# **[C#多态](http://www.cnblogs.com/200325074/archive/2008/05/23/1206116.html)**

多态性（C# 编程指南）转自MSDN

通过继承，一个类可以用作多种类型：可以用作它自己的类型、任何基类型，或者在实现接口时用作任何接口类型。这称为多态性。**C# 中的每种类型都是多态的**。类型可用作它们自己的类型或用作 Object 实例，因为任何类型都自动将 **Object** 当作基类型。

多态性不仅对派生类很重要，对基类也很重要。任何情况下，使用基类实际上都可能是在使用已强制转换为基类类型的派生类对象。基类的设计者可以预测到其基类中可能会在派生类中发生更改的方面。例如，表示汽车的基类可能包含这样的行为：当考虑的汽车为小型货车或敞篷汽车时，这些行为将会改变。基类可以将这些类成员标记为虚拟的，从而允许表示敞篷汽车和小型货车的派生类重写该行为。

# **多态性概述**

当派生类从基类继承时，它会获得基类的所有方法、字段、属性和事件。**面向对象的语言使用虚方法表达多态。**若要更改基类的数据和行为，您有两种选择：可以使用新的派生成员替换基成员，或者可以重写虚拟的基成员。  
  
使用新的派生成员替换基类的成员需要使用 new 关键字。如果基类定义了一个方法、字段或属性，则 **new** 关键字用于在派生类中创建该方法、字段或属性的新定义。**new** 关键字放置在要替换的类成员的返回类型之前。例如：

 1IMG_256public class BaseClass  
 2IMG_257{  
 3IMG_258    public void DoWork() { }  
 4IMG_259    public int WorkField;  
 5IMG_260    public int WorkProperty  
 6IMG_261    {  
 7IMG_262        get { return 0; }  
 8IMG_263    }  
 9IMG_264}  
10IMG_265public class DerivedClass : BaseClass  
11IMG_266{  
12IMG_267    public new void DoWork() { }  
13IMG_268    public new int WorkField;  
14IMG_269    public new int WorkProperty  
15IMG_270    {  
16IMG_271        get { return 0; }  
17IMG_272    }  
18IMG_273}  
19IMG_274

使用 **new** 关键字时，调用的是新的类成员而不是已被替换的基类成员。这些基类成员称为隐藏成员。如果将派生类的实例强制转换为基类的实例，就仍然可以调用隐藏类成员。例如：

DerivedClass B = new DerivedClass();  
B.DoWork();  // Calls the new method.  
  
BaseClass A = (BaseClass)B;  
A.DoWork();  // Calls the old method.

为了使派生类的实例完全接替来自基类的类成员，基类必须将该成员声明为虚拟的。这是通过在该成员的返回类型之前添加 **virtual** 关键字来实现的。然后，派生类可以选择使用**override** 关键字而不是 new，将基类实现替换为它自己的实现。例如：

public class BaseClass  
{  
    public virtual void DoWork() { }       
    public virtual int WorkProperty  
    {  
        get { return 0; }  
    }  
}  
public class DerivedClass : BaseClass  
{  
    public override void DoWork() { }  
    public override int WorkProperty  
    {  
        get { return 0; }  
    }  
}

字段不能是虚拟的，只有方法、属性、事件和索引器才可以是虚拟的。当派生类重写某个虚拟成员时，即使该派生类的实例被当作基类的实例访问，也会调用该成员。例如：

DerivedClass B = new DerivedClass();  
B.DoWork();  // Calls the new method.  
  
BaseClass A = (BaseClass)B;  
A.DoWork();  // Also calls the new method.

使用虚拟方法和属性可以预先计划未来的扩展。由于在调用虚拟成员时不考虑调用方正在使用的类型，所以派生类可以选择完全更改基类的外观行为。

无论在派生类和最初声明虚拟成员的类之间已声明了多少个类，虚拟成员都将永远为虚拟成员。如果类 A 声明了一个虚拟成员，类 B 从 A 派生，类 C 从类 B 派生，则类 C 继承该虚拟成员，并且可以选择重写它，而不管类 B 是否为该成员声明了重写。例如：

public class A  
{  
    public virtual void DoWork() { }  
}  
public class B : A  
{  
    public override void DoWork() { }  
}

public class C : B  
{  
    public override void DoWork() { }  
}

派生类可以通过将重写声明为密封的来停止虚拟继承。这需要在类成员声明中将 **sealed** 关键字放在 **override** 关键字的前面。例如：

public class C : B  
{  
    public sealed override void DoWork() { }  
}

在上面的示例中，方法 DoWork 对从 C 派生的任何类都不再是虚拟的。它对 C 的实例仍然是虚拟的 -- 即使将这些实例强制转换为类型 B 或类型 A。派生类可以通过使用 **new** 关键字替换密封的方法，如下面的示例所示：

public class D : C  
{  
    public new void DoWork() { }  
}

在此情况下，如果在 D 中使用类型为 D 的变量调用 DoWork，被调用的将是新的 DoWork。如果使用类型为 C、B 或 A 的变量访问 D 的实例，对 DoWork 的调用将遵循虚拟继承的规则，即把这些调用传送到类 C 的 DoWork 实现。  
  
已替换或重写某个方法或属性的派生类仍然可以使用基关键字访问基类的该方法或属性。例如：

public class A  
{  
    public virtual void DoWork() { }  
}  
public class B : A  
{  
    public override void DoWork() { }  
}

public class C : B  
{  
    public override void DoWork()  
    {  
        // Call DoWork on B to get B's behavior:  
        base.DoWork();  
  
        // DoWork behavior specific to C goes here:  
        // ...  
    }  
}  
  
注意

建议虚拟成员在它们自己的实现中使用 **base** 来调用该成员的基类实现。允许基类行为发生使得派生类能够集中精力实现特定于派生类的行为。未调用基类实现时，由派生类负责使它们的行为与基类的行为兼容。

http://www.cnblogs.com/200325074/articles/1206116.html