# 策略模式

## 基本概念

### 1、定义：

#### 策略模式：它定义了家族算法，并把其分别封装起来，让它们之间可以互相替换。这个模式让算法的变化，不影响到用户算法的使用。

**策略模式包含以下几个核心角色：**

环境（Context）：维护一个对策略对象的引用，负责将客户端请求委派给具体的策略对象执行。环境类可以通过依赖注入、简单工厂等方式来获取具体策略对象。

抽象策略（Abstract Strategy）：定义了策略对象的公共接口或抽象类，规定了具体策略类必须实现的方法。

具体策略（Concrete Strategy）：实现了抽象策略定义的接口或抽象类，包含了具体的算法实现。

## go语言中实现：

### 1、首先定义一个context类

type Context struct {  
 s Strategy  
}

### 2、再定义一个抽象的Strategy类

type Strategy interface {  
 Do()  
}

### 3、定义两个具体的Strategy类

type StrategyA struct{}  
  
type StrategyB struct{}

### 4、对两个具体的Strategy类进行do方法实现

func (s StrategyA) Do() {  
 fmt.Printf("StrategyA do.\n")  
}  
  
func (s StrategyB) Do() {  
 fmt.Printf("StrategyB do.\n")  
}

### 5、对context类进行Strategy的具体实现

func (c Context) SetNoFactory(s Strategy) {  
 c.s = s  
 c.s.Do()  
}

### 6、main函数进行实现

func main() {  
 var c Context  
 var sa StrategyA  
 var sb StrategyB  
 c.SetNoFactory(sa)  
 c.SetNoFactory(sb)  
}

## 与简单工厂共同实现：

### 1、对context类根据输入选择类型

func (c Context) Set(str string) {  
 var s Strategy  
 switch str {  
 case *StrategyAName*:  
 s = StrategyA{}  
 case *StrategyBName*:  
 s = StrategyB{}  
 }  
 c.s = s  
 c.s.Do()  
}

### 2、main函数进行实现

func main() {  
 var c Context  
 c.Set(*StrategyAName*)  
 c.Set(*StrategyBName*)  
}

优势：客户端仅需创造一个var c Context的变量，而之前需要定义三个变量，从而降低耦合度。