PHP设计模式之装饰器模式

工厂模式告一段落，我们来研究其他一些模式。不知道各位大佬有没有尝试过女装？据说女装大佬程序员很多哟。其实，今天的装饰器模式就和化妆这件事很像。相信如果有程序媛MM在的话，马上就能和你讲清楚这个设计模式。

Gof类图及解释

装饰这两个字，我们暂且把他变成化妆。首先你得有一张脸，然后打底，然后上妆，可以早上来个淡妆上班，也可以下班的时候补成浓妆出去嗨。当然，码农们下班的时间点正好是能赶上夜场的下半场的。话说回来，不管怎么化妆，你的脸还是你的脸，有可能可以化成别人不认识的另一个人，但这的的确确还是你的脸。这就是装饰器，对对象（脸）进行各种装饰（化妆），让这个脸更好看（增加职责）。

GoF定义：动态地给一个对象添加一些额外的职责，就增加功能来说，Decorator模式相比生成子类更为灵活

> GoF类图

> 代码实现

```php
interface Component{
    public function operation();
}

class ConcreteComponent implements Component{
    public function operation(){
        echo "I'm face!" . PHP_EOL;
    }
}
```

很简单的一个接口和一个实现，这里我们就把具体的实现类看作是一张脸吧！

```php
abstract class Decorator implements Component{
    protected $component;
    public function __construct(Component $component){
        $this->component = $component;
    }
```

```
}
```

抽象的装饰者类，实现Component接口，但并不实现operation()方法，让子类去实现。在这里主要保存一个Componet的引用，一会就要对他进行装饰。对应到上方的具体类，我们就是要准备给脸化妆啦！

```
class ConcreteDecoratorA extends Decorator{
    public $addedState = 1; // 没什么实际意义的属性，只是区别于ConcreteDecoratorB

    public function operation(){
        echo $this->component->operation() . "Push " . $this->addedState . " cream! " . PHP_EOL;
    }
}
class ConcreteDecoratorB extends Decorator{
    public function operation(){
        $this->component->operation();
        $this->addedBehavior();
    }

    // 没什么实际意义的方法，只是区别于ConcreteDecoratorA
    public function addedBehavior(){
        echo "Push 2 cream! " . PHP_EOL;
    }
}
```

两个具体装饰者。在这里我是涂了两次霜，毕竟是纯爷们，对化妆这事儿真的是不了解。好像第一步应该先是打粉底吧？不过这次就这样，我们这两个装饰器实现的就是给脸上涂两层霜。

- 从代码中可以看出，我们是一直对具体的那个ConcreteComponent对象来进行包装
- 再往下的话其实我们是对他的operation()这个方法包装了两次，每次都是在前一次的基础上加了一点点东西
- 不要纠结于A和B装饰器上的added属性和方法，他们只是GoF类图中用以区别这两个装饰器不是同一个东西，每个装饰器都可以干很多别的事，Component对象也不一定只有operation()这一个方法，我们可以选择性的去装饰对象中的全部或者部分方法
- 好像我们都继承了Component，直接子类一路重写不就行了，搞这费劲干嘛？亲，了解下组合的概念哟，我们的Decorator父类里面是一个真实对象的引用哦，解耦了自身哦，我们只给真实的对象

去做包装，您可别直接实例化装饰器来直接用

- 还是没懂？好处呢？老系统的类啊、方法啊你敢随便乱改？想给前任写的牛(S)逼(B)代码扩展新功能时不妨试试装饰器这货，说不定有奇效！

手机这玩意干不过某米、某O、某为，这没法玩呀，好吧，哥们去专心做手机壳吧！嗯，我先准备了一个透明壳（Component），貌似有点丑，没办法，谁叫哥们穷。给某米的加上各种纯色（DecoratorA1），然后背后印上各种颜色的植物（DecoratorB1）吧；某O的手机最近喜欢找流量明显做代言，那我给他的手机壳就用各种炫彩色（DecoratorA2）和明星的卡通头像（DecoratorB2）；最后的某为，好像手机已经开始引领业界潮流了，折叠屏这玩意不是要砸我这卖手机壳的生意嘛！！好吧，哥不给你们做了，还是跟我的某米、某O混去吧！！

完整代码：装饰器模式

实例

继续来发短信，之前我们用工厂模式解决了多个短信运营商的问题。这回我们要解决的是短信内容模板的问题。对于推广类的短信来说，根据最新的广告法，我们是不能出现“全国第一”、“全世界第一”这类的词语的，当然，一些不太文明的用语我们也是不能使用的。

现在的情况是这样的，我们有一个很早之前的短信模板类，里面的内容是固定的，老系统依然还是使用这个模板，老系统是面对的内部员工，对语言内容的要求不高。而新系统则需要向全网发送，也就是内外部的用户都要发送。这时，我们可以用装饰器模式来对老系统的短信模板进行包装。其实说简单点，我们就是用装饰器来做文本替换的功能。好处呢？当然是可以不去改动原来的模板类中的方法就实现了对老模板内容的修改扩展等。

> 短信发送类图

完整源码：短信发送装饰器方法

```php
<?php
// 短信模板接口
interface MessageTemplate{
    public function message();
}

// 假设有很多模板实现了上面的短信模板接口
// 下面这个是其中一个优惠券发送的模板实现
class CouponMessageTemplate implements MessageTemplate{
    public function message()    {
        return '优惠券信息：我们是全国第一的牛X产品哦，送您十张优惠券！';
    }
}
```

```php

// 我们来准备好装饰上面那个过时的短信模板
abstract class DecoratorMessageTemplate implements MessageTemplate{
    public $template;
    public function __construct($template)    {
        $this->template = $template;
    }
}

// 过滤新广告法中不允许出现的词汇
class AdFilterDecoratorMessage extends DecoratorMessageTemplate{
    public function message()    {
        return str_replace('全国第一', '全国第二', $this->template->message());
    }
}

// 使用我们的大数据部门同事自动生成的新词库来过滤敏感词汇，这块过滤不是强制要过滤的内容，可选择使用
class SensitiveFilterDecoratorMessage extends DecoratorMessageTemplate{
    public $bigDataFilterWords = ['牛X'];
    public $bigDataReplaceWords = ['好用'];
    public function message()    {
        return str_replace($this->bigDataFilterWords, $this->bigDataReplaceWords, $this->template->message());
    }
}

// 客户端，发送接口，需要使用模板来进行短信发送
class Message{
    public $msgType = 'old';
    public function send(MessageTemplate $mt)    {
        // 发送出去咯
        if ($this->msgType == 'old') {
            echo '面向内网用户发送' . $mt->message() . PHP_EOL;
        } else if ($this->msgType == 'new') {
            echo '面向全网用户发送' . $mt->message() . PHP_EOL;
        }

    }
}
```

```php

$template = new CouponMessageTemplate();
$message = new Message();

// 老系统，用不着过滤，只有内部用户才看得到
$message->send($template);

// 新系统，面向全网发布的，需要过滤一下内容哦
$message->msgType = 'new';
$template = new AdFilterDecoratorMessage($template);
$template = new SensitiveFilterDecoratorMessage($template);

// 过滤完了，发送吧
$message->send($template);

```

> 说明

- 装饰器的最大好处：一是不改变原有代码的情况下对原有代码中的内容进行扩展，开放封闭原则；二是每个装饰器完成自己的功能，单一职责；三是用组合实现了继承的感觉；
- 最适用于：给老系统进行扩展
- 要小心：过多的装饰者会把你搞晕的
- 不一定都是对同一个方法进行装饰，其实装饰者应该更多的用于对对象的装饰，对对象进行扩展，这里我们都是针对一个方法的输出进行装饰，但仅限此文，装饰器的应用其实更加广泛
- 装饰器的特点是全部都继承自一个主接口或类，这样的好处就是返回的对象是相同的抽象数据，具有相同的行为属性，否则，就不是装饰之前的对象，而是一个新对象了
- 有点不好理解没关系，我们这次的例子其实也很勉强，这个设计模式在《Head First设计模式》中有提到Java的I/O系列接口是使用的这种设计模式：FileInputStream、LineNumberInputStream、BufferInputStream等
- Laravel框架中的中间件管道，这里其实是多种模式的综合应用，其中也应用到了装饰器模式：Laravel HTTP——Pipeline 中间件装饰者模式源码分析
- 另外在Laravel中，日志处理这里也是对Monolog进行了装饰，有兴趣的同学可以去了解下