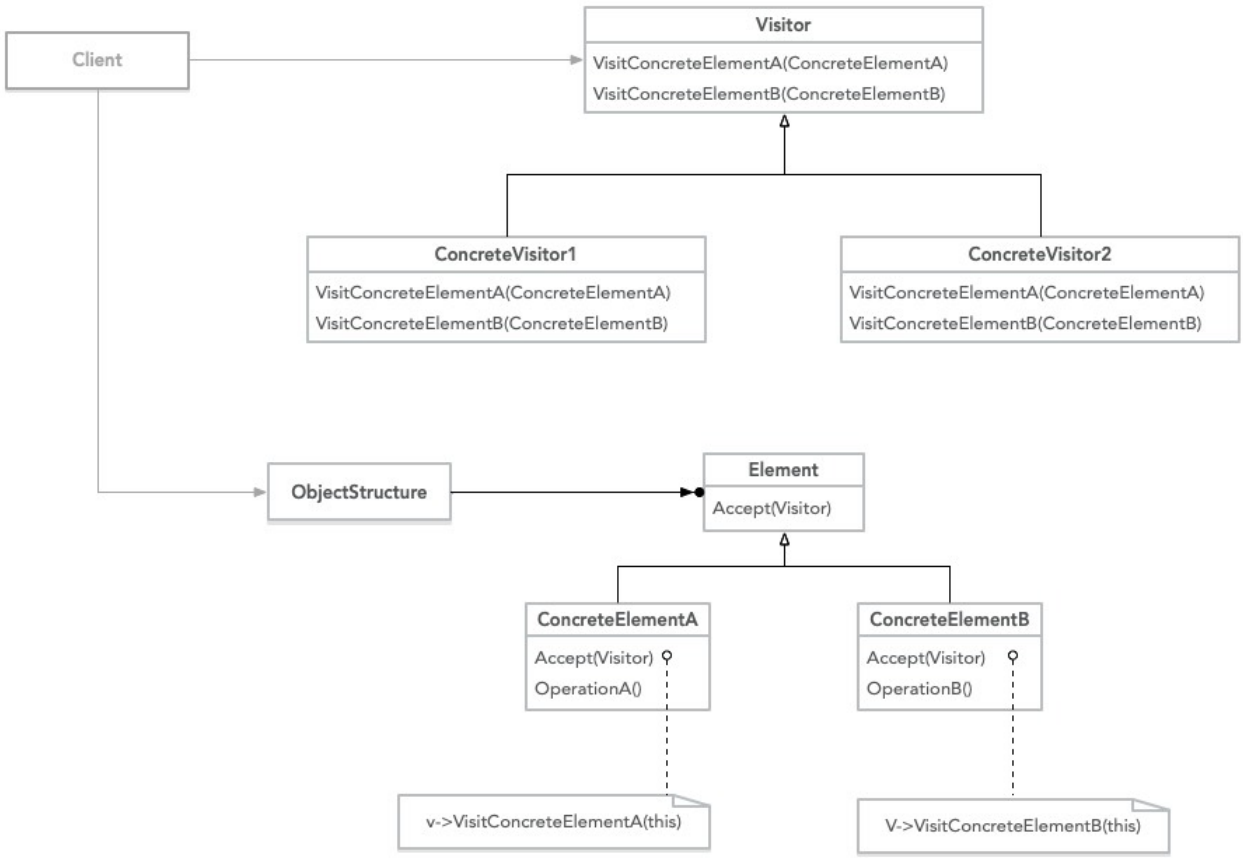
# PHP设计模式之访问者模式

访问者，就像我们去别人家访问，或者别人来我们家看望我们一样。我们每个人都像是一个实体，而来访的人都会一一的和我们打招呼。毕竟，我们中华民族是非常讲究礼数和好客的民族。访问者是GoF23个设计模式中最复杂的一个模式，也是各类设计模式教材都放在最后的一个模式。先不管难度如何，我们先看看它的定义和实现。

## Gof类图及解释

**GoF定义：表示一个作用于某对象结构中的各元素的操作。它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作**

> GoF类图




> 代码实现

```php
interface Visitor
{
    public function VisitConcreteElementA(ConcreteElementA $a);
    function VisitConcreteElementB(ConcreteElementB $b);
}

class ConcreteVisitor1 implements Visitor
```

```
{
    public function VisitConcreteElementA(ConcreteElementA $a)
    {
        echo get_class($a) . "被" . get_class($this) . "访问", PHP_EOL;
    }
    public function VisitConcreteElementB(ConcreteElementB $b)
    {
        echo get_class($b) . "被" . get_class($this) . "访问", PHP_EOL;
    }
}

class ConcreteVisitor2 implements Visitor
{
    public function VisitConcreteElementA(ConcreteElementA $a)
    {
        echo get_class($a) . "被" . get_class($this) . "访问", PHP_EOL;
    }
    public function VisitConcreteElementB(ConcreteElementB $b)
    {
        echo get_class($b) . "被" . get_class($this) . "访问", PHP_EOL;
    }
}
```

抽象的访问者接口及两个具体实现。可以看作是一家小两口来我们家作客咯!

```
interface Element
{
    public function Accept(Visitor $v);
}

class ConcreteElementA implements Element
{
    public function Accept(Visitor $v)
    {
        $v->VisitConcreteElementA($this);
    }
    public function OperationA()
    {
```

```
14
15      }
16  }
17
18  class ConcreteElementB implements Element
19  {
20      public function Accept(Visitor $v)
21      {
22          $v->VisitConcreteElementB($this);
23      }
24      public function OperationB()
25      {
26
27      }
28  }
```

元素抽象及实现，也可以看作是要访问的实体。当然就是我和我媳妇啦。

```
1   class ObjectStructure
2   {
3       private $elements = [];
4
5       public function Attach(Element $element)
6       {
7           $this->elements[] = $element;
8       }
9
10      public function Detach(Element $element)
11      {
12          $position = 0;
13          foreach ($this->elements as $e) {
14              if ($e == $element) {
15                  unset($this->elements[$position]);
16                  break;
17              }
18              $position++;
19          }
20      }
```

```
21
22      public function Accept(Visitor $visitor)
23      {
24          foreach ($this->elements as $e) {
25              $e->Accept($visitor);
26          }
27      }
28
29  }
```

这是一个对象结构，用于保存元素实体并进行访问调用。大家在客厅里见面，互相寒暄嘛，这里就是个客厅

```
1  $o = new ObjectStructure();
2  $o->Attach(new ConcreteElementA());
3  $o->Attach(new ConcreteElementB());
4
5  $v1 = new ConcreteVisitor1();
6  $v2 = new ConcreteVisitor2();
7
8  $o->Accept($v1);
9  $o->Accept($v2);
```

客户端的调用，总算让大家正式见面了，互相介绍握手。一次访问就愉快的完成了。

- 让访问者调用指定的元素。这里需要注意的，访问者调用元素的行为一般是固定的，很少会改变的。也就是VisitConcreteElementA()、VisitConcreteElementB()这两个方法。也就是定义对象结构的类很少改变，但经常需要在此结构上定义新的操作时，会使用访问者模式
- 需要对一个对象结构中的对象进行很多不同的并且不相关的操作，而你想避免让这些操作“污染”这些对象的类时，适用于访问者模式
- 访问者模式适合数据结构不变化的情况。所以，它是一种平常你用不上，但一旦需要的时候就只能用这种模式的模式。GoF：“大多时候你并不需要访问者模式，但当一旦你需要访问者模式时，那就是真的需要它了”。因为很少有数据结构不发生变化的情况
- 访问者模式的一些优缺点：易于增加新的操作；集中相关的操作而分离无关的操作；增加新的ConcreteElement类很困难；通过类层次进行访问；累积状态；破坏封装

**我们公司的账务，只有收入和支出两项（Element），但是不同的部门（Visitor）访问的时候会给出不同的内容。比如我查看的时候只需要查看每月或每季度的汇总数据即可，财务总监则需要详细的收**

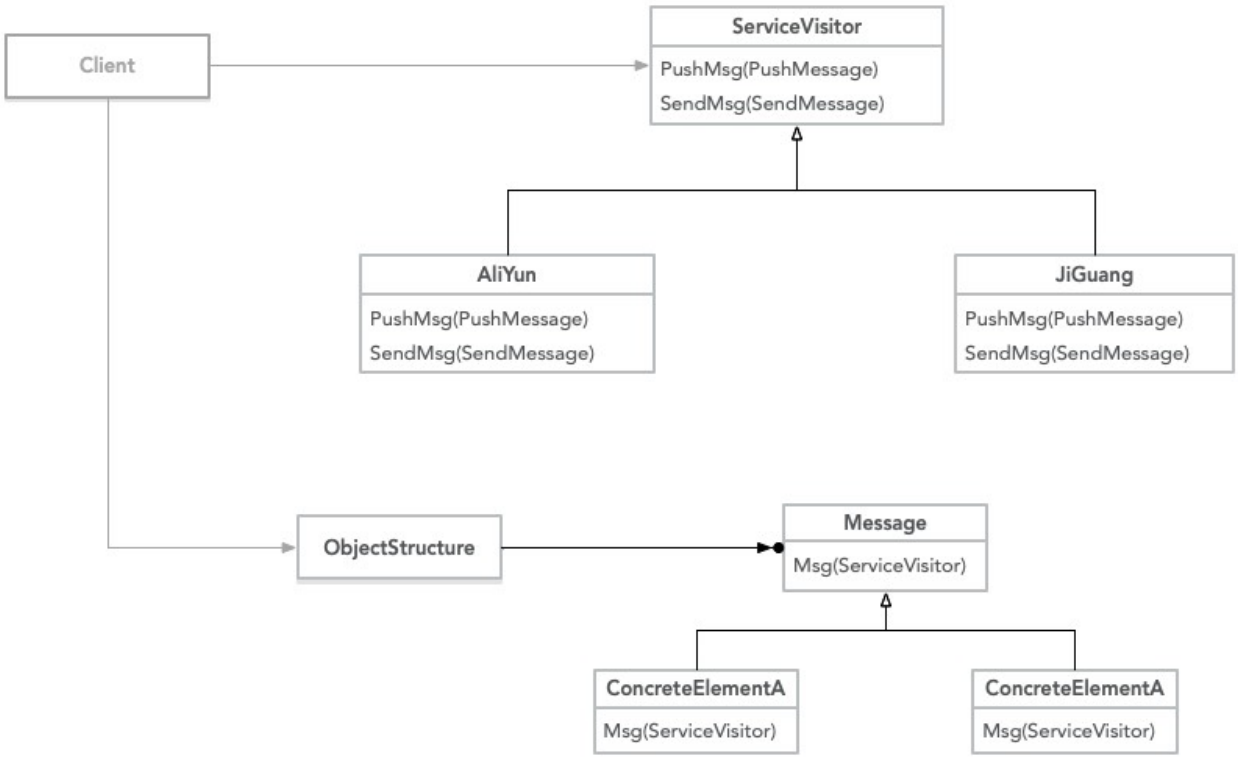
**支记录，而会计在做账时更是需要完整的明细。可见，公司的运营还真的是需要非常广泛的知识的，不仅是管理能力，账务知识也是必要了解的内容！！**

**完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/23.visitor/source/visitor.php**

## 实例

最后一个模式的例子还是回到我们的信息发送上来。同样的还是多个服务商，它们作为访问者需要去使用各自的短信发送及APP推送接口。这时，就可以使用访问者模式来进行操作，实现这些访问者的全部操作。

> 访问者模式信息发送




**完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/23.visitor/source/visitor-msg.php**

```php
<?php

interface ServiceVisitor
{
    public function SendMsg(SendMessage $s);
    function PushMsg(PushMessage $p);
}

class AliYun implements ServiceVisitor
```

```php
{
    public function SendMsg(SendMessage $s)
    {
        echo '阿里云发送短信! ', PHP_EOL;
    }
    public function PushMsg(PushMessage $p)
    {
        echo '阿里云推送信息! ', PHP_EOL;
    }
}

class JiGuang implements ServiceVisitor
{
    public function SendMsg(SendMessage $s)
    {
        echo '极光发送短信! ', PHP_EOL;
    }
    public function PushMsg(PushMessage $p)
    {
        echo '极光推送短信! ', PHP_EOL;
    }
}

interface Message
{
    public function Msg(ServiceVisitor $v);
}

class PushMessage implements Message
{
    public function Msg(ServiceVisitor $v)
    {
        echo '推送脚本启动: ';
        $v->PushMsg($this);
    }
}

class SendMessage implements Message
{
    public function Msg(ServiceVisitor $v)
```

```
    {
        echo '短信脚本启动：';
        $v->SendMsg($this);
    }
}

class ObjectStructure
{
    private $elements = [];

    public function Attach(Message $element)
    {
        $this->elements[] = $element;
    }

    public function Detach(Message $element)
    {
        $position = 0;
        foreach ($this->elements as $e) {
            if ($e == $element) {
                unset($this->elements[$position]);
                break;
            }
            $position++;
        }
    }

    public function Accept(ServiceVisitor $visitor)
    {
        foreach ($this->elements as $e) {
            $e->Msg($visitor);
        }
    }

}

$o = new ObjectStructure();
$o->Attach(new PushMessage());
$o->Attach(new SendMessage());
```

```php

$v1 = new AliYun();
$v2 = new JiGuang();

$o->Accept($v1);
$o->Accept($v2);
```

> 说明

- 我们假定发送短信和发送推送是不变的两个行为，也就是它们俩的数据结构是稳定不变的
- 这样我们就可以方便的增加ServiceVisitor，当增加百度云或者别的什么短信提供商时，就很方便的增加访问者就可以了
- 访问者模式比较适合数据结构稳定的结构。比如帐单只有收入支出、人的性别只有男女等