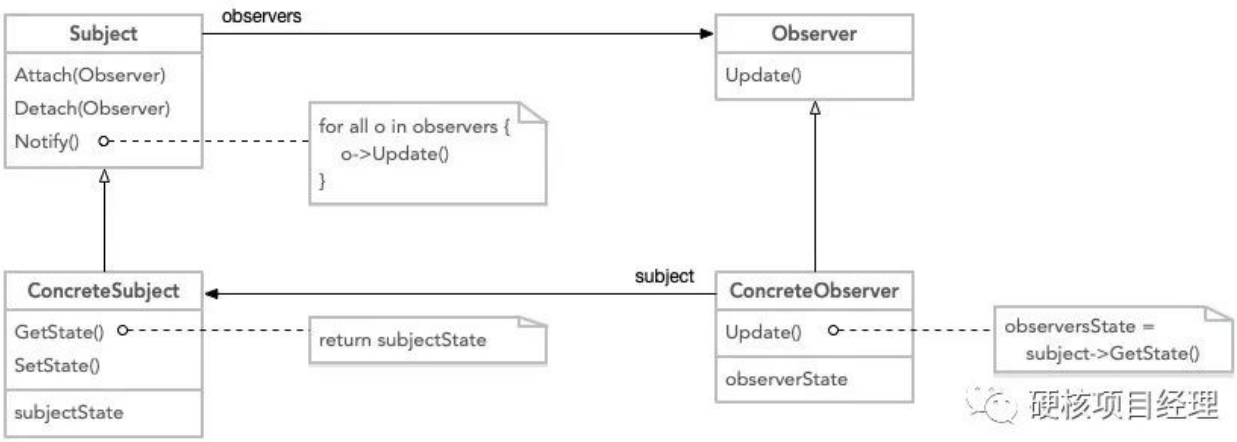
PHP设计模式之观察者模式

观察者，貌似在很多科幻作品中都会有这个角色的出现。比如我很喜欢的一部美剧《危机边缘》，在这个剧集中，观察者不停的穿越时空记录着各种各样的人或事。但是，设计模式中的观察者可不只是站在边上看哦，这里的观察者是针对主体发生的状态改变来做出对应的动作。

Gof类图及解释

GoF定义：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新

> GoF类图



> 代码实现

```php
interface Observer{
    public function update(Subject $subject): void;
}
```

观察者的抽象接口，没啥可说的吧，就是让你实现一个具体的Update就可以了

```php
class ConcreteObserver implements Observer{
    private $observerState = '';
    function update(Subject $subject): void    {
        $this->observerState = $subject->getState();
        echo '执行观察者操作！当前状态：' . $this->observerState;
    }
}
```

具体的观察者，实现update()方法，这里我们拿到了Subject类，从而可以获得其中的状态

```php
class Subject{
    private $observers = [];
    private $stateNow = '';
    public function attach(Observer $observer): void    {
        array_push($this->observers, $observer);
    }
    public function detach(Observer $observer): void    {
        $position = 0;
        foreach ($this->observers as $ob) {
            if ($ob == $observer) {
                array_splice($this->observers, ($position), 1);
            }
            ++$position;
        }
    }
    public function notify(): void    {
        foreach ($this->observers as $ob) {
            $ob->update($this);
        }
    }
}
```

Subject父类，维护一个观察者数组，然后有添加、删除以及循环遍历这个数组的方法，目的是能够方便的管理所有的观察者

```php
class ConcreteSubject extends Subject{
    public function setState($state)    {
        $this->stateNow = $state;
        $this->notify();
    }

    public function getState()    {
        return $this->stateNow;
    }
}
```

Subject的实现类，只是更新了状态，在这个状态发生改变的时候，调用观察者遍历的方法进行所有观察的update()操作

- 观察者，其实就是自身做了一个更新（update），而Subject，可以批量的执行观察者，请注意，我们不需要去修改目标类中的任何代码，只需要从外部添加就可以了，所以就让目标和观察者解耦互相之间不用关心对方的情况了
- 观察者可以记录目标的状态，也可以不用记录，比如我们发完短信后的数据库更新或者插入操作，只有短信接口发送成功后我们再修改短信数据的状态就可以了，不一定完全需要将目标的发送状态传送给观察者
- 当一个类在发生改变时，不知道可能会对其他多少类产生影响，这个时候观察者非常有用
- 观察者模式中还是存在着耦合，那就是目标类中有一个观察者对象列表，如果观察者没有实现update()方法，那么就会出现问题

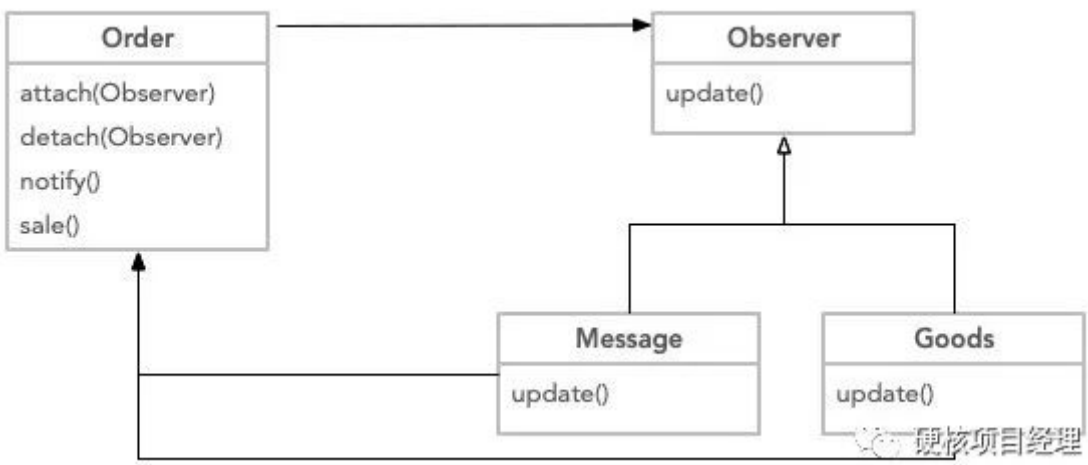
接着拿我们的手机工厂说事儿，这次好嘛，被一帮山寨机盯上了（观察者），我出什么功能（状态更新），他们就对应的出一样的功能（更新），而且还在我的基础上做了更多的东西，美其名曰：微创新！你说气人不气人。好吧，我也派出了一帮市场调查人员（观察者），去帮我观察别人家的手机都出了什么功能（状态更新），然后我们也照搬过来搞点微创新，大家共同进步嘛！！

完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/06.observer/source/observer.php

实例

这次我们从订单说起，不过还是有短信发送的事儿。当一般的电商平台有人下单之后，需要做的事情非常多，比如修改库存、发送短信或者推送告诉商家有人下单了，告诉买家下单成功了，支付成功了。总之就是一件事情的发生会导致各种事件的产生。其实，这里就引出了另一个非常出名的模式**订阅发布**模式。这个模式可以说是观察者的升级模式，这个系列的文章不会细讲，但是大家可以去看看Laravel中的**发布订阅**及**事件监听**方面的内容。

> 订单售出类图



完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/06.observer/source/order-observer.php

```php
interface Observer{
    public function update($obj);
}

class Message implements Observer{
    //....

    function update($obj)    {
        echo '发送新订单短信(' . $obj->mobile . ')通知给商家! ';
    }

    //....
}

class Goods implements Observer{
    //....

    public function update($obj)    {
        echo '修改商品' . $obj->goodsId . '的库存! ';
    }

    //....
}

class Order{
    private $observers = [];
    public function attach($ob)    {
        $this->observers[] = $ob;
    }

    public function detach($ob)    {
        $position = 0;
        foreach ($this->observers as $ob) {
            if ($ob == $observer) {
                array_splice($this->observers, ($position), 1);
```

```
            }
            ++$position;
        }
    }
    public function notify($obj)    {
        foreach ($this->observers as $ob) {
            $ob->update($obj);
        }
    }
    public function sale()    {
        // 商品卖掉了
        // ....
        $obj = new stdClass();
        $obj->mobile = '13888888888';
        $obj->goodsId = 'Order11111111';
        $this->notify($obj);
    }
}

$message = new Message();
$goods = new Goods();
$order = new Order();
$order->attach($message);
$order->attach($goods);

// 订单卖出了！！
$order->sale();
```

> 说明

- 我们没有完全的遵守GoF类图，虽说GoF是圣经，但也并不是我们必须要完全遵守的，我们可以针对具体的业务情况进行合适的裁剪使用
- 订单状态通过sale()方法产生变化后，直接调用notify方法进行观察者的调用
- 发短信、发推送都可以拆开由一个一个的观察者来实现，这些观察者不一定只有这一个方法，但只要实现共同的接口就可以了
- 商品库存和消息发送其实就是两个本身完全不沾边的类，但它们只需要实现一样的接口就好啦
- PHP的SPL扩展中已经为我们准备好了一套观察者接口，大家可以试试哦，使用原生支持的观察者模式能省不少事儿呢！

SPL扩展实现观察者模式-完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/06.observer/source/spl_observer.php