(object sender, EventArgs e)
{
        Init();//初始化界面，读取菜品表所有数据并填充数据集
        ShowMenuStyle(); //动态创建菜品分类的操作按钮
        ShowFoodMenu(); //显示菜品列表，不带参数表示显示所有的推荐菜品
}
```

//10."开台"按钮事件处理过程

```
private void btStartOrder_Click(object sender, EventArgs e)
{
        //判断订单编号是否存在，存在则表示顾客已开台，进入就餐状态
        if (OrderID > 0)
        {
            MessageBox.Show("已开台，请不要重复开台！", "提示");
            return;
        }
        //餐台只有两种状态：已结账[空闲]/开台[未结账]
        string sql = "Select count(*) From [order] Where TableID=" + TableID +
          " AND status='开台'";
        int count = (int)DBHelper.GetOne(sql);//返回记录数
        if (count > 0)
```

```
    {
        MessageBox.Show("请稍候，正在等待上一位顾客结账！", "提示");
        return;
    }
    //添加开台信息，并得到返回的订单编号
    sql = "insert into [Order] (tableID,status) ";
    sql += " output inserted.OrderID ";
    sql += " Values(" + TableID + ",'开台')";
    OrderID = (int)DBHelper.GetOne(sql);//保存返回的订单编号
    MessageBox.Show("开台成功，可以开始点菜啦！", "开台成功");
}
```

//11."查看"按钮事件处理过程

```
private void btFoodList_Click(object sender, EventArgs e)
{
    If(OrderID<=0) return;//非就餐状态，退出
    //取得当前订单，所有相关菜品信息
    string sql = "select FoodItem.FoodName , OrderDetail.Price , OrderDetail.count ,
    OrderDetail.Price*OrderDetail.count as 小计 ";
    sql += " From OrderDetail,FoodItem
    where OrderDetail.FoodID=FoodItem.FoodID and OrderID=" + OrderID;
    SqlDataReader dr = DBHelper.GetDataReader(sql);

    string FoodInfo = "你点的菜有:\r\n";//保存点菜信息
    int pos = 1;//序号
    float Fee = 0f;// 当前费用
    while (dr.Read())
    {
        FoodInfo += pos + " : " + dr["FoodName"] + "    " + dr["price"] + "    " +
        dr["Count"] + "\r\n";
        Fee += float.Parse(dr["小计"].ToString());
        pos++;
    }
    dr.Close(); //关闭dr对象同时将自动关闭连接对象
    FoodInfo += "\r\n当前金额为： " + Fee;
    MessageBox.Show(FoodInfo, "提示");
}
```

//12."推荐菜品"按钮事件处理过程

```
private void btHot_Click(object sender, EventArgs e)
{
        ShowFoodMenu();
}
```

//13."设置"按钮事件处理过程

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

```
            FrmSetter frm = new FrmSetter();//打开系统的设置窗体
            frm.ShowDialog();
        }

    }//FrmMain类结束
}//命名空间结束
```

代码说明

(1) 注释3：自定义方法ShowMenuStyle，实现根据菜系表FoodStyle的记录数，动态生成分类按钮，单击不同按钮时，将显示不同类别的所有菜品。

① DBHelper.GetDataReader(sql)：该语句实现通过自定义静态类DBHelper的方法执行查询，并返回查询结果对象，然后根据记录数，动态创建按钮，最后添加到流式布局面板控件FPanel1中。

② 在动态创建按钮后，为使其外观统一，将其外观属性设置为与已存在的按钮btHot一致。按钮必须设置FlatStyle属性为平面样式后，其属性FlatAppearance才有效，该属性作用是使得鼠标移过或单击按钮时，自动更改按钮的外观颜色。

(2) 注释4：动态按钮的事件组处理过程。实现单击按钮时，取得保存在按钮Name属性中的StyeID，并作为方法ShowFoodMenu的参数。

(3) 注释5：自定义方法ShowFoodMenu，实现根据参数StyleID，从数据集中查询菜品数据，并根据菜品信息动态创建标签控件，添加到流式布局面板控件FPanel2中从而显示所有菜品的图文信息。容器控件的SuspendLayout方法是暂时挂起控件，不刷新界面，在子控件添加完毕后，通过ResumeLayout方法显示到界面，避免每次添加控件都进行一次刷新。

(4) 注释6：动态标签的Click事件处理过程，实现取得代表具体菜品的标签信息，传递到具体的点菜窗体FrmAddFood中显示，根据该窗体返回的点菜数量，调用SaveFood方法，保存到数据库的订单明细表DetailOrder。

(5) 注释7：自定义的SaveFood方法，实现根据订单编号，将顾客选择的菜品信息及数量保存到订单明细表DetailOrder。

(6) 注释8：自定义方法Init，实现设置窗体背景色并初始化界面控件状态，然后调用DBHelper静态类的方法，读取菜品表FoodItem所有的数据，填充到数据集。使用数据集的目的是为了在顾客切换不同菜品类别时，不需要反复连接数据库去查询数据，这是数据集的一个典型应用场景。

(7) 注释13:“设置”按钮的事件处理过程，实现单击该按钮时，打开设置窗体FrmSetter。设置窗体用于验证用户权限，使得系统管理员才有更改餐台编号信息和执行关闭程序操作的权限，设置窗体的具体功能实现参见本章7.6.2.4节。

7.6.2.2 点菜窗体 FrmAddFood

FrmAddFood 是实现具体点菜功能的窗体。在 FrmAddFood 中显示顾客当前选择的具体菜品信息(名称、价格和图片)，并取得顾客选择该菜品的数量。实现思路为：在 FrmAddFood 窗体中定义必要的公共字段，保存从主窗体 FrmMain 传递过来的菜品信息，在 FrmAddFood 使用模式对话框打开时，将菜品信息显示到其相应控件中；通过顾客在该界面的不同操作，设置窗体的返回值来自动关闭窗体。主窗体 FrmMain 根据 FrmAddFood 窗体关闭时的返回值

来判断用户是否执行了确认操作，从而取得用户选择的菜品数量，运行效果如图 7.6 所示。

图 7.6　点菜窗体的运行效果

界面布局

为 Client 项目添加窗体，保存为 FrmAddFodd。向窗体添加 1 个 PictureBox 控件用以显示菜品图片；添加 4 个操作按钮，分别实现数量调节和确认或取消操作；添加 2 个标签，用于显示菜品名称和静态说明文字；添加 1 个只读文本框控件，显示选择的菜品数量，界面布局、控件名称和文本如图 7.7 所示。

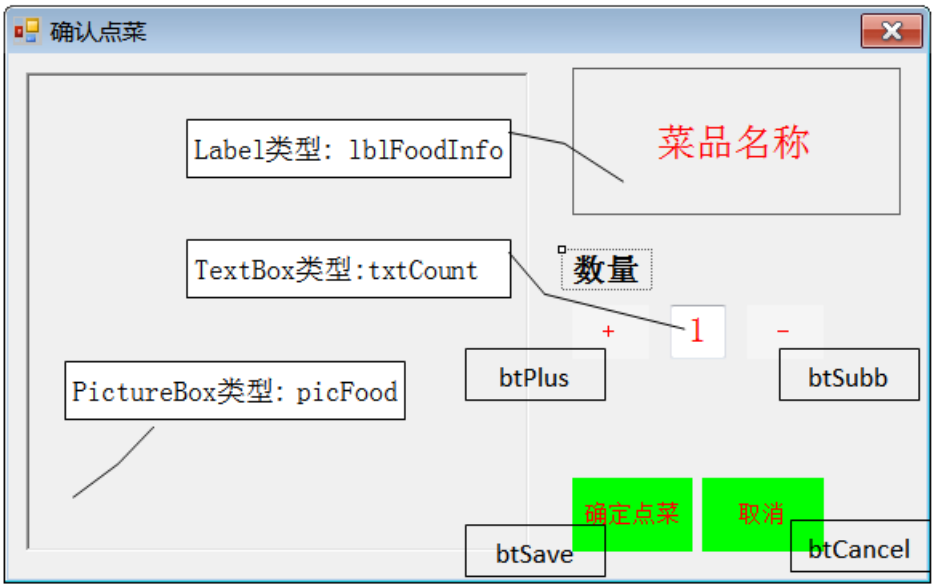


图 7.7　FrmAddFood 窗体界面布局

程序代码

```
//省略自动生成的命名空间
namespace Client
{
```

```
public partial class FrmAddFood : Form
{
    public FrmAddFood()    {    InitializeComponent();    }//构造方法
    //定义公共字段，保存从主窗体传递过来的值
    public Image Img;//菜品图片
    public string FoodName_Price; //菜品名称和价格
    public int Count = 0;//默认0为未点菜

    private void FrmAddFood_Load(object sender, EventArgs e)
    {
        BackColor = Color.Black; //设置窗体背景色
        //显示主窗体传递过来的菜品信息：菜品图片、菜品名称和价格
        picFood.Image = Img; //显示菜品图片
        lblFoodInfo.Text = FoodName_Price;//显示菜品名称和价格
    }

    //“确定点菜”按钮的事件处理过程
    private void btSave_Click(object sender, EventArgs e)
    {
        Count =int.Parse ( txtCount.Text);//取得菜品数量
        DialogResult = DialogResult.OK;// //设置窗体返回值，自动关闭窗体
    }

    //“取消”按钮的事件处理过程
    private void btCancel_Click(object sender, EventArgs e)
    {
        DialogResult = DialogResult.Cancel;//设置窗体返回值，自动关闭窗体
    }

    //增加菜品数量
    private void btPlus_Click(object sender, EventArgs e)
    {
        int count = int.Parse(txtCount.Text);
        count++;//菜品数量加1
        txtCount.Text = count.ToString();
    }

    //减少菜品数量
    private void btSubb_Click(object sender, EventArgs e)
    {
        int count = int.Parse(txtCount.Text);
        count--;//菜品数量减1
        //限制数量只能大于0
        if (count > 0) txtCount.Text = count.ToString();
```

```
        }
    }//窗体类结束
}命名空间结束
```

代码说明

如果一个窗体由其他窗体以模式窗体的方式打开，那么在设置这个窗体的 DialogResult 属性后，该窗体将自动关闭，其他窗体可以通过该窗体的 DialogResult 属性值来判断用户执行的操作。由于其余代码比较简单，请读者参照注释自行分析。

7.6.2.3 Program.cs 启动入口文件

Program.cs 文件是项目启动的入口文件。在客户端主窗体显示前，需要从 TableInfo.txt 文本文件中获取客户端对应的餐台编号信息，餐台编号是顾客结账的依据，该文件位于客户端当前启动位置。把餐台编号信息保存到文件的目的是无须每次启动客户端时都要重复执行设置餐台编号的操作，而是通过判断该文件是否存在来决定。如果该文件不存在，说明客户端是首次启动，未设置当前客户端对应的餐台编号，那么需要自动打开设置窗体，让系统管理员设置好并在保存后显示主窗体；如果该文件已存在，说明该餐台编号已经设置，那么可以直接显示主窗体。

程序代码

```
//省略自动生成的代码
using System.IO;
namespace Client
{
    static class Program
    {
        //定义保存餐台信息(编号和名称、状态)的公共静态变量
        public static string TableInfo = "";

        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            TableInfo = GetTableInfo();//1.从文件读取餐台信息

            //如果当前不存在餐台信息文件，打开设置窗体FrmSetter
            if (TableInfo == "")
            {
                FrmSetter frm = new FrmSetter();//2.创建FrmSetter对象
                DialogResult dr = frm.ShowDialog();
                if (dr != DialogResult.OK)//3.如果用户取消设置
```

```
            {
                Application.Exit();//退出应用程序
                return;
            }
        }

        //打开主窗体。如果能执行到这里，可以确定TableInfo变量已包含餐台信息
        Application.Run(new FrmMain());//4.打开客户端主窗体
    }//Main()方法结束

    //5.自定义方法，判断信息文件是否存在
    static string GetTableInfo()
    {    //信息文件的格式：餐台ID|餐台名称|状态
        if (!File.Exists(Application.StartupPath + "\\TableInfo.txt")) return ""; //6
        return    File.ReadAllText(Application.StartupPath + "\\TableInfo.txt");//7
    }
}//类结束
}//命名空间结束
```

代码说明

(1) 注释 1：调用自定义方法 GetTableInfo，根据当前应用程序所在位置的餐台信息文件是否存在来决定是否需要打开设置窗体。如果返回值是空字符串，说明首次使用本系统，需要先打开设置窗体；否则直接打开主窗体。

(2) 注释 2、注释 3：打开设置窗体。如果窗体返回值不是 DialogResult.OK，代表用户取消设置，或者用户无权限操作，那么关闭应用程序。

(3) 注释4：Application.Run(new FrmMain())语句表示以模式对话框的方式打开主窗体（模式对话框的概念在本书第4章已介绍），该语句等同于下面语句的组合：

```
FrmMain frm = new FrmMain();
frm.ShowDialog();
```

(4) 注释 5：自定义方法 GetTableInfo，实现判断当前应用程序启动位置是否包含餐台信息文件，如果存在则读取并返回文件的内容，否则返回空字符串。Application.StartupPath 属性代表当前应用程序的启动位置，File 的静态方法 ReadAllText 用于实现一次性读取整个文本文件的内容。

7.6.2.4 登录与设置窗体

设计登录与设置窗体 FrmSetter 的原因是根据实际出发，因为前台系统不能由顾客关闭，而必须由拥有权限的管理员来操作；另外设置餐台信息也需要通过检测用户权限来操作，因此把这两部分功能整合在一个窗体，方便使用。此外，设置餐台信息应该在系统首次使用时而不是每次启动时都要设置，为此，在当前应用程序启动位置建立一个保存餐台信息的文本文件，如果不存在该文件，那么打开本窗体进行设置，并保存设置结果；如果文件存在，直接打开主窗体。

该窗体是在主窗体单击“设置”按钮时和客户端首次启动运行时使用。

界面布局

为 Client 项目添加窗体，保存为 FrmSetter。为窗体添加 2 个文本框，分别用于输入用户名和密码；添加 3 个操作按钮；添加 1 个组合框，用于显示当前餐台编号的占用情况,，界面布局和控件名称、控件文本如图 7.8 所示。

图 7.8　FrmSetter 设置窗体界面布局

程序代码

```
//省略自动生成的代码
using System.Data.SqlClient;
namespace Client
{
    public partial class FrmSetter : Form
    {
        public FrmSetter()  {  InitializeComponent();  }//构造方法
        //1.Load事件处理过程
        private void FrmSetter_Load(object sender, EventArgs e)
        {
            AddCombo();//读取餐台信息，填充到组合框
        }

        //2.自定义方法，从数据库中取得的餐台信息并按一定格式填充到组合框
        void AddCombo()
        {
            cmbTableInfo.Items.Clear();
            string sql = "Select * From TableInfo Order by TableStatus";
            SqlDataReader dr = DBHelper.GetDataReader(sql);
            if (dr != null)
            {
                while (dr.Read())
                {
                    cmbTableInfo.Items.Add(dr["TableID"] + "|" + dr["TableName"] +
                      "|" + dr["TableStatus"]);
                }
                dr.Close();//关闭
```

```
        }

        //如果数据库没有数据，提示用户并中断操作
        if (cmbTableInfo.Items.Count == 0)
        {
            MessageBox.Show("请先在数据库录入餐台信息数据!", "提示");
            DialogResult = DialogResult.Cancel;
            return;
        }
        //如果有数据，默认选择第1项
        cmbTableInfo.SelectedIndex = 0;
}

//3.自定义方法，验证权限
bool CheckUser(string User, string Pwd)
{
        string sql = "Select count(*) From Users Where UserName='";
        sql += User + "' and Pwd='" + Pwd + "'";
        object obj = DBHelper.GetOne(sql);//无法连接数据库将返回null
        int count = obj == null ? 0 : (int)obj;
        return count > 0 ? true : false;
}

//4.正确设置后，保存餐台信息到文件，下次启动直接将进入主窗体
private void btSet_Click(object sender, EventArgs e)
{
        //4.1 帐号正确才可以设置
        if (!CheckUser(txtUser.Text, txtPwd.Text))
        {
            MessageBox.Show("帐号错误!", "提示");
            return;
        }
        //4.2 取得餐台信息
        string TableInfo = cmbTableInfo.Text;
        if (TableInfo == "")
        {
            MessageBox.Show("请选择餐台信息!", "提示");
            return;
        }
        //4.3 判断餐台编号是否被其他客户端占用
        if (TableInfo.IndexOf("已设置") >= 0)
        {
            MessageBox.Show("此餐台编号已被占用!", "提示");
            return;
```

```
            }
            //4.4 保存到程序启动位置，以便下次启动自动读取餐台信息
            System.IO.File.WriteAllText(Application.StartupPath + "\\TableInfo.txt", TableInfo);
            Program.TableInfo = TableInfo;

            //4.5 更新到数据库
            string sql = "Update TableInfo Set TableStatus='已设置' Where TableID=" +
            TableInfo.Split('|')[0];
            DBHelper.ExecSql(sql);

            //4.6 设置窗体返回值，自动关闭窗体
            DialogResult = DialogResult.OK;
        }

        //5.取消操作
        private void btCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
        }

        //6.“退出系统”按钮的事件处理过程
        private void btClose_Click(object sender, EventArgs e)
        {
            if (!CheckUser(txtUser.Text, txtPwd.Text))
            {
                MessageBox.Show("帐号错误!", "提示");
                return;
            }
            //6.1 帐号正确，直接退出应用程序并关闭所有打开的窗体
            Application.Exit();
        }
    }//类结束
}//命名空间结束
```

代码说明

(1) 注释 2：自定义方法 AddCombo，实现从数据库读取餐台信息并填充到组合框。餐台信息的数据格式自定义为“餐台 ID|餐台名称|餐台状态”，这里的餐台状态是指该餐台编号是否已给其他客户端占用，取值为未设置或已设置。

(2) 注释 3：自定义方法 CheckUser，根据文本框输入的用户名和密码，在数据库的 Users 表中判断是否存在该用户。这是验证帐号正确性的典型方式，即按条件判断记录数，如果记录数(即行数)大于 0 则存在记录，说明帐号正确，方法返回 true，否则错误，返回 false。由于 DBHelper 类的静态方法 GetOne 在设计时返回 object 类型，并在无法连接数据库时返回 null，因此这里需要判断后转换类型。

(3) 注释 4：注释 4.1 根据输入的用户名和密码判断帐号是否正确；注释 4.2 中，由于

组合框的样式设置为下拉列表框(只读)，只能通过 Text 属性判断选择的内容；注释 4.3 根据选择的内容判断是否存在代表已被占用的子串(存在，返回>=0 的值，否则返回-1)；注释 4.4 将餐台信息保存到文本文件中，并设置 Program.TableInfo 静态变量的值，该值在主窗体使用；注释 4.5 用于更新数据库，避免其他餐台的客户端使用；注释 4.6 设置窗体的返回值为 DialogResult.OK，并自动关闭窗体，该值在这里代表操作成功。如果在 Main 方法中打开本窗体并获得 DialogResult.OK 值，表示可以启动主窗体。

(4) 注释 5：关闭窗体，并让窗体返回 DialogResult.Cancel。由于 FrmSetter 窗体在 Program 中的 Main 方法和在主窗体中单击“设置”按钮都会使用到，该值在 Main 方法代表退出整个应用程序，在主窗体中，将返回到主窗体界面。

(5) 注释6：单击窗体“退出系统”按钮时，需要权限判断，该操作只在主窗体中准备退出系统时会用到。Application.Exit()与Close()方法的区别在于，前者关闭当前所有已打开的窗体，退出系统，后者仅关闭本窗体。

7.6.3 服务端设计

7.6.3.1 主窗体 FrmMain 设计

服务端的主窗体 FrmMain 是一个 MDI 窗体。本章之前并未专门介绍 MDI 窗体的相关内容，由于 MDI 窗体使用比较简单，这里作个补充介绍。

正如容器控件可以把其他控件作为子控件添加到其内部一样，一个窗体也可以作为其他窗体的容器，让其他窗体打开时，显示在其内部。这样的窗体在 C#中称为 MDI(Multiple Document Interface 多文档接口)窗体，主要目的是为了便于集中管理多个窗体。在 MDI 窗体内部打开多个窗体后，可以同时在多个窗体之间进行切换操作，也可设置窗体的排列方式。

要使一个窗体成为MDI窗体，可以通过设置其IsMdiContainer属性为true(可以在程序运行阶段通过代码动态设置，也可以在设计阶段在属性窗体中设置)。要使其他窗体打开时显示在MDI窗体内部，必须在运行时通过代码动态设置其MdiParent属性为已存在的MDI窗体对象，并通过Show方法显示为非模式窗体(不能是ShowDialog方法，因为该方法无法同时操作多个窗体)。

如果我们在一个项目在创建三个窗体，分别命名为 MDIForm、Form1 和 Form2。设置 MDIForm 的 MdiParent 属性为 true，使其成为 MDI 窗体，并在 MDIForm 打开时，在其内部显示 Form1 和 Form2。下面的代码实现让 Form1 和 Form2 成为 MDIForm 的子窗体并以平铺方式显示，运行效果如图 7.9 所示。

```
private void MDI_Load(object sender, EventArgs e)
{
    this.IsMdiContainer = true;//设置为MDI窗体，也可以在属性窗体中设置
    Form1 frm1 = new Form1();
    Form2 frm2 = new Form2();
    frm1.MdiParent = this;// 动态属性。设置其容器为MDI窗体对象
    frm2.MdiParent = this;//
    frm1.Show();//显示窗体
    frm2.Show();
    this.LayoutMdi(MdiLayout.TileHorizontal);//设置多窗体排列方式
```

}

其中，MdiParent 属性只能通过代码设置，代表该窗体显示在哪个 MDI 窗体内部。MDI 窗体的 LayoutMdi 方法代表当前 MDI 窗体内部多个窗体的排列方式。参数为 MdiLayout 枚举类型。MdiLayout 中的各个枚举值的含义如下：

(1) Cascade：所有 MDI 子窗口均层叠在 MDI 父窗体的工作区内。

(2) TileHorizontal：所有 MDI 子窗口均水平平铺在 MDI 父窗体的工作区内。

(3) TileVertical：所有 MDI 子窗口均垂直平铺在 MDI 父窗体的工作区内。

(4) ArrangeIcons：所有 MDI 子图标均排列在 MDI 父窗体的工作区内。

另外，还可以通过MDI窗体的MdiChildren集合属性，取得MDI窗体当前打开的所有子窗体对象。

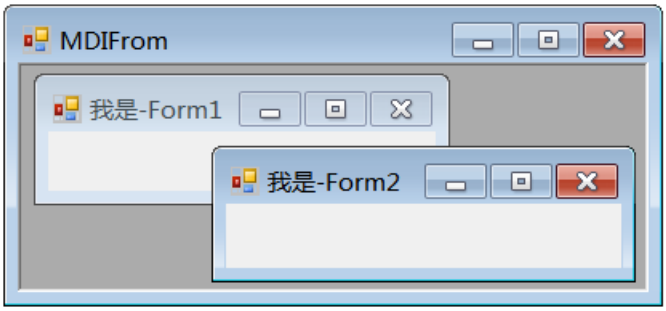


图 7.9 MDI　窗体运行效果

Server项目的主窗体FrmMain是MDI窗体，在项目中主要提供对其他窗体的操作接口，如菜单、工具栏等，具体功能则在各自窗体中独立实现。在主窗体FrmMain中，只有两个常用的窗体作为其子窗体，一个是维护菜品数据的窗体FrmFood，另外一个是用于结账操作的窗体FrmBill，其余窗体为非子窗体并以模式窗体方式显示。

以下是服务端主窗体 FrmMain 的设计与实现。

界面布局

在 Server 项目中添加 1 个新窗体，保存为 FrmMain。为 FrmMain 窗体添加菜单、工具栏和状态栏，实现的界面布局如图 7.10 所示，窗体的各个菜单项参见图 7.11。

图 7.10　主窗体界面布局

由于主窗体仅作为其他窗体的操作接口，在下面给出的代码中，仅包含主要窗体的操作代码，包括“基础资料”菜单中的菜品编辑、以及“操作”菜单中的结账、打印小票和营业报表功能实现；其余模式窗体的打开以及“窗口”菜单中实现窗体排列等功能都比较简单，

不再列出这些代码。

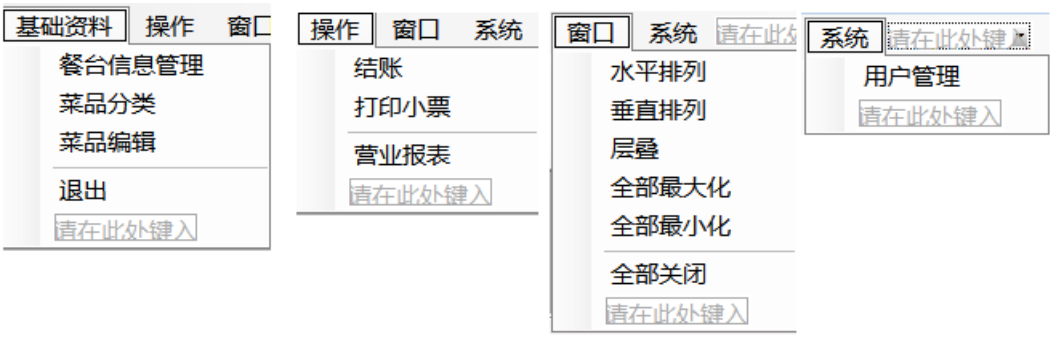


图 7.11　菜单列表

在图 7.11 中，主要菜单项和工具栏项按钮对应的文本和名称如下：

(1)　基础资料菜单

餐台信息管理：Menu_TableInfo

菜品分类：Menu_FoodStyle

菜品编辑：Menu_FoodItem

(2)　操作菜单

结账：Menu_Bill

打印小票：Menu_PrintBill

营业报表：Menu_PrintSales

(3)　工具栏按钮

结账单：Tool_BtBill

代码片段-FrmMain.cs

```
FrmBill bill = null;//定义窗体类字段
//1.自定义方法 根据窗体名称查找子窗体
bool FindForm(string formName)
{
    bool IsExit = false;
    foreach (Form f in MdiChildren)
    {
        if (f.Name == formName)
        {
            IsExit = true;
            break;
        }
    }
    return IsExit;
}
//2.自定义方法：显示子窗体
void ShowForm(string frmName, Form frm)
{
```

```
        frm.Name = frmName;
        frm.WindowState = FormWindowState.Maximized;
        frm.MdiParent = this;
        frm.Show();//显示非模式窗体
}
```
//3.Load事件处理过程，设置为MDI窗体并显示结账单窗体
```
private void FrmMain_Load(object sender, EventArgs e)
{
        this.IsMdiContainer = true;//设置为MDI窗体
        //让主窗体显示时，默认显示结账窗体
        bill = new FrmBill(0);
        ShowForm("Bill1", bill);//首先打开结账窗体
        bill.SetTabPage(0);//指定显示当前选项卡为结账
}
```
//4.工具栏"结账单"按钮Tool_BtBill事件处理过程
```
private void Tool_BtBill_Click(object sender, EventArgs e)
{
        Menu_Bill.PerformClick();//执行与菜单一致操作
}
```
//5."结账"菜单项Menu_Bill事件处理过程
```
private void Menu_Bill_Click(object sender, EventArgs e)
{
        if (!FindForm("Bill1"))
        {
                bill = new FrmBill();
                ShowForm("Bill1", bill);//打开结账窗体
        }
        bill.SetTabPage(0);//指定显示当前选项卡为结账
}
```
//6.”打印小票”菜单项Menu_PrintBill事件处理过程
```
private void Menu_PrintBill_Click(object sender, EventArgs e)
{
        if (!FindForm("Bill1"))
        {
                bill = new FrmBill();//结账窗体
                ShowForm("Bill1", bill);
        }
        bill.SetTabPage(1);//指定窗体显示当前选项卡为打印小票
}
```
//7.”营业报表”菜单项Menu_PrintSales事件处理过程
```
private void Menu_PrintSales_Click(object sender, EventArgs e)
{
        //选择时间范围的窗体FrmPrintTime
        FrmPrintTime frmTime = new FrmPrintTime();//打印窗体
```

```
    if (frmTime.ShowDialog() == DialogResult.OK)
    {
        FrmPrint frm = new FrmPrint();//共有的打印窗体
        frm.SelectIndex = 1;//报表选择1---营业额报表   2---打印小票
        frm.StartTime = frmTime.StartTime;
        frm.EndTime = frmTime.EndTime;
        frm.Show();
    }
}
//8.”菜品编辑”菜单项Menu_FoodItem事件处理过程
private void Menu_FoodItem_Click(object sender, EventArgs e)
{
    if (!FindForm("FrmFood1"))
    ShowForm("FrmFood1", new FrmFood());//显示菜品编辑窗体
}
//9. “菜品分类”菜单项Menu_FoodStyle事件处理过程
private void Menu_FoodStyle_Click(object sender, EventArgs e)
{
    FrmFoodStyle frm = new FrmFoodStyle();菜品分类窗体
    frm.ShowDialog();//显示为模式窗体
}
//省略其他菜单项事件处理过程
```

代码说明

(1) 主窗体的功能是打开其他窗体，具体功能在各自窗体中实现。在这里，将操作数据比较多的两个窗体“菜品编辑”和“结账”窗体作为主窗体的子窗体来实现，其他窗体以模式对话框方式显示。

(2) 注释1：自定义方法FindForm，实现根据给定窗体的名称，查找子窗体。主要用于判断当前操作的窗体是否已经打开，如果打开，不再重复打开。MdiChildren集合属性包含已经在MDI窗体中打开的所有子窗体对象。

(3) 注释 2：自定义方法 ShowForm，实现根据已创建好的窗体对象，通过设置其 MdiParent 属性使其作为主窗体的子窗体。注意这里设置窗体 Name 属性的目的是便于在主窗体中根据窗体名来查找子窗体，设置 WindowState 属性是让窗体在打开时最大化。

(4) 注释 3：实现主窗体显示时，将结账窗体显示为子窗体。由于结账功能和打印功能分属“结账”窗体中的两个不同选项卡，这里调用选项卡对象的 SetTabPage 方法来指定当前选项卡。

(5) 注释 4：工具栏中“结账单”按钮的事件处理过程，这里仅包含调用菜单项 Menu_Bill 的 PerformClick 方法的语句，表示执行菜单项 Menu_Bill 的 Click 事件处理过程。菜单项和工具栏按钮都可以当一般按钮使用，都具有 PerformClick 方法。

(6) 注释 5、注释 6：根据窗体名称查找子窗体 FrmBill，如果没有找到，则新建窗体并打开，同时调用 FrmBill 中自定义方法 SetTabPage，显示指定的选项卡。

(7) 注释 7：显示项目共用的报表打印窗体 FrmPrint。FrmPrintTime 是选择时间范围的窗体。

(8) 注释 8：显示菜品编辑 FoodItem 子窗体。

(9) 注释 9：以模式窗体方式打开 FrmFoodStyle 窗体。

7.6.3.2 结账窗体 FrmBill 设计

结账窗体 FrmBill 是项目主要窗体，也是 MDI 主窗体的子窗体之一，实现从数据库的餐台信息表 TableInfo 和订单表 Order 中读取数据，并将数据显示在数据表格中，这些数据代表不同餐台的顾客在不同时间范围的点餐信息。在该窗体实现了结账功能，以及对当前账单和历史账单的查询和打印功能。

结账窗体使用了位于工具箱“容器”组中的TabControl选项卡容器控件，该控件的选项卡用作数据表格的容器，在不同选项卡间切换时显示不同类型的账单信息。TabControl的集合属性TabPages包含一组已存在的选项卡TabPage对象，可以使用TabPages动态添加选项卡，也可以在TabControl的属性窗体的通过向导添加选项卡。使用TabPages[下标]或TabPages[选项卡名称]方式来引用具体的选项卡对象，如：

```
tabControl1.TabPages[0];//使用下标引用
tabControl1.TabPages["tabPage1"];//使用名称引用
```

每一个选项卡都相当于带标题栏的面板控件，与其他容器控件使用并无多大区别。可以添加或移除其他任意子控件。如果要设置一个选项卡为当前选项卡，可以通过TabControl对象的SelectIndex属性或SelectedTab属性来指定，前者使用下标赋值，后者使用TabPage对象赋值，如：

```
tabControl1.SelectedIndex = 0;
tabControl1.SelectedTab = tabPage1;
```

界面布局

为 Server 项目添加窗体，保存为 FrmBill。在窗体中添加 1 个 TabControl 控件，保持默认的两个选项卡。在选项卡 1 中，添加 1 个“结账”按钮，停靠在选项卡顶部，添加 1 个表格控件，填充剩余空间；在属性窗体中，将表格控件初始化为 4 列，各个控件的名称和文本参照图 7.12（a）中的界面布局。在选项卡 2 中，添加 1 个 Panel 控件，停靠在选项卡顶部，并为其添加 2 个 DateTimePicker 控件，1 个文本框和 2 个按钮；将最后添加的表格控件填充剩余的 Panel 空间，各个控件的名称和文本参照图 7.12（b）中的界面布局。

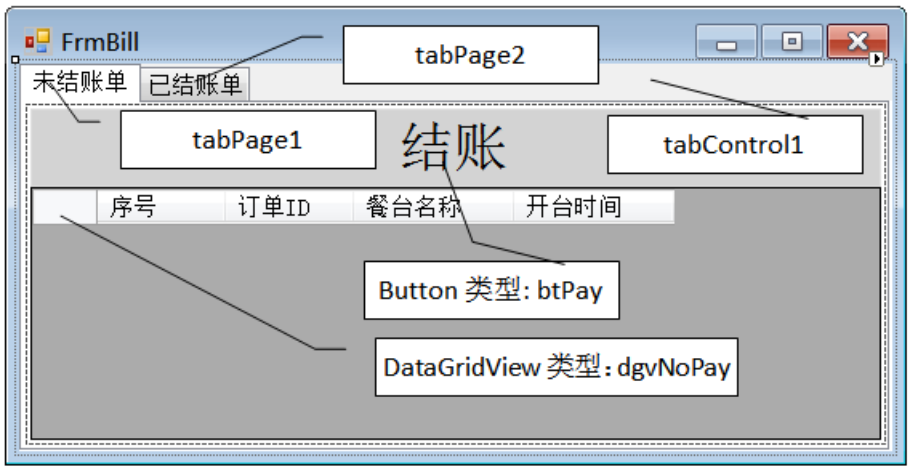


图 7.12 (a) 结账窗体-未结账单

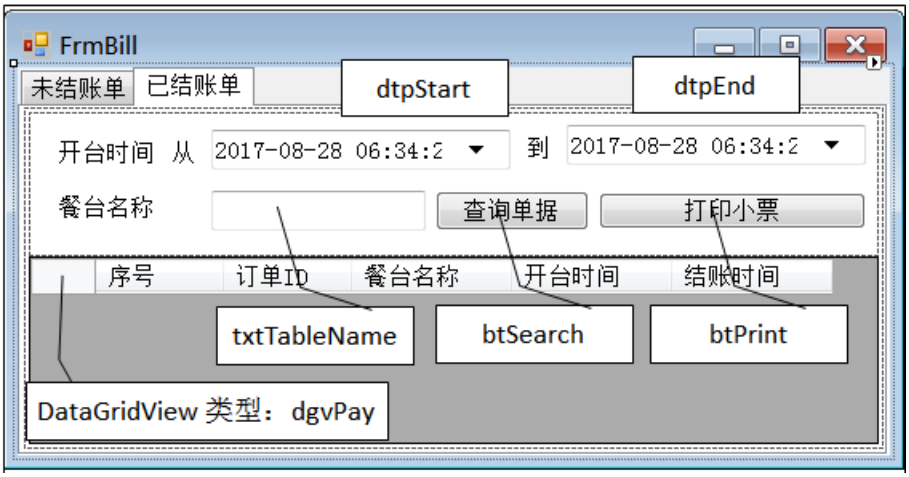


图 7.12 (b)　结账窗体-已结账单

程序代码

```
//省略自动添加的using部分
using System.Data.SqlClient;
namespace Server
{
    public partial class FrmBill : Form
    {
        public FrmBill()  {    InitializeComponent();  }//构造方法

        //1.自定义方法，使得外部可以在打开窗体时设置当前选项卡
        public void SetTabPage(int index = 0)
        {
            tabControl1.SelectedIndex = index;
        }

        //2.初始化表格
        void InitGridView(DataGridView dgv)
        {
            dgv.Rows.Clear();//清空行
            dgv.AllowUserToAddRows = false;//不允许直接添加行
            dgv.AllowUserToDeleteRows = false;//不允许直接删除
            dgv.AllowUserToResizeRows = false;//行高度不可调整
            dgv.MultiSelect = false;//单行选中
            dgv.SelectionMode = DataGridViewSelectionMode.FullRowSelect;//选中一行
            dgv.ReadOnly = true;//只读，不能直接修改
            dgv.RowHeadersVisible = false;//隐藏行头
        }
        //3.根据参数IsEnd，在表格中显示已结账单或未结账单
        void FillGridView(DataGridView dgv, bool IsEnd = false)
        {
```

```
        dgv.Rows.Clear();
        int rowIndex = 0;

        //先默认为未结账账单
        string WhereStr = " Where Status='开台'";//默认
        string sql = "Select [Order].OrderID,[TableInfo].TableName,[Order].StartTime ";
        sql += " From [TableInfo] inner join [Order] ON
        [TableInfo].TableID=[Order].TableID    " + WhereStr;

        //假如查询已结账账单
        if (IsEnd) //如果已结账，修改条件
        {
            //按条件查询已结账信息
            WhereStr = "    Where Status='已结账' and ([Order].StartTime>='" +
            dtpStart.Value + "' and [Order].StartTime<='" + dtpEnd.Value + "')";
            if (txtTableName.Text != "")//按餐台名称，模糊匹配
            {
                WhereStr += " and TableName Like %'" + txtTableName.Text + "'%";
            }
            sql = "Select [Order].OrderID, [TableInfo].TableName, [Order].StartTime,
              [Bill].EndTime " ;
            sql += " From [TableInfo] inner join [Order] ON [TableInfo].TableID=
            [Order].TableID INNER JOIN [Bill] ON [Order].OrderID=[Bill].OrderID" +
            WhereStr;
        }
        SqlDataReader dr = DBHelper.GetDataReader(sql);

        while (dr.Read())
        {
            dgv.Rows.Add();
            dgv.Rows[rowIndex].Cells[0].Value = rowIndex + 1;
            dgv.Rows[rowIndex].Cells[1].Value = dr["OrderID"];
            dgv.Rows[rowIndex].Cells[2].Value = dr["TableName"];
            dgv.Rows[rowIndex].Cells[3].Value = dr["StartTime"];
            //只在已结账账单显示结账时间
            if (IsEnd) dgv.Rows[rowIndex].Cells[4].Value = dr["EndTime"];
            rowIndex++;
        }
        dr.Close();//同时将关闭连接
    }

    //4.Load事件处理过程，初始化界面控件
    private void FrmBill_Load(object sender, EventArgs e)
    {
```

```
    //初始化日期时间控件，起始日期为昨天
    dtpStart.Value = DateTime.Now.AddDays(-1);
    InitGridView(dgvPay);//初始化未结账表格
    InitGridView(dgvNoPay);//初始化已结账表格
    FillGridView(dgvNoPay);//首先显示当前未结账单数据
}

//5."结账"按钮btPay的事件处理过程
private void btPay_Click(object sender, EventArgs e)
{
    Pay();//对选择的行进行结账操作
}

//6.自定义方法，实现结账功能
public void Pay()
{
    if (dgvNoPay.CurrentRow == null)
    {
        MessageBox.Show("没有选中要结账的单据！", "提示");
        return;//没有选择
    }

    int OrderID = int.Parse(dgvNoPay.CurrentRow.Cells["OrderID"].Value.ToString());
    //显示顾客点餐的详细信息：菜品名称、价格、数量，
    //该菜品项的消费金额小计(价格*数量)
    string sql = "Select [FoodItem].FoodName, [FoodItem].price,[OrderDetail].Count,
    [FoodItem].price*[OrderDetail].Count as  小计";
    sql += " From [FoodItem],[OrderDetail] Where ";
    sql += " [FoodItem].FoodID=[OrderDetail].FoodID and [OrderDetail].OrderID=" +
    OrderID;
    SqlDataReader dr = DBHelper.GetDataReader(sql);
    string s = "菜品名称\t单价\t数量\t小计\r\n";//显示格式
    float Pay = 0.0f;//累计消费金额
    while (dr.Read())
    {
        s += dr["FoodName"] + "," + dr["Price"] + "," + dr["Count"] + ",小计：" +
          dr["小计"] + "\r\n";
        Pay += float.Parse(dr["小计"].ToString());//每份菜品项消费金额的累计
    }
    dr.Close();

    //确认结账
    if (MessageBox.Show(s + "\r\n消费金额：" + Pay, "消费明细",
    MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
```

```
        {
            //订单表中，更改为"已结账状态"
            sql = "Update [Order] set    Status='已结账' where orderID=" + OrderID;
            DBHelper.ExecSql(sql);
            //插入账单表：订单ID、金额
            sql = "Insert Into Bill(OrderID,Money,UserID) Values(" + OrderID + "," + Pay
            + ",1)";
            DBHelper.ExecSql(sql);
            //结账成功后，从表格中移除该行信息
            dgvNoPay.Rows.Remove(dgvNoPay.CurrentRow);
        }
    }

    //7."查询"按钮事件处理过程，按条件查询已结账单，并显示到表格中
    private void btSearch_Click(object sender, EventArgs e)
    {
        FillGridView(dgvPay, true);
    }

    //8.打开打印窗体，并是打印窗体打印小票报表
    private void btPrint_Click(object sender, EventArgs e)
    {
        PrintBill();
    }

    //9.自定义方法，打开打印窗体
    public void PrintBill()
    {
        if (dgvPay.CurrentRow == null)
        {
            MessageBox.Show("请选择需要打印的点菜单！", "提示");
            return;
        }
        int OrderID = int.Parse(dgvPay.CurrentRow.Cells[1].Value.ToString());

        FrmPrint frm = new FrmPrint();
        frm.OrderID = OrderID;//打印窗体需要的查询用的条件
        frm.SelectIndex = 2;//窗体打印报表标记，1-打印营业报表，2-打印小票报表
        frm.Show();
    }
}//类结束
}//命名空间结束
```

代码说明

(1) 本窗体代码实现的操作逻辑为：① 顾客在客户端Client项目的主窗体中，单击“开台”按钮时，数据将保存到Order表中，当单击菜品项确认点菜时，菜品具体信息将保存到明细表OrderDetail中；② 在本窗体的结账界面中，先从订单表Order表中取得处于“开台”状态的OrderID、TableID和开台时间StartTime，并根据TableID，从TableInfo中取得餐台名称TableName，组合显示到表格dgvNoPay中，在选择行后，可以执行“结账”操作；③ 顾客结账时，需要显示消费清单和消费金额。为显示消费清单，首先根据选中行的订单表Order中的关键字段OrderID，从订单细节表OrderDetail中找到菜品编号FoodID，再根据FoodId从菜品表FoodItem查询菜品名称FoodName和价格Price，将顾客点餐的具体内容显示在对话框中，最后确认或取消结账。结账操作实现将订单表的状态设置为“已结账”状态，并插入相关数据到账单表Bill，作为历史数据；④ 在已结账界面中，根据查询条件，从订单表Order、餐台信息表TableInfo和账单表Bill中，取得订单表中的OrderID、开台时间StartTime、餐台信息表中的餐台名称TableName和账单表中的结账时间EndTime，显示到表格dgvPay中，以便打印包含消费项目的小票。

(2) 注释 1：自定义方法 SetTabPage，实现根据选项卡的下标，设置 TabControl 控件的当前选项卡。该方法是在主窗体打开 FrmBill 子窗体后，根据需要显示的状态是“未结账单”还是“已结账单”在外部调用的，因此方法使用 public 修饰符。

(3) 注释 2：自定义方法 InitGridView，实现根据不同的参数，初始化不同的表格。主要为了避免重复编写代码。这些属性也可以在属性窗口中预先设置。

(4) 注释 3：自定义方法 FillGridView，实现根据参数，将查询结果显示在不同表格中。当参数 IsEnd 为 false 时，读取“未结账单”的数据并显示在 dgvNoPay 表格中，否则，读取“已结账单”的数据显示在 dgvPay 表格中。由于窗体有两个表格，列名不能相同，因此这里采用下标方式来访问表格的单元格，注意表格列下标必须对应 dr 对象中的列名。方法中的 SQL 语句可能是难点，组合 SQL 语句的基本思路是，需要哪些字段信息(Select [表 1].字段名，[表 2].字段名…)，这些字段信息在哪个表（From 表 1,表 2…），这些表之间根据哪个字段来关联（WHERE [表 1].关联字段=[表 2].关联字段…）。

(5) 注释 6：自定义方法 Pay，实现结账功能。首先根据要显示的菜品名称、价格和数量和消费金额，通过 SQL 连接语句，实现多表查询，在信息对话框中显示消费清单和消费金额，然后修改订单表状态并从表格中移除，最后添加到账单表中。

(6) 注释 9：自定义方法 PrintBill，实现打开项目共用的打印窗体来打印消费清单。打印窗体在本项目中可以打印消费清单，也可以打印营业报表，因此，需要在创建打印窗体对象时，传递一个标识字段 SelectIndex 来决定打印哪个报表。同时，报表数据是根据 OrderID 字段读取的，因此也需要传递过去。下一节内容将介绍打印报表窗体 FrmPrint 的具体实现。

7.6.3.3 报表打印功能实现

在第 6 章“6.3 数据访问控件”节中介绍了如何向项目添加数据源，通过从“数据源”窗体以拖拽的方式将整个数据表拖放到窗体，可以快速建立数据表的编辑界面，而无须编写代码；同时，在介绍“6.4 数据报表”时，报表设计器中“报表数据”窗体中的报表数据源，也来自己添加到项目中的数据源。

在 Server 项目中，由于要用到数据库中所有的表，因此在为项目添加数据源时选择所有的表。添加数据源后，打开项目数据集设计器，可以看到已添加到项目数据集中所有的表及表之间的关系，如图 7.13 所示。

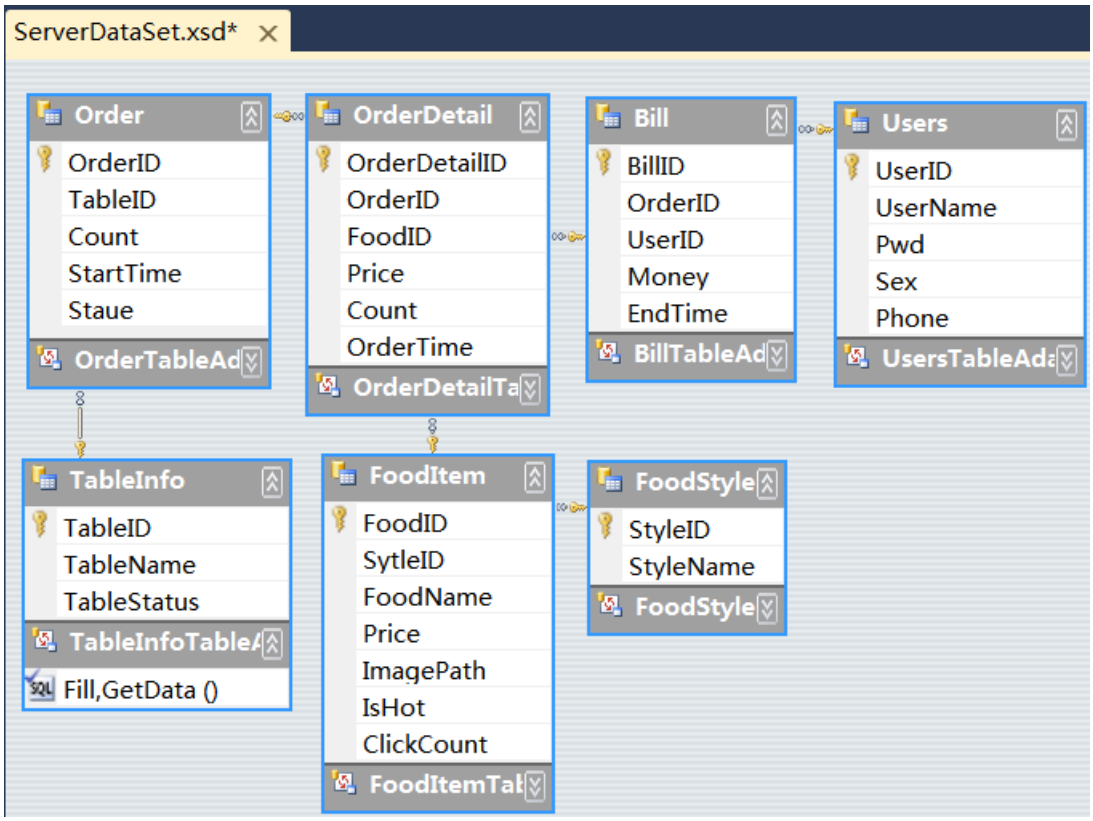


图 7.13　Server 项目数据集设计器

前面章节介绍的数据报表的数据只来自单个表。如果报表数据来自多个表，比如打印消费清单的报表，需要根据订单表 Order 的订单编号 OrderID，从菜品表 FoodItem 查询菜品名称和价格、从订单明细表 OrderDetail 查询菜品数量，进行统计后才能得到顾客的消费金额。在这时情况下，可以使用数据集设计器来协助我们创建自定义数据集，即同时创建数据表以及为数据表填充数据的数据集适配器，这个数据集将作为报表的数据来源。

以实现“打印小票”功能的报表为例，介绍整个报表制作和预览、打印实现过程。

1.　报表制作

(1)　添加报表数据源，制作报表

① 为项目添加数据源，并选择 Restaurant 数据库中所有的表，操作完成后可以在数据集看到如图 7.13 所示的内容。

② 创建自定义报表数据源。在数据集设计器的灰色区域，按右键，选择“添加”→“TableAdapter”，在出现的对话框中，连续按“下一步”，直到出现“TabelAdapter 配置向导”中的“输入 SQL 语句”对话框，单击对话框中的“查询生成器”按钮，选择“Bill”、“FoodItem”和“OrderDetail”三个表，通过“查询生成器”协助我们来完成 SQL 语句中的连接查询。完成的查询语句如下：

```
SELECT Bill.OrderID, FoodItem.FoodName, FoodItem.Price, OrderDetail.Count,  FoodItem.Price*
OrderDetail.Count as  小计,Bill.EndTime
FROM   FoodItem INNER JOIN
OrderDetail ON FoodItem.FoodID = OrderDetail.FoodID INNER JOIN
Bill ON OrderDetail.OrderID = Bill.OrderID AND Bill.OrderID = @OrderID
```

再次打开数据集设计器，将看到多了一个数据集视图，该数据集将作为报表的数据来源。双击其标题栏，修改标题为 Bill_Report，如图 7.14 所示。

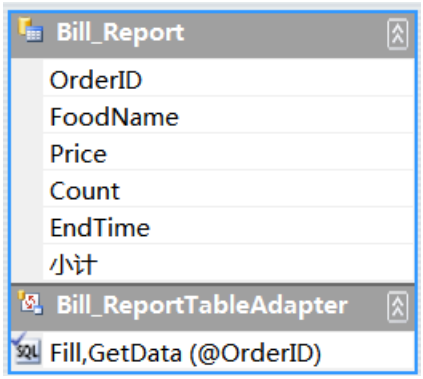


图 7.14　数据表

③ 为项目添加报表文件，保存为 ReportBill.rdlc。在报表设计状态，打开“报表数据”窗体，为报表添加数据源，操作步骤为：在“报表数据”窗体工具栏中单击“新建”→选择“数据集…”，在出现的“数据集属性”对话框中，参照图 7.15，依次输入数据集名称、选择项目数据源和选择“可用数据集”为 Bill_Report。

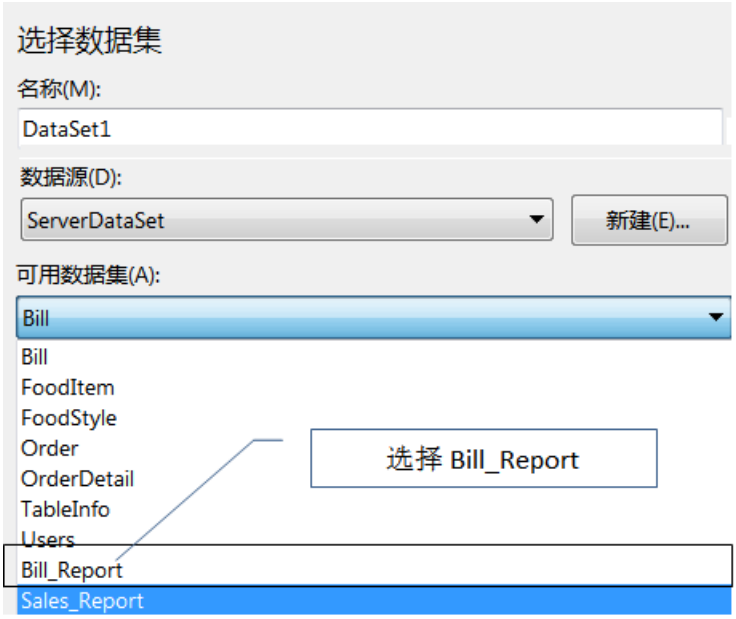


图 7.15　数据集属性对话框

④ 此后，创建报表的过程与第 6 章“应用实例 6.23”中介绍的过程完全一样。最终建立的消费清单报表如图 7.16 所示。

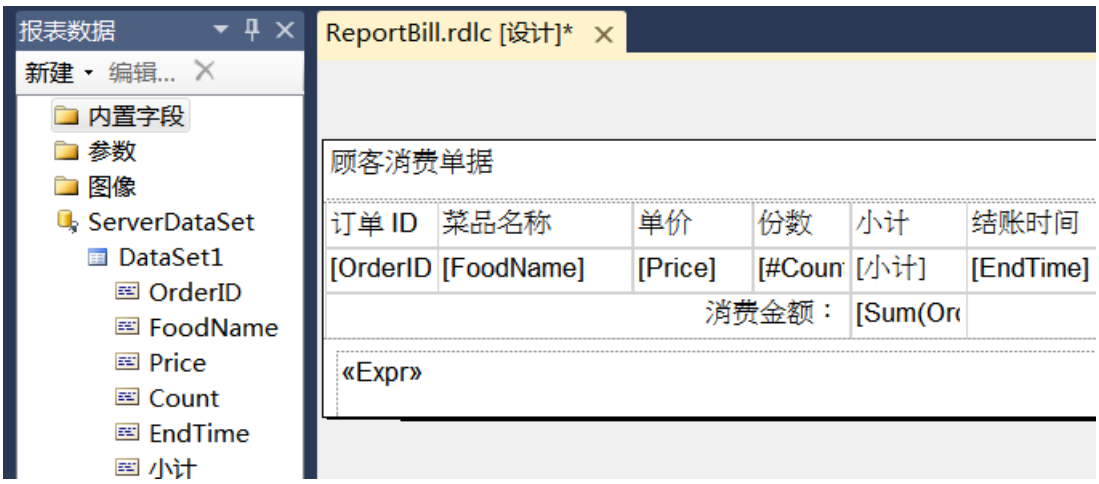


图 7.16　消费清单报表

(2)　以(1)中步骤②~④相同的方式创建营业报表 ReportSales.rdlc。

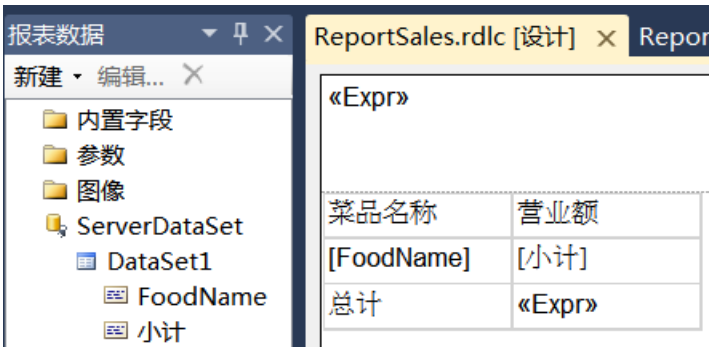


图 7.17　营业报表

其中营业报表的 SQL 查询语句如下：

```
SELECT　FoodItem.FoodName, SUM(OrderDetail.Price * OrderDetail.Count) AS 小计
FROM OrderDetail INNER JOIN FoodItem ON OrderDetail.FoodID = FoodItem.FoodID INNER
JOIN　Bill ON OrderDetail.OrderID = Bill.OrderID WHERE (Bill.EndTime >= @startTime) AND
(Bill.EndTime <= @endTime) GROUP BY FoodItem.FoodName
```

2.　“报表打印”窗体 FrmPrint 的设计

如果一个项目有多个报表文件需要现显示，可以只制作一个打印窗体，通过动态更改报表视图控件 ReportView 的数据源和报表文件来实现。本项目实现多个报表文件共用一个报表窗体 FrmPrint。项目中的报表文件 ReportBill.rdlc 和 ReportSales.rdlc，都通过打开 FrmPrint 窗体来实现预览和打印。下面是具体实现。

以下是报表打印窗体 FrmPrint 具体的设计与实现。

界面布局

为项目添加窗体，保存为 FrmPrint。添加 1 个报表视图控件 ReportView 并填充窗体，不添加其他任何组件。如图 7.18 所示，该窗体将在主窗体中打开。

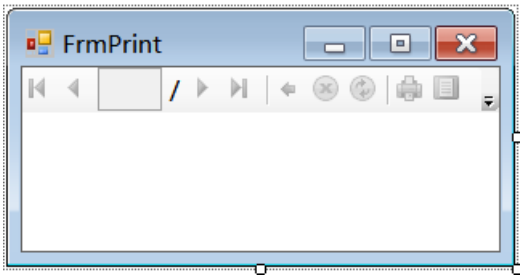


图 7.18　打印窗体界面布局

程序代码

```
//省略自动生成的using部分
using Microsoft.Reporting.WinForms;
using System.Data.SqlClient;
namespace Server
{
    public partial class FrmPrint : Form
    {
        public FrmPrint()   { InitializeComponent(); }//构造方法

        //1.选择报表的标识字段 ： 1---统计营业额   2---打印小票
        public int SelectIndex = 1;//默认

        //2.小票报表读取数据时，SQL语句所需参数
        public int OrderID = 0;//订单ID

        //3.营业报表读取数据时，SQL语句统计所需参数，统计日期
        //日期参数值可由外部传入，这里设置初始值
        public DateTime StartTime = DateTime.Now.AddYears(-10);
        public DateTime EndTime = DateTime.Now;

        //4.Load事件处理过程
        private void FrmPrint_Load(object sender, EventArgs e)
        {
            //创建项目数据集类型和数据源对象
            ServerDataSet ds = new Server.ServerDataSet();//项目数据集对象ds
            if (SelectIndex == 1)//营业报表
            {
                //自定义方法，初始化数据源和指定打印的报表
                SetReportSource("Sales_Report", "Server.ReportSales.rdlc", ds);

                //读取报表所需的数据并填充到数据表
                ServerDataSetTableAdapters.Sales_ReportTableAdapter da =
                new ServerDataSetTableAdapters.Sales_ReportTableAdapter();
                da.Fill(ds.Sales_Report, StartTime, EndTime);
```

```
            //创建报表所需的报表参数
            ReportParameter[] rp = new ReportParameter[2];
            rp[0] = new ReportParameter("StartTime", StartTime.ToString());
            rp[1] = new ReportParameter("EndTime", EndTime.ToString());
            reportViewer1.LocalReport.SetParameters(rp); //传递参数
        }
        else//已结账单的小票报表
        {
            //初始化数据源和指定打印的报表
            SetReportSource("Bill_Report", "Server.ReportBill.rdlc", ds);
            //读取报表所需数据并填充到数据表
            ServerDataSetTableAdapters.Bill_ReportTableAdapter da =
            new ServerDataSetTableAdapters.Bill_ReportTableAdapter();
            da.Fill(ds.Bill_Report, OrderID);
        }
        //刷新报表显示
        reportViewer1.RefreshReport();
    }

    //5.自定义方法：为报表设计器提供报表的数据源
    //目的是添加报表数据源，而报表数据源是ReportDataSource类型
    //Name属性指定报表数据集的名称，Value属性指定绑定数据源对象
    void SetReportSource(string dsTableName, string ReportName, DataSet ds)
    {
        reportViewer1.LocalReport.DataSources.Clear();

        //绑定源对象bs，用作报表数据源对象提供数据来源
        BindingSource bs = new BindingSource();
        bs.DataSource = ds;//绑定源对象的数据源
        bs.DataMember = dsTableName;//绑定数据成员：表名

        ReportDataSource rd = new ReportDataSource();//创建报表数据源对象
        rd.Name = "DataSet1";     //报表的数据集名称
        rd.Value = bs;                //设置数据源为绑定数据源bs
        reportViewer1.LocalReport.DataSources.Add(rd);//本地报表添加数据源

        // 设置报表名称
        reportViewer1.LocalReport.ReportEmbeddedResource = ReportName;
    }
}//类结束
}命名空间结束
```

代码说明

为了理解本段代码，首先要理解报表控件 ReportViewer 如何绑定报表文件和显示报表

数据的过程。为此我们向项目添加一个测试窗体 Test，并将 ReportViewer 控件拖放到窗体。当我们为报表控件选择要显示的报表文件 ReportBill.rdlc 后，我们会发现在窗体下方自动添加了如图 7.19 所示的三个组件。

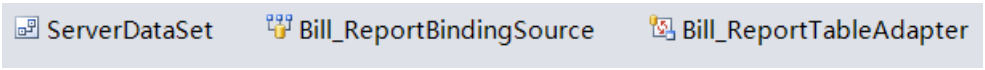


图 7.19　选择报表文件后自动添加的组件

这个操作结果是开发环境 Microsoft Visual Studio 2010 自动帮助我们完成的，运行结果就是在报表控件中能够显示我们在数据集设计器中自定义的数据表 Bill_Report 的数据。

开发环境是怎么帮我们做到的呢？实质上，当我们在报表控件中选择报表文件时，开发环境将分析报表的数据源，并根据报表的数据源在当前窗体创建项目数据集对象，同时根据报表文件选定的数据表创建数据适配器对象，这样我们在图 7.19 就看到了数据集组件 ServerDataSet 和数据适配器组件 Bill_ReportTableAdapter。而 Bill_ReportBindingSource 绑定源组件负责从数据集 ServerDataSet 读取 Bill_Report 数据表的数据，并向其他对象提供数据的来源。报表文件的数据只能由绑定源对象或数据表对象提供，也即绑定源控件在这里是作为报表的数据来源，报表控件 ReportViewer 根据报表文件的设计布局和报表文件的数据源，在运行时装载报表文件并显示。

在窗体的 Load 事件处理过程中我们还可以看到自动添加了如下的代码：

```
private void Test_Load(object sender, EventArgs e)
{
        // TODO: 这行代码将数据加载到表“ServerDataSet.Bill_Report”中。
        //您可以根据需要移动或删除它。
        this.Bill_ReportTableAdapter.Fill(this.ServerDataSet.Bill_Report,OrderID参数);
        this.reportViewer1.RefreshReport();//呈现报表内容
}
```

这段代码实现了数据适配器Bill_ReportTableAdapter根据其保存的SQL语句执行查询，并将查询结果填充到数据表ServerDataSet.Bill_Report，然后报表控件装载报表并显示。

整个实现过程如图 7.20 所示。

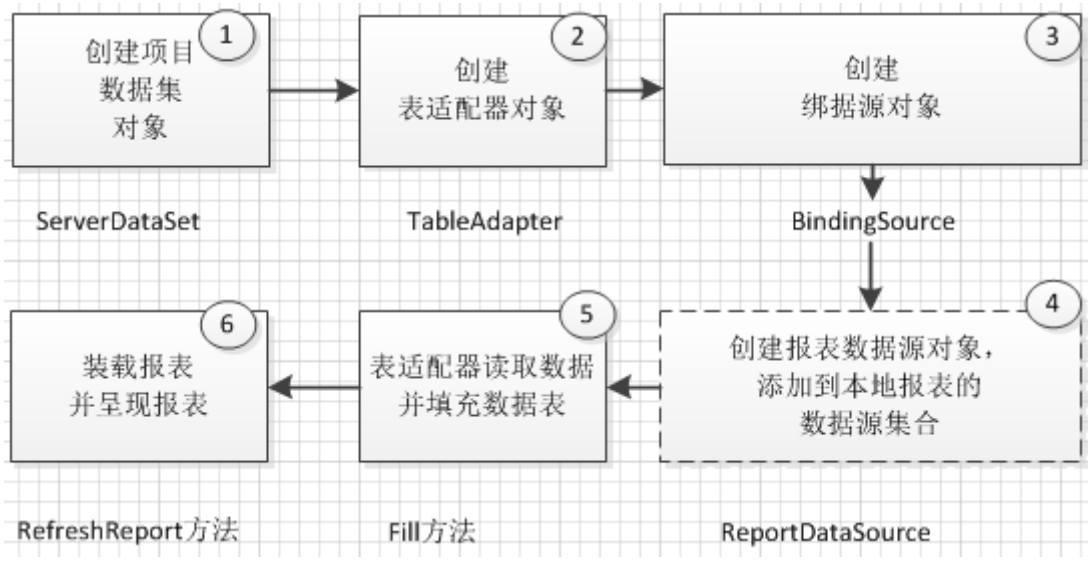


图 7.20　报表显示过程

在图 7.20 中我们会发现有一个 ReportDataSource 对象，这个就是报表文件的数据源对

象，报表控件将从该对象中获取报表文件所要求的数据，而它的数据由 BindingSource 提供。这个对象没有以组件形式出现在窗体下方，但在窗体的后台代码文件 Test.Designer.cs 中可以找到。ReportDataSource 需要设置了两个重要的属性，一个是 Name 属性，对应报表文件中的数据集名称，另一个是 Values 属性，对应绑定源 BindingSource 对象，在报表中只能由 BindingSource 或 DataTable 对象为其赋值。创建好的 ReportDataSource 对象，由报表控件的本地报表对象将其数据源集合中，这样报名控件在装载报表文件后才能显示报表文件的内容。

ReportDataSource 位于 Microsoft.Reporting.WinForms 命名空间中。

理解了这些关系后，为了实现共享报表控件来显示不同报表文件，参照在窗体自动创建的组件和自动生成的代码，我们可以使用代码来动态显示报表，其过程总结为：① 创建项目数据集对象，根据报表文件所需数据，创建并使用表适配器对象器读取数据并填充数据集；② 创建绑定源对象，设置其 DataSource 和 DataMember 属性分别为数据集对象和数据表名称。③ 创建 ReportDataSource 对象，并设置它的的 Name 和 Value 属性。④ 将 ReportDataSource 对象添加到报表控件的本地报表的数据源集合中，注意还需要设置本地报表绑定的报表文件名。最后调用报表控件的 RefreshReport 方法装载报表并呈现数据。

本例程序代码中的自定义方法 SetReportSource 就是按照上述思路和步骤来实现的。代码中，根据表名 dsTableName 和数据集对象 ds 作为绑定源据控件 bs 的属性值，而 bs 又作为报表数据源对象 rd 的数据来源，最后本地报表对象 reportViewer1.LocalReport 将 rd 添加到其数据源集合中，并设置 reportViewer1.LocalReport 的要装载的报表文件 ReportName。在数据集得到数据填充后，调用报表视图控件的 RefreshReport 方法，显示报表内容。这就是一个报表视图控件显示不同报表文件的代码实现。

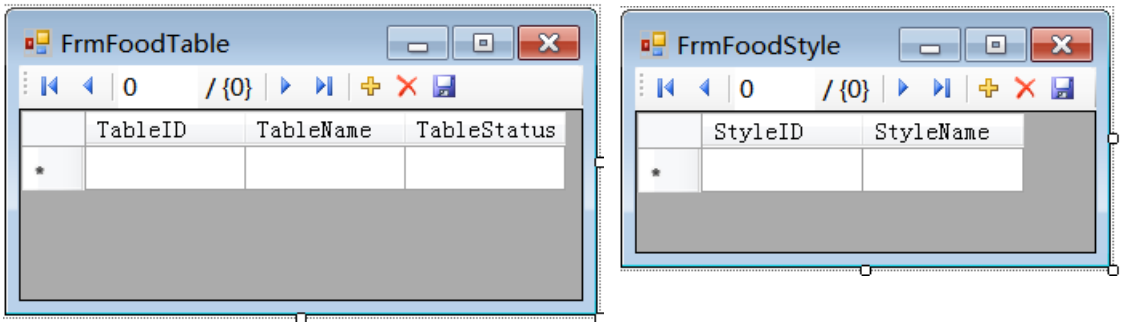
注意：使用项目中数据集和表适配器对象来查询数据完全是为了使用方便。也可以自己创建DataSet和SqlDataAdapter对象，连接数据库，执行SQL查询来填充数据到DataSet，而后作为绑定源控件的数据来源，不过这种方法的开发效率显然低于使用数据集设计器来完成。

课程设计参考-基础数据编辑

本项目的基础数据包括：餐台信息、菜品分类和菜品信息。系统应该首先实现基础数据的维护功能，因为数据库有了基础数据，才可以进行下一步的编写代码和调试。但由于基础数据内容比较少，功能实现也比较简单，因此放在本节来介绍。

1. 餐台信息编辑窗体 FrmFoodTable 和菜品分类 FrmFoodStyle 编辑窗体

餐台信息表 TableInfo 只包含 TableID(餐台编号)、TableName(餐台名称)和 TableStatus(状态)三个字段，而菜品分类表 FoodStyle 更简单，只有 StyleID(菜系编号)和 StyleName(菜系名称)两个字段。由于需要维护的信息非常少，为了提高开发效率，减少编写代码的工作量，实现快速开发，因此在创建窗体后，在“数据源”窗体中直接将显示的数据表拖放到窗体，自动生成表格控件、数据导航控件、绑定源组件以及表适配器等组件，然后让用户直接在表格控件中实现对数据的各种操作，这样基本无须编写代码就可实现数据的“增删改查”功能。FrmFoodTable 和 FrmFoodStyle 两个窗体的界面布局分别如图 7.21(a)和 7.21(b)所示。



(a) FrmFoodTable 窗体界面布局　　(b) FrmFoodStyle 窗体界面布局

图 7.21　基础数据维护窗体

2.　菜品信息编辑窗体 FrmFood 设计

菜品信息编辑窗体 FrmFood 负责菜品表 FoodItem 的数据维护，在该窗体中实现了对数据“增删改查”的典型操作。与结账窗体 FrmBill 一样，都作为 MDI 主窗体的子窗体运行。运行效果如图 7.22 所示。

图 7.22　菜品信息编辑窗体运行效果

以下是菜品信息编辑窗体 FrmFood 的设计与实现。

界面布局

为 Server 项目新建窗体，保存为 FrmFood。添加 1 个 Panel 控件，停靠在窗体顶端，并在其中添加 1 个文本框和 1 个按钮；添加 1 个 Panel 控件，停靠在窗体底部，并在其中添加 5 个按钮；添加一个 DataGridView 控件，填充窗体剩余控件，各个控件的名称和文本如图如图 7.23 中的界面布局所示。

在属性窗体中为表格控件 dgv 添加 8 列，对应菜品表 FoodItem 的所有字段，以及菜系表 FoodStyle 中的菜系名称，将表格各列的列名设置与表字段名一一对应，列标题修改为中文显示。

图 7.23　FrmFood 窗体界面布局

FrmFood 窗体实现的功能分析：① 在菜品信息编辑窗体中，窗体刚打开和执行窗体中的数据刷新功能时，都需要将菜品表的所有数据显示到表格中；而查找功能是根据按菜品名称进行模糊匹配查询，并在表格中显示查询结果。两种操作的不同之处在于，前者不需要查询条件，而后者需要根据条件查询，因此可以将数据显示功能设计为带查询条件参数的方法；② 删除操作是将选中的数据行从表格中移除，并同步更新数据库；③ 添加和修改的操作结果都要同时更新表格和数据库。不同的是，修改操作是将用户选中的行数据显示在修改界面以便核对修改，修改完毕后要退出修改界面；而添加操作需要初始化界面控件，保存数据后可以继续添加数据，直到用户关闭界面，由于两者大部分代码重叠，因此，这里让添加和修改的功能共享同一个窗体 FrmFoodItem 来实现，如图 7.24 所示。

图 7.24　FrmFoodItem 窗体界面布局

添加功能实现的思路是：打开 FrmFoodItem 窗体，初始化界面，检查录入数据的有效性后将数据保存到数据库，同时更新表格的数据显示。可连续添加数据，直到用户执行退出操作才关闭窗体。

修改功能实现的思路是：打开 FrmFoodItem 窗体前，将在表格中选中的数据，传递到该窗体进行修改，在检查数据的有效性后将数据更新到数据库，同时更新表格的数据显示，操作完毕后即关闭窗体。

在 FrmFoodItem 窗体中，如何判断当前要执行的操作是添加还是修改呢？这根据在 FrmFoodItem 窗体中定义了的公共整型字段 SelectIndex 来决定。如果该窗体用于执行修改操

作，在创建该窗体对象并准备显示时，将在表格中选中的行下标赋予该变量；否则，赋予该变量为-1，表示执行添加操作。为了同步显示主窗体表格控件的数据，将表格对象也一起传递到该窗体，以实现在该窗体直接操作主窗体的表格。

程序代码-菜品信息编辑窗体 FrmFood.cs 类文件

```
//省略自动生成的using部分
using System.Data.SqlClient;
namespace Server
{
    public partial class FrmFood : Form
    {
        public FrmFood()  {  InitializeComponent();  }//构造方法

        //1.初始化表格
        void InitGridView()
        {
            dgv.AllowUserToAddRows = false;//不允许直接添加行
            dgv.AllowUserToDeleteRows = false;//不允许直接删除
            dgv.AllowUserToResizeRows = false;//行高不可调整
            dgv.MultiSelect = false;//单行选择
            dgv.SelectionMode = DataGridViewSelectionMode.FullRowSelect;//完全行选择
            dgv.ReadOnly = true;//只读，不能直接在表格修改数据
            dgv.RowHeadersVisible = false;//隐藏行标头
        }

        //2.填充数据到表格，与"查找"功能合并
        void FillGridView(string OtherWhere="")//2.1 可选参数，仅用于"查找"功能
        {
            dgv.Rows.Clear();//清空表格
            int rowIndex = 0;//记录行下标的变量

            string sql = "Select FoodItem.* ,FoodStyle.StyleName from FoodItem,FoodStyle ";
            sql += " Where FoodStyle.styleID=FoodItem.StyleID";
            sql += OtherWhere;//仅"查询"功能带条件子句OtherWhere，默认为空字符串
            SqlDataReader dr = DBHelper.GetDataReader(sql);
            while (dr.Read())
            {
                dgv.Rows.Add();
                for (int i = 0; i < dr.FieldCount; i++)
                {
                    string colName = dr.GetName(i); //取得列名
                    //2.2 使用列名引用单元格
                    dgv.Rows[rowIndex].Cells[colName].Value = dr[colName];
                }
```

```
            rowIndex++;
        }
        dr.Close();
}
//3.Load事件处理过程
private void FrmFoodItemList_Load(object sender, EventArgs e)
{
        InitGridView();//初始化表格样式
        FillGridView();//读取数据，填充表格
}
//4."刷新"按钮的事件处理过程
private void btRefresh_Click(object sender, EventArgs e)
{
        FillGridView();//重新读取数据，填充表格
}

//5."添加"按钮的事件处理过程
private void btAdd_Click(object sender, EventArgs e)
{
        FrmFoodItem frm = new FrmFoodItem();
        frm.dgv = this.dgv; //5.1 传递表格对象
        frm.SelectIndex = -1;// 5.2 -1表示准备添加数据
        frm.ShowDialog();
}

//6."修改"按钮的事件处理过程
private void btUpdate_Click(object sender, EventArgs e)
{
        if (dgv.CurrentRow == null) return;//6.1 只有选中行才可以修改
        FrmFoodItem frm = new FrmFoodItem();
        frm.dgv = this.dgv;
        frm.SelectIndex = dgv.CurrentRow.Index;//6.2 当前选中的行下标
        frm.ShowDialog();
}
//7."删除"按钮的事件处理过程
private void btDel_Click(object sender, EventArgs e)
{
        if (dgv.CurrentRow == null) return;//只有选中行才可以删除
        //7.1删除数据库中的数据
        string sql = "Delete From FoodItem Where FoodID=";
        sql+=dgv.CurrentRow.Cells["FoodID"].Value;
        DBHelper.ExecSql(sql);

        //7.2从表格中移除当前行
```

```
            dgv.Rows.Remove(dgv.CurrentRow);
        }
        //8."关闭"按钮的事件处理过程
        private void btClose_Click(object sender, EventArgs e)
        {
            Close();
        }
        //9."查找"按钮的事件处理过程
        private void btFind_Click(object sender, EventArgs e)
        {
            FillGridView(" AND FoodName Like '%"+txtFoodName.Text+"%'");
        }
    }//类结束
}//命名空间结束
```

代码说明

(1) 注释1：自定义方法InitGridView，用于设置表格控件的相关属性，使表格控件仅仅起到数据显示的作用，限制用户不能直接在表格控件中进行数据编辑等操作，这些操作通过功能按钮或在新窗体中进行。此外，这些属性也都可以在表格的属性窗体中设置。

(2) 注释2：自定义方法FillGridView，实现读取菜品表FoodItem的数据并填充到表格控件中。由于菜系名称FoodName位于菜系表FoodStyle，因此要构建实现连接查询的SQL语句。注释1.1中，自定义方法设计为带可选参数OtherWhere，目的是为了在实现查询功能时，根据条件查询显示得到的数据。调用该方法时，如果参数为空(默认)，也就是没有附加的查询条件，则显示全部数据；注释2.2中，由于表格的列名设置了与字段名一致，因此，可以通过Cells[colName]的方式来使用列名引用表格的单元格，而dr[colName]是根据Select语句中的字段名读取数据，两者一一对应，方便赋值。

(3) 注释 5、注释 6：“添加”和“修改”按钮的事件处理过程，都实现打开 FrmFoodItem 窗体，并将表格控件传递到该窗体，使得在该窗体中可以直接引用表格控件，即在 FrmFoodItem 窗体中可以直接操作表格控件来更新表格数据。注释 5.2 和注释 6.2 中的 SelecIndex 是打开 FrmFoodItem 窗体时用来判断当前操作是添加还是修改数据，如果是添加数据操作，传递-1 作为标识；如果是修改操作，传递在表格中选中的行下标，以便取得该行数据来显示。

(4) 注释 7：“删除”按钮的事件处理过程，实现删除在表格控件中选中的行，表格控件的 CurrentRow 属性代表当前选择行，如果表格控件没有数据或没选中行，该值为 null。与修改操作类似，执行删除操作前必须判断是否存在当前行。删除数据操作的结果是从表格控件中移除选中行，并更新数据库。

(5) 注释 9：“查找”按钮的事件处理过程，实现根据文本框输入的内容，以模糊匹配方式构建查询语句中附加的条件子句，作为自定义方法 FillGridView 的参数，实现在表格控件中显示按条件查询的结果。

以下是菜品项窗体 FrmFoodItem 的实现代码。

程序代码-菜品项窗体 FrmFoodItem.cs 类文件

```
//省略自动生成的using部分
using System.IO;
```

```
using System.Data.SqlClient;
namespace Server
{
    public partial class FrmFoodItem : Form
    {
        public FrmFoodItem()  {  InitializeComponent();  }//构造方法
        public DataGridView dgv = null;//1.由主窗体传入
        public int SelectIndex = -1;//2.操作标记：>=0 修改数据；-1 新增数据
        string ImageFileName = ""; //3.保存图片名称

        //4.自定义方法，取得菜系表所有数据,添加到组合框
        void AddComb()
        {
            cmbStyle.Items.Clear();
            SqlDataReader dr = DBHelper.GetDataReader("select * from FoodStyle");
            while (dr.Read())
            {
                //自定义格式显示菜系信息：ID-名称
                cmbStyle.Items.Add(dr["StyleID"] + "-" + dr["StyleName"]);
            }
            dr.Close();
            if (cmbStyle.Items.Count > 0) cmbStyle.SelectedIndex = 0; //默认选择第1项
    }

        //5.修改操作时，显示选择的数据，以便修改
        void ShowUpdateData()
        {
            //显示当前需要修改的数据
            if (SelectIndex >= 0)//修改操作
            {
                //修改时，显示选中项
                cmbStyle.Text = dgv.Rows[SelectIndex].Cells["StyleID"].Value + "-" +
                dgv.Rows[SelectIndex].Cells["StyleName"].Value;//5.1 自动选中项
                txtFoodName.Text =
                dgv.Rows[SelectIndex].Cells["FoodName"].Value.ToString();

                txtPrice.Text = dgv.Rows[SelectIndex].Cells["Price"].Value.ToString();
                chkHot.Checked = (bool)dgv.Rows[SelectIndex].Cells["IsHot"].Value;
                ImageFileName = dgv.Rows[SelectIndex].Cells["ImagePath"].Value.ToString();
                picImg.Image = LoadImage(ImageFileName);//5.2装载图片文件
            }
        }

        //6.Load事件处理过程
```

```
private void AddFoodItem_Load(object sender, EventArgs e)
{
    AddComb();
    ShowUpdateData();
}

//7.数据验证
bool CheckData()
{
    if (cmbStyle.Items.Count == 0) return false;
    if (txtFoodName.Text == "") return false;
    float price;
    if (!float.TryParse(txtPrice.Text, out price)) return false;//7.1是否可以转换为实数
    return true;
}

//8.自定义方法,添加记录,并返回记录的标识
int InsertFoodItem(FoodItem fi)
{
    string sql = "Insert Into FoodItem(styleId,FoodName,price,ImagePath,IsHot) ";
    sql += " Output Inserted.FoodID ";//8.1需要返回新插入数据后的标识列的值
    sql += " Values(";
    sql += fi.StyleID + ",";
    sql += "'" + fi.FoodName + "',";
    sql += fi.Price + ",";
    sql += "'" + fi.ImagePath + "',";
    sql += fi.IsHot;
    sql += ")";
    int FoodID = (int)DBHelper.GetOne(sql);//8.2 执行SQL语句，返回单值结果
    return FoodID;
}

//9.自定义方法,修改记录
void UpdateFoodItem(FoodItem fi)
{
    string sql = "Update FoodItem Set ";
    sql += " StyleId=" + fi.StyleID + ",";
    sql += "FoodName='" + fi.FoodName + "',";
    sql += " price=" + fi.Price + ",";
    sql += " ImagePath='" + fi.ImagePath + "',";
    sql += " IsHot=" + fi.IsHot;
    sql += " Where FoodID=" + dgv.Rows[SelectIndex].Cells["FoodID"].Value;
    DBHelper.ExecSql(sql);
}
```

//10.自定义方法,更新表格显示

```
void UpdateGridView(FoodItem fi)
{
    int count = dgv.Rows.Count;//10.1 先默认添加操作，取得当前行数
    if (SelectIndex >= 0) count = SelectIndex;//10.2 如修改操作，则改变值
    else   dgv.Rows.Add();//10.3 添加新行

    dgv.Rows[count].Cells["FoodID"].Value = fi.FoodID;
    dgv.Rows[count].Cells["FoodName"].Value = fi.FoodName;
    dgv.Rows[count].Cells["StyleID"].Value = fi.StyleID;
    dgv.Rows[count].Cells["StyleName"].Value = fi.StyleName;
    dgv.Rows[count].Cells["Price"].Value = fi.Price;
    dgv.Rows[count].Cells["ImagePath"].Value = fi.ImagePath;
    dgv.Rows[count].Cells["IsHot"].Value = fi.IsHot;
    dgv.Rows[count].Cells["ClickCount"].Value = fi.ClickCount;
}
```

//11.将选择的图片复制到当前路径的Images文件夹中

```
void SaveImage()
{
    if (ImageFileName != "")
    {
        string CurImagePath = Application.StartupPath+"\\Images";
        if (!Directory.Exists(CurImagePath)) Directory.CreateDirectory(CurImagePath);
        File.Copy(picImg.Tag.ToString(), CurImagePath + "\\" + ImageFileName, true);
    }
}
```

//12.取得指定的图片文件

```
Image LoadImage(string FileName)
{
    string CurImagePath = Application.StartupPath + "\\Images";
    if(!File.Exists(CurImagePath+"\\"+FileName)) return null;
    return Image.FromFile(CurImagePath+"\\"+FileName);
}
```

//13."选择图片"按钮的事件处理过程

```
private void btImg_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "*.jpg|*.jpg|*.png|*.png";//选择图片文件
    openFileDialog1.FileName = "";
    openFileDialog1.Multiselect = false;
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
```

```
    {
        string filename = openFileDialog1.FileName;
        picImg.Tag = filename;//保存全路径名，以便复制文件
        picImg.Image = Image.FromFile(filename);
        FileInfo fi = new FileInfo(filename); //取得文件信息
        ImageFileName = fi.Name;//取得文件名
        //去掉扩展名，让菜品名称默认为图片名称
        txtFoodName.Text = fi.Name.Replace(fi.Extension, "");
    }
}
```

//14."保存"按钮的事件处理过程

```
private void btSave_Click(object sender, EventArgs e)
{
    if (!CheckData())
    {
        MessageBox.Show("数据不完整或有误！", "提示");
        return;
}

    //14.1 获取录入的数据项
    FoodItem foodItem = new FoodItem();
    foodItem.FoodID = -1;//添加数据时还没有获取该值，暂时设置为默认值-1
    foodItem.FoodName = txtFoodName.Text;
    foodItem.StyleID = int.Parse(cmbStyle.Text.Split('-')[0]);//14.2 取得菜系ID
    foodItem.StyleName = cmbStyle.Text.Split('-')[1];//14.3 取得菜系名称
    foodItem.Price = float.Parse(txtPrice.Text);
    foodItem.IsHot = chkHot.Checked ? 1 : 0;
    foodItem.ImagePath = ImageFileName;
    foodItem.ClickCount = 0;//添加数据时默认为0

    SaveImage();//保存图片
    if (SelectIndex >= 0)//假如是修改操作
    {
        //修改数据
        foodItem.FoodID = (int)dgv.Rows[SelectIndex].Cells["FoodID"].Value;
        foodItem.ClickCount = (int)dgv.Rows[SelectIndex].Cells["ClickCount"].Value;
        UpdateFoodItem(foodItem);//更新数据库
        UpdateGridView(foodItem);//更新表格数据显示
        Close();//更新数据后需要关闭窗体
        return;//修改完毕退出，不执行后面语句
    }

    //添加新项操作
```

```
            foodItem.FoodID = InsertFoodItem(foodItem);//插入记录后获取自动编号
            UpdateGridView(foodItem);//更新表格数据显示
            txtFoodName.Text = "";    txtPrice.Text = "";    ImageFileName = "";
        }

        //15."关闭"按钮事件处理过程
        private void btClose_Click(object sender, EventArgs e)
        {
            Close();
        }
    }//类结束
}//命名空间结束
```

代码说明

(1) 注释 1：保存外部传递过来的数据表格对象。

(2) 注释 2：标识字段，用于判断窗体当前要执行的操作是添加还是修改。

(3) 注释 3：保存外部传递过来的图片文件名称。

(4) 注释 4：自定义方法 AddComb，实现将菜系表 FoodStyle 的信息以“ID-名称”的格式添加到组合框，如果有数据则默认选择第 1 项。

(5) 注释 5：自定义方法 ShowUpdateData，在修改状态中，将选中数据行的内容显示到界面相应的控件中。注释 5.1 中，将组合框的 Text 属性设置为存在的项，实现自动选中该项；注释 5.2 中，调用自定义方法 LoadImage，装载指定的图片文件并将其显示在图片控件中。

(6) 注释 7：每个数值类型都有 TryParse 方法，该方法在类型转换无效时，不会出现异常，仅返回 false 值代表转换无效，否则，会将转换结果保存在该方法的输出参数中。

(7) 注释 8：构建插入记录时返回标识字段值的 SQL 语句。“Output Inserted”为 SQL Insert 语句中的关键字，必须置于 Values 关键字前，Inserted 后面带一个“.”，后紧跟标识字段名，当使用在注释 8.2 位置的方法执行该 SQL 语句后，将返回新增记录的标识字段值。

(8) 注释 10：添加或修改数据，并更新数据表格。注释 10.1~注释 10.3，根据对公共整形字段 SelectIndex 的判断，来决定是否在表格控件中增加新行。

(9) 注释 11：将选择的图片文件复制到当前应用程序所在位置的 Images 文件夹中，如果 Images 文件夹不存则创建。

(10) 注释 12：自定义方法 LoadImage，实现按给定的图片文件名，判断文件是否存在，存在则装载图片文件并返回 Image 对象。

(11) 注释13：设置“打开文件对话框”组件只能选择jpg和png类型的图片文件，并将选择的图片显示在图片控件中，同时将图片名称保存为菜品的名称。注释13.1中tag属性几乎是每个控件具有的object类型属性，可以在该属性保存额外的任意类型数据，这里用来保存图片文件的绝对路径，以便在SaveImage方法中找到该文件来保存。

(12) 注释14：cmbStyle.Text.Split('-')[0] 语句表示将组合框显示的文本，按字符“-”拆分成字符串数组，其中下标0表示数组的第1个元素，该语句相当于：

```
string[] s= cmbStyle.Text.Split('-');
foodItem.StyleID=(int)s[0];
```

这里组合框的文本cmbStyle.Text的格式为“菜系ID-菜系名”。

同样，cmbStyle.Text.Split('-')[1]语句也是按字符“-”拆分成字符串数组，但取的代表菜

系名称的第2个元素。

(13) 代码中使用了公共类 FoodItem，该类作为实体类使用，用于保存菜品表的一条记录。下面是 FoodItem 类的代码：

```
public class FoodItem
{
    public int FoodID { get; set; }
    public string FoodName { get; set; }
    public int StyleID { get; set; }
    public float Price{get;set;}
    public string ImagePath { get; set; }
    public int IsHot{get;set;}
    public int ClickCount{get;set;}
    //为了方便使用，添加对应FoodStyle表中的字段StyleName
    public string StyleName { get; set; }
}
```