PHP设计模式之工厂方法模式

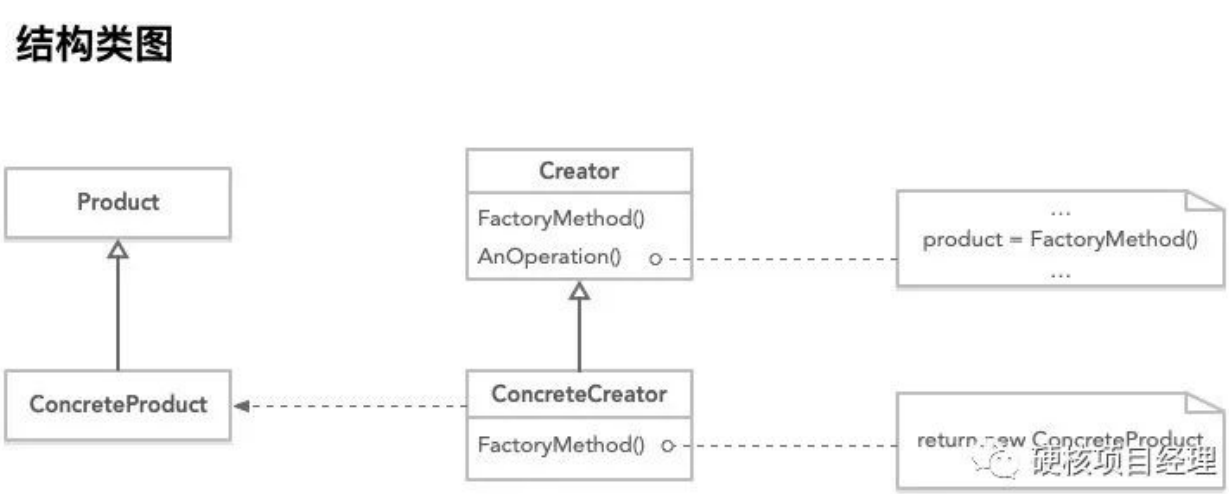
上回说到，简单工厂不属于GoF的二十三种设计模式，这回可就来真家伙了，大名顶顶的**工厂方法模式**前来报道！

GoF类图解释

工厂方法模式对比简单工厂来说，最核心的一点，其实就是将实现推迟到子类。怎么理解呢？我们可以将上回的简单工厂当做父类，然后有一堆子类去继承它。createProduct()这个方法在父类中也变成一个抽象方法。然后所有的子类去实现这个方法，不再需要用switch去判断，子类直接返回一个实例化的对象即可。

GoF定义：定义一个用于创建对象的接口，让子类决定实例化哪一个类。Factory Method使一个类的实例化推迟到其子类。

> GoF类图



- 类图中的Product为产品
- 类图中的Creator为创建者
- 创建者父类有一个抽象的FactoryMethod()工厂方法
- 所有创建者子类需要实现这个工厂方法，返回对应的具体产品
- 创建者父类可以有一个AnOperation()操作方法，直接返回product，可以使用FactoryMethod()去返回，这样外部只需要统一调用AnOperation()

> 代码实现

首先是商品相关的接口和实现类，和简单工厂的类似：

```
// 商品接口
interface Product{
    function show() : void;
```

```
}

// 商品实现类A
class ConcreteProductA implements Product{
    public function show() : void{
        echo "I'm A.\n";
    }
}
```

接下来是创建者的抽象和实现类：

```
// 创建者抽象类
abstract class Creator{

    // 抽象工厂方法
    abstract protected function FactoryMethod() : Product;

    // 操作方法
    public function AnOperation() : Product{
        return $this->FactoryMethod();
    }
}

// 创建者实现类A
class ConcreteCreatorA extends Creator{
    // 实现操作方法
    protected function FactoryMethod() : Product{
        return new ConcreteProductA();
    }
}
```

这里和简单工厂就有了本质的区别，我们去掉了恶心的switch，让每个具体的实现类来进行商品对象的创建。没错，单一和封闭，每个单独的创建者子类只在工厂方法中和一个商品有耦合，有没有其他商品和其他的工厂来跟客户合作过这个子类完全不知道。

同样还是拿手机来比喻：我是一个卖手机的批发商（客户Client，业务方），我需要一批手机（产品ProductA），于是我去让富X康（工厂Creator）来帮我生产。我跟富士康说明了需求，富士康说好的，让我的衡阳工厂（ConcreteCreatorA）来搞定，不需要总厂上，你这小单子，洒洒水啦。然后

过了一阵我又需要另一种型号的手机（产品ProductB），富士康看了看后又让郑州富士康（ConcreteCreatorB）来帮我生产。反正不管怎么样，他们总是给了我对应的手机。而且郑州工厂并不知道衡阳工厂生产过什么或者有没有跟我合作过，这一切只有我和总工厂知道。

完整代码：工厂方法模式

实例

场景：光说不练假把式，把上回的短信发送改造改造，我们依然还是使用上回的那几个短信发送商。毕竟大家已经很熟悉了嘛，不过以后要更换也说不定，商场如战场，大家还是利益为先。这样的话，我们通过工厂方法模式来进行解耦，就可以方便的添加修改短信提供商咯。

> 短信发送类图

> 代码实现

```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;
    }
}


abstract class MessageFactory{
    abstract protected function factoryMethod();
    public function getMessage(){
        return $this->factoryMethod();
    }
}

class AliYunFactory extends MessageFactory{
    protected function factoryMethod(){
        return new AliYunMessage();
    }
}

class BaiduYunFactory extends MessageFactory{
    protected function factoryMethod(){
        return new BaiduYunMessage();
    }
}

class JiguangFactory extends MessageFactory{
    protected function factoryMethod(){
        return new JiguangMessage();
    }
}

// 当前业务需要使用百度云
$factory = new BaiduYunFactory();
$message = $factory->getMessage();
echo $message->send('您有新的短消息，请查收');
```

完整源码：短信发送工厂方法

> 说明

- 和类图完全一致，基本不需要什么说明了吧，注意工厂方法模式的特点，实现推迟到了子类！！
- 业务调用的时候需要耦合一个Factory子类。确实是这样，如果你想一个统一的出口来调用，请在外面加一层简单工厂就好啦，这就当成一道思考题吧
- 不拘泥于目前的形式，可以不用抽象类，直接用一个接口来定义工厂方法，摒弃掉getMessage()方法，外部直接调用公开的模板方法(factoryMethod)即可