PHP设计模式之命令模式

命令模式，也称为动作或者事务模式，很多教材会用饭馆来举例。作为顾客的我们是命令的下达者，服务员是这个命令的接收者，菜单是这个实际的命令，而厨师是这个命令的执行者。那么，这个模式解决了什么呢？当你要修改菜单的时候，只需要和服务员说就好了，她会转达给厨师，也就是说，我们实现了顾客和厨师的解耦。也就是调用者与实现者的解耦。当然，很多设计模式可以做到这一点，但是命令模式能够做到的是让一个命令接收者实现多个命令（服务员下单、拿酒水、上菜），或者把一条命令转达给多个实现者（热菜厨师、凉菜厨师、主食师傅）。这才是命令模式真正发挥的地方！！

Gof类图及解释

GoF定义：将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可撤消的操作

> GoF类图

> 代码实现

```php
class Invoker
{
    public $command;

    public function __construct($command)
    {
        $this->command = $command;
    }

    public function exec()
    {
        $this->command->execute();
    }
}
```

首先我们定义一个命令的接收者，或者说是命令的请求者更恰当。类图中的英文定义这个单词是“祈求者”。也就是由它来发起和操作命令。

```php
abstract class Command
{
    protected $receiver;

    public function __construct(Receiver $receiver)
    {
        $this->receiver = $receiver;
    }

    abstract public function execute();
}

class ConcreteCommand extends Command
{
    public function execute()
    {
        $this->receiver->action();
    }
}
```

接下来是命令，也就是我们的“菜单”。这个命令的作用是为了定义真正的执行者是谁。

```php
class Receiver
{
    public $name;

    public function __construct($name)
    {
        $this->name = $name;
    }

    public function action()
    {
        echo $this->name . '命令执行了！', PHP_EOL;
    }
}
```

接管者，也就是执行者，真正去执行命令的人。

```php
// 准备执行者
$receiverA = new Receiver('A');

// 准备命令
$command = new ConcreteCommand($receiverA);

// 请求者
$invoker = new Invoker($command);
$invoker->exec();
```

客户端的调用，我们要联系好执行者也就是挑有好厨子的饭馆（Receiver），然后准备好命令也就是菜单（Command），最后交给服务员（Invoker）。

- 其实这个饭店的例子已经非常清晰了，对于命令模式真是完美的解析
- 那说好的可以下多份订单或者给多个厨师呢？别急，下面的代码帮助我们解决这个问题

完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/09.command/source/command.php

```php
<?php

class Invoker
{
    private $command = [];

    public function setCommand(Command $command)
    {
        $this->command[] = $command;
    }

    public function exec()
    {
        if(count($this->command) > 0){
            foreach ($this->command as $command) {
                $command->execute();
            }
        }
```

```php
    }

    public function undo()
    {
        if(count($this->command) > 0){
            foreach ($this->command as $command) {
                $command->undo();
            }
        }
    }
}

abstract class Command
{
    protected $receiver;
    protected $state;
    protected $name;

    public function __construct(Receiver $receiver, $name)
    {
        $this->receiver = $receiver;
        $this->name = $name;
    }

    abstract public function execute();
}

class ConcreteCommand extends Command
{
    public function execute()
    {
        if (!$this->state || $this->state == 2) {
            $this->receiver->action();
            $this->state = 1;
        } else {
            echo $this->name . '命令正在执行，无法再次执行了！', PHP_EOL;
        }

    }

```

```php
    public function undo()
    {
        if ($this->state == 1) {
            $this->receiver->undo();
            $this->state = 2;
        } else {
            echo $this->name . '命令未执行，无法撤销了！', PHP_EOL;
        }
    }
}

class Receiver
{
    public $name;
    public function __construct($name)
    {
        $this->name = $name;
    }
    public function action()
    {
        echo $this->name . '命令执行了！', PHP_EOL;
    }
    public function undo()
    {
        echo $this->name . '命令撤销了！', PHP_EOL;
    }
}

// 准备执行者
$receiverA = new Receiver('A');
$receiverB = new Receiver('B');
$receiverC = new Receiver('C');

// 准备命令
$commandOne = new ConcreteCommand($receiverA, 'A');
$commandTwo = new ConcreteCommand($receiverA, 'B');
$commandThree = new ConcreteCommand($receiverA, 'C');

// 请求者
```

```php
$invoker = new Invoker();
$invoker->setCommand($commandOne);
$invoker->setCommand($commandTwo);
$invoker->setCommand($commandThree);
$invoker->exec();
$invoker->undo();

// 新加一个单独的执行者，只执行一个命令
$invokerA = new Invoker();
$invokerA->setCommand($commandOne);
$invokerA->exec();

// 命令A已经执行了，再次执行全部的命令执行者，A命令的state判断无法生效
$invoker->exec();
```

- 这一次我们一次性解决了多个订单、多位厨师的问题，并且还顺便解决了如果下错命令了，进行撤销的问题
- 可以看出来，命令模式将调用操作的对象与知道如何实现该操作的对象实现了解耦
- 这种多命令多执行者的实现，有点像组合模式的实现
- 在这种情况下，增加新的命令，即不会影响执行者，也不会影响客户。当有新的客户需要新的命令时，只需要增加命令和请求者即可。即使有修改的需求，也只是修改请求者。
- Laravel框架的事件调度机制中，除了观察者模式外，也很明显的能看出命令模式的影子

我们的手机工厂和餐厅其实并没有什么两样，当我们需要代工厂来制作手机时，也是先下订单，这个订单就可以看做是命令。在这个订单中，我们会规定好需要用到的配件，什么型号的CPU，什么型号的内存，预装什么系统之类的。然后代工厂的工人们就会根据这个订单来进行生产。在这个过程中，我不用关心是某一个工人还是一群工人来执行这个订单，我只需要将这个订单交给和我们对接的人就可以了，然后只管等着手机生产出来进行验收咯！！

完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/09.command/source/command-up.php

实例

短信功能又回来了，我们发现除了工厂模式外，命令模式貌似也是一种不错的实现方式哦。在这里，我们依然是使用那几个短信和推送的接口，话不多说，我们用命令模式再来实现一个吧。当然，有兴趣的朋友可以接着实现我们的短信撤回功能哈，想想上面的命令取消是怎么实现的。

> 短信发送类图

完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/09.command/source/command-message.php

```php
<?php

class SendMsg
{
    private $command = [];

    public function setCommand(Command $command)
    {
        $this->command[] = $command;
    }

    public function send($msg)
    {
        foreach ($this->command as $command) {
            $command->execute($msg);
        }
    }
}

abstract class Command
{
    protected $receiver = [];

    public function setReceiver($receiver)
    {
        $this->receiver[] = $receiver;
    }

    abstract public function execute($msg);
}

class SendAliYun extends Command
{
    public function execute($msg)
    {
        foreach ($this->receiver as $receiver) {
            $receiver->action($msg);
        }
```

```
    }
}

class SendJiGuang extends Command
{
    public function execute($msg)
    {
        foreach ($this->receiver as $receiver) {
            $receiver->action($msg);
        }
    }
}

class SendAliYunMsg
{
    public function action($msg)
    {
        echo '【阿X云短信】发送：' . $msg, PHP_EOL;
    }
}

class SendAliYunPush
{
    public function action($msg)
    {
        echo '【阿X云推送】发送：' . $msg, PHP_EOL;
    }
}

class SendJiGuangMsg
{
    public function action($msg)
    {
        echo '【极X短信】发送：' . $msg, PHP_EOL;
    }
}

class SendJiGuangPush
{
    public function action($msg)
```

```
    {
        echo '【极X推送】发送：' . $msg, PHP_EOL;
    }
}

$aliMsg = new SendAliYunMsg();
$aliPush = new SendAliYunPush();
$jgMsg = new SendJiGuangMsg();
$jgPush = new SendJiGuangPush();

$sendAliYun = new SendAliYun();
$sendAliYun->setReceiver($aliMsg);
$sendAliYun->setReceiver($aliPush);

$sendJiGuang = new SendJiGuang();
$sendAliYun->setReceiver($jgMsg);
$sendAliYun->setReceiver($jgPush);

$sendMsg = new SendMsg();
$sendMsg->setCommand($sendAliYun);
$sendMsg->setCommand($sendJiGuang);

$sendMsg->send('这次要搞个大活动，快来注册吧！！');
```

> 说明

- 在这个例子中，依然是多命令多执行者的模式
- 可以将这个例子与抽象工厂进行对比，同样的功能使用不同的设计模式来实现，但是要注意的是，抽象工厂更多的是为了生产对象返回对象，而命令模式则是一种行为的选择
- 我们可以看出命令模式非常适合形成命令队列，多命令让命令可以一条一条执行下去
- 它允许接收的一方决定是否要否决请求，Receiver做为实现者拥有更多的话语权