

# C# 多态性

**多态性**意味着有多重形式。在面向对象编程范式中，多态性往往表现为"一个接口，多个功能"。

多态性可以是静态的或动态的。在**静态多态性**中，函数的响应是在编译时发生的。在**动态多态性**中，函数的响应是在运行时发生的。

## 静态多态性

在编译时，函数和对象的连接机制被称为早期绑定，也被称为静态绑定。C# 提供了两种技术来实现静态多态性。分别为：

- 函数重载
- 运算符重载

运算符重载将在下一章节讨论，接下来我们将讨论函数重载。

## 函数重载

您可以在同一个范围内对相同的函数名有多个定义。函数的定义必须彼此不同，可以是参数列表中的参数类型不同，也可以是参数个数不同。不能重载只有返回类型不同的函数声明。

下面的实例演示了几个相同的函数 **print()**，用于打印不同的数据类型：

### 实例

```
using System;
namespace PolymorphismApplication
{
    class Printdata
    {
        void print(int i)
        {
            Console.WriteLine("Printing int: {0}", i );
        }

        void print(double f)
        {
            Console.WriteLine("Printing float: {0}" , f);
        }

        void print(string s)
        {
            Console.WriteLine("Printing string: {0}", s);
        }
        static void Main(string[] args)
        {
            Printdata p = new Printdata();
            // 调用 print 来打印整数
            p.print(5);
        }
    }
}
```

```
// 调用 print 来打印浮点数
p.print(500.263);
// 调用 print 来打印字符串
p.print("Hello C++");
Console.ReadKey();
}
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Printing int: 5
Printing float: 500.263
Printing string: Hello C++
```

## 动态多态性

C# 允许您使用关键字 **abstract** 创建抽象类，用于提供接口的部分类的实现。当一个派生类继承自该抽象类时，实现即完成。

**抽象类**包含抽象方法，抽象方法可被派生类实现。派生类具有更专业的功能。

请注意，下面是有关抽象类的一些规则：

- 您不能创建一个抽象类的实例。
- 您不能在一个抽象类外部声明一个抽象方法。
- 通过在类定义前面放置关键字 **sealed**，可以将类声明为**密封类**。当一个类被声明为 **sealed** 时，它不能被继承。抽象类不能被声明为 sealed。

下面的程序演示了一个抽象类：

### 实例

```
using System;
namespace PolymorphismApplication
{
    abstract class Shape
    {
        abstract public int area();
    }
    class Rectangle: Shape
    {
        private int length;
        private int width;
        public Rectangle( int a=0, int b=0)
        {
            length = a;
            width = b;
        }
        public override int area ()
        {
            Console.WriteLine("Rectangle 类的面积: ");
            return (width * length);
        }
    }
}
```

```
    }  
}  
  
class RectangleTester  
{  
    static void Main(string[] args)  
    {  
        Rectangle r = new Rectangle(10, 7);  
        double a = r.area();  
        Console.WriteLine("面积: {0}",a);  
        Console.ReadKey();  
    }  
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Rectangle 类的面积：  
面积: 70
```

当有一个定义在类中的函数需要在继承类中实现时，可以使用**虚方法**。虚方法是使用关键字 **virtual** 声明的。虚方法可以在不同的继承类中有不同的实现。对虚方法的调用是在运行时发生的。

动态多态性是通过 **抽象类** 和 **虚方法** 实现的。

下面的程序演示了这点：

### 实例


```
using System;  
namespace PolymorphismApplication  
{  
    class Shape  
    {  
        protected int width, height;  
        public Shape( int a=0, int b=0)  
        {  
            width = a;  
            height = b;  
        }  
        public virtual int area()  
        {  
            Console.WriteLine("父类的面积: ");  
            return 0;  
        }  
    }  
    class Rectangle: Shape  
    {  
        public Rectangle( int a=0, int b=0): base(a, b)  
        {  
        }  
        public override int area ()  
        {  
        }  
    }  
}
```

```
        Console.WriteLine("Rectangle 类的面积: ");
        return (width * height);
    }
}
class Triangle: Shape
{
    public Triangle(int a = 0, int b = 0): base(a, b)
    {


    }
    public override int area()
    {
        Console.WriteLine("Triangle 类的面积: ");
        return (width * height / 2);
    }
}
class Caller
{
    public void CallArea(Shape sh)
    {
        int a;
        a = sh.area();
        Console.WriteLine("面积: {0}", a);
    }
}
class Tester
{
    static void Main(string[] args)
    {
        Caller c = new Caller();
        Rectangle r = new Rectangle(10, 7);
        Triangle t = new Triangle(10, 5);
        c.CallArea(r);
        c.CallArea(t);
        Console.ReadKey();
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Rectangle 类的面积:
面积: 70
Triangle 类的面积:
面积: 25
```



6 篇笔记

 写笔记