C# 封装 C# 可空类型

C# 方法

一个方法是把一些相关的语句组织在一起，用来执行一个任务的语句块。每一个 C# 程序至少有一个带有 Main 方法的类。

要使用一个方法，您需要：

- 定义方法
- 调用方法

C# 中定义方法

当定义一个方法时，从根本上说是在声明它的结构的元素。在 C# 中，定义方法的语法如下：

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
   Method Body
}
```

下面是方法的各个元素：

- **Access Specifier**：访问修饰符，这个决定了变量或方法对于另一个类的可见性。
- **Return type**：返回类型，一个方法可以返回一个值。返回类型是方法返回的值的数据类型。如果方法不返回任何值，则返回类型为 **void**。
- **Method name**：方法名称，是一个唯一的标识符，且是大小写敏感的。它不能与类中声明的其他标识符相同。
- **Parameter list**：参数列表，使用圆括号括起来，该参数是用来传递和接收方法的数据。参数列表是指方法的参数类型、顺序和数量。参数是可选的，也就是说，一个方法可能不包含参数。
- **Method body**：方法主体，包含了完成任务所需的指令集。

实例

下面的代码片段显示一个函数 *FindMax*，它接受两个整数值，并返回两个中的较大值。它有 public 访问修饰符，所以它可以使用类的实例从类的外部进行访问。

实例

```
class NumberManipulator
{
   public int FindMax(int num1, int num2)
   {
      /* 局部变量声明 */
      int result;

      if (num1 > num2)
```

```
         result = num1;
      else
         result = num2;

      return result;
   }
   ...
}
```

C# 中调用方法

您可以使用方法名调用方法。下面的实例演示了这点：

实例

```
using System;

namespace CalculatorApplication
{
   class NumberManipulator
   {
      public int FindMax(int num1, int num2)
      {
         /* 局部变量声明 */
         int result;

         if (num1 > num2)
            result = num1;
         else
            result = num2;

         return result;
      }
      static void Main(string[] args)
      {
         /* 局部变量定义 */
         int a = 100;
         int b = 200;
         int ret;
         NumberManipulator n = new NumberManipulator();

         //调用 FindMax 方法
         ret = n.FindMax(a, b);
         Console.WriteLine("最大值是： {0}", ret );
         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
最大值是： 200
```

您也可以使用类的实例从另一个类中调用其他类的公有方法。例如，方法 *FindMax* 属于 *NumberManipulator* 类，您可以从另一个类 *Test* 中调用它。

实例

```
using System;

namespace CalculatorApplication
{
    class NumberManipulator
    {
        public int FindMax(int num1, int num2)
        {
            /* 局部变量声明 */
            int result;

            if (num1 > num2)
                result = num1;
            else
                result = num2;

            return result;
        }
    }
    class Test
    {
        static void Main(string[] args)
        {
            /* 局部变量定义 */
            int a = 100;
            int b = 200;
            int ret;
            NumberManipulator n = new NumberManipulator();
            //调用 FindMax 方法
            ret = n.FindMax(a, b);
            Console.WriteLine("最大值是： {0}", ret );
            Console.ReadLine();

        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
最大值是： 200
```

递归方法调用

一个方法可以自我调用。这就是所谓的 **递归**。下面的实例使用递归函数计算一个数的阶乘：

实例

```
using System;

namespace CalculatorApplication
{
    class NumberManipulator
    {
        public int factorial(int num)
        {
            /* 局部变量定义 */
            int result;

            if (num == 1)
            {
                return 1;
            }
            else
            {
                result = factorial(num - 1) * num;
                return result;
            }
        }

        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
            //调用 factorial 方法
            Console.WriteLine("6 的阶乘是:  {0}", n.factorial(6));
            Console.WriteLine("7 的阶乘是:  {0}", n.factorial(7));
            Console.WriteLine("8 的阶乘是:  {0}", n.factorial(8));
            Console.ReadLine();

        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
6 的阶乘是:  720
7 的阶乘是:  5040
8 的阶乘是:  40320
```

参数传递

当调用带有参数的方法时，您需要向方法传递参数。在 C# 中，有三种向方法传递参数的方式：

方式	描述
值参数	这种方式复制参数的实际值给函数的形式参数，实参和形参使用的是两个不同内存中的值。在这种情况下，当形参的值发生改变时，不会影响实参的值，从而保证了实参数据的安全。

引用参数	这种方式复制参数的内存位置的引用给形式参数。这意味着，当形参的值发生改变时，同时也改变实参的值。
输出参数	这种方式可以返回多个值。

按值传递参数

这是参数传递的默认方式。在这种方式下，当调用一个方法时，会为每个值参数创建一个新的存储位置。

实际参数的值会复制给形参，实参和形参使用的是两个不同内存中的值。所以，当形参的值发生改变时，不会影响实参的值，从而保证了实参数据的安全。下面的实例演示了这个概念：

实例

```
using System;
namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void swap(int x, int y)
      {
         int temp;

         temp = x; /* 保存 x 的值 */
         x = y;    /* 把 y 赋值给 x */
         y = temp; /* 把 temp 赋值给 y */
      }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* 局部变量定义 */
         int a = 100;
         int b = 200;

         Console.WriteLine("在交换之前，a 的值:  {0}", a);
         Console.WriteLine("在交换之前，b 的值:  {0}", b);

         /* 调用函数来交换值 */
         n.swap(a, b);

         Console.WriteLine("在交换之后，a 的值:  {0}", a);
         Console.WriteLine("在交换之后，b 的值:  {0}", b);

         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
在交换之前，a 的值：100
在交换之前，b 的值：200
在交换之后，a 的值：100
在交换之后，b 的值：200
```

结果表明，即使在函数内改变了值，值也没有发生任何的变化。

按引用传递参数

引用参数是一个对变量的**内存位置的引用**。当按引用传递参数时，与值参数不同的是，它不会为这些参数创建一个新的存储位置。引用参数表示与提供给方法的实际参数具有相同的内存位置。

在 C# 中，使用 **ref** 关键字声明引用参数。下面的实例演示了这点：

实例

```
using System;
namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void swap(ref int x, ref int y)
      {
         int temp;

         temp = x; /* 保存 x 的值 */
         x = y;    /* 把 y 赋值给 x */
         y = temp; /* 把 temp 赋值给 y */
       }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* 局部变量定义 */
         int a = 100;
         int b = 200;

         Console.WriteLine("在交换之前，a 的值：  {0}", a);
         Console.WriteLine("在交换之前，b 的值：  {0}", b);

         /* 调用函数来交换值 */
         n.swap(ref a, ref b);

         Console.WriteLine("在交换之后，a 的值：  {0}", a);
         Console.WriteLine("在交换之后，b 的值：  {0}", b);

         Console.ReadLine();

      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
在交换之前，a 的值：100
在交换之前，b 的值：200
在交换之后，a 的值：200
在交换之后，b 的值：100
```

结果表明，*swap* 函数内的值改变了，且这个改变可以在 *Main* 函数中反映出来。

按输出传递参数

return 语句可用于只从函数中返回一个值。但是，可以使用 **输出参数** 来从函数中返回两个值。输出参数会把方法输出的数据赋给自己，其他方面与引用参数相似。

下面的实例演示了这点：

实例

```
using System;

namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void getValue(out int x )
      {
         int temp = 5;
         x = temp;
      }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* 局部变量定义 */
         int a = 100;

         Console.WriteLine("在方法调用之前，a 的值： {0}", a);

         /* 调用函数来获取值 */
         n.getValue(out a);

         Console.WriteLine("在方法调用之后，a 的值： {0}", a);
         Console.ReadLine();

      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
在方法调用之前，a 的值： 100
在方法调用之后，a 的值： 5
```

提供给输出参数的变量不需要赋值。当需要从一个参数没有指定初始值的方法中返回值时，输出参数特别有用。请看下面的实例，来理解这一点：

实例

```
using System;

namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void getValues(out int x, out int y )
      {
          Console.WriteLine("请输入第一个值:  ");
          x = Convert.ToInt32(Console.ReadLine());
          Console.WriteLine("请输入第二个值:  ");
          y = Convert.ToInt32(Console.ReadLine());
      }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* 局部变量定义 */
         int a , b;

         /* 调用函数来获取值 */
         n.getValues(out a, out b);

         Console.WriteLine("在方法调用之后，a 的值:  {0}", a);
         Console.WriteLine("在方法调用之后，b 的值:  {0}", b);
         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果（取决于用户输入）：

```
请输入第一个值:
7
请输入第二个值:
8
在方法调用之后，a 的值:  7
在方法调用之后，b 的值:  8
```

← C# 封装 C# 可空类型 →