第 2 章　变量与表达式

本章要点

- 变量的命名、类型以及赋值的方法
- 表达式以及运算符的优先级
- 值类型和引用类型

变量是 C#中的一个基本单位。通常而言，任何程序都离不开变量的参与。读者在理解变量的意义时，可以考虑程序的意义。程序之所以有其存在的价值，就是因为它可以提高我们的工作效率。变量的存在使其成为可能，这是由于变量可以存储不同的值或数据。

C#中的表达式是由运算符、变量以及标点符号依据一定的语法规则组合而成的。表达式是构成程序的主要元素。

2.1 变　　量

变量代表了存储单元，每个变量都有一个类型。这决定了这个变量可以存储什么值。C#是类型安全的语言，C#编译器会保证存储在变量中的值总是恰当的类型。可以通过赋值语句的操作来改变变量的值。

使用变量的一条重要原则是：变量必须先定义后使用。

变量可以在定义时赋值，也可以在定义的时候不赋值。一个定义时被赋了值的变量很好地定义了一个初始值。而一个定义时不被赋值的变量没有初始值。要给一个定义时没有被赋值的变量赋值必须是在一段可执行的代码中进行。

2.1.1 变量的声明

变量的作用非常重要，变量代表一个特定的数据项或值。与常量不同，变量可以反复赋值。变量关系到数据的存储，它就像一个盒子，里面存放着各种不同的数据。尽管计算机存储的数据都是 0 和 1 的组合，但变量却是存在类型差别的。读者比较熟悉的 int 类型就是一种变量类型。同样，变量还存在着许多其他类型。

若在程序中使用未声明的变量，编译时会报错。

C#中可以声明的变量类型并不仅限于 C#预先定义的那些。因为 C#有自定义类型的功能，开发人员可根据自己的需要，建立各种特定的数据类型，以方便存储复杂的数据。

C#中规定，使用变量前必须声明。变量的声明同时规定了变量的类型和

变量的名称。变量的声明采用如下的规则：

type name;

例如，下面的语句声明了一个整型变量 f：

int f;

C#中并不要求在声明变量的同时初始化变量(即为变量赋值)，但是为变量赋值通常是程序员的良好习惯，例如：

int d = 2;

上面这句代码在声明了 int 型变量的同时将其初值设置为 2。

可以在同一行中同时声明多个变量，例如：

bool a1 = true, a2 = false;

每一个变量都有自己的名称，但 C#规定不能用任意的字符作为变量名。变量的命名规则应该遵循标识符的命名规则。

2.1.2 变量的命名

在任何一种语言中，变量的命名都是有一定的规则的，当然 C#也不例外，若在使用中定义了不符合一定规则的变量，系统会自动报错。

基本的变量命名规则如下：

（1）变量名的第一个字符必须是字母、下划线(_)或者“@”。

（2）除第一个字符外，其余的字符可以是字母、数字、下划线的组合。

（3）不可以使用对 C#编译器而言有特定含义的名称(即 C#语言的库函数名称和关键字名称)作为变量名，如 using、namespace、struct 等。

例如，下面的变量名是错误的：

34abcde　　//变量名的第一个字符必须是字母、下划线( )或者“@”

Class　　//Class 是关键字

a-b-d　　//其余字符是字母、数字、下划线的组合，不能出现”-”

而下面的命名则是正确的：

abc

_myHello

还要强调一点，C#对于大小写字母是敏感的，所以在声明以及使用变量的时候要注意这些，例如 Variable、variable、VARIABLE 是 3 个不同的变量。

上面我们举了一些变量命名的例子。在实际的应用中，还是应该取那些有实际意义的英文名称，可以方便自己的操作，亦可使别人读代码时清晰明了，例如：

Name

Age

Length

在变量的命名过程中，遵循一定的规则是必需的。在.NET Framework 命名空间中有两种命名约定，分别为 PascalCase 和 camelCase。它们都应用到由多个单词组成的名称中，并指定名称中的每个单词除了第一个字母大写外，

其余字母都是小写。在 camelCase 中还有一个规则，即第一个单词须以小写字母开头。

下面是 PascalCase 变量命名的举例：

Age

SumOfApple

DayOfWeek

下面是 camelCase 变量命名的举例：

age

sumOfApple

dayOfWeek

Micorsoft 建议：对于简单的变量使用 camelCase 规则，而比较高级的命名则使用 PascalCase 规则。

2.1.3 变量的种类、赋值

在 C#语言中，我们把变量分为 7 种类型，分别是静态变量(Static Variables)、实例变量(Instance Variables)、数组变量(Array Variables)、值参数(Value Parameters)、引用参数(Reference Parameters)、输出参数(Output Parameters)、局部变量(Local Variables)。

下面举一个例子来给大家一个较为明确的概念：

```
class myClass
{
    int y = 2;
    public static int x = 1;
    bool Function(int[] s, int m, ref int i, out int j)
    {
        int w = 2;
        j = x + y + i + w;
    }
}
```

在上面的代码中，x 是静态变量，y 是实例变量，s 是数组变量，m 是值参数，i 是引用参数，j 是输出参数，w 是局部变量。

1．非静态变量

不带有 static 修饰符声明的变量称为实例变量(非静态变量)，例如：

```
int s = 2;
```

针对类中的非静态变量而言，一旦一个类的新的实例被创建，直到该实例不再被应用从而所在空间被释放为止，该非静态变量将一直存在。一个类的非静态变量应该在初始化时赋值。

2．静态变量

带有 static 修饰符声明的变量是静态变量。一旦静态变量所属的类被装载，直到包含该类的程序运行结束时，它将一直存在。静态变量的初始值就是该变量的默认值。静态变量最好在定义时赋值，例如：

```
public static int s = 5;
```

在使用静态变量时，不需要对其所在的类进行实例化(即不使用 new 关键字)就可以直接通过类来使用。

例 2.1 使用静态变量来记录网站的访问人数。

【步骤 1】启动 Visual Studio 2012 开发工具

【步骤 2】选择“文件”→“新建”→“项目”，弹出如图 2.1 所示的对话框，可以看到，左边是项目类型，右边是已安装的模板，包括“Windows 窗体应用程序”、“类库”、“控制台应用程序”等模板，它们指定了要创建的应用程序的类型。

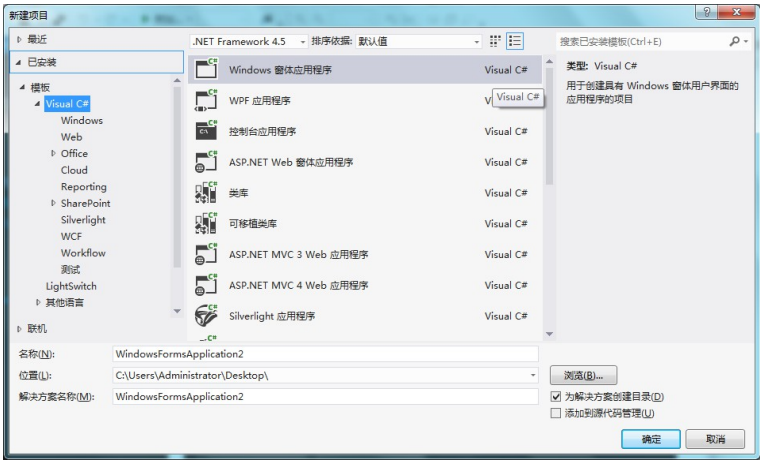

图 2.1 “新建项目”对话框

【步骤 3】在左边选择“Visual C#”，右边选择“控制台应用程序”，名称框中输入“ch02-1”，并选择项目的存放位置，如图 2.2 所示。

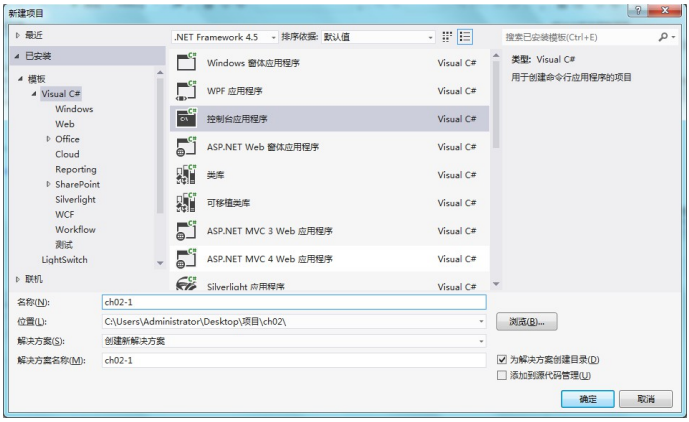

图 2.2 输入名称并选择存放位置

【步骤 4】确认“为解决方案创建目录”复选框已被选中，然后单击“确定”按钮。出现如图 2.3 所示的项目设计界面。

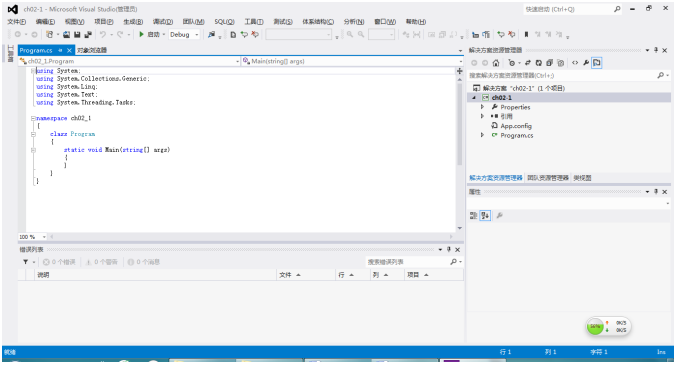

图 2.3　项目设计界面

在开始编写代码之前，先来解释一下一些名词

（1）namespace 关键字

namespace（命名空间）是 C#中组织代码的方式，这样可以把紧密相关的一些代码放在同一个命名空间中，大大提高管理和使用的效率。在这段代码中，VS 自动以项目的名称作为命名空间的名称 ch02_1

（2）using 关键字

在 C#中，使用 using 关键字来引用其他命名空间。在这段代码模板生成时，VS 就已经自动添加了 5 条 using 语句。

（3）class 关键字

C#是一种面向对象的语言，使用 class 关键字表示类，我们编写的任何代码都应该包含在一个类里面，类要包含在一个命名空间中。在程序模板生成时，VS 自动为我们起了一个类名 Program。如果不喜欢，可以改掉它。

（4）Main 方法

C#中的 Main（）方法是程序的大门，应用程序从这里开始运行。要注意的是，C#中的 Main（）方法首字母必须大写，Main（）方法的返回值可以 void 或 int，Main（）方法中的命令行参数是可以没有的。

（5）解决方案“ch02-1”。它是最顶级的解决方案文件，每个应用程序都有一个类似的文件。根据该项目的创建路径，找到“ch02-1”项目文件夹，就可以看到该解决方案文件，即“ch02-1.sln”。每个解决方案文件都包含了对一个或者多个项目文件的引用。

（6）ch02-1。“ch02-1”是 C#的项目文件，其名称为“ch02-1. csproj”。每个项目文件都引用一个或者多个包含项目源代码的文件。

（7）Properties。查看“ch02-1”项目文件夹可以知道，“Properties”是其中的一个文件夹，包含一个名为“AssemblyInfo.cs”的文件，它是一个特殊的文件，可以用它在一个属性中添加如“作者”、“日期”等属性。

（8）引用。该文件包含对程序可用的已编译代码的引用。

（9）Program.cs。从代码窗口上方的选项卡中，也可以找到

“Program.cs”，很明显，它是一个 C#源代码文件，用户编写的代码都包含在这个文件中，同时 VS2012 自动创建的一些源代码也被保存在其中。

【步骤 5】编写代码

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ch02_1
{
    class Program
    {
        static void Main(string[] args)
        {
            person.ShowTotalPeople();
            person MrGreen = new person(); //格林先生进入会场
            person.ShowTotalPeople();
            person MrSmith = new person();
            person MrAllen = new person();
            person.ShowTotalPeople();
        }
        class person
        {
            public static int TotalPeople = 0;
            //static 静态方法
            public static void ShowTotalPeople()
            {
                  Console.WriteLine("现在共有{0}个人访问网站",
TotalPeople);
            }
            //构造函数
            public person()
            {
                TotalPeople++;
            }
        }
    }
```

【步骤 6】生成并运行控制台应用程序

编写好程序代码后，接下来应当生成控制台应用程序，即编译代码并生成一个可执行的程序，具体方法是：选择生成/生成解决方案菜单项，生成的过程中会在代码编辑器的下方出现一个输出窗口，如图 2.4 所示。

图 2.4　输出窗口

在输出窗口中，提示程序编译完成，并显示了生成过程中发生的错误和警告。本程序相对比较简单，在输入正确的情况下，不会有任何的错误和警告

生成成功后，可以运行程序查看结果。具体的方法是：选择调试/开始执行（不调试）菜单项，即弹出一个命令窗口，显示程序的运行结果，如图 2.5 所示。

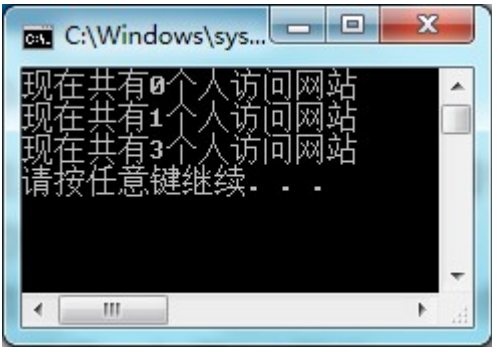

图 2.5　运行结果

在命令窗口中，显示了执行的结果，“请按任意键继续...”是自动生成的，等待用户按任意键终止执行，即关闭命令窗口。

通过上面的程序，我们基本上可以了解到静态变量的使用方法及效果。可以直接使用 person.TotalPeople 来存取变量，也可以直接使用 person.ShowTotalPeople() 来执行 ShowTotalPeople() 静态方法，由于 TotalPeople 是静态变量，所以不论在哪一个对象中，我们得到的都是同一个 TotalPeople 变量，所以可以用来累加。

3．数组变量

数组变量随该数组实例的存在而存在。每一个数组元素的初始值都是该数组元素类型的默认值。数组元素最好在初始化时被赋值。有关数组的内容，在后面的章节中将会有专门的介绍。

4．局部变量

局部变量是指在一个独立的程序块、for 语句、switch 语句或者 using 语句中声明的变量，它只在该范围内有效。当程序运行到这一范围时，该变量即开始生效，程序离开时变量就失效了。

与其他几种变量类型不同的是，局部变量不会自动被初始化，所以也就没有默认值，在进行赋值检查的时候，局部变量被认为是没有被赋值。

在局部变量的使用过程中要注意的是，变量在定义前是不可以使用的。

关于值参数、输出参数和引用参数，将会在后面的函数章节中予以讲解。

2.1.4 变量类型之间的转换

在程序的设计中，常常会遇到变量的类型转换问题。比如在进行数学四则运算时，int 类型的数值与 double 类型的数值可能混在一起进行运算，这样变量之间的类型转换就应运而生。

Visual C#中的变量类型转换常见的主要有以下 4 种方式：

- 通过隐式转换。
- 通过强制类型转换。
- 使用 ToString()方法。
- 使用 Convert 类。

1．隐式转换

隐式转换又称自动类型转换，若两种变量的类型是兼容的或者目标类型的取值范围大于源类型时，就可以使用隐式转换。隐式转换表如表 2.1 所示。

表 2.1　隐式转换的原类型以及目标类型对应表

原类型	可以转换至下列目标类型
sbyte	short、int、long、float、double、decimal
byte	short、ushort、int、uint、long、float、double、decimal
char	ushort、int、uint、long、float、double、decimal
int	long、float、double、decimal
uint	long、ulong、float、double、decimal
short	int、long、float、double、decimal
ushort	int、uint、long、float、double、decimal
long	float、double、decimal
ulong	float、double、decimal
float	double

2．强制类型转换

在程序的编写以及应用时，我们会发现上面讲述的隐式转换不能满足所有的需要，很多时候还是需要进行强制类型转换的。强制类型转换是一种指令，它告诉编译器将一种类型转换为另外一种类型。强制转换的缺点是可能产生的结果不够精确。具体的强制类型转换语法为：

(target-type)变量或表达式;

例 2.2 类型转换。

关键程序代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ch02_2
{
    class Program
    {
        static void Main(string[] args)
        {
            double a = 3.2;
            int b = 5;
            int c = (int)a + b;
            Console.WriteLine("c="+c);
        }
    }
}
```

运行结果为：

c=8

本例中若把代码语句 int c = (int)a + b;写为 int c = a + b;就会报错，这是因为直接的隐式转换不能把 double 类型的数据转化为 int 类型的数据，故在此例子中我们进行了强制转换。

3．ToString()方法

ToString()方法主要用于将变量转化为字符串类型，该方法是 C#语言中非常常见的一个方法。

字符串是 C#中的另外一种类型，表示一组字符，使用双引号包含，比如“Visual C#”就是一个常见的字符串。

前面我们介绍的各种类型的变量都可以通过 ToString() 方法转换为 String 类型，具体看下面把 int 型变量转化为 string 类型的小例子：

```
int i = 200;
string s = i.ToString();
string t = i.ToString()+”123”;//此处“+”为字符串的连接符号
```

这样字符串类型变量 s 的值就成为“200”，字符串类型变量 t 的值变成”200123”。

4．Convert 类

前面介绍了 ToString()方法，并且给出了 int 类型转化为 string 类型的实例，可是细心的读者会发现，这个方法无法实现逆向转换，即无法把一个 string 类型的变量转化为 int 类型的变量。

下面介绍可以解决上述问题的方法——使用 Convert 类进行显式转换。Convert 类提供了很多常用的转换方法，详见表 2.2。

表 2.2 Convert 类的常用方法

方 法	说 明
ToBase64CharArray()	将 8 位无符号整数数组的子集转换为用 Base64 数字编码的 Unicode
ToBase64String()	将 8 位无符号整数数组转换为它的等效 String 表示形式
ToBoolean()	将指定的值转换为等效的布尔值
ToByte()	将指定的值转换为 8 位无符号整数
ToChar()	将指定的值转换为 Unicode 字符
ToDateTime()	将指定的值转换为 DateTime 类型
ToDecimal()	将指定的值转换为 Decimal 数字
ToDouble()	将指定的值转换为双精度浮点数字
ToInt16()	将指定的值转换为 16 位有符号整数
ToInt32()	将指定的值转换为 32 位有符号整数
ToInt64()	将指定的值转换为 64 位有符号整数
ToSByte()	将指定的值转换为 8 位有符号整数
ToSingle()	将指定的值转换为单精度浮点数字
ToString()	将指定的值转换为与其等效的 String 形式
ToUInt16()	将指定的值转换为 16 位无符号整数
ToUInt32()	将指定的值转换为 32 位无符号整数
ToUInt64()	将指定的值转换为 64 位无符号整数

2.2 常 量

常量就是值在程序整个生命周期内始终不变的量。在声明常量时，要用到 const 关键字。常量在使用的过程中，不可以对其进行赋值的改变，否则系

统会自动报错。

常量声明的基本语法为：

```
[private/public/internal/protected] const
[int/double/long/bool/string/...] VariableName = value;
```

下面是一个声明常量的具体例子：

```
private const double PI = 3.1415926;
```

例 2.3 输入数量，求总价。

关键程序代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ch02_3
{
    class Program
    {
        static void Main(string[] args)
        {
            const double price = 400;
            int n;
            Console.WriteLine("请输入数量,以 0 退出");
            n = Convert.ToInt32(Console.ReadLine());
            while (n != 0)
            {
                Console.WriteLine("单价为{0}，数量为{1}，总价为
{2}",price,n,price*n);
                n = Convert.ToInt32(Console.ReadLine());
            }
        }
    }
```

以上程序当数量输入为 0 时，程序就运行结束了。结果如图 2.6 所示。

图 2.6　例 2.3 的运行结果

2.3 表　达　式

前面主要介绍了变量和常量，以及它们的声明、赋值和使用方法。本小节将介绍如何处理这些变量和常量。

C#中的表达式是由运算符、变量以及标点符号依据一定的规则组合创建起来的。运算符的范围非常广泛，简单的复杂的都有。在本章中，主要介绍数学运算符和赋值运算符，逻辑运算符暂时不讨论。

2.3.1 数学运算符

C#中的数学运算符有 5 种：

+　加法运算符
-　减法运算符
*　乘法运算符
/　除法运算符
%　取余运算符

上面的 5 种运算符都是二元的。而“+”与“-”运算符也可以是一元的，例如“++”和“--”，称为自增和自减运算符。以“++”为例，具体用法如下：

```
int i = 1;
i++;
```

此时 i 的值就变为了 2，“i++”这个表达式可以解释为“i=i+1”；表达式“++i”与“i++”的含义又有不同，下面的例子简单地进行了解释：

```
//前置++
int a = 1;
int b;
b = ++a; //运行结果是 b=2
//后置++
```

```
int a = 1;
int b;
b = a++; //运行结果是 b=1
```

通过对以上两个简单程序的对比，可以得知表达式 a++是先赋值、后进行自身的运算，而++a 正好是相反的，先进行自身的运算，而后再赋值。

1．加法运算符

加法运算符可以运用于整数类型、实数类型、枚举类型、字符串类型和代理类型。在这里只需要知道这些运算符可以对不同类型的变量进行运算就可以了。

我们知道，在数学运算中，结果可能是正无穷大、负无穷大，当然，也可能结果不存在。在 C#中，在处理这些特殊情况的时候，假设表达式为 Z=X+Y，X 与 Y 都是非 0 的有限值，如果 X 和 Y 数值相等，符号相反，则 Z 为正 0，如果 X+Y 结果太大，那么目标类型 Z 就被认作与 X+Y 符号相同的无穷值。如果 X+Y 太小，目标类型也无法表示，则 Z 为与 X+Y 同符号的零值。

另外，枚举类型和代理类型(Delegate)也可以进行加法运算。

例 2.4 枚举类型的加法运算。

关键程序代码如下：

```
class Program
    {
        enum Season
        { January,
February,March,April,May,June,July,August,September,October,Novembe
r,December};
        static void Main(string[] args)
        {
            Season w1 = Season.February;
            Season w2 = Season.June;
            Season w3 = w1 + 3;
            Console.WriteLine(w1);
            Console.WriteLine(w2);
            Console.WriteLine(w3);
        }
    }
```

程序运行结果如图 2.7 所示。

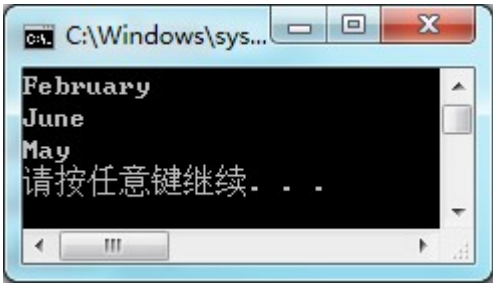

图 2.7 例 2.4 的运行结果

加法运算符还可以作用于 delegate 类型的变量，我们称之为合并。原型为：

D operator+(D x, D y);

其中 D 是一个 delegate 类型。

其中如果两个操作数是同一 delegate 类型 D 时，则加法运算符执行代表合并的运算。如果第一个操作数为 Null，那么结果是第二个操作数的值。反之，如果第二个操作数为 Null，则结果是第一个操作数的值。

2．减法运算符、乘法运算符、除法运算符

减法运算符、乘法运算符、除法运算符与上面所讲的加法运算符很类似，在这里就不再赘述。另外，有几点还是需要注意的，乘法运算符、除法运算符只适用于整数以及实数之间的操作；而且在使用除法运算符的时候，默认的返回值的类型与精度最高的操作数类型相同。比如，5/2 的结果是 2，而 5.0/2 的结果是 2.5。如果两个整数类型的变量相除又不能整除的话，返回的结果是不大于被除数的最大整数。

3．取余运算符

取余运算(又称求模运算)符用来求除法的余数，在 C#语言中，取余运算既适用于整数类型，也同样适用于浮点型。如 7%3 的结果为 1，7%1.5 的结果为 1。

2.3.2 赋值运算符

赋值运算符分为两种类型，第一种是简单赋值运算符，就是前面在程序里面已经多次使用的“=”号；第二种是复合赋值运算符，包含 5 类，具体的如表 2.3 所示。

表 2.3 复合赋值运算符

赋值运算符	赋值运算符类别	赋值运算符范例表达式	赋值运算符解释
+=	二元	a += value	a 被赋予 a 和 value 的和
-=	二元	a -= value	a 被赋予 a 和 value 的差
/=	二元	a /= value	a 被赋予 a 和 value 相除的商
*=	二元	a *= value	a 被赋予 a 和 value 的乘积
%=	二元	a% = value	a 被赋予 a 和 value 相除的余数

复合赋值的基本语法为：

a oper= value;　//这里的“oper”为某个数学运算符

这种语法形式与如下语法形式的作用等同：

a = a　oper　value;

以 a　oper=　value;为例，复合赋值的步骤如下。

(1)如果所选运算符的返回类型可以隐式转换成 value 的数据类型，则执行 a　=　a　oper　value;的运算，除此之外，仅对 a 执行一次运算。

(2)否则，若所选运算符是一个预定义运算符，则所选运算符的返回值类型可以显式地转化为 a 的类型，且 value 可以隐式地转化成 a 的类型，那么该运算符等价于 a　=　(T)(a　oper　value);运算，这里 T 是 a 的类型，除此之外，a 仅被执行一次。

(3)否则，复合赋值是无效的，且会产生编译时错误。

2.3.3 运算符的优先级

数学中所涉及的运算符优先级内容称为四则运算，口诀是：先乘除、后加减，有括号先算括号里面的。

在程序语言中，也完全遵从这些规则。

但是需要特别注意的是：程序中的运算符要比数学中的多一些，比如++、--、%等运算符就是一般数学运算中所没有的。另外数学运算符的优先级要排在赋值运算符之前。

下面举两个例子以便说明得更清楚些。

(1)　a=(b+c)*d;

在这句代码中，是先计算 b 与 c 的和，再用这个和与 d 进行相乘运算，所得到的积赋值给 a。

(2)　a%=(++b-c*d)%e;

可以将这句代码改写为：a=a%((++b-c*d)%e); 这样就可以按照基本的运算法则来计算了。具体的优先级如表 2.4 所示。

表 2.4　几种运算符优先级的比较

运算符优先级	运　算　符
优先级从高到低	++(前缀之用)、--(前缀之用)
	*、/、%
	+、-
	=、*=、/=、%=、+=、-=
	++(后缀之用)、--(后缀之用)

2.4 数 据 类 型

现实世界的数据类型是多种多样，我们必须让计算机了解需要处理什么

样的数据，以及采用哪种方式进行处理，按什么格式保存数据等。其实，任何一个完整的程序都可以看成是一些数据和作用于这些数据上的操作。每一种高级开发语言都为开发人员提供一组数据类型，不同的语言所提供的数据类型不尽相同。

对于程序中的每一个用于保存信息的量，在使用时都必须声明其数据类型，以便编译器为它分配内存空间。在 C#语言中，数据类型可以分为两种：值类型(Value Type)和引用类型(Reference Type)。

2.4.1 值类型

值类型变量存放的是数据本身，引用变量存储的是数据的引用。下面的语句声明了两个 int 型变量，它们的值相同，但是在内存中却占用不同的地址，彼此相互独立：

```
int i1 = 4;
int i2 = i1;
```

在 C#中，值类型可以分为简单类型、结构类型、枚举类型。

1．简单类型

简单类型(Simple Types)是直接由一系列元素组成的数据类型。C#语言给我们提供了一组已经定义好了的简单类型。单纯地从计算机的表示角度来看，这些简单类型可以分为整数类型、布尔类型、字符类型和实数类型。

(1) 整数类型

整数类型，顾名思义，就是变量的值为整数的值类型。计算机语言中的整数与数学上的整数定义有点差别，这是由于计算机的存储单元有限所造成的。C#语言中的整数类型分为 8 类：短字节型(sbyte)、字节型(byte)、短整型(short)、无符号短整型(ushort)、整型(int)、无符号整型(uint)、长整型(long)、无符号长整型(ulong)。一些变量名称前面的“u”是“unsigned”的缩写，表示不能在这些类型的变量中存储负号。这些不同的整数类型可以用于存储不同范围的数值，占用不同的内存空间，它们的列表如表 2.5 所示。

表 2.5 整型数据类型的分类

数据类型	特 征	取值范围
sbyte	有符号 8 位整数	−128 ~ 127
byte	无符号 8 位整数	0 ~ 255
short	有符号 16 位整数	−32768 ~ 32767
ushort	无符号 16 位整数	0 ~ 65535
int	有符号 32 位整数	−2,147,483,648 ~ 2,147,483,647
uint	无符号 32 位整数	0 ~ 4,294,967,295
long	有符号 64 位整数	−9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
ulong	无符号 64 位整数	0 ~ 18,446,744,073,709,551,615

(2) 布尔类型

在 C#语言中，布尔类型只有两种取值，即 true 或 false。在编写应用程序的流程时，布尔型的变量有着非常重要的作用。

(3)　字符类型

字符包括数字字符、英文字母、表达符号等。C#提供的字符类型按照国际上公认的标准，采用 Unicode 字符集。字符的定义方法如下：

```
char ch = 'A';
```

上面这行代码意思是：把‘A’这个字符值赋给变量 ch。

同样，我们可以把转义字符以及十六进制转义符赋值给字符类型的变量，有 C 语言或者 C++基础的读者对此应该比较了解。表 2.6 列出了一些转义字符及其含义。

表 2.6　转义字符及其含义

转义字符	字 符 名
\’	单引号
\”	双引号
\\	反斜杠
\0	空字符
\a	感叹号(产生鸣响)
\b	退格
\f	换页
\n	换行
\r	回车
\t	水平制表符
\v	垂直制表符

有关于字符和字符串的内容，我们将在第 4 章中进行详细的讲解。

(4)　实数类型

在 C# 中，实数类型分为单精度(Float)、双精度(Double)和十进制(Decimal)类型，它们的差别在于取值范围和精度不同。计算机对实数的运算速度大大低于对整数的运算。在对精度要求不高的计算中，我们可以采用单精度型，而采用双精度的结果将更为精确。Decimal 类型主要是为了方便在金融和货币方面的计算。在现代的企业应用程序中，不可避免地要进行大量的这方面的计算和处理，而在目前采用的大部分程序设计语言中，程序员都要自己定义货币类型等，这是一个遗憾。为此，C#专门提供这种数据类型来弥补这一遗憾。

当定义一个 Decimal 类型变量并且给其赋值的时候，使用 m 后缀以表示它是一个十进制类型，例如：

```
Decimal de = 2.38m;
```

若在这里我们把语句改写为：

```
Decimal de = 2.38;
```

那么在 Decimal 型变量 de 被赋值前，它将被编译器当作双精度(Double)类型来处理。

2．结构类型

一个结构类型(struct Types)可以声明构造函数、常数、字段、方法、属性、索引、操作符和嵌套类型。尽管列出来的功能看起来像一个成熟的类，但是在 C#中，结构和类的区别在于结构是一个值类型，而类是一个引用类型。与 C++相比，这里可以使用结构关键字定义一个类。

使用结构的主要思想是用于创建小型的对象，如 Point 和 FileInfo 等，这样可以节省内存，因为结构没有如类对象所需的那样有额外的引用产生。例如，当声明含有成千上万个对象的数组时，这将会引起巨大的差异。

定义结构和定义类几乎是完全一样的，具体见下面的例子：

```
struct myColor
{
    public int Red;
    public int Green;
    public int Blue;
}
```

到目前为止，我们看这个声明很像一个类。可以这样来使用所定义的这个结构：

```
myColor mc;
mc.Red = 255;
mc.Green = 0;
mc.Red = 0;
```

mc 就是一个 myColor 结构类型的变量。上面声明中的 Public 表示对结构类型的成员的访问权限，有关访问的细节问题我们将在后面部分详细讨论。

mc.Red=255;这条语句是对结构成员赋值。结构类型包含的成员类型没有限制，可以相同，也可以不同。例如还可以在此结构类型中添加类型为 string 的成员 ColorName，如下所示：

```
struct myColor
{
    public int Red;
    public int Green;
    public int Blue;
public string ColorName;
}
```

还可以把结构类型作为另一个结构的成员的类型，这也没有任何问题，例如：

```
struct Ball
```

```
{
public double Weight;
public double Radius;
    struct myColor
{
        public int Red;
        public int Green;
        public int Blue;
        public string ColorName;
}
}
```

这里，Ball 这个结构中又包括了 myColor 这个结构，myColor 结构包括 Red、Green、Blue、ColorName 这 4 个成员。

3．枚举类型

枚举(Enumeration)实际上是为一组在逻辑上密不可分的整数值提供便于记忆的符号。当我们希望变量提取的是一个固定集合中的值时，就可以使用枚举。例如，可以定义一个代表手机的枚举类型：

```
enum MobilePhone{Nokia, MotoRola, TCL, LG, Bird, NEC, Apple};
```

这时我们就可以声明 MobilePhone 这个枚举类型的变量：

```
MobilePhone mp;
```

下面介绍一下枚举类型与前面讲的结构类型的区别。

结构是由不同类型的数据组成的一组新的数据类型，结构类型的变量的值是由各个成员的值组合而成。而枚举则不同，枚举类型的变量在某一时刻只能取枚举中某一个元素的值。例如上面的枚举类型 MobilePhone 的变量 mp 的取值就必然是 Nokia 到 Apple 之间的某个值。赋值方式如下：

```
mp = Apple;
```

枚举类型中的默认值是 int 类型，第一个默认值是 0，其后从 1 递增。当然也可以自己给它赋值，例如：

```
enum MobilePhone{Nokia=4, MotoRola, TCL, LG, Bird, NEC, Apple};
```

若是这样定义的话，那么从 MotoRola 开始，逐个从 4 递增 1(即 MotoRola=5，TCL=6...)。

2.4.2 引用类型

引用类型与 C++中的引用类似，因为你可以将它们视作类型安全的指针。与纯粹的地址不同(地址可能指向你预期的东西，也可能不是)，引用(在不是 Null 时)总是确保指向一个对象，这个对象具有指定的类型而且已经在堆上分配了。另外，引用可以是 Null，这表示它当前不引用或不指向任何对象。C# 中的引用类型有 4 种：

- 类
- 数组
- 代理
- 接口

对这 4 种引用类型，在以后的章节中都会详细地介绍。其中数组将在第 4 章给予讲解；类、代理、接口将在第 7 章进行详细的介绍。

2.5 案 例 实 训

1. 案例说明

定义一个结构体 Student，该结构含有学号(Sno)、姓名(Sname)和年龄(Sage)三个字段。而后使用结构数组把键盘输入的两条记录分别保存在结构数组中，其中涉及到强制转换的问题，最后为了验证强制转换的结果，我们把结构数组的内容输出。

2. 程序代码

关键程序代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ch02_5
{
    class Program
    {
        public struct Student
        {
            public string Sname;
            public double Sno;
            public int Sage;
        }
        static void Main(string[] args)
        {
            Student[] stud = new Student[2];
            int k;
             for (k=1; k<=2; k++)
```

```
            {
                Console.WriteLine("第" + k + "条数据：");
                Console.WriteLine("学号：");
                                                    stud[k-1].Sno
=Convert.ToDouble(Console.ReadLine());
                Console.WriteLine("姓名：");
                stud[k-1].Sname = Console.ReadLine();
                Console.WriteLine("年龄：");
                stud[k-1].Sage = int.Parse(Console.ReadLine());
            }
            for (k=1; k<=2; k++)
            {
                Console.WriteLine("显示第" + k + "条数据：");
                Console.Write("学号为：" + stud[k-1].Sno);
                Console.Write("  姓名：" + stud[k-1].Sname);
                Console.WriteLine("  年龄：" + stud[k-1].Sage);
            }
            Console.ReadLine();
        }
    }
}
```

3. 运行结果和程序解释

程序的运行结果如图 2.8 所示。

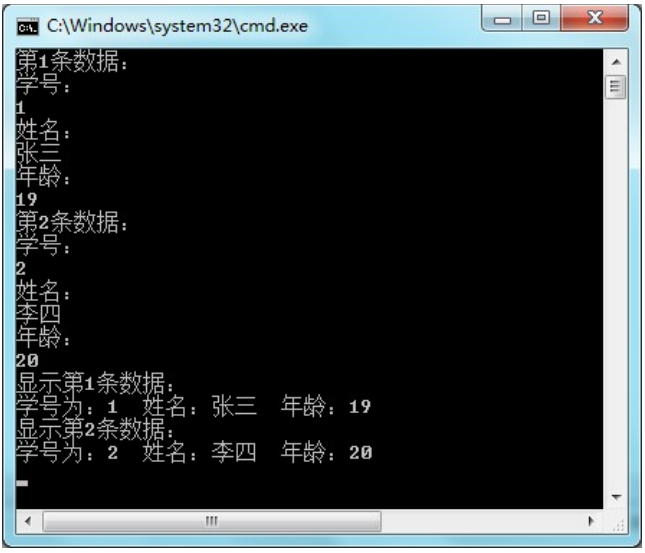

图 2.8　案例的运行结果

程序首先声明了一个结构体 Student，里面包含 Sname (姓名)、Sno (学号)、

Sage (年龄)，而后定义了结构体数组 stud，其数据的获取来源于键盘输入的结果。

2.6 小　　结

本章主要介绍了 C#应用程序的基础知识。包括变量、数据类型、表达式等，介绍了变量的声明、使用方法以及注意事项；还介绍了运算符的优先级和表达式的编写方法，重点介绍了值类型和引用类型的区分以及关联，通过实例进行解释说明。

在学习控制循环之前，本章的所有实例内容都是逐行顺序地执行的，在下面的章节中，将循序渐进地学习各种循环语句和条件语句，实现复杂的代码控制、以满足不同情况的编码需要。

2.7 习　　题

1. 计算题

求以下表达式的结果值。

(1) “Visual”+” C#”+” 2012”

(2) int a = 30, b = 22; b %= (++a - 3 * 4) % 5; 2

(3) int a=11;(a++*1/3） 3

(4) int a=6; a*=8/5+6; 42

2. 选择题

(1) 以下标识符中，正确的是_____。

A. _Time　　B. typeof

C. 3a　　D. a3#

(2) 引用类型和值类型的主要区别是_____。

A. 引用类型的实例可以赋值，而值类型不可以

B. 定义实例时，值类型可以生成引用，引用类型直接生成实例

C. 引用类型的实例可以赋值，而值类型不能

D. 定义实例时，值类型生成引用，引用类型直接生成实例

(3) 以下类型中，不属于值类型的是_____。

A. 整数类型

B. 布尔类型

C. 字符类型

D. 类类型

3.编程题

（1）新建控制台应用程序，输入“***”，输出“欢迎你，***”。