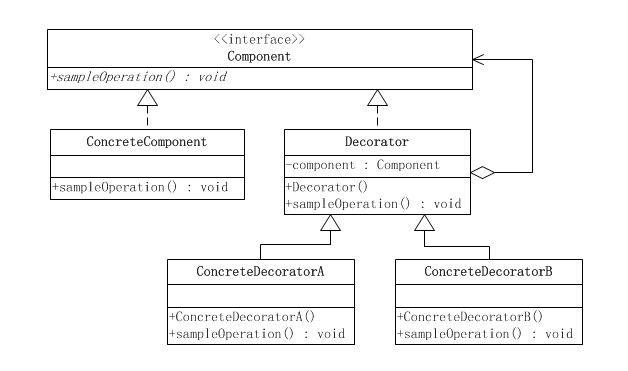
**装饰者模式-结构型模式**

**定义：**装饰者模式又名包装(Wrapper)模式，该模式以对客户端透明的方式动态地扩展对象的功能，是继承关系更弹性的一个很好的替代方案。

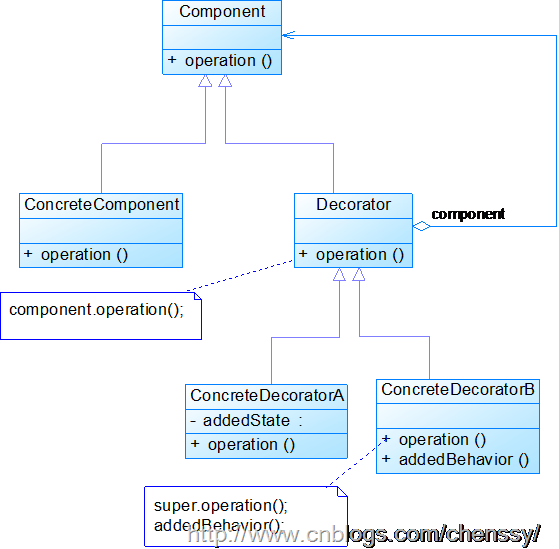
**目的：**是对已有的业务逻辑进一步封装，使其增加额外的功能，JAVA的IO流就大量使用了装饰着模式，用户使用的时候可以任意组装已到达自己想要的效果。

**结构：**装饰者模式以对客户端透明的方式动态地给一个对象加上更多的责任操作，也就是说客户端并不感觉到对象在装饰前和装饰后有什么不同。装饰者模式可以在不创造更多子类的情况下扩展对象的功能。

**类图：**



更具体的：



**角色：**

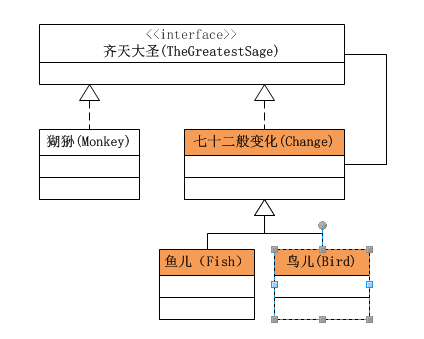
**抽象构件角色 Component：**定义一个抽象接口，以规范准备接收附加责任的对象，通过给这些对象动态添加职责实现。

**具体构件角色 ConcreteComponent：**定义一个将要接收附加责任的具体类，也就是装饰系统中的本来元素、本尊，也可以给这个对象添加一些职责。

**装饰角色 Decorator：**抽象装饰类，实现或继承自Component，持有一个构件对象的实例，并定义一个与抽象构件接口一致的接口，但是对于Component来说，是无需知道Decora的存在的。

**具体装饰角色 ConcreteDecorator：**负责给构件对象Component附加责任的功能。

**装饰者模式完整呈现例子：DecoratorSage—悟空七十二变**

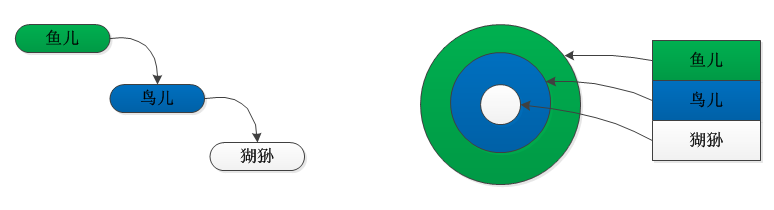


**抽象构件角色：**TheSage接口，定义一个move()方法，这是所有的具体构件类和具体装饰者必须实现的规范。

**具体构件角色：**Monkey，也就是本尊。

**抽象装饰角色：Change类**，保持对客户端的透明性封装，持有一个具体构件对象的实例(具体装饰者)且定义了一个与抽象构件TheSage一致的接口（其实是具体装饰者的父类），并将具体装饰行为委派给具体装饰角色。

**具体装饰角色：Bird、Fish类，**装饰具体构件-本尊 猴子，给其增加附加的功能或责任。

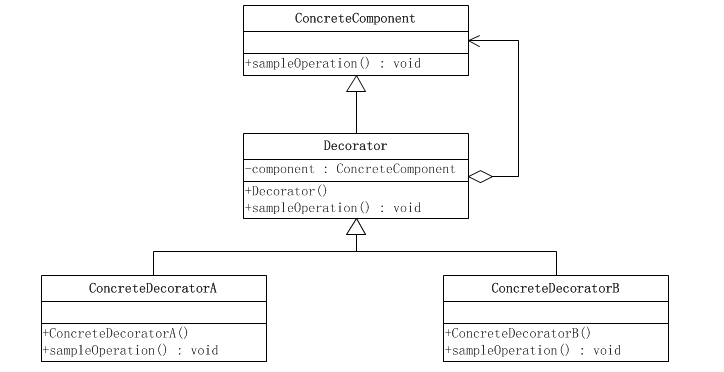


**如图所示，猴子经过变化附加了鸟儿能飞、鱼儿能游的功能，所以这不是一只普通的猴子，而是大圣！**

**大多数情况下，装饰模式的实现可能都会比上述例子要简单**

如果只有ConcreteComponent类，那么就可以考虑去掉抽象Component接口，进而把Decorator作为ConcreteComponent子类：

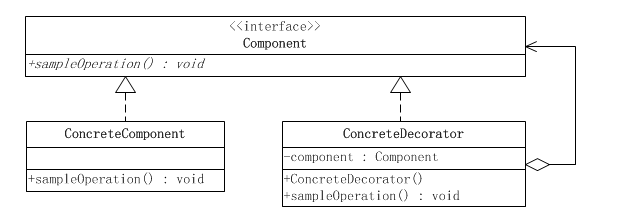
**参考DecratorFood代码：**



**只有一个具体装饰者角色ConcreteComponent的简化：**

如果只有一个具体的装饰者角色，那么就没必要建立一个单独Decorator类，可以把Decorator和ConcreteDecorator的责任合为一体。甚至只有两个ConcreateDecorator情况下都可以这样做。

**类图：**



**透明性要求：不一定要遵守，如果没有遵守可以定义为半透明装饰模式**

要求程序不能将具体构件、具体装饰者声明成自己，必须声明为抽象构件Component类型的变量。

**例如TheSage例子中下面的做法是不对的：**

Monkey sage = new Monkey();

Bird bird = new Bird(sage);

**而应该是：**

TheSage sage = new Monkey();

TheSage bird = new Bird(sage);

**半透明装饰模式：**

**装饰模式优点：**

**装饰模式缺点：**

**装饰模式在JAVA IO中的应用：**