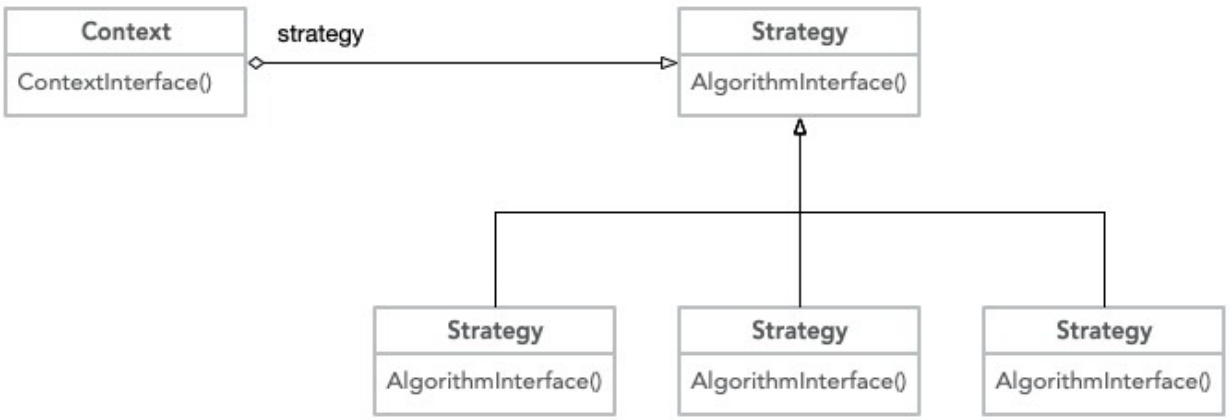
PHP设计模式之策略模式

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

```php
interface Strategy{
    function AlgorithmInterface();
}

class ConcreteStrategyA implements Strategy{
    function AlgorithmInterface(){
        echo "算法A";
    }
}

class ConcreteStrategyB implements Strategy{
    function AlgorithmInterface(){
        echo "算法B";
    }
}

class ConcreteStrategyC implements Strategy{
    function AlgorithmInterface(){
```

```
        echo "算法C";
    }
}
```

定义算法抽象及实现。

```
class Context{
    private $strategy;
    function __construct(Strategy $s){
        $this->strategy = $s;
    }
    function ContextInterface(){

        $this->strategy->AlgorithmInterface();
    }
}
```

定义执行环境上下文。

```
$strategyA = new ConcreteStrategyA();
$context = new Context($strategyA);
$context->ContextInterface();

$strategyB = new ConcreteStrategyB();
$context = new Context($strategyB);
$context->ContextInterface();

$strategyC = new ConcreteStrategyC();
$context = new Context($strategyC);
$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

```php
<?php

interface Message
{
    public function send();
}

class BaiduYunMessage implements Message
{
    function send()
    {
        echo '百度云发送信息！';
    }
}

class AliYunMessage implements Message
{
```

```php
    public function send()
    {
        echo '阿里云发送信息! ';
    }
}

class JiguangMessage implements Message
{
    public function send()
    {
        echo '极光发送信息! ';
    }
}

class MessageContext
{
    private $message;
    public function __construct(Message $msg)
    {
        $this->message = $msg;
    }
    public function SendMessage()
    {
        $this->message->send();
    }
}

$bdMsg = new BaiduYunMessage();
$msgCtx = new MessageContext($bdMsg);
$msgCtx->SendMessage();

$alMsg = new AliYunMessage();
$msgCtx = new MessageContext($alMsg);
$msgCtx->SendMessage();

$jgMsg = new JiguangMessage();
$msgCtx = new MessageContext($jgMsg);
$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