C# continue 语句 C# 方法

C# 封装

封装 被定义为"把一个或多个项目封闭在一个物理的或者逻辑的包中"。在面向对象程序设计方法论中，封装是为了防止对实现细节的访问。

抽象和封装是面向对象程序设计的相关特性。抽象允许相关信息可视化，封装则使开发者*实现所需级别的抽象*。

C# 封装根据具体的需要，设置使用者的访问权限，并通过 **访问修饰符** 来实现。

一个 **访问修饰符** 定义了一个类成员的范围和可见性。C# 支持的访问修饰符如下所示：

- public：所有对象都可以访问；
- private：对象本身在对象内部可以访问；
- protected：只有该类对象及其子类对象可以访问
- internal：同一个程序集的对象可以访问；
- protected internal：访问限于当前程序集或派生自包含类的类型。

Public 访问修饰符

Public 访问修饰符允许一个类将其成员变量和成员函数暴露给其他的函数和对象。任何公有成员可以被外部的类访问。

下面的实例说明了这点：

实例

```
using System;

namespace RectangleApplication
{
    class Rectangle
    {
        //成员变量
        public double length;
        public double width;

        public double GetArea()
        {
            return length * width;
        }
        public void Display()
        {
            Console.WriteLine("长度:  {0}", length);
            Console.WriteLine("宽度:  {0}", width);
            Console.WriteLine("面积:  {0}", GetArea());
        }
    }// Rectangle 结束

    class ExecuteRectangle
    {
```

```
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.length = 4.5;
            r.width = 3.5;
            r.Display();
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
长度： 4.5
宽度： 3.5
面积： 15.75
```

在上面的实例中，成员变量 length 和 width 被声明为 **public**，所以它们可以被函数 Main() 使用 Rectangle 类的实例 **r** 访问。

成员函数 *Display()* 和 *GetArea()* 可以直接访问这些变量。

成员函数 *Display()* 也被声明为 **public**，所以它也能被 *Main()* 使用 Rectangle 类的实例 **r** 访问。

Private 访问修饰符

Private 访问修饰符允许一个类将其成员变量和成员函数对其他的函数和对象进行隐藏。只有同一个类中的函数可以访问它的私有成员。即使是类的实例也不能访问它的私有成员。

下面的实例说明了这点：

实例

```
using System;

namespace RectangleApplication
{
    class Rectangle
    {
        //成员变量
        private double length;
        private double width;

        public void Acceptdetails()
        {
            Console.WriteLine("请输入长度: ");
            length = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("请输入宽度: ");
            width = Convert.ToDouble(Console.ReadLine());
        }
        public double GetArea()
        {
            return length * width;
        }
        public void Display()
```

```
        {
            Console.WriteLine("长度:  {0}", length);
            Console.WriteLine("宽度:  {0}", width);
            Console.WriteLine("面积:  {0}", GetArea());
        }
    }//end class Rectangle
    class ExecuteRectangle
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.Acceptdetails();
            r.Display();
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
请输入长度:
4.4
请输入宽度:
3.3
长度:  4.4
宽度:  3.3
面积:  14.52
```

在上面的实例中，成员变量 length 和 width 被声明为 **private**，所以它们不能被函数 Main() 访问。

成员函数 *AcceptDetails()* 和 *Display()* 可以访问这些变量。

由于成员函数 *AcceptDetails()* 和 *Display()* 被声明为 **public**，所以它们可以被 *Main()* 使用 Rectangle 类的实例 **r** 访问。

Protected 访问修饰符

Protected 访问修饰符允许子类访问它的基类的成员变量和成员函数。这样有助于实现继承。我们将在继承的章节详细讨论这个。更详细地讨论这个。

Internal 访问修饰符

Internal 访问说明符允许一个类将其成员变量和成员函数暴露给当前程序中的其他函数和对象。换句话说，带有 internal 访问修饰符的任何成员可以被定义在该成员所定义的应用程序内的任何类或方法访问。

下面的实例说明了这点：

实例

```
using System;

namespace RectangleApplication
{
    class Rectangle
```

```
    {
        //成员变量
        internal double length;
        internal double width;

        double GetArea()
        {
            return length * width;
        }
       public void Display()
        {
            Console.WriteLine("长度:  {0}", length);
            Console.WriteLine("宽度:  {0}", width);
            Console.WriteLine("面积:  {0}", GetArea());
        }
    }//end class Rectangle
    class ExecuteRectangle
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.length = 4.5;
            r.width = 3.5;
            r.Display();
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
长度:  4.5
宽度:  3.5
面积:  15.75
```

在上面的实例中，请注意成员函数 *GetArea()* 声明的时候不带有任何访问修饰符。如果没有指定访问修饰符，则使用类成员的默认访问修饰符，即为 **private**。

Protected Internal 访问修饰符

Protected Internal 访问修饰符允许在本类,派生类或者包含该类的程序集中访问。这也被用于实现继承。

← C# continue 语句 C# 方法 →