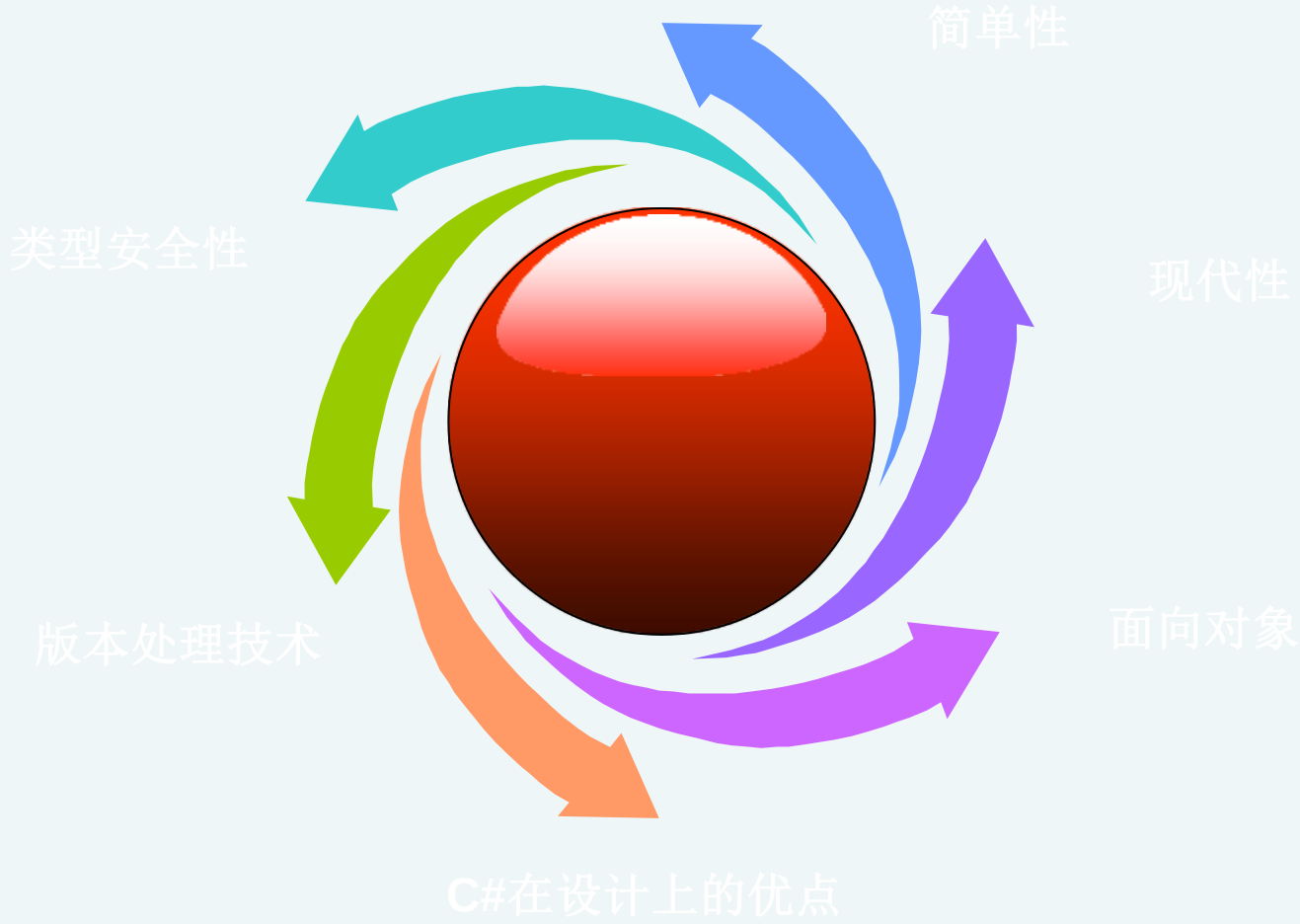
1.1 C#的优势

作为编程语言，C#是现代的、简单的、完全面向对象的，而且是类型安全的。重要的是，C#是一种现代编程语言。在类、名字空间、方法重载和异常处理等方面，C#去掉了C++中的许多复杂性，借鉴和修改了Java的许多特性，使其更加易于使用，不易出错。

下面列举了一些C#在设计上的优点：

1.1 C#的优势



简单性

1. 没有指针是C#的一个显著特性, 用户使用一种可操控的（Managed）代码进行工作时，直接的内存存取，将是不允许的。
2. 在C#中不再需要记住那些源于不同处理器结构的数据类型

现代性

1. 用户可以使用一个新的decimal数据类型进行货币计算。
2.C#通过代码访问安全机制来保证安全性，根据代码的身份来源，可以分为不同的安全级别，不同级别的代码在被调用时会受到不同的限制。

面向对象

1.C#支持面向对象的所有关键概念：封装、继承和多态性。

2. C#的继承机制只允许一个基类。如果需要多重继承，用户可以使用接口。

类型安全性

1.C#实施了最严格的类型安全机制来保护它自身及其垃圾收集器。
2. 边界检查。
3. 算术运算溢出检查。
4. C#中传递的引用参数是类型安全的。

版本处理技术

C#尽其所能支持DLL版本处理功能,虽然C#自己并不能保证提供正确的版本处理结果，但它为程序员提供了这种版本处理的可能性。有了这个适当的支持，开发者可以确保当他开发的类库升级时，会与已有的客户应用保持二进制级别上的兼容性。

1.2 第一个C#程序

首先看控制台应用程序的版本。

【例1.1】 在控制台窗口中输出“Hello World!”字样。

在Visual C#.NET开发环境中新建一个控制台应用程序项目，并在源代码文件中输入如下语句：

```
using System;
class HelloWorld
{
        public static void Main()
        {
                Console.WriteLine("Hello World!");
        }
}
```

程序运行结果下图所示。

Hello World!

将此内容保存到文件EX1_1.cs中，然后选择菜单“调试”→“启动”或直接按F5键运行此程序。

也可以不使用Visual C#.NET环境，直接用文本编辑工具输入以上代码，并且保存到Helloworld.cs文件中，再通过开始菜单的“程序”→“Microsoft Visual Studio 2005”→“Visual Studio Tools”→“Visual Studio 2005命令提示”打开命令行窗口，在命令窗口中输入：csc /target:exe EX1_1.cs

程序运行结果如图1.1所示。

Hello World!

下面再看Windows应用程序的版本。

【例1.2】 弹出一个消息对话框显示“Hello World!”字样。用“Windows应用程序”模板建立项目，或者使用文本编辑工具输入源代码如下：

```
using System;
using System.Windows.Forms;
class HelloWorld
{
        public static void Main()
   {
        MessageBox.Show("Hello World","Message from C#");
   }
}
```

这次需要增加对System.Windows.Forms命名空间的引用，选择菜单“项目” →“添加引用...”打开“添加引用”对话框，在列表中找到并选中“System.Windows.Forms.dll”，然后单击“选择”按钮，最后单击“确定”按钮完成添加。将文件另存为EX1_2.cs，在开发环境中按F5键编译运行程序，如果用命令行方式编译请参照控制台版，编译命令如下：

```
csc /target:winexe EX1_2.cs
```

程序的运行结果如图1.2所示。

通过上述两段代码来认识C#：

1. 代码最前面是以using关键字开始的命名空间导入语句，然后是使用class关键字对类HelloWorld的定义。

2. 命名空间是为了防止相同名字的不同标识符发生冲突而设计的隔离机制。

3. 在.NET框架类库中提供的不同组件都被包含在一定的命名空间中，所以要使用这些组件也必须通过using关键字开放相应的命名空间，使得相应的标识符对编译器可见，如果没有使用using关键字，那么相应的标识符就应包含完整的空间路径。

4. 由于C#是一种完全的面向对象的语言，所以不会有独立于类的代码出现，应用程序的入口也必须是类的方法，C#规定命名为Main的方法作为程序的入口

5. C#是一种大小写敏感的语言 ！！

第2章 C#编程基础

C#的基本数据类型、变量、常量、表达式、程序流程控制语句及数组等概念是C#程序设计的基础，掌握这些基本知识是编写正确程序的前提。

2.1 基本类型

C#的基本数据类型、变量、常量、表达式、程序流程控制语句及数组等概念是C#程序设计的基础，掌握这些基本知识是编写正确程序的前提。

第2章 C#编程基础

C#的基本数据类型、变量、常量、表达式、程序流程控制语句及数组等概念是C#程序设计的基础，掌握这些基本知识是编写正确程序的前提。

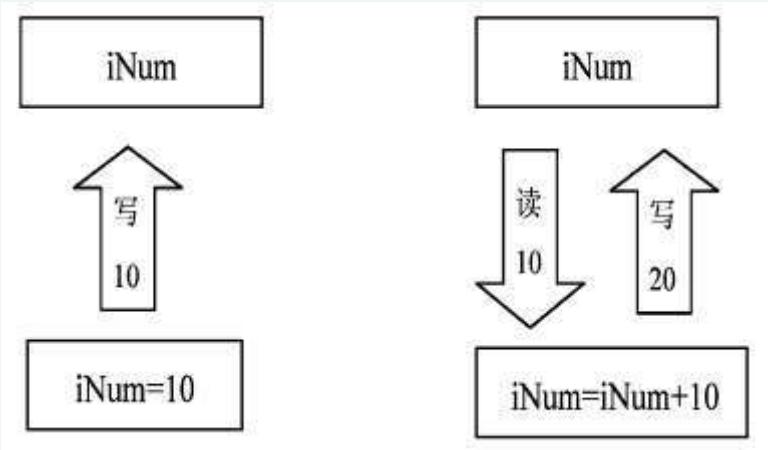
2.2.1 值类型

所谓值类型就是一个包含实际数据的量。当定义一个值类型的变量时，C#会根据它所声明的类型，以堆栈方式分配一块大小相适应的存储区域给这个变量，随后对这个变量的读/写操作就直接在这块内存区域进行。 .

例如：
int iNum=10; // 分配一个32位内存区域给变量iNum，并将10放入该内存区域

iNum=iNum+10; // 从变量iNum中取出值，加上10，再将计算结果赋给iNum

图2.1 值类型操作示意图



C#中的值类型包括：简单类型、枚举类型和结构类型。

表2.1 C#简单类型

C#关键字	.NET CTS类型名	说　明	范围和精度
bool	System.Boolean	逻辑值（真或假）	true, false
sbyte	System.SByte	8位有符号整数类型	–128~127
byte	System.Byte	8位无符号整数类型	0~255
short	System.Int16	16位有符号整数类型	–32768~32767
ushort	System.UInt16	16位无符号整数类型	0~65535
int	System.Int32	32位有符号整数类型	– ~ 7

续表

C#关键字	.NET CTS类型名	说　明	范围和精度
uint	System.Uint32	32位无符号整数类型	0~
long	System.Int64	64位有符号整数类型	– 8~ 7
ulong	System.UInt64	64位无符号整数类型	0~ 15
char	System.Char	16位字符类型	所有的Unicode编码字符
float	System.Single	32位单精度浮点类型	（大约7个有效十进制数位）
double	System.Double	64位双精度浮点类型	（大约15~16个有效十进制数位）
decimal	System.Decimal	128位高精度十进制数类型	（大约28~29个有效十进制数位）

表中“C#关键字”是指在C#中声明变量时可使用的类型说明符。

2.1.2 引用类型

引用类型包括class（类）、interface（接口）、数组、delegate（委托）、object和string。其中object和string是两个比较特殊的类型。object是C#中所有类型（包括所有的值类型和引用类型）的根类。string类型是一个从object类直接继承的密封类型（不能再被继承），其实例表示Unicode字符串。

一个引用类型的变量不存储它们所代表的实际数据，而是存储实际数据的引用。引用类型分两步创建：首先在堆栈上创建一个引用变量，然后在堆上创建对象本身，再把这个内存的句柄（也是内存的首地址）赋给引用变量。

例如：
string s1, s2;
s1="ABCD"; s2 = s1;
其中，s1、s2是指向字符串的引用变量，s1的值是字符串"ABCD"存放在内存的地址，这就是对字符串的引用，两个引用型变量之间的赋值，使得s2、s1都是对"ABCD"的引用，如图2.2所示。

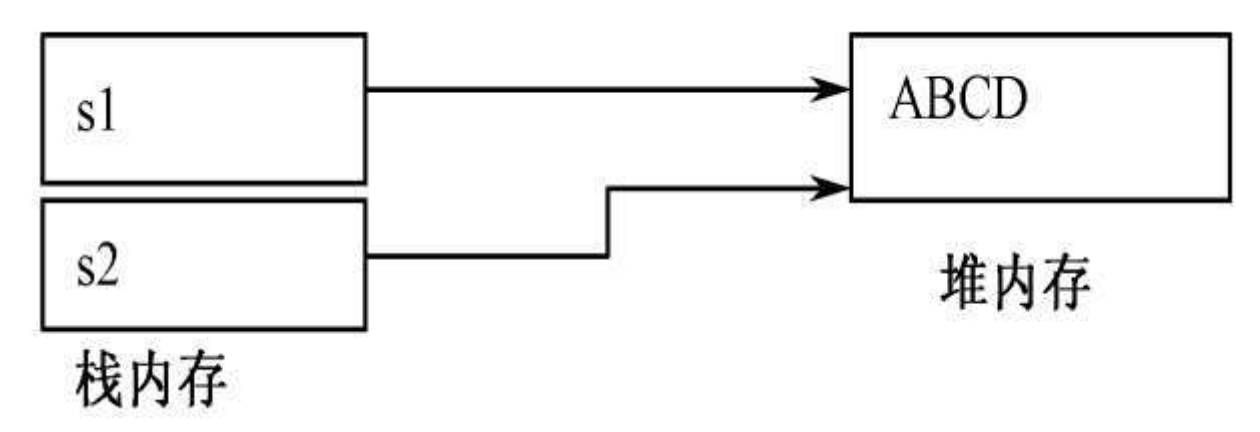


图2.2　　引用类型赋值示意

引用类型的值是引用类型实例的引用，特殊值null适用于所有引用类型，它表明没有任何引用的对象。当然也可能有若干引用类型的变量同时引用一个对象的实例，对任何一个引用变量作出修改都会导致该对象的值的修改。

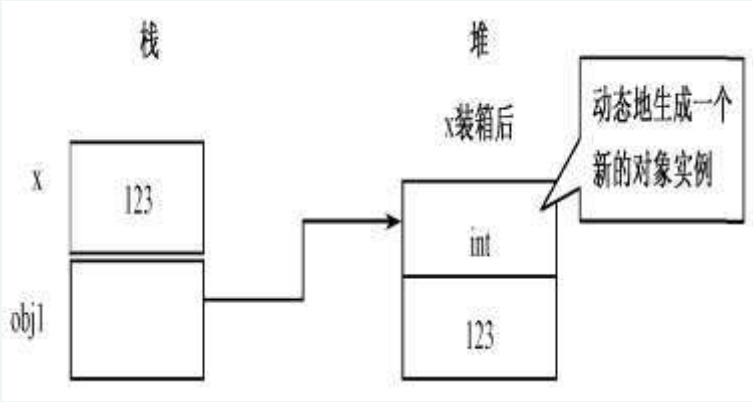
注意：堆栈是按先进后出（FILO）的原则存储数据项的一种数据结构；堆（heap）则是用于动态内存分配的一块内存区域，可以按任意顺序和大小进行分配和释放。C#中，值类型就分配在堆栈中，堆栈内存区域内保存着值类型的值，内存区域可以通过变量名来存取。引用类型分配在堆中，对象分配在堆时，返回的是地址，这个地址被赋值给引用。

2.1.3 值类型与引用类型的关系

可以把值类型与引用类型的值赋给object类型变量，C#用“装箱”和“拆箱”来实现值类型与引用类型之间的转换。“装箱”就是将值类型包装成引用类型的处理过程。当一个值类型被要求转换成一个object对象时，“装箱”操作自动进行，它首先创建一个对象实例，然后把值类型的值复制到这个对象实例，最后由object对象引用这个对象实例。

例如:

```
using System;
class Demo
{
    public static void Main ( )
     {
       int x = 123;
       object obj1=x;   // 装箱操作
       x = x+100;       // 改变x的值，此时obj1的值并不会随之改变
       Console.WriteLine (" x= {0}" , x );
                                // x=223
      Console.WriteLine (" obj1= {0}" , obj1 );
                                // obj1=123
          }
}
```



该实例的“装箱”操作说明如图2.3所示。

“拆箱”操作与“装箱”相反，它是将一个object类型转换成值类型。首先检查由object引用的对象实例值类型的包装值，然后把实例中的值复制到值类型的变量中。

注意：当一个“装箱”操作把值类型转换为一个引用类型时，不需要显式地强制类型转换；而拆箱操作把引用类型转换到值类型时，由于它可以强制转换到任何可以相容的值类型，所以必须显式地强制类型转换。

```
using System;
class Demo
{
    public static void Main ( )
    {
        int x = 123, y;
        object obj1=x;          // 装箱操作
        x = x+100;              // 改变x的值，此时obj1的值并不会随之改变
        y = (int) obj1;         // 拆箱操作，必须进行强制类型转换
      Console.WriteLine (" x= {0}" , x );                    // x=223
     Console.WriteLine (" obj1= {0}" , obj1 );
                                                            // obj1=123
   }
}
```

2.2 变量与常量

C#的基本数据类型、变量、常量、表达式、程序流程控制语句及数组等概念是C#程序设计的基础，掌握这些基本知识是编写正确程序的前提。

2.2.1 常量

常量，顾名思义就是在程序运行期间其值不会改变的量，它通常可以分为字面常量和符号常量。常量及其使用非常直观，以能读懂的固定格式表示固定的数值，每一种值类型都有自己的常量表示形式。

1. 整数常量

对于一个整数值，默认的类型就是能保存它的最小整数类型，其类型可以分为int、uint、long或ulong。如果默认类型不是想要的类型，可以在常量后面加后缀（U或L）来明确指定其类型。
在常量后面加L或l（不区分大小写）表示长整型。例如：

```
32          // 这是一个int类型
32L                 // 这是一个long类型
```

在常量后面加U或u（不区分大小写）表示无符号整数。例如：

```
128U                // 这是一个uint 类型
128UL               // 这是一个ulong类型
```

整型常量既可以采用十进制也可以采用十六进制，不加特别说明默认为十进制，在数值前面加0x（或0X）则表示十六进制数，十六进制基数用0~9、A~F（或a~f），例如：

```
0x20                // 十六进制数20，相当于十进制数32
0x1F                // 十六进制数1F，相当于十进制数31
```

2. 浮点常量

一般带小数点的数或用科学计数法表示的数都被认为是浮点数，它的数据类型默认为double类型，但也可以加后缀符表明三种不同的浮点格式数。
在数字后面加F（f）表示是float类型。
在数字后面加D（d）表示是double类型。在数字后面加M（m）表示是decimal类型。例如：

3.14，3.14e2，0.168E–2　　// 这些都是double类型常量，其中3.14e2相当于，
// 0.618E–2 相当于3.14F，0.168f　　// 这些都是float类型常量
3.14D，0.168d　　// 这些都是double类型常量
3.14M，0.168m　　// 这些都是decimal类型常量

3．字符常量

字符常量，简单地说，就是用单引号括起来的单个字符，如'A'，它占16位，以无符号整型数的形式存储这个字符所对应的Unicode代码。这对于大多数图形字符是可行的，但对一些非图形的控制字符（如回车符）则行不通，所以字符常量的表达有若干种形式。

单引号括起的一个字符，如'A'。

十六进制的换码系列，以“\x”或“\X”开始，后面跟4位十六进制数，如' \X0041' 。

Unicode码表示形式，以“\U”或“\u”开始，后面跟4位十六进制数，如' \U0041'。

显式转换整数字符代码，如（char）65。

字符转义系列，如表2.2所示。

表2.2 字符转义符

转义字符	含　义	Unicode码	转义字符	含　义	Unicode码
\'	单引号	\u0027	\b	退格符	\u0008
\"	双引号	\u0022	\f	走纸换页符	\u000C
\\	反斜线字符	\u005C	\n	换行符	\u000A
\0	空字符	\u0000	\r	回车符	\u000D
\a	警铃符	\u0007	\t	水平制表符	\u0009
\v	垂直制表符	\u000B			

4. 字符串常量

字符串常量是用双引号括起的零个或多个字符序列。C#支持两种形式的字符串常量，一种是常规字符串，另一种是逐字字符串。

1　　常规字符串。双引号括起的一串字符，可以包括转义字符。
例如：
"Hello, World\n"
"C:\\windows\\Microsoft"　　// 表示字符串C:\windows\Microsoft
2逐字字符串。在常规的字符串前面加一个@，就形成了逐字字符串，它的意思是字符串中的每个字符均表示本意，不使用转义字符。如果在字符串中需用到双引号，则可连写两个双引号来表示一个双引号。
例如：
@"C:\windows\Microsoft"　　// 与 "C:\\windows\\Microsoft" 含义相同
@"He said""Hello"" to me"　　// 与"He said\"Hello\" to me" 含义相同

5．布尔常量

它只有两个值：true和false。

6．符号常量

在声明语句中，可以声明一个标识符常量，但必须在定义标识符时就进行初始化，并且定义之后就不能再改变该常量的值。
具体的格式为：
const 类型 标识符=初值
例如：
const double PI=3.14159

2.2.2 变量

变量是在程序运行过程中其值可以改变的量，它是一个已命名的存储单元，通常用来记录运算的中间结果或保存数据。在C#中，每个变量都具有一个类型，它确定哪些值可以存储在该变量中。创建一个变量就是创建这个类型的实例，变量的特性由类型来决定。C#中的变量必须先声明后使用。声明变量包括变量的名称、数据类型，必要时指定变量的初始值。

声明变量的形式：

类型 标识符表;

或

类型 标识符[=初值] [, …];

标识符必须以字母或者_（下划线）开头，后面跟字母、数字和下划线的组合。例如，name、_Int、Name、x_1等都是合法的标识符，但C#是大小写敏感的语言，name、Name分别代表不同的标识符，在定义和使用时要特别注意。另外变量名不能与C#中的关键字相同，除非标识符是以@作为前缀的。

例如：

```
int     x ;                          // 合法
float   y1=0.0, y2 =1.0, y3 ;                  // 合法，变量说明的同时可以设置初始数值
string  char                         // 不合法，因为char是关键字
string  @char                        // 合法
```

C#允许在任何模块声明变量，模块开始于“{”，结束于“}”。每次进入声明变量所在的模块时，则创建变量并分配存储空间，离开这个模块时，则销毁这个变量并收回分配的存储空间。实际上，变量只在这个模块内有效，所以称为局部变量，这个模块区域就是变量的作用域。

2.3 运算符与表达式

表达式是由操作数和运算符构成的。操作数可以是常量、变量、属性等；运算符指示对操作数进行什么样的运算。因此，也可以说表达式就是利用运算符来执行某些计算并且产生计算结果的语句。C#提供大量的运算符，按需要操作数的数目来分，有一元运算符（如++）、二元运算符（如+，*）、三元运算符（如?：）。按运算功能来分，基本的运算符可以分为以下几类：

1 算术运算符
2 关系运算符
3 逻辑运算符
4 位运算符
5 赋值运算符
6 条件运算符
7 其他（分量运算符'．'，下标运算符'[]'等）

2.3.1 算术运算符

算术运算符作用的操作数类型可以是整型也可以是浮点型，运算符如表2.3所示

运算 符	含 义	示例（假设x，y是某一数值类型的变量）	运 算 符	含 义	示例（假设x，y是某一数值类型的变量）
+	加	x + y ； x+3 ;	%	取模	x % y ； 11%3 ; 11.0 % 3;
–	减	x – y ； y–1 ;	++	递增	++x ； x++ ;
*	乘	x * y ； 3*4 ;	– –	递减	– – x ； x– – ;
/	除	x / y ； 5/2; 5.0/2.0 ;			

其中：

- “+–*/ ”运算与一般代数意义及其他语言相同，但需要注意：当“/”作用的两个操作数都是整型数据类型时，其计算结果也是整型。

例如：

```
4/2       // 结果等于2
5/2       // 结果等于2
5/2.0     // 结果等于2.5
```

- “%”取模运算，即获得整数除法运算的余数，所以也称取余。

例如：

```
11%3    // 结果等于2
12%3    // 结果等于0
11.0%3   /        / 结果等于2，这与C/C++不同，它也可作用于浮点类型的
```

操作数

- “++”和“--”递增和递减运算符是一元运算符，它作用的操作数必须是变量，不能是常量或表达式。它既可出现在操作数之前（前缀运算），也可出现在操作数之后（后缀运算），前缀和后缀有共同之处，也有很大区别。

例如：

++x　先将x加一个单位，然后再将计算结果作为表达式的值。

x++　先将x的值作为表达式的值，然后再将x加一个单位。

不管是前缀还是后缀，它们操作的结果对操作数而言，都是一样的，操作数都加了一个单位，但它们出现在表达式运算中是有区别的。

例如：

```
int  x , y ;
x=5 ;  y=++x ;      //  x和y的值都等于6
x=5 ;  y=x++ ;      //  x的值是6，y的值是5
```

操作数

- “++”和“--”递增和递减运算符是一元运算符，它作用的操作数必须是变量，不能是常量或表达式。它既可出现在操作数之前（前缀运算），也可出现在操作数之后（后缀运算），前缀和后缀有共同之处，也有很大区别。

例如：

++x 先将x加一个单位，然后再将计算结果作为表达式的值。

x++ 先将x的值作为表达式的值，然后再将x加一个单位。

不管是前缀还是后缀，它们操作的结果对操作数而言，都是一样的，操作数都加了一个单位，但它们出现在表达式运算中是有区别的。

例如：

```
int  x , y ;
x=5 ;  y=++x ;    //  x和y的值都等于6
x=5 ;  y=x++ ;    //  x的值是6，y的值是5
```

2.3.2 关系运算符

关系运算符用来比较两个操作数的值，运算结果为布尔类型的值（true或false），如表2.4所示。

运 算 符	操　作	结果（假设x，y是某相应类型的操作数）
>	x>y	如果x大于y，则为true，否则为false
>=	x>=y	如果x大于等于y，则为true，否则为false
<	x<y	如果x小于y，则为true，否则为false
<=	x<=y	如果x小于等于y，则为true，否则为false
= =	x= =y	如果x等于y，则为true，否则为false
!=	x!=y	如果x不等于y，则为true，否则为false

在C#中，简单类型和引用类型都可以通过==或!=来比较它们的数据内容是否相等。对简单类型，比较的是它们的数据值；而对引用类型来说，由于它的内容是对象实例的引用，所以若相等，则说明这两个引用指向同一个对象实例，如果要测试两个引用对象所代表的内容是否相等，则通常会使用对象本身所提供的方法，如Equals( )。
如果操作数是string类型的，则在下列两种情况下被视为两个string值相等。
1　　两个值均为null。
2　　两个值都是对字符串实例的非空引用，这两个字符串不仅长度相同，并且每一个对应的字符位置上的字符也相同。关系比较运算“>、>=、<、<=”是以顺序作为比较的标准，所以它要求操作数的数据类型只能是数值类型，即整型数、浮点数、字符及枚举等类型。
布尔类型的值只能比较是否相等，不能比较大小。因为true和false值没有大小之分，例如，表达式true > flase在C#中是没有意义的。

在C#中，简单类型和引用类型都可以通过==或!=来比较它们的数据内容是否相等。对简单类型，比较的是它们的数据值；而对引用类型来说，由于它的内容是对象实例的引用，所以若相等，则说明这两个引用指向同一个对象实例，如果要测试两个引用对象所代表的内容是否相等，则通常会使用对象本身所提供的方法，如Equals( )。

如果操作数是string类型的，则在下列两种情况下被视为两个string值相等。

1　　两个值均为null。

2两个值都是对字符串实例的非空引用，这两个字符串不仅长度相同，并且每一个对应的字符位置上的字符也相同。关系比较运算“>、>=、<、<=”是以顺序作为比较的标准，所以它要求操作数的数据类型只能是数值类型，即整型数、浮点数、字符及枚举等类型。

布尔类型的值只能比较是否相等，不能比较大小。因为true和false值没有大小之分，例如，表达式true > flase在C#中是没有意义的。

2.3.3　逻辑运算符

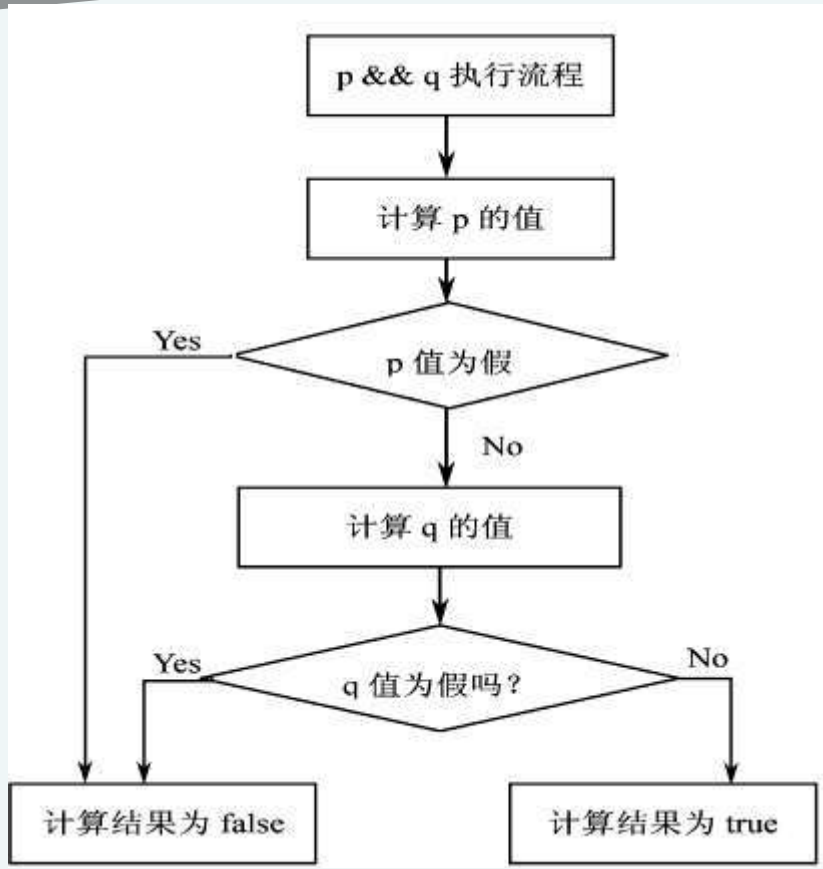
逻辑运算符是用来对两个布尔类型的操作数进行逻辑运算的，运算的结果也是布尔类型，如表2.5所示。

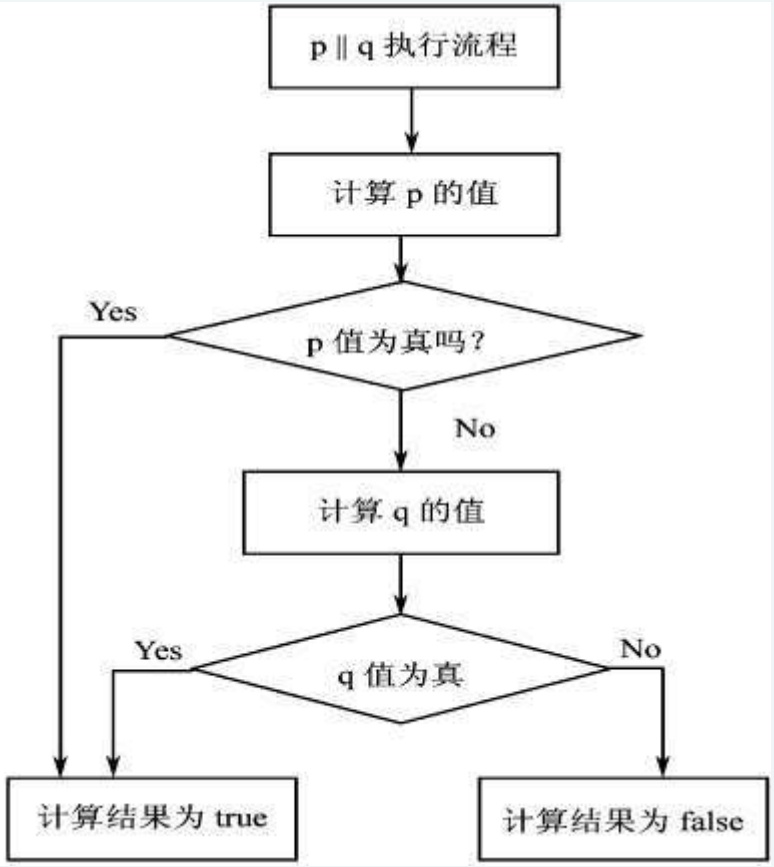
运 算 符	含　义	运 算 符	含　义
&	逻辑与	&&	短路与
|	逻辑或	||	短路或
^	逻辑异或	!	逻辑非

假设p、q是两个布尔类型的操作数，表2.6给出了逻辑运算的真值表。

p	q	p & q	p | q	p ^ q	! p
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	flase	flase	false	false	true

运算符“&&”和“||”的操作结果与“&”和“|”一样，但它们的短路特征使代码的效率更高。所谓短路就是在逻辑运算的过程中，如果计算第一个操作数时，就能得知运算结果就不会再计算第二个操作数，如图2.4所示。





2.3.4 位运算符

位运算符主要分为逻辑运算和移位运算，它的运算操作直接作用于操作数的每一位，所以操作数的类型必须是整数类型，不能是布尔类型、float或double等类型。

如表2.7所示。借助这些位运算符可以完成对整型数的某一位测试、设置，以及对一个数的位移动等操作，这对许多系统级程序设计非常重要。

运 算 符	含 义
&	按位与
|	按位或
^	按位异或
~	按位取反
>>	右移
<<	左移

按位异或运算有一个特别的属性，假设有两个整型数x和 y，则表达式 (x ^ y) ^ y值还原为x，利用这个属性可以创建简单的加密程序。例如

```
using System;
class Encode
{
        public static void Main( )
                {
                char ch1 = 'O' , ch2 = 'K' ;
                int key = 0x1f ;
                Console.WriteLine ("明文：" + ch1 + ch2 ) ;
                ch1 = (char) (ch1 ^ key ) ;
                ch2 = (char) (ch2 ^ key ) ;
                Console.WriteLine ("密文：" + ch1 + ch2 ) ;
                ch1 = (char) (ch1 ^ key ) ;
                ch2 = (char) (ch2 ^ key ) ;
                Console.WriteLine ("解码：" + ch1 + ch2 ) ;
        }
```

移位运算符有两个：一个左移（<<），一个右移（>>）。
语法形式：
value << num_bits
value >> num_bits
左操作数value是要被移位的值，右操作数num_bits是要移位的位数。
（1）左移。将给定的value向左移动num_bits位，左边移出的位丢掉，右边空出的位填0。
例如：0x1A << 2。左移过程如图2.5所示。

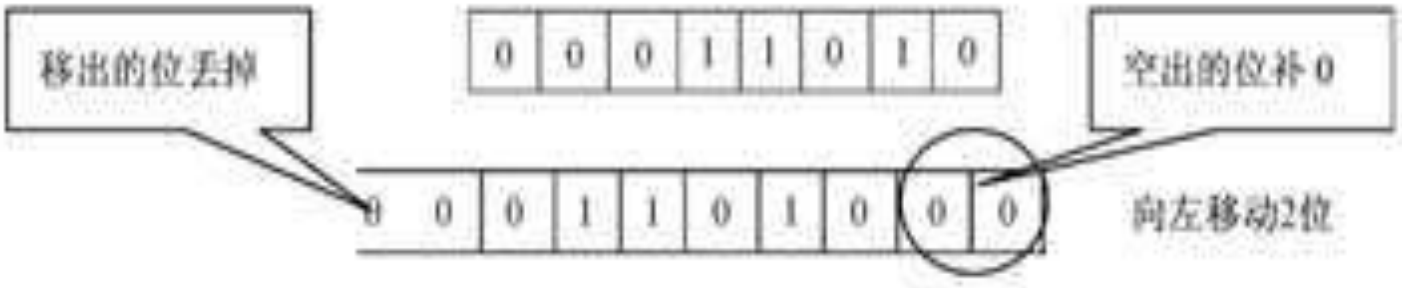


图 2.5 左移示例 1

例如：0x8A >> 2。右移过程如图2.8所示。

图 2.8　右移示例 2

例如：0x8AU >> 2。右移过程如图2.9所示。
右移运算符的作用相当于将value的值整除以 2^{num_bits}

图 2.9　右移示例 3

2.3.5 赋值运算符

赋值运算符有两种形式，一种是简单赋值运算符，另一种是复合赋值运算符。

1. 简单赋值运算符

语法形式：

var = exp

2. 复合赋值运算符

在进行如x = x +3 运算时，C#提供一种简化方式x +=3，这就是复合赋值运算。

语法形式：

var op= exp　　// op 表示某一运算符

等价的意义是： var = var op exp

除了关系运算符，一般二元运算符都可以和赋值运算符在一起构成复合赋值运算，如表2.8所示。

运 算 符	用 法 示 例	等价表达式	运 算 符	用 法 示 例	等价表达式
+=	x += y	x = x + y	&=	x &= y	x = x & y
– =	x– = y	x = x–y	|=	x |= y	x = x | y
*=	x *= y	x = x * y	^=	x ^= y	x = x ^ y
/=	x /= y	x = x / y	>>=	x >>= y	x = x >> y
%=	x %= y	x = x % y	<<=	x <<= y	x = x << y

2.3.6 条件运算符

语法形式：
exp1 ? exp2 : exp3

其中，表达式exp1的运算结果必须是一个布尔类型值，表达式exp2和exp3可以是任意数据类型，但它们返回的数据类型必须一致。
首先计算exp1的值，如果其值为true，则计算exp2值，这个值就是整个表达式的结果；否则，取exp3的值作为整个表达式的结果。

例如：

```
z = x > y ? x : y ;          // z 的值就是x，y中较大的一个值
z = x >=0 ? x : –x ;         // z的值就是x的绝对值
```

条件运算符“? :”是C#中唯一一个三元运算符。

2.3.7 运算符的优先级与结合性

当一个表达式含有多个运算符时，C#编译器需要知道先做哪个运算，这就是所谓的运算符的优先级，它控制各个运算符的运算顺序。

例如，表达式x+5*2是按 x+(5*2)计算的，因为“*”运算符比“+”运算符优先级高。

当操作数出现在具有相同优先级的运算符之间时，如表达式“10–6–2”按从左到右计算的结果是2，如果按从右到左计算，结果是6。当然“–”运算符是按从左到右的次序计算的，也就是左结合。

再如表达式“x=y=2”，它在执行时是按从右到左运算的，即先将数值2赋给变量y，再将y的值赋给x。所以“=”运算符是右结合的。

在表达式中，运算符的优先级和结合性控制着运算的执行顺序，也可以用圆括号“（）”显式地标明运算顺序，如表达式“(x+y)* 2”。

表2.9 运算符的优先级与结合性

类 别	运 算 符	结 合 性
初等项	. ( ) [ ] new typeof checked unchecked	从左到右
一元后缀	++ – –	从右到左
一元前缀	++ – – + – ! ~ (T) (表达式)	从右到左
乘法	* / %	从左到右
加法	+ –	从左到右
移位	<< >>	从左到右
关系和类型检测	< > <= >= is as	从左到右
相等	== !=	从左到右
逻辑与	&	从左到右
逻辑异或	^	从左到右
逻辑或	|	从左到右
条件与	&&	从左到右
条件或	||	从左到右
条件	? :	从右到左
赋值	= *= /= %= += – = <<= >>= &= ^= |=	从右到左