链滴

# 设计模式学习笔记之工厂模式

作者：hjljy
原文链接：https://ld246.com/article/1563204424283
来源网站：链滴

# 前言

这是一篇学习笔记，内容很多是来源于网上的资料，然后按照自己学习情况进行的总结，有些是自身感受，有些是网上比较好的资料的引用。

我的个人博客：海加尔金鹰

# 什么是工厂模式

在进行学习之前，我是只知道有工厂模式，但是在查阅了资料之后，工厂模式还是有很好几种的。

## 1. 简单工厂模式

### 1.1. 定义

简单工厂模式又称静态工厂模式，不属于23种GOF模式之一，属于创建型模式，由一个工厂类根据传的参数来返回对应类的实例或者根据调用工厂类的创建方法创建对应的实例。

### 1.2. 核心结构

- Factory （工厂类，实例创建者）：负责具体实例的创建，提供给外部调用者一个实例创建方法。
- Product （实例抽象接口）：所有需要工厂类创建的实例都应该实现了这个接口的。
- Product Bean （具体实例对象）：实现抽象接口，通过工厂类返回给调用者。

### 1.3. 代码实现

第一步：创建一个实例抽象接口

```
public interface Phone {
    void call();
}
```

第二部：创建多个具体实例对象

```
public class HuaWeiPhone implements Phone{
    @Override
    public void call() {
        System.out.println("为华为打Call");
    }
}

public class XiaoMiPhone implements  Phone {
    @Override
    public void call() {
        System.out.println("为小米打Call");
    }
}
```

第三步：创建工厂类

```
public class PhoneFactory {
    //方式一  通过静态方法创建对应的对象
    public static HuaWeiPhone createHuaWeiPhone(){
        return  new HuaWeiPhone();
    }
    public static XiaoMiPhone createXiaoMiPhone(){
        return  new XiaoMiPhone();
    }
    //方式二 通过参数创建对象
    public Phone  createPhone(String name){
        Phone phone = null;
        if("HuaWei".equals(name)){
            phone= new HuaWeiPhone();
        }else if("XiaoMi".equals(name)){
            phone= new XiaoMiPhone();
        }else{
            System.out.println("请输入正确的手机品牌");
        }
        return phone;
    }
}
```

第四步：创建DEMO进行测试

```
public class SimpleFactoryDemo {
    public static void main(String[] args) {
        //方式一 静态方法调用
        HuaWeiPhone huaWeiPhone = PhoneFactory.createHuaWeiPhone();
        XiaoMiPhone xiaoMiPhone = PhoneFactory.createXiaoMiPhone();
        huaWeiPhone.call();
        xiaoMiPhone.call();
        //方式二 方法参数调用
        PhoneFactory factory = new PhoneFactory();
```

```
        Phone huaWei = factory.createPhone("HuaWei");
        Phone xiaoMi = factory.createPhone("XiaoMi");
        huaWei.call();
        xiaoMi.call();
    }
}
```

第五步：查看结果

```
为华为打Call
为小米打Call
为华为打Call
为小米打Call
```

## 1.4. 优缺点

优点：

一个类的创建和使用被分离开来，降低了代码的耦合度。
当该类创建方式变化的时候，不必修改代码中该类所有的创建，只需修改工厂类当中的创建方式。

缺点：

如果需要在方法里写很多与对象创建有关的业务代码，而且需要的创建的对象还不少的话，我们要在个简单工厂类里编写很多个方法，每个方法里都得写很多相应的业务代码，而每次增加子类或者删除类对象的创建都需要打开这简单工厂类来进行修改。这会导致这个简单工厂类很庞大臃肿、耦合性高而且增加、删除某个子类对象的创建都需要打开简单工厂类来进行修改代码也违反了开-闭原则[1]

---

# 2. 工厂方法模式

## 2.1. 定义

由一个工厂接口，和若干个子类工厂组成，将具体的对象创建过程放在子类工厂当中。

## 2.2. 核心结构

抽象工厂（Abstract Factory）：提供了创建产品的接口，调用者通过它访问具体工厂的工厂方法 neProduct() 来创建产品。

具体工厂（ConcreteFactory）：主要是实现抽象工厂中的抽象方法，完成具体产品的创建。

抽象产品（Product）：定义了产品的规范，描述了产品的主要特性和功能。

具体产品（ConcreteProduct）：实现了抽象产品角色所定义的接口，由具体工厂来创建，它同具体厂之间一一对应。[2]

个人感觉对比简单工厂模式，由一个工厂类变成了多个工厂类，同时都实现了共同的抽象工厂，是JAA里面多态特征的典型应用场景。

## 2.3. 代码实现

第一步：创建抽象产品

```
public interface Phone {
    void call();
}
```

第二步：创建抽象工厂

```
public interface AbstractFactory {
    Phone createPhone();
}
```

第三步：创建具体产品

```
public class HuaWeiPhone implements Phone{
    @Override
    public void call() {
        System.out.println("为华为打Call");
    }
}

public class XiaoMiPhone implements  Phone {
    @Override
    public void call() {
        System.out.println("为小米打Call");
    }
}
```

第四步：创建具体工厂类

```
public class HuaWeiFactory implements AbstractFactory {
    @Override
    public Phone createPhone() {
        return new HuaWeiPhone();
    }
}

public class XiaoMiFactory implements AbstractFactory {
    @Override
    public Phone createPhone() {
        return new XiaoMiPhone();
    }
}
```

第五步：创建测试DEMO并查看结果

```
public class MethodFactoryDemo {
    public static void main(String[] args) {
        //  通过华为工厂生产的手机是华为手机
        HuaWeiFactory  huaWeiFactory = new HuaWeiFactory();
        huaWeiFactory.createPhone().call();
        // 通过小米工厂生产的手机是小米手机
        XiaoMiFactory xiaoMiFactory = new XiaoMiFactory();
        xiaoMiFactory.createPhone().call();
    }
```

```
}
```

输出结果：

```
为华为打Call
为小米打Call
```

## 2.4. 优缺点

优点：包含了简单工厂模式的所有优点，并且符合了开闭原则，当需要新增一个产品的时候，不需要改原有的结构代码，只需要新增一个具体的工厂类就行了。

缺点：代码量变多，类的组织结构变复杂了。

## 2.5. 工厂方法模式和简单工厂模式的对比

简单工厂模式：一个工厂搞定所有的产品创建，就相当于一个人负责一个项目，全栈工程师。

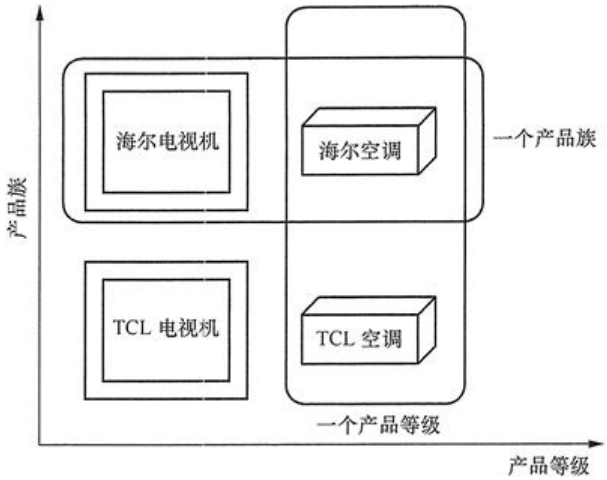
工厂方法模式：产品分给不同的工厂进行创建，就相当于一个项目，有前端，有后端，有UI等等，只责特定的部分。

# 3. 抽象工厂模式

## 3.1. 定义

抽象工厂模式: 提供一个创建一系列相关或相互依赖对象的接口, 而无需指定他们具体的类

产品族 : 将同一个具体工厂所生产的位于不同等级的一组产品称为一个产品族，图 1 所示的是海尔工和 TCL 工厂所生产的电视机与空调对应的关系图[3]




图片引用自：http://c.biancheng.net/view/1351.html

## 3.2. 核心结构

其核心结构和工厂方法模式一样。工厂方法模式是一种特殊的抽象工厂模式，就像正方形和长方形的系。

工厂方法模式只有一个抽象工厂，抽象工厂模式则是多个抽象工厂，每个抽象工厂代表一个产品族。

## 3.3. 代码实现

第一步： 在上面工厂方法模式的代码基础上新增产品接口类：

```
public interface NoteBook {
    void show();
}
```

第二步： 创建两个具体产品类

```
public class HuaWeiNoteBook implements NoteBook {
    @Override
    public void show() {
        System.out.println("华为笔记本");
    }
}

public class XiaoMiNoteBook implements NoteBook {
    @Override
    public void show() {
        System.out.println("小米笔记本");
    }
}
```

第三步：修改抽象工厂类以及对应工厂类的实现

注：此时一个抽象工厂类就是一个产品族。

```
public interface AbstractFactory {
     Phone CreatePhone();
     //新增加了一个笔记本类的创建
     NoteBook CreateNoteBook();
}

public class HuaWeiFactory implements AbstractFactory {

    @Override
    public Phone CreatePhone() {
        return new HuaWeiPhone();
    }

    @Override
    public NoteBook CreateNoteBook() {
        return new HuaWeiNoteBook();
    }
}

public class XiaoMiFactory  implements AbstractFactory{

    @Override
    public Phone CreatePhone() {
        return new XiaoMiPhone();
    }
```

```
    @Override
    public NoteBook CreateNoteBook() {
        return new XiaoMiNoteBook();
    }
}
```

### 3.4. 优缺点

优点：

1 类的创建与使用分离，耦合性降低。

2 具体工厂类切换方便，不同的产品之间进行切换非常方便。

缺点：

并不完全符合开闭原则。在上述代码当中新增一个产品，就需要对所有类进行修改，如果新增加一个品族，则不需要进行任何修改。

代码量更多，结构比较臃肿。

## 工厂模式和建造者模式的区别

工厂模式：专注的是实例的创建。

建造者模式：专注的是实例内部属性的构建。

## 后记

很多框架都使用到了工厂模式：

spring

quartz

shiro

mybatis 里面的DatasourceFactory

.....

到这里基本上创建型模式就学习完毕了。感觉创建型模式主要是对实例的创建进行的各种优化，提供种比较优秀的，高效的创建方式。

1. : 简单工厂模式、工厂模式以及抽象工厂模式
2. : 工厂方法模式（详解版）
3. :抽象工厂模式（详解版）