# 命令模式

## 一、基本概念 **1、定义：**

##### 命令模式（Command），讲一个请求封装成一个对象，从而使你可用不同的请求对客户进行参数化，对请求排队或者记录请求日志，以及支持可撤销的操作。

##### 优势是调用者可以避免直接与直接执行命令者进行接触，仅由中间的invoker进行交互即可满足自己的需求。

### 2、组成环节：

##### Invoker(调用类):可以要求该命令执行这个请求。

##### Command(命令类):用来生命执行操作的接口。

##### ConcreteCommand(具体命令类):将一个接收者对象绑定于一个动作，调用者接收者相应的操作，以实现Execute方法。

##### Invoker(调用类):可以要求该命令执行这个请求。

##### Receiver(执行类):知道如何实施与执行一个相关的操作，任何类都可能作为一个接收者。

## 二、go语言中实现

##### 1、首先定义Receive类，并且对Receive类中执行各具体命令的方法进行定义操作。

type Receiver struct{}  
  
func (r Receiver) ExcuteCommandA() {  
 println("Excute commandA.")  
}  
  
func (r Receiver) ExcuteCommandB() {  
 println("Excute commandB.")  
}

##### 2、对Command方法进行定义，并且对其子类ConcreteCommand方法进行定义，并且实现其中的Execute方法。

type Command interface {  
 Excute()  
}  
  
type ConcreteCommandA struct {  
 Receiver  
}  
  
func (c ConcreteCommandA) Excute() {  
 c.Receiver.ExcuteCommandA()  
}  
  
type ConcreteCommandB struct {  
 Receiver  
}  
  
func (c ConcreteCommandB) Excute() {  
 c.Receiver.ExcuteCommandB()  
}

##### 对Invoker方法进行定义，并且对其中的接受和执行命令方法进行定义操作。

type Invoker struct {  
 c Command  
}  
  
func (i \*Invoker) Set(c Command) {  
 i.c = c  
}  
  
func (i \*Invoker) Do() {  
 i.c.Excute()  
}

##### 通过main方法进行client端调用。

func main() {  
 i := Invoker{}  
 ca := ConcreteCommandA{}  
 cb := ConcreteCommandB{}  
  
 i.Set(ca)  
 i.Do()  
  
 i.Set(cb)  
 i.Do()  
}

##### 输出结果

**Excute commandA.**

**Excute commandB.**