第 5 章　表达式和运算符

计算机的核心功能就是运算。为了让计算机完成对应的工作，编程语言将常见的运算都预先设定好，并使用相应的符号表示。这些符号就是运算符。将这些符号和我们要处理的数据结合起来，就构成了表达式。本章将详细讲解 C#中的各种运算符和表达式。

5.1　运算的最小单位——表达式

表达式是一个由操作数（operand）和运算符（operator）构成的序列。运算符指定操作数的运算规则，即指示对表达式中的操作数进行什么样的运算。本节将详细介绍表达式。

5.1.1　表达式分类

在 C#语言中，根据操作数的类型可以把表达式分为以下 9 种类别，如表 5.1 所示。

表 5.1　C#中的操作数表

操作数类型	运　　算
值	每一个值都有关联的类型
变量	每一个变量都有关联的类型。如果该变量定义了相关的类型，则称该变量为已声明类型的变量
命名空间	C#程序是利用命名空间组织起来的
类型	指定变量数据的存储方式和操作方法
方法组	表示一组方法，并可以通过实例调用方法组中的方法
属性访问	每个属性访问都有关联的类型，即该属性的类型
事件访问	每个事件访问都有关联的类型，即该事件的类型
索引器访问	每个索引器访问都有关联的类型，即该索引器的元素类型
Nothing	出现在调用一个具有 void 返回类型的方法时

下面介绍两个特殊的表达式：this 和 new。它们隐藏了操作符，都有其特殊的作用。使用 this 关键字可以限定名称相同的成员，new 关键字作为 new 运算符时可以创建对象等。

5.1.2　this 关键字

this 关键字可以用来引用类的当前实例。this 关键字存在 3 种常用用法：限定名称相同的成员、将对象本身作为参数和声明索引器。下面介绍 this 的第一种用法，后面两种用法会在相应章节介绍。

限定名称相同的成员是指在某一个类中，如果某一个成员的名称和其他对象（如参数等）的名称相同，那么可以通过 this 关键字限定该成员。

【示例 5-1】下面创建一个名称为 Program 的类，并声明了一个名称为 i 的私有成员。

```
01  using System;
02  namespace 示例 5_1
03  {
04      class Program
05      {
06          private int i;
07          public Program(int i)
08          {
09              this.i = i;          //将 i 参数的值赋值给 Program 类的私有成员 i
10          }
11          static void Main(string[] args)
12          {
13              Program p = new Program(123);
14              Console.WriteLine("i="+p.i);
15              Console.ReadLine();
16          }
17      }
18  }
```

注意：在 Program(int i)函数中，Program 类的 i 成员的名称和 i 参数的名称相同，为了区分这两个对象，则需要使用 this 关键字来限定 Program 类的 i 成员。因此，this.i 表示 Program 类的 i 成员。

分析：在“this.i = i;”语句中，this.i 表示 Program 类的私有成员 i，i（“=”的右边表达式）表示 i 参数。因此，“this.i = i;”语句将 i 参数的值赋值给 Program 类的私有成员 i。运行结果如图 5.1 所示。

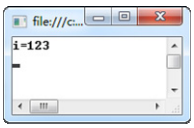

图 5.1　this 示例

5.1.3　new 关键字

new 关键字有 3 种用法，分别是作为 new 运算符、new 修饰符和 new 约束。本节只介绍 new 作为运算符的用法，其他两种用法在相应章节继续学习。当 new 关键字作为 new 运算符时，它可以用来创建对象和调用构造函数、创建匿名类型的实例、调用值类型的默认构造函数等。

【示例 5-2】下面使用 new 运算符创建一个类型为 object 的对象 o。

```
object obj = new object();          //new 运算符创建 object 类的对象 obj
```

注意：当 new 运算符创建类型的一个新实例时，但并非为该实例动态分配内存。对于值类型的实例而言，一般不需要分配该实例表示的变量之外的内存（值类型的实例

本身会占用一定的内存）。因此，在使用 new 运算符创建值类型的实例时，是不会发生动态分配内存。

5.2　运算的核心——运算符

运算符是表达式很重要的一部分，它指示对表达式中的操作数进行什么样的运算，如+、–、*、/、%等。根据运算符所需操作数的个数，可以把运算符分为一元运算符、二元运算符和三元运算符。本节将介绍有关运算符的知识。

5.2.1　算术运算符

算术运算是基本的数学运算。它包括加、减、乘、除。在 C#中，分别使用+、–、*、/表示。同时，C#还使用%表示不常用的求余运算。它们的基本语法格式如下：

```
left expression operator right expression
```

left expression 和 right expression 分别表示左操作数和右操作数，operator 表示运算符，可以为*、/、%、+和–。

【示例 5-3】下面使用+运算符计算 i 和 j 两个操作数（均为 int 类型）的和，结果保存为 result 变量。运行结果如图 5.2 所示。

```
01  using System;
02  namespace 示例 5_3
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              int i=10, j=10;
09              int result = i + j;      //使用+运算符计算变量 i 和 j 的和
10              Console.WriteLine("result="+result );
11              Console.ReadLine();
12          }
13      }
14  }
```

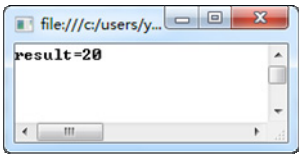

图 5.2　+运算符示例

如果左、右操作数的类型相同，则它们的运算结果的类型和操作数的类型相同。当运算结果超出了该类型表示的范围时，如果使用了 checked 运算符，则引发 System.OverflowException 异常。如果使用了 unchecked 运算符，则不引发异常，但是其结

果类型范围外的任何有效高序位都被放弃。

对于+运算符而言，如果其一个或两个操作数为 string 类型时，则+运算符执行字符串串联操作，而不是简单的算术运算。

【示例 5-4】下面使用+运算符 s1 和 s2（均为 string 类型）字符串串联起来，并保存为 s3 变量。

```
01  using System;
02  namespace 示例 5_4
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              string s1 = "abc";
09              string s2 = "def";
10              string s3 = s1 + s2;              //使用+运算符将两个字符串串起来
11              Console.WriteLine("s3="+s3);
12              Console.ReadLine();
13          }
14      }
15  }
```

分析：s3 变量的值为 abcdef 字符串。运行结果如图 5.3 所示。

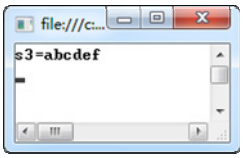

图 5.3　+运算符执行字符串串联操作示例

5.2.2　自增、自减运算符

自增、自减运算符都属于一元运算符，执行对操作数进行加 1 或者减 1 的操作。本节将详细介绍自增、自减运算符的使用方法。

1．++运算符

++运算符又称为增量运算符，其作用是对操作数进行加 1。将++运算符放在操作数的左边，称该运算符为前缀增量运算符。否则，称为后缀增量运算符。语法如下：

```
++ expression
```

或者

```
expression ++
```

expression 表示操作数。“++ expression”表达式首先计算 expression 的值，然后将其值增 1，并作为该表达式的结果。“expression ++”表达式首先计算 expression 的值，并作为该表达式的结果，然后将 expression 的值增 1。

【示例 5-5】下面使用++运算符分别计算 i 变量的前缀增量和后缀增量的值，结果分别

保存为 j 和 k 变量。

```
01  using System;
02  namespace 示例 5_5
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              int i = 10;
09              int j = ++i;          //i 先进行自加运算，将结果保存在变量 j 中
10              int k = i++;          //i 先将值赋给变量 k，再进行自加运算
11              Console.WriteLine("i="+i+",j="+j+",k="+k);
12              Console.ReadLine();
13          }
14      }
15  }
```

分析：“++i”表达式执行之后，i 变量的值为 11，并作为该表达式的值。因此，j 变量的值为 11。“k = i++”表达式首先获取 i 变量的值，并赋值给 k 变量，这时，k 变量的值为 11，然后再将 i 变量的值增 1，因此 i 变量的值为 12。最终，i、j 和 k 变量的值分别为 12、11 和 11。运行结果如图 5.4 所示。

图 5.4　使用++运算符示例

2．– –运算符

– –运算符又称为减量运算符，其作用是对操作数进行减 1。将– –运算符放在操作数的左边，称该运算符为前缀减量运算符。否则，称为后缀减量运算符。语法如下：

```
-- expression
```

或者

```
expression--
```

expression 表示操作数。“– – expression”表达式首先计算 expression 的值，然后将其值减 1，并作为该表达式的结果。“expression – –”表达式首先计算 expression 的值，并作为该表达式的结果，然后将 expression 的值减 1。

【示例 5-6】下面使用– –运算符分别计算 i 变量的前缀减量和后缀减量的值，结果分别保存为 j 和 k 变量。

```
01  using System;
02  namespace 示例 5_6
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
```

```
08            int i = 10;
09            int j = --i;              //i 先进行自减运算，结果保存在变量 j 中
10            int k = i--;              //i 先将值赋给变量 k，再进行自减运算
11            Console.WriteLine("i="+i+",j="+j+",k="+k);
12            Console.ReadLine();
13        }
14    }
15 }
```

分析：“– –i”表达式执行之后，i 变量的值为 9，并作为该表达式的值。因此，j 变量的值为 9。“k = i– –”表达式首先获取 i 变量的值，并赋值给 k 变量，这时，k 变量的值为 9，然后再将 i 变量的值减 1，因此 i 变量的值为 8。最终，i、j 和 k 变量的值分别为 8、9 和 9。运算结果如图 5.5 所示。

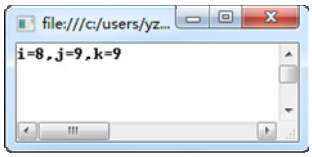

图 5.5　使用--运算符示例

5.2.3　逻辑运算符

&、^、|和!运算符称为逻辑运算符。&运算符计算两个操作数的按位逻辑 AND，|运算符计算两个操作数的按位逻辑 OR，^运算符计算两个操作数的按位逻辑 XOR。语法如下：

```
left expression operator right expression
```

left expression 和 right expression 分别表示左操作数和右操作数，operator 表示运算符，可以为&、^和|。

【示例 5-7】下面使用&、^和|运算符分别计算 i 和 j 两个操作数（均为 int 类型，值分别为 10 和 20）的位逻辑 AND、位逻辑 OR 和位逻辑 XOR，结果分别保存为保存为 c1、c2 和 c3 变量。

```
01 using System;
02 namespace 示例 5_7
03 {
04     class Program
05     {
06         static void Main(string[] args)
07         {
08             int i = 10, j = 20;
09             int c1 = i & j;                  //将 i 和 j 进行按位与运算
10             int c2 = i ^ j;                  //将 i 和 j 进行按位异或运算
11             int c3 = i | j;                  //将 i 和 j 进行按位或运算
12             Console.WriteLine("c1="+c1 );
13             Console.WriteLine("c2=" + c2 );
14             Console.WriteLine("c3=" + c3);
15             Console.ReadLine();
16         }
17     }
18 }
```

分析：c1、c2 和 c3 变量的值分别为 0、30 和 30。运行结果如图 5.6 所示。

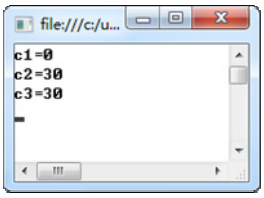

图 5.6　逻辑运算符示例

当左、右操作数均为 bool 类型时，&、^和|运算符分别计算两个操作数的逻辑与、异或和或。给定“x op y”表达式（op 为&、^或|），具体结果描述如下：

- 如果 x 和 y 均为 true，则 x & y 的结果为 true。否则，结果为 false。
- 如果 x 或 y 为 true，则 x | y 的结果为 true。否则，结果为 false。
- 如果 x 为 true 而 y 为 false，或者 x 为 false 而 y 为 true，则 x ^ y 的结果为 true。否则，结果为 false。即 x 和 y 的值不相同时，x ^ y 的结果才为 true。

【示例 5-8】下面使用^运算符计算 i 和 j 两个操作数（均为 bool 类型，值分别为 true 和 false）的位逻辑 XOR，结果保存为 k 变量。

```
01  using System;
02  namespace 示例 5_8
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              bool i = true;
09              bool j = false;
10              bool k = i ^ j;                  //将 i 和 j 进行按位异或运算
11              Console.WriteLine("k 的值为" +k);
12              Console.ReadLine();
13          }
14      }
15  }
```

分析：k 变量的值为 false。运行结果如图 5.7 所示。

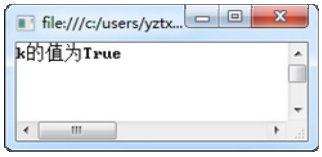

图 5.7　^运算符示例

!运算符又称为逻辑否定运算符。如果操作数的值为 true，则结果为 false；如果操作数为 false，则结果为 true。语法如下：

```
! expression
```

expression 表示操作数。

【示例 5-9】下面使用!运算符计算 i 变量（类型为 bool，值为 true）的逻辑否定的结果，

并保存为 j 变量。运行结果如图 5.8 所示。

```
using System;
namespace 示例 5_9
{
    class Program
    {
        static void Main(string[] args)
        {
            bool i = true;
            bool j = !i;                    //将! i 的值赋给布尔型变量 j
            Console.WriteLine("j 的值为" +j);
            Console.ReadLine();
        }
    }
}
```

图 5.8　!运算符示例

5.2.4　条件运算符

?:运算符称为条件运算符，它是 C#语言中唯一的一个三元运算符。语法如下：

```
条件表达式 ? resulta : resultb;
```

该表达式首先计算条件表达式的值。如果条件表达式的值为真，则计算 resulta 的值，并成为运算结果。否则计算 resultb，并成为运算结果。

【示例 5-10】下面使用?:运算符计算“i + j > 3000 ? i : j”表达式的值。其中，i 和 j 均为 int 类型，值分别为 10 和 20。

```
using System;
namespace 示例 5_10
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 10, j = 20;
            //如果表达式 i+j 的值大于 3000，则将 i 的值赋给 k，否则将 j 的赋给 k
            int k = i + j > 300 ? i : j;
            Console.WriteLine("k=" +k);
            Console.ReadLine();
        }
    }
}
```

分析：如果 i + j 表达式的值大于 300 时，则 k 的值等于 i 的值；否则 k 的值等于 j 的值。运行结果如图 5.9 所示。

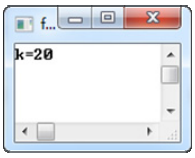

图 5.9　?:运算符示例

如果有一个包含?:运算符的表达式“j ? x : y”，则该表达式的计算步骤如下：

（1）计算 j 的值。

（2）若 j 的类型不为 bool 类型，则将 j 的值转换为 bool 类型。如果转换失败，则发生编译错误。

（3）如果 j 的值为 true，则计算 x 的值，并将结果作为该表达式的结果。

（4）如果 j 的值为 false，则计算 y 的值，并将结果作为该表达式的结果。

在使用?:运算符时，需要注意以下两点。

- ?:运算符的第一个操作数（条件表达式）必须是能够隐式转换为 bool 类型或实现了 operator true 的表达式。否则发生编译错误。
- ?:运算符的第二和 3 个操作数决定?:运算符表达式的结果的类型。第二和 3 个操作数的类型要么相同，要么能够从一个操作数的类型隐式转换为另外一个操作数的类型。否则发生编译时错误。

5.2.5　条件逻辑运算符

&&和||运算符称为条件逻辑运算符。&&运算符为逻辑与运算符，它计算左右操作数的逻辑与。||运算符为逻辑或运算符，它计算左右操作数的逻辑或。语法如下：

```
left expression operator right expression
```

left expression 和 right expression 分别表示左操作数和右操作数。operator 表示运算符，可以为&&和||。

【示例 5-11】下面使用&&运算符计算 i 和 j 两个操作数（均为 bool 类型）的和，结果保存为 k 变量。

```
01  using System;
02  namespace 示例 5_11
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              bool i = true;
09              bool j = false;
10              bool k = i && j;            //将 i 和 j 进行逻辑与运算
11              Console.WriteLine("k 的值为" +k);
12              Console.ReadLine();
13          }
14      }
15  }
```

分析：k 变量的值为 false。运行结果如图 5.10 所示。

图 5.10　逻辑与运算符示例

注意：对于 x && y 表达式而言，仅当 x 的值为 true 时，才计算 y 的值。对于 x || y 表达式而言，仅当 x 的值为 false 时，才计算 y 的值。

【示例 5-12】下面声明类型为 object 的 obj 变量，并省略初始化 obj 变量的代码。“if(obj == null || obj.ToString().Length <= 0)”表达式中的“obj == null”表达式，判断 obj 是否为 null。如果 obj 的值为 null，则“obj.ToString().Length <= 0”表达式不会被执行，从而保证了 obj.ToString().Length 语句不会产生运行时错误（当 obj 为 null 时，obj.ToString().Length 语句会产生运行时错误）。

```
object obj;
//...省略了给 obj 赋值的代码
if(obj == null || obj.ToString().Length <= 0)//在 if 语句的判断表达式中使用逻辑或运算
{
    //...省略其他代码
}
```

注意：当&&或||运算符的左右操作数均为 bool 类型时。x && y 表达式相当于 x ? y : false 表达式，x || y 表达式相当于 x ? true : y 表达式。

5.2.6　移位运算符

<<和>>运算符被称为移位运算符。<<运算符表示左移位，>>运算符表示右移位。语法如下：

```
expression operator count;
```

expression 表示被移位的表达式，count 表示移动的位数，operator 表示运算符，可以为<<和>>。

【示例 5-13】下面使用<<和>>运算符分别将 i（为 int 类型，值为 2008）左移和右移 2 位，结果分别保存为 c1 和 c2 变量。

```
01  using System;
02  namespace 示例 5_13
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              int i = 20;
09              int c1 = i << 2;          //将 i 左移 2 位
10              int c2 = i >> 2;          //将 i 右移 2 位
```

```
            Console.WriteLine("c1=" +c1);
            Console.WriteLine("c2=" + c2);
            Console.ReadLine();
        }
    }
}
```

分析：c1 变量的值为 80，c2 变量的值为 5。运行结果如图 5.11 所示。

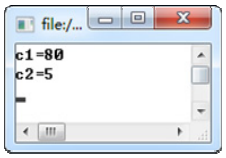

图 5.11　左移右移运算符示例

x << count 表达式将 x 向左移位 count 个位。计算方法为：放弃 x 中经移位后的高序位（如果保留这些高序位，则结果将超出结果类型的范围），将其余的位向左位移，并将空出来的低序位均设置为零。

x >> count 表达式将 x 向右移位 count 个位。计算方法如下：

- 当 x 为 int 或 long 类型时，放弃 x 的低序位，将剩余的位向右位移。如果 x 非负，则将高序空位位置设置为零。如果 x 为负，则将其设置为 1。
- 当 x 为 uint 或 ulong 类型时，放弃 x 的低序位，将剩余的位向右位移，并将高序空位位置设置为零。

5.2.7　关系运算符

==、!=、<、>、<=、>=运算符称为关系。语法如下：

```
left expression operator right expression
```

left expression 和 right expression 分别表示左操作数和右操作数，operator 表示运算符，可以为==、!=、<、>、<=、>=。其中，==、!=、<、>、<=和>=运算符为比较运算符，它们的具体计算方法如下：

- 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。
- x < y，如果 x 小于 y，则为 true，否则为 false。
- x > y，如果 x 大于 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 大于等于 y，则为 true，否则为 false。

【示例 5-14】下面使用==运算符比较 i 和 j 的值（均为 int 类型，值分别为 10 和 20）是否相等，结果保存为 k 变量。

```
using System;
namespace 示例 5_14
{
    class Program
    {
```

```
06          static void Main(string[] args)
07          {
08              int i = 10, j = 20;
09              bool k = i == j;            //因为 i 与 j 的值不相等，所以 k 为 false
10              Console.WriteLine("k 的值为" +k);
11              Console.ReadLine();
12          }
13      }
14  }
```

分析：k 变量的值为 False。运行结果如图 5.12 所示。

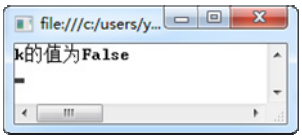

图 5.12　==运算符示例

注意：==、!=、<、>、<=和>=运算符的结果均为 bool 类型。

5.2.8　赋值运算符

=、*=、/=、%=、+=、-=、<<=、>>=、&=、^=和|=运算符被称为赋值运算符，它们能够为变量、属性、事件或索引器元素赋新值。语法如下：

```
left expression operator right expression
```

left expression 和 right expression 分别表示左操作数和右操作数，operator 表示运算符，如=、*=、/=等。

【示例 5-15】下面使用*=运算符计算两个操作数（i 和 j，它们的值分别为 10 和 20）的乘积，并赋值给左操作数 j。

```
01  using System;
02  namespace 示例 5_15
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              int i = 10,j =20;
09              j *= i;                 //将 i*j 的值赋给 j
10              Console.WriteLine("j=" +j);
11              Console.ReadLine();
12          }
13      }
14  }
```

分析：j 变量的值为 200。运行结果如图 5.13 所示。

注意：赋值运算符的左操作数必须是属于变量、属性访问、索引器访问或事件访问类别的表达式。

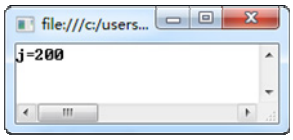

图 5.13　*=运算符示例

=运算符称为简单赋值运算符，它将右操作数的值赋予左操作数给定的变量、属性或索引器元素。op=运算符称为复合赋值运算符，“x op= y”表达式相当于“x = x op y”表达式。

5.2.9　运算优先级

C#语言中的运算符存在着优先级，在计算表达式的值时，必须遵循运算符的优先级的规则。算术表达式是最为常见的一种表达式。它由操作数和运算符组成，而且这些运算符之间是存在一定优先级的，如*运算符的优先级就大于+运算符的优先级。

特别地，当表达式包括多个运算符时，运算符的优先级控制各个运算符的计算顺序。对于 x+y*z 表达式而言，该表达式首先计算 y*z 表达式的值，然后再计算 y*z 表达式的结果与 x 的和。即该表达式等价于 x+(y*z)表达式，那是因为*运算符的优先级大于+运算符的优先级。根据运算符的优先级可以把 C#中的运算符分为以下 14 类，如表 5.2 所示。

表 5.2　C#中的运算符表

类　　型	运　算　符	说　　明
基本	x.y、f(x)、a[x]、x++、x--、new、typeof、checked、unchecked	new 运算符用于创建一个新对象
一元	+、–、!、~、++x、--x、(T)x	(T)x 为类型转换运算
乘除	*、/、%	
加减	+、–	
移位	<<、>>	
关系和类型检测	<、>、<=、>=、is、as	
相等	==、!=	
逻辑与	&	
逻辑异或	^	
逻辑或	|	
条件与	&&	
条件或	||	
条件	?:	
赋值	=、*=、/=、%=、+=、-=、<<=、>>=、&=、^=、|=	

注意：表 5.2 是按照从最高到最低的优先级顺序列举了 C#语言的所有运算符。

当操作数出现在具有相同优先级的两个运算符之间时，运算符的顺序与运算符本身特性相关，具体说明如下所示。

- 除了赋值运算符外，所有的二元运算符都是从左向右执行运算。如“x+y+z”表达式按照“(x+y)+z”表达式进行计算。

- 赋值运算符和?:条件运算符是从右向左执行运算。如 x=y=z 表达式按照 x=(y=z)表达式进行计算。

5.2.10　类型转换

类型转换是指将数据从一种类型转换成另一种类型。在赋值运算中，可能会发生类型转换。如果被赋值的变量类型与实际的变量类型不同，就需要进行类型转换。C#提供了类型转换运算符和类型测试运算符。本节将介绍类型转换的运算符。

(T)x 运算符将表达式显式转换为指定的类型，语法如下：

```
( type ) expression
```

type 表示类型，expression 表示被转换的表达式。

【示例 5-16】下面使用(T)x 运算符将 obj 变量（类型为 object）转换为 int 类型的 i 变量。

```
object obj = 10;
int i = ( object )obj;        //将变量 obj 从 object 型转换为 int 型
```

注意：对于(T)x 表达式而言，如果不存在从 x 的类型到 T 的显式转换，则发生编译错误。否则，结果为显式转换产生的值。

类型测试运算符包括 is 和 as 运算符。is 运算符用于动态检查对象的运行时类型是否与给定类型兼容，其结果也为 bool 类型。对于“e is T”表达式而言（其中，e 为表达式，T 为类型），如果 e 的结果能够隐式转换为 T 类型，则该表达式的结果为 true，否则为 false。

【示例 5-17】下面使用 is 运算符检查 i 变量（类型为 int）能否转换为 object 类型，结果保存为 j 变量。

```
01  using System;
02  namespace 示例 5_17
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              int i = 10;
09              bool j = i is object;   //判断变量 i 是否可以转换为 object 类型
10              Console.WriteLine("j 的值是" +j);
11              Console.ReadLine();
12          }
13      }
14  }
```

分析：j 变量的值为 true。C#中的每种类型都是直接或间接从 object 类类型派生的。因此，C#中的任何类型都可以转换为 object 类型。运行结果如图 5.14 所示。

as 运算符用于将一个值显式地转换为一个给定的引用类型。如果转换失败，则返回 null。

【示例 5-18】下面使用 as 运算符将 i 变量（类型为 int）转换为 object 类型的变量，并

保存为 j 变量。运行结果如图 5.15 所示。

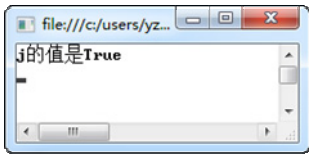

图 5.14　is 运算符示例

```
01  using System;
02  namespace 示例 5_18
03  {
04      class Program
05      {
06          static void Main(string[] args)
07          {
08              int i = 10;
09              object j = i as object;      //将 i 由 int 类型转换为 object 类型
10              Console.WriteLine("j 的值是" +j);
11              Console.ReadLine();
12          }
13      }
14  }
```

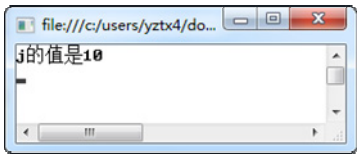

图 5.15　as 运算符示例

注意：as 运算符从不引发异常。对于“e as T”表达式而言，e 必须是表达式，T 必须是引用类型，且运算结果总是能够归属于 T 类型。

5.3　小　　结

本章主要介绍了 C#语言中的表达式和运算符，如表达式分类、this 关键字、new 关键字、运算符，以及运算符重载等。其中，重点是要掌握表达式、运算符和运算符重载，为后续编写 C#程序代码奠定基础。第 6 章将介绍 C#语言中的语句。

5.4　习　　题

【习题 5-1】程序填空题。该程序打印输出变量 i 的值，程序运行结果如图 5.16 所示。

```
using System;
```

```
namespace chapter5_1
{
    class Program
    {
        public int i;
        public Program(int i)
        {
            (1)                                  //将实参的值传给形参
        }
        static void Main(string[] args)
        {
            Program p = new Program(100);
            Console.WriteLine("i 的值是："+(2)    );//打印输出 i 的值
            Console.ReadLine ();
        }
    }
}
```

图 5.16　运行结果图

【习题 5-2】程序填空题。该程序打印输出 i 和 j 的和，程序运行结果如图 5.17 所示。

```
using System;
namespace Chapter5_2
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 123, j = 123;
            Console.WriteLine("i 和 j 的和是：");
            ____________________________//打印输出 i 和 j 的和
            Console.ReadLine ();
        }
    }
}
```

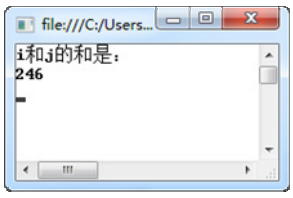

图 5.17　运行结果图

【习题 5-3】读下面的程序，写出程序的运行结果。然后将程序输入编译器，得出正确的运行结果，验证写出的结果是否正确。

```
using System;
namespace Chapter5_3
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 123;
            Console.WriteLine("i 放在自增运算符后面："+ ++i);
            Console.WriteLine("i 放在自增运算符前面："+ i++);
            Console.WriteLine("       i 的最终值是："+i);
            Console.ReadLine ();
        }
    }
}
```

【习题 5-4】读下面的程序，写出程序的运行结果。然后将程序输入编译器，得出正确的运行结果，验证写出的结果是否正确。

```
using System;
namespace Chapter5_4
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 123, j = 123;
            bool b;
            b = i > j;
            Console.WriteLine("b 的值是："+b);
            Console.ReadLine ();
        }
    }
}
```

【习题 5-5】程序填空题。该程序使用赋值运算符将 i 和 j 的和赋给 j，并将 j 的值打印输出。程序运行结果如图 5.18 所示。

```
using System;
namespace Chapter5_5
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 123, j = 123;
            _____________________//使用赋值运算符将 i 和 j 的和赋给 j
            Console.WriteLine("j 的值是："+j);
            Console.ReadLine ();
        }
    }
}
```

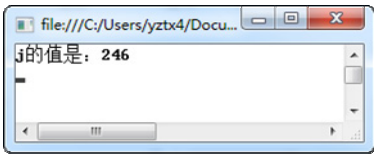

图 5.18　运行效果