

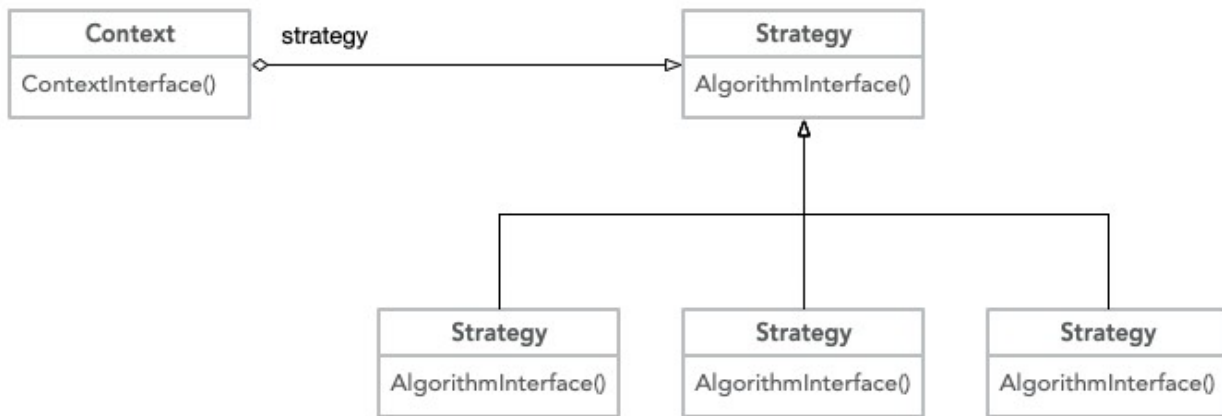
PHP设计模式之策略模式

策略模式，又称为政策模式，属于行为型的设计模式。

GoF类图及解释

GoF定义：定义一系列的算法，把它们一个个封装起来，并且使它们可以相互替换。本模式使得算法可独立于使用它的客户而变化。

GoF类图



代码实现

```
1 interface Strategy{
2     function AlgorithmInterface();
3 }
4
5 class ConcreteStrategyA implements Strategy{
6     function AlgorithmInterface(){
7         echo "算法A";
8     }
9 }
10
11 class ConcreteStrategyB implements Strategy{
12     function AlgorithmInterface(){
13         echo "算法B";
14     }
15 }
16
17 class ConcreteStrategyC implements Strategy{
18     function AlgorithmInterface(){
```

```
19         echo "算法C";
20     }
21 }
```

定义算法抽象及实现。

```
1 class Context{
2     private $strategy;
3     function __construct(Strategy $s){
4         $this->strategy = $s;
5     }
6     function ContextInterface(){
7
8         $this->strategy->AlgorithmInterface();
9     }
10 }
```

定义执行环境上下文。

```
1 $strategyA = new ConcreteStrategyA();
2 $context = new Context($strategyA);
3 $context->ContextInterface();
4
5 $strategyB = new ConcreteStrategyB();
6 $context = new Context($strategyB);
7 $context->ContextInterface();
8
9 $strategyC = new ConcreteStrategyC();
10 $context = new Context($strategyC);
11 $context->ContextInterface();
```

最后，在客户端按需调用合适的算法。

- 是不是非常简单的一个设计模式。大家有没有发现这个模式和我们最早讲过的**简单工厂**非常类似
- 那么他们的区别呢？
- 工厂相关的模式属于创建型模式，顾名思义，这种模式是用来创建对象的，返回的是new出来的对象。要调用对象的什么方法是由客户端来决定的

- 而策略模式属性行为型模式，通过执行上下文，将要调用的函数方法封装了起来，客户端只需要调用执行上下文的方法就可以了
- 在这里，我们会发现，需要客户端来实例化具体的算法类，貌似还不如**简单工厂**好用，既然这样的话，大家何不尝试一下结合工厂和策略模式一起来实现一个模式呢？
- 作为思考题将这个实现留给大家，提示：将Context类的__construct变成一个简单工厂方法

既然和简单工厂如此的相像，那么我们也按照简单工厂的方式来说：我们是一个手机厂商

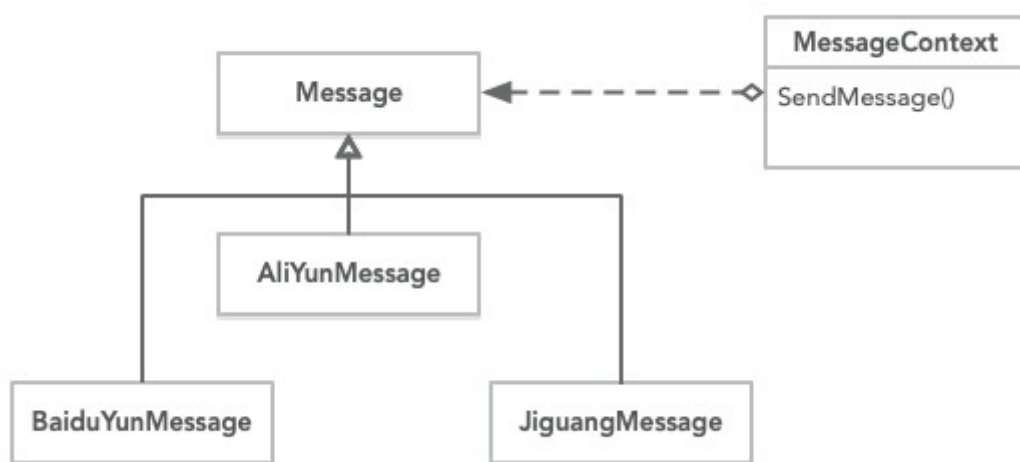
(Client)，想找某工厂 (ConcreteStrategy) 来做一批手机，通过渠道商 (Context) 向这个工厂下单制造手机，渠道商直接去联系代工厂 (Strategy)，并且直接将生产完成的手机发货给我 (ContextInterface())。同样的，我不用关心他们的具体实现，我只要监督那个和我们联系的渠道商就可以啦，是不是很省心！

完整代码：<https://github.com/zhangyue0503/designpatterns-php/blob/master/10.strategy/source/strategy.php>

实例

依然还是短信功能，具体的需求可以参看**简单工厂**模式中的讲解，但是这回我们使用策略模式来实现！

短信发送类图



完整源码：<https://github.com/zhangyue0503/designpatterns-php/blob/master/10.strategy/source/strategy-message.php>

```
1 <?php
2
3 interface Message
4 {
5     public function send();
6 }
7
```

```
8 class BaiduYunMessage implements Message
9 {
10     function send()
11     {
12         echo '百度云发送信息!';
13     }
14 }
15
16 class AliYunMessage implements Message
17 {
18     public function send()
19     {
20         echo '阿里云发送信息!';
21     }
22 }
23
24 class JiguangMessage implements Message
25 {
26     public function send()
27     {
28         echo '极光发送信息!';
29     }
30 }
31
32 class MessageContext
33 {
34     private $message;
35     public function __construct(Message $msg)
36     {
37         $this->message = $msg;
38     }
39     public function SendMessage()
40     {
41         $this->message->send();
42     }
43 }
44
45 $bdMsg = new BaiduYunMessage();
46 $msgCtx = new MessageContext($bdMsg);
47 $msgCtx->SendMessage();
```

```
48
49 $alMsg = new AliYunMessage();
50 $msgCtx = new MessageContext($alMsg);
51 $msgCtx->SendMessage();
52
53 $jgMsg = new JiguangMessage();
54 $msgCtx = new MessageContext($jgMsg);
55 $msgCtx->SendMessage();
```

说明

- 注意对比下类图，基本和**简单工厂**模式没什么区别
- 策略模式定义的是算法，从概念上看，这些算法完成的都是相同的工作，只是实现不同，但东西是死的，人是活的，具体想怎么用，还不是看大家的兴趣咯
- 策略模式可以优化单元测试，因为每个算法都有自己的类，所以可以通过自己的接口单独测试