**表达式（C# 编程指南）**

“表达式”是由一个或多个操作数以及零个或零个以上的运算符所组成的序列，可以通过计算得到一个值、对象、方法或命名空间等结果。表达式可以包含文本值、方法调用、运算符及其操作数，或简单名称。简单名称可以是变量、类型成员、方法参数、命名空间或类型的名称。

表达式可以使用运算符，而运算符又可以将其他表达式用作参数，或者使用方法调用，而方法调用的参数又可以是其他方法调用，因此表达式既可以非常简单，也可以非常复杂。下面是表达式的两个示例：

((x < 10) && ( x > 5)) || ((x > 20) && (x < 25))

System.Convert.ToInt32("35")

**表达式值**

在大部分使用表达式的上下文中，例如在语句或方法参数中，表达式应计算为某个值。如果 x 和 y 是整数，表达式 x + y 将计算为一个数值。表达式 new MyClass() 计算为对 MyClass 对象的新实例的引用。表达式 myClass.ToString() 计算为一个字符串，因为字符串是该方法的返回类型。然而，虽然命名空间名称归类为表达式，但它不计算为值，因此永远不能作为任何表达式的最终结果。命名空间名称不能传递给方法参数，不能用在新表达式中，也不能赋值给变量。命名空间名称只能用作较大表达式的子表达式。同样如此的还有类型（与 [System.Type](https://msdn.microsoft.com/zh-CN/library/system.type.aspx) 对象不同）、方法组名称（与特定方法不同）以及事件 [add](https://msdn.microsoft.com/zh-CN/library/cc713648.aspx) 和 [remove](https://msdn.microsoft.com/zh-CN/library/cc713642.aspx) 访问器。

每个值都有关联的类型。例如，如果 x 和 y 都是 **int** 类型的变量，则表达式 x + y 的值的类型也是 **int**。如果将该值赋给不同类型的变量，或者如果 x 和 y 是不同的类型，则应用类型转换规则。有关如何进行这种转换的更多信息，请参见[强制转换和类型转换（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173105.aspx)。

**溢出**

如果值大于值类型的最大值，数值表达式可能导致溢出。有关更多信息，请参见[Checked 和 Unchecked（C# 参考）](https://msdn.microsoft.com/zh-CN/library/khy08726.aspx)和 [显式数值转换表（C# 参考）](https://msdn.microsoft.com/zh-CN/library/yht2cx7b.aspx)。

**运算符的优先级和结合性**

计算表达式的方式由结合性和运算符优先级控制。有关更多信息，请参见 [运算符（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173145.aspx)。

除赋值表达式和方法调用表达式之外，大部分表达式都必须嵌在语句中。有关更多信息，请参见 [语句（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173143.aspx)。

**文本和简单名称**

最简单的两种表达式类型是文本和简单名称。文本是没有名称的常数值。例如，在下面的代码示例中，5 和 "Hello World" 都是文本值：

// Expression statements.

int i = 5;

string s = "Hello World";

有关文本的更多信息，请参见 [类型（C# 参考）](https://msdn.microsoft.com/zh-CN/library/3ewxz6et.aspx)。

在上面的示例中，i 和 s 都是用于标识局部变量的简单名称。在表达式中使用这些变量时，变量名称计算为当前在该变量的内存位置所存储的值。下面的示例演示了这种情况：

int num = 5;

System.Console.WriteLine(num); // Output: 5

num = 6;

System.Console.WriteLine(num); // Output: 6

**调用表达式**

在下面的代码示例中，对 DoWork 的调用是一个调用表达式。

DoWork();

方法调用要求使用方法的名称（如前面的示例中那样作为名称，或作为其他表达式的结果），后跟括号和任意方法参数。有关更多信息，请参见 [方法（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173114.aspx)。委托调用使用委托的名称和括号内的方法参数。有关更多信息，请参见 [委托（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/ms173171.aspx)。如果方法返回值，则方法调用和委托调用的计算结果为该方法的返回值。返回 void 的方法不能替代表达式中的值。

**查询表达式**

有关常规表达式的规则同样适用于查询表达式。有关更多信息，请参见 [LINQ 查询表达式（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/bb397676.aspx)。

**Lambda 表达式**

Lambda 表达式表示没有名称但可以具有输入参数和多个语句的“内联方法”。它们在 LINQ 中广泛用于向方法传递参数。Lambda 表达式被编译为委托或表达式树，具体取决于使用它们的上下文。有关更多信息，请参见 [Lambda 表达式（C# 编程指南）](https://msdn.microsoft.com/zh-CN/library/bb397687.aspx)。

**表达式树**

使用表达式树可以将表达式表示为数据结构。表达式树由 LINQ 提供程序广泛使用，用来将查询表达式转换为在其他某个上下文（如 SQL 数据库）中有意义的代码。有关更多信息，请参见[表达式树（C# 和 Visual Basic）](https://msdn.microsoft.com/zh-CN/library/bb397951.aspx)。

**备注**

只要从表达式中识别到变量、对象属性或对象索引器访问，该项的值都会用作表达式的值。在 C# 中，只要表达式的最终计算结果是所需的类型，表达式就可以放置在任何需要值或对象的位置。