

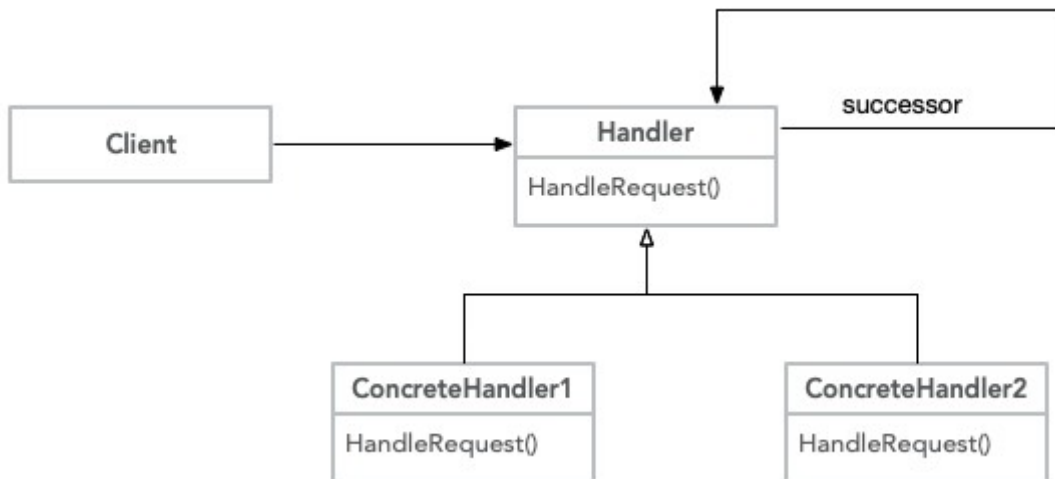
# PHP设计模式之责任链模式

责任链模式，属于对象行为型的设计模式。

## Gof类图及解释

**GoF定义：**使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系。将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它为止。

GoF类图



代码实现

```
1 abstract class Handler
2 {
3     protected $successor;
4     public function setSuccessor($successor)
5     {
6         $this->successor = $successor;
7     }
8     abstract public function HandleRequest($request);
9 }
```

定义抽象责任链类，使用\$successor保存后继链条。

```
1 class ConcreteHandler1 extends Handler
2 {
3     public function HandleRequest($request)
```

```

4      {
5          if (is_numeric($request)) {
6              return '请求参数是数字: ' . $request;
7          } else {
8              return $this->successor->HandleRequest($request);
9          }
10     }
11 }
12
13 class ConcreteHandler2 extends Handler
14 {
15     public function HandleRequest($request)
16     {
17         if (is_string($request)) {
18             return '请求参数是字符串: ' . $request;
19         } else {
20             return $this->successor->HandleRequest($request);
21         }
22     }
23 }
24
25 class ConcreteHandler3 extends Handler
26 {
27     public function HandleRequest($request)
28     {
29         return '我也不知道请求参数是啥了，你猜猜? ' . gettype($request);
30     }
31 }

```

三个责任链条的具体实现，主要功能是判断传入的数据类型，如果是数字由第一个类处理，如果是字符串，则第二个类处理。如果是其他类型，第三个类统一处理。

```

1 $handle1 = new ConcreteHandler1();
2 $handle2 = new ConcreteHandler2();
3 $handle3 = new ConcreteHandler3();
4
5 $handle1->setSuccessor($handle2);
6 $handle2->setSuccessor($handle3);

```

```

7
8 $requests = [22, 'aaa', 55, 'cc', [1, 2, 3], null, new stdClass];
9
10 foreach ($requests as $request) {
11     echo $handle1->HandleRequest($request) . PHP_EOL;
12 }

```

客户端的调用，依次实例化三个责任链实例，并指定链条成员。创建请求参数，之后通过责任链来进行结果判断。

- 责任链非常适合的一种场景，就是对请求参数进行逐层过滤，就像我们工作时使用钉钉之类的办公软件。当需要提加班或者休假申请时，那一层层审批流程就是这个模式最完美的解释
- 我们可以拦截请求，直接返回，也可以对请求内容进行完善修改交给下一个类来进行处理，但至少有一个类是要返回结果的。
- 请求不一定会被处理，也有可能完全不处理就返回或者传递给下一个处理类来进行处理

我们一直在说手机制造这个行业，之前我们一直是交给代工厂来进行手机的组装生产，这回，我们自己建立了一条流水线。而这个流水线，就非常像责任链模式，怎么说呢，从一台手机的装配说起。有操作员将手机主板（初始请求）放到流水线上，然后工人开始添加内存、CPU、摄像头（各种责任链条类进行处理），期间也会经过测试和调整以达到最佳出厂性能。最后拼装成一台完整的手机交到客户的手中，这种工作流是不是和责任链非常相似呢！！

完整代码：<https://github.com/zhangyue0503/designpatterns-php/blob/master/11.chain-of-responsiblity/source/chain.php>

## 实例

依然还是短信功能，但这次我们要实现的是一个短信内容过滤的子功能。大家都知道，我们对广告有着严格的规定，许多词都在广告法中被标记为禁止使用的词汇，更有些严重的词汇可能会引来不必要的麻烦。这时候，我们就需要一套过滤机制来进行词汇的过滤。针对不同类型的词汇，我们可以通过责任链来进行过滤，比如严重违法的词汇当然是这条信息都不能通过。一些比较严重但可以绕过的词，我们可以进行替换或者加星处理，这样，客户端不需要一大堆的if..else..来进行逻辑判断，使用责任链让他们一步步的进行审批就好啦！！

短信发送类图

完整源码：<https://github.com/zhangyue0503/designpatterns-php/blob/master/11.chain-of-responsiblity/source/chain-filter-message.php>

```

1 // 词汇过滤链条
2 abstract class FilterChain
3 {

```

```
4     protected $next;
5     public function setNext($next)
6     {
7         $this->next = $next;
8     }
9     abstract public function filter($message);
10 }
11
12 // 严辞词汇
13 class FilterStrict extends FilterChain
14 {
15     public function filter($message)
16     {
17         foreach (['枪X', '弹X', '毒X'] as $v) {
18             if (strpos($message, $v) !== false) {
19                 throw new \Exception('该信息包含敏感词汇! ');
20             }
21         }
22         if ($this->next) {
23             return $this->next->filter($message);
24         } else {
25             return $message;
26         }
27     }
28 }
29
30 // 警告词汇
31 class FilterWarning extends FilterChain
32 {
33     public function filter($message)
34     {
35         $message = str_replace(['打架', '丰胸', '偷税'], '*', $message);
36         if ($this->next) {
37             return $this->next->filter($message);
38         } else {
39             return $message;
40         }
41     }
42 }
43
```

```

44 // 手机号加星
45 class FilterMobile extends FilterChain
46 {
47     public function filter($message)
48     {
49         $message = preg_replace("/(1[3|5|7|8]\d)\d{4}(\d{4})/i", "$1****$2", $message);
50         if ($this->next) {
51             return $this->next->filter($message);
52         } else {
53             return $message;
54         }
55     }
56 }
57
58 $f1 = new FilterStrict();
59 $f2 = new FilterWarning();
60 $f3 = new FilterMobile();
61
62 $f1->setNext($f2);
63 $f2->setNext($f3);
64
65 $m1 = "现在开始测试链条1：语句中不包含敏感词，需要替换掉打架这种词，然后给手机号加上星：
13333333333，这样的数据才可以对外展示哦";
66 echo $f1->filter($m1);
67 echo PHP_EOL;
68
69 $m2 = "现在开始测试链条2：这条语句走不到后面，因为包含了毒X，直接就报错了！！！语句中不包含敏感
词，需要替换掉打架这种词，然后给手机号加上星：13333333333，这样的数据才可以对外展示哦";
70 echo $f1->filter($m2);
71 echo PHP_EOL;

```

## 说明

- 在这个例子中，我们对消息内容进行了各种处理。当有新的需求产生时，我们只需要加入新的过滤类，然后调整客户端的执行顺序即可
- 使用了next来标识下一步的操作，使用过Laravel的同学一定马上联想到了中间件。当然，用过Node开发服务器的同学更是不会陌生，早就对这个设计模式了如指掌了。
- 责任链的运用真的非常广泛，在各种工作流软件及中间件组件中都可以看到，同时配合Linux下的管道思想，可以把这个模式的优势发挥到极致
- Laravel的中间件，有兴趣的朋友翻翻源码，典型的责任链模式的应用哦