PHP设计模式之抽象工厂模式

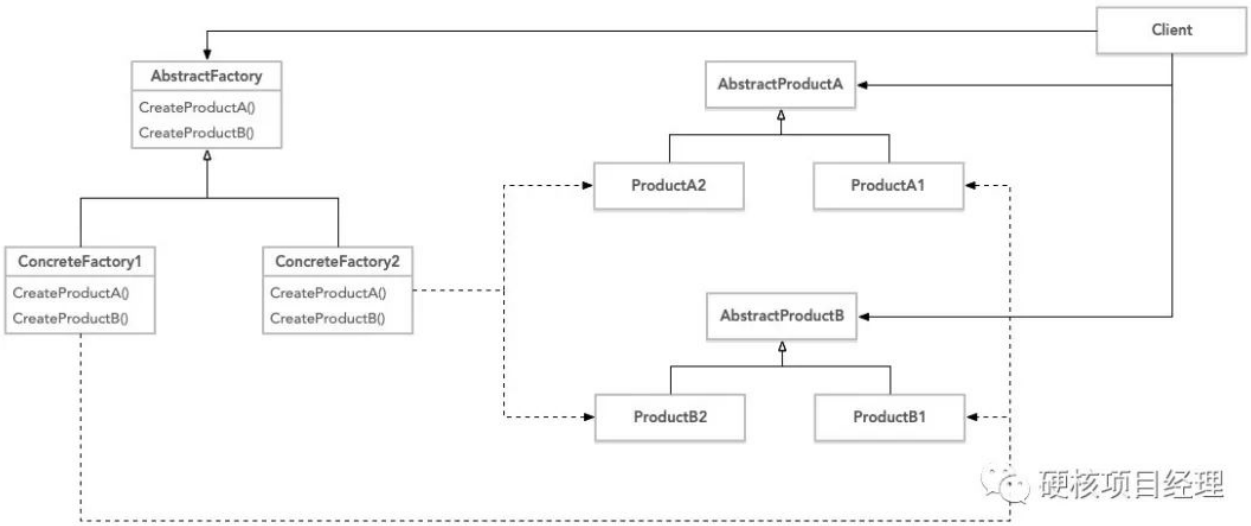
工厂模式系列中的重头戏来了，没错，那正是传闻中的**抽象工厂模式**。初次听到这个名字的时候你有什么感觉？反正我是感觉这货应该是非常高大上的，毕竟包含着“抽象”两个字。话说这两个字在开发中真的是有点高大上的感觉，一带上抽象两字就好像哪哪都很厉害了呢。不过，**抽象工厂**也确实可以说是工厂模式的大哥大。

Gof类图及解释

其实只要理解了工厂方法模式，就很容易明白抽象工厂模式。怎么说呢？还是一样的延迟到子类，还是一样的返回指定的对象。只是抽象工厂里面不仅仅只返回一个对象，而是返回一堆。

GoF定义：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

> GoF类图



- 左边是两个工厂1和2，都继承一个抽象工厂，都实现了CreateProductA和CreateProductB方法
- 工厂1生产的是ProductA1和ProductB1
- 同样的，工厂2生产的是ProductA2和ProductB2

> 代码实现

```php
// 商品A抽象接口
interface AbstractProductA
{
    public function show(): void;
}

// 商品A1实现
class ProductA1 implements AbstractProductA
```

```
{
    public function show(): void
    {
        echo 'ProductA1 is Show!' . PHP_EOL;
    }
}
// 商品A2实现
class ProductA2 implements AbstractProductA
{
    public function show(): void
    {
        echo 'ProductA2 is Show!' . PHP_EOL;
    }
}

// 商品B抽象接口
interface AbstractProductB
{
    public function show(): void;
}
// 商品B1实现
class ProductB1 implements AbstractProductB
{
    public function show(): void
    {
        echo 'ProductB1 is Show!' . PHP_EOL;
    }
}
// 商品B2实现
class ProductB2 implements AbstractProductB
{
    public function show(): void
    {
        echo 'ProductB2 is Show!' . PHP_EOL;
    }
}
```

商品的实现，东西很多吧，这回其实是有四件商品了分别是A1、A2、B1和B2，他们之间假设有这样的关系，A1和B1是同类相关的商品，B1和B2是同类相关的商品

```php
// 抽象工厂接口
interface AbstractFactory
{
    // 创建商品A
    public function CreateProductA(): AbstractProductA;
    // 创建商品B
    public function CreateProductB(): AbstractProductB;
}

// 工厂1，实现商品A1和商品B1
class ConcreteFactory1 implements AbstractFactory
{
    public function CreateProductA(): AbstractProductA
    {
        return new ProductA1();
    }
    public function CreateProductB(): AbstractProductB
    {
        return new ProductB1();
    }
}

// 工厂2，实现商品A2和商品B2
class ConcreteFactory2 implements AbstractFactory
{
    public function CreateProductA(): AbstractProductA
    {
        return new ProductA2();
    }
    public function CreateProductB(): AbstractProductB
    {
        return new ProductB2();
    }
}
```

而我们的工厂也是工厂1和工厂2，工厂1生产的是A1和B1这两种相关联的产品，工厂2生产的是A2和B2这两种商品。好吧，我知道这里还是有点抽象，可能还是搞不懂为什么要这样，我们继续以手机生

产来举例。

我们的手机品牌起来了，所以周边如手机膜、手机壳也交给了富X康（AbstractFactory）来帮我搞定。上回说到，我已经有几款不同类型的手机了，于是还是按原来那样，衡阳工厂（Factory1）生产型号1001的手机（ProductA1），同时型号1001手机的手机膜（ProductB1）和手机壳（ProductC1）也是衡阳工厂生产出来。而型号1002的手机（ProductA2）还是在郑州工厂（Factory2），这个型号的手机膜（ProductB2）和手机膜（ProductC2）也就交给他们来搞定吧。于是，我还是只去跟总厂下单，他们让不同的工厂给我生产了一整套的手机产品，可以直接卖套装咯！！

完整代码：抽象工厂模式

实例

是不是看得还是有点晕。其实说简单点，真的就是在一个工厂类中通过不同的方法返回不同的对象而已。让我们再次用发短信的实例来讲解吧！

场景：这次我们有个业务需求是，不仅要发短信，还要同时发一条推送。短信的目的是通知用户有新的活动参加，而推送不仅通知有新的活动，直接点击就可以进去领红包了，是不是很兴奋。还好之前我们的选择的云服务供应商都是即有短信也有推送接口的，所以我们就直接用抽象工厂来实现吧！

> 短信发送类图

```php
<?php

interface Message {
    public function send(string $msg);
}

class AliYunMessage implements Message{
    public function send(string $msg){
        // 调用接口，发送短信
        // xxxxx
        return '阿里云短信（原阿里大鱼）发送成功！短信内容：' . $msg;
    }
}

class BaiduYunMessage implements Message{
    public function send(string $msg){
        // 调用接口，发送短信
```

```
        // xxxxx
        return '百度SMS短信发送成功！短信内容：' . $msg;
    }
}

class JiguangMessage implements Message{
    public function send(string $msg){
        // 调用接口，发送短信
        // xxxxx
        return '极光短信发送成功！短信内容：' . $msg;
    }
}

interface Push {
    public function send(string $msg);
}

class AliYunPush implements Push{
    public function send(string $msg){
        // 调用接口，发送客户端推送
        // xxxxx
        return '阿里云Android&iOS推送发送成功！推送内容：' . $msg;
    }
}

class BaiduYunPush implements Push{
    public function send(string $msg){
        // 调用接口，发送客户端推送
        // xxxxx
        return '百度Android&iOS云推送发送成功！推送内容：' . $msg;
    }
}

class JiguangPush implements Push{
    public function send(string $msg){
        // 调用接口，发送客户端推送
        // xxxxx
        return '极光推送发送成功！推送内容：' . $msg;
    }
}
```

```php


interface MessageFactory{
    public function createMessage();
    public function createPush();
}

class AliYunFactory implements MessageFactory{
    public function createMessage(){
        return new AliYunMessage();
    }
    public function createPush(){
        return new AliYunPush();
    }
}

class BaiduYunFactory implements MessageFactory{
    public function createMessage(){
        return new BaiduYunMessage();
    }
    public function createPush(){
        return new BaiduYunPush();
    }
}

class JiguangFactory implements MessageFactory{
    public function createMessage(){
        return new JiguangMessage();
    }
    public function createPush(){
        return new JiguangPush();
    }
}

// 当前业务需要使用阿里云
$factory = new AliYunFactory();
// $factory = new BaiduYunFactory();
// $factory = new JiguangFactory();
$message = $factory->createMessage();
$push = $factory->createPush();
```

```
98 echo $message->send('您已经很久没有登录过系统了，记得回来哦！');
99 echo $push->send('您有新的红包已到帐，请查收！');
100
```

完整源码：短信发送工厂方法

> 说明

- 是不是很清晰了？
- 没错，我们有两个产品，一个是Message，一个是Push，分别是发信息和发推送
- 抽象工厂只是要求我们的接口实现者必须去实现两个方法，返回发短信和发推送的对象
- 你说我只想发短信不想发推送可以吗？当然可以啦，不去调用createPush()方法不就行了
- 抽象工厂最适合什么场景？很明显，一系列相关对象的创建
- 工厂方法模式是抽象工厂的核心，相当于多个工厂方法被放到一个大工厂中生产一整套产品（包含周边）而不是一件单独的产品