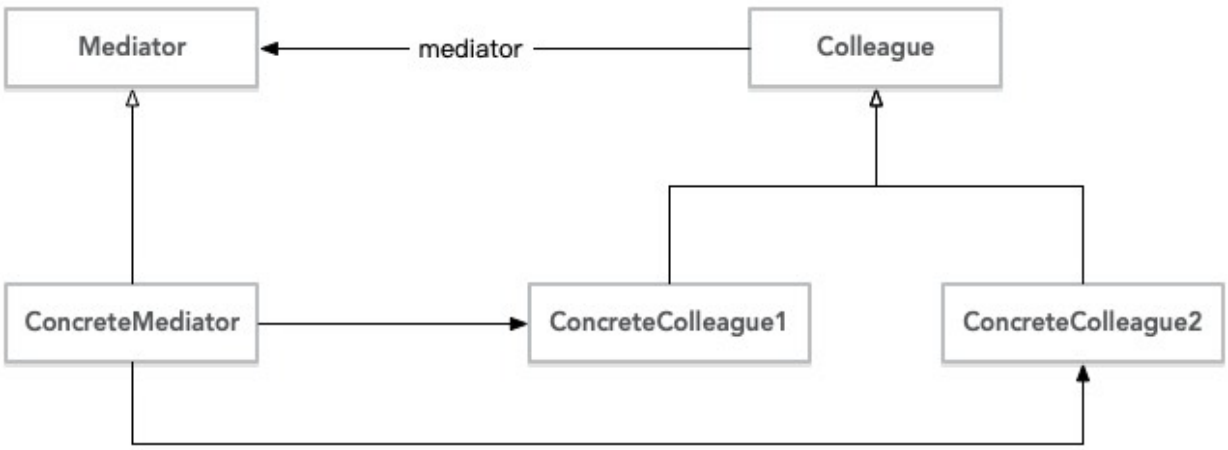
PHP设计模式之中介者模式

上回说道，我们在外打工的经常会和一类人有很深的接触，那就是房产中介。大学毕业后马上就能在喜欢的城市买到房子的X二代不在我们的考虑范围内哈。既然需要长期的租房，那么因为工作或者生活的变动，不可避免的一两年或者三五年就要和房产中介打一次交道。有的时候，我们租房并不一定会知道房主的信息，房主也不用知道我们的信息，全部都由中介来进行处理。在这里，中介就成为了我们沟通的桥梁，这种情况其实就像是房主出国了或者在外地有事儿而将房子完全的托管到了中介手中。类似于这种情况，在代码世界中，就是中介者模式的典型应用。

Gof类图及解释

GoF定义：用一个中介对象来封装一系列的对象交互。 中介者使各对象不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互

> GoF类图



> 代码实现

```
abstract class Mediator
{
    abstract public function Send(String $message, Colleague $colleague);
}

class ConcreteMediator extends Mediator
{
    public $colleague1;
    public $colleague2;

    public function Send(String $message, Colleague $colleague)
    {
```

```
        if ($colleague == $this->colleague1) {
            $this->colleague2->Notify($message);
        } else {
            $this->colleague1->Notify($message);
        }
    }
}
```

抽象出来的中介者和具体的实现，在这里，我们假定有固定的两个同事类，让他们互相对话，所以进入的同事是1的时候，就去调用2的Notify方法，相当于是让2接收到了1发来的消息

```
abstract class Colleague
{
    protected $mediator;
    public function __construct(Mediator $mediator)
    {
        $this->mediator = $mediator;
    }

}

class ConcreteColleague1 extends Colleague
{
    public function Send(String $message)
    {
        $this->mediator->Send($message, $this);
    }
    public function Notify(String $message)
    {
        echo "同事1得到信息: " . $message, PHP_EOL;
    }
}

class ConcreteColleague2 extends Colleague
{
    public function Send(String $message)
    {
        $this->mediator->Send($message, $this);
```

```
    }
    public function Notify(String $message)
    {
        echo "同事2得到信息：" . $message;
    }
}
```

同事类及具体的实现，这里我们要确认的一点就是，每一个同事类，只认识中介者，并不认识另外的同事类，这就是中介者的特点，双方不用认识。

```
$m = new ConcreteMediator();

$c1 = new ConcreteColleague1($m);
$c2 = new ConcreteColleague2($m);

$m->colleague1 = $c1;
$m->colleague2 = $c2;

$c1->Send("吃过饭了吗？");
$c2->Send("没有呢，你打算请客？");
```

客户端的调用就比较很简单啦！

- 是不是感觉这个模式很适合做一些通讯类的产品？没错，聊天社交、sns、直播之类的都很合适，因为这个模式就是能让用户与用户之间解耦，不需要让一个用户去维护所有有关联的用户对象
- 因为不需要用户去维护关系，所以也就顺便解决了关系之间的多对多维护的问题，同时，也不需要去修改用户类来进行关系的变更，保持了用户类的良好封装
- 但是，中介者集中维护可能导致这个类过于复杂和庞大
- 所以，模式不是万能的，一定要弄清楚业务场景进行取舍地使用
- 中介者适用于一组对象以定义良好但是复杂的方式进行通信的场合，以及想定制一个分布在多个类中的行为，而又不想生成太多子类的场合

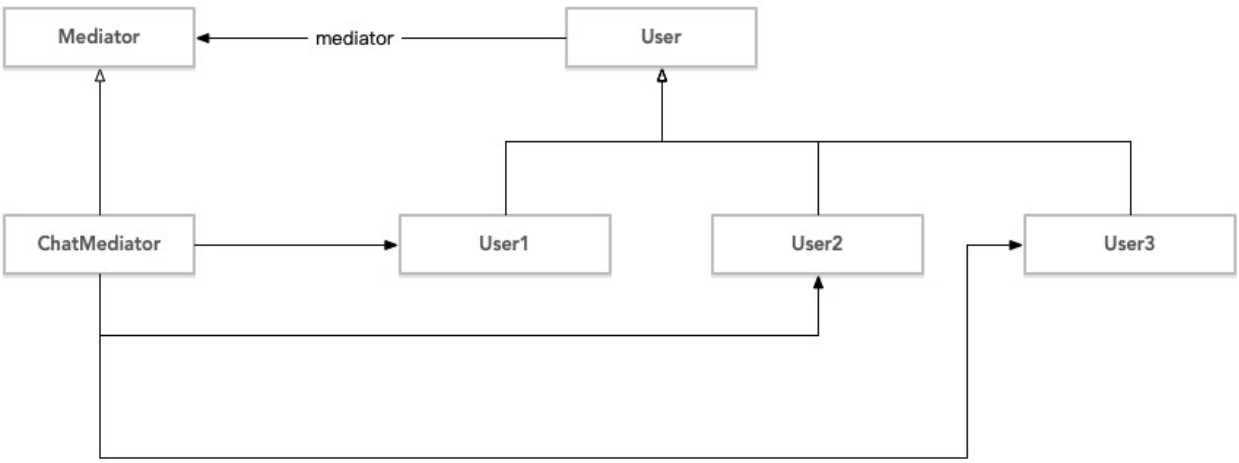
作为一名企业家，深知项目管理的重要性，而项目经理，在很多场合下就是一名中介者的角色。从组织角度看，一个项目的开始和结束，作为老板的我并不需要关心是由谁来具体编码实现，我要沟通的人只是项目经理。同理，其他辅助部门包括财务、人事、行政等，他们也不关心谁来写代码，而只需要和项目经理交流了解项目的情况以及需要配合的内容。在项目团队中，写代码的人呢？也不需要知道谁来给他发工资或者考勤问题出在哪里，这一切也交给项目经理解决就好了。所以说，项目经理负责制的项目开发，就是中介者模式的典型应用。我们的手机厂之所以发展的如此之快，也多亏了这些项目经理们，晚上请他们吃大餐去咯~~~

完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/15.mediator/source/mediator.php

实例

这回我们不发短信了，实现一个聊天室吧。一个简单的在线聊天室，需求就是让进入聊天室的用户都可以在线聊天，让我们来看看使用中介者模式来如何实现这个聊天室吧！

> 聊天室类图



完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/15.mediator/source/mediator-webchat.php

```php
<?php

abstract class Mediator
{
    abstract public function Send($message, $user);
}

class ChatMediator extends Mediator
{
    public $users = [];
    public function Attach($user)
    {
        if (!in_array($user, $this->users)) {
            $this->users[] = $user;
        }
    }

    public function Detach($user)
```

```
    {
        $position = 0;
        foreach ($this->users as $u) {
            if ($u == $user) {
                unset($this->users[$position]);
            }
            $position++;
        }
    }

    public function Send($message, $user)
    {
        foreach ($this->users as $u) {
            if ($u == $user) {
                continue;
            }
            $u->Notify($message);
        }
    }
}

abstract class User
{
    public $mediator;
    public $name;

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

class ChatUser extends User
{
    public function Send($message)
    {
        $this->mediator->Send($message . '(' . $this->name . '发送)', $this);
    }
    public function Notify($message)
```

```
    {
        echo $this->name . '收到消息：' . $message, PHP_EOL;
    }
}

$m = new ChatMediator();

$u1 = new ChatUser($m, '用户1');
$u2 = new ChatUser($m, '用户2');
$u3 = new ChatUser($m, '用户3');

$m->Attach($u1);
$m->Attach($u3);
$m->Attach($u2);

$u1->Send('Hello, 大家好呀！'); // 用户2、用户3收到消息

$u2->Send('你好呀！'); // 用户1、用户3收到消息

$m->Detach($u2); // 用户2退出聊天室

$u3->Send('欢迎欢迎！'); // 用户1收到消息

```

> 说明

- 有没有发现，中介者就是这个“聊天室”，由它来进行信息的传递转移
- 这里由于不固定用户人数，因此是一个数组维护的，当用户发送消息的时候，除了他自己，其他人都收到了这条消息
- 聊天室可以自由地进出用户，说实话，这个例子真的很像一个已经差不多实现功能了的聊天应用哦
- 果然中介者模式真的很适合通信方面的应用，但是，如果进入的用户非常多，$users列表就会越来越臃肿了哦，这就是上文中所述的中介者模式的问题所在