编　委　会

FOREWORD 前言

本书是为有志于从事.NET开发的读者编写的一本针有对性的书，旨在为读者点亮学习行程中的导航灯，使读者更加明确努力的方向，在短时间内把握学习的要领，增强.NET程序编写和开发的能力。

作者有幸通过全国计算机技术与软件专业技术职业资格考试中的数据库系统工程师、系统集成项目管理工程师和信息系统项目管理师资格考试，这都源于在信息系统项目建设过程中对该领域的热爱。上述考试通过后可以评聘技术员、助理工程师、工程师和高级工程师。

.NET涉及技术范围广，较有深度，学习.NET讲究的不仅是勤奋和坚持，更要讲究方法。对于个人，甚至整个软件行业，方法都是至关重要的。本书在组织结构和内容编排上，倾注了笔者许多的精力和心血，将个人的思考、心得及体会融入其中，相信能够为读者有效地学习.NET技术和软件开发技术打下良好的基础。

本书在写作风格和组织形式上与其他教材相比有独特的特点。本书内容的编写源于工程实践，基于作者多年的工程和教学经验所创作，书中尽可能覆盖最新且实用的.NET开发技术。在结构上，融入了作者多年的开发体会，把握由浅入深的原则，分步骤地讲解.NET的知识，每一个案例均给出了详细的说明。在内容表现形式上，本书以全新的角度撰写，运用生动的语言、深入浅出的方式讲解难点，尽量在案例分析过程中让读者理解、巩固和深化各个知识点，以帮助读者更好地学习。这些内容在实际培训中已取得了良好的效果。

作为一本教材，作者尽献家珍，精心编著，力求做到既“授之以鱼”，又“授之以渔”。本书编写了多个.NET领域的实践案例，案例中涉及的概念较丰富，阐述的问题较典型，介绍的经验较实用，力求读者可以从中获取实践经验，并使读者的学习思路从庞杂的知识点中得到升华。本书的读者对象需要具有一定的编程基础，并有志于不断提升自我。

读者在第一次阅读此书时，可能会对书中的某些概念和应用不能完全理解，但不必着急，学习需要循序渐进，更需要积累，希望读者能够反复研读此书，以体会编程学习中的奥妙。

本书试图在案例的选取与分析上，尽可能多地涉及.NET框架的内容，由于条件的限制，最终只选取了比较重要的几个部分。刘上朝名师工作室的老师们参与了本书习题勘误的工作。

本书在写作过程中，得到很多同事、学术界和计算机工程界朋友们的鼓励和帮助，在此特别要感谢新华教育集团电脑事业部教学运营中心和新华教育集团各电脑院校的大力支持，他们的帮助开拓了我们的研究思路。感谢众多热心的读者提出的意见和建议，这使本书能更加贴近读者。同时，本书在编写过程中，还参考了同行的一些资料和书籍，在此对相关的作者表示诚挚的感谢。

由于编者水平有限，书中难免有疏漏或不妥之处，恳请广大读者批评指正。

编 者

2020年4月

CONTENTS 目录

第1章 .NET框架知识

第2章 C#语言基础

第3章 面向对象编程

第4章 错误、调试和异常处理

第5章 WinForm组件

第6章 文件IO

第7章 网络编程

第8章 多线程编程

第9章 数据库与SQL

第10章 T-SQL高级编程

第11章 ADO.NET编程

第12章 LINQ编程

第1章

.NET框架知识

本章导读

Microsoft Visual Studio C#是Microsoft公司开发的一种使用简单、功能强大、表达丰富的语言。本章将介绍有关.NET和C#的基础知识，如什么是.NET Framework及其类库、公共语言运行时、程序集、命名空间等，最后简单介绍如何配置C#的开发环境。

学习目标

- 掌握.NET Framework的组成
- 掌握CLR的组成与工作原理
- 掌握命名空间、微软中间语言的原理及应用

技能要点

- CLR的分类及原理
- 命名空间与Visual Studio 2017的集成开发环境
- .NET Framework的组成
- FCL的原理及应用

实训任务

- Visual Studio 2017 IDE 环境的熟悉及应用

Chapter 01

1.1 .NET Framework

1.1.1 .NET Framework基本框架

.NET Framework是用于Windows的托管代码编程模型，用于支持生成和运行下一代应用程序。

.NET Framework具有两个主要组件：公共语言运行时和.NET Framework基础类库。

公共语言运行时是.NET Framework的基础，可以将其看作一个在执行时管理代码的代理，它提供内存管理、线程管理和远程处理等核心服务。它强制实施严格的类型安全并可提高程序的安全性、可靠性和准确性。学过Java的应该知道，从这点上来说，有点类似于Java的虚拟机。

.NET Framework的另一个主要组件是类库，它是一个面向对象的可重用类型集合。我们可以使用它开发多种应用程序，包括传统的命令行，如图形用户界面（GUI）或C#控制台命令的应用程序，还包括基于 ASP.NET 所提供的Web应用程序和传统的WinForm窗口程序。.NET Framework和Visual Studio.NET之间的关系如图1-1所示。

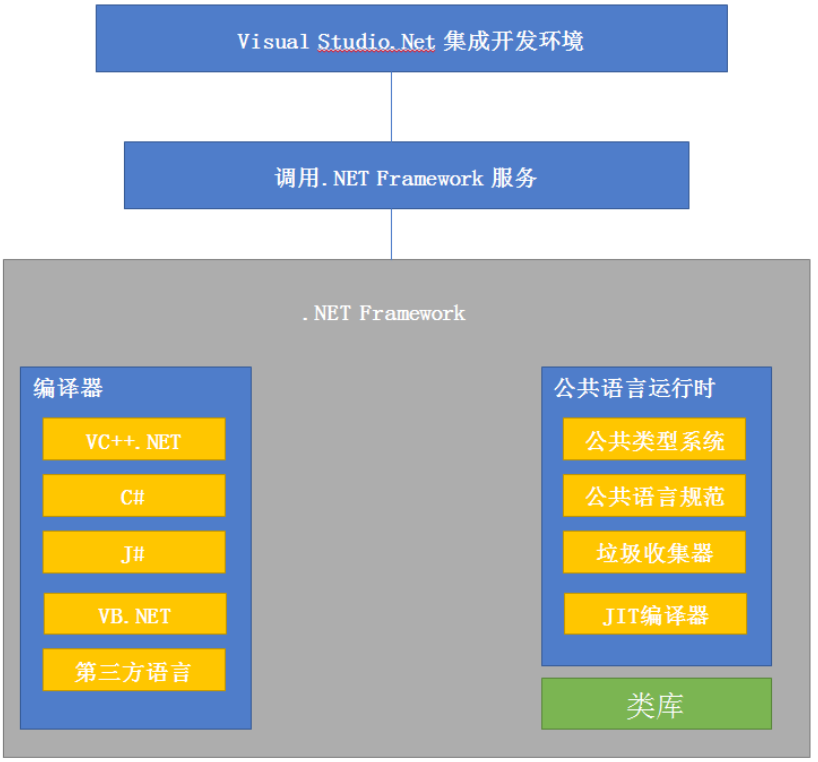

图1-1 .NET Framework和Visual Studio .NET之间的关系

.NET使得软件开发者创建运行在IIS服务上的Web应用程序相比其他编程语言或平台更容易。同时，创建传统的基于客户端和服务器的Windows应用程序和WPF窗体也非常容易。.NET开发平台如图1-2所示。

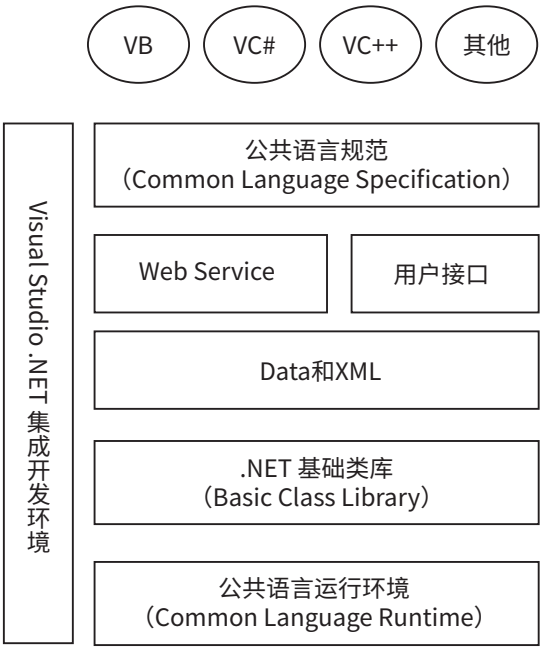

图1-2 .NET开发平台

1.1.2 .NET开发平台客户端应用程序开发

客户端应用程序主要是基于传统的C/S结构开发，软件开发后，需要在用户的客户端上安装应用程序，比如传统的QQ软件、基于企业内部网络使用的应用程序等。客户端应用程序通常使用标签、文本框、列表框、组合框、树形控件、工具栏、菜单控件、按钮和其他GUI元素，它们可能访问本地资源，如文件系统、打印机等。

另一种客户端应用程序是作为网页通过Internet部署的传统ActiveX控件。此应用程序非常类似于其他客户端的应用程序，它在本机执行，可以访问本地资源，并包含图形元素。

过去，开发人员结合使用C或者C++与Microsoft基础类MFC或快速应用程序开发（RAD）环境，如用Delphi或VB来创建此类应用程序。.NET Framework将这些现有产品的特点合并到单个且一致的开发环境中，大大简化了客户端应用程序的开发。

包含在.NET Framework中的Windows窗体类旨在用于 GUI 开发，可以轻松创建适应多变的商业需求所需的WinForm窗口、资源管理器、菜单、工具栏和其他屏幕元素。

1.2 公共语言运行时

CLR（Common Language Runtime）即公共语言运行时，是托管代码执行核心中的引擎。CLR为托管代码提供各种服务，如语言集成、异常处理、版本控制和部署支持、调试和分析服务等。要使CLR能够向托管代码提供服务，语言编译器必须生成一些元数据来描述代码中的类型、成员和引用。

1.2.1 公共语言运行时的特点

基于公共语言运行时的语言编译器开发的代码称为托管代码。托管代码具有许多优点，例如，语言集成、异常处理、增强的安全性、版本控制和部署支持、简化的组件交互模型、调试和分析服务等。

语言编译器和公共语言运行时的功能对于开发人员来说不仅很有用，而且很直观，这意味着公共语言运行时的某些功能可能在一个环境中比在另一个环境中更突出。用户对公共语言运行时的体验取决于所使用的语言编译器或工具。例如，如果读者是一位Visual Basic开发人员可能会注意到，有了公共语言运行时，Visual Basic语言的面向对象的功能比以前强了很多。

1.2.2 公共语言运行时的优点和功能

1. 优点

- 性能得到了提升和改进。
- 可以使用其他语言开发的控件。

- 增加语言功能，如OOP开发中的多态、重载和接口。
- 可以结构化异常处理和自定义属性支持。

2. 功能

- 自我描述的对象。
- 面向对象的设计。
- 非常强的类型安全。
- 集中了Visual Basic的简明性和C++的功能。
- 类似于Java和C++的语法和关键字。
- 使用委托取代函数指针，从而增强了类型安全和安全性。
- 生产本地代码。

1.2.3 公共类型系统

公共类型系统（CTS）在.NET框架内提供一组标准的数据类型和准则集，使得CLR可以在不同语言开发的应用程序之间管理这些标准化的类型，并且在不同计算机之间以标准化的格式进行数据通信。公共类型系统具有以下功能。

- CTS定义了所有应用程序使用的主要.NET数据类型，以及这些类型的内部格式。
- CTS允许不同语言开发的组件可以相互操作。

公共类型系统不仅定义了所有的数据类型，而且提供了面向对象的模型以及各种语言需要遵守的标准。CTS可以分为两大类，值类型和引用类型，同时这两种类型之间还可以进行强制转换，这种转换被称为装箱（Boxing）和拆箱（UnBoxing）。CTS的每一种类型都是对象，并继承自一个基类System.Object。

把值类型转换为引用类型称之为装箱。

把引用类型转换为值类型称之为拆箱。

1.2.4 公共语言规范

要和其他对象安全交互，不用管这些对象是以何种语言实现的，但对象必须只向调用方公开它们必须与之互用的所有语言的通用功能，为此定义了公共语言规范（CLS）。CLS是许多应用程序所需要的一套基本语言功能。

大多数由.NET Framework类库中的类型定义的成员都符合CLS。

CLS定义了所有基于.NET Framework的语言都必须支持的最小功能。CLS定义的规则可以概括为如下4点。

- CLS定义了原语数据类型，如Int32、Int64、Single、Double和Boolean。
- CLS定义了基于0的数组的支持。
- CLS定义了事件名和参数传递给事件的规则。
- CLS定义了命名变量的标准规则。例如，与CLS兼容的变量名必须以字母开头且不能包含空格。

除了上述标准外，CLS还定义了其他标准。任何语言都可以扩展基本的CLS需求，不鼓励使用非标准的功能，因为这样做妨碍了语言之间的互相操作性。完全CLS的语言称为兼容的CLS语言。

1.2.5 中间语言

在.NET框架中，公共语言基础结构使用公共语言规范来绑定不同的语言。通过要求不同的语言，至少要实现公共类型系统（CTS）包含在公共语言规范中的部分，公共语言基础结构允许不同的语言使用.NET框架。因此在.NET框架中，所有语言（C#、VB.NET等）最后都被转换为一种通用语言，即微软中间语言（MSIL）。

MSIL是将.NET代码转化为机器语言的一个中间过程，是一种介于高级语言和Intel汇编语言之间的伪汇编语言。当用户编译一个.NET程序时，编译器将源代码翻译成一组可以有效转换为本机代码且独立于CPU的指令。中间语言的主要特征如下所述。

- 具有使用特性。
- 属于强数据类型。
- 可以使用异常来处理错误。
- 面向对象和使用接口。
- 值类型和引用类型之间存在巨大反差。

中间语言的格式类似于程序集语言。程序集语言的语句直接与内部CPU体系结构支持的指令相关，但中间语言的格式通常不依赖于特定CPU的体系结构。也就是说，中间语言不直接引用CPU寄存器或者执行CPU指令。当用户执行中间语句格式的应用程序时，另一个JIT编译器的实用程序会进一步把中间语言转换为目标CPU可以执行的本机可执行文件。

.NET开发平台包括.NET Framework和.NET开发者使用的工具。.NET Framework是整个开发平台的基础，包括CLR和FCL，.NET开发者使用的工具包括Visual Studio.NET集成开发环境和.NET编程语言。希望读者能认真体会，并坚持学习。

1.3 .NET Framework类库

.NET Framework类库是一个与公共语言运行时紧密集成的可重用类型集合，它是一个由.NET中包含的类、接口和值类型组成的库。.NET Framework层次结构的基本类型为System.Object，也就是说System.Object类位于层次结构的最顶端，是.NET开发平台中的根，它提供了.NET Framework中所有类型的基本功能。表1-1列举了System.Object类中的一些基本服务。

表1-1 System.Object提供的服务

服务	说明
System.Object	提供了构造函数，而构造函数提供了从底层类型创建对象的机制
Equals	用于测试两个对象是否包含相同的数据，测试数据是否相同，而不是引用是否相同
GetHashCode	用于定义类型的哈希函数
GetType	用于返回对象的数据类型
ToString	用于把对象的值转换为字符串，大多数类中会重写该方法
ReferenceEquals	用于测试引用是否相等，也就是说测试两个对象变量是否引用了相同的类实例

1.4 命名空间

.NET Framework类库包含了大量的类，大约有3500个类。为了让程序设计人员快速找到所需要的类，.NET Framework类库被划分为许多命名空间，而每一个命名空间包含了功能相似的类。

1.4.1 命名空间的组织方式

将具有相似功能的相似类在逻辑上进行分组，称为命名空间。C#中的命名空间与Java语言中的包具有相似的功能。命名空间是一种逻辑组合，而不是物理组合。不在同一个文件中的多个类可以共同包含在一个命名空间中，这样就创建了一个逻辑结构。

一个程序集可以包含一个或多个命名空间。例如System和System.IO命名空间都保存在System.dll程序集中。前一个命名空间也可能保存在两个程序集中。表1-2列出了所有.NET Windows窗体应用程序都会使用的命名空间。

表1-2 窗体应用程序使用的命名空间

命名空间	说明
System	该命名空间定义了数据类型、事件和事件处理程序的基本类
System.Data	该命名空间包含提供数据访问功能的命名空间和类，这些命名空间构成了ADO.NET
System.Drawing	该命名空间包含了提供Windows图形设备接口的类，这些类定义了各种绘图的类型，如圆、长方形等
System.IO	该命名空间包含了数据的读取和写入的所有操作
System.Windows.Forms	该命名空间包含工具箱中的控件及窗体自身的类

续表

命名空间	说明
System.Net	该命名空间包含了用于网络通信的类或命名空间
System.Xml	该命名空间包含了用于处理XML数据的类

在.NET Framework中，Integer或者String被认为是类型，例如System命名空间在C#中是所有系统命名空间的根。

命名空间包含类、委托、结构、枚举和接口，它们都是类型。这些类型既有值类型又有引用类型，.NET开发者创建的每一个类型，以及用户在.NET应用程序中创建的类型都属于上述类型之一。其中，System命名空间包含了结构-值类型和类-引用类型；System.IO命名空间包含了类-引用类型和枚举类型。.NET Framework的所有组件以及开发者创建的所有组件都组织到包含类的命名空间中。如前所述，System命名空间可以包含其他命名空间或者类型，类可以包含成员，这些成员可以包含属性、方法，例如WriteLine方法与Console类用于控制台窗口输出字符串，ReadLine方法用于从键盘上接收字符串。

1.4.2 定义命名空间

前面已经对命名空间进行了概述，命名空间能为各种标识符创建一个已经命名的容器。这样，同名的两个类如果不在同一个命名空间中，相互是不会混淆的。要定义命名空间，就需要使用namespace关键字。例如，定义一个名字为ynxh的命名空间，该命名空间包含了两个类，具体代码如下所示。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ynxh
{
class persion
{
    private string name;
    private int age;

   public void GetInfo()
   {
     Console.WriteLine("请输入姓名和年龄: ");
     name=Console.ReadLine();
```

```
        age=Int.Parse(Console.ReadLine());
      }
       public void DispInfo()
       {
         Console.WriteLine("{0}的年龄为:{1}",name,age);
       }
    }
   Class program
   {
     Public  static void Main()
     {
       Persion per=new persion();
       Per.GetInfo();
     }
   }
}
```

上述定义的命名空间包含了两个类，一个类名为persion，用于处理人的相关信息；一个类名为program，用于对persion类进行实例化。把一个类放在命名空间中，可以有效地避免出现重名现象。该类的全名可用命名空间名称加句点和类的名称表示。在上述案例中，persion的全名为ynxh.persion，这样即使类名称相同，只要所属的命名空间不同，也可以在同一个程序中使用。当然C#中的命名空间还可以嵌套，比如下面的案例在C#语言中是支持的。

```
    Namespace namespace1
    {
        Namespace namespace2
        {
            Namespace namespace3
            {
                Class mycalss
                {
                }
            }
        }
    }
```

每一个命名空间的名称中应该包含它所在命名空间的名称，这些名称之间用句点分隔开，首先是最外层的命名空间名称，其次是内层的命名空间名称，最后是自己的命名空间名称。所以namespace2的全名是namespace1.namespace2，而myclass的全名是namespace1.namespace2.namespace3.myclass。所以在这里定义命名空间时也可以直接给出全名。

1.4.3 使用.NET Framework类库

在C#开发项目中.NET Framework类库被广泛用于各个方面，从文件系统访问和字符串操作到Windows窗体、WPF窗体和ASP.NET用户界面控件。该类库被组织到多个命名空间中，每个命名空间包含一组相关的类和结构。例如，System.Drawing命名空间包含多种类型，这些类型表示字体、画笔、线条、形状、颜色等。

要使用.NET Framework类库必须使用using指令引用相应的命名空间，才能在C#程序中使用该命名空间中的类。在某些情况下，还必须添加包含该命名空间的DLL的引用，在C#程序中系统才会自动添加对最常用类库DLL的引用。

下面就使用.NET Framework类库的相关内容来创建一个实例。该实例中，当用户在“登录系统”窗口中输入账号信息，然后单击“取消”按钮，将弹出相应的信息。实例代码如下所述。

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace xx
{
    public partial class ceshi : Form
    {
        public ceshi()
        {
            InitializeComponent();
        }
        private void button2_Click(object sender, EventArgs e)
        {
         if (MessageBox.Show("是否要关闭？ ", "询问信息",
MessageBoxButtons.OKCancel, MessageBoxIcon.Warning) ==
```

```
DialogResult.OK)
                    this.Close();
            }
        }
    }
```

演示效果如图1-3所示。

图 1-3　代码演示效果

上述代码只是使用.NET Framework类库的简单访问。在代码中，首先通过using语句引用相应的命名空间，从而可以调用该命名空间内的类、方法等，然后获取用户输入的账户和密码，如果单击“取消”按钮，则会弹出对话框，提示用户是否关闭该窗口。

1.5 配置C#环境

要开发C#应用程序，就必须先配置C#的开发环境。由于C#是Microsoft.NET平台中的一种语言，需要在.NET IDE环境中开发，因此，在C#开发前，安装.NET Framework是非常必要的。

Visual Studio.NET 2017是一套完整的开发工具，用于生成ASP.NET应用程序、MVC、Windows桌面应用程序和移动应用程序等。在Visual Studio 2017开发环境中支持Visual Basic.NET、Visual C++.NET、Visual C#.NET等开发语言。该环境允许它们共享并创建混合语言解决方案，这些语言都运用.NET Framework的功能，它提供了对简化ASP.NET Web应用程序和XML Web Services开发关键技术的访问。图1-4为Visual Studio 2017的集成开发环境。

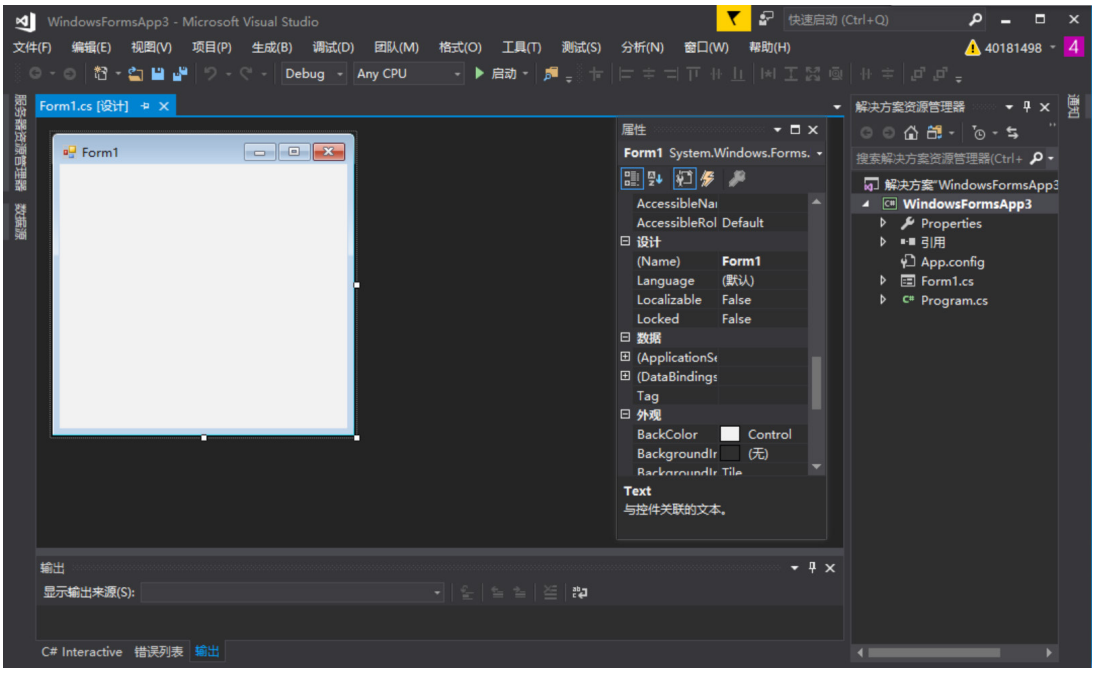

图1-4　Visual Studio 2017的集成开发环境

在图1-5项目的解决方案资源管理器中，可以组织和管理目前正在编辑的项目，可以重命名和删除文件，也可以创建新的文件和文件夹。在该窗口中，针对不同的项单击鼠标右键，在弹出的菜单中会根据不同的项显示不同的菜单项。在解决方案资源管理器的最上面，有一个工具栏，包括“属性”“刷新”“视图设计器”等工具，通过这些工具，可以方便、快捷地执行常用操作功能。

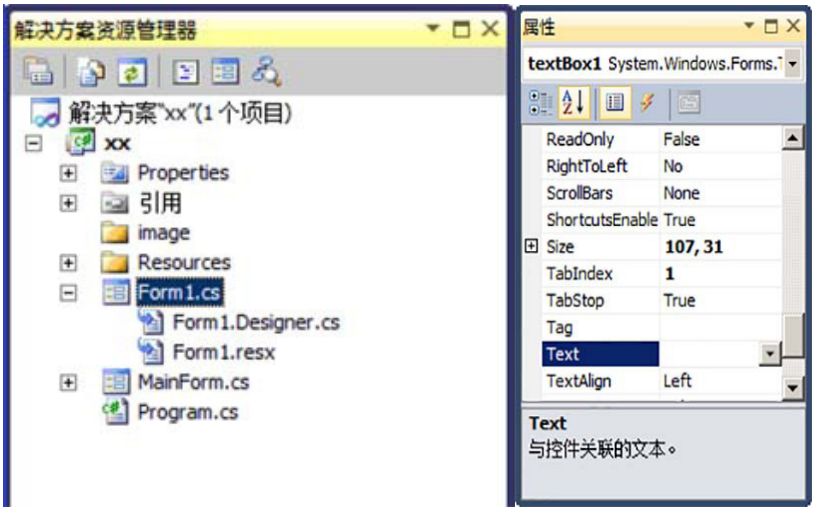

图1-5　解决方案资源管理器和属性窗口

本章总结

本章首先讲述了.NET Framework架构的知识，包括CLR；其次讲解了C#中命名空间的结构、如何引入命名空间、系统提供的常用命名空间有哪些；最后讲解了Visual Studio 2017集成开发环境的知识、如何创建项目、如何在集成开发环境中进行编译等。

练习与实践

【问答题】

1．.NET Framework组件包含哪几个部分，分别是什么，有何特点？
2．什么是命名空间，在C#中最原始的是哪个？
3．Visual Studio 2017的IDE环境相比Visual Studio 2015有哪些改进？
4．什么是装箱，什么是拆箱，分别有何特点？
5．微软中间语言有何特点，功能是什么？

【实训任务】

创建一个学生管理系统的解决方案	
项目背景介绍	Visual Studio 2017 IDE环境的熟悉与应用。
设计任务概述	在Visual Studio 2017中创建一个学生管理系统的解决方案，并保存在“d:\练习”文件夹中。所选择的开发语言为C#，并通过控制台输出一行文字，“这是我的第一个C#程序。”
实训记录	
教师考评	评语： 辅导教师签字：

第2章

C#语言基础

本章导读

编程语言通过语句来处理数据，而程序是由一系列按照某种顺序执行的语句组成，它是编程的基础知识。本章介绍变量的声明、初始化、引用，以及数据类型、运算符与表达式、结构类型、枚举类型、控制语句等方面的内容。

学习目标

- 掌握变量的声明、初始化和引用
- 理解并掌握如何使用数据类型定义变量
- 理解并掌握如何使用控制语句和数组

技能要点

- 数据类型、变量、常量的定义与引用
- 控制语句的使用

实训任务

- 通过控制台环境输出一个菱形，掌握循环嵌套

Chapter 02

2.1 变量和数据类型

变量是在程序执行过程中会发生变化的量。在C#语言中，变量必须事先定义，并且在程序范围内变量名称必须唯一，使用变量的名称来引用它所容纳的值。数据类型是指变量将在分配的存储空间中保存为哪一种类型的信息，C#的数据类型与C语言、Java语言的数据类型大部分相同，如果学过C语言和Java语言的读者再学习C#会有非常类似的感觉。

提示 需要注意的是，C#中的变量是区分大小写的，大写字母和小写字母不是同一变量，因为有些语言可能是不区分大小写的，如VB。

2.1.1 使用变量和数据类型

在使用变量之前必须先命名变量，命名变量需遵循一定的规范。下面是命名变量的一些要求和建议。

（1）不要创建只有大小写区别的标识符。

（2）变量开头的字母不要使用下划线，如不要使用这样的变量：_Name、_Sex等。

（3）由字母、数字、下划线和美元符（$）组成的变量，不能包括空格、标点符号和运算符。

（4）变量名最好见名知意，这样可以更好理解。具体的命名规则，不同的公司有不

同的习惯，在实际工作中还是以公司要求为主。

（5）变量名不能与C#中的保留字相同。

C#提供了大量的内建类型，图2-1列出了C#中的数据类型。

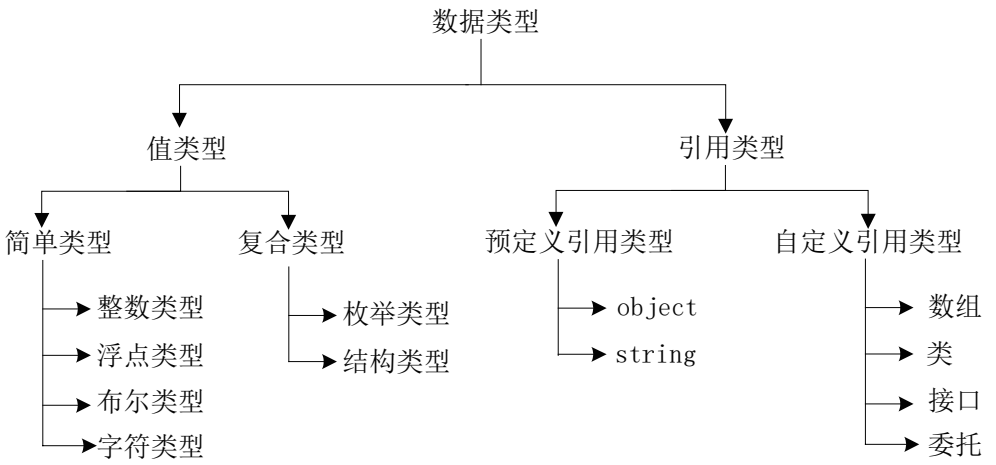

图2-1 C#中的数据类型

1. 值类型

值类型是直接存储值，比如一般使用的整型数据、字符型数据、布尔型数据和浮点型数据都属于值类型，值类型分为简单类型和复合类型，具体如表2-1所示。

表2-1 值类型的分类与说明

类型	说明	范围
Sbyte	8位有符号整数	-128~127
Short	16位有符号整数	-32768~32767
Int	32位有符号整数	-2147483648~2147483647
Long	64位有符号整数	-9223372036854775808~9223372036854775807
Byte	8位无符号整数	0~255
Ushort	16位无符号整数	0~65535
Uint	32位无符号整数	0~4294967295
Ulong	64位无符号整数	0~18446744073709551615
Float	有效数字到6~7位	-3.4*10^38 ~ 3.4*10^38
Double	有效数字到15~16位	-1.7*10^(-308) ~ 1.7*10^(308)
Decimal	有效数字到28位	- 1*10^38-1 ,1*10^38-1

布尔类型主要用来表示true/false值。C#中定义布尔类型时，需要使用bool关键字。例如，下面的代码是定义一个布尔类型的变量。

```
bool x = true;
```

在C#语言中，使用char/Char类定义字符，并且字符只能用单引号括起来。

2. 引用类型

引用类型是存储对值的引用，引用类型的变量又称为对象，C#中的引用类型主要是对象或者说是类和字符串。特别需要注意的是，尽管字符串类型是引用类型，但如果与相等运算符（“==”或“!=”）运算，则表示是比较字符串对象而不是引用的值。.NET中预定义的两种引用类型如表2-2所示。

表2-2 .NET中预定义的引用类型

类型	说明
Object	Object类型在.NET Framework中是Object的别名。在C#的统一类型系统中，所有类型（预定义类型、用户定义类型、引用类型和值类型）都是直接或间接从Object继承的
String	String类型表示零或更多Unicode字符组成的序列

命名一个变量时，它将包含一个随机的值，直到为其指派要给的新值。在C#中不允许编程人员使用未赋值的变量。如果申明了一个变量则必须给它分配一个初始值，一个变量必须赋值后才能使用，否则程序是无法编译的，这就是所谓的明确赋值规则。当把一个值赋值给一个值类型变量时，该值实际上被赋值到变量中去，而把一个值赋值给一个引用类型的时候，仅是引用被复制，实际的值仍然保留在原来的内存位置，但现在有两个对象指向它，也就是说引用了它。

引用变量是通过引用类型声明的变量。引用类型的变量不直接存储所包含的值，而是指向它所要存储的值。C#提供了以下几种引用类型。

1）字符串类型

在C#中有用于操作字符串的基本字符串类型。字符串类型主要用于存储一条语句或多个单词，一般情况下单个字母是字符，而多个字母组成的就称为字符串。

System.String的一个别名，它的使用类似下面的代码。

```
String mystring="这是一个字符串数据";
```

如果用户想要访问某个字符串数据的某个字符，可以使用如下方式。

```
Char ch=mystring[1];
```

字符串的比较可以用“==”比较操作符来实现，代码如下所述。

```
If(mystring==string)
{
//执行的程序
}
```

2）数组

同C、Java、PHP等主流编程语言一样，C#语言中也有数组类型，数组的定义为名称相同但下标不同的一组变量。数组可以存储整数对象、字符串对象或其他任何对象。在C#语言中数组可以分为一维数组和多维数组，实际上我们常用的是一维数组和二维数组，二维以上的数组在一般应用中很少用到。

数组的维数决定了相关数组元素的下标，最常用的数组是一维数组。一个多维数组具有一些特点：维数大于1；每个维的下标始于0；下标最大值是维的长度减1。类似代码如下所述。

```
String [] name={ "张三", "李四", "王五"};//等效于下面这句代码
name[0]= "张三";name[1]= "李四";name[2]= "王五";
```

C#中支持动态数组，也就是其长度不是固定的，代码如下所示。

```
Int a=10;
Int[] x=new int[a];
```

需要注意的是，C#中的数组与Java、PHP这些编程语言一样，数组下标在默认情况下从0开始。

3）类类型

类是面向对象编程的基本单位，它包含成员变量、成员方法、构造函数、析构函数等，具体如下所示。

```
class student
{
    private string name;
    private string sex;
    public student(string sex1)
    {
       this.sex=sex1;
    }
```

```
    Public string Name
    {
        Get
        {
             Return name;
        }
        Set
        {
             name=value;
        }
    }
}
```

上面定义了一个类，该类的名称为student，该类包括的数据成员有姓名、性别。类的函数成员有构造函数student，设置或返回学生姓名的方法是name。

4）接口类型

C#中的接口只有方法名，但这些方法没有执行代码，这就说明不能定义一个接口的变量，只能定义一个派生自该接口的类的对象。与类的类型相比，当定义一个类时，该类可以派生自多个接口，但只能派生自一个类。C#中的类都是单一继承的，如果需要多继承，则必须通过接口实现。定义接口的语法如下所述。

```
    Interface  Ipersion
    {
         Void ShowpInfo();
    }
```

在上面定义的接口只有一个方法，虽然不能从这个定义实例化一个对象，但可以从它派生出一个类。因此，该类必须使用ShowpInfo抽象方法，具体如下所示。

```
    Class classp:Ipersion
    {
         Public ShowpInfo()
         {
    //相关处理
           Console.WriteLine("hello C#!");
         }
    }
```

从上面可以看出接口成员与类成员的区别在于接口成员不允许被实现。

5）委托类型

C#中的委托类型和C语言或C++语言中的函数指针非常相似，使用委托类型可以把类内容方法的引用封装起来，然后通过委托来访问这些方法。C#中的委托类型有个特性就是不需要知道被引用的方法属于哪一个类的对象，只要函数的参数个数和返回值与委托类型对象中的一致就可以了。下面通过一个案例来说明这个问题。

首先，需要定义一个委托类型，该类型名为Ynxh，具体如下所述。

```
Delegate string Ynxh();
```

然后，定义一个类student，该类包含要命名为ShowInfo()的方法。这里要注意的是，该方法应与上面声明的Ynxh委托类型具有相同的参数个数和返回值。具体代码如下所述。

```
public class student
{
  public string ShowInfo()
  {
      Return "中国程序网http：//www.csdn.net";
  }
}
```

接着，定义一个测试类Test，在它的入口函数Main中声明一个student类和一个Ynxh委托类型，并将onestudent.ShowInfo用oneYnxh代替，其运行结果和直接调用onestudent.ShowInfo方法的效果是一样的，但是在调用过程中并不知道调用了哪个类的方法。类似代码如下所述。

```
public class Test
{
  public static void Main(string[] args)
  {
      student onestudent=new student();
      Ynxh oneYnxh=new Ynxh(onestudent.ShowInfo);
          //下面使用oneYnxh代替对象onestudent的方法
          System.Console.WriteLine(oneYnxh);
   }
}
```

2.1.2 声明和初始化变量

C#中变量的声明语法如下所述。

```
定义整型变量 int x=23;
定义字符串变量 string a1,a2,a3;
a1= "计算机及应用专业";
a2= "工程造价专业";
a3= "物联网与信息安全专业";
```

从上述语句可以看出，不但可以在一个语句中声明一个变量，而且也可以在一个语句中声明多个变量。这时，在语句开始部分指定的数据类型适用于所有变量，用逗号将变量名称隔开，并在语句末尾加上分号。

在C#中使用数值变量之前，必须为其指定值。也就是说，变量必须首先出现在等号的左边，然后才能用于等号的右边。在声明变量时，用户可以直接对它进行初始化。

如果引用一个变量而没有为它指定初始值，那么编译时将会生成错误。

2.1.3 数据类型的转换

在实际编程中可能经常需要进行数据类型转换，因为不同的数据类型是不能相互运算的，这点C#语言与C语言有比较明显的区别。比如，等号左边是字符数据，而等号右边是整数，这是非常明显的数据类型不匹配。如果运行系统报错，就需要进行数据类型的转换，让等号左边变量的数据类型和等号右边的数据类型一致，这样才能运算。不同的数据类型可以进行转换，比如整型和浮点型数据、数值型数据和字符型数据、字符型数据和日期时间型数据之间都能转换。

1. 将字符串转换为数值数据类型

在C#中可以使用Parse方法将控件的Text属性转换为数值形式，然后在计算中使用这个值。每一种数据类型都有自己的Parse方法，如int.Parse()、decimal.Parse()和double.Parse()等。用户可以将想要转换的文本串作为Parse方法的一个参数进行传递。

具体使用方式如下所述。

```
//改变数据类型
Decimal dprice=decimal.Parse(TextBox1.Text);
Int x=int.Parse(TextBox2.Text);
Decimal a=dprice*x;
```

这里，用Parse方法检查存储在参数中的值，并尝试在一个名为“解析”的过程中将其转换为一个数字。这意味着需要对字符逐个进行拆分，然后转换为另外一种格式。当Parse方法遇到一个不能转换为数字的值时，会发生错误。

2. 数值数据类型之间的转换

在C#中可以将数据从一种数值数据类型转换成另一种数值数据类型。有些转换可以隐式执行，而另外一些转换则是必须明确执行。一些数据类型如果在转换过程中丢失，则不能转换。

1）隐形转换

如果将一个值从范围较小的数据类型转换为较大的数据类型，这种转换对于该值没有丢失任何精度的风险，那么可以使用隐形转换来执行这种转换。

从decimal数据类型转换为其他数据类型不存在隐形转换，不能通过float隐形转换为decimal。

2）强制转换

如果用户想在不能进行隐形转换的数据类型之间进行转换，就必须使用“强制转换”，也就是指明要转换的类型。这里需要注意的是，如果某一次执行强制转换导致丢失有效数字，这时会生成一个异常。强制转换是在要转换的数据前面的圆括号中指定目标数据的类型。类似代码如下所述。

```
Int x=(int)dbx;//把一个双精度浮点型数据转换成整型
Float y=(float)dbx;//把一个双精度浮点型数据转换成单精度浮点型
```

强制转换用在知道要转换的值合适（不会丢失有效数字）时，才能执行从一个较大的数据类型到较小的数据类型的转换。小数值将会舍入以适应整型数据类型，转换为decimal的float或double值将会舍入，以便适应28位。此外，System命名空间的Convert类提供了强制类型转换的方法，具体可以参照微软官方帮助。

2.2 运算符与表达式

运算符指明了进行运算的类型，例如，加号“+”用于加法，减号“－”用于减法，星号“*”用于乘法，正斜杠“/”用于除法等。将运算符、常量、变量、函数连接起来便构成了表达式。C#语言中的运算符和表达式与Java语言中的运算符和表达式基本相同，学过Java的读者会发现两者非常相似，所以开发人员能很快地从Java语言的开发转移到C#语言的开发中来。

2.2.1 运算符

C#提供了算术运算符、逻辑运算符、递增运算符以及其他一些运算符。开发人员可以指定运算符在特定数据类型上的行为，如表2-3所示。

表2-3 C#中的运算符

含义	运算符	数目	结合性
单目	++、--、！	单目	←
算术	+、-、*、/、%	双目	→
移位	<<、>>	双目	→
关系	>、>=、<、<=、==、!=	双目	→
逻辑	&&、‖、！	双目	→
条件	?:	三目	←
赋值	=、+=、-=、*=、/=、%=	双目	←

2.2.2 表达式

表达式是语法正确的编程语句。表达式涉及的内容通常包括赋值计算以及真假判断等。通常情况下，一个表达式包含多个运算符时，就必须给出表达式中运算符的运算顺序。例如求表达式a+b*c的值，应该是先算b*c，然后再进行相加运算。

当在一个表达式中出现两个具有相同优先级的运算符时，运算符应按照出现表达式中的顺序由左至右执行。除了赋值运算符和条件运算符外，其他所有的二元运算符都是左结合的，也就是说在表达式中按照从左向右的顺序执行，例如，a＋b－c按(a＋b)－c进行求值。赋值运算符合条件运算符(?:)，则按照右结合的原则，即在表达式中按照从右到左的顺序执行，如a＝b＝c按照a＝(b＝c)进行求值。

2.3 控制语句

C#中常用的语句包括：分支语句（if…else 和switch语句）、循环语句（for、foreach、while和do…while语句）和跳转语句（goto、continue和break语句）。一个C#程序通过这些语句的组合实现特定的功能，本节将对这些内容进行介绍。

2.3.1 分支语句

当程序需要进行两个或两个以上的选择时，可以根据条件来判断，选择将要执行的

一组语句。在现实生活中需要进行判断和选择的情况很多。比如：从昆明出发上高速公路，到了一岔路口，有两个出口，一个是去杭州方向，一个是去瑞丽方向，驾车者到此处必须进行判断，根据自己的目的地，从二者中选择一条路径。在日常生活或工作中，类似这样需要判断的情况是司空见惯。

如果是星期六、星期日，则休息，否则需要上班（需要判断是不是周末）。

如果闰年，则二月份有29天，否则只有28天（需要判断是否是闰年）。

如果你年龄在18岁以上则有选举权和被选举权（需要判断是否满18岁）。

60岁以上老年人，入公园免费（需要判断是否满60岁）。

C#提供的选择语句有两种：if语句和switch语句。

1. if语句

if语句是最常用的分支语句，它根据布尔表达式的值判断是否执行后面的内嵌语句。默认情况下，if语句控制着下方紧随的一条语句的执行，通过语句块，if语句可以控制多个语句，if语句格式如下所述。

```
if(逻辑条件==true)
{
      语句块1;
}
else
{
      语句块2;
}
```

在上述格式中，当条件为真时，程序执行语句块1，如果条件为假则执行语句块2。

如果程序的逻辑判断关系比较复杂，通常会用条件判断嵌套语句，if语句可以嵌套使用，即在判断之中再判断，具体格式如下所述。

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

```
else
{
    if(布尔表达式2)
    {
    语句3;
    }
}
```

案例1

输入任意三个整数，计算并输出最大值。

分析：比较三个数中的最大值，可以先比较两个数的最大值，然后用两个数的最大值与第三个数比，如果比第三个数小，则最大值就是第三个数，否则就是它本身，具体代码如下所述。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
      static void Main(string[] args)
      {  int a,b,c,max;
           Console.WriteLine("请输入三个整数:");
           a= int.Parse(Console.ReadLine());
           b= int.Parse(Console.ReadLine());
           c= int.Parse(Console.ReadLine());
           if(a>=b)
              max=a;
           else
               max=b;
           if(max<=c)
               max=c;
           Console.WriteLine("最大值是:{0}",max);
       }
     }
  }
```

运行结果如图2-2所示。

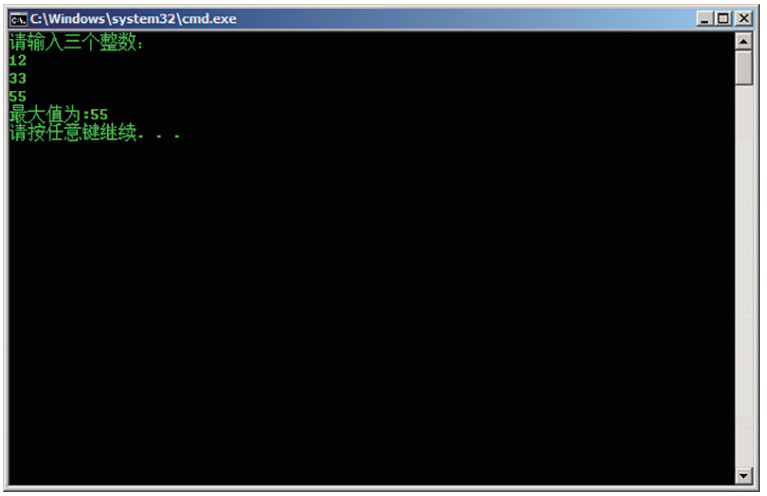

图2-2　比较数值

案例2

从键盘上输入一个不大于5位的正整数，计算其是几位数。

分析：题目要求不大于5位，所以最大值是99999，从这里可以知道，在10000~99999之间是5位数，1000~9999之间是4位数，100~999之间是三位数，10～99之间是两位数，0~9之间是一位数，具体代码如下所述。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b;
             Console.WriteLine("请输入不大于五位的正整数: ");
             a=int.Parse(Console.ReadLine());
             if (a>9999)
               b=5;
             else if (a>999)
               b=4;
             else if (a>99)
```

```
                b=3;
            else if(a>9)
                b=2;
             else
                b=1;
            Console.WriteLine("计算结果为:{0}", b);
        }
    }
}
```

运行结果如图2-3所示。

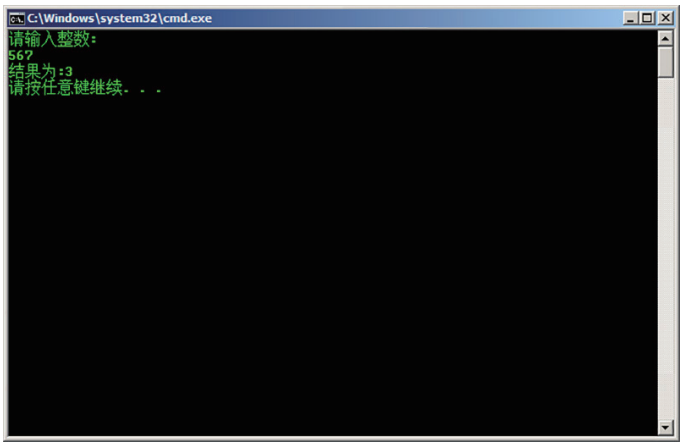

图 2-3　不大于5位正整数是几位数的运行结果

2. switch语句

1）switch语句的一般形式

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

2）执行过程

（1）当switch后面的“表达式”的值与某个case后面的“常量表达式”的值相同时，就执行该case后面的语句；当执行到break语句时，则当前语句执行结束。

（2）如果case后面的任何一个“常量表达式”的值与“表达式”的值都不匹配，则执行default后面的语句n+1，然后执行switch语句的下一条。具体流程如图2-4所示。

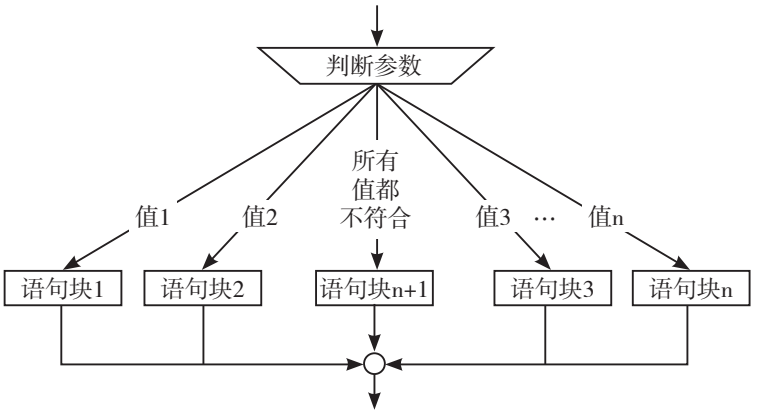

图2-4　switch语句执行流程图

3）说明

（1）switch后面的“表达式”可以是int、char和枚举型中的一种。

（2）每个case后面“常量表达式”的值必须是常量，不能是变量，也不能是表达式。

（3）case后面的常量表达式仅起语句标号的作用，不进行条件判断。系统一旦找到入口标号，就从此标号开始执行，不再进行标号判断，所以必须加上break语句，以便结束switch语句。

（4）各case及default子句的先后次序，不影响程序执行的结果。

（5）多个case子句，可共用同一语句。

（6）用switch语句实现的多分支结构程序，可以用if语句或if语句的嵌套来实现，具体选择哪种语句，由编程人员的个人习惯来选择。

案例3

已知某公司员工的保底薪水为500，某月所接工程的利润p与利润提成的关系如下所示。（计量单位：元）

p≤1000	没有提成
1000＜p≤2000	提成10%
2000＜p≤5000	提成15%
5000＜p≤10000	提成20%
10000＜p	提成25%

计算出员工的收入。

分析：为使用switch语句，必须将利润p与提成的关系转换成某些整数与提成的关系。分析本题可知，提成的变化点都是1000的整数倍（1000、2000、5000、……），如果将利润p整除1000，则为：

p≤1000	对应0、1
1000＜p≤2000	对应1、2
2000＜p≤5000	对应2、3、4、5

5000＜p≤10000　　对应5、6、7、8、9、10
10000＜p　　对应10、11、12、……

为解决相邻两个区间的重叠问题，最简单的方法就是利润先减1，然后再整除1000即为：

p≤1000　　对应0
1000＜p≤2000　　对应1
2000＜p≤5000　　对应2、3、4
5000＜p≤10000　　对应5、6、7、8、9
10000＜p　　对应10、11、12、……

具体代码如下所述。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int a, b;
            double c;
            Console.WriteLine("输入你的业绩");
            a=int.Parse(Console.ReadLine());
            b=(a-1)/1000;
            switch (b)
            {
                case 0: c = 500 + a * 0;  break;
                case 1: c = 500 + a *0.1;  break;
                case 2:
                case 3:
                case 4:c = 500 + a *0.15;  break;
                case 5:
                case 6:
                case 7:
                case 8:
                case 9:c = 500 + a *0.2;  break;
                default:c = 500 + a*0.25;  break;
```

```
                }
                Console.WriteLine("亲你这个月的工资是:{0}",c) ;
            }
        }
    }
```

运行结果如图2-5所示。

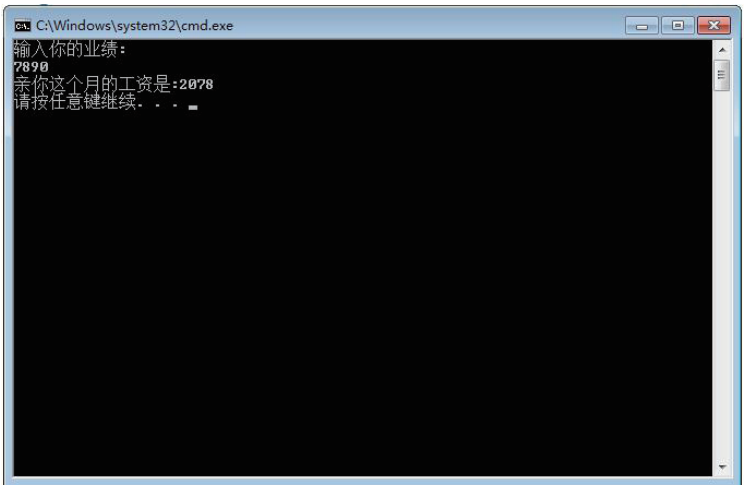

图2-5　计算利润与提成的关系

2.3.2　循环语句

1. 循环语句综述

顾名思义，循环语句就是重复执行，如果没有遇到退出条件就一直执行下去，所以在循环语句中需要有一种跳出循环的机制。比如一个案例，要求计算1+2+3+4+5+…+100的和。这个案例需要重复执行一百次，当循环执行到101次时，需要结束，因为案例只需要计算到100，当循环变量变为101的时候，循环条件不满足了，循环就必须结束。

C#语言提供了3种循环语句。为简化和规范循环结构程序设计，goto语句在实际程序开发过程中使用得非常少，因为goto语句的跳转，对结构化程序设计有很大的影响。

在C#语言中，可用以下语句实现循环。

（1）for语句。

（2）do-while语句。

（3）while语句。

（4）foreach语句。

（5）goto语句和if语句构成循环。

2. for循环

在这几条循环语句中，for语句最为灵活，不仅可用于循环次数已经确定的情况，也

可用于循环次数虽不确定，但给出了循环继续条件的情况。

1）for语句的一般格式

```
for(变量赋初值；循环继续条件；循环变量增值)
 { 循环体语句；}
```

2）for语句的执行过程

（1）执行“变量赋初值”表达式。

（2）判断“循环继续条件”表达式。如果表达式的值为真，执行（3），否则，转至（4）。

（3）执行循环体语句，并求解“循环变量增值”表达式，然后转向（2）。

（4）执行for语句的下一条语句。

3）说明

（1）“变量赋初值”“循环继续条件”和“循环变量增值”部分均可缺省，甚至全部缺省，但其间的分号不能省略。

（2）“循环变量赋初值”表达式，既可以是给循环变量赋初值的赋值表达式，也可以是与此无关的其他表达式。

例如，`for(sum=0;i<=100;i++) sum += i;`

`for(sum=0,i=1;i<=100;i++) sum += i;`

（3）“循环继续条件”部分是一个逻辑量，除一般的逻辑表达式外，也允许是数值或字符表达式。

案例4

计算1!+2!+3!+4!+5!。

分析：!表示阶乘，如4!=4*3*2*1，本题既要计算乘积又要求和，具体代码如下所述。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
   {
       static void Main(string[] args)
      {
         int s,t;
         s=0;  t=1;
         for(int i=1;  i<=5;  i++)
```

```
            {
                t=t*i;
                s=s+t;
            }
            Console.WriteLine("计算结果为:{0}",s);
        }
    }
}
```

运行结果如图2-6所示。

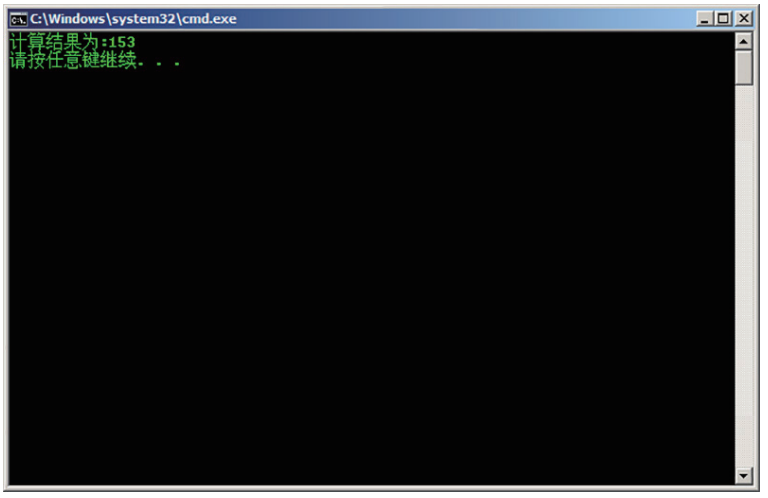

图2-6　运行结果

3. while语句

1）一般格式

```
while(循环逻辑条件)
{
   循环体语句;
}
```

2）执行过程

（1）求解“循环逻辑条件”表达式。如果表达式的值为真则跳转到（2），否则跳转到（3）。

（2）执行循环体语句组，然后跳转到（1）。

（3）执行while语句的下一条。

显然，while循环是for循环的一种简化形式。

案例5

计算并输出所有的水仙花数。

分析：水仙花数是一个三位数，它必须满足百位的立方、十位的立方、个位的立方之和与这个数相等。例如153=1*1*1+5*5*5+3*3*3，本题的关键在于求出任意一个三位数的百位、十位和个位。计算的方法很多，本题通过字符串的方式来获取，如任意一个三位数，从左边截取第一位为百位，从第二位开始截取，长度为1是十位，从第三位开始截取，长度为1是个位。具体代码如下所述。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
   {
       static void Main(string[] args)
       {
           int i, a, b, c;
           string s;
           i = 100;
           while (i<=999)
          {
              s =i.ToString();
              a=Convert.ToInt32(s.Substring(0,1));
              b=Convert.ToInt32(s.Substring(1,1));
              c=Convert.ToInt32(s.Substring(2,1));
              if (i==a * a * a + b * b * b + c * c * c)
                  Console.WriteLine("水仙花数为:{0}\n",i);
               ++i;
           }
       }
   }
}
```

运行结果如图2-7所示。

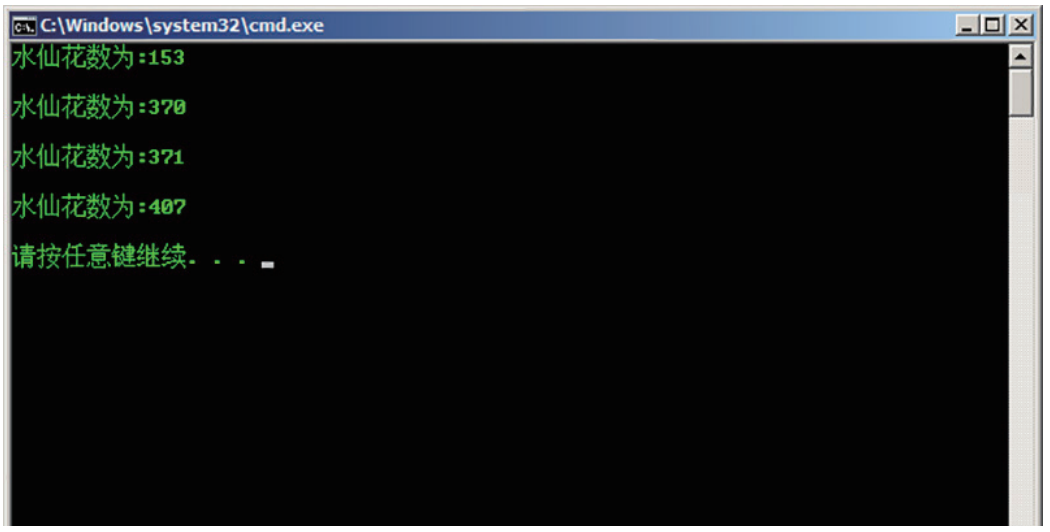

图2-7　水仙花数的运行结果

4. 循环嵌套

循环语句的循环体，又包含另一个完整的循环结构，称为循环的嵌套。循环嵌套的概念对所有高级语言都是一样的。

for语句和while语句允许嵌套，do-while语句也不例外。

5. break语句与continue语句

为了使循环控制更加灵活，C#语言提供了break语句和continue语句。

1）一般格式

```
break;
continue;
```

2）功能

（1）break：强行结束循环，循环到此结束，执行循环后的下一条语句。

（2）continue：结束当前这次循环，但整个循环没有结束，会根据循环条件判断后继续下一次循环。

3）说明

（1）break能用于循环语句和switch语句中，continue只能用于循环语句中。

（2）循环嵌套时，break和continue只影响包含它们的最内层循环，与外层循环无关。

6. foreach循环

案例6

计算任意一个字符串中的数字、字母、标点符号和其他符号的个数。

分析：该案例可以通过系统提供的函数来分别判断某个字符是不是数字、字母、标点符号或其他符号，如果是，则变量相应加1。具体代码如下所述。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
   {
       static void Main(string[] args)
       {
          string input;
          int a=0,b=0,c=0,d=0;
          Console.WriteLine("请输入一个字符串:\n");
          input=Console.ReadLine();
          foreach(char ch in input)   //"3we5,"
          {
               if(char.IsDigit(ch))
                  a++;
               else  if(char.IsLetter(ch))
                     b++;
               else  if(char.IsPunctuation(ch))
                     c++;
               else
                     d++;
           }
           Console.WriteLine("字母有{0}\n数字有{1}\n符号有{2}\
n其他有{3}\n",b,a,c,d);
       }
    }
}
```

运行结果如图2-8所示。

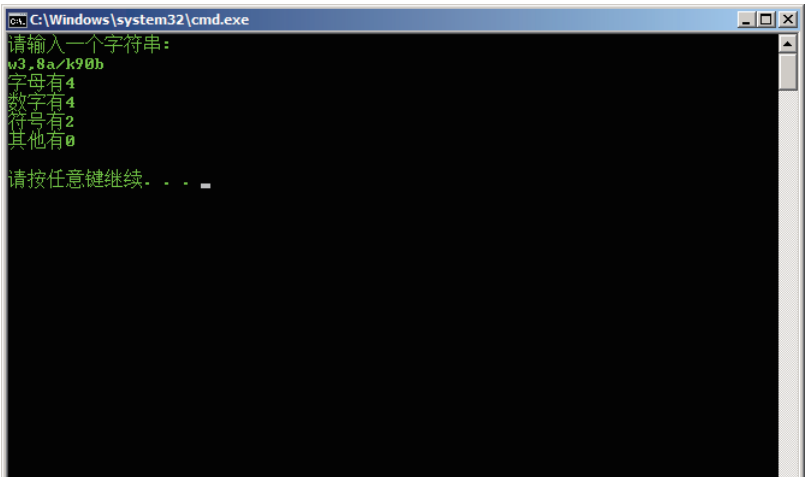

图2-8　运行结果

本章总结

本章首先讲述了C#的基础知识，如常量、变量的定义和使用，操作符和表达式的应用等；其次讲解了C#中的常用语句、分支语句、循环语句，如for循环、while循环、foreach循环以及continue和break语句的区别。

练习与实践

【问答题】

1．什么是表达式，C#中的逻辑表达式由哪些组成？

2．程序有哪几种结构，分别有什么特点？

3．C#中的循环语句有哪几种？while循环和do…while循环有什么区别？

【实训任务】

C#中各种语句的综合运用	
项目背景介绍	在C#的控制台环境中熟悉各个语句，并能综合运用。
设计任务概述	通过循环语句输出一个菱形，如下所示。 * *** ***** ******* ***** ** *
实训记录	
教师考评	评语： 辅导教师签字：