# 外观模式

1. **基本概念**

### **1、定义：**

外观（Facade）模式的定义：外观模式，为子系统中的一组接口提供一个一致的界面，这个设计模式定义了一个高层接口，这个接口使得这一子系统更加容易使用。

简单来说，外观模式就是让用户与底层实现的逻辑进行隔离，从而降低耦合度。

一般在底层逻辑多且复杂的时候进行使用。

### 2、组成环节：

外观类（Facade）：知道哪些子系统类负责处理请求，并将子系统合理的封装暴露给客户。

子系统集合类（SubSystem）：外观类的子类，实现子系统的功能，处理Façade对象指派的任务。与一般子类不一样的是，子系统集合类并不对Façade里的方法进行继承。

## 二、go语言中实现

##### 1、首先定义一个Facade的外观类

type FacadeInterface interface {  
 MethodOne()  
 MethodTwo()  
}  
  
type Facade struct{}

##### 2、实现外观类对客户端暴露的接口

func (f Facade) MethodOne() {  
 ca := ConcreteA{}  
 ca.MethodA()  
 cb := ConcreteB{}  
 cb.MethodB()  
}  
  
func (f Facade) MethodTwo() {  
 ca := ConcreteA{}  
 ca.MethodA()  
 cb := ConcreteC{}  
 cb.MethodC()  
}

##### 3、创建三个子系统类

type ConcreteA struct{}

type ConcreteB struct{}

type ConcreteC struct{}

##### 4、实现三个子系统类的具体方法

func (ca ConcreteA) MethodA() {  
 fmt.Println("it's ConcreteA")  
}  
  
func (cb ConcreteB) MethodB() {  
 fmt.Println("it's ConcreteB")  
}  
  
func (cc ConcreteC) MethodC() {  
 fmt.Println("it's ConcreteC")  
}

##### 5、通过main函数实现

func main() {  
 f := Facade{}  
 f.MethodTwo()  
 f.MethodOne()  
}

##### 5、输出结果

**it's ConcreteA**

**it's ConcreteC**

**it's ConcreteA**

**it's ConcreteB**