

PHP设计模式之享元模式

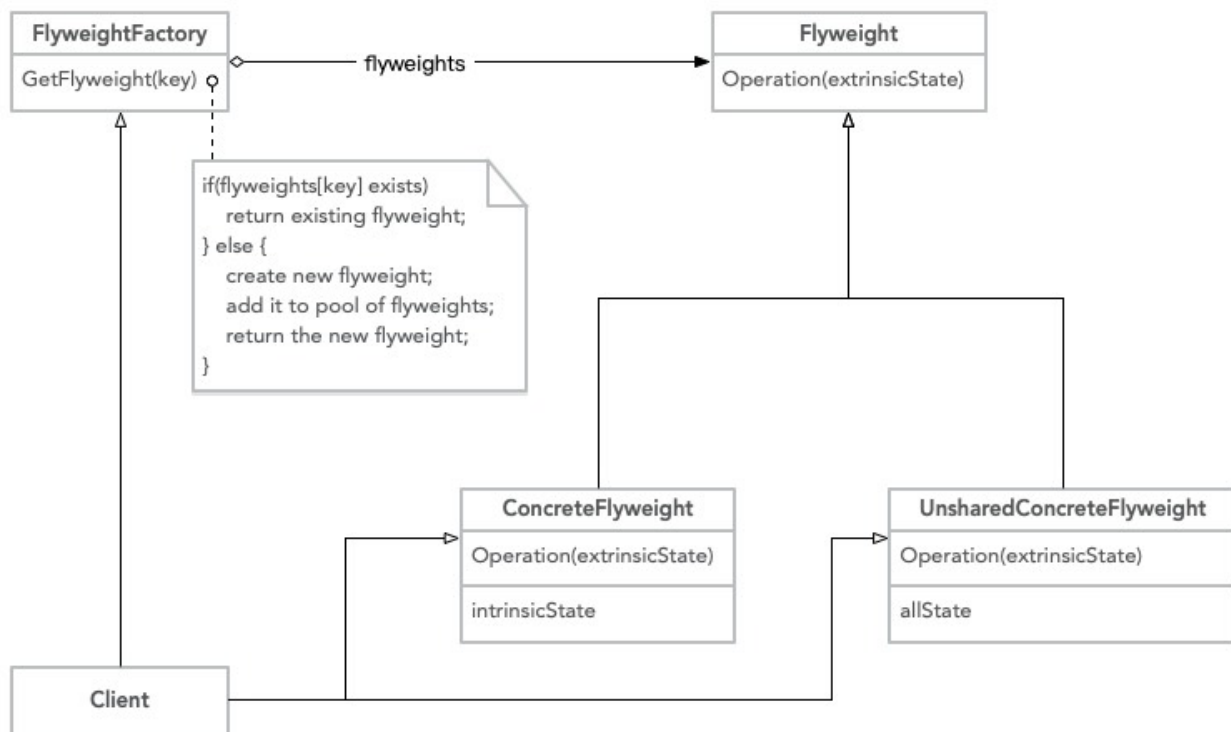
享元模式，“享元”这两个字在中文里其实并没有什么特殊的意思，所以我们要把它拆分来看。

“享”就是共享，“元”就是元素，这样一来似乎就很容易理解了，共享某些元素嘛。

Gof类图及解释

GoF定义：运用共享技术有效地支持大量细粒度的对象

GoF类图



代码实现

```
1 interface Flyweight
2 {
3     public function operation($extrinsicState) : void;
4 }
5
6 class ConcreteFlyweight implements Flyweight
7 {
8     private $intrinsicState = 101;
9     function operation($extrinsicState) : void
10    {
11        echo '共享享元对象' . ($extrinsicState + $this->intrinsicState) . PHP_EOL;
12    }
13 }
```

```

13 }
14
15 class UnsharedConcreteFlyweight implements Flyweight
16 {
17     private $allState = 1000;
18     public function operation($extrinsicState) : void
19     {
20         echo '非共享享元对象: ' . ($extrinsicState + $this->allState) . PHP_EOL;
21     }
22 }

```

定义共享接口以及它的实现，注意这里有两个实现，ConcreteFlyweight进行状态的共享，UnsharedConcreteFlyweight不共享或者说他的状态不需要去共享

```

1 class FlyweightFactory
2 {
3     private $flyweights = [];
4
5     public function getFlyweight($key) : Flyweight
6     {
7         if (!array_key_exists($key, $this->flyweights)) {
8             $this->flyweights[$key] = new ConcreteFlyweight();
9         }
10        return $this->flyweights[$key];
11    }
12 }

```

保存那些需要共享的对象，做为一个工厂来创建需要的共享对象，保证相同的键值下只会有唯一的对象，节省相同对象创建的开销

```

1 $factory = new FlyweightFactory();
2
3 $extrinsicState = 100;
4 $f1A = $factory->getFlyweight('a');
5 $f1A->operation(--$extrinsicState);
6
7 $f1B = $factory->getFlyweight('b');

```

```
8 $f1B->operation(--$extrinsicState);
9
10 $f1C = $factory->getFlyweight('c');
11 $f1C->operation(--$extrinsicState);
12
13 $f1D = new UnsharedConcreteFlyweight();
14 $f1D->operation(--$extrinsicState);
```

客户端的调用，让外部状态\$extrinsicState能够在各个对象之间共享

- 有点意思吧，这个模式的代码量可不算少
- 当一个应用程序使用了大量非常相似的对象，对象的大多数状态都可变为外部状态时，很适合享元模式
- 这里的工厂是存储对象列表的，不是像工厂方法或者抽象工厂一样去创建对象的，虽说这里也进行了创建，但如果对象存在，则会直接返回，而且列表也是一直维护的
- 享元模式在现实中，大家多少一定用过，各种池技术就是它的典型应用：线程池、连接池等等，另外两个一样的字符串String类型在php或Java中都是可以===的，这也运用到了享元模式，它们连内存地址都是一样的，这不就是一种共享嘛
- 关于享元模式，有一个极其经典的例子，比我下面的例子要好的多，那就是关于围棋的棋盘。围棋只有黑白两色，所以两个对象就够了，接下来呢？改变他们的位置状态就好啦！有兴趣的朋友可以搜搜哈！
- Laravel中的IoC容器可以看作是一种享元模式的实现。它把对象保存在数组中，在需要的时候通过闭包机制进行取用，也有一些类有共享一些状态属性的内容。大家可以翻看代码了解了解。

还是说到科技以换壳为本这件事上。毕竟，大家都还是喜欢各种颜色的手机来彰显自己的个性。之前说过，如果每种颜色我们都要做一条生产线的话那岂不是一项巨大的投入。还好，每个型号我们的工厂（享元工厂）只生产最基本的背景壳（对象），然后通过专门的印刷线（状态变化）来进行上色不就好啦！嗯，下一款Iphone早晚也会模仿我们的，看来我们得先把各种金、各种土豪色集齐才行，说不定还能召唤神龙呢！！

完整代码：<https://github.com/zhangyue0503/designpatterns-php/blob/master/13.flyweights/source/flyweights.php>

实例

果然不出意外的我们还是来发短信，这回的短信依然使用的阿里云和极光短信来进行发送，不过这次我们使用享元模式来实现，这里的享元工厂我们保存了两种不同类型的对象哦，通过内外状态来让它们千变万化吧！

短信发送类图

完整源码：<https://github.com/zhangyue0503/designpatterns-php/blob/master/13.flyweights/source/flyweights-message.php>

```
1 <?php
2
3 interface Message
4 {
5     public function send(User $user);
6 }
7
8 class AliYunMessage implements Message
9 {
10     private $template;
11     public function __construct($template)
12     {
13         $this->template = $template;
14     }
15     public function send(User $user)
16     {
17         echo '使用阿里云短信向' . $user->GetName() . '发送: ';
18         echo $this->template->GetTemplate(), PHP_EOL;
19     }
20 }
21
22 class JiGuangMessage implements Message
23 {
24     private $template;
25     public function __construct($template)
26     {
27         $this->template = $template;
28     }
29     public function send(User $user)
30     {
31         echo '使用极光短信向' . $user->GetName() . '发送: ';
32         echo $this->template->GetTemplate(), PHP_EOL;
33     }
34 }
35
36 class MessageFactory
```

```
37 {
38     private $messages = [];
39     public function GetMessage(Template $template, $type = 'ali')
40     {
41         $key = md5($template->GetTemplate() . $type);
42         if (!key_exists($key, $this->messages)) {
43             if ($type == 'ali') {
44                 $this->messages[$key] = new AliYunMessage($template);
45             } else {
46                 $this->messages[$key] = new JiGuangMessage($template);
47             }
48         }
49         return $this->messages[$key];
50     }
51
52     public function GetMessageCount()
53     {
54         echo count($this->messages);
55     }
56 }
57
58 class User
59 {
60     public $name;
61     public function GetName()
62     {
63         return $this->name;
64     }
65 }
66
67 class Template
68 {
69     public $template;
70     public function GetTemplate()
71     {
72         return $this->template;
73     }
74 }
75
76 // 内部状态
```

```
77 $t1 = new Template();
78 $t1->template = '模板1, 不错哟!';
79
80 $t2 = new Template();
81 $t2->template = '模板2, 还好啦!';
82
83 // 外部状态
84 $u1 = new User();
85 $u1->name = '张三';
86
87 $u2 = new User();
88 $u2->name = '李四';
89
90 $u3 = new User();
91 $u3->name = '王五';
92
93 $u4 = new User();
94 $u4->name = '赵六';
95
96 $u5 = new User();
97 $u5->name = '田七';
98
99 // 享元工厂
100 $factory = new MessageFactory();
101
102 // 阿里云发送
103 $m1 = $factory->GetMessage($t1);
104 $m1->send($u1);
105
106 $m2 = $factory->GetMessage($t1);
107 $m2->send($u2);
108
109 echo $factory->GetMessageCount(), PHP_EOL; // 1
110
111 $m3 = $factory->GetMessage($t2);
112 $m3->send($u2);
113
114 $m4 = $factory->GetMessage($t2);
115 $m4->send($u3);
```

```

116
117 echo $factory->GetMessageCount(), PHP_EOL; // 2
118
119 $m5 = $factory->GetMessage($t1);
120 $m5->send($u4);
121
122 $m6 = $factory->GetMessage($t2);
123 $m6->send($u5);
124
125 echo $factory->GetMessageCount(), PHP_EOL; // 2
126
127 // 加入极光
128 $m1 = $factory->GetMessage($t1, 'jg');
129 $m1->send($u1);
130
131 $m2 = $factory->GetMessage($t1);
132 $m2->send($u2);
133
134 echo $factory->GetMessageCount(), PHP_EOL; // 3
135
136 $m3 = $factory->GetMessage($t2);
137 $m3->send($u2);
138
139 $m4 = $factory->GetMessage($t2, 'jg');
140 $m4->send($u3);
141
142 echo $factory->GetMessageCount(), PHP_EOL; // 4
143
144 $m5 = $factory->GetMessage($t1, 'jg');
145 $m5->send($u4);
146
147 $m6 = $factory->GetMessage($t2, 'jg');
148 $m6->send($u5);
149
150 echo $factory->GetMessageCount(), PHP_EOL; // 4
151

```

说明

- 代码有点多吧，但其实一共是两种类型的类，生成了四种对象。这里每个类不同的对象是根据模板

来区分的

- 这样的组合还是比较方便的吧，再结合别的模式将工厂这里优化一下，嗯，前途不可限量，你们可以想想哦！
- 享元模式适用于系统中存在大量的相似对象以及需要缓冲池的场景，能够降低内存占用，提高效率，但会增加复杂度，需要分享内外部状态
- 最主要的特点是有一个唯一标识，当内存中已经有这个对象了，直接返回对象，不用再去创建了