高职高专
软件专业系列规划教材

Windows程序设计案例教程(C#)

新世纪高职高专教材编审委员会组编
主编 李德奇

大连理工大学出版社

新世纪

高职高专计算机软件专业系列规划教材

Windows 程序设计案例教程(C#)

新世纪高职高专教材编审委员会组编

主　编　李德奇　　副主编　李蓓蓓　赵春霞　柴宝仁

WINDOWS CHENGXU SHEJI ANLI JIAOCHENG (C#)

大连理工大学出版社

DALIAN UNIVERSITY OF TECHNOLOGY PRESS

图书在版编目(CIP)数据

Windows 程序设计案例教程:C#/李德奇主编. —大连:
大连理工大学出版社,2008.1
(高职高专计算机软件专业系列规划教材)
ISBN 978-7-5611-3593-0

Ⅰ.W… Ⅱ.李… Ⅲ.①窗口软件,Windows—程序设计—
高等学校:技术学校—教材②C 语言—程序设计—高等学校:
技术学校—教材 Ⅳ.TP316.7 TP312

中国版本图书馆 CIP 数据核字(2008)第 001174 号

大连理工大学出版社出版
地址:大连市软件园路 80 号 邮政编码:116023
电话:0411-84708842 邮购:0411-84703636 传真:0411-84701466
E-mail:dutp@dutp.cn URL:http://www.dutp.cn
大连理工印刷有限公司印刷 大连理工大学出版社发行

幅面尺寸:185mm×260mm 印张:21.5 字数:451 千字
印数:1~5000
2007 年 11 月第 1 版 2008 年 1 月第 1 次印刷

责任编辑:付 亮 责任校对:曲容立
封面设计:苏儒光

ISBN 978-7-5611-3593-0 定 价:34.00 元

总序

我们已经进入了一个新的充满机遇与挑战的时代，我们已经跨入了21世纪的门槛。

20世纪与21世纪之交的中国，高等教育体制正经历着一场缓慢而深刻的革命，我们正在对传统的普通高等教育的培养目标与社会发展的现实需要不相适应的现状作历史性的反思与变革的尝试。

20世纪最后的几年里，高等职业教育的迅速崛起，是影响高等教育体制变革的一件大事。在短短的几年时间里，普通中专教育、普通高专教育全面转轨，以高等职业教育为主导的各种形式的培养应用型人才的教育发展到与普通高等教育等量齐观的地步，其来势之迅猛，发人深思。

无论是正在缓慢变革着的普通高等教育，还是迅速推进着的培养应用型人才的高职教育，都向我们提出了一个同样的严肃问题：中国的高等教育为谁服务，是为教育发展自身，还是为包括教育在内的大千社会？答案肯定而且惟一，那就是教育也置身其中的现实社会。

由此又引发出高等教育的目的问题。既然教育必须服务于社会，它就必须按照不同领域的社会需要来完成自己的教育过程。换言之，教育资源必须按照社会划分的各个专业（行业）领域（岗位群）的需要实施配置，这就是我们长期以来明乎其理而疏于力行的学以致用问题，这就是我们长期以来未能给予足够关注的教育目的问题。

如所周知，整个社会由其发展所需要的不同部门构成，包括公共管理部门如国家机构、基础建设部门如教育研究机构和各种实业部门如工业部门、商业部门，等等。每一个部门又可作更为具体的划分，直至同它所需要的各种专门人才相对应。教育如果不能按照实际需要完成各种专门人才培养的目标，就不能很好地完成社会分工所赋予它的使命，而教育作为社会分工的一种独立存在就应受到质疑（在市场经济条件下尤其如此）。可以断言，按照社会的各种不同需要培养各种直接有用人才，是教育体制变

革的终极目的。

随着教育体制变革的进一步深入,高等院校的设置是否会同社会对人才类型的不同需要一一对应,我们姑且不论。但高等教育走应用型人才培养的道路和走研究型(也是一种特殊应用)人才培养的道路,学生们根据自己的偏好各取所需,始终是一个理性运行的社会状态下高等教育正常发展的途径。

高等职业教育的崛起,既是高等教育体制变革的结果,也是高等教育体制变革的一个阶段性表征。它的进一步发展,必将极大地推进中国教育体制变革的进程。作为一种应用型人才培养的教育,它从专科层次起步,进而应用本科教育、应用硕士教育、应用博士教育……当应用型人才培养的渠道贯通之时,也许就是我们迎接中国教育体制变革的成功之日。从这一意义上说,高等职业教育的崛起,正是在为必然会取得最后成功的教育体制变革奠基。

高等职业教育还刚刚开始自己发展道路的探索过程,它要全面达到应用型人才培养的正常理性发展状态,直至可以和现存的(同时也正处在变革分化过程中的)研究型人才培养的教育并驾齐驱,还需假以时日;还需要政府教育主管部门的大力推进,需要人才需求市场的进一步完善发育,尤其需要高职高专教学单位及其直接相关部门肯于做长期的坚忍不拔的努力。新世纪高职高专教材编审委员会就是由全国 100 余所高职高专院校和出版单位组成的旨在以推动高职高专教材建设来推进高等职业教育这一变革过程的联盟共同体。

在宏观层面上,这个联盟始终会以推动高职高专教材的特色建设为己任,始终会从高职高专教学单位实际教学需要出发,以其对高职教育发展的前瞻性的总体把握,以其纵览全国高职高专教材市场需求的广阔视野,以其创新的理念与创新的运作模式,通过不断深化的教材建设过程,总结高职高专教学成果,探索高职高专教材建设规律。

在微观层面上,我们将充分依托众多高职高专院校联盟的互补优势和丰裕的人才资源优势,从每一个专业领域、每一种教材入手,突破传统的片面追求理论体系严整性的意识限制,努力凸现高职教育职业能力培养的本质特征,在不断构建特色教材建设体系的过程中,逐步形成自己的品牌优势。

新世纪高职高专教材编审委员会在推进高职高专教材建设事业的过程中,始终得到了各级教育主管部门以及各相关院校相关部门的热忱支持和积极参与,对此我们谨致深深谢意;也希望一切关注、参与高职教育发展的同道朋友,在共同推动高职教育发展、进而推动高等教育体制变革的进程中,和我们携手并肩,共同担负起这一具有开拓性挑战意义的历史重任。

新世纪高职高专教材编审委员会

2001 年 8 月 18 日

此为试读,需要完整PDF请访问: www.ertongbook.com

前言

自2000年7月Microsoft展示它的新一代软件开发工具——Visual Studio. NET以来，就备受IT业界的关注，在应用程序开发领域中开创了一个新时代。目前，学习和使用VS. NET的计算机软件从业人员和爱好者越来越多，高等学校计算机专业程序设计课程以C#作为教学工具也越来越普遍，可见其技术的先进性和旺盛的生命力。

本书以管理信息系统项目设计为主线，以项目中常见的功能窗体作为案例，详细介绍了基于C#的Windows应用程序设计的基础知识和编程技巧。主要内容包括C#. NET的基础语法、Windows窗体和控件的应用、ADO. NET数据库访问技术、多窗体项目的设计等。最后通过一个综合应用实例，详细讲述了使用C#. NET开发管理信息系统的过程和方法。

本书内容翔实，案例丰富，讲解透彻，注释详细，实用性强，便于读者对基于C#. NET的Windows应用程序设计的理解和应用。本书适合作为高职高专计算机相关专业程序设计课程的教材，也可作为从事软件开发的技术人员的参考书。

本书作为高职高专计算机相关专业程序设计课程的教材来编写，其指导思想是遵从近年来高职高专教育教学改革的方向，在以就业为导向的专业课程体系框架下，以提高学生的专业技能为课程教学目标。本教材的主要特点在于从软件行业程序员岗位的专业技能出发，以Windows应用程序项目贯穿于全书，以项目中常见案例作为教学单元，将C#编程的知识和技能融合到各个案例中，使学生通过典型案例的学习来掌握Windows程序设计的知识和技能，符合目前软件技术专业教学改革的主流。

本书主要介绍了. NET的概念、C#的基础语法、Windows窗体和控件的应用、ADO. NET数据库访问技术、常见功能窗体的设计方法、多窗体项目的整合等。最后通过

新世纪

一个综合应用实例,详细讲述了使用 C#.NET 开发管理信息系统的过程和技术。学习完本书的内容,读者可以掌握使用 C# 语言编写 Windows 桌面应用程序和管理信息系统软件的方法。全书共分 8 章,其结构安排如下:

第 1 章 C# 语言编程基础介绍了.NET 的基本概念,C# 的基础语法,C# 面向对象编程的基础知识,文件操作和异常处理的基本知识。

第 2 章 Windows 窗体基本控件应用 介绍了 Windows 窗体和基本控件的应用,学习 Windows 桌面应用程序的设计方法。

第 3 章窗口框架控件应用介绍了菜单、工具栏、状态栏、通用对话框的应用,通过一个文本编辑器案例综合应用了这些窗口框架控件。

第 4 章 ADO.NET 数据库访问技术介绍了 ADO.NET 的数据访问基类和数据存储基类,SQL 语言的基本知识,Windows 应用程序中访问数据库的方法。

第 5 章基本数据访问窗体设计详细介绍了管理信息系统中典型的功能窗体的设计方法,例如登录、修改密码、录入数据、查询数据、修改数据等。

第 6 章多窗格数据处理窗体设计详细介绍了多窗格数据处理窗体的设计方法,这些窗体带有典型的 Windows 界面,类似于 Windows 资源管理器的风格。

第 7 章多窗体项目设计介绍了多窗体项目中处理窗体间关系的方法,例如窗体间的数据交换、窗体的继承和拥有、窗体的父子关系等。

第 8 章图书馆管理系统设计 通过《图书馆管理系统》这个较完整的项目,介绍管理信息系统的分析和实现过程以及设计中常用的技术。

本书写作中融合了作者多年的软件开发和教学经验,加入了近年来对 Visual Studio.NET的理解和实践。本书力求做到深入浅出、联系实际,使读者通过案例来学习和掌握 C#.NET 这个先进的 Windows 应用程序开发工具。参加本书编写工作的有李德奇(主编),李蓓蓓(副主编),赵春霞(副主编),柴宝仁(副主编),薛志良,冯向科,郭外萍。由于水平和经验的局限,本书的不足和纰漏之处在所难免,恳请广大读者批评指正。

所有意见和建议请发往:gzjckfb@163.com
联系电话:0411-84707492 84706104

编 者

2008 年 1 月

湖南铁道职业技术学院信息工程系 李德奇

目　录

第1章 C#语言编程基础

本章要点

Microsoft C#是一种新的完全面向对象的编程语言，它是 Visual Studio. NET 开发工具的重要组成部分，也是 VS. NET 的首选语言。C#以语法结构严谨和开发效率高而著称，它的完全面向对象的语言特性，并非继承自 C/C++，它的数据类型更加安全，类的成员结构有了很大的扩展，面向对象的编程手段上更加灵活方便。C#是为生成运行在. NET Framework 上的广泛的企业级应用程序设计的，C#程序必须运行在. NET Framework 之上。为了顺利进入应用程序开发，本章就. NET Framework 框架的基本知识和 C#的编程基础做出简明的阐述，以求迅速了解 C#语言的概貌。

在这一章，将学习到：

- . NET Framework 框架的基本知识
- C#的数据类型、运算符和表达式
- C#的程序结构
- C#面向对象程序设计基础
- C#的异常处理机制
- C#的文件操作
- 使用 C#设计 Windows 应用程序的基本方法

1.1 . NET 框架和公共语言运行库

. NET 是. NET Framework 的简称，通常称为. NET 框架，它是 Microsoft 的一个新的程序运行平台。C#借助于. NET 运行库在. NET 平台上运行，其各种特性与. NET 密切相关。

1.1.1 . NET 结构

. NET 框架是一种托管的、类型安全的代码执行环境。. NET Framework 管理程序执行的各个方面：启动代码，给它赋予相应的权限，为数据和指令分配内存，对不再需要的内存进行回收管理。图 1.1 显示了. NET 的层次结构。

<table>
<tr><td colspan="4">C# VB. NET …</td></tr>
<tr><td colspan="4">ADO. NET, ASP. NET, Web Services, User Interface…</td></tr>
<tr><td colspan="4">. NET Framework Class Library</td></tr>
<tr><td colspan="4">Common Language Runtime</td></tr>
<tr><td>Message Query</td><td>com+</td><td>IIS</td><td>WMI</td></tr>
<tr><td colspan="4">Win 32</td></tr>
</table>

图 1.1 . NET 的层次结构

图 1.1 中最下的两层被.NET 封装,意味着在.NET 中编程时,托管代码中不能调用 Win32 API 函数,也不能使用指针。

图 1.1 中的第 3 层和第 4 层称为公共语言运行库和.NET 框架类库,它们是.NET 的核心基础。

在公共语言运行库和.NET 框架类库之上有 ADO.NET、ASP.NET 和 Web Services 等基类库,这个庞大的基类库为编程任务提供了对象模型和服务..NET 基类库提供的大多数类型是安全和可扩展的,也可以继承这些基类派生出自己的类。

顶层是 Visual Studio .NET 的应用程序开发工具,支持像 C#、VB.NET 和 Visual C++.NET 等 Microsoft 语言编程。此外,也有其他公司提供用于 Visual Studio .NET 的编译器。

1.1.2 公共语言运行库

CLR(Common Language Runtime)称公共语言运行库,它负责管理和执行.NET 框架代码,符合 Visual Studio .NET 编译器规则的代码在执行时需要 CLR 的支持,这些代码称为托管代码(Managed Code)。CLR 的管理工作包括以下几项:

确定加载代码的方式和时机,并且管理对象在内存中的分配。代码执行时,CLR 将决定代码所需要的类和方法,并且按照需要将原先编译过的中间语言(IL)进行即时编译(JIT),形成可执行代码。

对托管所占用的内存进行管理,通过无用内存单元收集器(GC)实现无用内存的回收。

- 确保代码(由 CLR 执行的代码)的类型安全。CLR 负责确保以安全的方式使用类和其他类型。

通过使用基于异常处理机制的通用错误处理方式,可以对托管代码中的错误进行处理和传递。

维护 CLR 和应用程序的安全性。在.NET 框架中,安全性有两种形式:代码访问的安全性和角色的安全性,前者用于确保在安全的上下文中执行代码,后者控制对系统资源的访问。

1.1.3 中间语言(MSIL)和即时编译(JIT)

当编译.NET 程序时,应用程序不是被直接编译成二进制代码,而是被编译成 MSIL。MSIL(Microsoft Intermediate Language)称为中间语言,它是一种能被 CLR 理解的相当低级的指令集。这是部署应用程序所采用的形式,即一个或多个 MSIL 形式的可执行文件和 DLL 文件组成的程序集。这个程序集中,至少有一个包含设计为应用程序的入口点的可执行文件。

在 Visual Studio .NET 下,各种语言(例如 C#和 VB.NET)开发的应用程序,都会被编译成运行时环境的通用语言一中间语言,当应用程序加载后,CLR 根据需要将其再次编译成可执行代码。这种编译称为 JIT(Just-in-Time)即时编译。

此为试读,需要完整PDF请访问: www.ertongbook.com

MSIL有点像Java的字节代码，不同之处在于，Java的字节代码在执行时逐字节解释，而.NET的中间语言会被以模块为单位编译为本地二进制代码。代码一旦被编译，就开始执行并且作为本地代码被保存于内存区。因此，代码的每一部分仅被编译一次，程序执行无论何时扩展到被执行的代码，JIT编译器将在执行前编译它并且将它作为本地二进制代码存储在内存中。就是说，不需要运行的代码不会被编译，执行过的代码不需要再次编译，应用程序的性能得到了最大的优化。

由于中间语言在.NET下的通用性，就使得在Visual Studio .NET下开发程序时，在一种语言中可以使用另一种语言开发的类。例如，C#可以引用VB.NET的类，原因是VB.NET的类被编译成MSIL，C#程序也被编译成MSIL，它们在CLR中当然是通融的。

1.1.4 无用内存单元收集器(GC)

GC(Garbage Collection)称为无用内存单元收集器。有了GC，意味着当一个对象不再被使用时，.NET框架能够自动地回收那个对象使用的内存，这个过程就称为垃圾回收。

垃圾回收的原理来自于.NET的数据类型与内存分配机制。.NET将数据类型分为值类型和引用类型两大类：值类型的例子包括基元类型，例如整型(int)、布尔型(bool)、字符型(char)等，还有一些用户定义的结构型(struct)和枚举型(enum)。引用类型包括类、接口、委托和数组。值类型与引用类型的主要区别在于变量数据的访问方式。若要理解这个区别，需要一些内存分配的背景知识。

应用程序内存数据主要存放在两个区域：栈(Stack)和堆(Heap)。堆栈其实是一块内存，不过一端为堆，另一端为栈，是应用程序为保存程序数据而保留的。

栈供函数调用使用。当某函数fn1被调用时，其所有的非静态变量分配在栈中，称为压栈。当又有函数fn2被调用时，fn2也被压栈，这时fn1被压在fn2的底下。当对fn2的调用结束时，fn2的所有变量被从栈中清除，称为弹出。等到fn1调用结束，fn1也被弹出。所以栈是“先进后出”机制。

而堆为创建可重用的对象而保留。

所有的值类型变量在栈上分配内存，当值类型的变量脱离了作用域时，它就会被销毁，收回其内存。创建引用类型对象时，对象的数据保存在堆中，但在栈中有一个指向那个对象的引用。由于对象是可重用的，一个对象也可能不止1个引用存在。当一个引用的生命期结束时，这个引用被从栈中弹出，这时并不能去销毁堆中与之联系的那个对象，因为可能还有别的引用与之联系着。GC不时来搜索堆中对象的引用，当发现某个对象无任何引用与之联系时，断定这个对象是不必要的，就可将其从堆中销毁，收回其堆内存。

在正常情况下，垃圾回收是一个低优先级的线程。当处理器时间没有被更重要的任务占用时，它才会运行。但是当内存不够时，垃圾回收线程的优先级被提升，使无用内存被迅速回收，以缓解内存的紧张。此后，垃圾回收器线程的优先级又被降低了。

得益于垃圾回收机制的应用，程序员不必像原先那样使用new操作后时时要记得使

用 delete 操作,这样也使类的析构函数的任务大大减轻,内存泄露的可能性几乎不存在了。

采用这种非确定性的内存回收方法,是为了尽量提高应用程序的性能,并为应用程序提供出错较少的运行环境。但这样做也是要付出代价的。鉴于垃圾回收机制,所以不能确定对象何时被回收,所以无法控制何时执行一个类的析构函数,如此析构函数也就不能包括赖以在指定时间上运行的代码。实际上,需要较高资源开销的类通常要实现 Dispose()方法,在该类不再被使用时,显式地释放资源。

1.2 .NET 基类库

.NET 基类库是一个面向对象的类型和接口的集合,它为很多复杂的编程任务提供了对象模型和服务。.NET 基类库提供的大多数类型是安全可扩展的,使程序员可以生成新的类型,加入自己的功能合并到托管代码之中。

1.2.1 .NET 的命名空间

命名空间又称名空间或名字空间,它是组织应用程序的一种结构。使用命名空间是为了避免程序中类名的冲突。例如,某个程序员定义一个类来表示一个顾客,称这个类为 Customer。又有另一个程序员也定义了一个 Customer 类。编译器如何来区分是哪一个 Customer 呢,只要将它们放在两个不同的名空间就可识别两个不同的 Customer。

1. namespace 关键字

命名空间用关键字 namespace 来声明一个范围,可以在命名空间的范围内来组织程序代码。任何一段代码必须放在某一个命名空间内,以防止类名的冲突。使用命名空间的语法为:

```
namespace name[.name1…]
{
    type declarations          //类型声明
}
```

其中,name,name1 是命名空间的名字,它可以是任何合法的标识符,命名空间的名字之间可以包含圆点,圆点确定名空间的层次。

命名空间在缺省情况下会自动创建,它的名字就是项目名,此时的命名空间称为全局命名空间。每个文件都有这样一个命名空间。命名空间可以被公开访问,但不能被修改。

在命名空间中,可以声明一个或多个以下的类型:

- 另一个命名空间
- class
- interface
- struct
- enum

- delegate

既然可以在一个命名空间中创建另一个命名空间，表明命名空间是可以嵌套的。嵌套是为了理清名空间的层次，嵌套也可以用圆点表示。例如命名空间

```
namespace name
{
    namespace name1
    {
        class Customer
        {
            ...
        }
    }
}
```

与以下命名空间等价。

```
namespace name.name1
{
    class Customer
    {
        ...
    }
}
```

有了命名空间，类名可以用命名空间引导。例如 name.name1.Customer 出现在程序的任何地方都不会混淆它的出处。

2. using 关键字

using 用于物理打包和部署程序。using 具有两个用途，一是为一个命名空间创建一个别名，一是允许在一个命名空间中使用另一个命名空间的类，这样不会受到在命名空间中使用类的限制。

为一个命名空间创建一个别名的作用是简化命名空间。如果觉得 System.Windows.Forms 这个命名空间太长不容易写，可以使用 using 关键字来简化它：

```
using myspace=System.Windows.Forms;
```

此后，程序的任何地方见到 myspace 就是指 System.Windows.Forms。

using 的另一个作用是告诉编译系统，它下面使用的类名出自哪些命名空间。例如 C# 的 Windows 应用程序的头部，总会见到以下几行：

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

表明此后的程序中使用的类名来自以上的命名空间。例如有程序行：

```
private System.Windows.Forms.Button button1;
```

可以简化成：

```
private Button button1;
```

因为命名空间 System. Windows. Forms 在程序头部已经声明过,编译系统知道 Button 类的出处。若在程序中使用. NET 的基类名而编译时报错,请检查该类名的拼写是否有错,再检查该类所在的命名空间是否有声明。若是缺少命名空间的声明,请补加 using 声明,或者在类名前一并写上它的命名空间。

using 语句是惟一允许写在任何花括号{}之外的语句,并且其后必须带有分号";"。

1.2.2 .NET 基类库

.NET 基类库是 Microsoft 已经编写好的一个内容丰富的受管制的类代码集合,它可以完成以前要通过 Windows API 来完成的绝大多数任务,使程序员从烦琐的 API 函数调用中解脱出来,专心于应用程序事务的处理。这些类派生于与中间语言相同的对象模型,也基于单一继承性,每个类都可以实例化对象,也可以从中派生自己的类。在较简单的 Windows 编程中,程序员很少从头设计自己的新类。要么继承. NET 基类派生自己的类,设计窗体类就是最好的例子。要么由. NET 基类直接创建对象,在窗体中使用控件正是如此。

.NET 基类的另一个优点就是它的易用性,且都是自描述性的。例如求一个数的平方根,可以调用 Math 类的静态方法 Sqrt()方法(Math. Sqrt()),要取得当前的日期和时间,可以使用 DateTime 类的静态属性 Now(DateTime. Now),要使用一个按钮,只需创建一个 Button 类的对象等等。

Microsoft 尽量代替从事应用系统开发的程序员去做那些重复而繁杂的、有时又是难度很大的工作,其结果是将成果以. NET 基类的方式公开出来供程序员共享。. NET 基类库是一个庞大的类集,基本涵盖了编程的各个方面:

IL 提供的核心功能,例如公共类型系统中的基本数据类型

- Windows GUI 支持、控件等
- Web 窗体及控件(ASP. NET)
- 数据访问(ADO. NET)
- 目录访问
- 文件系统和注册表访问
- 联网和 Web 浏览
- .NET 属性和反射
- 访问 Windows 操作系统和环境变量
- 访问不同语言的源代码和编译器
- COM 互操作性
- 图形技术(GDI+)

这些基类被组织在一系列的命名空间中,这些命名空间是按照类的功能层次划分的。. NET 框架的根(Root)是 System,其他的名空间都在其下若干层次,用"."运算进行层次访问。典型的名空间结构如下:

System

System. Data

System. Data. SQLClient

表 1.1 介绍一些通用的基类命名空间。

表 1.1　　一些有代表性的. NET 命名空间

命名空间	说明
System	它是. NET 框架需要的很多低级别类型的根，也是基元数据类型的根，它还是. NET 基类库中所有其他命名空间的根。
System. Collections	它包含表示不同容器和集合类型的类，例如 ArrayList 等，也可以找到对实现自己的集合功能非常有用的抽象类，例如 CollectionBase。
System. ComponentModel	它包含涉及组件创建和包容的类，例如特征、类型转换器和许可提供程序等
System. Data	它包含数据库访问和操作的类，其下还有附加的命名空间
System. Data. Common	它包含一组由. NET 托管数据提供程序共享的类
System. Data. OleDb	它包含用于为 OLE DB 数据访问的类
System. Data. SqlClient	它包含用于为 SQL Server 数据访问的类
System. Drawing	它公开了 GDI+功能并提供用于绘图的类
System. IO	它提供用于文件系统 I/O 的类
System. Math	它包含常用的数学函数
System. Reflection	它支持在运行时获取信息和动态地创建类
System. Security	它包含用于处理权限、密码系统以及代码访问安全的类型
System. Threading	它包含用于实现多线程应用程序的类
System. Windows. Forms	它包含与创建标准 Windows 应用程序有关的类型。表示窗体和控件的类也在此命名空间

1.3　C# 控制台程序

在 VS. NET 集成开发环境中，可以编写控制台应用程序、Windows 应用程序、和 ASP. NET Web 应用程序等多种形式的应用程序。本书主要讲述 Windows 应用程序的设计方法，但在讲述 C# 语言的基本语法时，以控制台应用程序的形式来表述比较方便一些，本节先讲述控制台应用程序。

设计控制台应用程序，需要创建一个项目。在 VS. NET 开发环境中单击菜单【文件】→【新建】→【项目】，打开“新建项目”对话框，在“项目类型”选择框中选定“Visual C# 项目”，在“模板”选择框中选择“控制台应用程序”，在“名称”框中为项目命名，在“位置”框中指定项目的保存位置，单击【确定】按钮，就可以创建一个控制台应用程序项目。

1.3.1　控制台程序格式

创建一个控制台应用程序项目，项目取名为“hello”。该程序代码如下：

```
using System;
namespace hello
{
    class Class1
    {
        static void Main(string[] args)
```

```
        {
            Console.WriteLine("Hello world");
        }
    }
}
```

该程序中带阴影的一行代码是编者加入的,其余代码由开发器自动生成。程序的运行结果是在屏幕上输出一行文本"Hello world"。

以下来分析以下这个程序的格式。

1. using System

一般,每个程序的头部都有若干 using…行,它的作用是导入命名空间,必须以分号";"结束。using…行是惟一可以写在类外面的行。

using System;的作用是导入命名空间 Sysetm,表示程序中将要使用命名空间 System 下的命名空间或. NET 基类。本程序需要它的原因是程序中使用到 Console 类,Console 出自命名空间 System。对于这个简单的程序,去掉 using System;也行,只需要将 Console 前直接加上 System 引导,变成 System. Console. WriteLine()编译器也能理解。即使导入了名空间 System,有时 System 后的命名空间或者类名记不全,仍然用 System. 来引出后面的命名空间或者类名。

由此可见,在程序头部使用 using…导入命名空间可以简化语句行的书写。复杂的程序需要导入多个命名空间,一般每行导入 1 个命名空间。

2. namespace hello

namespace hello 指定本项目的命名空间。每一个项目最好指定一个命名空间,这个项目的所有程序文件都在这个命名空间中,便于管理。将来别的项目要使用这个项目的类,那时只要用 using hello 导入这个命名空间即可。

namespace hello 后面必须跟有一对花括号{},将这个命名空间中的所有类都写在这一对花括号{}之中。

建立一个项目之后,C# 会自动为这个项目给出一个命名空间,本例是 hello。你也可以修改这个名字,但要小心与其他文件的协调。本程序的 namespace hello{}是开发器自动创建的,编者没有去修改它。

3. class Class1

class Class1{ }是这个程序的一个类,这个类是由 C# 自动创建的,关键字 class 用来声明一个类,其后的 Class1 这个类的名字。

class Class1 后面必须跟有一对花括号{ },将这个类中的所有代码都写在这一对花括号{ }之中。C# 是完全面向对象的语言,除 using…行之外的任何程序成分都必须写在类中。每个 C# 程序至少需要 1 个类,程序也可以由多个类构成。

4. static void Main(string[] args)

这是类 Class1 的一个静态方法,是本程序的入口点,程序从这里启动。每一个程序都必须且只能有一个入口点,那就是 Main。Main 的第一个字母 M 必须大写,开头的 static 表明它必须是静态的,因为只有静态的方法才能在程序运行前创建。

Main 可以有返回值,也可以没有返回值,此处的 void 表示 Main 无返回值。Main 可

此为试读,需要完整PDF请访问: www.ertongbook.com