第 2 章　变量与表达式

主要内容

- 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
- 值类型以及引用类型

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# 是类型安全语言，并且每个 C# 编译器会保证存储在变量中的值总是恰当的类型。可以通过赋值语句操作或者使用”++”和”--”操作符改变变量的值。

2.1 变量

- 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

- 变量的声明采用如下的规则：

  <type> <name>;

  其中 type 是变量的类型， name 是变量的名称。

  例如：

  int a ;

  double d;

2.1.1 变量的声明

- 还可以在声明变量的同时为变量赋初值，如 ：

```
double d=2.4;
string s=“hello CSharp”;
```

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#.NET 也不例外，若在使用中定义了不符合一定规则的变量， C#.NET 语言系统会自动报错。
- 基本的变量命名规则如下：
  - 变量名的第一个字符必须是字母、下划线（ "_" ）或者 "@" 。
  - 除去第一个字母外，其余的字母可以是字母、数字、下划线的组合。
  - 不可以使用对 C# 编译器而言有特定含义的名字 (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 等等。
- 命名规则的第三条其实在写程序的时候系统会自动提示你的错误的，所以不必过于担心。

2.1.2 变量的命名

- 下面的变量名是错误的：
  - 345abc
  - class
  - w-d-m
- 下面的命名则是正确的：
  - wdm
  - _myVariable
  - VAR

2.1.2 变量的命名

- C#.NET 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。

2.1.2 变量的命名

- 在变量的命名过程中，命名遵循一定的规则是必须的。在 .NET Framework 名称空间中有两种命名约定，分别为 PascalCase 和 camelCase 。在名称中的大小写表示它们的前途。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。

2.1.2 变量的命名

- 下面是 PascalCase 变量命名的举例：
  - Age
  - SumOfApple
  - DayOfWeek

2.1.2 变量的命名

- 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，我们把变量分为七种类型，分别是：静态变量（Static Variables）、非静态变量（Instance Variables）、数组元素（Array Elements）、值参数（Value Parameters）、引用参数（Reference Parameters）、输出参数（Output Parameters）、局部变量（Local Variables）。

2.1.3 变量的种类、赋值

```
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;
  }
}
```

- 上面的代码中， x 是静态变量， y 是非静态变量， s[0]是数组元素， m 是值参数， i 是引用参数， j 是输出参数， w 是局部变量。

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）。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。同样鉴于定义赋值检查，一个类的非静态变量也应该在初始化时赋值。

静态变量

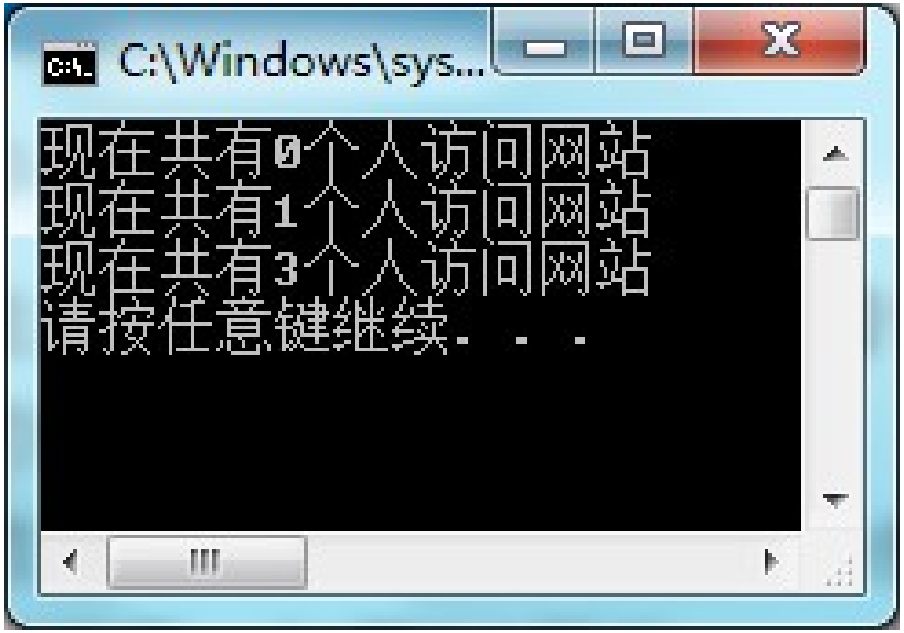
- 带有 static 修饰符声明的变量为静态变量。一旦静态变量所属的类被装载，直到包含该类的程序运行结束时，它将一直存在。静态变量的初始值就是该变量的默认值。为了便于定义赋值检查，静态变量最好在定义时赋值。如：
  - public static int s=5;
- 使用静态变量时，不需要对其所在的类进行实例化（即不使用 new 关键词）就可以直接通过类来使用。

关于使用静态变量的一个例子

- 例　2.1 使用静态变量来记录网站的访问人数

```
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++;
            }
        }
    }
}
```

程序的运行结果 ：


C:\Windows\sys...
现在共有0个人访问网站
现在共有1个人访问网站
现在共有3个人访问网站
请按任意键继续. . .

数组元素

- 数组元素也是变量的一种，该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。同样鉴于定义赋值检查，数组元素最好在初始时被赋值。

局部变量

- 局部变量是指一个独立的程序块，一个 For 语句，一个 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。

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 类型的数值和 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。

4 种转换方式

- 隐式转换
- 强制类型转换
- ToString （）方法
- Convert 类

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。

隐式转换的数据源类型以及目标类型对应表

数据源类型	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

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 ;

类型转换小例子

- 例 2.2 类型转换小例子。
- 程序代码：

```
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

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。
- 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，具体看下面一个把 int 型变量转化为 string 类型的小例子：

```
int i=200;
string s=i.ToString();
```

这样字符串类型变量 s 的值就是”200”。

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，即无法把一个 string 类型的转化为 int 类型的。

  下面介绍个可以解决上述问题的方法——使用 Convert 类进行显示转换。

Convert 类的常见方法

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

Convert 类的常见方法

方法	说明
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值始终不变的量。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;

- 下面是一个具体声明常量的例子：

  private const double PI=3.1415926;

2.3 表达式

- C# 中的表达式是由运算符、变量以及标点符号依据一定的法则组合创建起来的。运算符主要是用来定义类实例中表达式操作符的。运算符的范围非常之广泛，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，但是“＋”与“—”运算符也可以是一元的，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，下面的例子简单明了地解释了。

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

此程序运行的结果是： j=2 ；

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

此程序运行的结果是： j=1 ；

- 通过对以上 2 个简单程序的对比，可以得知表达式 i++ 是先赋值，后进行自身的运算，而 ++i 正好是相反的，先进行自身的运算，而后再赋值。

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、字符串类型和代表类型。在这里我们只需要知道这些操作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 就被认作和 X+Y 符号相同的无穷。如果 X+Y 太小，目标类型也无法表示，则 Z 为和 X+Y 同符号的零值。

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；而且在除法运算符使用的时候，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相除之值的最大整数。

取余运算符

- 取余运算符用来求除法的余数，在 C# 语言中，求模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 2 种类型，第一种是简单赋值运算符，就是我们前面在程序里面已经使用多次的“ =”号；第二种是复合赋值运算符，包含 5 类，具体的如下表所示

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 value 的值
+=	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

运算符的优先级

- 其实在小学数学的学习中，我们就涉及了运算符优先级的内容，那时叫做四则运算，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，也完全符合这道口诀。
- 但是需要我们注意的是：程序中的运算符要比数学中多一些，比如说： ++ 、 % 等运算符号。另外数学运算符的优先级要排在赋值运算符之前，举几个例子可能说明的更清楚些：

  a=(b+c)*d ;

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

  a%=(++b-c*d)%e;

  这个代码我们可以改写为： a= a% ((++b-c*d)%e); 这样我们就可以按照基本的运算法则来求了。

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

- 在 C# 语言中，数据类型可以分为两大部分：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 2 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;
```

- C# 中，值类型可以分为以下几种：
  - 简单类型（ Simple Types ）
  - 结构类型（ struct Types ）
  - 枚举类型（ Enumeration Types ）

简单类型

- 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。
- 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跟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导致的。 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。

- 布尔类型

在 C# 语言中，布尔类型只有 2 类，即“ True” 与“ False” 。在编写应用程序的逻辑流程时（通俗地讲，就是在判断某条件是符合逻辑的还是不符合逻辑的），布尔型的变量有着非常重要的分支作用。

简单类型

- 字符类型

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

- 实数类型

  在 C# 中，实数类型分为单精度（ Float ）、双精度（ Double ）和十进制类型（ Decimal ），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 类型主要用于方便我们在金融和货币方面的计算。

结构类型

- 一个结构类型 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。

结构类型

- 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。

枚举类型

- 枚举（ Enum ）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，我们就可以使用枚举。例如，我们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;
```

枚举类型与结构类型的比较

- 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。

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
  - 委托
  - 接口

  这 4 种引用类型我们在以后的章节里都会详细地介绍。

小结

- 本章主要介绍了创建有效 C# 应用程序的许多基础知识，分析了在创建控制台应用程序工程时， Visual Studio.NET 生成的最基础的控制台应用程序代码。
- 本章的主要内容是变量以及表达式，主要介绍了变量的声明、使用方法以及注意事项；还特别介绍了运算符的优先级。

  最后，我们介绍了值类型以及引用类型，这两种类型的区分以及关联将贯穿全书，似很重要的内容。

谢谢