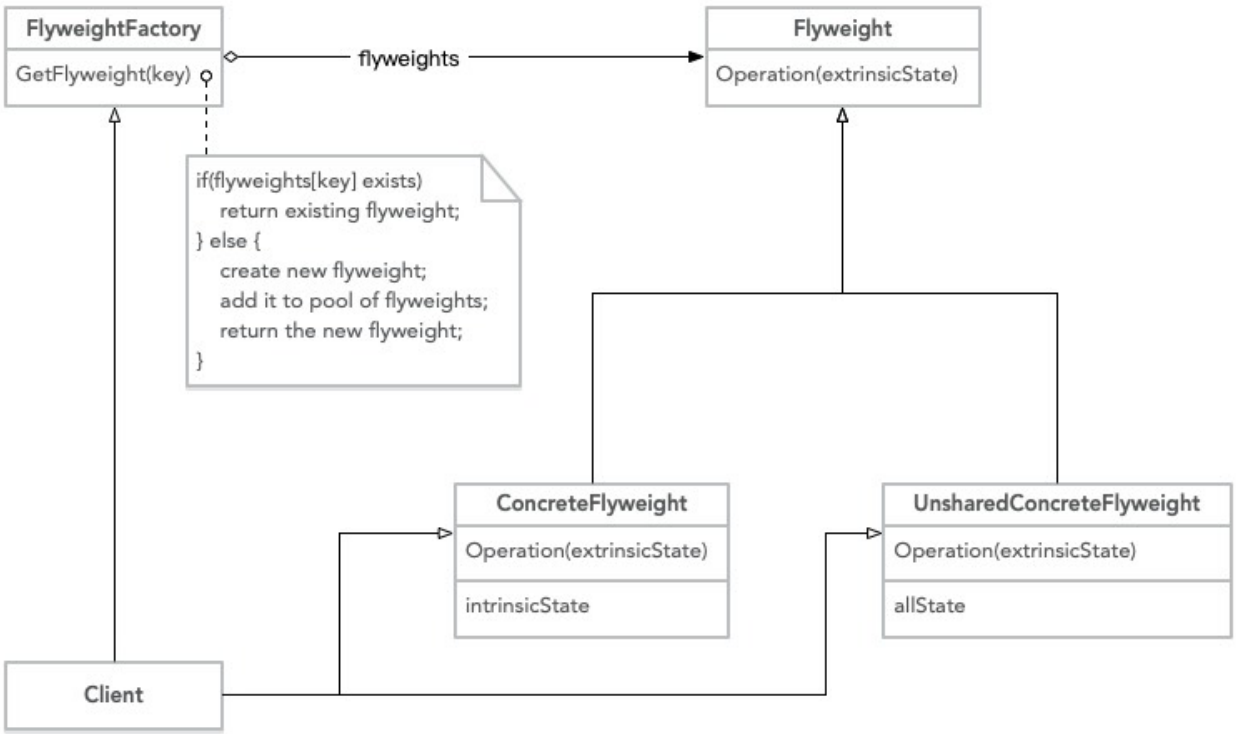
PHP设计模式之享元模式

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

```php
interface Flyweight
{
    public function operation($extrinsicState) : void;
}

class ConcreteFlyweight implements Flyweight
{
    private $intrinsicState = 101;
    function operation($extrinsicState) : void
    {
        echo '共享享元对象' . ($extrinsicState + $this->intrinsicState) . PHP_EOL;
    }
```

```
}

class UnsharedConcreteFlyweight implements Flyweight
{
    private $allState = 1000;
    public function operation($extrinsicState) : void
    {
        echo '非共享享元对象：' . ($extrinsicState + $this->allState) . PHP_EOL;
    }
}
```

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

```
class FlyweightFactory
{
    private $flyweights = [];

    public function getFlyweight($key) : Flyweight
    {
        if (!array_key_exists($key, $this->flyweights)) {
            $this->flyweights[$key] = new ConcreteFlyweight();
        }
        return $this->flyweights[$key];
    }
}
```

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

```
$factory = new FlyweightFactory();

$extrinsicState = 100;
$flA = $factory->getFlyweight('a');
$flA->operation(--$extrinsicState);

$flB = $factory->getFlyweight('b');
```

```php
$flB->operation(--$extrinsicState);

$flC = $factory->getFlyweight('c');
$flC->operation(--$extrinsicState);

$flD = new UnsharedConcreteFlyweight();
$flD->operation(--$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的对象，对象的大多数状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。

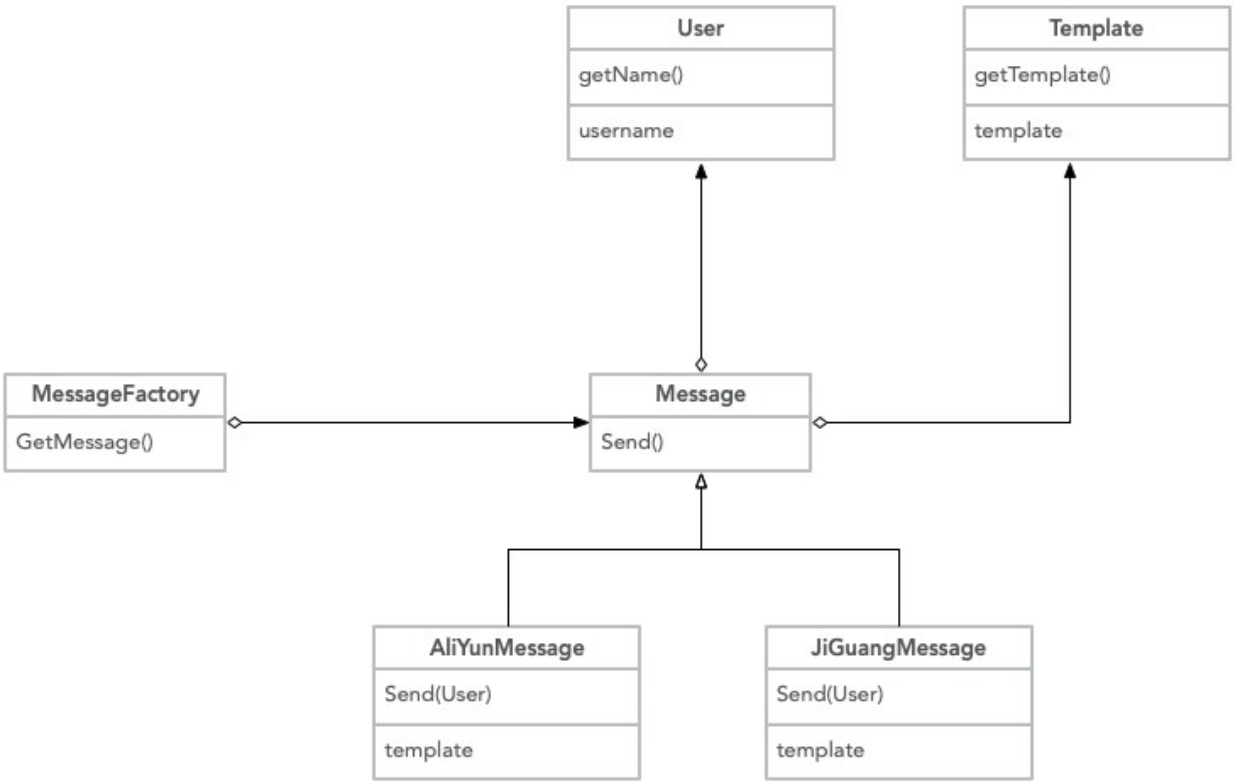
还是说到科技以换壳为本这件事上。毕竟，大家都还是喜欢各种颜色的手机来彰显自己的个性。之前说过，如果每种颜色我们都要做一条生产线的话那岂不是一项巨大的投入。还好，每个型号我们的工厂（享元工厂）只生产最基本的背景壳（对象），然后通过专门的印刷线（状态变化）来进行上色不就好啦！嗯，下一款Iphone早晚也会模仿我们的，看来我们得先把各种金、各种土豪色集齐才行，说不定还能召唤神龙呢！！

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

```php
<?php

interface Message
{
    public function send(User $user);
}

class AliYunMessage implements Message
{
    private $template;
    public function __construct($template)
    {
        $this->template = $template;
    }
    public function send(User $user)
    {
        echo '使用阿里云短信向' . $user->GetName() . '发送：';
        echo $this->template->GetTemplate(), PHP_EOL;
    }
```

```php
}

class JiGuangMessage implements Message
{
    private $template;
    public function __construct($template)
    {
        $this->template = $template;
    }
    public function send(User $user)
    {
        echo '使用极光短信向' . $user->GetName() . '发送：';
        echo $this->template->GetTemplate(), PHP_EOL;
    }
}

class MessageFactory
{
    private $messages = [];
    public function GetMessage(Template $template, $type = 'ali')
    {
        $key = md5($template->GetTemplate() . $type);
        if (!key_exists($key, $this->messages)) {
            if ($type == 'ali') {
                $this->messages[$key] = new AliYunMessage($template);
            } else {
                $this->messages[$key] = new JiGuangMessage($template);
            }
        }
        return $this->messages[$key];
    }

    public function GetMessageCount()
    {
        echo count($this->messages);
    }
}

class User
{
```

```php
    public $name;
    public function GetName()
    {
        return $this->name;
    }
}

class Template
{
    public $template;
    public function GetTemplate()
    {
        return $this->template;
    }
}

// 内部状态
$t1 = new Template();
$t1->template = '模板1，不错哟！';

$t2 = new Template();
$t2->template = '模板2，还好啦！';

// 外部状态
$u1 = new User();
$u1->name = '张三';

$u2 = new User();
$u2->name = '李四';

$u3 = new User();
$u3->name = '王五';

$u4 = new User();
$u4->name = '赵六';

$u5 = new User();
$u5->name = '田七';

```

```php
// 享元工厂
$factory = new MessageFactory();

// 阿里云发送
$m1 = $factory->GetMessage($t1);
$m1->send($u1);

$m2 = $factory->GetMessage($t1);
$m2->send($u2);

echo $factory->GetMessageCount(), PHP_EOL; // 1

$m3 = $factory->GetMessage($t2);
$m3->send($u2);

$m4 = $factory->GetMessage($t2);
$m4->send($u3);

echo $factory->GetMessageCount(), PHP_EOL; // 2

$m5 = $factory->GetMessage($t1);
$m5->send($u4);

$m6 = $factory->GetMessage($t2);
$m6->send($u5);

echo $factory->GetMessageCount(), PHP_EOL; // 2

// 加入极光
$m1 = $factory->GetMessage($t1, 'jg');
$m1->send($u1);

$m2 = $factory->GetMessage($t1);
$m2->send($u2);

echo $factory->GetMessageCount(), PHP_EOL; // 3

$m3 = $factory->GetMessage($t2);
$m3->send($u2);

```

```php
$m4 = $factory->GetMessage($t2, 'jg');
$m4->send($u3);

echo $factory->GetMessageCount(), PHP_EOL; // 4

$m5 = $factory->GetMessage($t1, 'jg');
$m5->send($u4);

$m6 = $factory->GetMessage($t2, 'jg');
$m6->send($u5);

echo $factory->GetMessageCount(), PHP_EOL; // 4

```

> 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