**装饰器模式**

装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。

**这种模式将职责划分出来弄成一个个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。**

**介绍**

**意图：** 动态地给一个对象添加一些额外的职责。

**主要解决：** 一般的，我们为了扩展一个类经常使用继承方式实现，由于继承为类引入静态特征，并且随着扩展功能的增多，子类会很膨胀。

**何时使用：** 在不想增加很多子类的情况下扩展类。

**如何解决：** 将具体功能职责划分，同时继承装饰者模式。

**关键代码：**  1、Component 类充当抽象角色，不应该具体实现。 2、修饰类引用和继承 Component 类，具体扩展类重写父类方法。

**应用实例：**  孙悟空有 72 变，当他变成"庙宇"后，他的根本还是一只猴子，但是他又有了庙宇的功能。

**优点：** 装饰类和被装饰类可以独立发展，不会相互耦合，装饰模式是继承的一个替代模式，装饰模式可以动态扩展一个实现类的功能。

**缺点：** 多层装饰比较复杂。

**使用场景：**  1、扩展一个类的功能。 2、动态增加功能，动态撤销。

**注意事项：** 可代替继承。

**桥接模式与装饰的区别:  
装饰模式:**  
      这两个模式在一定程度上都是为了减少子类的数目，避免出现复杂的继承关系。但是它们解决的方法却各有不同，**装饰模式把子类中比基类中多出来的部分放到单独的类(包装类)里面，以适应新功能增加的需要**，当我们把描述新功能的类封装到基类的对象里面时，就得到了所需要的子类对象，这些描述新功能的类通过组合可以实现很多的功能组合 .

**桥接模式：**  
          **桥接模式则把原来的基类的实现化细节抽象出来(draw())，在构造到一个实现化的结构中(DrawAPI)，然后再把原来的基类(RedCircle,GreenCircle)改造成一个抽象化的等级结构(DrawAPI<-RedCircle/GreenCircle)，这样就可以实现系统在多个维度上的独立变化 。**