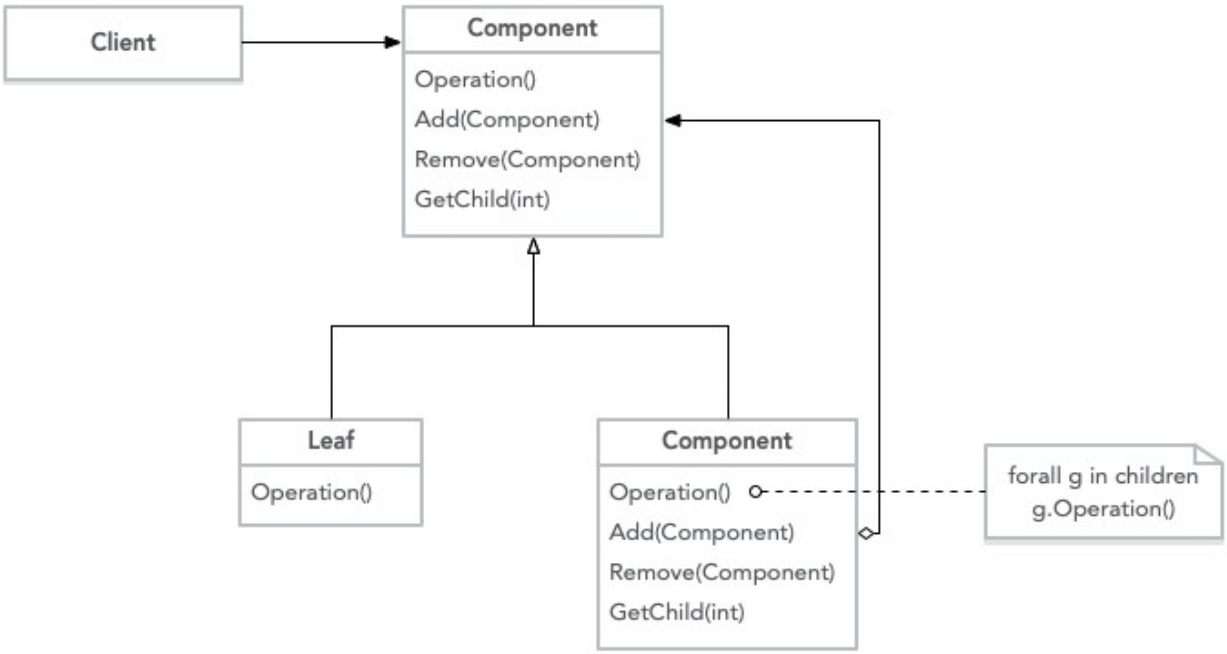
PHP设计模式之组合模式

互联网公司流行扁平化管理，也就是管理层级尽量少于或者不超过三层，作为一个底层的码农，你的CEO和你的职级也就相差3层以内。但是很多传统企业，则会有非常深的层级关系，从数据结构看，这种按职能进行分组的组织架构非常像一颗树。而我们今天介绍的组合模式的作用就和这个企业组织架构层级的模式非常类似。

Gof类图及解释

GoF定义：将对象组合成树形结构以表示“部分-整体”的层次结构。Composite使得用户对单个对象和组合对象的使用具有一致性

> GoF类图



> 代码实现

```php
abstract class Component
{
    protected $name;

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

    abstract public function Operation(int $depth);

```

```
    abstract public function Add(Component $component);

    abstract public function Remove(Component $component);
}
```

抽象出来的组合节点声明，在适当情况下实现所有类的公共接口的缺省行为，是所有子节点的父类。

```
class Composite extends Component
{
    private $componentList;

    public function Operation($depth)
    {
        echo str_repeat('-', $depth) . $this->name . PHP_EOL;
        foreach ($this->componentList as $component) {
            $component->Operation($depth + 2);
        }
    }

    public function Add(Component $component)
    {
        $this->componentList[] = $component;
    }

    public function Remove(Component $component)
    {
        $position = 0;
        foreach ($this->componentList as $child) {
            ++$position;
            if ($child == $component) {
                array_splice($this->componentList, ($position), 1);
            }
        }
    }

    public function GetChild(int $i)
    {
        return $this->componentList[$i];
```

```
    }
}
```

具体的节点实现类，保存下级节点的引用，定义实际的节点行为。

```
class Leaf extends Component
{
    public function Add(Component $c)
    {
        echo 'Cannot add to a leaf' . PHP_EOL;
    }
    public function Remove(Component $c)
    {
        echo 'Cannot remove from a leaf' . PHP_EOL;
    }
    public function Operation(int $depth)
    {
        echo str_repeat('-', $depth) . $this->name . PHP_EOL;
    }
}
```

叶子节点，没有子节点的最终节点。

- 从来代码来看，完全就是一颗树的实现
- 所有的子节点和叶子节点都可以处理数据，但叶子节点为终点
- 你希望用户可以忽略组合对象与单个对象的不同，统一地使用组合结构中的所有对象时，就应该考虑使用组合模式
- 用户不用关心到底是处理一个叶节点还是处理一个组合组件 ，也就用不着为定义组合而写一些选择判断语句了
- 组合模式可以让客户一致性地使用组合结构和单个对象

接着文章最开头的例子来说，在我们的组织架构中，一项任务下达到最底的人员时，会经历多个层级。我还是比较喜欢传统一起的企业管理方式。通常是一名总监对应多个主管，一名主管对应多位经理，一位经理对应多位组长，一名组长对应多名员工。当一个通知下发时，每一层级的工作人员都要做出回应，并将通知继续下发到下属员工那里，同时从下属哪里获得反馈。这样，我们就不知不觉地在实践中完成了一次组合模式的应用。突然感觉自己棒棒哒，感觉人生已经到达了巅峰！！

完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/14.composite/source/composite.php

实例

短信短信，这个功能我们可以是翻来覆去的用了。这次也不例外。这一回我们的网站后台的功能是要针对不同分站和不同来源的用户进行短信的发送。在这里，我们依然只关注短信发送这件事儿，我们希望给你不同渠道角色但包含统一行为的用户，你来进行发送就行了，这样的功能似乎并不难吧！

> 短信发送类图

完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/14.composite/source/composite-msg.php

```php
<?php

abstract class Role
{
    protected $userRoleList;
    protected $name;
    public function __construct(String $name)
    {
        $this->name = $name;
    }

    abstract public function Add(Role $role);

    abstract public function Remove(Role $role);

    abstract public function SendMessage();
}

class RoleManger extends Role
{
    public function Add(Role $role)
    {
        $this->userRoleList[] = $role;
    }

    public function Remove(Role $role)
    {
        $position = 0;
        foreach ($this->userRoleList as $n) {
```

```php
            ++$position;
            if ($n == $role) {
                array_splice($this->userRoleList, ($position), 1);
            }
        }
    }

    public function SendMessage()
    {
        echo "开始发送用户角色: " . $this->name . '下的所有用户短信', PHP_EOL;
        foreach ($this->userRoleList as $role) {
            $role->SendMessage();
        }
    }
}

class Team extends Role
{

    public function Add(Role $role)
    {
        echo "小组用户不能添加下级了！", PHP_EOL;
    }

    public function Remove(Role $role)
    {
        echo "小组用户没有下级可以删除！", PHP_EOL;
    }

    public function SendMessage()
    {
        echo "小组用户角色: " . $this->name . '的短信已发送！', PHP_EOL;
    }
}

// root用户
$root = new RoleManger('网站用户');
$root->add(new Team('主站用户'));
$root->SendMessage();

```

```php
// 社交版块
$root2 = new RoleManger('社交版块');
$managerA = new RoleManger('论坛用户');
$managerA->add(new Team('北京论坛用户'));
$managerA->add(new Team('上海论坛用户'));

$managerB = new RoleManger('sns用户');
$managerB->add(new Team('北京sns用户'));
$managerB->add(new Team('上海sns用户'));

$root2->add($managerA);
$root2->add($managerB);
$root2->SendMessage();

```

> 说明

- 当我要发送论坛版块的用户时，我可以自由地添加各地方站的叶子节点来控制发送对象
- 你可以把整个$root2的发送看作是一个整体，不同的版块和地区看成是部分
- 这个组合可以一直向下延伸，直到深度的叶子节点结束，这个度当然是由自己来把控，很清晰明了