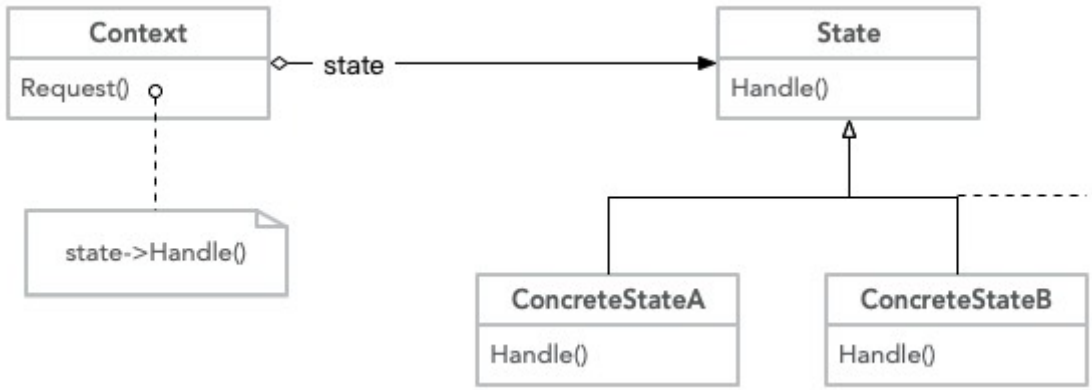
PHP设计模式之状态模式

状态模式从字面上其实并不是很好理解。这里的状态是什么意思呢？保存状态？那不就是备忘录模式了。其实，这里的状态是类的状态，通过改变类的某个状态，让这个类感觉像是换了一个类一样。说起来有点拗口吧，先学习概念之后再看。

Gof类图及解释

GoF定义：允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它的类

> GoF类图



> 代码实现

```php
class Context
{
    private $state;
    public function SetState(State $state): void
    {
        $this->state = $state;
    }
    public function Request(): void
    {
        $this->state = $this->state->Handle();
    }
}
```

一个上下文类，也可以看作是目标类，它的内部有一个状态对象。当调用Request()的时候，去调用状态类的Handle()方法。目的是当前上下文类状态的变化都由外部的这个状态类来进行操纵。

```php
interface State
{
    public function Handle(): State;
}

class ConcreteStateA implements State
{
    public function Handle(): State
    {
        echo '当前是A状态', PHP_EOL;
        return new ConcreteStateB();
    }
}

class ConcreteStateB implements State
{
    public function Handle(): State
    {
        echo '当前是B状态', PHP_EOL;
        return new ConcreteStateA();
    }
}
```

抽象状态接口及两个具体实现。这两个具体实现实际上是在相互调用。实现的效果就是上下文类每调用一次Request()方法，内部的状态类就变成别一个状态。就像一个开关，在打开与关闭中来回切换一样。

```php
$c = new Context();
$stateA = new ConcreteStateA();
$c->SetState($stateA);
$c->Request();
$c->Request();
$c->Request();
$c->Request();
```

客户端的实现，实例化上下文对象并设置初始的状态，然后通过不停的调用Request()对象来实现开关状态的切换。

- 看出门道了嘛？这里把状态的变化给封装到外部的实现类去了，并不是这个上下文或者目标类内部来进行状态的切换了
- 那么状态模式的意义呢？这个默认类图的例子过于简单，其实状态模式的真正目的是为了解决复杂的if嵌套问题的，把复杂的if嵌套条件放到一个个的外部状态类中去判断，在后面的实例中我们会看到
- 适用于：一个对象的行为取决于它的状态，并且它的必须在运行时刻根据状态改变自己的行为；一个操作中含有大量的多分支条件语句，且这些分支依赖于该对象的状态；
- 状态模式的特点是：它将与特定状态相关的行为局部化；它使得状态转换显式化；State对象可以被共享；
- 常见于订单系统、会员系统、OA系统中，也就是流程中会出现各种状态变化的情况，都可以使用状态模式来进行整体的设计与架构

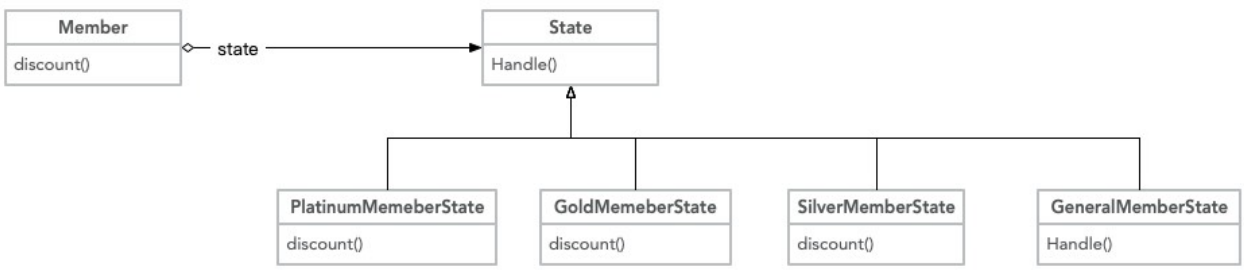
我们的手机系统内定制了自己的商城系统，可以在手机上方便的下单购买我们的商品。一个订单（Context）会有多种状态（State），比如未支付、已支付、订单完成、订单退款等等一大堆状态。我们把这些状态都放在了对应的状态类里去实现，不同的状态类都会再去调用该状态下一步的动作，比如已支付后就等待收货、退款后就等待买家填写物流单号等，这样，状态模式就在我们的商城中被灵活的运用起来咯！！

完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/22.state/source/state.php

实例

通常的商城应用中都会有会员体系的存在，一般等级越高的会员可以享受的折扣也会越多，这个时候，运用状态模式就能很轻松的获得会员的等级折扣。当然，最主要的是，使用状态模式可以在需要添加或者删除会员等级时只添加对应的会员折扣状态子类就可以了。其他业务代码都不需要变动，我们一起来看看具体实现吧！

> 会员折扣图



完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/22.state/source/state-member.php

```
1 <?php
2
```

```php
class Member
{
    private $state;
    private $score;

    public function SetState($state)
    {
        $this->state = $state;
    }

    public function SetScore($score)
    {
        $this->score = $score;
    }

    public function GetScore()
    {
        return $this->score;
    }

    public function discount()
    {
        return $this->state->discount($this);
    }
}

interface State
{
    public function discount($member);
}

class PlatinumMemeberState implements State
{
    public function discount($member)
    {
        if ($member->GetScore() >= 1000) {
            return 0.80;
        } else {
            $member->SetState(new GoldMemberState());
            return $member->discount();
```

```
        }
    }
}

class GoldMemberState implements State
{
    public function discount($member)
    {
        if ($member->GetScore() >= 800) {
            return 0.85;
        } else {
            $member->SetState(new SilverMemberState());
            return $member->discount();
        }
    }
}

class SilverMemberState implements State
{
    public function discount($member)
    {
        if ($member->GetScore() >= 500) {
            return 0.90;
        } else {
            $member->SetState(new GeneralMemberState());
            return $member->discount();
        }
    }
}

class GeneralMemberState implements State
{
    public function discount($member)
    {
        return 0.95;
    }
}

$m = new Member();
```

```php
$m->SetState(new PlatinumMemeberState());

$m->SetScore(1200);
echo '当前会员' . $m->GetScore() . '积分，折扣为：' . $m->discount(), PHP_EOL;

$m->SetScore(990);
echo '当前会员' . $m->GetScore() . '积分，折扣为：' . $m->discount(), PHP_EOL;

$m->SetScore(660);
echo '当前会员' . $m->GetScore() . '积分，折扣为：' . $m->discount(), PHP_EOL;

$m->SetScore(10);
echo '当前会员' . $m->GetScore() . '积分，折扣为：' . $m->discount(), PHP_EOL;
```

> 说明

- 如果不使用状态模式，在Member的discount()方法中，我们可能需要写很多层if...else...判断条件
- 同时，这也带来了方法体会越来越长，越来越难以维护的问题
- 状态模式正是为了解决这个问题而存在的
- 当discount()行为的结果依赖于Member对象本身的状态（会员分）时，状态模式就是最佳的选择了，也就是上面所说的一个对象的行为取决于它的状态