**简单的说策略模式就是**对一组算法，将每个算法封装到具有共同接口的独立的类中，从而使得它们可以相互替换

适用场景：根据不同等级的客户或根据不同的节日 打折不同8折,9折

**<?php***/\*\*  
 \* Created by PhpStorm.  
 \* User: SWESWE  
 \* Date: 2019/10/12  
 \* Time: 13:42  
 \*/  
//抽象策略角色---通常由一个接口或抽象类实现---作用：给具体的决策类提供接口***interface** strategy  
{  
 **public function** countPrice($price);  
}  
*//具体决策角色--用于实现具体的算法***class** StrategyA **implements** strategy{  
  
 **public function** countPrice($price)  
 {  
 *//* ***TODO: Implement countPrice() method.* echo"A方案价格+100"**;  
 **return** $price+100;  
 }  
}  
  
**class** StrategyB **implements** strategy{  
  
 **public function** countPrice($price)  
 {  
 *//* ***TODO: Implement countPrice() method.* echo"B方案价格+120"**;  
 **return** $price+120;  
 }  
}  
**class** StrategyC **implements** strategy{  
  
 **public function** countPrice($price)  
 {  
 *//* ***TODO: Implement countPrice() method.* echo"C方案价格+140"**;  
 **return** $price+140;  
 }  
}  
*//环境角色--简单的说这一角色就是告诉你那个算法要干什么  
//持有抽象策略角色的引用***class** setPrice{  
 **private $strategy**;*//用于接收传入的策略对象* **public function** \_\_construct($instance)  
 {  
 $this->**strategy**=$instance;  
 }  
 **public function** setPriceShow($price){  
 **return** $this->**strategy**->countPrice($price);  
 }  
}  
*//客户端操作*$client=**new** StrategyA();  
$setprice=**new** setPrice($client);  
$data=$setprice->setPriceShow(100);  
**echo** $data;

我们之前在选择出行方式的时候，往往会使用if-else语句，也就是用户不选择A那么就选择B这样的一种情况。这种情况耦合性太高了，而且代码臃肿，有了策略模式我们就可以避免这种现象，策略模式遵循开闭原则，实现代码的解耦合。扩展新的方法时也比较方便，只需要继承策略接口就好了上面列出的这两点算是策略模式的优点了，但是不是说他就是完美的，有很多缺点仍然需要我们去掌握和理解，

客户端必须知道所有的策略类，并自行决定使用哪一个策略类。策略模式会出现很多的策略类。