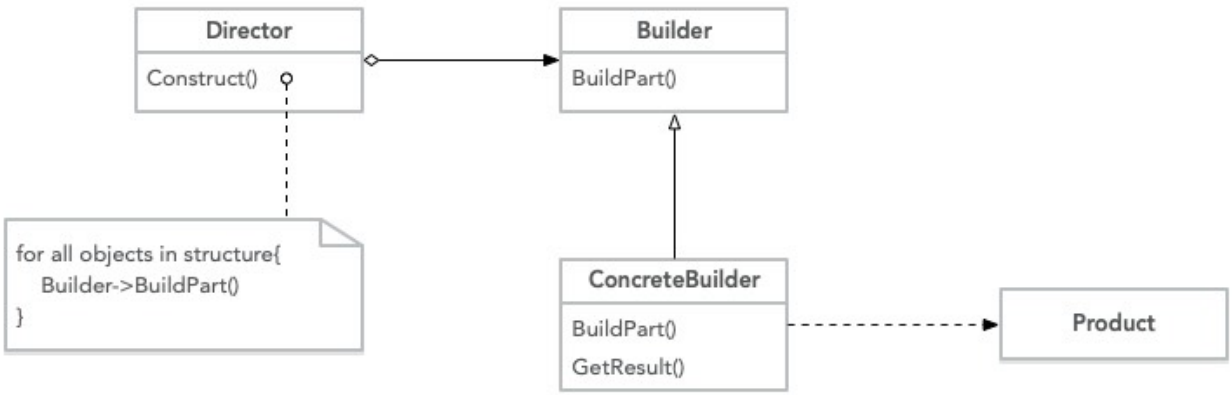
PHP设计模式之建造者模式

建造者模式，也可以叫做生成器模式，builder这个词的原意就包含了建筑者、开发者、创建者的含义。很明显，这个模式又是一个创建型的模式，用来创建对象。那么它的特点是什么呢？从建筑上来说，盖房子不是一下子就马上能把一个房子盖好的，而是通过一砖一瓦搭建出来的。一个房子不仅有砖瓦，还有各种管道，各种电线等等，由它们各个不部分共同组成了一栋房子。可以说，建造者模式就是这样非常形象的由各种部件来组成一个对象（房子）的过程。

Gof类图及解释

GoF定义：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示

> GoF类图



> 代码实现

```php
class Product
{
    private $parts = [];

    public function Add(String $part): void
    {
        $this->parts[] = $part;
    }

    public function Show(): void
    {
        echo PHP_EOL . '产品创建 ----', PHP_EOL;
        foreach ($this->parts as $part) {
            echo $part, PHP_EOL;
        }
```

```
    }
}
```

产品类，你可以把它想象成我们要建造的房子。这时的房子还没有任何内容，我们需要给它添砖加瓦。

```
interface Builder
{
    public function BuildPartA(): void;
    public function BuildPartB(): void;
    public function GetResult(): Product;
}

class ConcreteBuilder1 implements Builder
{
    private $product;

    public function __construct()
    {
        $this->product = new Product();
    }

    public function BuildPartA(): void
    {
        $this->product->Add('部件A');
    }
    public function BuildPartB(): void
    {
        $this->product->Add('部件B');
    }
    public function GetResult(): Product
    {
        return $this->product;
    }
}

class ConcreteBuilder2 implements Builder
{
```

```
    private $product;

    public function __construct()
    {
        $this->product = new Product();
    }

    public function BuildPartA(): void
    {
        $this->product->Add('部件X');
    }
    public function BuildPartB(): void
    {
        $this->product->Add('部件Y');
    }
    public function GetResult(): Product
    {
        return $this->product;
    }
}
```

建造者抽象及其实现。不同的开发商总会选用不同的品牌材料，这里我们有了两个不同的开发商，但他们的目的一致，都是为了去盖房子（Product）。

```
class Director
{
    public function Construct(Builder $builder)
    {
        $builder->BuildPartA();
        $builder->BuildPartB();
    }
}
```

构造器，用来调用建造者进行生产。没错，就是我们的工程队。它来选取材料并进行建造。同样的工程队，可以接不同的单，但盖出来的都是房子。只是这个房子的材料和外观不同，大体上的手艺还是共通的。

```php
$director = new Director();
$b1 = new ConcreteBuilder1();
$b2 = new ConcreteBuilder2();

$director->Construct($b1);
$p1 = $b1->getResult();
$p1->Show();

$director->Construct($b2);
$p2 = $b2->getResult();
$p2->Show();
```

最后看看我们的实现，是不是非常简单，准备好工程队，准备好不同的建造者，然后交给工程队去生产就好啦！！

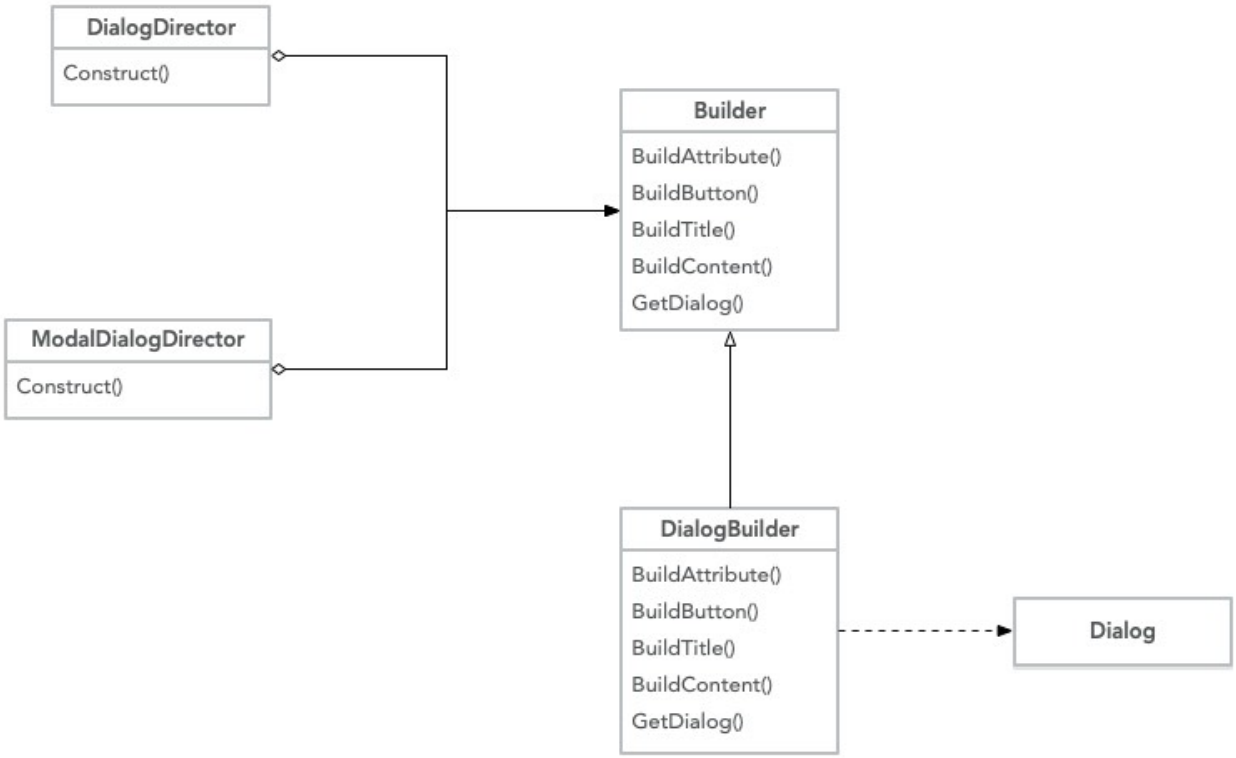
- 其实这个模式要解决的最主要问题就是一个类可能有非常多的配置、属性，这些配置、属性也并不全是必须要配置的，一次性的实例化去配置这些东西非常麻烦。这时就可以考虑让这些配置、属性变成一个一个可随时添加的部分。通过不同的属性组合拿到不同的对象。
- 上面那一条，在GoF那里的原文是：它使你改变一个产品的内部表示；它将构造代码和表示代码分开；它使你可以对构造过程进行更精细的控制。
- 再说得简单一点，对象太复杂，我们可以一部分一部分的拼装它！
- 了解过一点Android开发的一定不会陌生，创建对话框对象AlterDialog.builder
- Laravel中，数据库组件也使用了建造者模式，你可以翻看下源码中Database\Eloquent和Database\Query目录中是否都有一个Builder.php

大家都知道，手机组装是一件复杂的过程，于是，不同型号的手机我们都有对应的图纸（Builder），将图纸和配件交给流水线上的工人（Director），他们就会根据图纸使用配件来生产出我们所需要的各种不同型号的手机（Product）。很明显，我们都是伟大的建造者，为我们的产业添砖加瓦！！！
完整代码：https://github.com/zhangyue0503/designpatterns-php/blob/master/16.builder/source/builder.php

实例

前面说过Android中有很多Dialog对话框都会用建造者模式来实现，作为一家手机厂的老板，定制化的Android系统也是非常重要的一个部分。就像X米一样，从MIUI入手，先拿下了软件市场，让大家觉得这个系统非常好用，然后再开始开发手机。这就说明软硬件的确是现代手机的两大最重要的组成部分，缺了谁都不行。这回，我们就用建造者模式简单的实现一套对话框组件吧！

> 对话框类图



完整源码：https://github.com/zhangyue0503/designpatterns-php/blob/master/16.builder/source/builder-dialog.php

```php
<?php

class Dialog
{
    private $attributes = [];
    private $buttons = [];
    private $title = '';
    private $content = '';

    public function AddAttributes($attr)
    {
        $this->attributes[] = $attr;
    }
    public function AddButtons($button)
    {
        $this->buttons[] = $button;
    }
    public function SetTitle($title)
    {
        $this->title = $title;
```

```php
    }
    public function SetContent($content)
    {
        $this->content = $content;
    }

    public function ShowDialog(){
        echo PHP_EOL, '显示提示框 === ', PHP_EOL;
        echo '标题: ' . $this->title, PHP_EOL;
        echo '内容: ' . $this->content, PHP_EOL;
        echo '样式: ' . implode(',', $this->attributes), PHP_EOL;
        echo '按扭: ' . implode(',', $this->buttons), PHP_EOL;
    }
}

interface Builder
{
    public function BuildAttribute($attr);
    public function BuildButton($button);
    public function BuildTitle($title);
    public function BuildContent($content);
    public function GetDialog();
}

class DialogBuilder implements Builder{
    private $dialog;
    public function __construct(){
        $this->dialog = new Dialog();
    }
    public function BuildAttribute($attr){
        $this->dialog->AddAttributes($attr);
    }
    public function BuildButton($button){
        $this->dialog->AddButtons($button);
    }
    public function BuildTitle($title){
        $this->dialog->SetTitle($title);
    }
    public function BuildContent($content){
        $this->dialog->SetContent($content);
```

```
    }
    public function GetDialog(){
        return $this->dialog;
    }
}

class DialogDirector {
    public function Construct($title, $content){

        $builder = new DialogBuilder();

        $builder->BuildAttribute('置于顶层');
        $builder->BuildAttribute('居中显示');

        $builder->BuildButton('确认');
        $builder->BuildButton('取消');

        $builder->BuildTitle($title);
        $builder->BuildContent($content);

        return $builder;
    }
}

class ModalDialogDirector {
    public function Construct($title, $content){

        $builder = new DialogBuilder();

        $builder->BuildAttribute('置于顶层');
        $builder->BuildAttribute('居中显示');
        $builder->BuildAttribute('背景底照');
        $builder->BuildAttribute('外部无法点击');

        $builder->BuildButton('确认');
        $builder->BuildButton('取消');

        $builder->BuildTitle($title);
        $builder->BuildContent($content);
```

```
100
101         return $builder;
102     }
103 }
104
105 $d1 = new DialogDirector();
106 $d1->Construct('窗口1', '确认要执行操作A吗？')->GetDialog()->ShowDialog();
107
108 $d2 = new ModalDialogDirector();
109 $d2->Construct('窗口2', '确认要执行操作B吗？')->GetDialog()->ShowDialog();
110
```

> 说明

- 这回我们的产品就有点复杂了，有标题、内容、属性、按扭等
- 建造过程其实都一样，但这里我们主要使用了不同的构造器。普通对话框外面的东西是可以点击的，而模态窗口一般会有遮罩层，就是背景变成透明黑，而且外面的东西是不能再点击的
- 如果每次都直接通过构造方法去实例化窗口类，那要传递的参数会很多，而现在这样，我们就可以通过建造者来进行组合，让对象具有多态的效果，能够呈现不同的形态及功能