第 5 章

表达式与运算符

5.1 利用运算进行字符串加密

视频 6
表达式
运算符

5.1.1 案例描述

在程序编写的过程中，有些字符串需要进行加密操作，移位是进行加密的一种。通过本节的学习可完成如图 5-1 所示的一个简单的加密字符串的小案例。

图 5-1 案例展示图

5.1.2 知识引入

运算符是表示各种不同运算的符号。在 C# 中，运算符有多个级别，如表 5-1 所示。

表 5-1 运算符

类　　别	运算符	说　　明	表达式
算术运算符	+	执行加法运算（如果两个操作数是字符串，则该运算符用作字符串连接，将一个字符串添加到另一个字符串的末尾）	操作数 1+操作数 2
	−	执行减法运算	操作数 1−操作数 2
	*	执行乘法运算	操作数 1 * 操作数 2
	/	执行除法运算	操作数 1/操作数 2
	%	获得进行除法运算后的余数	操作数 1%操作数 2
	++	将操作数加 1	操作数++或++操作数
	−−	将操作数减 1	操作数−−或−−操作数
	~	将一个数按位取反	~操作数

续表

类　　别	运算符	说　　明	表达式
比较运算符	>	检查一个数是否大于另一个数	操作数 1>操作数 2
	<	检查一个数是否小于另一个数	操作数 1<操作数 2
	>=	检查一个数是否大于或等于另一个数	操作数 1>=操作数 2
	<=	检查一个数是否小于或等于另一个数	操作数 1<=操作数 2
	==	检查两个值是否相等	操作数 1==操作数 2
	!=	检查两个值是否不相等	操作数 1!=操作数 2
条件运算符	?:	检查给出的第一个表达式是否为真。如果为真,则计算操作数 1；否则计算操作数 2。这是唯一带有 3 个操作数的运算符	表达式? 操作数 1:操作数 2
赋值运算符	=	给变量赋值	操作数 1=操作数 2
逻辑运算符	&&	对两个操作数执行逻辑“与”运算	操作数 1&&操作数 2
	||	对两个操作数执行逻辑“或”运算	操作数 1||操作数 2
	!	对操作数执行逻辑“非”运算	! 操作数
强制类型转换符	()	将操作数强制转换为给定的数据类型	(数据类型) 操作数
成员访问符	.	用于访问数据结构的成员	数据结构.成员
快捷运算符	+=		运算结果=操作数 1+操作数 2
	-=		运算结果=操作数 1-操作数 2
	*=		运算结果=操作数 1*操作数 2
	/=		运算结果=操作数 1/操作数 2
	%=		运算结果=操作数 1%操作数 2

1. 算术运算符

算术运算符用于对操作数进行算术运算。C#的算术运算符同数学中的算术运算符是很相似的。

```
int iresult,irem;
double dresult,drem;
iresult = 10/3;
irem = 10 % 3;
dresult = 10.0/3.0;
drem = 10.0 % 3.0;
Console.WriteLine("10/3 = {0}\t10 % 3 = {1}",iresult,irem);
Console.WriteLine("10.0/3.0 = {0}\t10.0 % 3.0 = {1}",dresult,drem);
```

程序的输出结果如图 5-2 所示。

特殊的算术运算符：

++为自增运算符。

--为自减运算符。

```
C:\Users\Administrator\source\repos\WindowsFormsApp1\Exam\bin\Debug\Exam.exe
10/3=3    10%3=1
10.0/3.0=3.33333333333333          10.0%3.0=1
```

图 5-2　程序效果图(一)

作用：使变量的值自动增加 1 或者减少 1。

例如，

```
x = x + 1;
```

可以被写成

```
++x;      //前缀格式
```

或者

```
x++;     //后缀格式
```

当一个自增或自减运算符在它的操作数前面时，C#将在取得操作数的值前执行自增或自减操作。

如果运算符在操作数的后面，C#将先取得操作数的值，然后进行自增或自减运算。

```
x = 8;
y = ++x;
```

在这种情况下，x 和 y 被赋值为 9。

但是，如果代码如下：

```
x = 8;
y = x++;
```

那么 y 被赋值为 8，x 被赋值为 9。

```
int x = 5;
int y = x-- ;
Console.WriteLine("y = {0}",y);
y = --x;
Console.WriteLine("y = {0}",y);
```

运行结果如图 5-3 所示。

```
C:\Users\Administrator\source\repos\WindowsFormsApp1\Exam\bin\Debug\Exam.exe
y=5
y=3
```

图 5-3　程序效果图(二)

```
int Val1 = 2;
int Val2 = 3;
Console.WriteLine("Val1 * Val2 = {0}",Val1 * Val2);
Console.WriteLine("Val1/Val2 = {0}",Val1/Val2);
Console.WriteLine("Val1 % Val2 = {0}",Val1 % Val2);
Console.WriteLine(++Val1);
Console.WriteLine( -- Val2);
Console.WriteLine(Val1++);
Console.WriteLine(Val2 -- );
```

运行结果如图 5-4 所示。

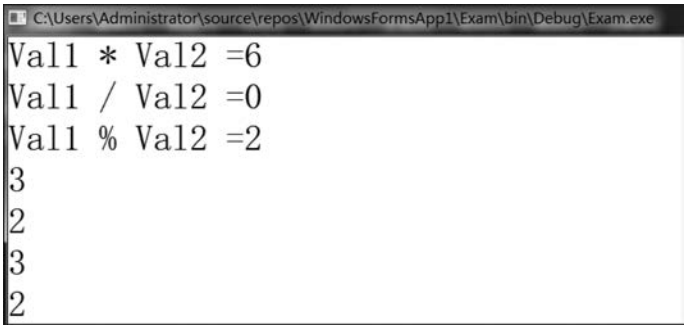

图 5-4　程序效果图(三)

注意：＋＋、－－只能用变量，不能用于常量或表达式，例如，5＋＋或－－(x＋y)都是错误的。

2. 赋值运算符

赋值运算符用于将一个数据赋予一个变量，赋值操作符的左操作数必须是一个变量，赋值结果是将一个新的数值存放在变量所指示的内存空间中。

```
int x = 8;
x = x + x;
```

可以把表达式的值通过复合赋值运算符赋予变量，这时复合赋值运算符右边的表达式是作为一个整体参加运算的。

```
int a = 8,b = 3;
a % = b * 2 - 5;      /* 相当于 a % = (b * 2 - 5),它与 a = a % (b * 2 - 5)是等价的. */
```

对变量可以进行连续赋值。例如，

```
int z = 3;
x = y = z;      //等价于 x = (y = z).
```

3. 关系运算符

关系运算符用于比较两个值的大小，关系运算的结果不是 True 就是 False。

```
bool a = 'a'<'b';            //a 的值为 True
a = 3 + 6 > 5 - 2            //a 的值为 True
using System;
class Count
{
static void Main(){
int a = 50;
int x = 30;
int y = 60;
int b;
b = x + y;
bool j;
j = a == b - 40;
Console.WriteLine("a = b is{0}",j);
}
}
```

该程序运行后，输出结果如图 5-5 所示。

图 5-5　程序效果图(四)

4. 逻辑运算符

逻辑运算符用于表示两个布尔值之间的逻辑关系，逻辑运算结果是布尔类型。

逻辑非(!)：运算的结果是原先的运算结果的逆。

逻辑与(&&)：只有两个运算对象都为 True，结果才为 True；只要其中有一个是 False，结果就为 False。

逻辑或(||)：只要两个运算对象中有一个是 True，结果就为 True，只有两个条件均为 False，结果才为 False。

当需要多个判定条件时，可以很方便地使用逻辑运算符将关系表达式连接起来。例如，

```
x > y&&x > 0
```

如果表达式中同时存在着多个逻辑运算符，那么逻辑非的优先级最高，逻辑与的优先级高于逻辑或。例如，

```
3 > 2||!(5 - 3 < 6)&&'a'<'b'
```

5. 位运算符

1) ～运算符

把二进制数的 0 转换为 1，1 转换为 0。例如，6 的二进制表示为 00000110，～6 的结果

为 11111001。

2）& 运算符

0&0=0,0&1=0,1&0=0,1&1=1。

例如，

```
7 的二进制表示:00000111
11 的二进制表示:00001011
————————————————————
"&"运算的结果是:00000011
```

即

```
7&11 = 3
```

3）|运算符

0|0=0,0|1=1,1|0=1,1|1=1。

例如，

```
7 的二进制表示:00000111
11 的二进制表示:00001011
————————————————————
"|"运算的结果是:00001111
```

即

```
7|11 = 15
```

4）^运算符

0^0=0,0^1=1,1^0=1,1^1=0。

例如，

```
7 的二进制表示:00000111
11 的二进制表示:00001011
————————————————————
"^"运算的结果是:00001100
```

即

```
7^11 = 12
```

5）<<运算符

二进制位全部按位左移，高位被丢弃，低位顺序补 0。例如，

```
7 的二进制表示:00000111
7 << 1 结果是 00001110(十进制是 14)
```

6）>>运算符

二进制位全部按位右移。例如，

```
7 的二进制表示:00000111
7 >> 1 结果是 00000011(十进制是 3)
```

6. 条件运算符

格式：

```
操作数 1?操作数 2:操作数 3
```

含义：进行条件运算时，首先判断问号前面的布尔值是 True 还是 False，如果是 True，则值等于操作数 2 的值；如果为 False，则值等于操作数 3 的值。

例如，

```
条件表达式"6 > 8?15 + a:39",由于 6 > 8 的值为 False,所以整个表达式的值是 39.
```

7. 其他运算符

1）字符串连接符（+）

将两个字符串连接在一起，形成新的字符串。例如，

```
"abc" + "efg"          //结果是 abcefg
"36812" + "3.14"       //结果是 368123.14
```

2）is 运算符

is 运算符用于检查表达式是否指定的类型，如果是，结果为 True，否则结果为 False。例如，

```
int k = 2;
bool isTest = k is int;      //isTest = True
```

3）sizeof 运算符

sizeof 运算符用于获得值类型数据在内存占用的字节数。例如，

```
int a = sizeof(double);        //a = 8
```

5.1.3　案例实现

1. 案例分析

字符串加密采用末尾字符不变的加密方式，下标为 0～length-1 的字符串中的字符统一加 3 再转换为 char 类型。

2. 代码实现

```
static void Main(string[ ] args)
            {
                //定义输入的字符串和加密后的字符串两个变量
                string strInput, strOutput;
                //提示用户
                Console.WriteLine("请输入一个需要加密的字符串:");
                //接收用户输入的字符串
                strInput = Console.ReadLine();
                Console.WriteLine("您输入的字符串为: {0}", strInput);
                //调用移位加密的方法
                strOutput = Encrypt(strInput);
                //输出结果
                Console.WriteLine("加密后的字符串为:{0}", strOutput);
                Console.ReadKey();
            }

            private static string Encrypt(string strInput)
            {
                //定义两个变量
                string strFont,                         //前面字符串
                       strEnd;                          //结尾的字符
                string strOutput;                       //结果字符串
                char[ ] charFont;                       //字符数组
                int i, len, intFont;

                len = strInput.Length;                  //获取字符串长度
                strFont = strInput.Remove(len - 1, 1);
                strEnd = strInput.Remove(0, len - 1);
                charFont = strFont.ToCharArray(); //将除最后一个字符之外的字符串转换为
                                                  //字符数组
                for (i = 0; i < strFont.Length; i++)
                {
                    intFont = (int)charFont[i] + 3;     //每一个字符转换为 int 类型并加 3

                    charFont[i] = Convert.ToChar(intFont); //将加 3 之后的 int 类型的字符
                                                           //转换为字符类型
                }
                strFont = "";
                //将加密后的除最后一个字符之外的字符串拼接成功
                for (i = 0; i < charFont.Length; i++)
                {
                    strFont += charFont[i];
                }
                strOutput = strEnd + strFont;           //拼接结果字符串
                return strOutput;
            }
```

3. 运行结果

案例的运行结果如图 5-6 所示。

```
E:\代码\Exam0909\ConsoleApp1\bin\Debug\ConsoleApp1.exe
请输入一个需要加密的字符串:
1234567jdfhhghf
您输入的字符串为: 1234567jdfhhghf
加密后的字符串为:f456789:mgikkjk
```

图 5-6　程序效果图(五)

5.2　控制台版简单计算器

5.2.1　案例描述

在程序编写的过程中,避免不了需要做大量的数据运算的工作。C#编程中数据运算也同样重要,本节将完成如图 5-7 所示的一个简单的计算器的小案例。

```
C:\Users\Administrator\source\repos\WindowsFormsApp1\Exam\bin\Debug\Exam.exe
请输入第一个数:
20
请输入第二个数:
7
请选择运算符: 1.+  2.-  3.x  4.÷  5.%
2
20-7=13
```

图 5-7　案例展示图

5.2.2　知识引入

表达式是运算符、常量和变量等组成的符号序列。

1. 算术表达式

算术表达式是用算术运算符将运算对象连接起来的符合语法规则的式子。

自增运算符和自减运算符的优先级别高于其他的算术运算符。例如,表达式 8+x++,不应被看作 8+(x++)。如果 x 的原值是 6,则表达式 8+x++的值是 14,运算结束后 x 的值是 7。

2. 赋值表达式

由赋值运算符将变量和表达式连接起来的式子称为赋值表达式。例如，

```
y = x = 8 * 8 + 3        //x = 8 * 8 + 3;y = x
```

这个赋值表达式的值是 67。由于赋值运算符的结合性是自右至左的，所以 y＝x＝8＊8＋3 和 y＝(x＝8＊8＋3)是等价的。

3. 关系表达式

用关系运算符将两个表达式连接起来的式子称为关系表达式。关系表达式的值是布尔类型，即真(True)或假(False)。例如，

```
x = 8;
y = 6;
z = x > y + 3;          //结果为 false
a = x > y&&z;           //结果为 false
```

4. 逻辑表达式

用逻辑运算符将关系表达式或者逻辑值连接起来的式子称为逻辑表达式。逻辑表达式的值只能取 true 或 false。

3 个逻辑运算符的运算顺序为“逻辑非”最高，其次是“逻辑与”，最后为“逻辑或”。例如，

```
!(3 > 6)||(5 < 8)&&(2 > = 9)||(7 > = 1)
```

5. 条件表达式

由条件运算符和表达式组成的式子称为条件表达式。例如，

```
8 > 3?5:2;
```

其结果为 5，因为 8＞3 为 true，则整个表达式的值为“:”前面表达式的值，这里是常数 5。例如，已知：

```
int i = 0;
bool result = false
result = (++i) + i == 2?true:false;
```

则变量 result 的值为多少？

注意：表达式 i＋＋和＋＋i 的区别。

在实际运算中，往往有多个运算符参与运算，这时要把握一个问题：优先级与结合性问题。在 C＃中，优先级和结合性如表 5-2 所示。

表 5-2　优先级和结合性

优　先　级	说　　明	运　算　符	结　合　性
1	括号	()	从左到右
2	自加/自减运算符	++/--	从右到左
3	乘法运算符 除法运算符 取模运算符	* / %	从左到右
4	加法运算符 减法运算符	+ -	从左到右
5	小于 小于或等于 大于 大于或等于	< <= > >=	从左到右
6	等于 不等于	= !=	从左到右
7	逻辑与	&&	从左到右
8	逻辑或	||	从左到右
9	赋值运算符和快捷运算符	=　+=　*= /=　%=　-=	从右到左

5.2.3　案例实现

1. 案例分析

(1) 简单计算器首先需要两个运算数；

(2) 在输入运算数时，需要判断运算数是否是 int 类型，判断输入的字符串是否为 int 类型，此处用转换为 int 类型数据是否发生异常来判断；

(3) 输入运算数之后要输入另一个运算符；

(4) 由运算数与运算符共同组成运算表达式计算出运算结果。

2. 代码实现

```
static void Main(string[ ] args)
            {
                //先输入第一个数
                Console.WriteLine("请输入第一个数:");
                string num1 = Console.ReadLine();
                //接着判断输入的这个数是否为整数,如果不是整数,提示重新输入第一个数
                //实参:真正进行方法中使用的参数
                int number1 = CheckNum(num1);
                //先输入第二个数
                Console.WriteLine("请输入第二个数:");
                string num2 = Console.ReadLine();
                int number2 = CheckNum(num2);
                //选择运算符
                Console.WriteLine("请选择运算符:1. + 2. - 3.x  4. ÷ 5. %");
```

```
            string fun = Console.ReadLine();
            GetResult(fun, number1, number2);
            Console.ReadLine();

        }
        /// <summary>
        /// 检测这个字符串是否能够转换为 32 位有符号整数
        /// </summary>
        /// <param name="num">要进行判断的字符串</param>
        static int CheckNum(string num)
        {
            try
            {
                int i = int.Parse(num);
                return i;
            }
            catch (Exception e)
            {
                Console.WriteLine("输入有误,请重新输入:");
                string str = Console.ReadLine();
                //递归算法
                return CheckNum(str);
            }
        }

        static void GetResult(string fun, int num1, int num2)
        {
            int res = 0;
            string yun = "";
            switch (fun)
            {
                case "1":
                    res = num1 + num2;
                    yun = "+";
                    break;
                case "2":
                    res = num1 - num2;
                    yun = "-";
                    break;
                case "3":
                    res = num1 * num2;
                    yun = "x";
                    break;
                case "4":
                    res = num1 / num2;
                    yun = "÷";
                    break;
                case "5":
                    res = num1 % num2;
```

```
                yun = "%";
                break;
            default:
                Console.WriteLine("请重新选择:");
                string str = Console.ReadLine();
                GetResult(str, num1, num2);
                return;
        }
        Console.WriteLine("{0}{3}{1}={2}", num1, num2, res, yun);
    }
```

3. 运行结果

案例的运行结果如图 5-8 所示。

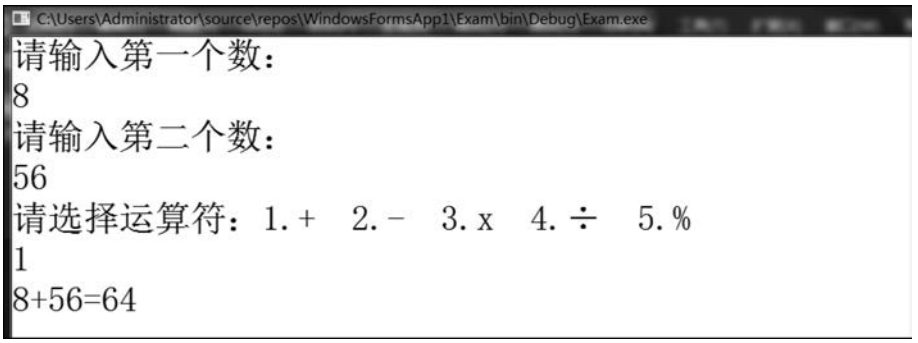

图 5-8　程序效果图

本章小结

本章的主要内容有表达式与运算符,常用的几种运算符是本章重点掌握的内容。在程序的开发过程中,运算符会被频繁地使用,可见其重要性。最常用到的运算符有算术运算符、赋值运算符、关系运算符、逻辑运算符、移位运算符以及一些特殊的运算符。同时表达式中出现不同运算符时运算符的优先级问题也属于本章的重点内容。