# 装饰器模式

1. **基本概念**

### **1、定义：**

装饰器（Decorator）模式的定义：指在不改变现有对象结构的情况下，动态地给该对象增加一些职责（即增加其额外功能）的模式，它属于对象结构型模式。

简单来说，就是当我们相对一个方法进行不同组合方式的时候，就可以用到装饰器模式，例如先穿内裤再穿外裤 或者先穿裤再传内裤这样。

### 2、组成环节：

抽象对象类（Component）：定义了原始对象和装饰器对象的公共接口或抽象类，可以是具体组件类的父类或接口。

具体对象类（Concrete Component）：是被装饰的原始对象，它定义了需要添加新功能的对象。

抽象装饰类（Decorator）：继承自抽象组件，它包含了一个抽象组件对象，并定义了与抽象组件相同的接口，同时可以通过组合方式持有其他装饰器对象。

具体装饰类（Concrete Decorator）：实现了抽象装饰器的接口，负责向抽象组件添加新的功能。具体装饰器通常会在调用原始对象的方法之前或之后执行自己的操作。

## 二、go语言中实现

##### 1、首先定义一个Component类型

type component interface {  
 draw() string  
}

##### 2、定义两个具体对象类

type square struct {  
 c component  
}

type circle struct {  
 c component  
}

##### 3、定义一个抽象装饰类

type decorate struct {  
 c component  
}

##### 4、定义两个具体装饰类

type redDecorate struct {  
 d decorate  
}  
  
type blueDecorate struct {  
 d decorate  
}

##### 5、为以上具体类添加上draw函数的实现方法

func (c circle) draw() string {  
 return "circle "  
}

func (s square) draw() string {  
 return "square "  
}

func (d decorate) draw() string {  
 return d.c.draw()  
}

func (r redDecorate) draw() string {  
 return r.d.c.draw() + "red"  
}  
  
func (b blueDecorate) draw() string {  
 return b.d.c.draw() + "blue"  
}

##### 6、通过main函数实现

func main() {  
 //var co component  
 var c = circle{}  
 var s = square{}  
  
 var rs = redDecorate{d: decorate{s}}  
 var bs = blueDecorate{d: decorate{s}}  
 var rc = redDecorate{d: decorate{c}}  
 var bc = blueDecorate{d: decorate{c}}  
 //r.draw(co)  
 //b.draw(co)  
 a := rs.draw()  
 b := rc.draw()  
 d := bs.draw()  
 e := bc.draw()  
 log.Println(a, b, d, e)  
}

##### 7、输出结果

**square red circle red square blue circle blue**