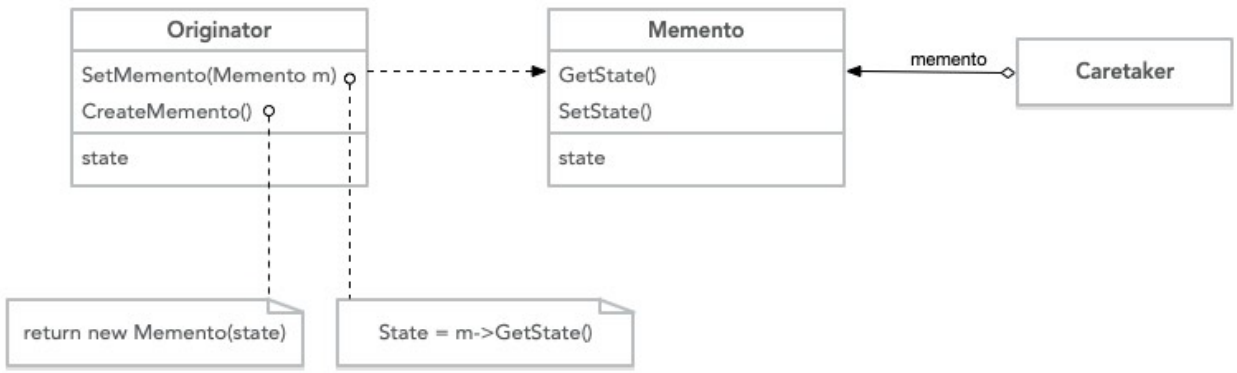
PHP设计模式之备忘录模式

备忘录，这个名字其实就已经很形象的解释了它的作用。典型的例子就是我们原来玩硬盘游戏时的存档功能。当你对即将面对的大BOSS有所顾虑时，一般都会先保存一次进度存档。如果挑战失败了，直接读取存档就可以恢复到挑战BOSS前的状态，然后你就开开心心的再去练一会级回来解决这个大BOSS就好了。不过，为了以防万一，在挑战BOSS之前存个档总是好的。另外一个例子就是我们码农们天天要用到的代码管理工具Git或者Svn了。每次的提交都像是一次存档备份，当新代码出现问题的时候，直接回滚恢复就行了。这些，都是备忘录模式的典型应用，下面就一起来看看这个模式吧。

Gof类图及解释

GoF定义：在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到原先保存的状态

> GoF类图



> 代码实现

```php
class Originator
{
    private $state;
    public function SetMeneto(Memento $m)
    {
        $this->state = $m->GetState();
    }
    public function CreateMemento()
    {
        $m = new Memento();
        $m->SetState($this->state);
        return $m;
    }

```

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

    public function ShowState()
    {
        echo $this->state, PHP_EOL;
    }
}
```

原发器，也可以叫做发起人。它有一个内部状态（state），这个状态可以在不同的情况下进行改变。当某一个事件发生时，需要将这个状态恢复到原先的状态。在这里，我们有一个CreateMemento()用于创建一个备忘录（存档），有一个SetMeneto()用于还原状态（读档）。

```
class Memento
{
    private $state;
    public function SetState($state)
    {
        $this->state = $state;
    }
    public function GetState()
    {
        return $this->state;
    }
}
```

备忘录，非常简单，就是用于记录状态。将这个状态以对象的形式保存，就可以让原发器非常方便地创建很多存档用于记录各种不同的状态。

```
class Caretaker
{
    private $memento;
    public function SetMemento($memento)
    {
        $this->memento = $memento;
```

```
    }
    public function GetMemento()
    {
        return $this->memento;
    }
}
```

负责人，也叫做管理者类，保存备忘录，当需要的时候从这里取出备忘录。它只负责保存，不能修改备忘录。在复杂的应用中，可以将这里做成列表，就像游戏中可以选择性的展现多条存档记录供玩家选择。

```
$o = new Originator();
$o->SetState('状态1');
$o->ShowState();

// 保存状态
$c = new Caretaker();
$c->SetMemento($o->CreateMemento());

$o->SetState('状态2');
$o->ShowState();

// 还原状态
$o->SetMeneto($c->GetMemento());
$o->ShowState();
```

客户端的调用中，我们的原发器初始化状态后进行了保存，然后人为的更改了状态。这时只需要通过负责人将状态还原回来就可以了。

- 备忘录模式说白了就是让一个外部类B来保存A的内部状态，然后在适当的时候可以方便的还原这个状态。
- 备忘录模式的应用场景其实非常多，浏览器的回退、数据库的备份还原、操作系统的备份还原、文档的撤销重做、棋牌游戏的悔棋等等
- 这个模式能够保持对原发器的封装，也就是这些状态需要对外部的对象隐藏，所以只能交给一个备忘录对象来记录
- 状态在原发器和备忘录之间的拷贝可能带来性能问题，特别是大型对象的复杂繁多的内部状态，而且也会带来一些编码方面的漏洞，比如漏掉某些状态

Mac的时光机功能大家有了解过吧，可以将电脑恢复到某一时间点的状态下。其实windows的ghost也是类似的功能。我们的手机操作系统上也决定开发这样的一个功能。当我们点击时光机备份时，将手机上所有的资料、数据、状态信息都压缩保存起来，如果用户允许的话，我们将这个压缩包上传到我们的云服务器上避免占用用户的手机内存，否则就只能保存到用户的手机内存中了。当用户的手机需要恢复到某个时间点，我们将所有的时光机备份列出，用户只需要用手指轻轻一按就可以把手机系统状态恢复到当时的样子了，是不是非常方便！！

完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/17.memento/source/memento.php

实例

这次又回到短信发送的例子上来。通常我们做短信或者邮件发送这些功能时，会有一个队列从数据库或者缓存中读取要发送的内容进行发送，如果成功了就不管了，如果失败了会将短信的状态改成失败或者重发。在这里，我们直接将它改回到之前未发送的状态然后等待下次发送的队列再次执行发送。

> 短信发送类图

完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/17.memento/source/memento-message.php

```php
<?php
class Message
{
    private $content;
    private $to;
    private $state;
    private $time;

    public function __construct($to, $content)
    {
        $this->to = $to;
        $this->content = $content;
        $this->state = '未发送';
        $this->time = time();
    }

    public function Show()
    {
        echo $this->to, '---', $this->content, '---', $this->time, '---', $this->state, PHP_EOL;
```

```php
    }

    public function CreateSaveSate()
    {
        $ss = new SaveState();
        $ss->SetState($this->state);
        return $ss;
    }

    public function SetSaveState($ss)
    {
        if ($this->state != $ss->GetState()) {
            $this->time = time();
        }
        $this->state = $ss->GetState();
    }

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

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

}

class SaveState
{
    private $state;
    public function SetState($state)
    {
        $this->state = $state;
    }
    public function GetState()
    {
        return $this->state;
    }
```

```php
}

class StateContainer
{
    private $ss;
    public function SetSaveState($ss)
    {
        $this->ss = $ss;
    }
    public function GetSaveState()
    {
        return $this->ss;
    }
}

// 模拟短信发送
$mList = [];
$scList = [];
for ($i = 0; $i < 10; $i++) {
    $m = new Message('手机号' . $i, '内容' . $i);
    echo '初始状态：';
    $m->Show();

    // 保存初始信息
    $sc = new StateContainer();
    $sc->SetSaveState($m->CreateSaveSate());
    $scList[] = $sc;

    // 模拟短信发送，2发送成功，3发送失败
    $pushState = mt_rand(2, 3);
    $m->SetState($pushState == 2 ? '发送成功' : '发送失败');
    echo '发布后状态：';
    $m->Show();

    $mList[] = $m;
}

// 模拟另一个线程查找发送失败的并把它们还原到未发送状态
sleep(2);
```

```php
foreach ($mList as $k => $m) {
    if ($m->GetState() == '发送失败') { // 如果是发送失败的，还原状态
        $m->SetSaveState($scList[$k]->GetSaveState());
    }
    echo '查询发布失败后状态：';
    $m->Show();
}
```

> 说明

- 短信类做为我们的原发器，在发送前就保存了当前的发送状态
- 随机模拟短信发送，只有两个状态，发送成功或者失败，并改变原发器的状态为成功或者失败
- 模拟另一个线程或者脚本对短信的发送状态进行检查，如果发现有失败的，就将它重新改回未发送的状态
- 这里我们只是保存了发送状态这一个字段，其他原发器的内部属性并没有保存
- 真实的场景下我们应该会有一个重试次数的限制，当超过这个次数后，状态改为彻底的发送失败，不再进行重试了