全国高等职业教育计算机类规划教材·实例与实训教程系列

C# 项目实训教程

于润伟　主　编
温东新　主　审

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书在讲解必要的 C#基础知识后，结合六个案例详细说明 C#软件开发流程、界面和控件的应用、串口通信软件的开发、链接数据库和使用水晶报表等知识技能点，注重精讲多练，配备丰富的例题和习题，便于读者学习及领会 C#的编程方法和应用技巧。

本书可作为高职高专院校软件工程、网络技术、计算机应用等专业的教材，也可作为广大科技工作者、教师学习 C#的参考书。为方便教学，本书配有电子教案和程序源代码及部分习题答案。

图书在版编目(CIP)数据

C#项目实训教程/于润伟主编．—北京：电子工业出版社，2009.2
全国高等职业教育计算机类规划教材·实例与实训教程系列
ISBN 978-7-121-08135-4

Ⅰ.C… Ⅱ.于… Ⅲ.C 语言－程序设计－高等学校：技术学校－教材 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2009）第 008054 号

责任编辑：赵云峰 刘少轩
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱 邮编 100036
开　　本：787×1 092 1/16 印张：11 字数：280 千字
印　　次：2009 年 2 月第 1 次印刷
印　　数：4 000 册 定价：18.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

序

20 世纪 90 年代以来，以计算机和通信技术为推动力的信息产业在我国获得了前所未有的发展，全国各企事业单位对信息技术人才求贤若渴，高等教育计算机及相关专业毕业生供不应求。随后几年，我国各高等院校、众多培训机构相继开设计算机及相关专业，积极扩大招生规模，不久即出现了计算机及相关专业毕业生供大于求的局面。纵观近十年的就业市场变化，计算机专业毕业生经历了“一夜成名、求之不得”的宠幸，也遭遇了“千呼百应、尽失风流”的冷落。

这个时代深深地镌刻着信息的烙印，这个时代是信息技术人才尽情展示才能的舞台。目前我国的劳动力市场，求职人数过剩，但满足企业要求的专业人才又很稀缺。这种结构性的人才市场供求矛盾是我国高等教育亟待解决的问题，更是“以人为本，面向人人”为目标的职业教育不可推卸的责任。

电子工业出版社，作为我国出版职业教育教材最早的出版社之一，是计算机及相关专业高等职业教材重要的出版基地。多年来，我们一直在教材领域为战斗在职业教育第一线的广大职业院校教育工作者贡献着我们的力量，积累了丰富的职业教材出版经验。目前，计算机专业高等教育正处于发展中的关键时期，我们有义务、有能力协同全国各高等职业院校，共同探寻适合社会发展需要的人才培养模式，建设满足高等职业教育需求的教学资源——这是我们出版“全国高等职业教育计算机类规划教材·实例与实训教程系列”的初衷。

关于本系列教材的出版，我们力求做到以下几点：

（1）面向社会人才市场需求，以培养学生技能为目标。工学结合、校企结合是职业教育发展的客观要求，面向就业是职业教育的根本落脚点。本系列教材内容体系的制定是广大高职教育专家、一线高职教师共同智慧的结晶。我们力求教材内容丰富而不臃肿、精简而不残缺，实用为主、够用为度。

（2）面向高职学校教师，以方便教学为宗旨。针对每个课程的教学特点和授课方法，我们为其配备相应的实训指导、习题解答、电子教案、教学素材、阅读资料、程序源代码、电子课件、网站支持等一系列教学资源，广大教师均可从华信教育资源网（www.huaxin.edu.cn）免费获得。

（3）面向高职学校学生，以易学、乐学为标准。以实例讲述理论、以项目驱动教学是本系列教材的显著特色。这符合现阶段我国高职学生的认知规律，能够提高他们的学习兴趣，增强他们的学习效果。

这是一个崭新的开始，但永远没有尽头。高等职业教育教材的建设离不开广大职业教育工作者的支持，尤其离不开众多高等职业院校教师的支持。我们诚挚欢迎致力于职业教育事业发展的有识之士、致力于高等职业教材建设的有才之士加入到我们的队伍中来，多批评，勤点拨，广结友，共繁荣，为我国高等职业教育的发展贡献我们最大的力量！

电子工业出版社高等职业教育分社

前　言

C#是微软公司推出的新一代编程语言，具有简洁、灵活、安全、面向对象和高度兼容性等特点，兼具 Visual BASIC 的高效性和 Visual C++的强大功能，是微软.NET 战略的重要组成部分，广泛应用于数据库管理、Web 程序设计、软件开发等方面，为 C#程序员提供了广泛的就业机会。

本教材基于 Visual Studio 2008 开发平台，以 C/S（服务器/客户端）方式为主。在简单讲解 C#基础知识后，选择工资计算器、串口调试器、八数码游戏、电池生产流转单管理系统、锂电池成本核算系统、财务管理系统六个案例，详细说明 C#软件的开发流程、界面和控件的应用、串口通信软件的开发、链接数据库和使用水晶报表等知识技能点，便于读者学习及领会 C#的编程方法和应用技巧。具体内容如下：

第 1 章：讲述数据类型、表达式、程序的运行和调试方法等基础知识，让读者学会安装 Visual Studio 2008 开发平台，熟悉开发环境和程序调试技巧。

第 2 章：以企业工资计算器为例，说明 Label、TextBox、Button 控件，Container、MessageBox、Convert 类，if-else、using 语句的应用，介绍软件窗体的设计方法。

第 3 章：学习使用 for、try-catch 语句和第三方控件 MsComm 设计工业机调试串口通信的调试器。

第 4 章：通过制作小游戏软件，学习 Panel、GroupBox、TabControl 控件，Point、Random 类，#region、while 语句的应用。

第 5 章：主要讲解 MainMenu、RadioButton、ComboBox 控件，DataSet 等类的使用，重点是链接 Access 数据库，以及查询、录入、删除数据的方法。

第 6 章：主要讲解多级别登录的实现，水晶报表控件、BindingManagerBase 类、switch 语句的使用方法，重点是学习多级别登录的实现方法。

第 7 章：讲解 ToolBar、ImgeList 控件，SqlConnection、SqlCommand 类，return 语句的使用，重点是链接 SQL Server 数据库，实现数据添加、查询、编辑与删除的方法。

结合目前高职高专的教学特点，建议总学时为 78 学时，其中课堂教学 40 学时、上机练习 14 学时、实训教学 24 学时。

本书由黑龙江农业工程职业学院于润伟、侯南、哈尔滨光宇集团张文多编写，哈尔滨工业大学计算机科学系温东新主审。在编写过程中，得到了哈尔滨工业大学固泰电子股份有限公司、哈尔滨光宇集团自动化公司的大力支持，在此表示真诚的谢意。

由于编者水平有限，对一些问题的理解和处理难免有不当之处，衷心希望使用本书的读者批评指正。

编　者

2008 年 9 月

目　录

第 1 章　C#概述

本章介绍 C#软件的来历、特点和开发环境，讲述数据类型、表达式、程序的运行和调试方法等基础知识，让读者学会安装 Visual Studio 2008 开发平台，熟悉开发环境和程序调试技巧，以一个结构完整、内容简单的 C#程序作为实训项目，使读者能够读懂程序，学会使用开发平台编辑、编译并运行、调试程序。

1.1　认识C#

C#和 VC#都是 Visual C#语言的简称。C#诞生于 2000 年，2001 年被 ECMA（欧洲计算机制造者协会）规定为高级语言开发标准（ECMA-334），2003 年被 ISO（国际标准化组织）规定为高级语言开发标准（ISO/IEC23270）。

1.1.1　C#的来历

1995 年美国 SUN 公司推出面向对象的开发语言 Java，由于具有跨平台、跨语言的特点，一些基于 C/C++的应用开发人员转向从事基于 Java 的应用开发，使微软公司在软件开发领域受到了前所未有的冲击。为了迎接挑战，微软另辟蹊径，决定推出其进军互联网的.NET 计划。.NET 代表了一个集合、一个环境、一个编程的基本结构，作为一个开发平台支持下一代互联网，这是一项非常庞大的计划，也是微软今后发展的战略核心。C#是.NET 的关键性语言，是整个.NET 平台的基础。由于 C#与 Windows 的体系结构相似，因此 C#很容易被开发人员所熟悉。

微软公司在 2000 年 6 月举行的“职业开发人员技术大会”上正式发布了 VC#语言，其英文名为 VC-Sharp。微软公司对其定义是：“VC#是一种类型安全的、现代的、简单的，由 C 和 C++衍生出来的面向对象的编程语言，它是牢牢根植于 C 和 C++语言之上的，并可立即被 C 和 C++开发人员所熟悉。VC#的目的就是综合 Visual BASIC 的高生产率和 C++的行动力。”

2002 年 1 月，微软公司公布.NET Framework 1.0 正式版，与此同时，Visual Studio.NET 2002 也同步发行。2003 年 4 月 23 日，微软公司推出.NET Framework 1.1 和 Visual Studio.NET 2003。2004 年 6 月，在 TechEd Europe 会议上，微软发布.NET Framework 2.0 Beta1 和 Visual Studio 2005 Beta1。2005 年 4 月，微软公司发布 Visual Studio 2005 Beta2 测试版。2005 年 11 月，微软公司发布 Visual Studio 2005 和 SQL Server 2005 正式版。

2007 年 11 月底，微软公司发布 Visual Studio 2008 和.NET Framework 3.5。Visual Studio 2008 是面向 Windows Vista、Office 2007、Web 2.0 的下一代开发工具，代号“Orcas”，经历了大约 18 个月的开发，是对 Visual Studio 2005 全方位的升级，引入了 250 多个新特性，整合了对象、关系型数据、XML（eXtensible Markup Language，可扩展标记语言）的访问方式，语言更加简洁、高效。

1.1.2 C#的特点

C#在带来对应用程序快速开发的同时，并没有牺牲 C/C++程序员所关心的各种特性。对于 C/C++程序员来说，C#依然能够熟练应用。而对于初级学习者来说，快速应用程序开发的思想与简洁的语法将会使学习者迅速成为一名熟练的开发人员。另外，C#是专门为.NET 应用而开发出的语言，这从根本上保证了 C#与.NET 框架的完美结合，在.NET 运行库的支持下，.NET 框架的各种优点在 C#中表现得淋漓尽致。

1．简洁的语法

C#代码在.NET 框架提供的“可操纵”环境下运行，不允许直接内存操作。其最大的特色是没有了指针，与此相关的是那些在 C++中被疯狂使用的操作符已经不再出现（例如：“::”、“→”和“;”，C#只支持一个“.”）。语法中的冗余是 C++中的常见的问题，C#对此进行了简化，只保留了常见的形式，其他的冗余形式从其语法结构中被清除了出去。

2．面向对象设计

C#具有面向对象语言所应有的一切特性：封装、继承和多态性。在 C#的类型系统中，每种类型都可以看做一个对象，且只允许单继承，即一个类不会有多个基类，从而避免了类型定义的混乱。C#中没有全局函数、全局变量和全局常数，所有的都封装在一个类之中，这使得代码具有更好的可读性，并且减少了发生命名冲突的可能。

3．与Web的紧密结合

SOAP（Simple Object Access Protocal，简单对象访问协议）的使用，使得 C#能与 Web 紧密结合，使大规模深层次的分布式开发成为可能。由于有了 Web 服务框架的帮助，对程序员来说，网络服务看起来就像是C#的本地对象，程序员们能够利用已有的面向对象的知识与技巧开发Web服务，仅需要使用简单的 C#语言结构和 C#组件就能够方便地为 Web 服务，并允许通过 Internet 被运行在任何操作系统上的任何语言所调用。例如 XML 已经成为网络中数据结构传递的标准，C#允许直接将 XML 数据映射成为结构，这样就可以有效地处理各种数据。

4．完整的安全性与错误处理

C#先进的设计思想可以消除软件开发中的许多常见错误，并提供了包括类型安全在内的安全性检查。为了减少开发中的错误，C#会帮助开发者通过更少的代码完成相同的功能，这不但减轻了编程人员的工作量，同时更有效地避免了错误的发生。C#中不能使用未初始化的变量，对象的成员变量由编译器将其设置为零，当局部变量未经初始化而被使用时，编译器将做出提示。

5．版本处理技术

升级软件系统中的组件（模块）是一件容易产生错误的工作。在代码修改过程中可能对现存的软件产生影响，很有可能导致程序的崩溃。为了帮助开发人员处理这些问题，C#在语言中内置了版本控制功能。例如函数重载必须被显示声明，而不会像在 C++或 Java 中经常发生的那样不经意地进行，这可以防止代码级错误的出现，还能保留版本化的特性。另一个相关的特性是对接口和接口继承的支持，这些特性可以保证复杂的软件可以被方便地开发和升级。

6．灵活性和兼容性

在简化语法的同时，C#并没有失去灵活性。例如 C#不能用来开发硬件驱动程序，在默认的状态下没有指针等。如果需要，C#允许将某些类或类的某些方法声明为非安全的，这样一来，就能够使用指针、结构和静态数组，并且调用这些非安全代码不会带来任何其他的问题。

C#允许与需要传递指针型参数的 API（Application Programming Interface，应用程序编程接口）进行交互操作，DLL（Dynamic Linkable Library，动态链接库文件）的任何入口点都可以在程序中进行访问。C#遵守.NET 公用语言规范（CLS），从而保证了 C#组件与其他语言组件间的互操作性。

1.2 C#开发环境

C#是 Visual Studio 2008 的一部分，同其他的.NET 语言一样，都必须在.NET 框架环境下运行。因此，要建立一个完整的 C#开发平台，必须安装 Visual Studio 2008 和.NET Framework SDK（Software Development Kit，软件开发工具包）。

Visual Studio 2008 开发环境采用由公共语言运行库和类库两部分组成的框架结构。其中公用语言运行库（CLR）是.NET Framework 的基础，提供诸如内存管理、线程管理和远程处理等核心服务，强制实施严格的类型安全检查，确保系统的安全性和可靠性；类库是一个由.NET Framework SDK 中包含的类、接口和值类型组成的库，提供对系统功能的访问，是建立应用程序、组件和控件的基础。

1.2.1 安装Visual Studio 2008

Visual Studio 2008 可以到微软网站下载试用版，共有 7 部分。需要注意的是安装过程可能很漫长，大概 1 个多小时（具体时间取决于计算机配置）。另外，安装 Visual Studio 2008 的同时会安装.NET3.5。如果之前安装过.NET3.5 测试版，请先卸载该版本，否则可能导致安装失败。下载完毕后执行以下安装操作：

（1）利用虚拟光驱将下载的文件打开。先单击 part1.exe 文件，自动合并 7 部分文件成为一个完整的 ISO（镜像文件，由多个文件通过刻录软件或者镜像文件制作工具制作而成的）文件包，使用虚拟光驱加载该 ISO 文件，然后运行安装程序（autorun 或 setup.exe）。加载后会自动弹出一个安装对话框，如果没有出现的话可到“我的电脑”里找到虚拟光驱将其打开。如果有安装盘，可以选择自动运行或者双击 autorun 或 setup.exe 进行安装，都会出现如图 1.1 所示的安装对话框。

图 1.1　安装对话框

图 1.2　加载文件

（2）复制文件到临时文件夹。单击对话框中的【安装 Visual Studio 2008】选项后，弹出如图 1.2 所示的加载文件对话框。

这是 Visual Studio 2008 正在加载安装文件的对话框，是第一个安装界面。加载安装文件后，会有提示是否要发送安装体验信息到 Microsoft，若选择不发送，直接单击【下一步】按钮。

（3）协议与安装密匙。这里需要选择是否同意许可协议，选择同意（不同意是不能安装的）。然后出现的是安装密匙，官方网站上提供的是试用版，安装密匙是填好的。输入用户名，然后单击【下一步】按钮。

（4）选择安装方式，如图 1.3 所示。

图 1.3　选择安装方式

此处可选择默认安装方式、完全安装方式或者是自定义安装方式。本书选择自定义安装方式，可以设置 Visual Studio 2008 的安装目录，设置完成后，单击【下一步】按钮。如果选择的是默认安装方式或者完全安装方式会跳过步骤（5）。

（5）选择安装组件，如图 1.4 所示。

图 1.4　选择安装组件

在自定义安装方式下，可以选择默认的安装组件，当然也可以根据实际情况来选择安装组件（如果有些没有安装的话，以后还可以修改添加）。选择完成以后，单击【安装】按钮。

（6）安装过程，如图 1.5 所示。

图 1.5　安装过程

在图 1.5 中，左边是将要安装的组件，左边下面的进程条显示当前组件的安装进度；右边是 Visual Studio 2008 的一些介绍。.NET Framework 3.5 的安装速度很慢，安装文件近 200MB。

（7）安装完成。安装完成的对话框如图 1.6 所示，单击【完成】按钮并退出。

图 1.6　安装完成

Visual Studio 2008 版界面相当漂亮，特别是在 Vista 系统下看起来相当华丽，和 Visual Studio 2005 相比较而言，占用资源要多的多，其他差别请参考相关文章。另外，Visual Studio 2008 自带了 Web Server，调试程序的时候再也不用担心虚拟目录等设置问题了。

需要指出的是，从微软官方网站下载的这个版本是 90 天试用版，90 天过期后需要购买正版密匙进行激活才能正常使用。

1.2.2　启动Visual Studio 2008

启动 Visual Studio 2008 一般有两种方法：一种是直接双击桌面上的 Visual Studio 2008 图

标启动；另一种是单击“开始”→“所有程序”→“Microsoft Visual Studio 2008”→“Microsoft Visual Studio 2008”进行启动。

1．起始页

首次启动 Visual Studio 2008，首先会看到如图 1.7 所示的“选择默认环境设置”界面。

图 1.7　选择默认环境设置

选择“Visual C#开发设置”，单击【启动 Visual Studio】按钮，屏幕上会弹出如图 1.8 所示界面，这是 Visual Studio 正在为第一次使用配置环境。

图 1.8　配置环境

配置环境完成后，会自动打开“起始页”，如图 1.9 所示。

图 1.9　起始页

其中有“最近的项目”、“开始”、“Visual Studio 标题新闻”、“Visual Studio 开发人员新闻”四个选项。在“最近的项目”部分中包含最近打开的链接，Visual Studio 2008 第一次被载入时，这一部分是空的；“开始”部分为开发人员提供帮助示例的搜索与下载；“Visual Studio 标题新闻”部分提供了浏览新闻和文章的链接；“Visual Studio 开发人员新闻”部分可以浏览 MSDN（微软开发人员网络）在线资源库。

2．创建新程序

如果要创建一个 C#程序，可以单击“最近的项目”部分中的“创建项目”，或者选择“文件”→“新建”→“项目”，将弹出如图 1.10 所示的对话框。

图 1.10　新建项目窗体

默认情况下，Visual Studio 2008 将项目和解决方案命名为 Windows Forms Application1，存储相关文件的默认位置是上次创建项目时所处的位置。Visual Studio 2008 第一次运行时，默认的文件夹是“My Documents”文件夹中的“Visual Studio 2008\Projects”文件夹，用户可以改变项目存储的文件夹名称和路径。在为项目选择名称和位置后，在“新建项目”对话框中单击【确定】按钮，IDE（Integrated Development Environment，集成开发环境）的外观将发生改变，如图 1.11 所示。

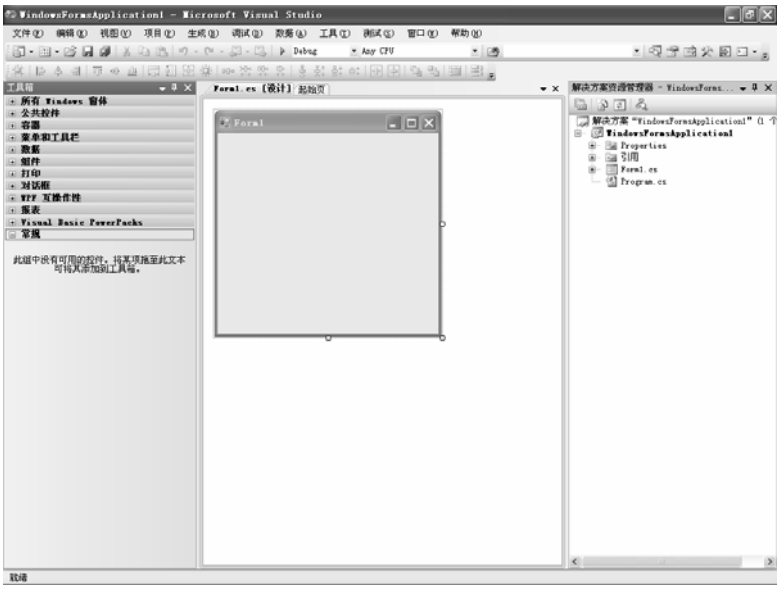


图 1.11　创建新项目后的 Visual Studio 2008 环境

带有 Form1 标识的矩形框被称为“窗体”，显示在工具箱中的图形标识被称为“控件”。窗体和控件是程序的图形用户界面（GUI），是用户和程序进行交互所用的图形组件。通过键盘输入信息和单击鼠标，用户可以向程序输入数据，程序在 GUI 中显示出指令和其他信息供用户阅读。

3．Visual Studio 2008 窗口

Visual Studio 2008 向用户提供了众多窗口，本书只简单介绍开发 C#应用程序所必需的窗口。一般初次启动时，很多窗口都是在界面上可见的，如若是呈隐藏状态，可从“视窗”菜单中选择所需窗口进行访问。

图 1.12　解决方案资源管理器窗口

（1）解决方案资源管理器：其窗口列出了所有在此解决方案中的文件。当 Visual Studio 2008 第一次启动时，“解决方案资源管理器”窗口是空的，没有显示任何文件，只有创建一个新项目或者现存的项目被打开时，才能显示出这个项目的内容。解决方案的启动项目是执行解决方案时所运行的项目，以粗体文本在窗口中显示。对于单个项目的解决方案，启动项目（Windows Forms Application1）是唯一的项目。C#文件是 Form1.cs，包含了程序的代码。解决方案资源管理器窗口如图 1.12 所示。

项目和解决方案左边的加号框（或减号框）用于展开（或折叠）以树状显示的选项，亦可通过双击文件夹的名称来展开（或折叠）。窗口中有一个工具栏，鼠标移动到图标上会出现文字提示。工具栏有一个图标用于重新加载解决方案中的文件（刷新），另一个图标用于显示解决方案中的所有文件（包括隐藏的文件）。根据所选文件的类型，工具栏中图标的数目会发生变化。

（2）工具箱：包含了可重用组件，用于自定义应用程序。使用可视化编程时，程序员能在窗体中“拖放”控件，而不用自己去写代码。工具箱如图 1.13 所示。

图 1.13　工具箱

工具箱包含多个相关组件的组，主要包括数据、组件和 Windows 窗体等选项卡，可以通

过单击组名展开。组的第一个选项不是控件，而是鼠标指针，单击可以取消当前选中的图标。注意：这里没有工具提示，因为工具箱图标已经标上了控件的名称。下面将对工具箱中的数据、组件和 Windows 窗体选项卡分别进行介绍。

① “工具箱”→“数据”选项卡：显示可以添加到 C#窗体和组件中的数据对象，当创建一个具有关联设计器的项目时将显示工具箱的“数据”选项卡。默认情况下，工具箱将出现在 Visual Studio 2008 集成开发环境中。如果需要显示工具箱，可从“视图”菜单中选择“工具箱”；若要直接转到某一组件的.NET Framework 参考主题，可以选中工具箱中的选项或设计器栏中的组件选项，然后按 F1 键。

② “工具箱”→“组件”选项卡：显示可以添加到 Visual C#设计器的组件。除了 Visual Studio 2008 随附的.NET Framework 组件（如 Message Queue 和事件日志组件）外，还可以在此选项卡中创建组件或添加第三方组件。若要显示此选项卡，可以从“视图”菜单中选择“工具箱”，在“工具箱”中选择“组件”选项卡。

③ “工具箱”→“Windows 窗体”选项卡：显示可用于 Windows 应用程序的 Windows 窗体控件和对话框的列表。“Windows 窗体”选项卡中的控件和组件按估计的使用频率排序。若要按字母顺序排序，可以用鼠标右键单击“工具箱”选项卡，选择按字母顺序排序。

使用某些控件功能时，可能需要以非常规的方式使用。例如为了在窗体中装配多个 RadioButton 和 CheckBox 控件，使其能够作为一个组工作，则必须先将一个 GroupBox 控件添加到窗体上，然后再将所需的控件添加到该 GroupBox 中。此外，当将某些不具有用户界面的控件（例如 Timer 或 ImageList 控件）添加到窗体上时，这些控件将出现在 Windows 窗体设计器底部的横栏中，而不是出现在窗体本身的图面上。工具箱的“组件”和“数据”选项卡还包含可用于 Windows 窗体设计的控件和组件。

（3）属性窗口：用于操作一个窗口或控件的属性。每个控件都有自己的一组属性，属性定义了控件的信息，诸如大小、颜色和位置等。属性窗口如图 1.14 所示。

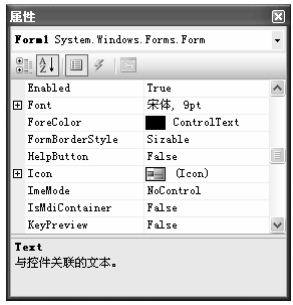


图 1.14 “属性”窗口

在属性窗口的下方有选中属性的描述；在属性窗口的顶部是一个下拉列表，被称为组件选择，此列表显示了当前正在修改的组件，可以使用列表来选择组件进行编辑。

（4）帮助菜单：包括许多选项，按字母序列索引显示目录项供用户浏览；搜索特性允许用户基于一些搜索关键词来查找特定的帮助文档，也可以使用筛选器缩小搜寻范围，只搜寻 C#相关的文件。动态帮助提供了一系列基于当前内容（即鼠标指针附近位置的项目）的提示。要打开动态帮助，可选择“帮助”菜单中的动态帮助命令，但对于一些用户，动态帮助会减缓 Visual Studio 2008 的运行速度。动态帮助如图 1.15 所示。

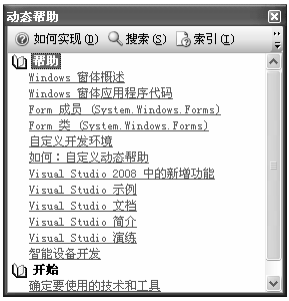


图 1.15　动态帮助

Visual Studio 2008 还提供了上下文敏感的帮助，除了立即显示出相关帮助而不显示列表外，上下文敏感帮助类似于动态帮助。要使用上下文敏感帮助，选择相关内容并按 F1 键即可。

1.3　数据类型

C#数据类型主要有值类型和引用类型两大类：值类型包括简单类型、结构类型和枚举类型；引用类型包括类类型、数组类型、接口类型和委托类型。引用类型所存储的实际数据是当前引用对象的地址。对于引用类型，有可能两个变量引用相同的对象，因而可能出现对一个变量的操作影响到其他变量的情况。这里仅介绍值类型中的简单类型和结构类型，引用类型中的类类型和数组类型，其他数据类型将在涉及的项目中讲解。

1.3.1　简单类型

简单类型包括整数类型、浮点类型、小数类型、字符类型和布尔类型等。受计算机存储单元数量的限制，任何一种数据类型都有一定的取值范围和精度。

1．整数类型

整数类型的数据值只能是整数。C#提供 8 种整数类型：Sbyte（8 位有符号整数）、Byte（8 位无符号整数）、Short（16 位有符号整数）、Ushort（16 位无符号整数）、Int（32 位有符号整数）、Uint（32 位无符号整数）、Long（64 位有符号整数）和 Ulong（64 位无符号整数）。其中 Int 和 Uint 两种是常用类型，Int 的取值范围为-2^{32}～$2^{32}-1$、Uint 的取值范围为 0～$2^{64}-1$。

2．浮点类型

数学意义上的小数在 C#中用浮点类型的数据表示。浮点类型的数据包含两种：单精度浮点型（float）和双精度浮点型（double），其区别在于取值范围和精度的不同。计算机对浮点数据的运算速度大大低于对整数的运算速度，数据的精度越高对计算机的资源要求就越高。因此在对精度要求不高的情况下，可以采用单精度类型，而在精度要求比较高的情况下使用双精度类型。float 类型占 4 个字节（32 位），取值范围在$\pm1.5\times10^{-45}$～3.4×10^{38}之间，精度为 7 位有效数字；double 占 8 个字节（64 位），取值范围在$\pm5.0\times10^{-324}$～1.710^{308}，精度为 15～16 位有效数字。

3．小数类型

小数类型数据是高精度的类型数据，占用 16 个字节（128 位），主要为了满足需要高精

度的财务和金融计算等领域。小数类型的数据取值范围在 1.0×10^{-28}～7.9×10^{28} 之间，精度为 29 位有效数字。小数类型数据的范围远远小于浮点型，不过其精度比浮点型高得多，所以相同的数字对于两种类型来说可能表达的内容并不相同。注意：小数类型数据的后面必须跟 m 或者 M，表示是小数类型，例如 3.14m、0.28m 等，否则就会被解释成浮点类型数据，导致数据类型不匹配。

4．字符类型

C#提供的字符类型数据按照国际上公认的标准，采用 Unicode 字符集。一个 Unicode 字符占 2 个字节（16 位），可以用来表示世界上大部分语言种类，所有 Unicode 字符的集合构成字符类型。字符类型的类型标识符是 char，因此也可称为 char 类型。

凡是在单引号中的一个字符，就是一个字符常数，例如：‘A’、‘b’、‘*’、‘8’、‘0’ 等。在表示一个字符常数时，单引号内的有效字符数量有且仅有一个，并且不能是单引号或者反斜杠。为了表示单引号或反斜杠等特殊的字符常数，C#提供了转义符，在需要表示这些特殊常数的地方，可以使用这些转义符来替代字符，如表 1.1 所示。

表 1.1　C#常用的转义符

转义符	代表字符名称	转义符	代表字符名称
\'	单引号	\f	换页
\"	双引号	\n	换行
\\	反斜杠	\r	换行并移至最前
\0	空字符（Null）	\t	水平方向的 Tab
\a	发出一个警告	\v	垂直方向的 Tab
\b	倒退一个字符		

5．布尔类型

布尔类型数据用于表示逻辑真或逻辑假，其类型标识符是 bool。布尔类型常数只有两种值：ture（代表“真”）和 false（代表“假”）。布尔类型数据主要应用在流程控制中，多为关系或逻辑表达式的运算结果，程序员往往通过读取或设定布尔类型数据的方式控制程序执行方向。

1.3.2　结构类型

结构类型是把各种不同类型的数据组合在一起形成的组合类型，是用户可以自定义的数据类型。例如一个学生的个人记录可能包括：学号、姓名、性别、年龄、电话，这些信息的类型不同，可以使用结构类型存储。

1．结构的声明

结构类型需要先声明后使用，声明结构类型要使用 struct 关键字，语法格式如下：

```
struct    标识符
{
//结构成员定义
}
```

说明：

（1）struct 关键字表示声明的是一种结构类型。

（2）标识符必须是 C#合法的标识符，用来确定所定义的结构。

（3）由一对花括号括起来的部分称为结构体，定义了结构中所包含的各种成员。

【例 1-1】 定义一个学生信息的结构类型 Student，包括：姓名、年龄、性别和所在系等信息。程序代码如下：

```
struct Student
{
    char name[8]; int age; char sex[2]; char depart[20];
}
```

2．结构成员的访问

结构成员可分为两类：一类是实例成员，另一类是静态成员。若成员名前有 static 关键字，则该成员为静态成员，否则为实例成员。静态成员通过结构名来访问，而实例成员的访问是通过创建结构类型的变量来访问的。创建结构类型变量的一般形式如下：

结构名　标识符；

说明：结构名为已声明的结构类型名称，标识符必须是 C#合法的标识符，用来表示该结构类型的变量。

【例 1-2】 定义一个结构类型用于存放平面坐标的 X 轴和 Y 轴数据，并通过访问结构成员显示该坐标。程序代码如下：

```
public struct CoOrds
{
    public int x, y;
    public CoOrds(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
class TestCoOrdsNoNew
{
    static void Main()
    {
        CoOrds coords1;
        coords1.x = 10;
        coords1.y = 20;
        System.Console.Write("CoOrds 1: ");
        System.Console.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y);
    }
}
输出为：
CoOrds 1: x = 10, y = 20
```

1.3.3 类类型

类类型是 C#中功能最为强大的数据类型。像结构一样，类也定义数据的类型和行为。

1．类的声明

类类型的声明与结构类型的声明很类似，不同的是类的声明要使用 class 关键字，其格式如下：

```
[类修饰符] class 类名
    {
        类的成员;
    }
```

类的修饰符有多个，C#支持的类修饰符常用的有：new、public、protected 和 private，其含义分别如下：

（1）new：新建类，表示由基类继承而来、与基类同名的成员；

（2）public：公有类，表示外界可以不受限制的访问；

（3）protected：保护类，表示可以访问同类或从该类派生的类；

（4）private：私有类，表示只有同类才能访问。

以上类修饰符可以两个或多个组合起来使用，但需要注意下面几点：

（1）在一个类声明中，同一类的修饰符不能多次出现，否则会出错；

（2）new 类修饰符仅允许在嵌套类中表示类声明时使用，表明类中隐藏了由基类继承而来的、与基类同名的成员；

（3）在使用 public、protected 和 private 这些类修饰符时，要注意这些类修饰符不仅表示所定义类的访问特性，而且还表明类中成员声明时的访问特性，并且它们的可用性也会对派生类造成影响；

（4）省略类修饰符，则默认为私有修饰符 private。

2．类成员的访问

在 C#中，按照类的成员是否为函数将其分为两大类，一种以函数形式出现，称为成员函数，另一种不以函数形式体现，称为成员变量。C#中成员主要有以下几种：

（1）公有成员：这种成员允许类的内部或外界直接访问，修饰符是 public，这是限制最少的一种访问方式；

（2）私有成员：外界不能直接访问该成员变量或成员函数，对该成员变量或成员函数的访问只能由该类中的其他函数来完成；

（3）保护成员：对于外界该成员是隐藏的；

类的每个成员都需要设定访问修饰符，不同的修饰符会造成对成员访问能力的不同。类的常用成员有以下类型：

（1）常量：代表与类相关的常数值。

（2）变量：类中的变量。

（3）方法：完成类中各种计算或功能操作。

（4）属性：定义类的值，并对其提供读、写操作。

（5）事件：由类产生的通知，用于说明发生了什么事情。

【例 1-3】 类成员的声明与访问。

```
using System;
class Kid
{
    private int age;
    private string name;
    public Kid()
    {
        name = "N/A";
    }
    public Kid(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
    public void PrintKid()
    {
        Console.WriteLine("{0}, {1} years old.", name, age);
    }
}
class MainClass
{
    static void Main()
    {
        Kid kid1 = new Kid("Craig", 11);
        Kid kid2 = new Kid("Sally", 10);
        Kid kid3 = new Kid();
        Console.Write("Kid #1: ");
        kid1.PrintKid();
        Console.Write("Kid #2: ");
        kid2.PrintKid();
        Console.Write("Kid #3: ");
        kid3.PrintKid();
    }
}
```

程序结果：

```
Kid #1：Craig, 11 years old.
Kid #2：Sally, 10 years old.
Kid #3：N/A, 0 years old.
```

在此例中声明了两个类，第一个是 Kid 类，包含两个私有字段（name 和 age）和两个公共方法；第二个类 MainClass 用来包含 Main。

1.3.4 数组类型

数组类型是把一些类型相同的数据组合在一起形成的组合类型。在 C#中，数组是一维的（只有一个下标）或者是多维的（有多个下标），其中一维数组使用最普遍。对于每一维，数组中数组元素的个数叫这个维的数组长度。无论是一维数组还是多维数组，每个维的下标都是从 0 开始的，下标结束于这个维的数组长度减 1。这里以一维数组为例介绍数组类型。

1．数组的定义

数组在使用前必须先定义。定义一维数组的格式如下：

数组类型[ ] 数组名:

说明：数组类型可以是各种数据类型（如 double 型或类类型），表示数组元素的类型；数组名可以是 C#合法的标识符；在数组名与数据之间是一个空的方括号。

例如：“char [ ] CharA;”定义了一个字符型的一维数组、“int[ ] IntArr;”定义了一个整型一维数组、“string[ ] StringA1;”定义了一个字符串型一维数组。

2．数组的初始化

在定义数组后，必须对其进行初始化才能使用。初始化数组有两种方法：动态初始化和静态初始化。

（1）动态初始化。动态初始化需要借助 new 运算符，为数组元素分配内存空间，并为数组元素赋初值。动态初始化数组的格式如下：

数据类型[ ] 数组名=new 数据类型[数组长度];

例如：“int[ ] IntArr = new int[10];”定义了一个整型数组，包含了从 Arr[0]到 Arr[9]这 5 个元素。new 运算符创建数组，并用默认值对数组元素进行初始化。在本例中，所有数组元素的值都被初始化为 0。用户也可以为其赋予其他数值，例如：“int[ ] IntArr=new int [10]{1，2，3，5，6，8，9，12，21，38};”，此时数组元素的初始化值就是大括号中列出的元素值。

定义其他类型数组的方法也是一样的。例如下面的语句用于定义一个储存 5 个字符串元素的数组，并对其进行初始化：“string[ ] StringArr=new string[5];”

（2）静态初始化。如果数组中包含的元素不多，而且初始元素可以穷举时，可以采用静态初始化的方法。静态初始化数组时，必须与数组定义结合在一起，否则程序就会出错。静态初始化数组的格式如下：

数据类型[ ] 数组名={元素 1，元素 2，…};

用这种方法对数组进行初始化时，无须说明数组元素的个数，只需按顺序列出数组中的全部元素即可，系统会自动计算并分配数组所需的内存空间。例如：“double[ ] A={3.6，9.18};”“string[ ] student={“wang”，“zhang”，“li”，“cheng”};”

在 C#中，数组初始化时需要注意以下几个方面：

（1）动态初始化数组可以把定义与初始化分开在不同的语句中进行，例如：

```
int[ ]  IntA;                          //定义数组
IntA=new int[5];                       //动态初始化。初始化元素的值均为 0
也可以写成：
IntA=new int[5]{1,2,3,4,5};            //动态初始化，元素值为花括号中列出的值
```

方括号中表示数组元素个数的“5”可以省略，因为后面大括号中已列出了数组中的全部元素。

（2）静态初始化数组必须与数组定义结合在一条语句中，否则程序就会出错。例如：

```
int[ ]  IntB= {1,3,6,7,2,8};                    //定义与静态初始化在同一条语句中
```

（3）在数组初始化语句中，如果花括号中已明确列出了数组中的元素，即确定了元素个数，则表示数组元素个数的值（即方括号中的数值）必须是常数或常量，并且该数值必须与数组元素个数一致。

3．数组元素的访问

定义一个数组，并对其进行初始化后，就可以访问该数组中的元素了。在 C#中是通过数组名和下标值来访问数组元素的。数组下标就是元素索引值，表示被访问数组元素在内存中的相对位置。在 C#语言中，数组下标从 0 开始，到数组长度减去 1 结束。在访问数组元素时，其下标可以是一个整型常量或是整型表达式。

【例 1-4】 定义、初始化和访问数组。

```
using System;
class DeclareArraysSample
{
    public static void Main()
    {
        // 一维数组
        int[] numbers = new int[5];
        // 多维数组
        string[,] names = new string[5,4];
        // 数组的数组（交错数组）
        byte[][] scores = new byte[5][];
        // 创建交错数组
        for (int i = 0; i < scores.Length; i++)
        {
            scores[i] = new byte[i+3];
        }
        // 打印每行的长度
        for (int i = 0; i < scores.Length; i++)
        {
            Console.WriteLine("Length of row {0} is {1}", i, scores[i].Length);
        }
    }
}
```

程序的结果：

```
Length of row   0   is   4
Length of row   1   is   5
Length of row   2   is   6
```

```
Length of row  3  is  7
Length of row  4  is  8
```

1.4 表达式

表达式是由变量、常数和运算符组成的算式，是各种程序设计语言中最基本的数据运算或数据处理过程。单个常数或变量是表达式的特殊情况。

1.4.1 标识符

标识符是一串字符，在程序中用来代表一个名字，定义一个变量、一个类或生成一个对象时，其名字就是标识符。C#的标识符必须符合以下规则：

（1）标识符以字母或下画线开头，其后可以跟任意字母、数字或下画线。例如 This_01、exam、_a1b2c 都是合法的标识符，7yuan、my3%、$4 都是非法字符。

（2）标识符严格区分字母的大小写。例如：abc 和 Abc 是两个不同的标识符。

（3）关键字不可以作为标识符。关键字是计算机程序设计语言中具有固定含义的特殊标识符。例如 int、double、char 等。

（4）在关键字前加入字符@，可以使关键字变成合法字符。例如@int、@char。

1.4.2 常量

常量是在程序运行过程中其值不变的量。要通过关键字 const 来声明常量，格式如下：

const　类型标识符　常量名 = 表达式

说明：类型标识符表示所定义常量的数据类型；常量名必须是合法的标识符，程序通过常量名来访问常量；表达式的计算结果就是定义常量的值。例如：“const double price=12.34;”该语句定义了一个 double 型的常量 price，其数值为 12.34。

需要注意的是常量只能被赋予初始值，赋值后，该常量的值在程序运行过程中不能改变，也不能对该常量赋值。

1.4.3 变量

变量是程序运行过程中用于存放数据的存储单元，其值在程序的运行过程中可以改变。C#规定，任何变量必须先定义后使用。

1．变量的定义

一次可以定义一个或多个同类型的变量，其格式如下：

类型标识符　变量名 1，变量名 2，…

说明：变量名只要是合法的标识符即可，但为保证程序的可读性，变量名最好使用具有实际意义的英文单词进行组合，单词的第一个字母通常大写。例如 Add1、Sum2、User 等。

2．变量的赋值

变量本身只是一个能保存某种类型具体数据的内存单元（这里所说的“内存单元”不一定以字节为单位）。对于程序而言，可以使用变量名来访问这个具体的内存单元。变量的赋值就是将数据保存到变量内存单元的过程。在 C#中，给一个变量值的格式如下：

变量名=表达式

说明：这里的表达式同数学中的表达式类似。如 6+8、8+a−b 都是表达式。在各种计算机语言中，单个常数或者变量，也可以构成表达式。由单个常数或者变量构成的表达式的值，就是这个常数或者变量本身。

在程序中，可以给一个变量多次赋值，变量的当前值等于最近一次给变量所赋的值。在对变量进行赋值时，表达式值的类型必须与变量的类型相同。

1.4.4 运算符

运算符是表示各种不同运算的符号。C#提供了丰富的运算符，根据运算类型可分为算术、赋值、位、关系、逻辑和条件运算符，如表 1.2 所示。

表 1.2 C#运算符

类别	符号	意义	说明	类别	符号	意义	说明
算术运算符	+	加法/取正	++和--的作用是使变量的值自动加 1 或减 1，例如 x++和 x=x+1 是一样的	关系运算符	>	大于	用于在程序中比较两个值的关系，运算结果是布尔型
	−	减法/取负			<	小于	
	*	乘法			>=	大于等于	
	/	除法			<=	小于等于	
	%	取余数			==	等于	
	++	自增			!=	不等于	
	−−	自减		赋值运算符	=	赋值	赋值运算符的结合性是从右向左，例如 x=y=z 相当于 x=(y=z)
逻辑运算符	!	逻辑非	用于比较两个值的逻辑关系，运算结果是布尔型		+=	相当于 x=x+1	
	&&	逻辑与			−=	相当于 x=x−1	
	‖	逻辑或			*=	相当于 x=x*1	
位运算符	~	按位取反	位运算符是对运算对象的二进制数位进行运算，依次选取运算对象的每个数位进行相应运算		/=	相当于 x=x/1	
	&	按位与			%=	相当于 x=x%1	
	|	按位或			&=	相当于 x=x&1	
	<<	按位左移			|=	相当于 x=x|1	
	>>	按位右移			^=	相当于 x=x^1	
	^	按位异或			>>=	相当于 x=x>>1	
条件运算符	? :	式 1：式 2：式 3	若式 1 的值为真，整个表达式等于式 2 的值；若为假，则等于式 3 的值		<<=	相当于 x=x<<1	

在对包含多种运算符的表达式求值时，如果有括号，先计算括号里面的表达式，然后依据运算符的优先级和结合性确定运算顺序。C#运算符的优先级（由高到低）如表 1.3 所示。

表 1.3 C#运算符的优先级

优先级	类别	运算符	优先级	类别	运算符
1	一元	+ − ! ~ ++ −−	8	按位异或	^
2	乘除取余	* / %	9	按位或	|

续表

优先级	类别	运算符	优先级	类别	运算符
3	加减	+ -	10	逻辑与	&&
4	移位	<< >>	11	逻辑或	||
5	关系	< > <= >=	12	条件	? :
6	等式	== !=	13	赋值	= *= /= += -= <<= >>= &= ^= |=
7	按位与	&			

当运算符两边的优先级相同时，由运算符的结合性决定运算顺序。赋值运算符和条件运算符是右结合的，即运算按照从右到左的顺序执行；其他二元运算符都是左结合，即运算按照从左到右的顺序执行。

1.5 C#程序的运行与调试

1.5.1 第一个C#程序

使用 Visual Studio 2008 提供的项目模板创建一个控制台应用程序，这个程序在 Windows 窗口显示“我的第一个C#程序”。步骤如下：

（1）单击“开始”→“所有程序”→“Microsoft Visual Studio 2008”→“Microsoft Visual Studio 2008”选项，启动 Visual Studio 2008；

（2）单击“文件”→“新建”→“项目”选项，打开“新建项目”对话框；

（3）在“模板”中选择“控制台应用程序”选项，在对话框下面的“名称”文本框中输入“Welcome”；

（4）单击【确定】按钮，关闭“新建项目”对话框，Visual Studio 2008 会自动为用户生成如下代码：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Welcome
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

（5）删除以上代码，输入如下代码：

```
/*导入.NET 系统类库提供的命名空间 System*/
using System;
class Welcome//定义类
```

```
{
static void Main()//程序的入口
{
    Console.WriteLine("我的第一个C#程序");//输出“我的第一个C#程序”
}
}
```

这样，第一个 C#程序就创建好了；

（6）在 Visual Studio 2008 中，用户可以采用两种方式运行程序：一种是调试运行；一种是不进行调试而直接运行。要调试运行程序，可以单击调试按钮 ▶、使用“调试”→“开始”命令或者直接按下 F5 键；要直接运行程序，则使用“调试”→“开始执行”选项或者同时按下 Ctrl 与 F5 两键。上述例子的运行结果如图 1.16 所示。

图 1.16　我的第一个 C#程序

1.5.2　程序的结构

一个简单的 C#程序的基本结构为：命名空间、类、Main()、注释方法等。下边就结合上例中的程序来分析 C#应用程序的结构。

1．命名空间

在上节的程序中，第一条语句“using System;”语句表示导入 System 命名空间。“Console.WriteLine("我的第一个 C#程序")”语句中的“Console”是 System 命名空间所包含的系统类库中定义好的一个类，代表系统控制台，即字符界面的输入和输出。

C#程序用命名空间来组织代码的，要访问某个命名空间中的类或对象，必须用如下语法：

命名空间.类名

由于 Console 类位于 System 命名空间中，所以用户访问 Console 类时，完整的写法应该是：

System. Console

但是在程序的第一行使用了：

using System;

这条语句用 using 语句导入 System 命名空间，这样在本程序中可以直接使用 System 命名空间中所有的类和对象，所以要访问 Console 类，就不用写“System.Console”，而直接写“Console”即可。

2．类

C#要求其程序中的每一个元素都要属于一个类。上节程序的第二行“class Welcome”声明了一个类，类的名字叫 Welcome，这个程序的功能就是依靠它来完成的。C#程序由花括号

“{”和“}”构成，程序中每一对花括号“{}”构成一个块。花括号总是成对出现，可以嵌套，即块内可以出现子块，嵌套深度不受限制，可以嵌套任意层，但一定要保证“{”和“}”成对出现，否则，程序就是错误的。

3．Main（）方法

C#程序入口从下面的代码开始：

```
static void Main()
```

这行代码所定义的其实是类 Welcome 的一个静态方法。C#规定，名字为 Main()的静态方法就是程序的入口。当程序执行时，就直接调用这个方法。这个方法包括一对花括号“{”和“}”，在两个括号间的语句就是该方法所包含的可执行语句，也就是该方法所要执行的功能，本例中该方法要执行的功能就是输出“我的第一个 C#程序”字符串。该方法的执行从左花括号“{”开始，到右大括号“}”结束。

4．注释

程序编写过程中常常要对程序中比较重要或需要注意的地方做些说明，但这些说明又不参与程序的执行。通常是采用注释的方法将这些说明写在程序中。在 C#中，提供两种注释方法：

（1）每一行中“//”后面的内容作为注释内容，该方式只对本行生效；

（2）需要多行注释时，在第一行之前使用“/*”，在末尾一行使用“*/”，也就是说被“/*”与“*/”之间包围的内容都作为注释内容。

1.5.3　程序的调试

在程序设计过程中，或多或少都会出现一些错误，一般可以归纳为语法错误和算法错误两种：语法错误通常是在程序输入时产生的，如函数名拼写错误、括号不匹配等错误。由于这些错误的存在，文件程序不能完成全部运行过程，会在发生错误处停止运行，并显示错误提示信息，因此这些错误在运行的过程中就能够发现，可以直接调试修改；算法错误是由于解题思路不正确或对问题的理解不准确而引起的，通常在运行时不会有错误提示信息，只有发现计算结果有较大的偏差、不符合设计要求、产生了意想不到的结果等情况时，才能根据结果的差异进行分析和判断，这可能是一个比较复杂的过程，通常都是采用设置断点来调试程序的。设置和取消断点的方法如下：

（1）鼠标左键单击某代码行左边的灰色区域，设置断点，再次单击则取消断点；

（2）右键单击某代码行，在弹出的菜单中，选择“断点”→“插入断点”或者“删除断点”；

（3）鼠标指向某代码行，按下 F9 键进行设置或取消断点。

【例 1-5】　编写一个简单的 Windows 应用程序，实现简单的交互。

（1）新建 VC#项目，选择模板中的 Windows 应用程序；

（2）项目名称为“HeyGuy”。从“工具箱”中为窗体安放 Button 控件和 textbox 控件，并设置控件属性；

（3）左键双击“解决方案资源管理器”中的“Program.cs”，修改代码为：

```
static void Main()
{
MessageBox.Show("请输入你的姓名");
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new Form1());
}
```

（4）双击【确定】按钮，为其添加事件处理程序，添加的程序代码如下：

```
private void button1_Click(object sender, System.EventArgs e)
{
    string strName;
    strName=textBox1.Text;
    MessageBox.Show("Hi "+ strName+"!");
}
```

如果有错误，则会弹出一个如图 1.17 所示的提示框。

图 1.17　提示框

无论选择“是”还是“否”，都会在窗口界面的下部出现一个任务列表，显示所有的输出错误，通过双击任务可以对其进行修改，选择后这些错误会自动隐藏在下部，通过单击表头可以打开。

利用断点调试程序时，运行一次软件后，会自动停止在断点处，接着参用上述方法继续执行，反复多次直到修改完所有错误为止。

在【例 1-5】中，首先找到想设置断点的代码行“strName=textBox1.Text;”设置该代码行为断点，如图 1.18 所示。

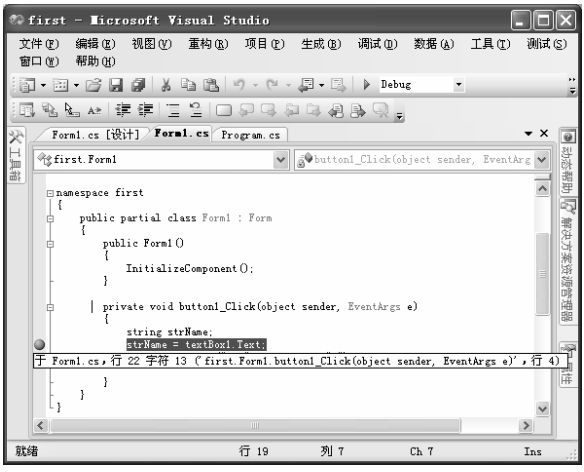


图 1.18　设置断点

设置有断点的行代码显示为红色，当鼠标移动到红色圆点位置时，就会提示该断点代码

行的位置信息。运行程序时，运行到设置断点的代码行，程序中断，代码行显示为黄色，红色圆点上会有一个箭头，如图 1.19 所示。

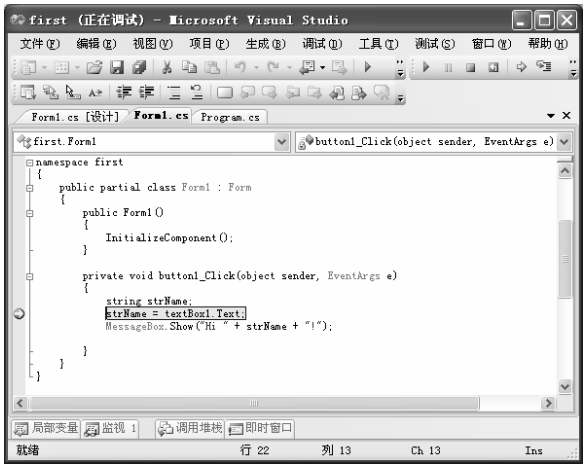


图 1.19　程序运行到断点

继续使用以上方式运行程序，完成程序的调试。在【例 1-5】中，如果代码误输入为：

```
private void button1_Click(object sender, System.EventArgs e)
{
    string strName;
    strName=textBox.Text;
    MessageBox.Show("Hi "+ strName+"!");
}
```

运行程序，则弹出如图 1.20 所示错误提示。

图 1.20　错误提示

错误显示已成功生成过，选择【是】按钮，直接调用上次成功的生成；选择【否】按钮，则跳出运行，错误列表中显示的错误信息如图 1.21 所示。

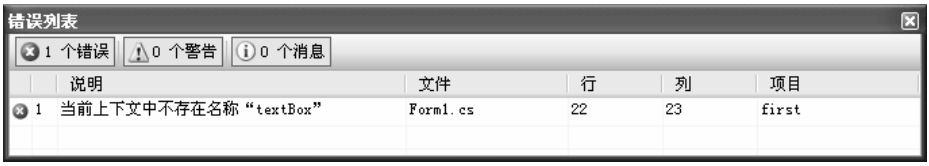


图 1.21　错误信息

1.6 实训：调试C#程序

【例 1-6】 编写一个简单的 Windows 应用程序，调试并运行程序。

（1）新建 VC#项目，选择模板中的 Windows 应用程序；

（2）项目名称为“Test1”。在“工具箱”的“组件”中选择“Timer”控件；

（3）双击 Form1 窗体，为其添加事件处理程序，程序如下：

```
private void Form1_Load(object sender, EventArgs e)
{
      this.timer1.Enabled = true;
this.Opacity = 0;
}
```

（4）双击 Timer1 控件，为其添加事件处理程序，程序如下：

```
private void timer1_Tick(object sender, EventArgs e)
{
      if (this.Opacity <1)
    {
            this.Opacity = this.Opacity +0.05;
     }
     else
     {
            this.timer1.Enabled = false ;
     }
 }
```

（5）调试并运行程序；

（6）运行结果是一个渐显的窗体。其中运行过程中的一个画面如图 1.22 所示。

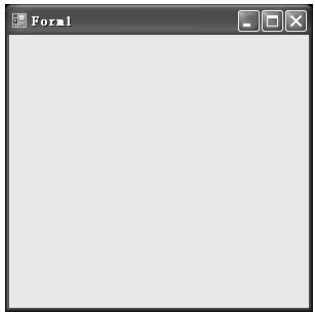


图 1.22 渐显的窗体

【例 1-7】 编写一个 Windows 应用程序，调试并运行程序，找出错误。

（1）新建 VC#项目，选择模板中的 Windows 应用程序；

（2）项目名称为“Test2”，添加三个 TextBox、两个 Label 和一个 Button 控件，如图 1.23 所示；

图 1.23　桌面布局

（3）双击 Button1 控件，为其添加事件处理程序，程序如下：

```
private void button1_Click(object sender, EventArgs e)
    {
        Double a;
       a = Convert.ToDouble(textBox1.Text) / Convert.ToDouble(textBox2.Text);
       textBox.Text = Convert.ToString(a);
    }
```

（4）调试并运行程序。发现错误如图 1.24 所示；

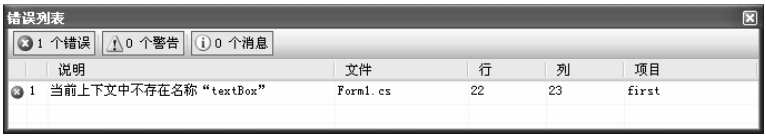


图 1.24　显示的出错信息

（5）修改错误并运行。

习　　题

1．填空

（1）C#是一种由_____和_____衍生出来的面向对象的编程语言。

（2）C#的目的就是综合 Visual BASIC 的_____和 C++的_____。

（3）通常采用设置_____来调试程序的。

2．选择

（1）类库是一个由.NET Framework SDK 中包含的（　　）组成的库。

A．类　　B．值类型　　C．接口　　D．命名空间

（2）公用语言运行库（CLR）是.NET Framework 的基础，提供诸如（　　）等核心服务。

A．内存管理　　B．开发流程　　C．线程管理　　D．远程处理

（3）解决方案资源管理器的功能是（　　）。

A．编写源代码

B．显示项目中的所有文件和项目设置

C．用于显示选定目标对象的属性

D．用于显示程序在编译过程中产生的错误

（4）选出不属于值类型的（　　）。

A．char、int 和 float　　B．枚举类型

C．引用类型　　D．结构类型

（5）无符号整数类型有（　　）。

A．byte　　B．ushort　　C．uint　　D．bit

3．注释方法有哪些？需要注意什么？

4．简述设置和取消断点的方法。

第 2 章　工资计算器

一个软件从构思到使用，首先是使用方根据自己的需求下达任务书；接着软件开发人员根据任务书，通过简单分析，写出相应的计划；待得到使用方的批准后，进行细致的项目分析，制定开发方案，进而按照方案进行开发。本章以企业工资计算器为例，讲解 Label、TextBox、Button 控件，Container、MessageBox、Convert 类，if-else、using 语句的应用，介绍软件窗体的设计方法。

2.1　项目说明

2.1.1　任务书

（1）项目名称：工资计算器。
（2）工作期限：1 个工作日。
（3）工作任务：编写工资计算系统，公式如下：
月工资＝日工资×天数＋加班费＋满勤奖＋补助－保险－个人所得税
其中：日工资、天数、加班费、满勤奖、补助、保险为输入项；
满勤奖和补助只有满勤人员才能获得；
加班费根据个人加班数量及日工资的多少而定，需人工输入；
保险为个人缴纳部分，每月取整浮动；
个人所得税和月工资为输出项，都是通过单击相应按钮进行计算；
个人所得税按国家规定的税率计算，税率表如表 2.1 所示。

表 2.1　税率表

级数	含税级距	税率（%）	速算扣除数	说明
1	不超过 500 元的	5	0	本表含税级距指以每月收入额减除费用两千元后的余额
2	超过 500 元至 2000 元的部分	10	25	
3	超过 2000 元至 5000 元的部分	15	125	
4	超过 5000 元至 20000 元的部分	20	375	

（4）项目需达到的技术性能：企业财务部门可以使用该计算器方便地计算员工工资，能够为企业节约人力、物力。

2.1.2　计划书

1．工作内容

（1）分析项目要完成的功能，确定所使用的软件开发工具和开发系统环境，安装好所需

的工具软件和系统环境，准备好所需资料；

（2）项目需求分析，确立开发方案，进行软件的概念分析、功能结构分析、逻辑设计和界面的初步设计等；

（3）软件的物理设计，模块功能设计，代码的初步实施；

（4）软件的代码实施，代码的功能测试，各模块的组装调试，软件的整体调试，软件注释文档的完善；

（5）创建软件系统的安装文件，发布软件的测试版本，并与用户共同完成软件的整体测试与功能完善。

2．项目分析

（1）本软件用 C#进行开发。C#是 Visual Studio 2008 的一部分，故需先行安装 Visual Studio 2008。

（2）本软件是公司内部使用，单机作业，所以界面简洁、使用方便才是最主要的。

（3）软件需要五个输入框，用来存储日工资、天数、满勤、补助、保险。输入框选用控件 TextBox 实现，并且每个输入框前用控件 Label 加以标识。

（4）软件有两个输出框，用于显示个人所得税和工资的总额。为便于个人所得税的计算，添加一个税前工资总额的输出框。

（5）添加两个按钮，分别用于控制工资计算和清除输入错误时的数据。

（6）需要计算的个人所得税分为四部分（参见表 2.1），每部分有不同的计算方法，使用 if-else 分支语句进行计算。

（7）值得注意的是：textBox1.Text 返回值的类型是 string 类型，是不可以进行数学计算的，需要用 Convert 类转换成 double 类型的操作数进行计算。

2.2 项目准备

2.2.1 控件

本项目涉及 Label、TextBox、Button 等控件，这些控件都可以在“工具箱”的“公共控件”里找到。标签控件 Label 用于显示不可更改的短字符串，文本框控件 TextBox 与标签控件 Label 类似，可以显示可更改的各类字符串，而按钮控件 Button 主要用于单击事件的执行操作。

1．Label控件

Label 控件用于显示用户不能编辑的文本或图像，常用于标识窗体上的对象。该控件显示的字符串，在程序运行时用户不能编辑。Label 控件中显示的长字符串在小屏幕的设备上可能显示不完整，因此最好使用 Label 控件显示短字符串。使用 Label 控件应注意以下问题：

（1）Label 控件必须放在 Form 或 Panel 控件中，或者放在控件的模板中。

（2）使用“属性”窗口中的 Text 属性可设置 Label 控件显示的字符串。如果用这种方式指定 Text 属性，则内部文本始终优先。但是如果以编程方式设置 Text 属性，则会自动移除内部文本，新设置的属性优先。

（3）通过设置 BackColor（文本或图形的背景颜色）、ForeColor（前景色）、Font（文本的字体）、BorderStyle（外框）等属性来更改 Label 控件的外观。也可以通过编写程序在运行时改变这些属性。

2．TextBox控件

TextBox 控件通常用于可编辑文本，在程序运行时用户可以编辑，不过也可使其成为只读控件。文本框可以显示多个行，对文本换行使其符合控件的大小以及添加基本的格式设置。TextBox 控件为在该控件中显示的或输入的文本提供一种格式化样式。

用户在文本框中输入的文本存储在控件的 Text 属性中，该属性是从 TextControl 基类继承的。TextBox 控件可以屏蔽密码的输入（如果用户的设备支持此行为）。使用 TextBox 控件应注意以下问题：

（1）TextBox 控件必须放在 Form 或 Panel 控件内，或控件模板内。

（2）应用程序使用 Text 属性为 TextBox 控件设置初始字符串。

（3）Size 属性指定期望的输入字符串宽度（即所含字符数）。TextBox 控件根据 Size 的值来缩放其输入框的长度，值“0”表示 TextBox 控件使用其初始大小设置。输入字符串（或初始字符串）包含的字符数可以大于 Size 指定的字符数。如果用户输入的字符串对输入框来说太长，则以前输入的字符会向左滚动。

（4）使用 TextAlign 属性可以设置控件的对齐方式。对齐方式可以设置为 Left、Center 或 Right。如果没有指定对齐方式，则 TextBox 控件使用默认的对齐方式。

（5）使用 ForeColor 改变前景色、使用 BackColor 改变背景色、使用 Font 属性改变文本的外观。

（6）若要使用 TextBox 控件检索用户密码，可在“属性”窗口中将 PasswordChar 属性设置为想显示的符号，用以代替输入密码，例如“******”等。

3．Button控件

Button 控件允许用户通过单击来执行某个操作，即可以显示文本，又可以显示图像。使用 Button 控件应注意以下问题：

（1）每当用户单击按钮时，即调用 Click 事件处理程序，可将代码放入 Click 事件处理程序来执行所选择的操作。

（2）在外观上，单击按钮时显示按下状态，随后弹起。

（3）按钮上显示的文本包含在 Text 属性中。如果文本超出按钮宽度，则换到下一行。但如果控件无法容纳文本的总体高度，将剪裁文本。

（4）Text 属性可以包含快捷键，允许用户通过同时按下 Alt 键和访问键来单击控件。

（5）文本的外观受 Font 属性和 TextAlign 属性控制。

（6）可以使用 Image 和 ImageList 属性显示图像。

2.2.2 类

本项目涉及 MessageBox 和 Convert 类，其中 MessageBox 类用于显示消息框，而 Convert 类将一个数据类型转换为另一个数据类型。

1．MessageBox类

MessageBox类用于显示可包含文本、按钮和符号（通知并指示用户）的消息框。调用静态方法 MessageBox.Show能够显示消息框，消息框中的标题、消息、按钮和图标由传递给该方法的参数确定。

【例 2-1】 下面的示例说明如何使用MessageBox提示用户填入“name”。该示例假定从具有 Button的现有窗体调用此方法。程序代码如下：

```
protected void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("You must enter a name.", "Name Entry Error",
    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
```

程序运行结果如图 2.1 所示。

图 2.1 运行结果

从图 2.1 中可以看出，在此例子中所调用的 MessageBox.Show方法中各参数的意义分别是：

“You must enter a name.”表示要提示的内容；
“Name Entry Error”表示提示框的标题；
“MessageBoxButtons.OK”表示按钮的类型；
“MessageBoxIcon.Exclamation”表示提示时所显示的图标类型。

2．Convert类

Convert类是将一个基本数据类型转换为另一个基本数据类型。该类的返回值是与指定类型值等效的类型。受支持的基类型是 Boolean、Char、Sbyte、Byte、Int16、Int32、Int64、Uint16、Uint32、Uint64、Single、Double、Decimal、DateTime和 String。

从某类型到其自身的转换只返回该类型，不实际执行任何转换。另外，无法产生有意义结果的转换也不实际执行任何转换，并将引发 InvalidCastException（异常）。一般会引发异常转换如下：

（1）从 Char转换为 Boolean、Single、Double、Decimal或 DateTime，或从这些类型转换为 Char；

（2）从 DateTime转换为除 String之外的任何类型，或从任何类型（String除外）转换为 DateTime；

（3）如果数字类型转换导致精度丢失（即某些最低有效位丢失），不引发异常。但若结果超出了特定转换方法的返回值类型所能表示的范围，则将引发异常。例如，当将 Double 转换为 Single 时，可能会发生精度丢失，但并不引发异常。但是，如果 Double 的值太大，无法由

Single 表示，则将引发溢出异常。

【例 2-2】 下面的示例说明此类中的一些转换方法，包括 ToInt32、ToBoolean 和 ToString。

```
double dNumber = 23.15;
//返回值为 23
int iNumber = System.Convert.ToInt32(dNumber);
//返回值为 True
bool bNumber = System.Convert.ToBoolean(dNumber);
//返回值为“23.15”
string strNumber = System.Convert.ToString(dNumber);
//返回值为‘2’
char chrNumber = System.Convert.ToChar(strNumber[0]);
//返回值为‘2’
```

2.2.3 语句

本项目涉及 if-else 语句，该语句是条件分支语句。根据条件选择要执行的语句，主要有以下 3 种形式：

1．基本形式

最简单的选择结构语句，其一般形式如下：

```
if（表达式）
    语句 1;
else
    语句 2;
```

说明：表达式多为关系或逻辑表达式；如果表达式为真（非零），就执行语句 1，然后再执行语句 2 之后的语句；如果表达式为假（零），就直接执行语句 2 及其后的语句；如果默认 else 和语句 2，则成为单分支结构：若表达式为真，就执行语句 1 及其后的语句；如果表达式为假，则跳过语句 1 执行其后的语句。

2．嵌套形式

在 if-else 语句中可以嵌套使用多层 if-else 语句，其一般形式如下：

```
if （表达式 1）
    if （表达式 2）
        …
        语句 1;
    else
        语句 2;
else
     语句 3；
```

说明：如果表达式 1 为真（非零），则执行内层的 if-else 语句，然后再执行语句 3 之后的语句；如果表达式 1 为假（零），执行语句 3 及其后的语句。

嵌套的层数没有限制，但层数太多会降低程序的可读性。另外，如果在某个分支需要执

行多条语句，可以使用花括号。例如：

```
if （表达式 1)
{
    if （表达式 2）
        语句 1;
            语句 2;
            …
}
else
    语句 2;
```

【例 2-3】 从键盘输入一个字符，编写程序检查输入字符是否为字母字符；如果输入的字符是字母，则检查是大写还是小写；在任何一种情况下，都需要显示相应的消息。

```
using System;
public class IfTest
{
    public static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char) Console.Read();
        if (Char.IsLetter(c))
            if (Char.IsLower(c))
                Console.WriteLine("The character is lowercase.");
            else
                Console.WriteLine("The character is uppercase.");
        else
            Console.WriteLine("The character is not an alphabetic character.");
    }
}
```

输入：2

示例输出：

```
Enter a character: 2
The character is not an alphabetic character.
```

3．扩展形式

扩展形式类似于嵌套形式，其一般形式如下：

```
if （表达式 1）
    语句 1;
else if （表达式 2）
    语句 2;
…
else if （表达式 n）
    语句 n;
else
```

```
        语句 m;
```

说明：先判断表达式 1 的值，若为真，则执行语句 1，然后执行语句 m 之后的语句；若表达式 1 的值为假，进而判断表达式 2，若为真，则执行语句 2，然后执行语句 m 之后的语句；如果表达式 2 的值也为假，则继续判断表达式 n，如此下去，若所有表达式都为假，则执行 else 后的语句 m 及其后的语句。else 语句可以默认。

【例 2-4】 检查输入字符是否是小写字符、大写字符或数字，如果都不是，则显示不是字母字符。

```
using System;
public class IfTest
{
    public static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char) Console.Read();
        if (Char.IsUpper(c))
            Console.WriteLine("The character is uppercase.");
        else if (Char.IsLower(c))
            Console.WriteLine("The character is lowercase.");
        else if (Char.IsDigit(c))
            Console.WriteLine("The character is a number.");
        else
            Console.WriteLine("The character is not alphanumeric.");
    }
}
```

输入：E

示例输出：

```
Enter a character: E
The character is uppercase.
```

2.2.4 命名空间

简单来说命名空间就是类的逻辑分组。命名空间就像一个文件夹，其内的对象就像一个个文件。不同文件夹内的文件可以重名，在使用重名的文件时，只需要说明在哪个文件夹下的就可以。通常要在文件的顶部列出类的命名空间，前面加上 using 关键字加以标识。由于 Microsoft 提供的许多常用的类都包含在 System 命名空间中，因此 C#源代码通常都是以语句“using System;”开头的。

1．System

System 命名空间包含基本类和基类，这些类是定义常用的值和引用数据类型、事件、事件处理程序、接口、属性和异常处理的。其他类提供的服务是用来支持数据类型转换、方法参数操作、数学运算、远程或本地程序调用、应用程序环境管理、对托管与非托管应用程序的监控。

2．System.Drawing

System.Drawing 命名空间提供了对 GDI（Graphics Device Interface，图形设备接口）和基本图形功能的访问。

3．System.Collections.Generic

System.Collections.Generic 命名空间包含定义泛型集合的接口和类，泛型集合允许用户创建强类型集合，能提供比非泛型强类型集合更好的类型安全性和性能。

4．System.ComponentModel

System.ComponentModel 命名空间提供用于实现组件或控件运行时行为的类。此命名空间包括用于实现属性和类型转换器、绑定到数据源以及授权组件的基类和接口。该命名空间中的类分为以下类别：

① 核心组件类：Component、IComponent、Container和 IContainer。

② 组件授权类：License、LicenseManager、LicenseProvider和 LicenseProviderAttribute。

③ 属性类：Attribute。

④ 说明符和持久性类：TypeDescriptor、EventDescriptor 和 PropertyDescriptor。

⑤ 类型转换器类：TypeConverter。

5．System.Windows.Forms

System.Windows.Forms 命名空间包含用于创建基于 Windows 应用程序的类，以便充分利用 Windows 操作系统中提供的用户界面功能，所包含的类如下：

① Control 类。用于给窗体中显示的所有控件提供基本功能。

② Form 类。表示应用程序内的窗口。包括对话框、无模式窗口、多文档界面（MDI）客户端窗口及父窗口。

③ 菜单和工具栏。Windows 窗体包含一组丰富的类，通过这些类，用户可以创建自定义工具栏和菜单，使用 ToolStrip、MenuStrip、ContextMenuStrip 和 StatusStrip 创建工具栏、菜单栏、上下文菜单以及状态栏。

④ 布局。Windows 窗体中的若干重要的类有助于控制显示图面（如窗体或控件）中控件的布局。

⑤ 数据和数据绑定。Windows 窗体为与数据源（如数据库和 XML 文件）的绑定定义了丰富的架构。

⑥ 组件。System.Windows.Forms 命名空间提供除控件之外的一些类，这些类不是从 Control 类派生的，但仍然向基于 Windows 的应用程序提供可视化功能。

⑦ 通用对话框。Windows 提供了许多通用对话框，在执行诸如打开和保存文件、操作字体或文本颜色，或打印之类的任务时，这些通用对话框可使应用程序具有一致的用户界面。

6．System.Data

System.Data 命名空间提供对表示 ADO.NET 结构的类的访问。通过 ADO.NET 可以生成一些组件，用于有效管理多个数据源的数据。在断开连接的情形下（如 Internet），ADO.NET

提供在多层系统中请求、更新和协调数据的工具。ADO.NET 结构也可在客户端应用程序（如 ASP.NET 创建的 Windows 窗体或 HTML 页）中实现。

7．System.Linq

System.Linq 命名空间提供类和接口，支持使用语言集成查询（LINQ）。

8．System.Text

System.Text 命名空间包含表示 ASCII、Unicode、UTF-7 和 UTF-8 字符编码的类，用于将字符块转换为字节块和将字节块转换为字符块的抽象基类。

2.3 项目开发

一个项目的开发主要分为窗体设计和代码设计两大块。其中窗体设计主要是为了界面的显示，代码设计主要是为了功能的实现。

2.3.1 窗体设计

本软件是一个简单的工资计算器，致力于企业工资的计算，界面要简洁大方、方便实用。参考第一章讲述的方法新建项目“工资计算器”，储存在 D 盘根目录下。

1．窗体属性设置

进入“Form1.cs[设计]”窗体，右键选择“属性”，修改“外观”中的“Text”为“工资计算器”，如图 2.2 所示。

图 2.2　Form1.cs[设计]属性窗体

2．添加控件

（1）日工资输入框的制作。选择“工具箱”→“Windows 窗体”，左键双击“Label”或右键选中“Label”拖曳到 Form1 窗体中，修改“外观”中的“Text”为“日工资”，将其命名为“日工资”；左键双击“Text”或左键选中“Text”拖曳到 Form1 窗体中，修改“外观”中的“Text”为“”，将其设置为空白；左键双击“Label”或左键选中“Label”拖曳到 Form1

窗体中，修改“外观”中的“Text”为“元/天”，这是日工资的单位。

（2）同理做好天数、满勤、补助、总额、保险的输入框。

（3）个人所得税的输出框的制作。选择“工具箱”→“Windows 窗体”，左键双击“Label”或左键选中“Label”拖曳到 Form1 窗体中，修改“外观”中的“Text”为“个人所得税”，将其命名为“个人所得税”；左键双击“Text”或左键选中“Text”拖曳到 Form1 窗体中，修改“外观”中的“Text”为“”，将其设置为空白，修改“ReadOnly”的属性值为“True”；左键双击“Label”或左键选中“Label”拖曳到 Form1 窗体中，修改“外观”中的“Text”为“元”，这是个人所得税的单位。

（4）同理做好总额、总工资的输入框。完成后的工资计算器界面如图 2.3 所示。

图 2.3　工资计算器界面

2.3.2　代码设计

Visual Studio 2008 将代码分为两部分存储。自动生成的部分存储在文件“Program.cs”中，而设计的代码存储在文件“Form1.cs”中。

1．Program.cs

在本章中简单对“Program.cs”中的代码加以说明，以后的各章就不另行叙述了。本部分代码按功能主要分为三部分：

（1）访问命名空间代码。在 C#中，大多数应用程序从一个 using 指令节开始。该节列出应用程序将会频繁使用的命名空间，避免程序员在每次使用其中包含的方法时都要指定完全限定的名称。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
```

（2）使用命名空间来控制范围代码。namespace 关键字用于声明一个范围。在项目中创建范围有助于组织代码，并提供创建全局唯一类型的途径。在下面的代码中，通过命名空间工资计算器定义名为 Form1 的类。

```
namespace 工资计算器
{
    static class Program
    {
        /// <summary>
```

```
        /// 应用程序的主入口点
        /// </summary>
        [STAThread]
    }
}
```

（3）软件启动代码。在当前线程上开始运行标准应用程序消息循环。系统默认启动工资计算器自动播放 Form1 中的内容，启动代码如下：

```
static void Main()
{
  Application.EnableVisualStyles();
  Application.SetCompatibleTextRenderingDefault(false);
  Application.Run(new Form1());
}
```

2．Form1.cs

代码设计时，所编写的大多程序是储存在 Form1.cs 文件中。

（1）【取消】按钮代码。左键双击【取消】按钮，进入代码编写，可以看到代码编辑器中已经有如下代码：

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace 工资计算器
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
          private void button1_Click(object sender, EventArgs e)
        {
        }
    }
}
```

在“private void button1_Click(object sender, System.EventArgs e){}”的{}添加代码，清空所有输入项：

```
private void button1_Click(object sender, System.EventArgs e)
```

```
        {
            textBox1.Text ="";
            textBox2.Text ="";
            textBox3.Text ="";
            textBox4.Text ="";
            textBox5.Text ="";
            textBox6.Text ="";
            textBox7.Text ="";
            textBox8.Text ="";
        }
```

C#的代码编写是基于 C 语言的，简单易懂。如果输入的代码语法准确，下一步会有输入信息的提示，例如：输入“textBox1”后接着输入“.”，则 C#会自动出现一个下拉菜单，里面有“textBox1”所有的属性项便于选择。选择时也不必一个个查询，可以直接输入首字母进行选择。

（2）【计算】按钮代码。双击【计算】按钮输入如下代码：

```
        private void button2_Click(object sender, System.EventArgs e)
        {
        textBox5.Text = Convert.ToString(Convert.ToDouble(textBox1.Text) * Convert.ToDouble(textBox2.
Text) + Convert.ToDouble(textBox3.Text)  + Convert.ToDouble(textBox4.Text));
        if(Convert.ToDouble(textBox5.Text)>=6600)
            textBox7.Text =Convert.ToString((Convert.ToDouble(textBox5.Text) - 6600) * 0.2 + 375);
        else
            if(Convert.ToDouble(textBox5.Text)>=3600)
            textBox7.Text =Convert.ToString((Convert.ToDouble(textBox5.Text) - 3600) * 0.15 + 125);
            else
        if(Convert.ToDouble(textBox5.Text)>=2100)
        textBox7.Text =Convert.ToString((Convert.ToDouble(textBox5.Text) - 2100) * 0.1 + 25);
        else
        if(Convert.ToDouble(textBox5.Text)>=1600)
        textBox7.Text =Convert.ToString((Convert.ToDouble(textBox5.Text) - 1600) * 0.05);
        else
        textBox7.Text = "0";
        textBox8.Text = Convert.ToString(Convert.ToDouble(textBox5.Text) - Convert.ToDouble(textBox6.
Text) - Convert.ToDouble(textBox7.Text));
        }
```

其中：

```
        ① textBox5.Text = Convert.ToString(Convert.ToDouble(textBox1.Text) * Convert. ToDouble (textBox2.
Text) + Convert.ToDouble(textBox3.Text)  + Convert.ToDouble (textBox4. Text));
```

表示公式：总额＝日工资×天数＋满勤奖＋补助。

```
        ② textBox8.Text = Convert.ToString(Convert.ToDouble(textBox5.Text) - Convert. ToDouble(textBox6.
Text) - Convert.ToDouble(textBox7.Text));
```

表示公式：工资＝总额－保险－扣税。

③ 通过判断语句 if-else 进行税金计算：

工资超出 5000 元部分，即总额大于 7000 元部分，扣税＝（总额－7000）×税率＋375；

工资超出部分介于 2000 和 5000 元之间的，即总额介于 4000 和 7000 元之间的部分，扣税＝（总额－4000）×税率＋125；

工资超出部分介于 500 和 2000 元之间的，即总额介于 2500 和 4000 元之间的部分，扣税＝（总额－2500）×税率＋25；

工资超出部分少于 500 元的，即总额介于 2000 和 2500 元之间的部分，扣税＝（总额－2000）×税率。

（3）纠错代码设计。主体代码编写完成后，需要考虑纠错问题：

① 没有输入数据，单击“计算”按钮，则提示“请输入数据”，其代码为：

```
if(textBox1.Text =="||textBox2.Text = =""||textBox3.Text = =""||textBox1.Text = ="" ||textBox6.Text =="")
        MessageBox.Show ("请输入数据","提示");
    else{}
```

② 如果天数超过 31 天，则提示“你输入的天数不正确”，使用代码：

```
if(Convert.ToDouble(textBox2.Text)>31)
        MessageBox.Show ("你输入的天数不正确","错误");
    else{}
```

③ 总额、扣税、工资只是显示部分，不可以输入数据，分别设置其“属性”中“行为”下的“ReadOnly”项为“True”，亦可左键双击 Form1 窗体添加如下代码：

```
private void Form1_Load(object sender, System.EventArgs e)
    {
        textBox5.ReadOnly =true;
        textBox7.ReadOnly =true;
        textBox8.ReadOnly =true;
        }
```

2.4 实训：考试成绩统计软件的开发

1．项目要求

（1）可以手动输入语文、数学、英语、综合的分数。

（2）有一个计算总分按钮，总分可更改，亦可直接输入总分进行平均分的计算。

（3）有一个计算平均分按钮，平均分计算出来后不可更改。

（4）有一个清除数据按钮，可以清空所有数据。

2．设计提示

启动 Visual Studio 2008 后，首先要新建项目。进行系统设计时，需要注意以下问题：

（1）需要有四个文本框，分别用于输入语文、数学、英语、综合的分数；

（2）有一个【计算总分】按钮，一个文本框用于显示和输入总分；

（3）有一个【计算平均分】按钮，一个标签用于显示平均分，也可以使用文本框，设置属性为不可更改即可；

（4）有一个【清除数据】按钮。

3．项目评价

项目评价是在教师的主持下，通过项目负责人的讲解演示，评估项目的完成情况，评价内容如下：

（1）工具类软件界面是否简洁，操作是否简便；

（2）使用过程中有无错误提示；

（3）能否实现基本功能。

习　　题

1．填空

（1）按钮上显示的文本包含在_____属性中，如果文本超出按钮宽度，则_____。但是，如果控件无法容纳文本的总体高度，则_____。

（2）System.Windows.Forms 命名空间包含用于创建基于_____的应用程序的类，以便充分利用 Microsoft Windows 操作系统中提供的丰富的用户界面功能。

（3）System.Data 命名空间提供对表示_____结构的类的访问。

2．选择

（1）MessageBox 类显示可包含（　　）的消息框。

A．文本　　B．按钮　　C．文本框　　D．符号

（2）下列描述错误的是（　　）。

A．TextBox 控件使用时不用放在 Form 或 Panel 控件内。

B．使用 TextAlign 属性可以设置控件的对齐方式。

C．应用程序使用 Text 属性为 TextBox 控件设置初始字符串。

D．使用 ForeColor、Font 属性可以自定义文本的外观。

（3）下列描述中正确的是（　　）。

A．每当用户单击按钮时，即调用 Click 事件处理程序。

B．Button 控件允许用户通过双击来执行操作，即可以显示文本，又可以显示图像。

C．在外观上，单击按钮时显示按下状态，随后弹起。

D．按钮上显示的文本包含在 Caption 属性中。

3．说说 Label 控件与 TextBox 控件的异同。

4．代码“MessageBox.Show ("请输入数据","提示");”实现什么功能？

5．说明以下两段程序代码有什么区别？分别实现什么功能？

```
（1）if (x > 10)
    if (y > 20)
    Console.Write("Statement_1");
    else
Console.Write("Statement_2");
（2）if (x > 10)
```

```
{
    if (y > 20)
Console.Write("Statement_1");
}
else
    Console.Write("Statement_2");
```

6．写出下列程序代码的功能：

```
    public static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char) Console.Read();
        if (Char.IsUpper(c))
            Console.WriteLine("The character is uppercase.");
        else if (Char.IsLower(c))
            Console.WriteLine("The character is lowercase.");
        else if (Char.IsDigit(c))
            Console.WriteLine("The character is a number.");
        else
            Console.WriteLine("The character is not alphanumeric.");
}
```

第 3 章　串口调试器

本章主要讲解 for 语句的使用方法，如何利用 try-catch 语句处理异常，重点讲解使用第三方控件 MsComm 设计工业机串口通信的调试器。

3.1　项目说明

3.1.1　任务书

（1）项目名称：串口调试器。

（2）工作期限：2 个工作日。

（3）工作任务：计算机上的串行接口有 9 芯和 25 芯两种，其中 9 芯较为常用。9 芯串口有上下两排针脚，上排 5 个针脚对应 1～5 号线，下排 4 个针脚对应 6～9 号线。串行通信是指数据一位一位地依次传输，每一位数据占据一个固定的时间长度，特别适用于计算机与计算机、计算机与外设之间的远距离通信。在硬件连接上只要连接三条导线即可，这三条线分别是发送线、接收线和地线，一台计算机的发送端要连到另外一台计算机的接收端，所以收、发数据线（2、3 号线）要换接；而地线（9 芯串口为 5 号线，25 芯串口为 7 号线）是公用的，直接接起来就可以。线序如表 3.1 所示。

表 3.1　连接线序

9 芯串口对 9 芯串口	9 芯串口对 25 芯串口	25 芯串口对 25 芯串口
2⊙←→⊙3	2⊙←→⊙3	2⊙←→⊙3
3⊙←→⊙2	3⊙←→⊙2	3⊙←→⊙2
5⊙←→⊙5	5⊙←→⊙7	7⊙←→⊙7

一台普通的 PC 通常有两个串口，用于与外部设备连接进行通信。但有时会发生通信不畅的现象，这有可能是 PC 的串口出现了问题，本任务要求编写串口调试器，用于检验计算机串口通信是否通畅。

（4）任务要求：

① 准备一台拥有两个串口的 PC，按照表 3.1 的说明使用跳线，将串口 COM1 和 COM2 连接，使计算机可以单独作业，自发自收。

② 使用其中一个串口进行发送数据，手动发送一个数据包，包内含有一字节 0ffh（十六进制数，十进制数为 255）。本数据包无须显示。

③ 使用另一个串口接受数据，并显示在界面上，且要将每次接收的数据意义显示在界面上。

（5）术语解析：串行接口简称串口，也就是 COM 接口，是采用串行通信协议的扩展接口。串口通信的概念非常简单，串口按位（bit）发送和接收字节，尽管比按字节（byte）的

并行通信方式慢，但使用的数据线少，在远距离通信中可以节约通信成本。串口通信最重要的参数有数据包、数据位、停止位、奇偶校验和波特率，对于两个进行通信的端口，这些参数必须匹配。

① 数据包：指传送的一个数据字节，包括开始位、停止位、数据位和奇偶校验位。

② 数据位：衡量通信中实际数据位的参数。当计算机发送一个信息包，代表信息数据的位数一般是 5、7 或 8 位，取决于选取的通信协议。例如标准的 ASCII 码是 0～127（7 位），扩展的 ASCII 码是 0～255（8 位）。

③ 停止位：用于表示单个包的最后一位。典型的值为 1、1.5 和 2 位。

④ 奇偶校验：在通信过程中常会出现各种干扰，使所传输的信息发生错误，如某位 1 变成 0 或 0 变成 1。最简单和最常用的检错方法就是奇偶校验。如采用偶校验传送 7 位二进制信息，则在 7 个信息位后加一个偶校验位，如果前 7 位中 1 的个数是偶数，则第 8 位加 0；如果前 7 位中 1 的个数是奇数，则第 8 位加 1，这样便使整个字符代码（共 8 位）1 的个数恒为偶数。接收端如检测到某字符代码中 1 的个数不是偶数，即可判定该为错码而不予接收，通知发送端重发。同理也可采用奇校验。

⑤ 波特率：这是一个衡量通信速度的参数，表示每秒钟传送的数据位（bit）数。例如 300 波特率表示每秒钟发送 300 个数据位（bit）。波特率和传输距离成反比，采用高波特率通信常用于距离很近的仪器间通信。

3.1.2 计划书

1．工作内容

（1）分析项目要完成的功能，确定所使用的软件开发工具和开发系统环境，安装好所需工具软件和系统环境，准备好所需资料（约需要时间：1 个小时）。

（2）项目需求分析，确立开发方案，进行软件的概念分析、软件功能结构分析、逻辑设计、界面的初步设计等（约需要时间：2 个小时）。

（3）软件的物理设计，模块功能设计，代码的初步实施（约需要时间：1/2 个工作日）。

（4）软件的代码实施，代码的功能测试，各模块的组装调试，软件的整体调试，软件注释文档的完善（约需要时间：1 个工作日）。

（5）创建软件系统的安装文件，发布软件的测试版本，并与用户完成软件的整体测试与功能完善（约需要时间：1 个小时）。

2．项目分析

（1）准备一台拥有两个串口的 PC，按照表 3.1 的说明使用跳线将串口 COM1 和 COM2 连接，使计算机可以单独作业，自发自收。

（2）本软件需要使用串口通信控件，在这里使用第三方控件，直接调用 VB6.0 中的 MsComm。

（3）由于本软件需要实现的是自发自收，编程时需要调用两个 MsComm 控件进行编程。一个用于发送数据包，一个用于接收数据包。

（4）发送的数据包无须显示，接收的数据包需要显示在界面上。本软件只需一个接收区域，使用控件 TextBox 即可满足要求。

（5）由于每次显示的数据包都要显示在界面上，所以采用逐行显示的方法。

（6）任务书中要求手动发送数据包，则软件需要一个按钮，用于控制信息的手动发送。

3.2 项目准备

3.2.1 MsComm控件

Microsoft Communications Control（简称 MsComm）是 Microsoft 公司提供的简化 Windows 下串行通信编程的 ActiveX 控件，为应用程序提供了通过串行接口收发数据的简便方法，对于从事工控和单片机工作的人来说串口编程是很常用且很重要的。由于该控件不是 C#自带控件，若想使用，必须在 Windows 的 System32 目录下安装 MsComm32.ocx 文件，而且必须经过注册。

（1）自动注册。MsComm32.ocx 文件可以通过安装 VB6.0 来获得，微软也指出这样不会有冲突。安装 VB6.0 后，MsComm 控件是自动注册的。

（2）手动注册。先将 MsComm.ocx 复制到 System32 目录下，然后新建一个文本文档，在文档里输入以下代码（此段代码在网上可以很方便地搜索到）：

```
REGEDIT4
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}]
@="Microsoft Communications Control, version 6.0"
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\InprocServer32]
@="C:\\WINNT\\SYSTEM\\MSCOMM32.OCX"
"ThreadingModel"="Apartment"
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\VersionIndependent
ProgID]
@="MSCOMMLib.MsComm"
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\ProgID]
@="MSCOMMLib.MsComm.1"
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\TypeLib]
@="{648A5603-2C6E-101B-82B6-000000000014}"
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Version]
@="1.1"
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Implemented
Categories]
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Implemented
Categories\{40FC6ED5-2438-11CF-A3DB-080036F12502}]
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Implemented
Categories\{40FC6ED4-2438-11CF-A3DB-080036F12502}]
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Implemented
Categories\{0DE86A57-2BAA-11CF-A229-00AA003D7352}]
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Implemented
Categories\{0DE86A53-2BAA-11CF-A229-00AA003D7352}]
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Implemented
Categories\{0DE86A52-2BAA-11CF-A229-00AA003D7352}]
[HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Implemented
```

```
Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4}]
        [HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Programmable]
        [HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\Control]
        [HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\MiscStatus]
        @="0"
        [HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\MiscStatus\1]
        @="132497"
        [HKEY_CLASSES_ROOT\CLSID\{648A5600-2C6E-101B-82B6-000000000014}\ToolboxBitmap32]
        @="C:\\WINNT\\SYSTEM\\MSCOMM32.OCX, 1"
        [HKEY_CLASSES_ROOT\MSCOMMLib.MsComm]
        @="Microsoft Communications Control, version 6.0"
        [HKEY_CLASSES_ROOT\MSCOMMLib.MsComm\CLSID]
        @="{648A5600-2C6E-101B-82B6-000000000014}"
        [HKEY_CLASSES_ROOT\MSCOMMLib.MsComm\CurVer]
        @="MSCOMMLib.MsComm.1"
        [HKEY_CLASSES_ROOT\MSCOMMLib.MsComm.1]
        @="Microsoft Communications Control, version 6.0"
        [HKEY_CLASSES_ROOT\MSCOMMLib.MsComm.1\CLSID]
        @="{648A5600-2C6E-101B-82B6-000000000014}"
        [HKEY_LOCAL_MACHINE\Software\CLASSES\TypeLib\{648A5603-2C6E-101B-82B6-000000000
014}]
        [HKEY_LOCAL_MACHINE\Software\CLASSES\TypeLib\{648A5603-2C6E-101B-82B6-000000000
014}\1.1]
        @="Microsoft Comm Control 6.0"
        [HKEY_LOCAL_MACHINE\Software\CLASSES\TypeLib\{648A5603-2C6E-101B-82B6-000000000
014}\1.1\FLAGS]
        @="2"
        [HKEY_LOCAL_MACHINE\Software\CLASSES\TypeLib\{648A5603-2C6E-101B-82B6-000000000
014}\1.1]
        [HKEY_LOCAL_MACHINE\Software\CLASSES\TypeLib\{648A5603-2C6E-101B-82B6-000000000
014}\1.1\win32]
        @="C:\\WINNT\\SYSTEM\\MSCOMM32.OCX"
        [HKEY_LOCAL_MACHINE\Software\CLASSES\TypeLib\{648A5603-2C6E-101B-82B6-000000000
014}\1.1\HELPDIR]
        @="C:\\WINNT\\HELP"
        [HKEY_CLASSES_ROOT\Licenses\4250E830-6AC2-11cf-8ADB-00AA00C00905]
        @="kjljvjjjoquqmjjjvpqqkqmqykypoqjquoun"
```

输入完成后保存，将其命名为 MsComm32.reg，关闭后，左键双击该文件进行注册。

（3）安装。新建程序后，单击“工具”→“选择工具箱项”，打开对话框。选择“COM 组件”项，在出现控件中，找到“Microsoft Communications Control，version 6.0”，选择该项，单击【OK】按钮，就会在工具箱中看到 MsComm 控件的电话图标，将其拖到窗体（Form）中就可以使用了。

（4）参数。MsComm 控件的主要参数有以下几种：

① CommPort：设置并返回通信端口号。

② Settings：以字符串的形式设置并返回波特率、奇偶校验位、数据位、停止位。

③ PortOpen：设置并返回通信端口的状态，也可以打开和关闭端口。
④ Input：从接收缓冲区返回和删除字符。
⑤ Output：向传输缓冲区写一个字符串。

3.2.2 语句

本项目涉及 for 和 try-catch 语句，for 语句主要用于重复执行一条语句或一个语句块，而 try-catch 语句主要用于程序的异常处理。

1．for语句

for 语句是一种重要的循环语句，能够根据循环条件重复执行一条语句或一个语句块，直到条件不具备为止。其一般形式如下：

```
for （表达式 1；表达式 2；表达式 3）
{
循环体
}
```

说明：表达式 1 通常是赋值语句，作为初始化循环计数器的表达式；表达式 2 多为关系表达式，用于测试循环终止条件；表达式 3 是递增或递减循环计数器的表达式。

for 语句首先计算表达式 1，然后计算表达式 2；当表达式 2 的值为真时，执行循环体，再计算表达式 3；再次计算表达式 2 的值，若还为真，仍要执行循环体，计算表达式 3……；直到表达式 2 的值为假，才跳出循环。

由于对表达式 2 的计算发生在循环执行之前，因此 for 语句可能执行零次。另外，for 语句的所有表达式都可默认，但两个分号不能默认。

【例 3-1】 通过一个循环语句输出自然数 1～5。

```
using System;
public class ForLoopTest
{
    public static void Main()
    {
        for (int i = 1; i <= 5; i++)
            Console.WriteLine(i);
    }
}
```

该段代码的输出为：

```
1
2
3
4
5
```

2．try-catch语句

为了捕获并处理异常，可以把可能异常的语句放到 try 子句中。当这些语句在执行过程中出现异常时，try 子句就会捕获这些异常，并转移到相应的 catch 子句中。如果在 try 子句中没有异常，就会执行 try-catch 语句后面的代码，而不会执行 catch 子句中的代码。在通常情况下，try 子句伴着多个 catch 子句，每一个 catch 子句对应一种特定的异常。try-catch 语句的一般语法格式为：

```
try
{
    语句块
}
catch（异常对象声明 1）
{
    语句块 1
}
catch（异常对象声明 2）
{
    语句块 2
}
…
```

说明：当位于 try 子句中的语句产生异常时，系统就会在对应 catch 子句中进行查找，看是否有与抛出的异常类型相同的 catch 子句，如果有就会执行该子句中的语句；如果没有则继续查找，该过程会一直继续下去，直至找到一个匹配的 catch 子句为止；如果一直没找到，则运行时将会产生一个未处理的异常错误。

【例 3-2】 下面所示的程序不仅捕获了除 0 错误，也捕获了数组访问越界错误。程序代码如下：

```
using System;
class ExcelDemo2
{
    static void Main()
    {
        int[] arr1 ={ 2, 5, 8, 3, 13, 32, 56, 61 };
        int[] arr2 ={ 1, 0, 2, 3, 0, 4 };
        for (int j = 0; j < arr1.Length; j++)
        {
            try
            {
                Console.WriteLine("{0}/{1}={2}", arr1[j], arr2[j], arr1[j] / arr2[j]);
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("除数不能为 0");
            }
            catch (IndexOutOfRangeException e)
```

```
                {
                    Console.WriteLine("数组访问越界");
                }
            }
        }
    }
```

该程序的执行结果如下：

```
2/1=2
除数不能为 0
8/2=4
3/3=1
除数不能为 0
32/4=8
数组访问越界
数组访问越界
```

在上例中，每个 catch 语句只响应类型相匹配的异常。有时需要捕获所有的异常，而不管是什么类型异常，使用不带参数的 catch 语句就可以做到这一点。

3.3 项目开发

在 C#.NET 中新建一个项目，命名为“串口调试器”，将文件保存在 D 盘。

3.3.1 界面设计

1．设置接收显示区

选择“工具箱”→“Windows 窗体”，左键双击 Textbox 或左键选中 Textbox 拖曳到 Form1 窗体中，然后进行以下设置：

（1）修改“外观”中的“Text”为“接收区”。

（2）设置 BorderStyle 属性为“Fixed3D”，使该文本框有立体效果。

（3）设置 ScrollBars 属性为“Vertical”，使接收区带有滚动条。

（4）设置 Multiline 属性为“true”，使接收区可以跨越多行。

（5）设置 ReadOnly 属性为“true”，使接收区的文本不能被更改。

2．添加【发送】按钮

选择“工具箱”→“Windows 窗体”，左键双击 Button 或左键选中 Button 拖曳到 Form1 窗体中，修改“外观”中的“Text”为“发送”。

3．添加通信控件MsComm

左键双击或拖动“Microsoft Communications Control，version 6.0 到 Form1 窗体中，这时在 Form1 中就会出现一个电话图标，这个图标就是 MsComm 控件，如图 3.1 所示。

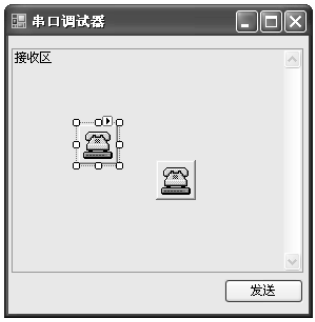


图 3.1　串口调试器

右键单击 MsComm 控件可以打开“属性”窗体，进行如下设置：

（1）设置“缓存”选项卡下的“R 阈值”为“1”。表示接收缓冲区收到每一个字符都会使 MsComm 控件产生 OnComm 事件，若属性设置为 0（默认值）则不产生 OnComm 事件。

（2）在“硬件”选项卡中勾选“RTS 有效”，表示请求到发送有效。

（3）设置“Settings”值为“9600,N,8,1”，表示波特率为 9600Hz，奇偶校验为无校验，数据位数为 8 位，停止位数为 1 位。

3.3.2　代码设计

（1）初始化窗体代码。在初始化串口时，要注意设置串口号：

```
axMsComm1.CommPort = 1;
axMsComm2.CommPort = 2;
```

如果忽略，则串口会重复打开，将会导致串口操作失败，这是不允许的。为了使程序顺利执行，还要考虑报错问题，使用 try 块，在其后面紧接着添加一个 catch 语句，以便在 try 块中的内容发生错误时对其进行处理。在 Form1 窗体的空白位置左键双击，添加如下代码：

```
private void Form1_Load(object sender, EventArgs e)
    {
        try
        {
            axMsComm1.CommPort = 1;
            axMsComm2.CommPort = 2;
            axMsComm1.PortOpen = true;
            axMsComm2.PortOpen = true;
        }
        catch
        {
            MessageBox.Show("串口操作失败");
        }
    }
```

（2）【发送】按钮代码。左键双击【发送】按钮，添加如下代码：

```
private void button1_Click(object sender, System.EventArgs e)
        {
         byte[] bytOut=new byte[1];
         bytOut[0]=255;
         axMsComm2.Output=bytOut;
    }
```

这里只发一个字节 0ffh（十六进制数，十进制为 255），如果要发送更多的数据，可以定义更大的数组。byte[]定义发送的字节数，即向下发送的数据包的字节数，而 bytOut[0]来定义所发送的数据包为 255。串口 2 发送数据包。

（3）通信代码。因为命名空间的问题不能给 axMsComm1.InputMode 赋 0 或 1 这样的值。使用如下语句解决该问题：

```
axMsComm1.InputMode= MsCommLib.InputModeConstants.comInputModeBinary；（赋 0）
axMsComm1.InputMode= MsCommLib.InputModeConstants.comInputModeText；（赋 1）
```

双击通信控件 axMsComm1，添加下列代码：

```
private void axMsComm1_OnComm(object sender, System.EventArgs e)
    {
        string strIn="" ;
        byte[] bytIn;
        object objIn;
        int i;
      axMsComm1.InputMode=MsCommLib.InputModeConstants.comInputModeBinary ;
        objIn=axMsComm1.Input ;
        bytIn =(byte[])objIn;
        for (i=0;i<=(bytIn.Length-1);i++ )
        {
            strIn +="\r\n"+ " "+bytIn[i].ToString("X");
        }
        textBox1.Text+=strIn;
    }
```

该段代码定义接受数据包的信息，其中 objIn 用于装载接收的数据，for 语句部分用于解析接收的数据包，转化为十六进制数据。\n 为换行，\r 为回车。

3.4 实训：MsComm控件的应用

1．项目要求

（1）准备一台拥有两个串口的计算机，使用跳线将串口 COM1 和 COM2 连接，使计算机可以单独作业，自发自收。

（2）界面由发送区和接收区组成，有一个【发送】按钮可以方便用户手动控制。

（3）使用其中的一个串口进行发送数据，手动发送一个数据包，包内数据可以手动输入。要求数据为单字节数据。

（4）使用另一个串口接收数据，并显示在界面上。例如：在发送区输入：255，在接收区显示：255。

（5）工作期限：2 个工作日。

2．设计提示

（1）发送区使用文本框，以便于输入发送信息。

（2）接收区的内容不可更改，可使用 Label 控件，也可采用 TextBox 控件，通过设置属性来实现。

（3）使用两个通信控件，注意设置串口号。

（4）使用中断方式读取信息，注意设置读信息的中断字符数，如果为 0 是不读取的。

（5）使用按钮手动发送数据。

3．项目评价

项目评价是在教师的主持下，通过项目负责人的讲解演示，评估项目的完成情况。评价内容如下：

（1）工具类软件界面是否友好，使用是否方便；

（2）程序使用时不要有错误信息提示，使程序中断运行；

（3）发送数据后，是否有数据回传。

习　　题

1．填空

（1）C#常见的转义符，\n 表示_____，\r 表示_____。

（2）MsComm 控件通过_____和_____，为应用程序提供串行通讯功能。

（3）Textbox 控件的属性中，设置_____属性为_____，使该文本框有立体的效果。设置_____属性为_____，使接收区带有滚动条。设置_____属性为_____，使接收区可以跨越多行。设置_____属性为_____，使接收区的文本不能被更改。

2．选择

（1）下面对于 MsComm 控件属性的描述错误的是（　　）。

A．CommPort 设置并返回通信端口号

B．PortOpen 设置并返回通信端口的状态，也可以打开和关闭端口

C．Input 向传输缓冲区写一个字符串

D．Settings 以字符串的形式设置并返回波特率、奇偶校验、数据位、停止位

（2）设置 TextBox 控件 Multiline 属性为“true”的作用是（　　）。

A．使接收区可以跨越多行

B．使接收区带有滚动条

C．使接收区的文本不能被更改

D．使该文本框有立体的效果

3．MsComm 的全称是什么？如何向 C#添加 MsComm 控件？

4．简述 MsComm 各个属性的作用。

5．写出下列程序代码的输出。

```
using System;
public class ForLoopTest
{
    public static void Main()
    {
        for (int i = 1; i <5; i++)
            Console.WriteLine(i);
    }
}
```

6．写出下列程序代码实现的功能。

```
try
{
    axMsComm1.PortOpen =true;
    axMsComm2.PortOpen =true;
}
catch
{
    MessageBox.Show ("串口操作失败");
}
```

第 4 章　八数码游戏

通过制作小游戏软件，学习 Panel、GroupBox、TabControl 控件，Point、Random 类，#region 和 while 语句的使用。

4.1　项目说明

4.1.1　任务书

（1）项目名称：八数码游戏。

（2）工作期限：3 个工作日。

（3）工作任务：人工智能里面有一个著名的八数码问题，在一个 3×3 的方阵中有 8 个数码，刚开始时它们是无序排列的，并剩下一个空格，空格可以在方阵中移动，要求通过有限次数的移动，使得数字方阵达到有序状态。人工智能里面是要求通过某种搜索方式找到该问题的解，这里则是通过用户自己操作移动数码满足游戏的要求。具体要求如下：

① 需要 8 个数码和一个空格共同组成一个 3×3 的方阵。

② 设置两种游戏目标（即两种有序排列状态），供用户进行选择。

③ 达到游戏目标，提示游戏成功“恭喜恭喜，你已经得到正确结果”，并显示“最终移动的次数”。

（4）项目需达到的技术性能：用户可以使用软件进行游戏，鼠标控制，操作顺畅。

4.1.2　计划书

1．时间分配

（1）分析项目要完成的功能，确定所使用的软件开发工具和开发系统环境，安装好所需工具软件和系统环境，备好所需资料（约需要时间：1/2 个工作日）。

（2）项目需求分析，确立开发方案，进行软件的概念分析、功能结构分析、逻辑设计和界面的初步设计等（约需要时间：1/2 个工作日）。

（3）软件的物理设计，模块功能设计，代码的初步实施（约需要时间：1 个工作日）。

（4）软件的代码实施，代码的功能测试，各模块的组装调试，软件的整体调试，软件注释文档的完善（约需要时间：1/2 个工作日）。

（5）创建软件系统的安装文件，发布软件的测试版本，并与用户完成软件的整体测试与功能完善（约需要时间：1/2 个工作日）。

2．项目分析

（1）游戏共分为三个区域：游戏区、初始状态区和目标状态区。

（2）区域化分可使用 Panel、GroupBox、TabControl 控件。根据各自的特性，游戏区采用 Panel 控件，初始状态区采用 GroupBox 控件，目标状态区采用 TabControl 控件。

（3）游戏区顾名思义就是用于操作得以完成游戏的区域。该区域包含 8 个数码和一个空格，用户游戏过程中，这几个数码和空格是可以移动的。使用 Label 控件进行设计。

（4）初始状态区。用于显示 8 个数码的初始状态，便于游戏时做参考，使用 Label 控件进行设计。

（5）目标状态区。用于显示游戏完成时欲达到的有序状态，使用 Label 控件进行设计。

（6）需要设置一个按钮，用于手动开始游戏，并且在游戏过程中可以手动重新开始游戏。

（7）达到游戏目标，提示游戏成功“恭喜恭喜，你已经得到正确结果”，使用 Message.Box 类，并显示“最终移动的次数”，使用 Label 控件加以实现。

4.2 项目准备

4.2.1 控件

1．Panel控件

Panel 控件也称为面板控件，用于为其他控件提供可识别的分组，常用于按功能细分窗体。使用 Label 控件应注意以下问题：

（1）移动 Panel 控件时，包含的所有控件也将一同移动。

（2）Panel 控件包含的其他控件可单独移动，也可以通过 Controls 属性进行访问。

（3）若要显示滚动条，可将 AutoScroll 属性设置为“true”。

（4）BackColor、BackgroundImage 和 BorderStyle 属性用来自定义面板的外观。

（5）BorderStyle 属性确定面板轮廓为无可视边框（None）、简单线条（FixedSingle），还是阴影线条（Fixed3D）。

2．GroupBox控件

GroupBox 控件也称为分组框控件，功能和用法类似于 Panel 控件，不同之处是它没有滚动条，但能够显示标题。其标题由 Text 属性定义。

3．TabControl控件

TabControl 控件可显示多个选项卡，这些选项卡类似于笔记本中的分隔卡和档案柜文件夹中的标签，选项卡中可包含图片和其他控件。TabControl 控件可用来产生多页对话框，这种对话框出现在 Windows 操作系统中的许多地方，如显示器控制面板。

TabControl 控件最重要的属性是 TabPages。其包含的每个选项卡就是一个 TabPages 对象，单击选项卡时，将为相应的 TabPages 对象引发 Click 事件。

4.2.2 类

1．IContainer类

IContainer 类提供容器的功能。容器是在逻辑上包含零个或更多个组件的对象，能够封装和跟踪组件，可以在多种方案下使用组件和容器，包括可视化方案和非可视化方案。要成为容器，类必须实现 IContainer 接口，该接口支持添加、移除和检索组件的方法。

2．Point类

Point 类表示在二维平面中定义点的 x 和 y 坐标的有序对。由于该类在程序集 using System.Drawing 中，所以在使用 Point 类时，需添加如下代码：

```
using System.Drawing;
```

3．Random类

Random 类能够实现伪随机数生成器。伪随机数生成器是一种能够产生满足某些随机性统计要求的数字序列的设备。伪随机数是以相同的概率从一组有限的数字中选取的，所选数字并不具有完全的随机性，因为它们是用一种确定的数学算法选择的，但是从实用的角度而言，其随机程度已足够了。

随机数的生成从种子开始，如果反复使用同一个种子，就会生成相同的数字系列。产生不同序列的一种方法是使种子值与时间相关，从而对于 Random 类的每个新实例，都会产生不同的数字系列。

为了提高软件性能，可创建一个 Random 类，以便随着时间的推移生成很多随机数，而不用重复新建 Random 类来生成随机数。

4.2.3 语句

1．#region语句

在使用 Visual Studio 2008 代码编辑器的大纲显示功能时，应用#region 语句能够展开或折叠代码块。#region 语句的一般形格式如下：

```
#region    name
```

说明：name是希望给予将出现在代码编辑器中的区域名称。另外，必须用 #endregion指令终止#region块。

【例 4-1】 #region 块的应用。

```
#region MyClass definition
public class MyClass
{
    public static void Main()
    {
    }
}
```

```
#endregion
```

2．while语句

while 语句是一种与 for 语句有所区别的循环语句，其一般形式如下：

```
while（表达式）
{
    循环体；
}
```

说明：表达式用于测试循环终止条件。首先计算表达式，当其值为真时，执行循环体；再计算表达式的值，若还为真，仍要执行循环体……；直到表达式的值为假，才跳出循环。

一般来说，已知循环次数时使用 for 语句，不知道循环次数时使用 while 语句。例如在 100 本书中找到其中一本有特殊标记的书，不知道要看多少本书才会找到，则需要用 while 语句，循环的条件是找到书为止（可能看两本就找到了）；如果要在这 100 本书中挑出全部有破损的书，则要用 for 循环，因为每本都需要看，即看 100 次。

【例 4-2】 使用 while 循环语句找出小于 6 的自然数。

```
using System;
class WhileTest
{
    public static void Main()
    {
        int n = 1;
        while (n < 6)
        {
            Console.WriteLine("Current value of n is {0}",n);
            n++;
        }
    }
}
```

其输出为：

```
Current value of n is 1
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5
```

4.3 项目开发

在 Visual Studio 2008 中新建一个项目，命名为“八数码游戏”，将文件保存在 D 盘中。

4.3.1 界面设计

（1）启动 Visual Studio 2008，单击“项目”部分中的“新建项目”，或者选择“文件”→

“新建”→“项目”，新建一个 C#项目。

（2）这时会自动生成一个 Form1.cs 窗体，右键单击选择“属性”，修改“外观”中的“Text”为“八数码游戏”。

（3）选择“工具箱”→“Windows 窗体”，左键双击 Panel 或左键选中 Panel 拖曳到 Form1 窗体中，将其命名为“mypanel”。

（4）在 Panel 中添加 9 个 Label，分别命名为“img1”、“img2”、“img3”、“img4”、“img5”、“img6”、“img7”、“img8”、“img0”，并修改“外观”中的“Text”为“1”、“2”、“3”、“4”、“5”、“6”、“7”、“8”和空白，在“外观”中的 Font 设置字体类型，在此设置字体为“Microsoft Sans Serif”、字型为“粗体”、大小为“72”。

（5）选择“工具箱”→“Windows 窗体”，左键双击 GroupBox 或左键选中 GroupBox 拖曳到 Form1 窗体中，其默认名称为 groupBox1。

（6）在“GroupBox”中添加 9 个 Label，分别命名为“lbl 1”、“lbl 2”、“lbl 3”、“lbl 4”、“lbl 5”、“lbl 6”、“lbl 7”、“lbl 8”、“lbl 9”，并修改“外观”中的“Text”为“1”、“2”、“3”、“4”、“5”、“6”、“7”、“8”和空白，并在“外观”中的 Font 设置字体类型。

（7）选择“工具箱”→“Windows 窗体”，左键双击 TabControl 或左键选中 TabControl 拖曳到 Form1 窗体中，单击“杂项”中“TabPage”后的按钮，在弹出的“TabPage 集合编辑器”中添加“tabPage1”、“tabPage2”，分别修改“外观”中的“Text”为“目标状态 1”、“目标状态 2”。

（8）分别在“目标状态 1”、“目标状态 2”中各添加 9 个 Label，并分别将 9 个 Label 的“外观”中的“Text”修改为“1”、“2”、“3”、“4”、“5”、“6”、“7”、“8”和空白，并在“外观”中的 Font 设置字体类型。

（9）添加【开始游戏】按钮。

（10）添加 Label，用于显示文本“移动次数”。

（11）添加 Label，命名为“step_lbl”，设置“外观”中的“Text”为“0”，用于显示移动次数。完成的八数码游戏界面如图 4.1 所示。

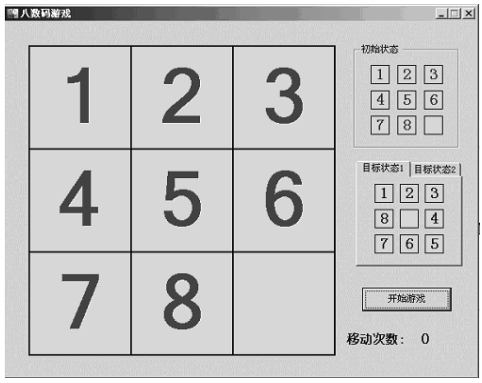


图 4.1　八数码游戏界面

4.3.2　代码设计

1．声明

在声明中定义全局变量，添加代码如下：

```
//初始化目标位置
public static byte[] targetPosition = {1,2,3,8,0,4,7,6,5};
//初始化 9 个坐标点，用于储存 1～8 各数字及空白块的位置
public    Point pos1 = new Point(0,0);
public    Point pos2 = new Point(130,0);
public    Point pos3 = new Point(260,0);
public    Point pos4 = new Point(0,130);
public    Point pos5 = new Point(130,130);
public    Point pos6 = new Point(260,130);
public    Point pos7 = new Point(0,260);
public    Point pos8 = new Point(130,260);
public    Point pos9 = new Point(260,260);
//初始化移动次数
public int moveSteps = 0;
//初始化数字的位置
public static byte[] numPosition = {0,0,0,0,0,0,0,0,0};
```

2.【开始游戏】按钮代码

左键双击【开始游戏】按钮，添加代码如下：

```
private void start_btn_Click(object sender, System.EventArgs e)
    {
        moveSteps = 0;
        //显示移动的次数，单击开始游戏自动清零
        step_lbl.Text = "0";
        numPosition.Initialize();
        mypanel.Enabled = true;
        start_btn.Text = "重新开始";
        InitGame();
    }
```

做一些清理工作，给出各使用量的初始值。当单击一次以后，【开始游戏】按钮变成【重新开始】按钮。调用 InitGame()来初始化游戏区和缩略图（InitGame()是自定义的类）。

3．初始化函数InitGame()

在代码编写器中书写如下代码：

```
private void InitGame()
{
    //定义局部变量
        int i;
        int m;
        int n;
        Random rnd = new Random();
        #region  随机交换坐标并保存初始状态
        Point[] pArray = {pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9 };
```

```
            for (i=0; i<20; i++)
            {
                rnd = new Random(unchecked((int)DateTime.Now.Ticks) + i*i);
                m = rnd.Next(0, 9);
                rnd = new Random(unchecked((int)DateTime.Now.Ticks));
                n = rnd.Next(0, 9);
                Point p = pArray[m];
                pArray[m] = pArray[n];
                pArray[n] = p;
            }
            img0.Location = pArray[0];
            img1.Location = pArray[1];
            img2.Location = pArray[2];
            img3.Location = pArray[3];
            img4.Location = pArray[4];
            img5.Location = pArray[5];
            img6.Location = pArray[6];
            img7.Location = pArray[7];
            img8.Location = pArray[8];
            //将坐标保存到数组：
            numPosition[3*img0.Location.Y/130+img0.Location.X/130] = 0;
            numPosition[3*img1.Location.Y/130+img1.Location.X/130] = 1;
            numPosition[3*img2.Location.Y/130+img2.Location.X/130] = 2;
            numPosition[3*img3.Location.Y/130+img3.Location.X/130] = 3;
            numPosition[3*img4.Location.Y/130+img4.Location.X/130] = 4;
            numPosition[3*img5.Location.Y/130+img5.Location.X/130] = 5;
            numPosition[3*img6.Location.Y/130+img6.Location.X/130] = 6;
            numPosition[3*img7.Location.Y/130+img7.Location.X/130] = 7;
            numPosition[3*img8.Location.Y/130+img8.Location.X/130] = 8;
            #endregion
```

#region 初始化缩略图区域，共 9 个，只列出“lbl1”和“lbl2”的代码，可同样编辑“lbl3”～“lbl0”的代码：

```
        if (numPosition[0] != 0)
        {
            lbl1.BackColor = Color.LightGray;
            lbl1.Text = numPosition[0].ToString();
        }
        else
        {
            lbl1.Text = string.Empty;
            lbl1.BackColor = Color.LightGreen;
        }
        if (numPosition[1] != 0)
        {
            lbl2.BackColor = Color.LightGray;
```

```
        lbl2.Text = numPosition[1].ToString();
    }
    else
    {
        lbl2.Text = string.Empty;
        lbl2.BackColor = Color.LightGreen;
    }
    …
    }
```

4．目标状态改变事件

在代码编写器中书写如下代码：

```
        private void tabControl1_SelectedIndexChanged(object sender, System.EventArgs e)
        {
            if (tabControl.SelectedIndex == 0)
            {
                byte[] state = {1,2,3,8,0,4,7,6,5};
                state.CopyTo(targetPosition, 0);
            }
            else
            {
                byte[] state = {1,2,3,4,5,6,7,8,0};
                state.CopyTo(targetPosition, 0);
            }
        }
```

单击 tabControl 选项卡，用 state 做传导，使 targetPosition 中的序列发生变化。

5．数字移动规则

在代码编写器中书写如下代码：

```
    public void PositionExchg(object sender, System.EventArgs e)
    {
        int i;
        int xx;
        int yy;
        byte tmpNum;
        Label mylbl = (Label)sender;
        int mylbl_x = mylbl.Location.X;
        int mylbl_y = mylbl.Location.Y;
        for (i=0;numPosition[i]!=0; i++);
        //求得 0 的坐标  即空格左上角的坐标
        xx = (i % 3) * 130;
        yy = (i / 3) * 130;
        //水平左移和右移的情况
        if (mylbl_x+130 == xx && mylbl_y == yy || mylbl_x−130 == xx && mylbl_y == yy)
```

```
        {
            mylbl.Location = new Point(xx, mylbl_y);
            img0.Location = new Point(mylbl_x, yy);
            int j = 0;
            while (numPosition[j++] != Convert.ToInt32(mylbl.Text));
            tmpNum = numPosition[i];
            numPosition[i] = numPosition[j-1];
            numPosition[j-1] = tmpNum;
            moveSteps++;
            step_lbl.Text = moveSteps.ToString();
            //检查是否得到目标状态
            GetResult();
            return;
        }
        //垂直上移和下移的情况
        if (mylbl_y+130 == yy && mylbl_x == xx || mylbl_y-130 == yy && mylbl_x == xx)
        {
            mylbl.Location = new Point(mylbl_x,yy);
            img0.Location = new Point(xx,mylbl_y);
            int j = 0;
            while (numPosition[j++] != Convert.ToInt32(mylbl.Text));
            tmpNum = numPosition[i];
            numPosition[i] = numPosition[j-1];
            numPosition[j-1] = tmpNum;
            moveSteps++;
            step_lbl.Text = moveSteps.ToString();
            //检查是否得到目标状态
            GetResult();
            return;
        }
    }
```

左键双击 Label img1，添加代码“PositionExchg(sender,e);”，调用数字移动规则类。同样左键双击 Label img2～Label img8，为其添加代码。

在这里创建的 img1～img8 为 130×130 的方块。

6．游戏成功提示

在代码编写器中书写如下代码：

```
        private void GetResult()
        {
            int i;
            for (i=0; i<9;i++)
            {
                if ((byte)numPosition.GetValue(i) != (byte)targetPosition.GetValue(i))
                {
                    break;
```

```
                }
            }
            if (i == 9)
            {
                MessageBox.Show("恭喜恭喜，你已经得到正确结果\n\n 共计移动了"+moveSteps.
                ToString()+"次","八数码游戏",MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
```

4.4 实训：推箱子游戏

1．项目要求

经典的推箱子是一个古老的游戏，目的是训练逻辑思考能力。在一个狭小的仓库中，要求把木箱放到指定的位置，稍不小心，就会出现箱子无法移动或者通道被堵住的情况，所以需要巧妙地利用有限的空间和通道，合理安排移动的次序和位置，才能顺利地完成任务。

编写程序完成推箱子游戏，要求如下：

（1）棋盘采用 3×3 模式。

（2）游戏中有一只箱子和一个推动箱子的小人。

（3）箱子只能推动不能拉动。

（4）游戏成功提示“恭喜恭喜!”，并显示箱子移动次数。

2．设计提示

本游戏可以看成是八数码游戏中一个数字的还原，不同的是要根据小人的位置移动箱子。例如：小人在箱子左侧，箱子只能向右侧移动，同时小人也要跟着一起移动。

3．项目评价

项目评价是在教师的主持下，通过项目负责人的讲解演示，评估项目的完成情况，评价内容如下：

（1）工具类软件界面是否简洁，操作是否简便；

（2）使用过程中有无错误提示；

（3）能否实现基本功能。

习　　题

1．填空

（1）Panel 控件若要显示滚动条，应将其_____属性设置为“true”。

（2）GroupBox 控件用于_____，分组框的标题由_____属性定义。当移动单个 GroupBox 控件时，包含的所有控件_____移动。

（3）GroupBox 控件类似于 Panel 控件；但只有_____控件显示标题，而且只

有_____控件可以有滚动条。

（4）随机函数可以用_____控件实现。

（5）TabControl 控件最重要的属性是_____，包含单独的选项卡。

（6）Point 表示_____的有序对。

2．该段代码折叠后显示：

```
#region MyClass definition
public class MyClass
{
    public static void Main()
    {
    }
}
#endregion
```

3．下列代码的输出为：

```
using System;
class WhileTest
{
    public static void Main()
    {
        int n = 1;
        while (n < 6)
        {
            Console.WriteLine("Current value of n is {0}", n);
            n++;
        }
    }
}
```

第 5 章　电池生产流转单管理系统

企业生产中常常有大量的生产表单需要管理。本章以电池生产流转单为例，讲解 MainMenu 控件、RadioButton 控件、DataGrid 控件、ComboBox 控件、DateTimePicker 控件的用法，学习 OleDbDataAdapter 类、DataSet 类、OleDbConnection 类、OleDbCommand 类的使用方法，学会链接 Access 数据库以及查询、录入、删除数据的方法。

5.1　项目说明

5.1.1　任务书

（1）项目名称：电池生产流转单管理系统。

（2）工作期限：5 个工作日。

（3）工作任务：编写电池生产流转单管理系统对电池流转单进行管理，参考表 5.1 和表 5.2。

表 5.1　电池生产流转单

电池生产流转单　编号 GY.SD/C2-30			
电池型号			
电池编码			
装配—封盖			
装配数量		装配交付时间	月　　日
装配交付人		封盖接收人	
装配方式	手工□　机械□	检查员	
封盖—灌酸			
封盖数量		封盖交付时间	月　　日
成品数量		封盖交付人	
废品数量		灌酸接收人	
封盖方式	胶封□　热封□	检查员	
灌酸—充电			
灌酸数量		灌酸交付时间	月　　日
成品数量		灌酸交付人	
废品数量		充电交付人	
检查员			
充电—包装			
充电数量		充电交付时间	月　　日

续表

充电—包装			
充电电压低		测开路电压低	
成品数量		充电交付人	
废品数量		包装接收人	
检查员		备注	
包装—入库			
包装数量		包装交付时间	
成品数量		包装交付人	
废品数量		入库接收人	
检查员		备注	

表 5.2　电池生产流转单（附表）

电池生产流转单（附表）　编号 GY.SD/C2—30			
模板生产日期	模板数量	模板型号	操作者

设计要求如下：

① 具有数据录入、数据修改、数据删除等数据编辑模块。

② 具有查询模块，可以按电池型号、电池编码等信息查询。其中电池编码为 10 位数字，前 6 位是生产日期，采用年的后两位、月份两位、日期两位的格式，后 4 位是随机编码。例如 2008 年 1 月 4 日生产电池的电池编码为 0801040001～0801049999 之间的任意一组数。

③ 查询时，可以输入日期进行模糊查询。

（4）项目需达到的技术性能：企业现行的业务管理模式处在对纸质资料进行手工整理和实物存档模式，计算机只作为有限的辅助工具，如 Word 文档、Excel 表格、CAD 设计等简单的应用。制作本软件可方便企业用户对电池流转单报表进行管理，使企业管理趋于自动化。

5.1.2　计划书

1．工作内容

（1）分析项目要完成的功能，确定所使用的软件开发工具和开发系统环境，安装好所需工具软件和系统环境，备好所需资料（约需要时间：1/2 个工作日）。

（2）分析项目需要记录的数据，确定数据结构，确定采用的数据库管理系统，创建数据库（约需要时间：1/2 个工作日）。

（3）项目需求分析，确立开发方案，进行软件的概念分析、软件功能结构分析、逻辑设计、界面的初步设计等（约需要时间：1 个工作日）。

（4）软件的物理设计，模块功能设计，代码的初步实施（约需要时间：2 个工作日）。

（5）软件的代码实施，代码的功能测试，各模块的组装调试，软件的整体调试，软件注释文档的完善（约需要时间：1/2 个工作日）。

（6）创建软件系统的安装文件，发布软件的测试版本，并与用户完成软件的整体测试与功能完善（约需要时间：1/2 个工作日）。

2．项目分析

（1）本软件是单机运行，数据量比较大，数据库采用 Access。

（2）本软件主要分为数据录入和数据查询两个模块，另外添加版本信息模块，便于开发人员管理并升级软件。主界面使用 MainMenu 控件编辑菜单，方便实现主体控制。

（3）数据录入模块是操作员录入数据、修改数据的窗口。界面与表格要方便操作员的录入；修改数据时采用先导入到界面，再进行修改的方法。

① 界面数据的录入信息大部分选用 TextBox 控件。

② 时间信息选用 DateTimePicker 控件。

③ 多个信息选择使用 ComboBox 控件。

④ 选择互斥信息选用 RadioButton 控件。

⑤ 标识信息选用 Label 控件。

⑥ 使用三个按钮，分别用于保存数据、导入数据、修改数据。

（4）查询数据模块使用两个 Grid 控件直接显示主表和附表两个表格中的内容，查询方式采用 if-else 语句进行判断，可以对数据按电池型号、按电池编码、按电池型号和电池编码组合等查询方式。任务书中要求可以模糊查询，使用%代替未知数据信息。使用两个 Button 按钮控件手动控制数据的查询与取消。

（5）数据库使用 OleDbConnection 类进行连接，使用 DataSet 保存数据，使用 OleDbDataAdapter 填充 DataSet 和更新数据源。

（6）版本信息使用 Label 控件进行简单标注。

5.2 项目准备

5.2.1 控件

1．MainMenu控件

Windows 窗体 MainMenu 控件在运行时显示一个菜单。如果在 Windows 窗体设计器中添加该控件，则菜单设计器能够可视地建立主菜单结构，主菜单的所有子菜单和单个项均为 MenuItem 对象。使用 MainMenu 控件应注意以下问题：

（1）通过将 DefaultItem 属性设置为“true”，可以将某菜单项指定为默认项。单击菜单时，默认项以粗体文本显示。

（2）菜单项的 Checked 属性为“true”或“false”，表示该菜单项是否被选定。

（3）菜单项的 RadioCheck 属性自定义选定项的外观：如果 RadioCheck 设置为“true”，则该项旁边出现一个单选按钮；如果 RadioCheck 设置为“false”，则该项旁边出现一个复选标记。

2．RadioButton控件

Windows 窗体 RadioButton 控件为用户提供由两个或多个互斥选项组成的选项集。当用户选择某单选按钮时，同一组中的其他单选按钮不能同时被选定。使用 RadioButton 控件应注

意以下问题：

（1）当 Checked 属性设置为“true”时，单击 RadioButton 控件能够调用 Click 事件处理程序。

（2）当 Checked 属性值更改时，将引发 CheckedChanged 事件。

（3）如果 AutoCheck 属性设置为“true”（默认）时，则当选择单选按钮时，将自动清除该组中所有其他单选按钮的选定状态。通常仅当使用验证代码以确保选定的单选按钮是允许的选项时，才将该属性设置为“false”。

（4）用 Text 属性设置控件内显示的文本，该属性可以包含访问键快捷方式。访问键允许用户通过同时按下 Alt 键和访问键来访问控件。

3．DataGrid控件

Windows 窗体 DataGrid 控件是数据绑定列表控件，在表中显示来自数据源的表格数据。DataGrid 控件允许选择和编辑这些表格数据以及对其进行排序。默认情况下，DataGrid 控件将为数据源中的每个字段生成一个绑定列，每个数据字段都按照在数据库内的存储顺序显示在单独的列中。

4．ComboBox控件

Windows 窗体 ComboBox 控件用于在下拉组合框中显示数据。使用 ComboBox 控件应注意以下问题：

（1）默认情况下，ComboBox 控件分两个部分显示：顶部是一个允许用户输入列表项的文本框，在 Text 属性中设置；第二个部分是列表框，显示用户可以从中进行选择的项的列表，在 Items 属性中设置。

（2）SelectedIndex 属性返回一个整数值，该值与选定的列表项相对应。在代码中更改 SelectedIndex 值，可以通过编程方式更改选定项；列表中的相应项将出现在组合框的文本框部分。如果未选定任何项，则 SelectedIndex 值为−1；如果选定列表中的第一项，则 SelectedIndex 值为 0；如果选定列表中的第二项，则 SelectedIndex 值为 1；依此类推。

（3）SelectedItem 属性与 SelectedIndex 属性类似，但返回其本身，通常是一个字符串。

（4）Items.Count 属性反映列表中的项数，并且 Items.Count 属性值总比 SelectedIndex 的最大可能值大 1，因为 SelectedIndex 是从零开始的。

5．DateTimePicker控件

Windows 窗体 DateTimePicker 控件使用户得以从日期或时间列表中选择单个项。在用来表示日期时，显示为两部分：一个下拉列表（带有以文本形式表示的日期）和一个网格（在单击列表旁边的向下箭头时显示）。使用 DateTimePicker 控件应注意以下问题：

（1）作为替代网格的方法，将 ShowUpDown 属性设置为“true”时，会出现的向上和向下按钮，有助于编辑时间而不是日期。

（2）当 ShowCheckBox 属性设置为“true”时，该控件中的选定日期旁边将显示一个复选框。当选中该复选框时，选定的日期时间值可以更新。当复选框为空时，显示为不可用。

（3）该控件的 MaxDate 和 MinDate 属性确定日期和时间的范围。

（4）Value 属性包含该控件设置的当前日期和时间。该值可以用以下四种格式（由

Format 属性设置）显示：Long、Short、Time 或 Custom。

（5）如果选择自定义格式，则必须将 CustomFormat 属性设置为适当的字符串。

5.2.2 类

1．OleDbDataAdapter类

OleDbDataAdapter 类位于“工具箱”的“数据”项中，表示一组数据命令和一个数据库连接，用于填充 DataSet 和更新数据源。OleDbDataAdapter 类充当 DataSet 和数据源之间用于检索和保存数据的桥接器。OleDbDataAdapter 通过以下方法提供这个桥接器：

（1）使用 Fill 将数据从数据源加载到 DataSet 中，使用 Update 将 DataSet 中所做的更改发回数据源。

（2）当 OleDbDataAdapter 填充 DataSet 时，将为返回的数据创建必需的表和列。

2．DataSet类

DataSet 是 ADO.NET 结构的主要组件，是从数据源中检索到的数据在内存中的缓存。ADO.NET 数据集是以 XML 形式表示的数据视图，是一种数据关系视图，数据集可以类型化或非类型化，具体如下：

（1）类型化 DataSet 类从基类 DataSet 类继承，所以此类型化类承接 DataSet 类的所有功能，并且可与将 DataSet 类的实例作为参数的方法一起使用。

（2）非类型化数据集没有相应的内置架构。与类型化数据集一样，非类型化数据集也包含表、列等，但其只作为集合公开。

在数据集中，默认情况下表和列的名称不区分大小写，即数据集中名为 Customers 的表也可能是指 customers，这符合包括 SQL Server 在内的许多数据库的命名规则，即数据元素的名称无法通过大小写区分。

3．OleDbConnection类

OleDbConnection 类位于“工具箱”的“数据”项中。一个 OleDbConnection 对象，表示到数据源的一个唯一连接。在客户端/服务器数据库系统的情况下，等效于到服务器的一个网络连接。OleDbConnection 对象的某些方法或属性可能不可用，这取决于本机“OLE DB”提供程序所支持的功能。

当创建 OleDbConnection 的实例时，所有属性都设置为初始值。如果超出范围，则不会将其关闭，必须通过调用 Close 或 Dispose 显示关闭该连接。

【例 5-1】 下面的示例创建一个 OleDbConnection。

```
        myConnectionString = "Provider=SQLOLEDB;Data Source=localhost;Initial Catalog=Northwind;
Integrated Security=SSPI;";
        OleDbConnection myConnection = new OleDbConnection(myConnectionString);
```

4．OleDbCommand类

OleDbCommand 类表示要对数据源执行的 SQL 语句或存储过程。当创建 OleDbCommand 的实例时，读/写属性将被设置为初始值。OleDbCommand 类对数据源执行命令的方法如下：

（1）ExecuteReader：执行返回行的命令。如果用 ExecuteReader 执行如 SQL SET 语句等命令，则可能达不到预期的效果。

（2）ExecuteNonQuery：执行 SQL INSERT、DELETE、UPDATE 和 SET 语句等命令。

（3）ExecuteScalar：从数据库中检索单个值，例如一个聚合值。

（4）可以重置 CommandText 属性并重复使用 OleDbCommand 对象。但是在执行新的命令或先前命令之前，必须关闭 OleDbDataReader。

（5）如果执行 OleDbCommand 的方法 OleDbConnection 生成致命的 OleDbException（例如，SQL Server 严重级别等于或大于 20）的错误，连接可能会关闭。但是，用户可以重新打开连接并继续操作。

【例 5-2】 以下示例将 OleDbCommand 和 OleDbDataAdapter 以及 OleDbConnection 一起使用，从 Access 数据库中选择行，然后返回已填充的 DataSet。向该示例传递一个已初始化的 DataSet、一个连接字符串、一个查询字符串（是一个 SQL SELECT 语句）和一个表示源数据库表名称的字符串。

```
public void ReadMyData(string myConnString)
{
    string mySelectQuery = "SELECT OrderID, CustomerID FROM Orders";
    OleDbConnection myConnection = new OleDbConnection(myConnString);
    OleDbCommand myCommand = new OleDbCommand(mySelectQuery, myConnection);
    myConnection.Open();
    OleDbDataReader myReader = myCommand.ExecuteReader();
    try
    {
       while (myReader.Read())
       {
          Console.WriteLine(myReader.GetInt32(0) + ", " + myReader.GetString(1));
       }
    }
        finally
    {
       myReader.Close();
       myConnection.Close();
    }
}
```

5.3 项目开发

在 Visual Studio 2008 中新建一个项目，命名为“电池生产流转单管理系统”，将文件保存在 D 盘中。通过项目分析，本软件窗体分为系统主控制窗体、数据录入窗体、查询数据窗体和版本信息窗体。

5.3.1 数据库操作

1．数据库、表设计

本系统所需数据库为 Access 数据库，在“D:\电池生产流转单”目录下创建名为“dc.mdb”的数据库，并在库中建立主表、附表两个表，数据库及表结构如图 5.1 所示。

2．连接数据库

（1）选择“工具箱”→“数据”→OleDbConnection。

（2）在界面的任意位置单击，界面下方就会出现一个名为 OleDbConnection1 的连接，左键单击或右键选择“属性”。

（3）在 ConnectionString 的下拉菜单中选择“新建连接”，打开“数据库链接属性”选项卡。

（4）在“提供程序”中选择“Microsoft Jet 4.0 OLE DB Provider”，单击【下一步】按钮，打开“连接”项，单击“选择或输入数据库名称”后的按钮，选择“dc.mdb”，单击【确定】按钮。

除上述方法，还可以用代码实现。工业上一般应用这种方法，因为这比较直观、易操作。

图 5.1　数据库及表结构

5.3.2 系统主控制窗体

系统的主控制界面是用户进入到各个界面的总控制界面，当系统启动就可进入到此界面进行相关操作。

1．窗体界面设计

（1）设计窗体外观。在新建项目时会自动新建一个窗体 Form1。具体属性设置如下：

① 将“设计”的 name 项修改为“main”，通常主窗体名称都会被命名为“main”，在以

后的各章里就不再重复了。窗体通常是根据其功能命名成相关的英文或是拼音缩写。

② 单击“布局”下 StartPosition 后的下拉菜单，选择“CenterScreen”，这表示窗体第一次出现的位置在计算机的正中央，亦可以选择其他位置，详细情况可以按下 F1 键查询。

③ 单击“外观”下的 BackgroundImage 后的按钮，选择背景图片。在 Text 后添加“电池生产流转单管理系统 1.0”，这是要在窗体边框上显示的文本。

在单击相应的选项时，在“属性”窗体的下方会有相应的中文说明，可以根据自己的习惯、喜好进行设计。

（2）添加菜单。将“工具箱”中“Windows 窗体”中的 MainMenu 拖曳到窗体中，就会看到一个上边写着“请在此输入”的可输入的文本框，首先填写“数据录入”，在“数据录入”后边的文本框中输入“查询数据”，在“查询数据”后输入“版本信息”，在“版本信息”后输入“退出系统”。和 Word 一样，这些在最后形成的工具软件中都是按钮，C#都会给这些按钮起一个默认的名字，这个只是在编程的时候会看到，也可以根据用户喜好进行重新命名。

最后完成的系统主控制界面如图 5.2 所示。

图 5.2　电池生产流转单管理系统主控制界面

2．窗体代码设计

（1）【退出系统】按钮代码。左键双击【退出系统】按钮，添加如下代码：

```
private void menuItem2_Click(object sender, System.EventArgs e)
{
Application.Exit();//实现退出系统的功能
}
```

（2）【数据录入】按钮代码。左键双击【数据录入】按钮，添加如下代码：

```
private void menuItem3_Click(object sender, System.EventArgs e)
{
    Form inputForm=new input();
    inputForm.ShowDialog();
//打开输入数据窗体，打开后，该窗体处于最上层，其他窗体不可编辑
}
```

（3）【查询数据】按钮代码。左键双击【查询数据】按钮，添加如下代码：

```
private void menuItem4_Click(object sender, System.EventArgs e)
{
    Form selectForm=new select();
    selectForm.ShowDialog();
    //打开查询数据窗体
    }
```

（4）【版本信息】按钮代码。左键双击【版本信息】按钮，添加如下代码：

```
private void menuItem4_Click(object sender, System.EventArgs e)
{
    Form helpForm=new help();
    helpForm.ShowDialog();
}
```

5.3.3 数据的录入窗体

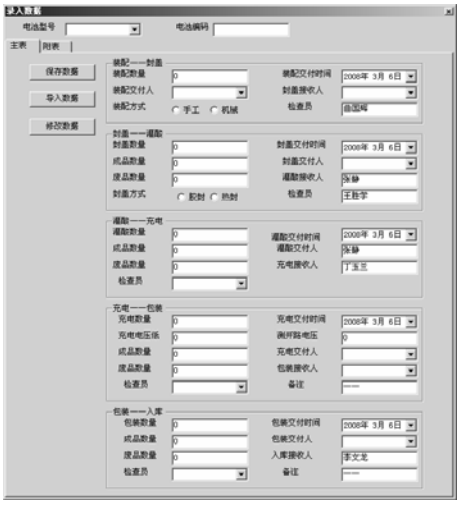
1．窗体界面窗体

为了便于操作者录入信息，界面应尽可能贴近表格，防止录入错误。

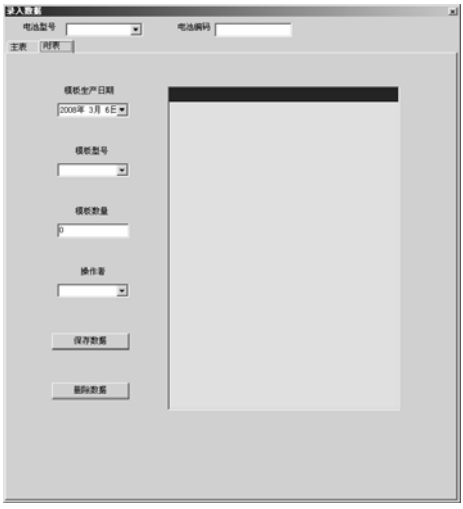
（1）选择“项目”→“添加 Windows 窗体”，添加一个新窗体。在“外观”下的 Text 后输入“数据”，在“设计”的 name 项中输入“input”。

（2）单击“工具箱”中的“Windows 窗体”下的 tabControl，在窗体选择一个合适的位置放置，右键选择“属性”，单击“属性”中“杂项”下的 tabPage 后的按钮，打开“tabPage 集合编辑器”，单击【添加】按钮，分别添加 tabPage1、tabPage2，分别设置“tabPage 属性”中的“外观”下的 Text 为“主表”、“附表”。

（3）将“工具箱”中的“Windows 窗体”下的 Label、TextBox、ComboBox、Button、DateTimePicker、RadioButton 拖曳到窗体中，分别按图 5.3（a）和图 5.3（b）所示布局。



（a）



（b）

图 5.3　录入数据界面

本界面是用于向数据库中添加数据的。从界面上可以看到录入界面的主表项中包含了保存数据、导入数据、修改数据等功能，而在其他项中包含了保存数据及删除数据等功能。在实际应用时，由于数据量大，但数据单一，可以为一些常用数据添加默认值，这样输入时会比较方便。

2．配置数据适配器OleDbDataAdapter1

接着需要配置 OleDbDataAdapter（数据适配器），这个组件位于“工具箱”的“数据”选项卡中，在向窗体添加时会自动弹出如图 5.4 所示的“数据适配器配置向导”，亦可右键单击 OleDbDataAdapter 选择“配置数据适配器”打开配置向导。

单击【下一步】按钮，再单击【新建连接】按钮，选择“dc.mdb”数据库，选好后，连续单击两次【下一步】按钮，这时会出现如图 5.5 所示的窗体。

单击“查询生成器”，在【添加表】中选择“主表”，单击【添加】按钮，然后单击【关闭】按钮，在表中选择“所有列”，单击【确定】按钮，继续单击【下一步】按钮，这时会出现如图 5.6 所示的界面，说明配置数据适配器成功。

图 5.4　数据适配器配置向导

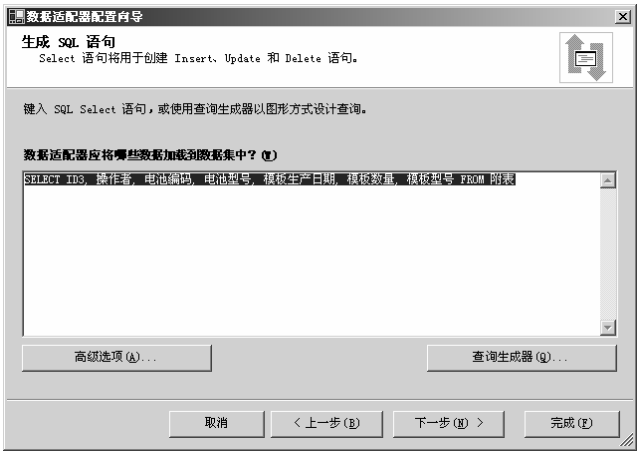


图 5.5　生成 SQL 语句

图 5.6　数据适配器成功界面

同理配置 OleDbDataAdapter2、OleDbDataAdapter3。

3．添加dataSet、dataGrid

dataSet、dataGrid 都是用于存储数据的，不妨多添加几个，以便数据的查询与修改。

4．窗体代码设计

（1）主表选项卡【保存数据】按钮代码。本系统录入的数据是以“电池编码”和“电池型号”来唯一定义一组数据的，所以在存储数据前，需首先看一下该组数据是否存在，如果存在，则提示“主表已保存本批次电池数据”；如果不存在，则转到添加数据函数 InsertZB()（保存主表信息），代码如下：

```
        private void button1_Click(object sender, System.EventArgs e)
        {
    //判断是否输入了“电池编码”和“电池型号”
            if(comboBox12.Text!=""&&textBox2.Text!="")
            {
    //查询该组信息在数据库中是否存在
                MyDataSet.Clear();
            oleDbDataAdapter3.SelectCommand.CommandText="SELECT * FROM 主表 WHERE
[电池编码] = '" + textBox2.Text +"' and [电池型号] = '" + comboBox12.Text + "' ;
            oleDbDataAdapter3.SelectCommand.Connection.Open();
            oleDbDataAdapter3.SelectCommand.ExecuteNonQuery();
            oleDbDataAdapter3.SelectCommand.Connection.Close();
            oleDbDataAdapter3.Fill(MyDataSet,"主表");
            DataTable dataTable0=MyDataSet.Tables[0];
    //如果存在，则提示“主表中已保存本批次电池数据”
            if(dataTable0.Rows.Count!=0)
            {
                MessageBox.Show("主表中已保存本批次电池数据","提示！");
            }
    //如果不存在存储信息
```

```
                else
                {
                    try
                    {
                        InsertZB();
                        MessageBox.Show ( "数据保存完毕！", "提示！" ) ;
                    }
        //错误判断：显示错误信息
                    catch ( Exception ed )
                    {
                        MessageBox.Show ( "添加数据发生" + ed.ToString ( ) , "错误！" ) ;
        //关闭数据库链接
                        oleDbDataAdapter1.InsertCommand.Connection.Close();
                    }
                }
            }
            else
            {
                MessageBox.Show ( "电池型号及电池编码不能为空！", "错误！" ) ;
            }
        }
```

（2）主表选项卡【修改数据】按钮代码。双击【修改数据】按钮，修改分为两步：一是删除原信息，二是重新录入信息，代码如下：

```
        private void button3_Click(object sender, System.EventArgs e)
            {
        //对话框提示“是否修改当前数据”，选择“是”则继续，“否”则停止
        DialogResult r = MessageBox.Show ( "是否修改当前记录！", "修改记录！", MessageBoxButtons.
YesNo , MessageBoxIcon.Question ) ;
                int ss = ( int ) r ;
                if ( ss == 6 ) // 按“确定”按钮
                {
                    load();
        //删除该组数据
            oleDbDataAdapter3.DeleteCommand.CommandText="DELETE FROM 主表 WHERE [电池编
码]= '" + textBox2.Text +"' and [电池型号] = '" + comboBox12.Text + "'";
                oleDbDataAdapter3.DeleteCommand.Connection.Open();
                oleDbDataAdapter3.DeleteCommand.ExecuteNonQuery();
                oleDbDataAdapter3.DeleteCommand.Connection.Close();
        //录入信息；录入完成后，提示“数据修改完毕”
                InsertZB();
                MessageBox.Show ( "数据修改完毕！", "提示！" ) ;
                }
            }
```

（3）附表选项卡【保存数据】按钮代码。左键双击【保存数据】按钮，添加如下代码：

```
        private void button5_Click(object sender, System.EventArgs e)
            {
                if(comboBox12.Text!=""&&textBox2.Text!="")
                {
                    try
                    {
                        load();
                oleDbDataAdapter1.InsertCommand.CommandText="INSERT INTO 附表(操作者，电
池编码，电池型号，模板生产日期，模板数量，模板型号) VALUES (' " +comboBox11.Text +" ', ' " +
textBox2.Text +" ', ' " + comboBox12.Text +" ',' " +  dateTimePicker18. Value+" ', ' " + textBox84.Text +" ', ' " +
comboBox10.Text +" ')";
                        oleDbDataAdapter1.InsertCommand.Connection.Open();
                        oleDbDataAdapter1.InsertCommand.ExecuteNonQuery();
                        oleDbDataAdapter1.InsertCommand.Connection.Close();
                        dataSet1.Clear();
                        oleDbDataAdapter1.Fill(dataSet1);
                        dataGrid1.SetDataBinding(dataSet1,"附表");
                        MessageBox.Show ( "数据保存完毕！ " , "提示！ " ) ;
                        }
                        catch ( Exception ed )
                        {
                            MessageBox.Show ( "添加数据记录发生 " + ed.ToString () , "错误！ " ) ;
                            oleDbDataAdapter1.InsertCommand.Connection.Close();
                        }
                    }
                    else
                    {
                        MessageBox.Show ( "电池型号及电池编码不能为空！ " , "错误！ " ) ;
                    }
                }
```

（4）附表选项卡【删除数据】按钮代码。左键双击【删除数据】按钮，添加代码如下：

```
        private void button6_Click(object sender, System.EventArgs e)
            {
                textID.Text=dataGrid1[dataGrid1.CurrentRowIndex,0].ToString();
                DialogResult r = MessageBox.Show ( "是否删除当前记录！ " , "删除当前记录！ " , Message
BoxButtons.YesNo , MessageBoxIcon.Question ) ;
                int ss = ( int ) r ;
                    if ( ss == 6 ) // 按“确定”按钮
                    {
                        try
                        {
                            oleDbDataAdapter1.DeleteCommand.CommandText="DELETE FROM 附表
WHERE [ID3] = " + textID.Text + "";
                            oleDbDataAdapter1.DeleteCommand.Connection.Open();
                            oleDbDataAdapter1.DeleteCommand.ExecuteNonQuery();
```

```
                    oleDbDataAdapter1.DeleteCommand.Connection.Close();
                    dataSet1.Clear();
                    oleDbDataAdapter1.Fill(dataSet1);
                    dataGrid1.SetDataBinding(dataSet1,"附表");
                }
                catch ( Exception ed )
                {
                    MessageBox.Show ( "删除记录错误信息:" + ed.ToString () , "错误！ " ) ;
                    oleDbDataAdapter1.DeleteCommand.Connection.Close();
                }
            }
        }
```

由于删除数据时，需要一个唯一标识来确定删除行的位置，该位置需要储存在一个TextBox 中，所以在界面中添加一个 TextBox，并将其命名为“textID”。

（5）界面初始化代码。初始化系统，在附表选项卡中显示数据库中“附表”的信息。添加代码如下：

```
    private void input_Load(object sender, System.EventArgs e)
        {
            dataSet0.Clear();
            this.oleDbDataAdapter1.SelectCommand.CommandText="SELECT  模板生产日期, 模板数量, 模板型号,操作者 FROM 附表";
            this.oleDbDataAdapter1.SelectCommand.Connection.Open();
                this.oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
            this.oleDbDataAdapter1.SelectCommand.Connection.Close();
            dataSet0 = new DataSet("附表");
            oleDbDataAdapter1.Fill(dataSet0,"附表");
            dataGrid1.SetDataBinding(dataSet0,"附表");
            radioButton1.Checked=true;
            radioButton6.Checked=true;
        }
```

（6）向数据库中的主表添加信息。添加代码如下：

```
    private void InsertZB()
        {
            if(radioButton1.Checked)
            {
            oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO 主表(电池型号,电池编码, 工步, 数量,交付时间,交付人,接收人,检查员,方式) VALUES ('" + comboBox12.Text +"', '" + textBox2.Text +" ', '" + "装配—封盖" +"', '" + textBox4.Text +" ','" + dateTimePicker1.Value
+"', '" + comboBox1.Text +"', '" + comboBox8.Text +"', '" + textBox7.Text +"','"+"手工"+"')";
                oleDbDataAdapter3.InsertCommand.Connection.Open();
                oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                oleDbDataAdapter3.InsertCommand.Connection.Close();
            }
            else
```

```
                {
            oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO 主表(电池型号,电池编码,
工步, 数量,交付时间,交付人,接收人,检查员,方式) VALUES ('" + comboBox12.Text +"', '" + textBox2.Text +"',
'" + "装配—封盖" +"', '" + textBox4.Text +"','" +  dateTimePicker1.Value +"', '" + comboBox1.Text +"', '" +
comboBox8.Text +"', '" + textBox7.Text +"','"+"机械"+"')";
                    oleDbDataAdapter3.InsertCommand.Connection.Open();
                    oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                    oleDbDataAdapter3.InsertCommand.Connection.Close();
                }
                if(radioButton6.Checked)
                {
            oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO 主表(电池型号,电池编码,
工步, 数量, 成品数量, 废品数量,交付时间,交付人,接收人,检查员,方式) VALUES ('" + combo Box12.Text +" ',
'" + textBox2.Text +"', '" + "封盖—灌酸" +"', '" + textBox14.Text +"', '" + textBox16.Text +"', '" + textBox13.Text
+"','" +  dateTimePicker3.Value +"', '" + comboBox2. Text +"' ,'" + textBox12.Text +"', '" + textBox11.Text
+"' ,'"+"胶封"+"')";
                    oleDbDataAdapter3.InsertCommand.Connection.Open();
                    oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                    oleDbDataAdapter3.InsertCommand.Connection.Close();
                }
                else
                {
            oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO 主表(电池型号,电池编码,
工步, 数量, 成品数量, 废品数量,交付时间,交付人,接收人,检查员,方式) VALUES ('" + combo Box12.Text +"',
'" + textBox2.Text +"', '" + "封盖—灌酸" +"', '" + textBox14.Text +"', '" + textBox16.Text +"', '" + textBox13.Text
+"','" +dateTimePicker3.Value +"', '" + comboBox2.Text +"' ,'" + textBox12.Text +"', '" + textBox11.Text +"' ,'"+"热
封"+"')";
                    oleDbDataAdapter3.InsertCommand.Connection.Open();
                    oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                    oleDbDataAdapter3.InsertCommand.Connection.Close();
                }
            oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO 主表(电池型号,电池编码,
工步, 数量, 成品数量, 废品数量,交付时间,交付人,接收人,检查员) VALUES ('" + comboBox12.Text +"', '" +
textBox2.Text +"', '" + "灌酸—充电" +"', '" + textBox3.Text +"','" +textBox8.Text +"','" +textBox17.Text +"','"
+dateTimePicker2.Value +"', '" + textBox9.Text +"', '" +textBox10.Text +"', '" + comboBox3.Text +"')";
                    oleDbDataAdapter3.InsertCommand.Connection.Open();
                    oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                    oleDbDataAdapter3.InsertCommand.Connection.Close();
            oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO 主表(电池型号,电池编码,
工步, 数量, 成品数量, 废品数量,交付时间,交付人,接收人,检查员,备注,充电电压低,测开路电压低) VALUES
('" + comboBox12.Text +"', '" + textBox2.Text +"', '" + "充电—包装" +"', '" +textBox24.Text +"', '" +
textBox26.Text +"', '" + textBox23.Text +"','" +dateTimePicker4.Value +"', '" + comboBox6.Text +"', '" +
comboBox5.Text +"', '" + comboBox4.Text +"', '" + textBox35.Text +"', '" + textBox20.Text +"', '" + textBox19.Text
+"')";
                    oleDbDataAdapter3.InsertCommand.Connection.Open();
                    oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                    oleDbDataAdapter3.InsertCommand.Connection.Close();
```

```
            oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO 主表(电池型号,电池编码,
工步, 数量, 成品数量, 废品数量,交付时间,交付人,接收人,检查员,备注) VALUES ('" + combo Box12.Text +"',
'" + textBox2.Text +"', '" + "包装—入库" +"', '" + textBox32.Text +"', '" + textBox28.Text +"', '" + textBox31.Text
+"','" +   dateTimePicker5.Value +"', '" + comboBox7. Text +"', '" + textBox30.Text +"', '" + comboBox9.Text +"', '"
+ textBox34.Text +"')";
                        oleDbDataAdapter3.InsertCommand.Connection.Open();
                        oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                        oleDbDataAdapter3.InsertCommand.Connection.Close();
                }
```

5.3.4 数据的查询窗体

1．窗体界面设计

同录入界面一样，查询界面亦需尽可能地贴近表格，如图 5.7 所示。

图 5.7　查询数据界面

在查询数据模块里可以依据一定的条件查询数据。选择“项目”→“添加 Windows 窗体”，添加一个新窗体。在“外观”下的 Text 后输入“查询数据”，在“设计”的 name 项中输入“Select”。其他同“录入数据”窗体。

2．窗体代码设计

查询数据同导入数据代码基本相似，只不过查询时可以模糊查询，而不必像导入数据时必须输入正确信息。

（1）【取消】按钮代码。左键双击【取消】按钮，添加如下代码：

```
    private void button8_Click(object sender, System.EventArgs e)
        {
            comboBox12.Text="";
            textBox2.Text="";
        }
```

（2）【查找】按钮代码。左键双击【查找】按钮，添加如下代码：

```
        private void button1_Click(object sender, System.EventArgs e)
        {
            if(textBox2.Text!="")
            {
                if(textBox2.Text.Length < 6)
                {
                    MessageBox.Show("电池编码输入长度不够","提示！");
                }
                else
                {
                    if(comboBox12.Text!="")
                    {
                        dataSet1.Clear();
                    oleDbDataAdapter2.SelectCommand.CommandText="SELECT * FROM 主表 WHERE
 [电池编码] like '%" + textBox2.Text.Substring(0,6) +"% ' and [电池型号] = '" + combo Box12.Text + "'" ;
                        oleDbDataAdapter2.SelectCommand.Connection.Open();
                        oleDbDataAdapter2.SelectCommand.ExecuteNonQuery();
                        oleDbDataAdapter2.SelectCommand.Connection.Close();
                        oleDbDataAdapter2.Fill(dataSet1,"主表");
                        dataGrid2.SetDataBinding(dataSet1,"主表");
                        DataTable dataTable0=dataSet1.Tables[0];
                        if(dataTable0.Rows.Count==0)
                        {
                            MessageBox.Show("主表中没有找到相关信息","查询错误！");
                        }
                        dataSet2.Clear();
                    oleDbDataAdapter1.SelectCommand.CommandText="SELECT * FROM 附表 WHERE
[电池编码] like '%" + textBox2.Text.Substring(0,6) +"%' and [电池型号] = '" + combo Box12.Text + "'" ;
                        oleDbDataAdapter1.SelectCommand.Connection.Open();
                        oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                        oleDbDataAdapter1.SelectCommand.Connection.Close();
                        oleDbDataAdapter1.Fill(dataSet2,"附表");
                        dataGrid1.SetDataBinding(dataSet2,"附表");
                        DataTable dataTable2=dataSet2.Tables[0];
                        if(dataTable2.Rows.Count==0)
                        {
                            MessageBox.Show("附表项没有找到相关信息","查询错误！");
                        }
                    }
                    else
                    {
                        dataSet1.Clear();
                    oleDbDataAdapter2.SelectCommand.CommandText="SELECT * FROM 主表 WHERE
[电池编码] like '%" + textBox2.Text.Substring(0,6) +"%' " ;
                        oleDbDataAdapter2.SelectCommand.Connection.Open();
```

```
                    oleDbDataAdapter2.SelectCommand.ExecuteNonQuery();
                    oleDbDataAdapter2.SelectCommand.Connection.Close();
                    oleDbDataAdapter2.Fill(dataSet1,"主表");
                    dataGrid2.SetDataBinding(dataSet1,"主表");
                    DataTable dataTable0=dataSet1.Tables[0];
                    if(dataTable0.Rows.Count==0)
                    {
                        MessageBox.Show("主表中没有找到相关信息","查询错误！");
                    }
                    dataSet2.Clear();
                oleDbDataAdapter1.SelectCommand.CommandText="SELECT * FROM 附表 WHERE
[电池编码] like '%" + textBox2.Text.Substring(0,6) +"%' " ;
                    oleDbDataAdapter1.SelectCommand.Connection.Open();
                    oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                    oleDbDataAdapter1.SelectCommand.Connection.Close();
                    oleDbDataAdapter1.Fill(dataSet2,"附表");
                    dataGrid1.SetDataBinding(dataSet2,"附表");
                    DataTable dataTable2=dataSet2.Tables[0];
                    if(dataTable2.Rows.Count==0)
                    {
                        MessageBox.Show("附表项没有找到相关信息","查询错误！");
                    }
                }
            }
        }
        else
        {
            if(comboBox12.Text!="")
            {
                dataSet1.Clear();
            oleDbDataAdapter2.SelectCommand.CommandText="SELECT * FROM 主表 WHERE
[电池型号] = '" + comboBox12.Text + "'" ;
                oleDbDataAdapter2.SelectCommand.Connection.Open();
                oleDbDataAdapter2.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter2.SelectCommand.Connection.Close();
                oleDbDataAdapter2.Fill(dataSet1,"主表");
                dataGrid2.SetDataBinding(dataSet1,"主表");
                DataTable dataTable0=dataSet1.Tables[0];
                if(dataTable0.Rows.Count==0)
                {
                    MessageBox.Show("主表中没有找到相关信息","查询错误！");
                }
                dataSet2.Clear();
            oleDbDataAdapter1.SelectCommand.CommandText="SELECT * FROM 附表 WHERE [电
池型号] = '" + comboBox12.Text + "'" ;
                oleDbDataAdapter1.SelectCommand.Connection.Open();
                oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
```

```
                    oleDbDataAdapter1.SelectCommand.Connection.Close();
                    oleDbDataAdapter1.Fill(dataSet2,"附表");
                    dataGrid1.SetDataBinding(dataSet2,"附表");
                    DataTable dataTable2=dataSet2.Tables[0];
                    if(dataTable2.Rows.Count==0)
                    {
                        MessageBox.Show("附表项没有找到相关信息","查询错误！");
                    }
                }
                else
                {
                    MessageBox.Show ( "请输入查询条件 ","提示！" ) ;
                }
            }
        }
```

在 C#中，用%代替字符串进行模糊查询。

5.3.5 版本窗体

版本窗体记录软件的版本信息，版本界面没有自主编辑代码，全部为系统自动生成代码。版本信息记录了该系统的完成日期和编辑的版本号，便于修改。如图 5.8 所示。

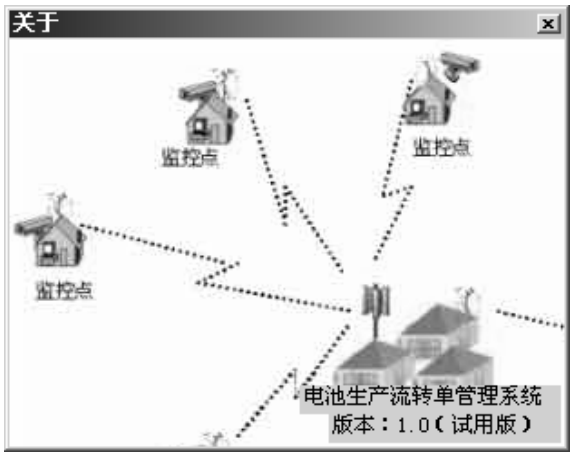


图 5.8 版本信息窗体

5.4 实训：电池“充电—包装”管理系统

1．项目要求

在本章完成的软件基础上添加报表，如表 5.3 所示。

表 5.3　电池生产流转单（表 2）

电池型号			
主表充电交付时间		月　　日	
充电—包装			
放电电压		测开路电压低	
二次放电		一次放电	
放电时间	月　　日	放电时间	月　　日
放电数量		放电数量	
合格数量		合格数量	
不合格数量		不合格数量	
废品数量		废品数量	
充电交付时间	月　　日	充电交付时间	月　　日
充电交付人		充电交付人	
包装接收人		包装接收人	
检查员		检查员	
备注		备注	
三次放电		二次放电	
放电时间	月　　日	放电时间	月　　日
放电数量		放电数量	
合格数量		合格数量	
不合格数量		不合格数量	
废品数量		废品数量	
充电交付时间	月　　日	充电交付时间	月　　日
充电交付人		充电交付人	
包装接收人		包装接收人	
检查员		检查员	
备注		备注	
充电电压低		测开路电压低	
充电时间	月　　日	充电时间	月　　日
充电数量		充电数量	
合格数量		合格数量	
废品数量		废品数量	
充电交付时间	月　　日	充电交付时间	月　　日
充电交付人		充电交付人	
包装接收人		包装接收人	
检查员		检查员	
备注		备注	

结合表 5.3 的内容，编写电池生产流转单管理系统，要求如下：

（1）有数据录入、数据修改、数据删除等数据编辑模块。

（2）有查询模块，可以按电池型号查询。

2．设计提示

（1）需要三个按钮，方便数据的录入、修改和添加。
（2）数据录入界面按照表 5.3 的表单尽量贴近设计，便于用户使用。
（3）查询模块中，按电池型号查询时可采用模糊查询，使用%匹配字符串。

3．项目评价

项目评价是在教师的主持下，通过项目负责人的讲解演示，评估项目的完成情况，评价内容如下：
（1）工具类软件界面是否简洁，操作是否简便；
（2）使用过程中有无错误提示；
（3）能否实现基本功能。

习　　题

1．填空
（1）利用 DateTimePicker 控件可以获得_____。
（2）Windows 窗体 MainMenu 组件在运行时显示一个_____。
（3）DataSet 用于_____，就像是一个缓存区，可以方便地处理数据。
（4）如果 OleDbConnection 超出范围，则_____。因此，必须通过调用_____或_____显式关闭该连接。

2．选择
（1）下列语句中，关于 MainMenu 描述正确的是（　　）。
A．主菜单的所有子菜单和单个项均为 MenuItem 对象
B．通过将 DefaultItem 属性设置为“true”，可以将某菜单项指定为默认项。单击菜单时，默认项以粗体文本显示
C．菜单项的 RadioCheck 属性自定义选定项的外观：如果 RadioCheck 设置为“true”，则该项旁边出现一个单选按钮；如果 RadioCheck 设置为“false”，则该项旁边出现一个复选标记
D．菜单项的 Checked 属性为“true”或“false”，它指示该菜单项是否选定
（2）DateTimePicker 控件 Value 属性包含该控件设置的当前日期和时间。该值格式有哪些（　　）。
A．Long　　B．Short　　C．Time　　D．Custom

3．写出按照电池型号查找数据的 SQL 语句。
4．OleDbDataAdapter 的作用是什么？
5．OleDbCommand 表示什么？

第 6 章　锂电池原材料成本管理系统

在企业中，大量的数据计算浪费物力和财力。锂电池原材料成本管理系统就是针对这一问题设计的，本系统涉及水晶报表控件、BindingManagerBase 类、switch 语句的使用方法，重点是学习多级别登录的实现方法。

6.1　项目说明

6.1.1　任务书

（1）项目名称：锂电池原材料成本管理系统软件。

（2）工作期限：7 个工作日。

（3）任务要求：

① 具有数据录入、数据修改、数据删除等数据编辑模块。

② 具有查询模块，可以按型号、批号以及××××年××月××日～××××年××月××日查找。

③ 能够按型号、按工序显示成本，并能够显示成品率和成本趋势图。

④ 输入、输出数据分为配料和涂布两类，配料和涂布又分为正极和负极两种数据。

a．配料正极。配料正极原料主要有 $LiCoO_2$、AB、BP2000、PVDF、NMP 等，配料正极输入参数如下：

- 单价 X（单位为元/Kg），数量 Y（单位为 Kg），工艺百分比 T 等。只有材料 NMP 特殊，其单价 x（单位为元/Kg），数量 y（单位为 Kg）。
- 电池批号、电池型号、工人日工资 A 及相应的人数 B。
- 成品数量 M（单位为 Kg），实际平均增重量 m（单位为 g/片），固形物含量 N（单位为个）。

配料正极输出参数如下：

- 成品率：$S1^{+}=M\times m/\Sigma y$（各组份质量和）
- 本工序成本：$Z1^{+}=(\Sigma(X\times Y)+x\times y+A\times B)/(M\times N/m/1000)$（单位：元/片）
- 累计成本：$W=Z1^{+}$（单位：元/片）
- 废品损失：$Q1^{+}=\Sigma((Y-M\times N\times T)\times X)+(y-M\times(1-N))\times x$（单位：元）

累计废品损失：$Q=Q1^{+}$

b．配料负极。配料负极原料主要有 MAG、SBR、CMC；LON25、SBR、CMC；MGP、AB、PVDF、VGCF、CS、NMP 等，配料负极输入参数、输出参数的计算方法与正极相同，成品率用 $S1^{-}$表示、本工序成本用 $Z1^{-}$表示、累计成本用 W 表示、废品损失用 $Q1^{-}$表示、累计废品损失用 Q 表示。

c．涂布正极。涂布正极原料为箔，输入参数如下：

● 箔的单价 C（单位为元/Kg），重量 D（单位为 Kg），工艺百分比 T（同配料的工艺百分比 T）。

● 片子的箔重 G（单位为 g），浆料数量（同配料的成品数量 M）（单位为 Kg），固形物含量 N（同配料的固形物含量 N），实际平均增重量 m（单位为 g），成品片数 $K1^{+}$。

● 批号，型号，日工资 A 及相应的人数 B。

涂布正极输出参数如下：

● 成品率：$S2^{+}=K1^{+}\times(m/1000)/(M\times N)$

● 本工序成本：$Z2^{+}=((\sum((M\times N-K1+\times m/1000)\times T\times X)+((M\times N-K1^{+}\times m/1000)/N)\times(1-N)\times x)+(C\times D+A\times B))/K1^{+}$（单位为元/片）

注：X 为配料中的各物质单价，x 为配料中 NMP 的单价。

累计成本：$W=Z1^{+}+Z2^{+}$（单位为元/片）

● 废品损失：$Q2^{+}=(\sum((M\times N-K1^{+}\times m/1000)\times T\times X)+((M\times N-K1^{+}\times m/1000)/N)\times(1-N)\times x+(D-K1^{+}\times G/1000)\times C$（单位为元）

累计废品损失：$Q=Q1^{+}+Q2^{+}$

d．涂布负极说明。涂布负极原料为箔，输入参数与正极相同，负极输出参数：

● 成品率：$S1^{-}=K1^{-}\times(m/1000)/(M\times N)$

● 本工序成本：$Z2^{-}=((\sum((M\times N-K1^{+}\times m/1000)\times T\times X)+((M\times N-K1^{+}\times m/1000)/N)\times(1-N)\times x)+(C\times D+A\times B))/K1^{-}$（单位为元/片）

注：X 为配料中的各物质单价，x 为配料中 NMP 的单价。

● 累计成本：$W=Z1^{-}+Z2^{-}$（单位为元/片）

● 废品损失：$Q2^{-}=(\sum((M\times N-K1^{+}\times m/1000)\times T\times X)+((M\times N-K1^{+}\times m/1000)/\mathrm{N})\times(1-N)\times x+(D-K1^{+}\times G/1000)\times C$（单位为元）

● 累计废品损失：$Q=Q1^{-}+Q2^{-}$

6.1.2 计划书

1．工作内容

（1）分析项目要完成的功能，确定所使用的软件开发工具和开发系统环境，安装好所需工具软件和系统环境，备好所需资料（约需要时间：1/2 个工作日）。

（2）分析项目所需记录的数据，确定数据结构，确定采用的数据库管理系统，创建数据库（约需要时间：1/2 个工作日）。

（3）项目需求分析，确立开发方案，进行软件的概念分析、功能结构分析、逻辑设计、界面的初步设计等（约需要时间：2 个工作日）。

（4）软件的物理设计，模块功能设计，代码的初步实施（约需要时间：2 个工作日）。

（5）软件的代码实施，代码的功能测试，各模块的组装调试，软件的整体调试，软件注释文档的完善（约需要时间：1 个工作日）。

（6）创建软件系统的安装文件，发布软件的测试版本，并与用户完成软件的整体测试与功能完善（约需要时间：1 个工作日）。

2．项目分析

（1）本系统只是简单的数据记录、查询，虽然数据量比较大，但是选用 Access 数据库足可满足存储条件。

（2）本软件有登录界面，分为管理员和用户两个级别进行登录。

（3）登录系统后，分为管理员和用户两种模式。

① 管理员模式分为管理用户、数据的录入和删除、查询数据、成本、图表等模块，另外添加版本信息模块，便于开发人员管理并升级软件。主界面使用控件 MainMenu 编辑菜单，方便实现主体控制。

② 用户模式只包含数据的录入、查询数据、成本、图表、版本信息等模块。主界面也是使用控件 MainMenu 编辑菜单。

（4）管理员可以方便地添加、修改、删除用户信息，并且亦可进行取消操作。用户信息使用 Grid 控件进行显示，使用 TextBox 控件和 Label 控件组合用于信息的输入。由于需要控件获得唯一标识码来删除信息，可以使用 ReadOnly 设为 False 的 TextBox 控件。

（5）数据录入模块包括计算成本、清除数据、保存数据、导入数据、修改数据等功能。

① 数据包括配料和涂布，采用 Panel 和 GroupBox 控件进行分区，使用 TabControl 控件进行统一管理。

② 由于每项信息又有正极和负极的区分，并且相互之间没有联系，通过控件 ComboBox 进行控制，选择正极时显示正极信息，选择负极时显示负极信息。

③ 录入信息大部分采用 TextBox 控件、时间信息采用控件 DateTimePicker、标识信息使用 Label 控件。

④ 负极配料有多种，使用 ComboBox 控件进行选择控制。

⑤ 使用五个按钮，分别控制计算成本、清除数据、保存数据、导入数据、修改数据。

（6）查询数据模块使用两个 Grid 控件直接显示正极数据和负极数据两个表格的内容，查询方式采用 if-else 语句进行判断，可以对数据进行按型号、按批号及按时间查询，使用三个 Button 按钮手动控制数据的查询。

（7）删除数据分为导入数据和删除数据两步，界面与查询界面类似，使用四个 Grid 控件显示输入参数、其他参数、正极成本、负极成本，使用两个 Button 按钮手动控制数据的导入和删除。

（8）显示成本和图表界面使用水晶报表控件 CrystalReportViewer 进行显示。

（9）数据库使用 OleDbConnection 类进行连接，使用 DataSet 保存数据，使用 OleDb DataAdapter 填充 DataSet 和更新数据源。

（10）版本信息使用控件 Label 进行简单标注。

6.2 项目准备

6.2.1 CrystalReportViewer控件

.NET Framework 中使用 CrystalReportViewer 控件将数据绑定到报表并显示报表。要在 Windows 项目中选择一个报表，并使用 CrystalReportViewer 控件来查看该报表，必须将一个

ReportDocument 组件拖放到页面上，并为其配置一个报表，之后在代码隐藏类中将其绑定到 CrystalReportViewer 控件。

Crystal Reports for Visual Studio 2008 已经简化了 CrystalReportViewer 控件的报表选择，将 CrystalReportViewer 控件添加到 Windows 窗体后，可以从“属性”窗口中的 ReportSource 属性中直接选择报表。ReportSource 属性显示一个组合框，单击该组合框将提供一个浏览选项，以及项目内所有嵌入式报表的列表。

Visual Studio 2008 中设有“设计时预览功能”，能够在设计时就预览报表。在 Crystal Reports for Visual Studio.NET 以前的版本中，在将报表以文件目录路径的形式分配给“属性”窗口中的 ReportSource 属性时，Web 项目中的 CrystalReportViewer 控件提供了设计时预览，现在 Windows 项目也拥有此功能。只需在“属性”窗口中将报表分配给 ReportSource 属性，该报表的预览便会在设计时显示在 Windows 窗体中。

6.2.2 类

1．BindingManagerBase类

BindingManagerBase 类管理绑定到相同数据源和数据成员的所有 Binding 对象，该类为抽象类。在编程时须在程序的模块中手动声明，代码如下：

```
private System.Windows.Forms.BindingManagerBase myBindingManagerBase;
```

使用 BindingManagerBase，可以对 Windows 窗体上绑定到相同数据源的数据绑定控件进行同步。例如，假定某窗体包含两个 TextBox 控件，绑定到相同数据源的不同列（数据源可以是包含客户名称的 DataTable，而列可能包含名字和姓氏）。这两个控件必须同步以便一起显示同一客户的正确姓名，可以设置 Position 属性来指定控件所指向的 DataTable 中的行。若要确定列表中存在的行数，可使用 Count 属性。

2．BindingContext类

BindingContext 类管理从 Control 类继承的任意对象的 BindingManagerBase 对象集合。每个 Windows 窗体至少有一个 BindingContext 对象，此对象管理该窗体的 BindingManagerBase 对象。使用 BindingContext 创建或返回 BindingManagerBase，最常见的是使用 Form 类的 BindingContext 来返回窗体上数据绑定控件的 BindingManagerBase 对象。如果使用容器控件（如 GroupBox、Panel 或 TabControl）来包含数据绑定控件，则可以仅为该容器控件及其内部控件创建一个 BindingContext，这使得窗体的每一部分均可由 BindingManagerBase 对象管理。若要返回特定的 BindingManagerBase 对象，必须将下列参数之一传递给 Item 属性：

（1）如果所需的 BindingManagerBase 不需要导航路径，则仅需要传递数据源。例如，如果 BindingManagerBase 管理一组 Binding 对象，这些对象使用 ArrayList 或 DataTable 作为 DataSource，则不需要导航路径。

（2）数据源和导航路径。当 BindingManagerBase 管理一组 Binding 对象，数据源包含多个对象时，需要导航路径（设置为 Item 属性的 dataMember 参数）。例如，一个 DataSet 可以包含由 DataRelation 对象链接的多个 DataTable 对象，在这种情况下，需要导航路径启用 BindingContext 来返回正确的 BindingManagerBase。

6.2.3 switch语句

switch 语句是一个多分支控制语句，通过将控制传递给其体内的一个 case 语句来处理多个选择。其一般形式如下：

```
switch（表达式）
{
    case 常量表达式 1：
        语句 1；
        break；
        case 常量表达式 1：
        语句 1；
        break；
        …
    case 常量表达式 n：
      语句 n；
      break；
    [default:
      语句 n+1；
      break；]
}
```

说明：首先计算 switch 后面的表达式值；如果表达式的值等于“常量表达式 1”的值，则执行语句 1，然后通过 break 语句退出 switch 结构，执行位于整个 switch 结构后面的语句；如果表达式的值不等于“常量表达式 1”的值，则判断表达式的值是否等于“常量表达式 2”的值，依此类推，直到最后一个语句。

如果 switch 后的表达式与任何一个 case 后的常量表达式的值都不相等，若有 default 语句，则执行 default 后的语句 *n*+1，然后执行位于整个 switch 结构下面的语句；若无 default 语句，则退出 switch 结构，执行位于整个 switch 结构下面的语句。

【例 6-1】 使用 switch 语句编写咖啡计价器，咖啡每小杯 25（元），续杯 25（元），大杯 50（元）。

```
using System;
class SwitchTest
{
    public static void Main()
    {
        Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch(n)
        {
            case 1:
                cost += 25;
                break;
            case 2:
```

```
                    cost += 25;
                    goto case 1;
                case 3:
                    cost += 50;
                    goto case 1;
                default:
                    Console.WriteLine("Invalid selection. Please select 1, 2, or 3.");
                    break;
            }
            if (cost != 0)
                Console.WriteLine("Please insert {0} cents.", cost);
            Console.WriteLine("Thank you for your business.");
        }
    }
```

输入：2

示例输出：

```
Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

在前面的示例中，整型变量 *n* 用于 switch 语句。注意还可以直接使用字符串变量 *s*，在这种情况下，可以按下列方式使用 switch 语句：

```
switch(s)
{
    case "1":
        ...
    case "2":
        ...
}
```

虽然从一个 case 标签到另一个 case 标签的贯穿不受支持，但可以堆栈 case 标签，例如：

```
case 0:
case 1:
```

6.3 项目开发

在 Visual Studio 2008 中新建一个项目，命名为“锂电池原材料成本管理系统”，将文件保存在 D 盘中。通过项目分析，本软件窗体分为主窗体、用户管理窗体、录入数据窗体、删除数据窗体、查询数据窗体、成本窗体、图表窗体和版本窗体。

6.3.1 报表设计

（1）右键单击“解决方案浏览器”，在弹出的菜单中选择“添加”→“添加新项”→“Crystal

Report”。

（2）在“Crystal Report 库”中选择“作为空白报表”单选按钮，最后单击【确定】按钮，将弹出水晶报表设计器。

（3）右键单击报表中的“详细资料区”，选择“数据库”→“添加/删除数据库...”。

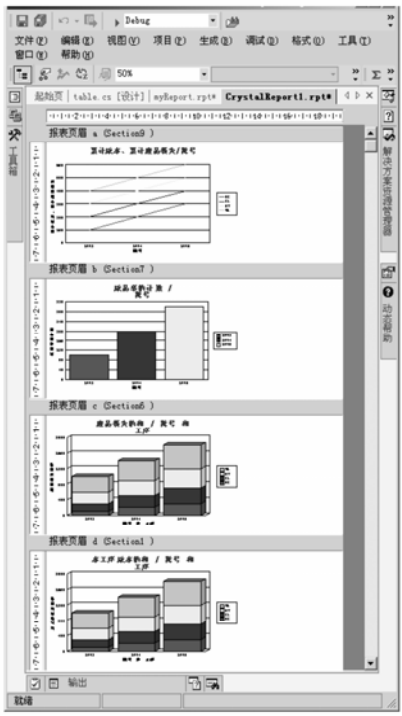
（4）在弹出的“数据库专家”中，展开“OLE DB（ADO）”选项，此时会弹出另外一个“OLE DB（ADO）”窗口。

（5）在“OLE DB (ADO)”弹出窗口中，选择“Microsoft Jet 4.0 OLE DB Provider”，然后单击【下一步】按钮。

（6）指定连接的信息，数据库名称为“D:\Cost\cost.mdb”，数据类型为“Access”，其他使用默认值。单击【下一步】按钮，最后单击【完成】按钮。这时就能在“数据库专家”窗口中显示选择的数据库。

（7）展开“Cost”数据库，展开“表”，选择表“outData”、“outData1”并将其加到“选定的表”区中，单击【确定】按钮。在“字段资源浏览器”的左边“数据库字段”区中显示选择的表，以及表中的字段。拖放需要的字段进入报表的“详细资料”区。字段名将会自动出现在“页眉”区。如果想修改头部文字，则可以右键单击“页眉”区中的文字，选择“编辑文本对象”选项并进行编辑。编辑完成后保存，这样就有了一个水晶报表文件。如图 6.1（a）和图 6.1（b）所示。

（a）报表“MyReport”



（b）报表“CrystalReport1”

图 6.1　水晶报表文件

6.3.2 数据库操作

1．数据库、表设计

本系统所需数据库为 Access 数据库，在“D:\锂电池原材料成本管理系统”目录下创建名为 cost.mdb 的数据库，创建表，如图 6.2 所示。

图 6.2　数据库及表结构

2．连接数据库

（1）选择“工具箱”→“数据”→“OleDbConnection”。

（2）在界面的任意位置单击，界面下方就会出现一个名为“OleDbConnection1”的连接，左键单击或右键单击选择“属性”。

（3）在 ConnectionString 的下拉菜单中选择“新建连接”，打开“数据库链接属性”选项卡。

（4）在“提供程序”中选择“Microsoft Jet 4.0 OLE DB Provider”，单击【下一步】按钮，打开“连接”项，单击“选择或输入数据库名称”后的按钮，选择“cost.mdb”，单击【确定】按钮。

除上述方法，还可以用代码实现。工业上一般应用这种方法，因为其比较直观、易操作。

6.3.3 登录窗体

1．窗体界面设计

当系统启动时，首先在屏幕中出现一个欢迎界面，需要登录才可以使用该系统。

（1）设计窗体外观。

① 新建项目时自动建立的窗体名为 Form1，在“设计”的 name 项中修改为“login”。

② 单击“布局”下 StartPosition 后的下拉菜单，选择 CenterScreen（窗体第一次出现在计算机屏幕的正中央）。

③ 单击“外观”下的 BackgroundImage 后的按钮，选择背景图片。

④ 在 Text 后添加“欢迎登录锂电池原材料成本管理系统”。

（2）添加用户和密码。将“工具箱”中“Windows 窗体”中的 Label、TextBox、Button 拖曳到窗体中，建立如图 6.3 所示的设计界面。

此界面功能很明确，就是用来用户登录的，用户分为两个级别：管理员和普通用户。管理员登录后可以管理用户，所以管理员和普通用户的界面会有点小差别。

图 6.3　登录界面

2．窗体代码设计

（1）【取消】按钮代码。左键双击【取消】按钮，添加如下代码：

```
        private void btnCancel_Click(object sender, System.EventArgs e)
        {
            this.textUserName.Text="";
            this.textUserPassword.Text="";
        }
```

（2）【确定】按钮代码。左键双击【确定】按钮，添加如下代码：

```
        private void btnOk_Click(object sender, System.EventArgs e)
        {
            if ( textUserName.Text != "" && textUserPassword.Text != "")
            {
            if ( textUserName.Text != "管理员" && textUserPassword.Text != "123456")
            {
                MessageBox.Show("登录成功","欢迎！",MessageBoxButtons.OK, MessageBoxIcon.
Information);
                this.Visible=false;
                Form main=new MainForm(this.textUserName.Text,"system");
                main.ShowDialog();
                this.Close();
            }
            else
            {
                oleDbDataAdapter1.SelectCommand.CommandText="SELECT * FROM system WHERE
[User] = '" + textUserName.Text + "'" + " AND [Password] = '" + textUserPassword.Text + "'";
                oleDbDataAdapter1.SelectCommand.Connection.Open();
                oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter1.SelectCommand.Connection.Close();
                oleDbDataAdapter1.Fill(dataSet1);
                dataGrid1.SetDataBinding(dataSet1,"system");
                try
                {
                    MessageBox.Show("登录成功","欢迎！",MessageBoxButtons.OK, MessageBoxIcon.
Information);
```

```
                this.Visible=false;
            Form main=new MainForm1(this.textUserName.Text,"system");
                main.ShowDialog();
                this.Close();
            }
            catch
            {
                MessageBox.Show("登录失败","登录",MessageBoxButtons.OK, MessageBoxIcon.
Warning);
            }
        }
        }
        else
        {
            MessageBox.Show ( "请输入用户名和密码！ ", "提示！ " ) ;
        }
    }
```

6.3.4 主窗体

1．窗体界面设计

（1）设计管理员登录窗体外观。

① 新建窗体，在“设计”的 name 项中修改为“MainForm”，单击“布局”下 StartPosition 后的下拉菜单，选择 CenterScreen（窗体第一次出现在计算机屏幕的正中央）。

② 单击“外观”下的 BackgroundImage 后的按钮，选择背景图片。在 Text 后添加“锂电原材料成本管理系统”。

③ 单击“窗口样式”下的 Icon 后的按钮，选择 Icon 图片。

（2）添加管理员登录窗体的菜单项。

① 将“工具箱”中“Windows 窗体”中的 MainMenu 拖曳到窗体中，就会看到一个上边写着“请在此输入”的可输入文本框。

② 设计一级菜单。在“请在此输入”文本框中填写“文件”，在“文件”后边的文本框中输入“管理”，在“管理”后边的文本框中输入“数据”，在“数据”后边的文本框中输入“查询”，在“查询”后边的文本框中输入“帮助”。

③ 设计二级菜单。在“文件”下输入“退出”，将其命名为“menuExit”；在“管理”下输入“用户”，将其命名为“menuUser”；在“数据”下输入“录入”，将其命名为“menuInput”；在“录入”下输入“删除”，将其命名为“menuDelete”；在“查询”下输入“数据”，将其命名为“menuSelect”；在“数据”下输入“成本”，将其命名为“menuCost”；在“成本”下输入“图表”，将其命名为“menuTable”；在“帮助”下输入“版本”，将其命名为“menuAbout”。设计好的主界面如图 6.4 所示。

图 6.4　管理员登录成功的主界面

（3）普通用户登录成功的主窗体。

在“解决方案资源管理器”中右键单击“MainForm.cs”，单击“复制”，再使用键盘快捷键 Ctrl 键与 V 键同时按，“解决方案资源管理器”中就会出现一个“副本 MainForm.cs”，修改名称为“MainForm1.cs”。由于普通用户的主界面不能对用户进行管理，也不能对数据进行录入与删改，只能进行查询，所以普通用户的主界面在设计菜单时，删除“管理”和“数据”项。

2．窗体代码设计

（1）【退出】按钮代码。左键双击【退出】按钮，添加如下代码：

```
private void menuExit_Click(object sender, System.EventArgs e)
{
    Application.Exit();
}
```

（2）“管理”菜单下【用户】按钮代码。左键双击【用户】按钮，添加如下代码：

```
private void menuUser_Click(object sender, System.EventArgs e)
{
    Form userForm=new user();
    userForm.ShowDialog();
}
```

（3）“数据”菜单下【输入】按钮代码。左键双击【输入】按钮，添加如下代码：

```
private void menuInput_Click(object sender, System.EventArgs e)
{
    Form inputForm=new input();
    inputForm.ShowDialog();
}
```

（4）“数据”菜单下【删除】按钮代码。左键双击【删除】按钮，添加如下代码：

```
private void menuDelete_Click(object sender, System.EventArgs e)
{
    Form deleteForm=new delete();
```

```
        deleteForm.ShowDialog();
    }
```

（5）“查询”菜单下【数据】按钮代码。左键双击【数据】按钮，添加如下代码：

```
    private void menuSelect_Click(object sender, System.EventArgs e)
    {
        Form selectForm=new select();
        selectForm.ShowDialog();
    }
```

（6）“查询”菜单下【成本】按钮代码。左键双击【成本】按钮，添加如下代码：

```
    private void menuCost_Click(object sender, System.EventArgs e)
    {
        Form cbForm=new cb();
        cbForm.ShowDialog();
    }
```

（7）“查询”菜单下【图表】按钮代码。左键双击【图表】按钮，添加如下代码：

```
    private void menuTable_Click(object sender, System.EventArgs e)
    {
        Form tableForm=new table();
        tableForm.ShowDialog();
    }
```

（8）“帮助”菜单下【版本】按钮代码。左键双击【版本】按钮，添加如下代码：

```
    private void menuabout_Click(object sender, System.EventArgs e)
    {
        Form aboutForm=new about();
        aboutForm.ShowDialog();
    }
```

6.3.5 用户管理窗体

1．窗体界面设计

（1）新建窗体，在“设计”的 name 项中修改为“user”，修改“外观”下的 Text 为“用户信息”。

（2）添加两个 GroupBox 控件，分别将“外观”下的 Text 改为“用户信息”和“操作”。

（3）在 GroupBox“用户信息”组中添加三个 Label，分别将“外观”的 Text 改为“用户名称”、“密码”及“用户类型”；添加四个“TextBox”，分别将“设计”的 Name 改为“textUser”、“textPassword”、“textType”及“textID”；将“textID”中“行为”下的“Visible”设为“False”，将控件隐藏（这个控件在最后运行的软件中是不可见的）。

（4）在 GroupBox“操作”组中添加四个 Button 控件，分别将“外观”的 Text 改为“添加”、“修改”、“删除”及“取消”。

（5）添加一个 DataGrid，在“外观”的 CaptionText 中输入“用户信息”。

设计好的用户管理界面如图 6.5 所示。

图 6.5　用户管理界面

2．窗体代码设计

（1）【添加】按钮代码。左键双击【添加】按钮，添加如下代码：

```
private void buttonInsert_Click(object sender, System.EventArgs e)
{
    try
    {
        //判断所有字段是否添完，添完则执行，反之弹出提示
        if ( textUser.Text != "" && textPassword.Text != "" && textType.Text!="")
        {
            insert();
            zero();
            conn();
        }
        else
        {
            MessageBox.Show ( "必须填满所有字段值！", "错误！" ) ;
        }
    }
    catch ( Exception ed )
    {
        MessageBox.Show ( "添加数据记录发生 " + ed.ToString ( ) , "错误！" ) ;
        oleDbDataAdapter1.InsertCommand.Connection.Close();
    }
}
```

（2）【修改】按钮代码。左键双击【修改】按钮，添加如下代码：

```
private void buttonOver_Click(object sender, System.EventArgs e)
{
    try
    {
        //判断所有字段是否添完，添完则执行，反之弹出提示
```

```
                if ( textUser.Text != "" && textPassword.Text != "" && textType.Text!="")
                {
                    textID.Text=dataGrid1[dataGrid1.CurrentRowIndex,2].ToString();
                    delete();
                    insert();
                    zero();
                    conn();
                }
                else
                {
                    MessageBox.Show ( "必须填满所有字段值！" , "错误！" ) ;
                }
            }
            catch ( Exception ed )
            {
                MessageBox.Show ( "修改数据记录发生 " + ed.ToString () , "错误！" ) ;
                oleDbDataAdapter1.InsertCommand.Connection.Close();
            }
        }
```

（3）【删除】按钮代码。左键双击【删除】按钮，添加如下代码：

```
        private void buttonDelete_Click(object sender, System.EventArgs e)
        {
            textID.Text=dataGrid1[dataGrid1.CurrentRowIndex,2].ToString();
            DialogResult   r = MessageBox.Show ( "是否删除当前记录！" , "删除当前记录！" , Message
BoxButtons.YesNo , MessageBoxIcon.Question ) ;
            int ss = ( int ) r ;
            if ( ss == 6 ) // 按“确定”按钮
            {
                try
                {
                    delete();
                    zero();
                    conn();
                    zero();
                }
                catch ( Exception ed )
                {
                   MessageBox.Show ( "删除记录错误信息: " + ed.ToString ( ) , "错误！" ) ;
                    oleDbDataAdapter1.DeleteCommand.Connection.Close();
                }
            }
        }
```

（4）【取消】按钮代码。左键双击【取消】按钮，添加如下代码：

```
        private void buttonBack_Click(object sender, System.EventArgs e)
```

```
        {
            zero();
        }
```

（5）“删除数据”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private void delete()
        {
            oleDbDataAdapter1.DeleteCommand.CommandText="DELETE FROM system WHERE [UserID]=
" + textID.Text + "";
            oleDbDataAdapter1.DeleteCommand.Connection.Open();
            oleDbDataAdapter1.DeleteCommand.ExecuteNonQuery();
            oleDbDataAdapter1.DeleteCommand.Connection.Close();
        }
```

（6）“连接数据库”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private void conn()
        {
            myDataSet.Clear();
            oleDbDataAdapter1.Fill(myDataSet);
            dataGrid1.SetDataBinding(myDataSet,"system");
        }
```

（7）“初始化”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private void zero()
        {
            textUser.Text ="";
            textPassword.Text="";
            textType.Text="";
            textID.Text="";
            this.textUser.Focus();
        }
```

（8）“向数据库中添加数据”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private void insert()
        {
            oleDbDataAdapter1.InsertCommand.CommandText="INSERT INTO system([Password], [User],
UserType) VALUES ( '" + textPassword.Text + "' ,'" + textUser.Text + "' ,'"+ textType.Text +"')";
            oleDbDataAdapter1.InsertCommand.Connection.Open();
            oleDbDataAdapter1.InsertCommand.ExecuteNonQuery();
            oleDbDataAdapter1.InsertCommand.Connection.Close();
        }
```

6.3.6　录入数据窗体

1．窗体界面设计

（1）添加窗体“input”，修改“外观”下的 Text 为“录入数据”。

（2）单击“工具箱”中的“Windows 窗体”下的 TabControl，在窗体选择一合适位置放置；右键单击选择“属性”，单击“属性”中“杂项”下的 tabPage 后的按钮，打开“tabPage 集合编辑器”；单击“添加”添加 tabPage1、tabPage2、tabPage3、tabPage4，分别设置“tabPage 属性”中“外观”的“Text”为“配料”、“涂布”。

（3）添加 Label、TextBox、ComboBox、Button、DataGrid、DateTimePicker 控件，如图 6.6（a）、图 6.6（b）所示设计录入数据界面。

2．窗体代码设计

（1）“配料”选项卡【计算成本】按钮代码。左键双击【计算成本】按钮，添加如下代码：

```
private void buttonPlus1_Click(object sender, System.EventArgs e)
{
    DataPlus();
}
```

（a）

（b）

图 6.6　录入数据界面

（2）“配料”选项卡【清除数据】按钮代码。左键双击【清除数据】按钮，添加如下代码：

```
private void buttonClear1_Click(object sender, System.EventArgs e)
{
    ClearData();
}
```

（3）“配料”选项卡【保存数据】按钮代码。左键双击【保存数据】按钮，添加如下代码：

```
private void buttonSave1_Click(object sender, System.EventArgs e)
{
    if (textBox1.Text==""||textBox2.Text=="")
    {
        MessageBox.Show("没有填写型号或批号","保存数据", MessageBoxButtons.OK,Message
BoxIcon.Error);
    }
    else
    {
        try
        {
            CheckAllInput1();
        }
        catch( Exception ed)
        {
        MessageBox.Show ( "保存数据库发生 " + ed.ToString ( ) , "错误！ " ) ;
        }
    }
}
```

（4）“配料”选项卡【导入数据】按钮代码。左键双击【导入数据】按钮，添加如下代码：

```
private void buttonIn1_Click(object sender, System.EventArgs e)
{
    if(textBox1.Text!="" && textBox2.Text!="")
    {
        if(comboBox1.SelectedIndex!=-1)
        {
            switch (comboBox1.SelectedIndex)
            {
                case 0:
                {
                SelectData1(1);
                myBindingManagerBase=this.BindingContext[dataSet2, "step"];
                    try
                    {
                    textBox17.Text=data2[data2.CurrentRowIndex,0].ToString();
                    textBox18.Text=data2[data2.CurrentRowIndex,1].ToString();
                    textBox19.Text=data2[data2.CurrentRowIndex,2].ToString();
                    textBox20.Text=data2[data2.CurrentRowIndex,3].ToString();
```

```
                textBox21.Text=data2[data2.CurrentRowIndex,4].ToString();
                //……数据与文本框一一对应
                }
                catch
                {
                    MessageBox.Show ( "该型号或批号的数据不存在 "  , "导入数据！" ) ;
                }
            }
                break;
            case 1:
            {
            SelectData1(2);
        myBindingManagerBase=this.BindingContext[dataSet2, "step"];
            DataTable dataTable=dataSet2.Tables[0];
            textBox56.Text=dataTable.Rows.Count.ToString();
                try
                {
                    if(textBox56.Text=="3")
                    {
                        comboBox3.SelectedIndex=1;
                        comboBox3.Text="";
                label27.Text=data1[data1.CurrentRowIndex,0].ToString();
            textBox27.Text=data1[data1.CurrentRowIndex,2].ToString();
            textBox33.Text=data1[data1.CurrentRowIndex,1].ToString();
            textBox39.Text=data1[data1.CurrentRowIndex,3].ToString();
                myBindingManagerBase.Position+=1;
                label28.Text=data1[data1.CurrentRowIndex,0].ToString();
            textBox28.Text=data1[data1.CurrentRowIndex,2].ToString();
            textBox34.Text=data1[data1.CurrentRowIndex,1].ToString();
            textBox40.Text=data1[data1.CurrentRowIndex,3].ToString();
                myBindingManagerBase.Position+=1;
                label29.Text=data1[data1.CurrentRowIndex,0].ToString();
            textBox29.Text=data1[data1.CurrentRowIndex,2].ToString();
            textBox35.Text=data1[data1.CurrentRowIndex,1].ToString();
            textBox41.Text=data1[data1.CurrentRowIndex,3].ToString();
                    }
                    else
                    {
                        comboBox3.SelectedIndex=2;
                        comboBox3.Text="";
                        this.label30.Visible=true;
                label27.Text=data1[data1.CurrentRowIndex,0].ToString();
            textBox27.Text=data1[data1.CurrentRowIndex,2].ToString();
            textBox33.Text=data1[data1.CurrentRowIndex,1].ToString();
            textBox39.Text=data1[data1.CurrentRowIndex,3].ToString();
                myBindingManagerBase.Position+=1;
                //……按照 textBox56.Text=="3"导入数据的方法导入 6 组数据
```

```
                    }
                textBox17.Text=data2[data2.CurrentRowIndex,0].ToString();
                textBox18.Text=data2[data2.CurrentRowIndex,1].ToString();
                textBox19.Text=data2[data2.CurrentRowIndex,2].ToString();
                textBox20.Text=data2[data2.CurrentRowIndex,3].ToString();
                textBox21.Text=data2[data2.CurrentRowIndex,4].ToString();
                    }
                    catch
                    {
                        MessageBox.Show ( "该型号或批号的数据不存在 "  , "导入数据！" ) ;
                    }
                }
                    break;
            }
        }
        else
        {
            MessageBox.Show("没有选择正负极","修改数据", MessageBoxButtons.OK,Message
BoxIcon.Error);
        }
    }
    else
    {
        MessageBox.Show("请输入型号和批号","修改数据", MessageBoxButtons.OK,Message
BoxIcon.Error);
    }
}
```

（5）“配料”选项卡代码。左键双击【修改数据】按钮，添加如下代码：

```
private void buttonOver1_Click(object sender, System.EventArgs e)
{
    if (textBox1.Text==""||textBox2.Text=="")
    {
        MessageBox.Show("没有填写型号或批号","保存数据", MessageBoxButtons.OK,Message
BoxIcon.Error);
    }
    else
    {
        try
        {
            CheckAllInput2();
        }
        catch( Exception ed)
        {
            MessageBox.Show ( "保存数据库发生 " + ed.ToString ( ) , "错误！" ) ;
        }
```

```
        }
    }
```

（6）“涂布”选项卡【计算成本】按钮代码。左键双击【计算成本】按钮，添加如下代码：

```
    private void buttonPlus2_Click(object sender, System.EventArgs e)
    {
        DataPlus1();
    }
```

（7）“涂布”选项卡【清除数据】按钮代码。左键双击【清除数据】按钮，添加如下代码：

```
    private void buttonClear2_Click(object sender, System.EventArgs e)
    {
        ClearData();
    }
```

（8）“涂布”选项卡【保存数据】按钮代码。左键双击【保存数据】按钮，添加如下代码：

```
        private void buttonSave2_Click(object sender, System.EventArgs e)
        {
            if (comboBox2.SelectedIndex==-1)
            {
                MessageBox.Show("没有选择正负极","保存数据", MessageBoxButtons.OK,Message
BoxIcon.Error);
            }
            else
            {
                if (textBox44.Text==""||textBox45.Text==""||textBox46.Text==""|| textBox47.Text==""
||text Box48.Text==""||textBox49.Text==""||textBox50.Text=="")
                {
                    MessageBox.Show("数据输入不全","保存数据",MessageBoxButtons.OK,Message
BoxIcon.Error);
                }
                else
                {
                    if (comboBox2.SelectedIndex==0)
                    {
                        dataSet1.Clear();
                        SelectData(3);
                        this.oleDbDataAdapter1.Fill(dataSet1);
                        DataTable dataTable5=dataSet1.Tables[0];
                        if(dataTable5.Rows.Count==0)
                        {
                            dataSet1.Clear();
                            SelectData(1);
                            this.oleDbDataAdapter1.Fill(dataSet1);
                            dataTable5=dataSet1.Tables[0];
                            if(dataTable5.Rows.Count==0)
```

```
                {
                    MessageBox.Show("由于没有保存过配料正极的数据,所以不能保存涂布正极的数据","保存数据",MessageBoxButtons.OK,MessageBoxIcon.Error);
                }
                else
                {
                    DataPlus1();
                    InsertData(3);
                    this.comboBox1.SelectedIndex=-1;
                    this.comboBox2.SelectedIndex=-1;
                    this.comboBox3.SelectedIndex=-1;
                    ClearData();
                    MessageBox.Show("保存数据成功","保存数据",MessageBoxButtons.OK,MessageBoxIcon.Information);
                }
            }
            else
            {
                MessageBox.Show("已经保存过本批号涂布正极的数据","保存数据",MessageBoxButtons.OK,MessageBoxIcon.Error);
            }
        }
        else
        {
            dataSet1.Clear();
            SelectData(4);
            this.oleDbDataAdapter1.Fill(dataSet1);
            DataTable dataTable6=dataSet1.Tables[0];
            if(dataTable6.Rows.Count==0)
            {
                dataSet1.Clear();
                SelectData(2);
                this.oleDbDataAdapter1.Fill(dataSet1);
                dataTable6=dataSet1.Tables[0];
                if(dataTable6.Rows.Count==0)
                {
                    MessageBox.Show("由于没有保存过配料负极的数据,所以不能保存涂布负极的数据","保存数据",MessageBoxButtons.OK,MessageBoxIcon.Error);
                }
                else
                {
                    DataPlus1();
                    InsertData(4);
                    this.comboBox1.SelectedIndex=-1;
                    this.comboBox2.SelectedIndex=-1;
                    this.comboBox3.SelectedIndex=-1;
                    ClearData();
```

```
                                MessageBox.Show("保存数据成功","保存数据",MessageBoxButtons.
                                OK,MessageBoxIcon.Information);
                            }
                        }
                        else
                        {
                            MessageBox.Show("已经保存过本批号涂布负极的数据","保存数据",
MessageBoxButtons.OK,MessageBoxIcon.Error);
                        }
                    }
                }
            }
        }
```

（9）“涂布”选项卡【导入数据】按钮代码。左键双击【导入数据】按钮，添加如下代码：

```
        private void buttonIn2_Click(object sender, System.EventArgs e)
        {
            if(textBox1.Text!="" && textBox2.Text!="")
            {
                if(comboBox2.SelectedIndex!=-1)
                {
                    switch (comboBox2.SelectedIndex)
                    {
                        case 0:
                            SelectData1(3);
                            break;
                        case 1:
                            SelectData1(4);
                            break;
                    }
                    try
                    {
                textBox46.Text=dataGrid2[dataGrid2.CurrentRowIndex,0].ToString();
                textBox47.Text=dataGrid2[dataGrid2.CurrentRowIndex,1].ToString();
                textBox48.Text=dataGrid2[dataGrid2.CurrentRowIndex,2].ToString();
                textBox49.Text=dataGrid2[dataGrid2.CurrentRowIndex,3].ToString();
                textBox50.Text=dataGrid2[dataGrid2.CurrentRowIndex,4].ToString();
                textBox44.Text=dataGrid1[dataGrid1.CurrentRowIndex,1].ToString();
                textBox45.Text=dataGrid1[dataGrid1.CurrentRowIndex,2].ToString();
                    }
                    catch
                    {
                MessageBox.Show ("该型号或批号的数据不存在", "导入数据！");
                    }
                }
                else
```

```
                {
                    MessageBox.Show("没有选择正负极","修改数据", MessageBoxButtons.OK,Message
BoxIcon.Error);
                }
            }
            else
            {
                MessageBox.Show("请输入型号和批号","修改数据", MessageBoxButtons.OK,Message
BoxIcon.Error);
            }
        }
```

（10）“涂布”选项卡【修改数据】按钮代码。左键双击【修改数据】按钮，添加如下代码：

```
        private void buttonOver2_Click(object sender, System.EventArgs e)
        {
            if (comboBox2.SelectedIndex==-1)
            {
                MessageBox.Show("没有选择正负极","修改数据",MessageBoxButtons.OK, MessageBoxIcon.
Error);
            }
            else
            {
                if (textBox44.Text==""||textBox45.Text==""||textBox46.Text==""|| textBox47.Text==""||text
Box48.Text==""||textBox49.Text==""||textBox50.Text=="")
                {
                    MessageBox.Show("数据输入不全","修改数据",MessageBoxButtons.OK, Message
BoxIcon.Error);
                }
                else
                {
                    if (comboBox2.SelectedIndex==0)
                    {
                        DataPlus1();
                        DeleteData(3);
                        InsertData(3);
                        this.comboBox1.SelectedIndex=-1;
                        this.comboBox2.SelectedIndex=-1;
                        this.comboBox3.SelectedIndex=-1;
                        ClearData();
                        MessageBox.Show("修改数据成功","修改数据", MessageBoxButtons.OK,Message
BoxIcon.Information);
                    }
                    else
                    {
                        DataPlus1();
                        DeleteData(4);
```

```
                    InsertData(4);
                    this.comboBox1.SelectedIndex=-1;
                    this.comboBox2.SelectedIndex=-1;
                    this.comboBox3.SelectedIndex=-1;
                    ClearData();
                    MessageBox.Show("修改数据成功","修改数据", MessageBoxButtons.OK,MessageBoxIcon.Information);
                }
            }
        }
    }
```

（11）初始化代码。左键双击窗体中没有控件的部分，添加如下代码：

```
    private void input_Load(object sender, System.EventArgs e)
    {
        panel1.Visible=false;
        this.label27.Text="MGP";
        this.label28.Text="AB";
        this.label29.Text="PVDF";
        this.label30.Text="VGCF";
        this.label31.Text="CS";
        this.label32.Text="NMP";
    }
```

（12）tabControl1 初始化代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
    private void tabControl1_SelectedIndexChanged(object sender, System.EventArgs e)
    {
        this.tabPage2.Text="配料";
        this.tabPage1.Text="涂布";
        comboBox3.SelectedIndex=-1;
        comboBox2.SelectedIndex=-1;
        comboBox1.SelectedIndex=-1;
    }
```

（13）“按工序查找数据”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
    private void SelectData(int i)
    {
        if(i==1)
        {
            this.oleDbDataAdapter1.SelectCommand.CommandText="SELECT * FROM [inData] WHERE [型号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" +" AND [工序] = '1'"+" ORDER BY [工序] ASC";
            this.oleDbDataAdapter1.SelectCommand.Connection.Open();
            this.oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
```

```
                this.oleDbDataAdapter1.SelectCommand.Connection.Close();
                oleDbDataAdapter4.SelectCommand.CommandText="SELECT [固形物含量],[实际平均增重量],[成品数量],[日工资],[人数] FROM [step1] WHERE [型号] ='" + textBox1.Text + "' and [批号] ='" + textBox2.Text + "' and [工序] = '"+ 1 +"'";
                oleDbDataAdapter4.SelectCommand.Connection.Open();
                oleDbDataAdapter4.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter4.SelectCommand.Connection.Close();
                dataSet3 = new DataSet("step1");
                oleDbDataAdapter4.Fill(dataSet3,"step1");
                data2.SetDataBinding(dataSet3,"step1");
                this.oleDbDataAdapter3.SelectCommand.CommandText="SELECT [名称],[单价],[数量],[工艺百分比] FROM [step] WHERE [型号] = '" + textBox1.Text + "'" + " AND [批号] = '" + text Box2.Text + " '" + " AND [工序] = '1' ";
                this.oleDbDataAdapter3.SelectCommand.Connection.Open();
                this.oleDbDataAdapter3.SelectCommand.ExecuteNonQuery();
                this.oleDbDataAdapter3.SelectCommand.Connection.Close();
                this.oleDbDataAdapter3.Fill(dataSet2);
                dataSet2 = new DataSet("step");
                oleDbDataAdapter3.Fill(dataSet2,"step");
                data1.SetDataBinding(dataSet2,"step");
            }
        //工序 2（i=2）：配料（负极）、工序 3（i=3）：涂布（正极）、工序 4（i=4）：涂布（负极）的查找代码与工序 1（i=1）：配料（正极）的代码相似，稍做修改即可
        }
```

（14）“添加数据”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private void InsertData(int gx)
        {
            if(gx==1)
            {
                this.oleDbDataAdapter2.InsertCommand.CommandText="INSERT INTO [inData]( [型号],[批号],[工序],[录入日期]) VALUES ( '" + textBox1.Text + " ' ,'" + textBox2.Text + " ' ,'"+ "1" + " ', '" + dateTimePicker1.Value.Date + "')";
                this.oleDbDataAdapter2.InsertCommand.Connection.Open();
                this.oleDbDataAdapter2.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter2.InsertCommand.Connection.Close();
                this.oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO [step] ( [名称],[型号],[批号],[单价],[数量],[工艺百分比],[工序]) VALUES ( '" + label3.Text + "','" + textBox1.Text + "' , '" + textBox2.Text + "', '" + textBox8. Text + "', '" + textBox3.Text + "', '" + textBox13.Text + "','" + "1" + "')";
                this.oleDbDataAdapter3.InsertCommand.Connection.Open();
                this.oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter3.InsertCommand.Connection.Close();
                this.oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO [step] ( [名称],[型号],[批号],[单价],[数量],[工艺百分比],[工序]) VALUES ( '" + label4.Text + "' ,'" + textBox. Text + "' ,'" + textBox2.Text + "', '" + textBox9. Text + "', '" + textBox4.Text + "' ,'" + text Box14.Text + "','" + "1" + "')";
```

```
                this.oleDbDataAdapter3.InsertCommand.Connection.Open();
                this.oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter3.InsertCommand.Connection.Close();
                this.oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO [step]( [名称],[型
号],[批号],[单价],[数量],[工艺百分比],[工序]) VALUES ( '" + label5.Text + "' ,'" + textBox. Text + "' ,'" +
textBox2.Text + "', '" + textBox10.Text + "', '" + textBox5.Text + "' ,'" + text Box15.Text + "','" + "1" + "')";
                this.oleDbDataAdapter3.InsertCommand.Connection.Open();
                this.oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter3.InsertCommand.Connection.Close();
                this.oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO [step]( [名称],[型
号],[批号],[单价],[数量],[工艺百分比],[工序]) VALUES ( '" + label6.Text + "' ,'" + text Box1.Text + "' ,
'" + textBox2.Text + "', '" + textBox11.Text + "', '" + textBox6.Text + "' ,'" + textBox16.Text + "','" + "1" + "')";
                this.oleDbDataAdapter3.InsertCommand.Connection.Open();
                this.oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter3.InsertCommand.Connection.Close();
                this.oleDbDataAdapter3.InsertCommand.CommandText="INSERT INTO [step]( [名称],[型
号],[批号],[单价],[数量],[工艺百分比],[工序]) VALUES ( '" + label7.Text + " ' ,'" + textBox. Text + "' , '" +
textBox2.Text + "','" + textBox12.Text + "','" + textBox7.Text + "' ,'"+ "0" +"', '" + "1" + "')";
                this.oleDbDataAdapter3.InsertCommand.Connection.Open();
                this.oleDbDataAdapter3.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter3.InsertCommand.Connection.Close();
                this.oleDbDataAdapter4.InsertCommand.CommandText="INSERT INTO [step1]( [型号],[批
号],[固形物含量],[实际平均增重量],[成品数量],[日工资],[人数],[工序]) VALUES ('"+text Box1.Text + "' ,
'" + textBox2.Text + "' , '" + textBox17.Text + "', '" + textBox18.Text + "','"+ textBox19.Text + "', '"+ textBox20.
Text + "', '" + textBox21.Text + "', '" + "1" + "')";
                this.oleDbDataAdapter4.InsertCommand.Connection.Open();
                this.oleDbDataAdapter4.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter4.InsertCommand.Connection.Close();
                this.oleDbDataAdapter4.InsertCommand.CommandText="INSERT INTO [outData]( [型号],
 [批号],[工序],[成品率],[本工序成本],[累计成本],[废品损失],[累计废品损失],[录入日期]) VALUES ( '" +
textBox1.Text + "' ,'" + textBox2.Text + "' ,'"+ "配料(正极)" + "' ,'" + textBox22.Text + "' ,'" + textBox23.Text + "' ,'"
+ textBox24.Text + "' ,'"+ textBox25.Text+"','"+textBox26.Text+ "' ,'"+ dateTimePicker1.Value.Date + "')";
                this.oleDbDataAdapter4.InsertCommand.Connection.Open();
                this.oleDbDataAdapter4.InsertCommand.ExecuteNonQuery();
                this.oleDbDataAdapter4.InsertCommand.Connection.Close();
            }
        //工序 2（i=2）：配料（负极）、工序 3（i=3）：涂布（正极）、工序 4（i=4）：涂布（负极）
的添加数据代码与工序 1（i=1）：配料（正极）的代码相似，稍做修改即可
        }
```

（15）“删除数据”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private void DeleteData(int i)
        {
            if(i==1)
            {
```

```
                this.oleDbDataAdapter2.DeleteCommand.CommandText="DELETE FROM [inData] WHERE
[型号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" +" AND [工序] = '1'";
                this.oleDbDataAdapter2.DeleteCommand.Connection.Open();
                this.oleDbDataAdapter2.DeleteCommand.ExecuteNonQuery();
                this.oleDbDataAdapter2.DeleteCommand.Connection.Close();
                this.oleDbDataAdapter3.DeleteCommand.CommandText="DELETE FROM [step] WHERE
[型号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" +" AND [工序] = '1'";
                this.oleDbDataAdapter3.DeleteCommand.Connection.Open();
                this.oleDbDataAdapter3.DeleteCommand.ExecuteNonQuery();
                this.oleDbDataAdapter3.DeleteCommand.Connection.Close();
                this.oleDbDataAdapter4.DeleteCommand.CommandText="DELETE FROM [step1] WHERE
[型号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" +" AND [工序] = '1'";
                this.oleDbDataAdapter4.DeleteCommand.Connection.Open();
                this.oleDbDataAdapter4.DeleteCommand.ExecuteNonQuery();
                this.oleDbDataAdapter4.DeleteCommand.Connection.Close();
                this.oleDbDataAdapter4.DeleteCommand.CommandText="DELETE FROM [outData] WHERE
[型号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" +" AND [工序] = '配料(正极)'";
                this.oleDbDataAdapter4.DeleteCommand.Connection.Open();
                this.oleDbDataAdapter4.DeleteCommand.ExecuteNonQuery();
                this.oleDbDataAdapter4.DeleteCommand.Connection.Close();
            }
        //工序 2（i=2）：配料（负极）、工序 3（i=3）：涂布（正极）、工序 4（i=4）：涂布（负极）
的删除代码与工序 1（i=1）：配料（正极）的代码相似，稍做修改即可
        }
```

（16）配料（正极）“成品率计算”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private double CalculateRate1()
        {
            double sum1,sum2;
            sum1=Convert.ToDouble(textBox17.Text)*Convert.ToDouble(textBox19.Text);
            sum2=Convert.ToDouble(textBox3.Text)+Convert.ToDouble(textBox4.Text)+Convert.ToDouble
(textBox5.Text)+Convert.ToDouble(textBox6.Text);
            return(sum1/sum2);
        }
```

配料（负极）的“成品率计算”与配料（正极）相似，不同的是要考虑两种配料方案。

（17）涂布“成品率计算”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
        private double CalculateRate3()
        {
            double sum1,sum2,sum3;
            sum3=Convert.ToDouble(dataGrid2[0,2].ToString()) * Convert.ToDouble(dataGrid2[0,0].ToString());
            sum1=Convert.ToDouble(textBox48.Text);
            sum2=Convert.ToDouble(textBox47.Text)/1000;
            return(sum1*sum2/sum3);
```

```
}
```

（18）配料（正极）“本工序成本计算”代码。右键单击界面，选择“查看代码”，打开代码编辑窗口，输入如下代码：

```
private double Cost1()
{
    double sum1,sum2;
    sum1=Convert.ToDouble(textBox3.Text)*Convert.ToDouble(textBox8.Text) + Convert.ToDouble
(textBox4.Text)*Convert.ToDouble(textBox9.Text) + Convert.ToDouble(textBox5.Text)*Convert. ToDouble
(textBox10.Text) + Convert.ToDouble(textBox6.Text)*Convert.ToDouble(textBox11. Text) + Convert.ToDouble
(textBox7.Text)*Convert.ToDouble(textBox12.Text) + Convert. ToDouble (textBox20.Text)*Convert.ToDouble
(textBox21.Text);
    sum2=Convert.ToDouble(textBox19.Text)* Convert.ToDouble(textBox17.Text)/(Convert.ToDouble
    (textBox18.Text)/ 1000);
    return(sum1/sum2);
}
```

不同工序“本工序成本计算”的代码都是相似的，不同的是配料（负极）要考虑两种配料方案。

（19）“累计成本计算”代码。配料为第一道工序，本工序成本就是累计成本，而涂布是第二道工序，累计成本需要加上配料的本工序成本，由于配料的本工序成本是由若干项输入数据计算而得，所以我们在计算的过程中要调用输入数据。最简单的方式是使用 dataGrid 控件保存数据及调用数据。其中涂布正极的累计成本代码如下：

```
private double CalculateCost3()
{
    double sum1,sum2,sum3,sum4,x=0;
    int i;
    for(i=0;i<=4;i++)
    {
        if(dataGrid1[i,0].ToString().Trim()=="NMP")
        {
            x=Convert.ToDouble(dataGrid1[i,2].ToString());
        }
    }
    sum1=(Convert.ToDouble(dataGrid2[0,2].ToString())*Convert.ToDouble(dataGrid2[0,0].ToString()))–
 (Convert.ToDouble(textBox48.Text)*(Convert.ToDouble(textBox47.Text)/1000));
    sum2=Convert.ToDouble(dataGrid1[0,2].ToString())*Convert.ToDouble(dataGrid1[0,3].ToString())
+Convert.ToDouble(dataGrid1[1,2].ToString())*Convert.ToDouble(dataGrid1[1,3].ToString())+Convert.ToDouble
(dataGrid1[2,2].ToString())*Convert.ToDouble(dataGrid1[2,3].ToString())+Convert.ToDouble(dataGrid1[3,2].
ToString())*Convert.ToDouble(dataGrid1[3,3].ToString())+Convert.ToDouble(dataGrid1[4,2].ToString())*
Convert.ToDouble(dataGrid1[4,3].ToString());
    sum3=sum1/Convert.ToDouble(dataGrid2[0,0].ToString())*(1-Convert.ToDouble(dataGrid2[0,0].
ToString()))*x;
    sum4=(Convert.ToDouble(textBox44.Text)*Convert.ToDouble(textBox45.Text)+Convert.ToDouble
(textBox49.Text)*Convert.ToDouble(textBox50.Text));
    return((sum1*sum2+sum3+sum4)/Convert.ToDouble(textBox48.Text));
```

```
    }
```

（20）“废品损失计算”代码。以“配料（正极）”为例，废品损失计算的代码如下：

```
    private double Waste1()
    {
        double sum1,sum2,sum3,sum4,sum5;
        sum1=(Convert.ToDouble(textBox3.Text)-Convert.ToDouble(textBox19.Text)*Convert.ToDouble
(textBox17.Text)*Convert.ToDouble(textBox13.Text))*Convert.ToDouble(textBox8.Text);
        sum2=(Convert.ToDouble(textBox4.Text)-Convert.ToDouble(textBox19.Text)*Convert.ToDouble
(textBox17.Text)*Convert.ToDouble(textBox14.Text))*Convert.ToDouble(textBox9.Text);
        sum3=(Convert.ToDouble(textBox5.Text)-Convert.ToDouble(textBox19.Text)*Convert.ToDouble
(textBox17.Text)*Convert.ToDouble(textBox15.Text))*Convert.ToDouble(textBox10.Text);
        sum4=(Convert.ToDouble(textBox6.Text)-Convert.ToDouble(textBox19.Text)*Convert.ToDouble
(textBox17.Text)*Convert.ToDouble(textBox16.Text))*Convert.ToDouble(textBox11.Text);
        sum5=(Convert.ToDouble(textBox7.Text)-Convert.ToDouble(textBox19.Text)*(1-Convert.ToDouble
(textBox17.Text)))*Convert.ToDouble(textBox12.Text);
        return(sum1+sum2+sum3+sum4+sum5);
    }
```

（21）“累计损失计算”代码。配料为第一道工序，废品损失就是累计损失，而涂布是第二道工序，累计成本需要加上配料的废品损失，由于配料的废品损失是由若干项输入数据计算而得，所以在计算的过程中要调用输入数据，可使用 dataGrid 控件保存数据及调用数据。其中涂布正极的累计损失计算代码如下：

```
    private double CalculateWaste3()
    {
        double sum1,sum2,sum3,sum4,x=0;
        int i;
        for(i=0;i<=4;i++)
        {
            if(dataGrid1[i,0].ToString().Trim()=="NMP")
            {
                x=Convert.ToDouble(dataGrid1[i,2].ToString());
            }
        }
        sum1=(Convert.ToDouble(dataGrid2[0,2].ToString())*Convert.ToDouble(dataGrid2[0,0].ToString()))–
(Convert.ToDouble(textBox48.Text)*(Convert.ToDouble(textBox47.Text)/1000));
        sum2=Convert.ToDouble(dataGrid1[0,2].ToString())*Convert.ToDouble(dataGrid1[0,3].ToString())
+Convert.ToDouble(dataGrid1[1,2].ToString())*Convert.ToDouble(dataGrid1[1,3].ToString())+Convert.ToDouble
(dataGrid1[2,2].ToString())*Convert.ToDouble(dataGrid1[2,3].ToString())+Convert.ToDouble(dataGrid1[3,2].
ToString())*Convert.ToDouble(dataGrid1[3,3].ToString())+Convert.ToDouble(dataGrid1[4,2].ToString())*
Convert.ToDouble(dataGrid1[4,3].ToString());
        sum3=sum1/Convert.ToDouble(dataGrid2[0,0].ToString())*(1-Convert.ToDouble(dataGrid2[0,0].
ToString()))*x;
        sum4=(Convert.ToDouble(textBox44.Text)-Convert.ToDouble(textBox48.Text)*Convert.ToDouble
(textBox46.Text)/1000)*Convert.ToDouble(textBox45.Text);
        return(sum1*sum2+sum3+sum4);
```

```
        }
```

（22）纠错代码。当输入非数字时提示“输入非法字符”，并清空该输入框。添加代码如下：

```
        private void textBox3_KeyDown(object sender, System.Windows.Forms.KeyEventArgs e)
        {
            if (CheckInputCharater(Convert.ToInt32(e.KeyCode))==2)
            {
                MessageBox.Show("输入非法字符","输入出错",MessageBoxButtons.OK, MessageBoxIcon.
Error);
                textBox3.Text="";
            }
        }
```

6.3.7　删除数据窗体

1．窗体界面设计

（1）添加窗体“delete”，修改“外观”下的“Text”为“录入数据”。

（2）添加两个 Label，分别将“外观”下的“Text”改为“型号”、“批号”。添加两个 TextBox 控件用于输入相关信息。

（3）添加两个 Button，分别将“外观”下的“Text”改为“导入数据”、“删除数据”。

（4）添加四个 DataGrid，分别在“外观”下的 CaptionText 中输入“输入参数”、“其他参数”、“正极成本”、“负极成本”。

完成的删除数据界面如图 6.7 所示。

图 6.7　删除数据界面

2．窗体代码设计

（1）【导入数据】按钮代码。添加代码如下：

```
        private void button1_Click(object sender, System.EventArgs e)
        {
            oleDbDataAdapter2.SelectCommand.CommandText="SELECT * FROM outData1 WHERE  [型
号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" ;
            oleDbDataAdapter2.SelectCommand.Connection.Open();
```

```
                oleDbDataAdapter2.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter2.SelectCommand.Connection.Close();
                dataSet1 = new DataSet("outData1");
                oleDbDataAdapter2.Fill(dataSet1,"outData1");
                dataGrid1.SetDataBinding(dataSet1,"outData1");
                oleDbDataAdapter2.SelectCommand.CommandText="SELECT * FROM outData WHERE[型
号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" ;
                oleDbDataAdapter2.SelectCommand.Connection.Open();
                oleDbDataAdapter2.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter2.SelectCommand.Connection.Close();
                dataSet1 = new DataSet("outData");
                oleDbDataAdapter2.Fill(dataSet1,"outData");
                dataGrid2.SetDataBinding(dataSet1,"outData");
                oleDbDataAdapter3.SelectCommand.CommandText="SELECT * FROM step WHERE[型号] =
'" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'";
                oleDbDataAdapter3.SelectCommand.Connection.Open();
                oleDbDataAdapter3.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter3.SelectCommand.Connection.Close();
                dataSet1 = new DataSet("step");
                oleDbDataAdapter3.Fill(dataSet1,"step");
                dataGrid3.SetDataBinding(dataSet1,"step");
                oleDbDataAdapter4.SelectCommand.CommandText="SELECT * FROM step1 WHERE [型号] =
'" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'";
                oleDbDataAdapter4.SelectCommand.Connection.Open();
                oleDbDataAdapter4.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter4.SelectCommand.Connection.Close();
                dataSet1 = new DataSet("step1");
                oleDbDataAdapter4.Fill(dataSet1,"step1");
                dataGrid4.SetDataBinding(dataSet1,"step1");
            }
```

（2）【删除数据】按钮代码。添加代码如下：

```
            private void button2_Click(object sender, System.EventArgs e)
            {
                if(textBox1.Text!="" && textBox1.Text!="")
                {
                    oleDbDataAdapter1.DeleteCommand.CommandText="DELETE FROM inData WHERE [型
号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" ;
                    oleDbDataAdapter1.DeleteCommand.Connection.Open();
                    oleDbDataAdapter1.DeleteCommand.ExecuteNonQuery();
                    oleDbDataAdapter1.DeleteCommand.Connection.Close();
                    oleDbDataAdapter2.DeleteCommand.CommandText="DELETE FROM outData WHERE [型
号] = '" + textBox1.Text + "'" + " AND [批号] = '" + textBox2.Text + "'" ;
                    oleDbDataAdapter2.DeleteCommand.Connection.Open();
                    oleDbDataAdapter2.DeleteCommand.ExecuteNonQuery();
                    oleDbDataAdapter2.DeleteCommand.Connection.Close();
```

```
                    oleDbDataAdapter2.DeleteCommand.CommandText="DELETE FROM outData1 WHERE
[型号] = '" + textBox1.Text + "' " + " AND [批号] = '" + textBox2.Text + "' ;
                    oleDbDataAdapter2.DeleteCommand.Connection.Open();
                    oleDbDataAdapter2.DeleteCommand.ExecuteNonQuery();
                    oleDbDataAdapter2.DeleteCommand.Connection.Close();
                    oleDbDataAdapter3.DeleteCommand.CommandText="DELETE FROM step WHERE[型
号] = '" + textBox1.Text + "' " + " AND [批号] = '" + textBox2.Text + "';
                    oleDbDataAdapter3.DeleteCommand.Connection.Open();
                    oleDbDataAdapter3.DeleteCommand.ExecuteNonQuery();
                    oleDbDataAdapter3.DeleteCommand.Connection.Close();
                    oleDbDataAdapter4.DeleteCommand.CommandText="DELETE FROM step1 WHERE[型
号] = '" + textBox1.Text + "' " + " AND [批号] = '" + textBox2.Text + "' ;
                    oleDbDataAdapter4.DeleteCommand.Connection.Open();
                    oleDbDataAdapter4.DeleteCommand.ExecuteNonQuery();
                    oleDbDataAdapter4.DeleteCommand.Connection.Close();
                }
                else
                {
                    MessageBox.Show("请输入型号和批号","提示");
                }
            }
```

6.3.8 查询数据窗体

1．窗体界面设计

（1）添加窗体“select”，修改“外观”下的“Text”为“查询数据”。

（2）添加四个 Label，分别将“外观”下的“Text”改为“型号”、“批号”、“日期”、“至”。添加两个 TextBox。

（3）添加三个 Button，将“外观”下的“Text”均改为“查找”。

（4）添加两个 DataGrid，分别在“外观”下的 CaptionText 中输入“正极数据”、“负极数据”。完成的查找数据界面如图 6.8 所示。

图 6.8　查找数据界面

2．窗体代码设计

（1）按型号【查找】按钮代码。添加代码如下：

```
        private void button1_Click(object sender, System.EventArgs e)
        {
        try
            {
                oleDbDataAdapter1.SelectCommand.CommandText="SELECT 批号,工序, 本工序成本, 成品率, 废品损失, 累计成本, 累计废品损失,录入日期 FROM [outData] WHERE [型号] = '"+System.Convert.ToString(textBox1.Text) +"'";
                oleDbDataAdapter1.SelectCommand.Connection.Open();
                oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter1.SelectCommand.Connection.Close();
                dataSet3 = new DataSet("outData");
                oleDbDataAdapter1.Fill(dataSet3,"outData");
                dataGrid1.SetDataBinding(dataSet3,"outData");
                oleDbDataAdapter1.SelectCommand.CommandText="SELECT 批号,工序, 本工序成本, 成品率, 废品损失, 累计成本, 累计废品损失,录入日期 FROM [outData1] WHERE [型号] = '"+System.Convert.ToString(textBox1.Text) +"'";
                oleDbDataAdapter1.SelectCommand.Connection.Open();
                oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter1.SelectCommand.Connection.Close();
                dataSet1 = new DataSet("outData1");
                oleDbDataAdapter1.Fill(dataSet1,"outData1");
                dataGrid2.SetDataBinding(dataSet1,"outData1");
            }
            catch( Exception ed)
            {
                MessageBox.Show ( "查找数据库发生 " + ed.ToString ( ) , "错误！ " ) ;
                oleDbDataAdapter1.SelectCommand.Connection.Close();
            }
        }
```

（2）按批号【查找】按钮代码。添加代码如下：

```
        private void button2_Click(object sender, System.EventArgs e)
        {
            try
            {
                oleDbDataAdapter1.SelectCommand.CommandText="SELECT 型号,工序, 本工序成本, 成品率, 废品损失, 累计成本, 累计废品损失,录入日期 FROM [outData] WHERE [批号] ='" + textBox2.Text + "'" ;
                oleDbDataAdapter1.SelectCommand.Connection.Open();
                oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                oleDbDataAdapter1.SelectCommand.Connection.Close();
                dataSet3 = new DataSet("outData");
                oleDbDataAdapter1.Fill(dataSet3,"outData");
                dataGrid1.SetDataBinding(dataSet3,"outData");
```

```
                    oleDbDataAdapter1.SelectCommand.CommandText="SELECT 型号,工序, 本工序成本, 成品率, 废品损失, 累计成本, 累计废品损失,录入日期 FROM [outData1] WHERE [批号] = '"+System.Convert.ToString(textBox2.Text) +"'";
                    oleDbDataAdapter1.SelectCommand.Connection.Open();
                    oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                    oleDbDataAdapter1.SelectCommand.Connection.Close();
                    dataSet1 = new DataSet("outData1");
                    oleDbDataAdapter1.Fill(dataSet1,"outData1");
                    dataGrid2.SetDataBinding(dataSet1,"outData1");
                }
                catch( Exception ed)
                {
                    MessageBox.Show ( "查找数据库发生 " + ed.ToString ( ) , "错误！ " ) ;
                    oleDbDataAdapter1.SelectCommand.Connection.Close();
                }
            }
```

（3）按日期【查找】按钮代码。添加代码如下：

```
            private void button3_Click(object sender, System.EventArgs e)
            {
                try
                {
                    oleDbDataAdapter1.SelectCommand.CommandText="SELECT 型号,批号,工序, 本工序成本, 成品率, 废品损失, 累计成本, 累计废品损失 FROM [outData] WHERE [录入日期] BETWEEN ( '" +dateTimePicker1.Value.Date+ "') AND ('" +dateTimePicker2.Value.Date + "')";
                    oleDbDataAdapter1.SelectCommand.Connection.Open();
                    oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                    oleDbDataAdapter1.SelectCommand.Connection.Close();
                    dataSet3 = new DataSet("outData");
                    oleDbDataAdapter1.Fill(dataSet3,"outData");
                    dataGrid1.SetDataBinding(dataSet3,"outData");
                    oleDbDataAdapter1.SelectCommand.CommandText="SELECT 型号,批号,工序, 本工序成本, 成品率, 废品损失, 累计成本, 累计废品损失 FROM [outData1] WHERE [录入日期] BETWEEN '" + dateTimePicker1.Value.Date + "' AND '" +dateTimePicker2.Value.Date + "'";
                    oleDbDataAdapter1.SelectCommand.Connection.Open();
                    oleDbDataAdapter1.SelectCommand.ExecuteNonQuery();
                    oleDbDataAdapter1.SelectCommand.Connection.Close();
                    dataSet1 = new DataSet("outData1");
                    oleDbDataAdapter1.Fill(dataSet1,"outData1");
                    dataGrid2.SetDataBinding(dataSet1,"outData1");
                }
                catch( Exception ed)
                {
                    MessageBox.Show ( "查找数据库发生 " + ed.ToString ( ) , "错误！ " ) ;
                    this.oleDbConnection1.Close();
                }
            }
```

6.3.9　成本、图表和版本窗口

1．成本窗体

（1）添加窗体“cost”，修改“外观”下的“Text”为“成本”。

（2）拖放一个 Crystal Report Viewer 控件到页面中去。调出 Crystal Report Viewer 控件的属性窗口，选择 DataBindings 区单击【...】按钮。

（3）“Crystal Report Viewer 数据绑定窗口”中，在右边的“可绑定属性”中选择 ReportSource，并在右下角的“自定义绑定表达式”中指定.rpt 文件路径，选择报表文件“CrystalReport1”。此时能够从 Crystal Report Viewer 控件中看到使用一些虚拟数据组成的报表文件的预览。

注意：在上面的例子中，Crystal Report Viewer 可以在设计时直接调用真实的数据，因为此时数据已经保存。在这种情况下，设计时如果没有保存数据，是不能显示数据的。取而代之的是显示一些虚拟的数据，只有在执行时才会显示真实的数据。完成的“成本显示数据”界面如图 6.9 所示。

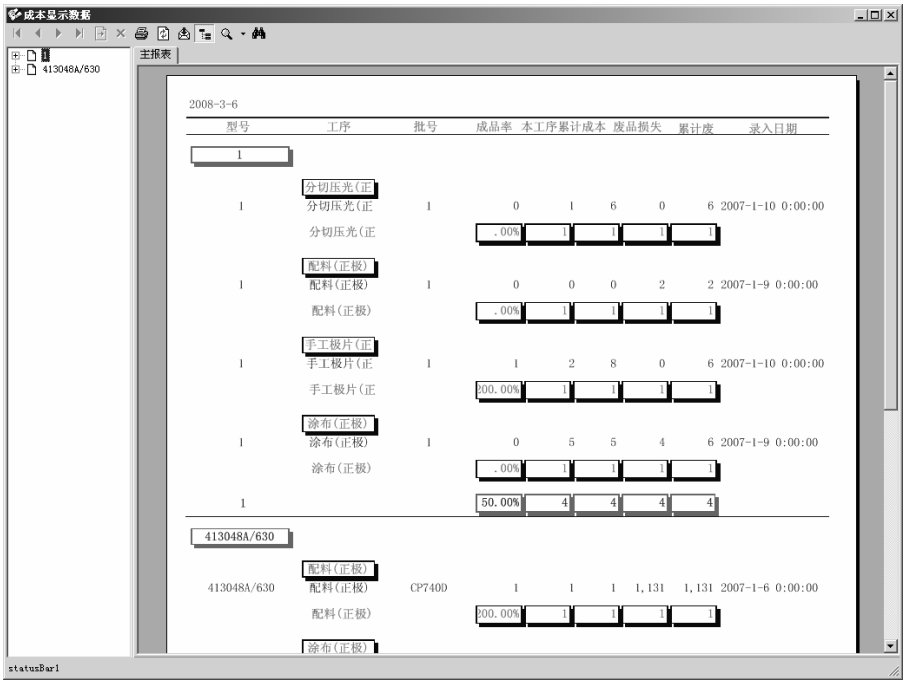


图 6.9　成本显示数据界面

本部分无代码添加。

2．图表窗体

（1）添加窗体“table”，修改“外观”下的“Text”为“图表”。

（2）打开“工具箱”，并将一个 CrystalReportViewer 拖到窗体上。通过拖放操作调整到希望的大小并将其移动到所需位置。

（3）在“属性”窗口中进行浏览，为 ReportSource 属性选择报表文件“myReport”。

设计完成的“图表显示成本”界面如图 6.10 所示。

本部分无代码添加。

3．版本窗体

（1）添加窗体“about”，单击“外观”下的 BackgroundImage 后的按钮，选择背景图片。在“Text”后添加文字“关于锂电池原材料成本管理系统”。

（2）单击“窗口样式”下的 Icon 后的按钮，选择图标图片。

（3）添加 TextBox，在“外观”下的“Text”输入软件相关信息。修改“行为”下的“Multiline”为“True”，使文本可以多行显示，修改“ReadOnly”为“True”，使该文本在使用时不可更改。设计完成的界面如图 6.11 所示。

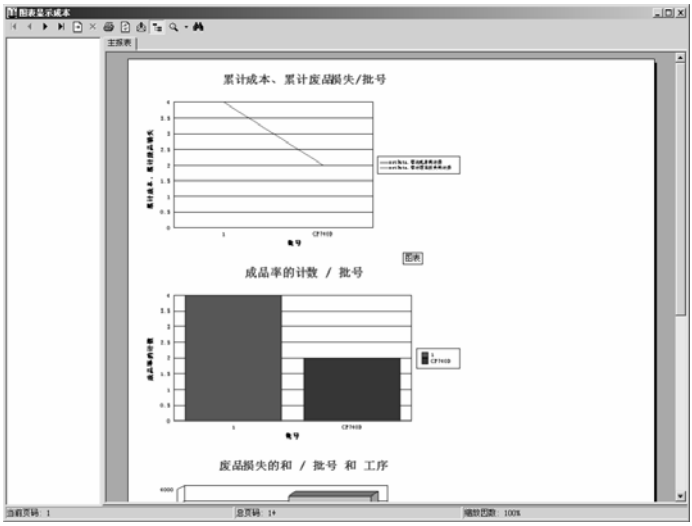


图 6.10　图表显示成本界面

图 6.11　版本界面

本部分无代码添加。

6.4　实训：锂电池原材料成本管理系统软件

1．项目要求

在本章的基础上添加分切压光和手工极片两个类别，要求如下：

（1）具有数据录入、数据修改、数据删除等数据编辑模块。

（2）具有查询模块，可以按型号、批号以及××××年××月××日～×××××年××月××日查找。

（3）分别按型号和工序显示成本。

（4）显示成品率和成本趋势图。

（5）正负极要有合计成本、累计合计成本、合计废品损失和累计合计废品损失。

（6）具体参数。负极参数与正极参数相同；分切压光输入参数有批号、型号，分切压光成品片数 $K2^{+}$，日工资 A 及相应的人数 B。输出参数及其计算公式如下：

① 成品率：$S3^{+}=K2^{+}\times(m/1000)/(M\times \mathrm{N})$

② 本工序成本：$Z3^{+}=(\sum(((K1^{+}-K2^{+})\times m/1000)\times T\times X)+((K1^{+}-K2^{+})\times G/1000)\times C+A\times B)/K2^{+}$

③ 累计成本：$W=Z1^{+}+Z2^{+}+Z3^{+}$

④ 废品损失：$Q3^{+}=\sum(((K1^{+}-K2^{+})\times m/1000)\times T\times X)+((K1^{+}-K2^{+})\times G/1000)\times C$

⑤ 累计废品损失：$Q=Q1^{+}+Q2^{+}+Q3^{+}$

2．设计提示

仿照本章的内容，在本章所讲系统的基础上添加模块分切压光，注意计算的准确性。

3．项目评价

项目评价是在教师的主持下，通过项目负责人的讲解演示，评估项目的完成情况，评价内容如下：

（1）工具类软件界面是否简洁，操作是否简便；

（2）使用过程中有无错误提示；

（3）能否实现基本功能。

习　　题

1．填空

（1）CrystalReportViewer 控件用来_____报表。

（2）CrystalReportViewer 控件中的“设计时预览功能”，作用是_____报表。

（3）BindingManagerBase管理绑定到_____的所有 Binding对象。

（4）删除数据用_____语句。

2．选择

（1）在本章中，下列关于数据库的说法正确的是（　　）。

A．使用 OleDbConnection 类进行连接

B．使用 DataSet 保存数据

C．使用 OleDbDataAdapter 填充 DataSet 和更新数据源

D．使用 SqlCommand 查询数据

（2）向数据库中添加数据使用（　　）语句。

A．delete　　B．select　　C．insert　　D．update

3．分析下列代码实现的功能。

```
myBindingManagerBase = this.BindingContext[dataSet2, "step"];
label3.Text=data1[data1.CurrentRowIndex,0].ToString();
myBindingManagerBase.Position+=1;
label4.Text=data1[data1.CurrentRowIndex,0].ToString();
```

4．写出下列代码的输出。

```
using System;
class SwitchTest
{
public static void Main()
{
    Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
    Console.Write("Please enter your selection: ");
    string s = Console.ReadLine();
    int n = int.Parse(s);
    int cost = 0;
    switch(n)
    {
        case 1:
            cost += 25;
            break;
        case 2:
            cost += 25;
            goto case 1;
        case 3:
            cost += 50;
            goto case 1;
        default:
            Console.WriteLine("Invalid selection. Please select 1, 2, or 3.");
            break;
        }
        if (cost != 0)
            Console.WriteLine("Please insert {0} cents.", cost);
        Console.WriteLine("Thank you for your business.");
    }
}
```

第 7 章　财务管理系统

本章是一个完整的企业财务管理系统，主要讲解 ToolBar、ImgeList 控件、SqlConnection、SqlCommand 类的使用，重点是链接 SQL Server 数据库，实现数据添加、查询、编辑与删除的方法。

7.1　项目说明

7.1.1　任务书

（1）项目名称：财务管理系统软件。

（2）工作期限：16 个工作日。

（3）工作任务：编写财务管理系统软件，可以方便地进行账务核算、报表处理、出纳管理及财务分析。

① 基础数据管理功能：可以方便地对会计科目进行设置，包括会计科目模块和账户设置模块。

● 会计科目模块可以对科目代码、科目名称、助记码、科目类别、余额等项目进行管理，包括增加、修改、删除、查询等功能，其中查询可以按照科目代码、科目名称或科目类别分别进行。

● 账户设置模块可以清除原来数据、建立新账簿、启用这个新账簿能够试算平衡。

② 凭证管理功能：包括凭证输入和凭证过账两个模块。

● 凭证输入模块可以对凭证编号、凭证字号、日期、制单人、会计期间、过账状态、借方合计、贷方合计进行管理，包括增加、修改、删除、查询等功能，其中查询可以按照凭证编号、会计期间和日期分别进行。

● 凭证过账是在数据表“分录表”中分别查找凭证编号、借方和贷方，凭证编号的数量记为凭证数，并计算借方和贷方数据的总和，分别记为借方金额和贷方金额，通过过账模块将该数据显示给用户并将分录表及凭证表中的已过账数据清零。

③ 账簿管理功能：可以查看总账和详细的明细账管理，包括总分类账查询模块和明细账查询模块。

● 总分类账查询模块可以按照凭证编号和有无发生余额对本期汇总账簿进行查询。

● 明细账查询模块可以按照科目代码对本期明细账簿进行查询，并可以选择要在表中显示的项目。

④ 结账及报表功能：可以方便地结账及管理报表，包括试算平衡表、科目余额表和资产负债表模块。

● 在结账前，需要对账簿进行试算平衡，这就需要一个试算平衡表。试算平衡表通过对期初借方、期初贷方、本期发生借方、本期发生贷方、期末借方和期末贷方中的数据进行综

合计算，判断账目是否平衡。如果期初借方与期初贷方的总和相等、本期发生借方与本期发生贷方的总和相等、期末借方与期末贷方的总和相等，则为平衡，否则为不平衡。

● 期末结账时，先将没有过账的凭证过账，接着计算科目余额表，再统计资产负债表。

其中：本年借方累计发生额＝本年借方累计发生额＋本期借方发生额；

本年贷方累计发生额＝本年贷方累计发生额＋本期贷方发生额；

本期借方余额＝期初借方余额＋本期借方发生额；

本期贷方余额＝期初贷方余额＋本期贷方发生额。

然后清除汇总账簿和明细账簿，进入新的会计期间，最后创建一个新会计期间的科目余额表，也就将原有的科目余额表数据清空。

● 资产负债表模块用来计算资产负债并用报表的形式显示。其中没有列出的明细项目都归到其他资产和其他负债里，表格按照科目代码统计。具体过程详见相关资料。

⑤ 相关资料。

● 计算资产负债表，相关公式及科目代码范围如下：

现金及现金等价物＝本期借方余额－本期贷方余额（101≤科目代码≤111）

应收账款＝本期借方余额－本期贷方余额（1≤科目代码≤129 且不等于 125）

坏账准备（贷方）＝本期贷方余额－本期借方余额（科目代码等于 125）

应收账款净值＝应收账款－坏账准备

流动资产总计＝现金及现金等价物＋应收账款净值

固定资产原值＝本期借方余额－本期贷方余额（科目代码等于 171）

累计折旧（贷方）＝本期贷方余额－本期借方余额（科目代码等于 175）

固定资产总计＝固定资产原值－累计折旧

其他资产＝本期借方余额－本期贷方余额（131≤科目代码≤195 并且科目代码不等于 171 或 175）

资产总计＝流动资产总计＋固定资产总计＋其他资产

● 计算负债及所有者权益，负债类都是贷方金额，相关公式及科目代码范围如下：

应付账款＝本期贷方余额－本期借方余额（201≤科目代码≤204）

预收账款＝本期贷方余额－本期借方余额（科目代码为 206）

应付工资＝本期贷方余额－本期借方余额

其他负债＝本期贷方余额－本期借方余额（209≤科目代码≤281 且科目代码不等于 215）

负债总计＝应付账款＋预收账款＋应付工资＋其他负债

● 计算所有者权益，相关公式及科目代码如下：

实收资本＝本期贷方余额－本期借方余额（科目代码等于 301）

资本公积＝本期贷方余额－本期借方余额（科目代码等于 311）

盈余公积＝本期贷方余额－本期借方余额（科目代码等于 313）

未分配利润＝本期贷方余额－本期借方余额（科目代码等于 322）

所有者权益总计＝实收资本＋资本公积＋盈余公积＋未分配利润

负债及所有者权益总计＝负债总计＋所有者权益总计

7.1.2 计划书

1．工作内容

（1）分析项目要完成的功能，确定所使用的软件开发工具和开发系统环境，安装好所需工具软件和系统环境，准备好所需资料（约需要时间：1 个工作日）。

（2）分析项目所需记录的数据，确定数据结构，确定采用的数据库管理系统，创建数据库（约需要时间：3 个工作日）。

（3）项目需求分析，确立开发方案，进行软件的概念分析、功能结构分析、逻辑设计、界面的初步设计等（约需要时间：2 个工作日）。

（4）软件的物理设计，模块功能设计，代码的初步实施（约需要时间：6 个工作日）。

（5）软件的代码实施，代码的功能测试，各模块的组装调试，软件的整体调试，软件注释文档的完善（约需要时间：3 个工作日）。

（6）创建软件系统的安装文件，发布软件的测试版本，并与用户完成软件的整体测试与功能完善（约需要时间：1 个工作日）。

2．项目分析

（1）本系统是财务管理软件，数据管理量大，选用 SQL Server 数据库。

（2）本软件为公司财务所专用，单机作业，存储量大。

（3）主界面使用控件 MainMenu 编辑菜单，方便实现主体控制。软件根据任务书的要求可以分为四大模块：基础数据管理、凭证管理、账簿管理、结账及报表：

① 在基础数据管理模块下设置会计科目和项目设计。

② 在凭证管理模块下设置凭证输入和凭证过账。

③ 在账簿管理模块下设置总分类账和明细账。

④ 在结账及报表模块下设置试算平衡表、期末结账和资产负债表三个表格。

（4）在会计科目设计界面下，用户可以添加、修改、删除科目的详细信息，并根据科目代码、科目名称和科目类别进行简单查询。

① 控制按钮使用 ToolBar 控件、ImageList 控件和 Button 控件共同完成。

② 使用 GroupBox 进行信息的分组。

③ 使用 Grid 管理数据信息。

④ 由于科目类别、余额等项目只有几个选项供选择，使用 ComboBox 控件可以方便用户的输入。

（5）账户设置界面使用两个 Grid 控件显示表中的信息，使用三个按钮 Button 控件来控制新账簿的建立与启用，并且可以试算平衡。

（6）凭证输入界面与会计科目设计界面类似，只不过多了一个 Grid 控件用于显示数据信息。

（7）总分类账界面设计如下：

① 使用 GroupBox 进行信息的分组。

② 使用一个 Grid 控件显示数据信息。

③ 使用 Label 控件、TextBox 控件制作一个凭证编号输入框。

④ 用 Button 控件来制作一个手动搜索按钮。

⑤ 添加 CheckBox 控件来标志是否显示余额。

（8）明细账界面设计。明细账与总分类账界面相似，具体如下：

① 使用一个 Grid 控件显示数据信息。

② 使用 GroupBox 进行信息的分组。

③ 使用 Label 控件、TextBox 控件制作一个科目代码输入框。

④ 使用 Button 控件来制作一个手动搜索按钮。

⑤ 添加 ComboBox 控件来选择要显示在表中的项目。

（9）试算平衡表界面只包括两个 Grid 控件和一个 GroupBox 控件，GroupBox 控件的主要功能是标志试算结果平衡。

（10）资产负债表界面使用报表控件 CrystalReportViewer 进行显示，添加 NumericUpDown 控件用于选择要统计的会计期间，选择完毕后，使用控件 Button 制作按钮来手动显示报表。

（11）数据库使用 SqlConnection 类进行连接，使用 DataSet 保存数据，使用 SqlDataAdapter 填充 DataSet 和更新数据源。

（12）版本信息使用控件 Label 进行简单标注。

7.2 项目准备

本项目涉及 ToolBar、NumericUpDown、CheckBox 等控件，SqlConnection、SqlCommand 等类，还有 ImageList 组件。

7.2.1 控件

1．ToolBar控件

Windows 窗体 ToolBar 控件用做窗体上的控制条，用于显示一行下拉菜单和一些可激活命令的位图按钮。因此，单击工具栏按钮相当于选择菜单命令。可将按钮配置为以普通按钮、下拉菜单或分隔符等形式来显示和使用。通常情况下，工具栏包含的按钮和菜单与应用程序菜单结构中的选项一一对应，以提供对应用程序常用功能和命令的快速访问。

ToolBar控件通过向Buttons集合添加Button对象来创建工具栏。可以使用集合编辑器向ToolBar控件添加按钮；应为每个Button对象指定文本或图像，也可以两者都指定。图像由一个关联的 ImageList组件提供。运行时，可使用Add方法和Remove方法分别向ToolBarButtonCollection中添加按钮或从中移除按钮。若要为Toolbar按钮进行编程，可向ToolBar控件的ButtonClick事件添加代码，使用ToolBarButtonClickEventArgs类的Button属性来确定所单击的按钮。

2．NumericUpDown控件

Windows 窗体 NumericUpDown 控件看起来像是一个文本框与一对箭头的组合，用户可以单击箭头来调整数值。该控件显示并设置选择列表中的单个数值。用户可以通过单击向上和向下按钮、单击向上键和向下键来增大和减小数字。单击向上键时，数字数值逐渐增加直到最大值为止；单击向下键时，数字值逐渐减小直到最小值为止。此类控件经常用于音乐播放器上的音量控件。某些 Windows 控制面板应用程序中使用了数值 up-down 控件。

该控件可以采用多种格式（包括十六进制）显示数字，其主要属性为 Value、Maximum

（默认值为 100）、Minimum（默认值为 0）和 Increment（默认值为 1）。Value 属性设置该控件中选定的当前数字。Increment 属性设置用户单击向上或向下按钮时值的调整量。该控件的主要方法是 UpButton 和 DownButton。

3．CheckBox控件

Windows 窗体 CheckBox 控件指示某特定条件是打开的还是关闭的。常用于为用户提供“是/否”或“真/假”选项。可以成组使用复选框（CheckBox）控件以显示多重选项，用户可以从中选择一项或多项。

复选框（CheckBox）控件和单选按钮（RadioButton）控件的相似之处在于都是用于指示用户所选的选项，不同之处在于单选按钮组中一次只能选择一个单选按钮，复选框（CheckBox）控件，则可以选择任意数量的复选框。

复选框可以使用简单数据绑定连接到数据库中的元素。多个复选框可以使用 GroupBox 控件进行分组，这对于外观以及用户界面设计很有用，因为成组控件可以在窗体设计器上同时移动。

CheckBox 控件有两个重要属性 Checked 和 CheckState，其中 Checked 属性返回“true”或“false”；CheckState 属性返回“CheckState.Checked”或“CheckState.Unchecked”。但如果 ThreeState 属性设置为“true”，CheckState 还可能返回“CheckState.Indeterminate”，在不确定状态下，复选框以浅灰色显示，表示该选项不可用。

7.2.2 组件ImageList

组件 ImageList 就是一个图像列表。一般情况下，这个属性用于存储一个图像的集合，这些图像用做工具栏图标或 TreeView 控件上的图标。许多控件都包含 ImageList 属性，这个属性一般和 ImageList 属性一起使用。ImageList 属性设置为 ImageList 组件的一个实例，ImageIndex 属性设置为 ImageList 中应在控件中显示的图像的索引。使用 ImageIndex.Image 属性的 Add 方法可以把图像添加到 ImageList 组件中，Images 属性返回一个 ImageCollection。

设计软件时可以将图像列表用于任何具有 ImageList 属性的控件，与图像列表相关联的控件包括：ListView、TreeView、ToolBar、TabControl、Button、CheckBox、RadioButton 和 Label 控件。

7.2.3 类

1．SqlConnection类

SqlConnection类表示创建并打开SQL Server数据库链接。对于客户端/服务器数据库系统，相当于到服务器的网络连接。SqlConnection与 SqlDataAdapter和 SqlCommand一起使用，以便在连接Microsoft SQL Server数据库时提高性能。当创建SqlConnection的实例时，所有属性都设置为初始值。如果SqlConnection超出范围，则不会将其关闭，必须通过调用 Close或 Dispose 显式关闭该连接。

如果执行SqlCommand的方法生成 SqlException，那么当严重度等于或小于 19 时，SqlConnection将仍保持打开状态。当严重度等于或大于 20 时，服务器通常会关闭 SqlConnection，但用户可以重新打开连接并继续操作。

2．SqlDataAdapter类

SqlDataAdapter类表示用于填充 DataSet和更新SQL Server数据库的一组数据命令和一个数据库连接，是 DataSet和SQL Server之间的桥接器，用于检索和保存数据，不能继承此类。SqlDataAdapter通过对数据源使用适当的Transact-SQL语句映射 Fill（更改DataSet中的数据以匹配数据源中的数据）和 Update（更改数据源中的数据以匹配DataSet中的数据）来提供这一桥接。

当SqlDataAdapter填充DataSet时，将为返回的数据创建必要的表和列。但是，除非MissingSchemaAction属性设置为 AddWithKey，否则这个隐式创建的架构中就将不包括主键信息。也可以在使用 FillSchema为数据集填充数据前，让SqlDataAdapter创建DataSet的架构（包括主键信息）。

SqlDataAdapter与 SqlConnection和 SqlCommand一起使用，以便在连接到Microsoft SQL Server 数据库时提高性能。SqlDataAdapter 还包括 SelectCommand、InsertCommand、DeleteCommand、UpdateCommand 和 TableMappings属性，使数据的加载和更新更加方便。当创建SqlDataAdapter的实例时，读/写属性将被设置为初始值。

3．SqlCommand类

SqlCommand 类表示要对 SQL Server 数据库执行的一个 Transact-SQL 语句或存储过程，不能继承此类。当创建 SqlCommand 的实例时，读/写属性将被设置为初始值。SqlCommand 特别提供了以下对 SQL Server 数据库执行命令的方法：

（1）ExecuteReader：执行返回行的命令。为了提高性能，ExecuteReader使用Transact-SQL sp_executesql系统存储过程调用命令，因此，如果用于执行像Transact-SQL SET语句这样的命令，ExecuteReader可能无法获得预期效果。

（2）ExecuteNonQuery：执行 Transact-SQL INSERT、DELETE、UPDATE及SET语句等命令。

（3）ExecuteScalar：从数据库中检索单个值（例如一个聚合值）。

（4）ExecuteXmlReader：将CommandText发送到 Connection并生成一个 XmlReader对象。

可以重置CommandText属性并重载SqlCommand对象。但是，在执行新的命令或先前命令之前，必须关闭 SqlDataReader。如果执行SqlCommand的方法生成SqlException，那么当严重度等于或小于 19 时，SqlConnection将仍保持打开状态。当严重度等于或大于 20 时，服务器通常会关闭SqlConnection。但是用户可以重新打开连接并继续操作。

4．DataView类

DataView类表示用于排序、筛选、搜索、编辑和导航 DataTable的可绑定数据自定义视图。DataView的一个主要功能是允许数据与Windows窗体和Web窗体绑定。

另外可自定义DataView来表示 DataTable中数据的子集。该功能用于将两个控件与同一个DataTable绑定，但显示数据的不同版本。例如，一个控件可与显示表中所有行的DataView绑定，而另一个控件可配置为只显示已从DataTable中删除的行。若要创建数据的筛选和排序视图，请设置RowFilter和 Sort属性，然后使用 Item属性返回单个 DataRowView，还可使用 AddNew 和 Delete方法从行的集合中进行添加和删除。在使用这些方法时，可设置 RowStateFilter属性以便指定只有已经被删除的行或新行才可由DataView显示。

7.3 项目开发

在 Visual Studio 2008 中新建一个项目，命名为“财务管理软件”，将文件保存在 D 盘中。通过项目分析，本软件窗体分为系统主窗体、科目设置窗体、账户设置窗体、会计凭证输入窗体、总分类账查询窗体、明细账窗体、资产负债表窗体和试算平衡表窗体。

7.3.1 数据库操作

1．数据库、表设计

本系统使用 SQL 数据库。在 Visual Studio 2008 开发环境中，启动“服务器资源管理器”窗口，在 SQL Server 服务器节点右击，在弹出的快捷菜单中选择“新建数据库”命令打开“创建数据库”对话框，在“新数据库名”文本框中输入“caiwubook”，选择使用“Windows NT 集成安全性”访问方式。在“查询分析器”中选数据库“caiwubook”，并运行“财务管理系统实例程序.sql”中的代码。表结构如图 7.1 所示。

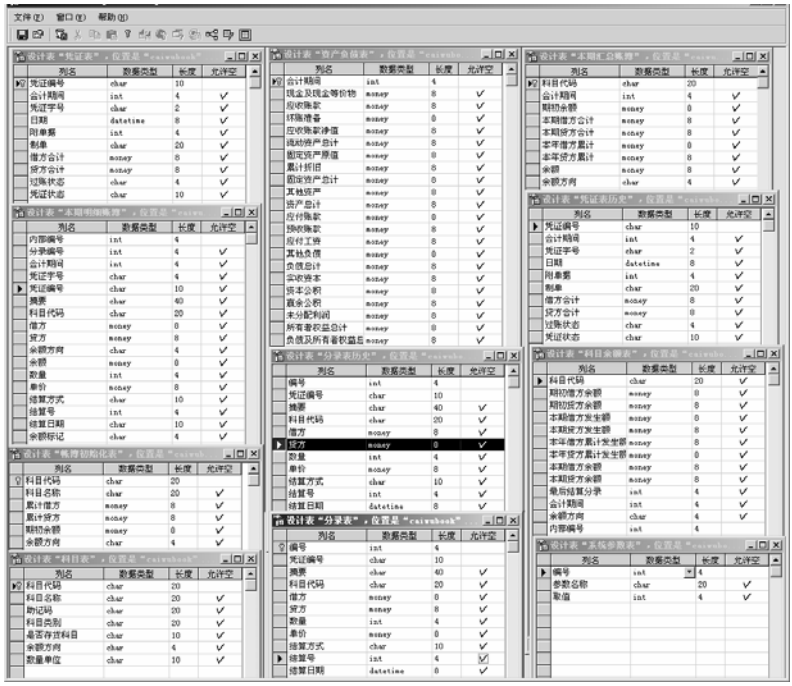


图 7.1　表结构

2．连接数据库

在窗口中直接添加 SqlConnection 对象并通过设置 ConnectiongString 属性来连接数据库。具体步骤如下：

（1）单击 ConnectiongString 后的下拉菜单，会弹出“数据链接属性”菜单。

（2）在“提供程序”中选择“Microsoft OLE DB Provider for SQL Server”，单击【下一步】按钮。

（3）指定连接的信息，服务器名称为当前计算机的名称，在“输入登录服务器的信息”中选择“使用 Windows NT 集成安全设置”，在“服务器上选择数据库”中选择“caiwubook”。

（4）单击【确定】按钮，完成连接。

7.3.2 报表设计

添加报表的具体步骤如下：

（1）右键单击“解决方案浏览器”，在弹出的菜单中选择“添加”→“添加新项”→“Crystal Report”。

（2）在“Crystal Report 库”中选择“作为空白报表”单选按钮，最后单击【确定】按钮。

（3）在弹出的水晶报表设计器上，右键单击报表中的“详细资料区”，选择“数据库”→“添加/删除数据库.....”。

（4）在弹出的“数据库专家”中，选择扩展“OLE DB（ADO）”选项，此时会弹出另外一个“OLE DB（ADO）”窗口。

（5）在“OLE DB （ADO）”弹出窗口中，选择“Microsoft OLE DB Provider for SQL Server”然后单击【下一步】按钮。

（6）指定连接的信息，服务器名称为当前计算机的名称，在“集成安全”后的方框中打上勾，表示选中，其他使用默认值。

（7）单击【下一步】按钮，最后单击【完成】按钮。这时就能在“数据库专家”窗口中看到选择的数据库。

（8）打开“caiwubook”数据库，展开“表”，选择所有表并将其加到“选定的表”区中，单击【确定】按钮。在“字段资源浏览器”左边的“数据库字段”区中将显示所选择的表，以及表中的字段。

（9）拖放需要的字段进入报表的“详细资料”区，字段名将会自动出现在“页眉”区。如果需要修改头部文字，则可以右击“页眉”区中的文字，选择“编辑文本对象”选项并进行编辑。

（10）保存，这样就有了一个水晶报表文件，如图 7.2 所示。

图 7.2　报表“CrystalReport1”

7.3.3 系统主窗体

系统的主控制界面是用户进入到各个界面的总控制界面，当系统启动就可进入到此界面进行相关操作。

1．窗体界面设计

（1）设计窗体外观。在新建项目时会自动新建一个窗体 Form1，进行下列操作：

① 单击窗体“设计”菜单的 name 项，修改为“MainForm”。

② 单击“布局”下 StartPosition 后的下拉菜单，选择“CenterScreen”，这表示窗体首次打开时是在电脑的正中央，亦可以选择其他位置，详细情况可以按下 F1 键查询。

③ 单击“外观”下的“BackgroundImage”后的按钮，选择背景图片，在 Text 后添加“财务管理系统”，这是要在窗体边框上显示的文本。

在选择相应的选项时，在“属性”窗体的下方会有相应的中文说明，可以根据自己的习惯、喜好进行设计。

（2）添加菜单。将“工具箱”中“Windows 窗体”中的 MainMenu 拖曳到窗体中，就会看到一个上边写着“请在此输入”的可输入文本框，首先填写“基础数据管理”，在其后边的文本框中输入“凭证管理”，在后边的文本框中输入“账簿管理”，在其后边的文本框中输入“结账及报表”，在其后边的文本框中输入“退出系统”。

在“基础数据管理”下边的文本框中输入“会计科目设置”和“账户设置”，在“凭证管理”后边的文本框中输入“凭证输入”和“凭证过账”，在“账簿管理”后输入“总分类账”和“明细账”，在“结账及报表”后输入“试算平衡表”、“期末结账”和“资产负债表”。最后创建出的系统主控制界面如图 7.3 所示。

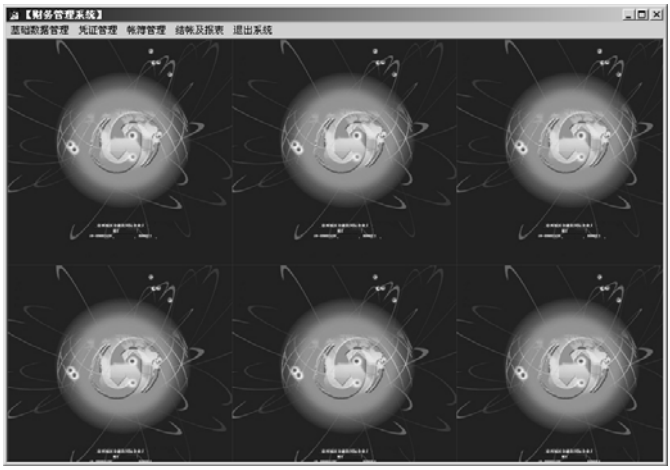


图 7.3 系统主控制界面

2．窗体代码设计

（1）【会计科目设置】按钮代码。双击“基础数据管理”下的【会计科目设置】按钮添加如下代码：

```
//-----------------显示科目设置窗体-----------------------
private void menuItem2_Click(object sender, System.EventArgs e)
{
    //通过窗体名称查询该窗体是否已经存在，如存在则显示，否则就新创建一个
    if (this.checkChildFrmExist("SubjectSetting") == true)
    {
        return;
    }
    SubjectSetting newFrm=new SubjectSetting();
    newFrm.MdiParent = this;
    newFrm.Show();
}
```

（2）【账户设置】按钮代码。添加如下代码：

```
//-----------------显示账户设置窗体---------------
private void menuItem3_Click(object sender, System.EventArgs e)
{
    //通过窗体名称查询该窗体是否已经存在，如存在则显示，否则就新创建一个
    if (this.checkChildFrmExist("AccountSetting") == true)
    {
        return;
    }
    AccountSetting newFrm=new AccountSetting();
    newFrm.MdiParent = this;
    newFrm.Show();
}
```

（3）【凭证输入】按钮代码。添加如下代码：

```
//-----------------显示凭证输入窗体---------------
private void menuItem5_Click(object sender, System.EventArgs e)
{
    //通过窗体名称查询该窗体是否已经存在，如存在则显示，否则就新创建一个
    if (this.checkChildFrmExist("WarrentInput") == true)
    {
        return;
    }
    WarrentInput newFrm=new WarrentInput();
    newFrm.MdiParent = this;
    newFrm.Show();
}
```

（4）【凭证过账】按钮代码。添加如下代码：

```
//--------------执行凭证过账功能---------------
private void menuItem6_Click(object sender, System.EventArgs e)
{
    DialogResult  result=MessageBox.Show("凭证过账后即不可再修改，是否过账","确认过账
```

```
",MessageBoxButtons.OKCancel,MessageBoxIcon.Warning);
                if(result==DialogResult.OK)
                {
                    string strConn = "workstation id=localhost;Integrated Security=SSPI; database=caiwubook";
                    SqlConnection cn=new SqlConnection(strConn);
                    cn.Open();
                    SqlCommand cmd=cn.CreateCommand();
                    cmd.CommandText="select count(distinct 凭证编号) 凭证数,sum(借方) "+"借方金
额,sum(贷方) 贷方金额 from 分录表";
                    SqlDataReader dr=cmd.ExecuteReader();
                    dr.Read();
                    string warrentCount=dr.GetValue(0).ToString();//读入凭证数
                    string debit=dr.GetValue(1).ToString();//读入借方金额
                    string loan=dr.GetValue(2).ToString();//读入贷方金额
                    string messageStr="过账成功!\n"+"过账凭证总数为:"+warrentCount.Trim()+"\n"+"借方金
额合计:"+debit.Trim()+"\n"+"贷方金额合计:"+loan.Trim()+"\n";
                    dr.Close();//准备显示的过账信息
                try
                {
                    cmd.CommandText="exec sf_凭证过账";
                    cmd.ExecuteNonQuery();
                    MessageBox.Show(messageStr,"凭证过账成功!",MessageBoxButtons.OK,MessageBoxIcon.
Information);
                    cmd.CommandText="delete from 分录表";//删除已经过账的分录数据
                    cmd.ExecuteNonQuery();
                    cmd.CommandText="delete from 凭证表";//删除已经过账的凭证数据
                    cmd.ExecuteNonQuery();
                }
                catch(Exception express)
                {
                    MessageBox.Show(express.ToString(),"凭证过账失败",MessageBoxButtons.OK,Message
BoxIcon.Error);
                    return;
                }
            }
        }
```

（5）【总分类账】按钮代码。添加如下代码：

```
        //--------------执行总分类账查询功能----------------
        private void menuItem8_Click(object sender, System.EventArgs e)
        {
            //通过窗体名称查询该窗体是否已经存在，如存在则显示，否则就新创建一个
            if (this.checkChildFrmExist("LedgerQuery") == true)
            {
                return;
            }
```

```
            LedgerQuery newFrm=new LedgerQuery();
            newFrm.MdiParent = this;
            newFrm.Show();
        }
```

（6）【明细账】按钮代码。添加如下代码：

```
        //--------------执行明细账查询功能---------------
        private void menuItem9_Click(object sender, System.EventArgs e)
        {
            //通过窗体名称查询该窗体是否已经存在，如存在则显示，否则就新创建一个
            if (this.checkChildFrmExist("DetailQuery") == true)
            {
                return;
            }
            DetailQuery newFrm=new DetailQuery();
            newFrm.MdiParent = this;
            newFrm.Show();
        }
```

（7）【试算平衡表】按钮代码。添加如下代码：

```
        //-------------显示使用试算平衡窗体---------------
        private void menuItem11_Click(object sender, System.EventArgs e)
        {
            //通过窗体名称查询该窗体是否已经存在，如存在则显示，否则就新创建一个
            if (this.checkChildFrmExist("TrialBalance") == true)
            {
                return;
            }
            TrialBalance newFrm=new TrialBalance(false);
            newFrm.Text+="正式使用";
            newFrm.MdiParent = this;
            newFrm.Show();
        }
```

（8）【期末结账】按钮代码。添加如下代码：

```
        //------------调用储存过程，实现期末结账--------------
        private void menuItem12_Click(object sender, System.EventArgs e)
        {
            DialogResult  result=MessageBox.Show("结账后将进入下一会计期间，是否结账","确认结账
",MessageBoxButtons.OKCancel,MessageBoxIcon.Warning);
            if(result==DialogResult.OK)
            {
                string strConn = "workstation id=localhost;Integrated Security=SSPI; database=caiwubook";
                SqlConnection cn=new SqlConnection(strConn);
                cn.Open();
                try
```

```
                {
                    SqlCommand cmd=cn.CreateCommand();
                    cmd.CommandText="exec sf_期末结账";
                    cmd.ExecuteNonQuery();
                    MessageBox.Show("期末结账成功!","结账成功",MessageBoxButtons.OK,Message
BoxIcon.Information);
                }
                catch(Exception express)
                {
                    MessageBox.Show(express.ToString(),"期末结账失败",MessageBoxButtons.OK,Message
BoxIcon.Error);
                    return;
                }
            }
        }
```

（9）【资产负债表】按钮代码。添加如下代码：

```
        //---------显示资产负债表----------
        private void menuItem13_Click(object sender, System.EventArgs e)
        {
            //通过窗体名称查询该窗体是否已经存在，如存在则显示，否则就新创建一个
            if (this.checkChildFrmExist("FinancialReport") == true)
            {
                return;
            }
            FinancialReport newFrm=new FinancialReport();
            newFrm.MdiParent = this;
            newFrm.Show();
        }
```

（10）【退出系统】按钮代码。添加如下代码：

```
        //---------关闭窗体------------
        private void menuItem14_Click(object sender, System.EventArgs e)
        {
            this.Close();
        }
```

7.3.4 科目设置窗体

1．窗体界面设计

（1）设计窗体外观。添加新窗体 Form1，在“设计”的 name 项中修改名称为“SubjectSetting”。单击“外观”下的 BackgroundImage 后的按钮选择背景图片。在 Text 后添加“科目设置”，这是要在窗体边框上显示的文本。

（2）添加工具栏。将“工具箱”中“Windows 窗体”中的 ImageList 拖曳到窗体中，其默认名称为“imageList1”，单击“外观”下的 Images 后的[…]，进入“Image 集合编辑器”，如

图 7.4 所示添加十个图标。

将“工具箱”→“Windows 窗体”中的 ToolBar 拖曳到窗体中，单击“行为”→ImageList 后的下拉菜单，会发现刚才添加的“imageList1”，单击“行为”→Buttons 后的［…］，进入“ToolBarButton 集合编辑器”，如图 7.4 所示添加十个成员（在图 7.4 中添加十个成员）。进行如下设置：

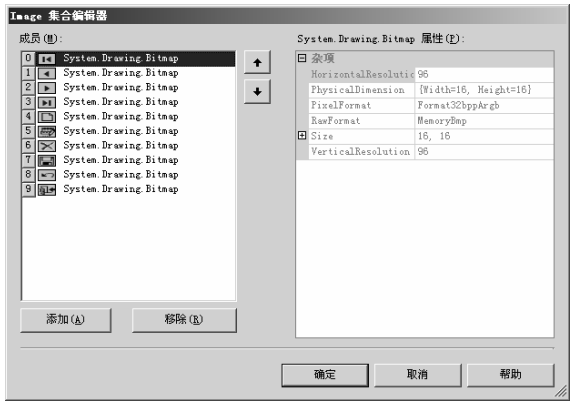


图 7.4　Image 集合编辑器

成员 0：name 为“tBtnFirst”，Text 为“首记录”（该文本将在按钮上显示），ToolTipText 为“首记录”（该文本在鼠标滑过按钮时显示），在 ImageIndex 中选择该按钮所显示的图标“0”。

成员 1：name 为“tBtnPre”，Text 为“上一记录”，ToolTipText 为“上一记录”，ImageIndex 为“1”。

成员 2：name 为“tBtnNext”，Text”为“下一记录”，ToolTipText 为“下一记录”，ImageIndex 为“2”。

成员 3：name 为“tBtnLast”，Text 为“尾记录”，ToolTipText 为“尾记录”，“ImageIndex 为“3”。

成员 4：name 为“tBtnNew”，Text 为“新增”，ToolTipText 为“新增”，ImageIndex 为“4”。

成员 5：name 为“tBtnEdit”，Text 为“修改”，ToolTipText 为“修改”，ImageIndex 为“5”。

成员 6：name 为“tBtnDelete”，Text 为“删除”，ToolTipText 为“删除”，ImageIndex 为“6”。

成员 7：name 为“tBtnSubmit”，Text 为“提交”，ToolTipText 为“提交”，“ImageIndex 为“7”。

成员 8：name 为“tBtnCancel”，Text 为“取消”，ToolTipText 为“取消”，ImageIndex 为“8”。

成员 9：name 为“tBtnQuit”，Text 为“退出”，ToolTipText 为“退出”，ImageIndex”为“9”。

设置后的“ToolBarButton 集合编辑器”如图 7.5 所示。

（3）添加 SqlDataAdapter。左键双击“工具箱”中“数据”下的 SqlDataAdapter，打开“数据适配器配置向导”。

① 单击【下一步】按钮，选择数据库“caiwubook”。

② 单击【下一步】按钮，单击【查询生成器】按钮，选择表“科目表”，在表中在选择“所有列”，单击【确定】按钮。

③ 单击【下一步】按钮，完成数据配置。

④ 添加的 SqlDataAdapter 默认名称为 sqlDataAdapter1，修改 Name 中的名称为“da1”。

图 7.5　ToolBarBotton 集合编辑器

⑤ 添加 DataSet。右键单击 da1，选中“生成数据集”，在“选择数据集”选择“新建”，在“选择要添加到数据集中的表”中选择“科目表”，勾选“将此数据集添加到设计器”，单击【确定】按钮，就会自动生成 dataSet11。按照前面讲过的方法设计界面，如图 7.6 所示。

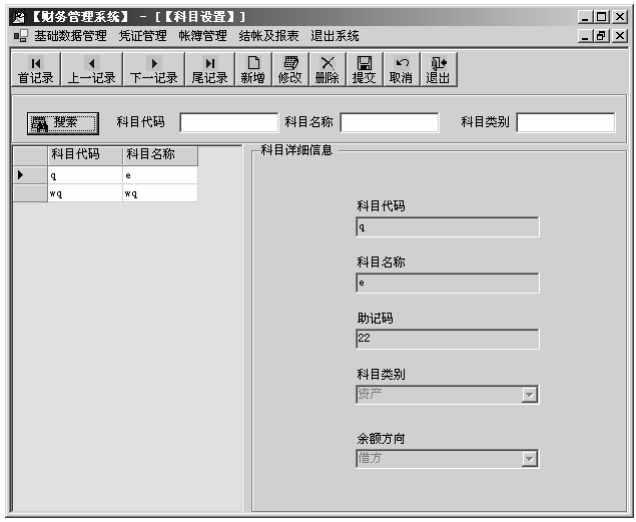


图 7.6　科目设置窗体

2．窗体代码设计

（1）初始化代码。左键双击窗体中无控件填充的部分，添加如下代码：

```
//-------------显示窗体，填充数据------------
private void SubjectSetting_Load(object sender, System.EventArgs e)
{
    //为数据集添加数据项浏览控制-
    cmOrders=(CurrencyManager) BindingContext[dataSet11,"科目表"];
    //读入全部数据
    da1.SelectCommand.Parameters[0].Value="%%";
    da1.SelectCommand.Parameters[1].Value="%%";
```

```
        da1.SelectCommand.Parameters[2].Value="%%";
        da1.Fill(dataSet11);
    }
```

（2）【搜索】按钮代码。左键双击【搜索】按钮，添加如下代码：

```
    //----------根据输入搜索数据----------
    private void btnSearch_Click(object sender, System.EventArgs e)
    {
        da1.SelectCommand.Parameters[0].Value="%%";
        da1.SelectCommand.Parameters[1].Value="%%";
        da1.SelectCommand.Parameters[2].Value="%%";
        //根据用户在文本框中的输入来设置 SQL 查询的参数
        if(txt1.Text.Trim()!="")
        {
            da1.SelectCommand.Parameters[0].Value="%"+txt1.Text.Trim()+"%";
        }
        if(txt2.Text.Trim()!="")
        {
            da1.SelectCommand.Parameters[1].Value="%"+txt2.Text.Trim()+"%";
        }
        if(txt3.Text.Trim()!="")
        {
            da1.SelectCommand.Parameters[2].Value="%"+txt3.Text.Trim()+"%";
        }
        //清空数据表，并根据新设置的查询参数重新填充
        dataSet11.科目表.Clear();
        da1.Fill(dataSet11);
    }
```

（3）工具条“ToolBar”内的按钮使用代码。左键双击“ToolBar”工具条，添加，添加如下代码：

```
    //--------------处理工具栏事务--------------
    private void toolBar1_ButtonClick(object sender, System.Windows.Forms. ToolBarButtonClickEventArgs e)
    {
        if (e.Button.ToolTipText == "首记录")
        {
            this.dataGrid1.UnSelect(cmOrders.Position); //取消原选中的行
            cmOrders.Position = 0;
            this.dataGrid1.Select(cmOrders.Position); //选中当前行
            this.dataGrid1.CurrentRowIndex = cmOrders.Position; //移动表头指示图标
            return;
        }
        if (e.Button.ToolTipText == "上一记录")
        {
            if (cmOrders.Position >= 0)
            {
```

```
                this.dataGrid1.UnSelect(cmOrders.Position);
                cmOrders.Position--;
                this.dataGrid1.Select(cmOrders.Position);
                this.dataGrid1.CurrentRowIndex = cmOrders.Position;
            }
            return;
        }
        if (e.Button.ToolTipText == "下一记录")
        {
            if (cmOrders.Position <= cmOrders.Count-1)
            {
                this.dataGrid1.UnSelect(cmOrders.Position);
                cmOrders.Position++;
                this.dataGrid1.Select(cmOrders.Position);
                this.dataGrid1.CurrentRowIndex = cmOrders.Position;
            }
            return;
        }
        if (e.Button.ToolTipText == "尾记录")
        {
            this.dataGrid1.UnSelect(cmOrders.Position);
            cmOrders.Position = cmOrders.Count-1;
            this.dataGrid1.Select(cmOrders.Position);
            this.dataGrid1.CurrentRowIndex = cmOrders.Position;
            return;
        }
        if(e.Button.ToolTipText=="新增")
        {
            cmOrders.AddNew();
            //设置默认值
            comboBox1.SelectedIndex=0;
            comboBox2.SelectedIndex=0;
            SetModifyMode(true);
        }
        if(e.Button.ToolTipText=="修改")
        {
            SetModifyMode(true);
        }
        if(e.Button.ToolTipText=="删除")
        {
            DialogResult result=MessageBox.Show("确认删除？","删除数据",MessageBoxButtons.
OKCancel);
            if(result==DialogResult.OK)
                if(cmOrders.Count>0)
                    cmOrders.RemoveAt(cmOrders.Position);
                else
                    MessageBox.Show("表中为空，已无可删除数据","提示",MessageBoxButtons.
```

```
OK,MessageBoxIcon.Error);
                }
                if(e.Button.ToolTipText=="提交")
                {
                    if(txt4.Text.Trim()=="")//检查非空字段
                    {
                        MessageBox.Show("科目代码不能为空","提示",MessageBoxButtons.OK,Message
BoxIcon.Error);
                        return;
                    }
                    if(txt5.Text.Trim()=="")
                    {
                        MessageBox.Show("科目名称不能为空","提示",MessageBoxButtons.OK,Message
BoxIcon.Error);
                        return;
                    }
                    cmOrders.EndCurrentEdit();
                    if(dataSet11.GetChanges()!=null)
                    {
                        try
                        {
                            da1.Update(dataSet11);
                            SetModifyMode(false);
                        }
                        catch(Exception express)
                        {
                            MessageBox.Show(express.ToString(),"提示",MessageBoxButtons.OK,Message
BoxIcon.Error);
                            dataSet11.RejectChanges();
                        }
                    }
                    return;
                }
                if (e.Button.ToolTipText == "取消")
                {
                try
                {
                    cmOrders.CancelCurrentEdit();   //取消编辑
                    SetModifyMode(false);
                }
                catch(Exception express)
                {
                    MessageBox.Show(express.ToString(),"提示",MessageBoxButtons.OK,MessageBoxIcon.Error);
                }
                return;
            }
            if(e.Button.ToolTipText=="退出")
```

```
        {
            if(dataSet11.HasChanges())
            {
                DialogResult result=MessageBox.Show("数据集有被修改但尚未提交的数据，是否提交？
","确认",MessageBoxButtons.OKCancel);
                if(result==DialogResult.OK)
                    da1.Update(dataSet11);
            }
            this.Close();
          }
        }
```

（4）其他代码。添加如下代码：

```
        //--------------对控件的属性做设置----------------
        private void SetModifyMode(bool blnEdit)
        {
            //设置文本框和下拉列表框属性
            txt4.ReadOnly=!blnEdit;
            txt5.ReadOnly=!blnEdit;
            txt6.ReadOnly=!blnEdit;
            comboBox1.Enabled=blnEdit;
            comboBox2.Enabled=blnEdit;
            //设置搜索按钮属性
            btnSearch.Enabled=!blnEdit;
        }
```

7.3.5 账户设置窗体

1．窗体界面设计

（1）设计窗体外观。添加新窗体 Form1，在“设计”的 name 项中修改名称为“AccountSetting”。单击“外观”→BackgroundImage 后的按钮选择背景图片。在 Text 后添加“账户设置”，这是要在窗体边框上显示的文本。

（2）添加按钮。添加三个按钮，Name 分别设置为“btnNewAccount”、“btnUseAccount”、“btnBalance”，Text 分别设置为“建立新账簿”、“启用新账簿”、“试算平衡”。

（3）添加 DataGrid。添加两个 DataGrid，分别用于显示表“账簿初始化表”和“系统参数表”中的数据。

（4）添加 SqlDataAdapter。左键双击“工具箱”中“数据”下的 SqlDataAdapter，打开“数据适配器配置向导”。

① 单击【下一步】按钮，选择数据库“caiwubook”。

② 单击【下一步】按钮，单击【查询生成器】按钮，选择表“账簿初始化表”，在表中在选择“所有列”，单击【确定】按钮。

③ 单击【下一步】按钮，完成数据配置。

④ 添加的 SqlDataAdapter 默认名称为 sqlDataAdapter1，修改 Name 中的名称为“da1”。

⑤ 同样的添加名为“da2”的 SqlDataAdapter。打开“数据适配器配置向导”，选择表“系统参数表”。

（5）添加 DataSet。右键单击 da1，选中“生成数据集”，在“选择数据集”选择“新建”，在“选择要添加到数据集中的表”中选择“科目表”，勾选“将此数据集添加到设计器”，单击【确定】按钮，就会自动生成 dataSet21。设计好的账户设置窗体界面如图 7.7 所示。

图 7.7　账户设置窗体

2．窗体代码设计

（1）界面初始化代码。左键双击窗体中无控件填充的部分，添加如下代码：

```
private void AccountSetting_Load(object sender, System.EventArgs e)
{
    da1.Fill(dataSet21);
    da2.Fill(dataSet21);
}
```

（2）【建立新账户】按钮代码。左键双击【建立新账户】按钮，添加如下代码：

```
//----------建立新账户----------
private void btnNewAccount_Click(object sender, System.EventArgs e)
{
    DialogResult  result=MessageBox.Show("是否清除原有账簿信息并初始化新账簿？","警告",MessageBoxButtons.OKCancel,MessageBoxIcon.Warning);
    if(result==DialogResult.OK)
    {
        string strConn = "workstation id=localhost;Integrated Security=SSPI; database=caiwubook";
        SqlConnection cn=new SqlConnection(strConn);
        cn.Open();
        SqlCommand cmd=cn.CreateCommand();
        cmd.CommandText="exec sf_初始化账户";
        try
        {
            cmd.ExecuteNonQuery();
```

```
                dataSet21.Clear();//刷新数据集
                da1.Fill(dataSet21);
                da2.Fill(dataSet21);
            }
            catch(Exception express)
            {
                MessageBox.Show(express.ToString(),"建立新账户失败", MessageBoxButtons.
OK,MessageBoxIcon.Error);
            }
        }
    }
```

（3）【启用新账户】按钮代码。左键双击【启用新账户】按钮，添加如下代码：

```
    //-----------启用新账户----------
    private void btnUseAccount_Click(object sender, System.EventArgs e)
    {
        da1.Update(dataSet21);//将用户对新账簿中数据的修改提交到数据库
        string strConn = "workstation id=localhost;Integrated Security=SSPI; database=caiwubook";
        SqlConnection cn=new SqlConnection(strConn);
        cn.Open();
        SqlCommand cmd=cn.CreateCommand();
        cmd.CommandText="select sum(累计借方) 借方,sum(累计贷方) 贷方 from 账簿初始化表";
        SqlDataReader dr=cmd.ExecuteReader();
        dr.Read();
        if(dr.GetValue(0).ToString().Trim()!=dr.GetValue(1).ToString().Trim())//判断初始化后账簿是否
平衡
        {
            MessageBox.Show("新账簿初始化不平衡，请检查试算平衡表后重新输入","无法启用新
账簿",MessageBoxButtons.OK,MessageBoxIcon.Error);
            return;
        }
        else
        {
            DialogResult result=MessageBox.Show("试算结果平衡，启用新账簿将删除所有历史记
录，是否继续？","确认启用新账簿",MessageBoxButtons.OKCancel,MessageBoxIcon.Warning);
            if(result==DialogResult.OK)
            {
                dr.Close();
                cmd.CommandText="exec sf_启用账户";
                cmd.ExecuteNonQuery();
                dataSet21.Clear();//刷新数据集
                da1.Fill(dataSet21);
                da2.Fill(dataSet21);
                MessageBox.Show("新账户启用成功!");
            }
            return;
        }
```

```
}
```

（4）【试算平衡】按钮代码。左键双击【试算平衡】按钮，添加如下代码：

```
//----------新账户的试算平衡--------
private void btnBalance_Click(object sender, System.EventArgs e)
{
    da1.Update(dataSet21);//将用户对新账簿中数据的修改提交到数据库
    TrialBalance newFrm=new TrialBalance(true);
    newFrm.Text+="初始化账户";
    newFrm.Show();
}
```

7.3.6 会计凭证输入窗体

1．窗体界面设计

（1）设计窗体外观。添加新窗体 Form1，在“设计”的 name 项中修改名称为“WarrentInput”。单击“外观”下的 BackgroundImage 后的按钮选择背景图片。在 Text 后添加“会计凭证输入”，这是要在窗体边框上显示的文本。

（2）添加工具栏。将“工具箱”中“Windows 窗体”中的 ImageList 拖曳到窗体中，其默认名称为“imageList1”，单击“外观”下的 Images 后的[…]，进入“Image 集合编辑器”，如图 7.4 所示为添加十个图标。

将“工具箱”中“Windows 窗体”中的 ToolBar 拖曳到窗体中，单击“行为”下的 ImageList 后的下拉菜单，会发现刚才添加的“imageList1”在它的选项中，选中它。单击“行为”下的 Buttons 后的［…］，进入“ToolBarButton 集合编辑器”，如图 7.4 所示添加十个成员。

（3）添加 SqlDataAdapter。左键双击“工具箱”中“数据”下的 SqlDataAdapter，打开“数据适配器配置向导”。

① 单击【下一步】按钮，选择数据库“caiwubook”。

② 单击【下一步】按钮，单击【查询生成器】按钮，选择表“凭证表”，在表中在选择“所有列”，单击【确定】按钮。

图 7.8 会计凭证输入窗体

③ 单击【下一步】按钮，完成数据配置。

④ 添加的 SqlDataAdapter 默认名称为 sqlDataAdapter1，修改 Name 中的名称为“da1”。

⑤ 同理配置数据“da2”，其中表选择为“分录表”和“科目表”。

（4）添加 DataSet。右键单击 da1，选中“生成数据集”，在“选择数据集”选择“新建”，在“选择要添加到数据集中的表”中选择“科目表”，勾选“将此数据集添加到设计器”，单击【确定】按钮，就会自动生成 dataSet1。

（5）其他。按照前面讲过的方法设计界面，如图 7.8 所示。

2．窗体代码设计

（1）界面初始化代码。左键双击窗体中无控件填充的部分，添加如下代码：

```
    //------------创建窗体时，读入全部数据------------
    private void WarrentInput_Load(object sender, System.EventArgs e)
    {
        da1.SelectCommand.Parameters[0].Value="%%";//设置参数
da1.SelectCommand.Parameters[1].SqlDbType=System.Data.SqlDbType.VarChar;
da1.SelectCommand.Parameters[2].SqlDbType=System.Data.SqlDbType.VarChar;
        da1.SelectCommand.Parameters[1].Value="%%";
        da1.SelectCommand.Parameters[2].Value="%%";
        da1.Fill(dataSet11.凭证表);
        DataGridStateControl();//设置分录表表格
        //最初显示时，分录表显示第一个凭证记录的明细信息
        da2.SelectCommand.Parameters[0].Value=txt4.Text.Trim();
        newTable.Clear();
        da2.Fill(newTable);
        dataGrid2.ReadOnly=true;
        //创建窗体时处于浏览状态，不允许双击分录表添加记录
        this.enableDoubleClick=false;
        //绑定数据导航控制
        cmOrders=(CurrencyManager) BindingContext[dataSet11,"凭证表"];          }
```

（2）【搜索】按钮代码。左键双击【搜索】按钮，添加如下代码：

```
    //------------根据输入信息来检索数据-------------
    private void btnSearch_Click(object sender, System.EventArgs e)
    {
        da1.SelectCommand.Parameters[0].Value="%%";
        da1.SelectCommand.Parameters[1].Value="%%";
        da1.SelectCommand.Parameters[2].Value="%%";
        //根据用户在文本框中的输入来设置 SQL 查询的参数
        if(txt1.Text.Trim()!="")
        {
            da1.SelectCommand.Parameters[0].Value="%"+txt1.Text.Trim()+"%";
        }
        if(txt2.Text.Trim()!="")
        {
            da1.SelectCommand.Parameters[1].Value="%"+txt2.Text.Trim()+"%";
        }
        if(txt3.Text.Trim()!="")
        {
            da1.SelectCommand.Parameters[2].Value="%"+txt3.Text.Trim()+"%";
        }
        //清空数据表，并根据新设置的查询参数重新填充
        dataSet11.凭证表.Clear();
        da1.Fill(dataSet11);
```

```
            //显示分录数据
            da2.SelectCommand.Parameters[0].Value=txt4.Text.Trim();
            newTable.Clear();
            da2.Fill(newTable);
            dataGrid2.ReadOnly=true;
        }
```

（3）工具条“ToolBar”内的按钮使用代码。左键双击“ToolBar”工具条添加如下代码：

```
        //--------------工具栏事务处理----------------
        private void toolBar1_ButtonClick(object sender, System.Windows.Forms. ToolBarButtonClickEvent
Args e)
        {
            if (e.Button.ToolTipText == "首记录")
            {
                this.dataGrid1.UnSelect(cmOrders.Position); //取消原选中的行
                cmOrders.Position = 0;
                this.dataGrid1.Select(cmOrders.Position); //选中当前行
                this.dataGrid1.CurrentRowIndex = cmOrders.Position; //移动表头指示图标
                return;
            }
            if (e.Button.ToolTipText == "上一记录")
            {
                if (cmOrders.Position >= 0)
                {
                    this.dataGrid1.UnSelect(cmOrders.Position);
                    cmOrders.Position--;
                    this.dataGrid1.Select(cmOrders.Position);
                    this.dataGrid1.CurrentRowIndex = cmOrders.Position;
                }
                return;
            }
            if (e.Button.ToolTipText == "下一记录")
            {
                if (cmOrders.Position <= cmOrders.Count-1)
                {
                    this.dataGrid1.UnSelect(cmOrders.Position);
                    cmOrders.Position++;
                    this.dataGrid1.Select(cmOrders.Position);
                    this.dataGrid1.CurrentRowIndex = cmOrders.Position;
                }
                return;
            }
            if (e.Button.ToolTipText == "尾记录")
            {
                this.dataGrid1.UnSelect(cmOrders.Position);
                cmOrders.Position = cmOrders.Count-1;
```

```
            this.dataGrid1.Select(cmOrders.Position);
            this.dataGrid1.CurrentRowIndex = cmOrders.Position;
            return;
        }
        if(e.Button.ToolTipText=="新增")
        {
            cmOrders.AddNew();
            //设置按钮
            SetModifyMode(true);
            //设置默认值
            //查询当前会计期间
            string strConn = "workstation id=localhost;Integrated Security=SSPI; database=caiwubook";
            SqlConnection cn=new SqlConnection(strConn);
            cn.Open();
            SqlCommand cmd=cn.CreateCommand();
            cmd.CommandText="select 取值 from 系统参数表 where 参数名称='当前会计期间'";
            txt7.Text=cmd.ExecuteScalar().ToString();//读入当前会计期间
            //自动计算最大编号
            cmd.CommandText="select max(凭证编号) 最大编号 from 凭证表";
            object maxResult=cmd.ExecuteScalar();
            int maxID=1;
            if(maxResult!=System.DBNull.Value)
            //如果当前凭证表为空，则新编号为 000001
                maxID=Convert.ToInt32(maxResult)+1;
            int length=maxID.ToString().Length;
            switch(length)
            {
                case 1:
                    txt4.Text="00000"+maxID.ToString();
                    break;
                case 2:
                    txt4.Text="0000"+maxID.ToString();
                    break;
                case 3:
                    txt4.Text="000"+maxID.ToString();
                    break;
                case 4:
                    txt4.Text="00"+maxID.ToString();
                    break;
                case 5:
                    txt4.Text="0"+maxID.ToString();
                    break;
            }
            txt5.Text=DateTime.Now.ToShortDateString();//当前时间
            txt8.Text="未过";//新增凭证的状态都是"未过"
            comboBox1.SelectedText="收";//设置凭证字号
            //允许双击分录表表格添加分录数据
```

```
                this.enableDoubleClick=true;
                dataGrid2.ReadOnly=false;
                DataGridStateControl();
                newTable.Clear();
            }
            if(e.Button.ToolTipText=="修改")
            {
                SetModifyMode(true);
                this.enableDoubleClick=true;
                dataGrid2.ReadOnly=false;
            }
            if(e.Button.ToolTipText=="删除")
            {
                DialogResult result=MessageBox.Show("将删除凭证记录以及下属的分录信息，是否确
认？","删除数据",MessageBoxButtons.OKCancel);
                if(result==DialogResult.OK)
                    if(cmOrders.Count>0)
                    {
                        try
                        {
                            string strConn = "workstation id=localhost;Integrated Security=SSPI;database=
caiwubook";
                            SqlConnection cn=new SqlConnection(strConn);
                            cn.Open();
                            SqlCommand cmd=cn.CreateCommand();
                            cmd.CommandText="delete from 分录表 where 凭证编号='"+txt4.Text+
"';//删除分录记录
                            cmd.ExecuteNonQuery();
                            cmOrders.RemoveAt(cmOrders.Position);
                            da1.Update(dataSet11);
                        }
                        catch(Exception express)
                        {
                        MessageBox.Show(express.ToString(),"错误",MessageBoxButtons.OK,Message
BoxIcon.Error);
                            return;
                        }
                    }
                    else
                        MessageBox.Show("表中为空，已无可删除数据","提示",MessageBoxButtons.
OK,MessageBoxIcon.Error);
            }
            if(e.Button.ToolTipText=="提交")
            {
                if(txt9.Text!=txt10.Text)
                {
                    MessageBox.Show("借贷双方不平衡，请检查并修改分录数据后重新输入","错误
```

```
",MessageBoxButtons.OK,MessageBoxIcon.Warning);
                    return;
                }
                cmOrders.EndCurrentEdit();//结束当前编辑操作并提交修改
                int rowCount=newTable.Rows.Count;
                if(rowCount==0)
                {
                    MessageBox.Show("请先输入凭证表分录数据后再保存","警告",MessageBoxButtons.
OK,MessageBoxIcon.Error);
                    return;
                }
                try
                {
                    if (dataSet11.GetChanges()!=null)
                    {
                        this.da1.Update(dataSet11);//首先，先保存凭证信息
                    }
                    string strConn = "workstation id=localhost;Integrated Security=SSPI; database=caiwubook";
                    SqlConnection cn=new SqlConnection(strConn);
                    cn.Open();
                    SqlCommand cmd=cn.CreateCommand();
                    cmd.CommandText="delete from 分录表 where 凭证编号='"+txt4.Text+"'";
                    cmd.ExecuteNonQuery();//其次，再将该凭证信息对应的分录记录删除
                    for(int i=0;i<rowCount;i++)//最后，再逐行重新插入分录数据
                    {
                        if(dataGrid2[i,0].ToString().Trim()!="")
                        //如该行数据为空，则不保存该行
                        {
                            cmd.CommandText="insert into 分录表([凭证编号], [摘要], [科目代码],
[借方], [贷方]) " +"values('"+txt4.Text+"','"+dataGrid2[i,4].ToString()+"', "+dataGrid2[i,0].ToString()+"',CAST('"+
dataGrid2[i,2].ToString()+"'as money),CAST('"+dataGrid2[i,3].ToString()+"' as money))";
                            cmd.ExecuteNonQuery();
                        }
                    }
                    this.enableDoubleClick=false;
                }
                catch(Exception express)
                {
                    MessageBox.Show(express.ToString(),"提示",MessageBoxButtons.OK, MessageBoxIcon.
Error);
                    dataSet11.RejectChanges();
                }
                SetModifyMode(false);
                return;
            }
            if (e.Button.ToolTipText == "取消")
            {
```

```
                try
                {
                    cmOrders.CancelCurrentEdit();  //取消编辑
                    SetModifyMode(false);
                    if(txt4.Text.Trim()!="")//恢复显示明细信息
                    {
                        da2.SelectCommand.Parameters[0].Value=txt4.Text.Trim();
                        newTable.Clear();
                        da2.Fill(newTable);
                        dataGrid2.ReadOnly=true;
                    }
                }
                catch(Exception express)
                {
                    MessageBox.Show(express.ToString(),"提示",MessageBoxButtons.OK, MessageBoxIcon.
Error);
                }
                return;
            }
            if(e.Button.ToolTipText=="退出")
            {
                this.Close();
            }
        }
```

（4）设置明细表格各列的属性。添加如下代码：

```
        private void DataGridStateControl()
        {
            newTable = new DataTable();
            newTable.Columns.Add("科目代码",typeof(string));
            newTable.Columns.Add("科目名称",typeof(string));
            newTable.Columns.Add("借方",typeof(decimal));
            newTable.Columns.Add("贷方",typeof(decimal));
            newTable.Columns.Add("摘要",typeof(string));
            this.dataGrid2.DataSource = newTable;
            newTable.Rows.Add(newTable.NewRow());
            //向表中添加一行
            DataGridTableStyle ts = new DataGridTableStyle();
            DataGridTextBoxColumn aColumnTextColumn;
            ts.AllowSorting = false;
            ts.AlternatingBackColor = Color.LightGray;
            ts.MappingName = newTable.TableName;
            int numCols = newTable.Columns.Count;
            for (int i = 0;i< numCols;i++)
            {
                aColumnTextColumn = new DataGridTextBoxColumn();
```

```
                    if( i==0 || i==1 )//表中允许编辑【借方】，【贷方】和【备注】三个字段
                    {
                        aColumnTextColumn.ReadOnly=true;
                    }
                    if (   i == 1 )//当鼠标单击第 1 列时，允许响应
                    {
                        aColumnTextColumn.TextBox.MouseDown += new MouseEventHandler(TextBoxMouse
DownHandler);
                    }
                    if(i==1)
                    {
                        aColumnTextColumn.Width=100;//设置科目名称宽度
                    }
                    if ( i == 4 )
                    {
                        aColumnTextColumn.Width = 160;//设置宽度
                    }
                    aColumnTextColumn.MappingName = newTable.Columns[i].ColumnName;
                    aColumnTextColumn.HeaderText = newTable.Columns[i].ColumnName;
                    aColumnTextColumn.NullText = "";
                    aColumnTextColumn.Format = "N"; //设置为数字格式显示
                    ts.GridColumnStyles.Add(aColumnTextColumn);
                }
                dataGrid2.TableStyles.Add(ts);
            }
```

（5）根据指针指向的凭证表数据显示其分录表数据。添加如下代码：

```
        private void dataGrid1_CurrentCellChanged(object sender, System.EventArgs e)
        {
            da2.SelectCommand.Parameters[0].Value=txt4.Text.Trim();
            newTable.Clear();
            da2.Fill(newTable);
            dataGrid2.ReadOnly=true;
        }
```

（6）确定鼠标在表格中的单击模式。添加如下代码：

```
        private void dataGrid2_MouseDown(object sender, System.Windows.Forms. MouseEventArgs e)
        {
            //获取当前单击鼠标时的时间
            DataGrid myGrid      = (DataGrid)sender;
            System.Windows.Forms.DataGrid.HitTestInfo myHitTest;
            myHitTest = dataGrid2.HitTest(e.X,e.Y);
            if ( myHitTest.Type == System.Windows.Forms.DataGrid.HitTestType.Cell )
                gridMouseDownTime = DateTime.Now;
        }
```

（7）处理鼠标在 dataGrid 上的单击事件。添加如下代码：

```
private void TextBoxMouseDownHandler(object sender, MouseEventArgs e )
{
    //第一个判断条件：在单元格的 textbox 中的双击(即单击 DataCell,使它获得焦点后,然后再响应该 Cell 中的双击事件)
    //第二个判断条件：DataGrid 的 DoubleClick(直接双击 Cell 就响应双击事件,无须先单击 Cell,使它获得焦点,然后再响应双击事件)
    //判断时间间隔是否小于控制面板中所定义的双击间隔时间
    if((e.Button == MouseButtons.Left && e.Clicks == 2 || DateTime.Now < gridMouseDownTime.AddMilliseconds(SystemInformation.DoubleClickTime))&&(this.enableDoubleClick==true))
    {
        this.doubleClicked();
    }
}
```

（8）双击分录表表格，添加新的分录数据。添加如下代码：

```
private void doubleClicked()
{
    newTable.Rows.Add(newTable.NewRow()); //向表中添加一行，保持总有一个新行
    SelectSubject newFrm=new SelectSubject();
    newFrm.ShowDialog();
    int rowNumber=dataGrid2.CurrentCell.RowNumber;
    dataGrid2[rowNumber,0]=mID;
    dataGrid2[rowNumber,1]=mName;
    dataGrid2[rowNumber,2]="0";
    dataGrid2[rowNumber,3]="0";
    SendKeys.Send("{Tab}");//向活动应用程序发送 Tab 键,跳到下一控
}
```

（9）计算输入的分录表借贷总额。添加如下代码：

```
private void dataGrid2_CurrentCellChanged(object sender, System.EventArgs e)
{
    if(this.enableDoubleClick==true)
    {
        int rowCount=newTable.Rows.Count;
        decimal debit=0;
        decimal loan=0;
        for(int i=0;i<rowCount;i++)
        {
            if(dataGrid2[i,2]!=DBNull.Value)
            {
                debit+=Convert.ToDecimal(dataGrid2[i,2]);
                loan+=Convert.ToDecimal(dataGrid2[i,3]);
            }
        }
        txt9.Text=debit.ToString();
```

```
                txt10.Text=loan.ToString();
            }
        }
```

（10）对控件的属性做设置。添加如下代码：

```
        private void SetModifyMode(bool blnEdit)
        {
            //设置文本框和下拉列表框属性
            txt4.ReadOnly=!blnEdit;
            txt5.ReadOnly=!blnEdit;
            comboBox1.Enabled=blnEdit;
            //设置搜索按钮属性
            btnSearch.Enabled=!blnEdit;
            //设置两个表格的 ReadOnly 属性
            dataGrid1.ReadOnly=!blnEdit;
            dataGrid2.ReadOnly=!blnEdit;
        }
```

7.3.7　总分类账查询窗体

1．窗体界面设计

（1）设计窗体外观。添加新窗体 Form1，在“设计”的 name 项中修改名称为“LedgerQuery”。单击“外观”→BackgroundImage 后的按钮选择背景图片。在 Text 后添加“总分类账查询”，这是要在窗体边框上显示的文本。

（2）添加【搜索】按钮。添加【搜索】按钮，将其命名为“btnSearch”。

（3）添加凭证编号输入框。使用 Label 和 TextBox 实现完成凭证编号输入框，并将 TextBox 的 Name 属性设为“txt1”，Text 属性设为空。

（4）添加选项“无发生余额不显示”。左键双击“工具箱”中 CheckBox，将 Text 属性设为“无发生余额不显示”。

（5）添加 SqlDataAdapter。左键双击“工具箱”中“数据”下的 SqlDataAdapter，打开“数据适配器配置向导”。

① 单击【下一步】按钮，选择数据库“caiwubook”。

② 单击【下一步】按钮，单击【查询生成器】按钮，选择表“本期汇总账簿”和“科目表”，在表中选择“所有列”，单击【确定】按钮。

③ 单击【下一步】按钮，完成数据配置。

④ 添加的 SqlDataAdapter 默认名称为 sqlDataAdapter1，使用默认名称。

（6）添加 DataSet。右键单击 da1，选中“生成数据集”，在“选择数据集”选择“新建”，在“选择要添加到数据集中的表”中选择“科目表”，勾选“将此数据集添加到设计器”，单击【确定】按钮，就会自动生成 dataSet1。

（7）添加 DataView。左键双击“工具箱”中“数据”下的“DataView”，使用 Name 属性默认名称“dataView1”，Table 属性选择“dataSet11.本期汇总账簿”。设计好的界面如图 7.9 所示。

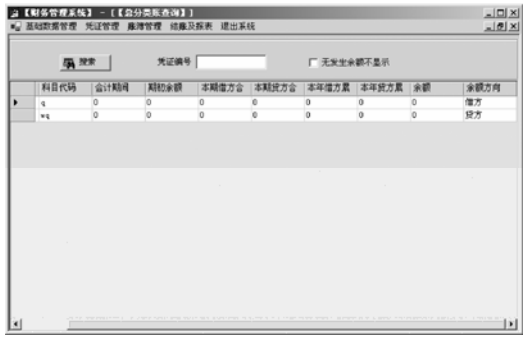


图 7.9　总分类账查询窗体

2．窗体代码设计

（1）初始化代码。左键双击窗体中无控件填充的部分，添加如下代码：

```
//----------显示窗口时读入全部数据--------------
private void LedgerQuery_Load(object sender, System.EventArgs e)
{
    sqlDataAdapter1.SelectCommand.Parameters[0].Value="%%";
    sqlDataAdapter1.Fill(dataSet11);
}
```

（2）【搜索】按钮代码。左键双击【搜索】按钮，添加如下代码：

```
//------------根据科目代码选择数据------------
private void btnSearch_Click(object sender, System.EventArgs e)
{
    sqlDataAdapter1.SelectCommand.Parameters[0].Value="%%";
    if(txt1.Text.Trim()!=null)
    {
        sqlDataAdapter1.SelectCommand.Parameters[0].Value="%"+txt1.Text.Trim()+"%";
        dataSet11.Clear();
        sqlDataAdapter1.Fill(dataSet11);
    }
}
```

（3）复选框“无发生金额不显示”代码。左键双击“无发生金额不显示”复选框添加如下代码：

```
//-----------根据有无发生金额筛选显示的数据------------
private void checkBox1_CheckedChanged(object sender, System.EventArgs e)
{
    if(checkBox1.Checked)
        dataView1.RowFilter="余额<>0";
    else
        dataView1.RowFilter="";
}
```

7.3.8 明细账窗体

1．窗体界面设计

（1）设计窗体外观。添加新窗体 Form1，在“设计”的 name 项中修改名称为“DetaiQuery”。单击“外观”→BackgroundImage 后的按钮选择背景图片。在 Text 后添加“明细账查询”，这是要在窗体边框上显示的文本。

（2）添加【搜索】按钮。添加【搜索】按钮，将其命名为“btnSearch”。

（3）添加科目代码输入框。使用 Label 和 TextBox 实现完成凭证编号输入框，并将 TextBox 的 Name 属性设为“txt1”，Text 属性设为空。

（4）添加选项“选择要显示在表中的项目”。左键双击“工具箱”中的 GroupBox，将 Text 属性设为“选择要显示在表中的项目”。左键双击“工具箱”中 ComboBox，将 Text 属性设为空。

（5）添加 SqlDataAdapter。左键双击“工具箱”中“数据”下的 SqlDataAdapter，打开“数据适配器配置向导”。

① 单击【下一步】按钮，选择数据库“caiwubook”。

② 单击【下一步】按钮，单击【查询生成器】按钮，选择表“本期明细账簿”和“科目表”，在表中在选择“所有列”，单击【确定】按钮。

③ 单击【下一步】按钮，完成数据配置。

④ 添加的 SqlDataAdapter 默认名称为 sqlDataAdapter1，使用默认名称。

（6）添加 DataSet。右键单击 da1，选中“生成数据集”，在“选择数据集”选择“新建”，在“选择要添加到数据集中的表”中选择“科目表”，勾选“将此数据集添加到设计器”，单击【确定】按钮，就会自动生成 dataSet11。

（7）添加 DataView。左键双击“工具箱”中“数据”下的“DataView”，使用 Name 属性默认名称“dataView1”，Table 属性选择“dataSet1.本期明细账簿”。设计好的界面如图 7.10 所示。

图 7.10 明细账查询窗体

2．窗体代码设计

（1）初始化代码。左键双击窗体中无控件填充的部分，添加如下代码：

```
//----------显示窗口时读入全部数据--------------
private void DetailQuery_Load(object sender, System.EventArgs e)
{
```

```
            sqlDataAdapter1.SelectCommand.Parameters[0].Value="%%";
            sqlDataAdapter1.Fill(dataSet11);
            comboBox1.Items.Add("全部数据");//添加选择显示全部数据的选项
            foreach(DataRow aRow in dataSet11.本期明细账簿.Rows)//根据明细表中的科目名称，设置下拉字典
            {
                string newStr=aRow["科目名称"].ToString().Trim();
                comboBox1.Items.Add(newStr);
            }
        }
```

（2）【搜索】按钮代码。左键双击【搜索】按钮，添加如下代码：

```
        //------------根据科目代码选择数据------------
        private void btnSearch_Click(object sender, System.EventArgs e)
        {
            sqlDataAdapter1.SelectCommand.Parameters[0].Value="%%";
            if(txt1.Text.Trim()!=null)
            {
                sqlDataAdapter1.SelectCommand.Parameters[0].Value="%"+txt1.Text.Trim()+"%";
                dataSet11.Clear();
                sqlDataAdapter1.Fill(dataSet11);
                //重新设置下拉选项
                comboBox1.Items.Clear();
                comboBox1.Items.Add("全部数据");
                foreach(DataRow aRow in dataSet11.本期明细账簿.Rows)
                {
                    string newStr=aRow["科目名称"].ToString().Trim();
                    comboBox1.Items.Add(newStr);
                }
            }
        }
```

（3）下拉菜单“comboBox1”代码。左键双击“comboBox1”下拉菜单添加如下代码：

```
        //------------筛选在表中显示的项目------------
        private void comboBox1_SelectedIndexChanged(object sender, System.EventArgs e)
        {
            if(comboBox1.SelectedIndex==0)//若选中显示全部数据则全部显示
            {
                dataView1.RowFilter="";
            }
            else
            {
                dataView1.RowFilter="科目名称='"+comboBox1.Text+"'";//否则根据选中的科目名称进行选择
            }
        }
```

7.3.9 资产负债表窗体

1．窗体界面设计

（1）设计窗体外观。添加新窗体 Form1，在“设计”的 name 项中修改名称为“Financial Report”。单击“外观”→BackgroundImage 后的按钮选择背景图片。在 Text 后添加“资产负债表”，这是要在窗体边框上显示的文本。

（2）添加【显示报表】按钮。添加一个 Button 按钮，将 Name 属性设为“显示报表”，Text 属性设为“btnSummary”。

（3）添加“选择要统计的会计期间”输入框。添加一个 Label，将 Text 属性设为“选择要统计的会计期间”。左键双击“工具箱”中的“NumericUpDown”，设置 Value 属性为“1”。

（4）添加显示报表控件“CrystalReportViewer”。左键双击“工具箱”中的“CrystalReport Viewer”。

按照前面讲过的方法设计界面，设计好的界面如图 7.11 所示。

图 7.11 资产负债表窗体

2．窗体代码设计

（1）初始化代码。左键双击窗体中无控件填充的部分，添加如下代码：

```
//------------读入系统参数中的当前会计期间-----------
private void FinancialReport_Load(object sender, System.EventArgs e)
{
    string conStr="workstation id=localhost;Integrated Security=SSPI;Database=caiwubook;";
    SqlConnection cn=new SqlConnection(conStr);
    cn.Open();
    //读入当前会计期间
    SqlCommand cmd=cn.CreateCommand();
    cmd.CommandText="select 取值 from 系统参数表 where 编号='3'";
    this.numericUpDown1.Value=Convert.ToDecimal(cmd.ExecuteScalar());
}
```

（2）【显示报表】按钮代码。左键双击【显示报表】按钮，添加如下代码：

```
//----------显示所选择的会计期间的报表--------------
private void btnSummary_Click(object sender, System.EventArgs e)
{
    string conStr="workstation id=localhost;Integrated Security=SSPI;Database=caiwubook;";
    SqlConnection cn=new SqlConnection(conStr);
    cn.Open();
    //调用储存过程，计算当前会计期间的资产负债表
    SqlCommand cmd=cn.CreateCommand();
    int index=Convert.ToInt32(this.numericUpDown1.Value);
    cmd.CommandText="exec sf_计算资产负债表 "+index.ToString();
    cmd.ExecuteNonQuery();
    //显示资产负债报表
    string sql="select * from 资产负债表 where 会计期间='"+index.ToString()+"'";
    SqlDataAdapter da=new SqlDataAdapter(sql,cn);
    DataSet ds=new DataSet();
    da.Fill(ds);
    CrystalReport1 rpt=new CrystalReport1();
    //连接报表数据源
    rpt.SetDataSource(ds.Tables[0]);
    crystalReportViewer1.ReportSource=rpt;
}
```

7.3.10 试算平衡表窗体

1．窗体界面设计

试算平衡表有两个 DataGrid 和一个 GroupBox 组成，设计好的试算平衡表窗体如图 7.12 所示。

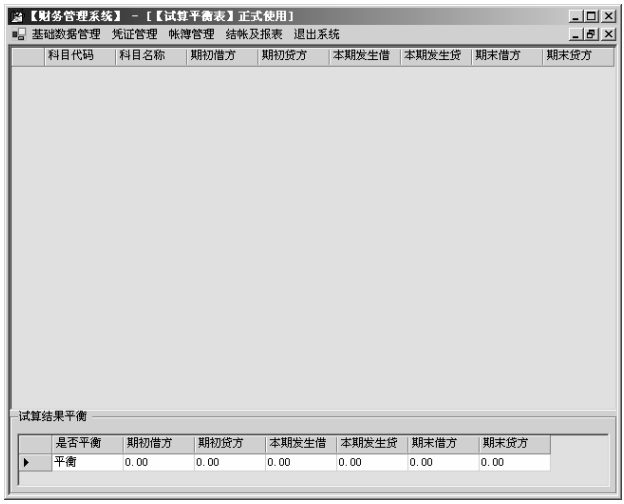


图 7.12 试算平衡表窗体

2．窗体代码设计

（1）初始化代码。左键双击窗体中无控件填充的部分，添加如下代码：

```
//-------------创建窗体时显示数据-----------
private void TrialBalance_Load(object sender, System.EventArgs e)
{
    string strConn = "workstation id=localhost;Integrated Security=SSPI; database=caiwubook";
    SqlConnection cn=new SqlConnection(strConn);
    cn.Open();
    SqlDataAdapter da=new SqlDataAdapter(sqlStr,cn);
    DataTable newTable=new DataTable();
    da.Fill(newTable);
    dataGrid1.DataSource=newTable;
    this.sumTable(newTable);
}
```

（2）统计数据，判断是否平衡。添加如下代码：

```
private void sumTable(DataTable tbl)
{
    decimal iniDebit=0;//期初借方
    decimal iniLoan=0;//期初贷方
    decimal thisDebit=0;//本期发生借方
    decimal thisLoan=0;//本期发生贷方
    decimal finDebit=0;//期末借方
    decimal finLoan=0;//期末贷方
    string isBalance="平衡";
    if(tbl.Rows.Count>0)
    {
        foreach(DataRow aRow in tbl.Rows)
        {
            iniDebit+=Convert.ToDecimal(aRow["期初借方"]);
            iniLoan+=Convert.ToDecimal(aRow["期初贷方"]);
            thisDebit+=Convert.ToDecimal(aRow["本期发生借方"]);
            thisLoan+=Convert.ToDecimal(aRow["本期发生贷方"]);
            finDebit+=Convert.ToDecimal(aRow["期末借方"]);
            finLoan+=Convert.ToDecimal(aRow["期末贷方"]);
        }
        if(iniDebit!=iniLoan||thisDebit!=thisLoan||finDebit!=finLoan)
            isBalance="不平衡";
    }
    DataTable sumTable = new DataTable();
    sumTable.Columns.Add("是否平衡",typeof(string));
    sumTable.Columns.Add("期初借方",typeof(decimal));
    sumTable.Columns.Add("期初贷方",typeof(decimal));
    sumTable.Columns.Add("本期发生借方",typeof(decimal));
    sumTable.Columns.Add("本期发生贷方",typeof(decimal));
```

```
            sumTable.Columns.Add("期末借方",typeof(decimal));
            sumTable.Columns.Add("期末贷方",typeof(decimal));
            this.dataGrid2.DataSource = sumTable;
            sumTable.Rows.Add(sumTable.NewRow()); //向表中添加一行
            //设置表格格式
            DataGridTableStyle ts = new DataGridTableStyle();
            DataGridTextBoxColumn aColumnTextColumn;
            ts.AllowSorting = false;
            ts.MappingName = sumTable.TableName;
            int numCols = sumTable.Columns.Count;
            for (int i = 0;i< numCols;i++)
            {
                aColumnTextColumn = new DataGridTextBoxColumn();
                aColumnTextColumn.MappingName = sumTable.Columns[i].ColumnName;
                aColumnTextColumn.HeaderText = sumTable.Columns[i].ColumnName;
                aColumnTextColumn.NullText = "";
                aColumnTextColumn.Format = "N"; //设置为数字格式显示
                ts.GridColumnStyles.Add(aColumnTextColumn);
            }
            dataGrid2.TableStyles.Add(ts);
            //将结果显示在表格中
            dataGrid2[0,0]=isBalance;
            dataGrid2[0,1]=iniDebit.ToString();
            dataGrid2[0,2]=iniLoan.ToString();
            dataGrid2[0,3]=thisDebit.ToString();
            dataGrid2[0,4]=thisLoan.ToString();
            dataGrid2[0,5]=finDebit.ToString();
            dataGrid2[0,6]=finLoan.ToString();
            if(isBalance=="不平衡")//如果试算结果不平衡，则用红色来提醒用户
            {
                groupBox1.Text="试算结果不平衡!";
                groupBox1.BackColor=Color.Red;
            }
            else
            {
                groupBox1.Text="试算结果平衡";
            }
        }
```

7.4 实训：人力资源管理系统软件

1．项目要求

编写人力资源管理系统软件，可以方便地管理企业人员的录用与解聘并且为每一个员工计算工资，功能要求如下：

（1）职员信息管理：职员信息维护与查询。其中职员信息包括：职员编号、性别、民族、籍贯、出生日期、年龄、文化程度、毕业院校、健康状况、婚姻状况、身份证号、家庭电话、办公室电话、手机号码。

（2）薪资福利管理：包括当月工资管理、工资发放历史。其中工资计算公式为：

月工资＝日工资×天数＋加班费＋满勤奖＋补助－保险－个人所得税

其中满勤奖和补助只有满勤人员才能获得；保险为个人缴纳部分，每月取整浮动；个人所得税按国家规定的计算，税率表如表 7.1 所示。

表 7.1　税率表

级数	含税级距	税率（%）	速算扣除数	说明
1	不超过 500 元的	5	0	本表含税级距指以每月收入额减除费用两千元后的余额
2	超过 500 元至 2000 元的部分	10	25	
3	超过 2000 元至 5000 元的部分	15	125	
4	超过 5000 元至 20000 元的部分	20	375	

2．设计提示

仿照本章内容链接 SQL 数据库。窗体要有登录窗体、主窗体、职员信息窗体、薪资福利窗体。

3．项目评价

项目评价是在教师的主持下，通过项目负责人的讲解演示，评估项目的完成情况，评价内容如下：

（1）工具类软件界面是否简洁，操作是否简便；

（2）使用过程中有无错误提示；

（3）能否实现基本功能。

习　　题

1．填空

（1）Windows窗体 ToolBar控件在窗体中用做_____，用以显示一列_____和可激活命令的_____。

（2）Windows 窗体 ImageList 组件用于_____，这些图像随后可由控件显示。

（3）Connection 对象表示_____。

（4）SqlDataAdapter 表示用于填充_____和更新_____的一组数据命令和一个数据库连接。

（5）SqlCommand 表示要对 SQL Server 数据库执行的一个_____或_____。

（6）DataView表示用于_____、_____、_____、_____和_____的 DataTable的可绑定数据的自定义视图。

2．选择

（1）ToolBar 控件的图像由一个关联的（　　）组件提供。

A．TextBox　　B．ImageList　　C．Picture　　D．Image

（2）关于复选框（CheckBox）控件，下列描述正确的是（　　）。

A．一次只能选择一个按钮

B．用于指示用户所选的选项

C．可以选择任意数量的按钮

D．通过单击向上和向下按钮、按向上键和向下键或输入一个数字来增大和减小数字

3．复选框控件和单选按钮控件的异同?

4．Windows窗体NumericUpDown控件的作用?

部分习题答案

第 1 章

1.（1）C　C++　（2）高生产率　行动力　（3）断点

2.（1）ABC　（2）ACD　（3）B　（4）C　（5）BCD

第 2 章

1.（1）Text　换到下一行　剪裁文本　（2）Windows　（3）ADO.NET

2.（1）ABD　（2）A　（3）BCD

5.（1）如果条件 y > 20 求得 false 值，则显示 Statement_2。（2）Statement_2 与条件 x >10 关联，条件 x > 10 求得 false 值时，将显示 Statement_2。

6．程序代码的功能：检查输入字符是否是小写字符、大写字符或数字，如果都不是，就显示不是字母字符

第 3 章

1.（1）换行　回车　（2）串行端口传输　接收数据　（3）BorderStyle　Fixed3D　ScrollBars　Vertical　Multiline　True　ReadOnly　True

2.（1）B　（2）A

6．程序代码的输出为：1　2　3　4　5

7．程序代码实现的功能：如果串口打开时发生错误，则提示：串口操作失败

第 4 章

1.（1）AutoScroll　（2）为其他控件提供可识别的分组　Text　也会一起　（3）GroupBox　Panel　（4）Random　（5）TabPages　（6）在二维平面中定义点的整数 x 和 y 坐标

2．该段代码折叠后显示：# region　MyClass definition

3．代码的输出为：

```
Current value of n is 1
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5
```

第 5 章

1.（1）时间　（2）菜单　（3）暂时存储数据　（4）不会将其关闭　Close　Dispose

2.（1）ABCD　　（2）ABCD

第 6 章

1.（1）绑定并显示　　（2）设计时预览报表　　（3）相同数据源和数据成员　　（4）delete 语句

2.（1）ABC　　（2）C

3. 输入：2

```
代码的输出：
Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

第 7 章

1.（1）窗体上的控制条　　下拉菜单　　位图按钮　　（2）存储一个图像的集合　　（3）创建并打开数据库连接　　（4）DataSet　　SQL Server数据库　　（5）Transact-SQL 语句　存储过程　　（6）排序　　筛选　　搜索　　编辑　　导航

2.（1）B　　（2）BC

参 考 文 献

1．刘甫迎，刘光会，王蓉，蒋建强编著．C#程序设计教程．北京：电子工业出版社，2007.2
2．孙永强主编．Visual C#.NET 入门与提高．北京：清华大学出版社，2002.8
3．郭胜主编．C#.NET 程序设计教程．北京：清华大学出版社，2002.9
4．明月创作室编著．Visual C#编程精彩百例．北京：人民邮电出版社，2001.9
5．王昊亮，李刚编著．Visual C#.程序设计教程．北京：清华大学出版社，2003.3
6．马骏编著．C#网络应用编程基础．北京：人民邮电出版社，2006.10
7．张怀庆编著. Visual C#.NET 编程精粹 150 例．北京：冶金工业出版社，2005.9
8．[美] Harvey M Deitel著，须德等译．C#大学教程．北京：电子工业出版社，2004.1

反侵权盗版声明