链滴

工厂模式之简单工厂

作者：leap

原文链接：https://ld246.com/article/1501429076408

来源网站：链滴

工厂模式之简单工厂

什么是工厂模式?

工厂模式分为**简单工厂模式**，**工厂方法模式**，**抽象工厂模式**。复杂度由低到高，分别解决了不同场景下**象创建**的问题。

为什么需要工厂模式?

很多时候我们通过反射就可以非常灵活的创建对象，那么为什么还要使用工厂模式？在Java中创建对的方法通常有以下几种：

1. 通过 new关键字直接创建对象。
2. 通过 反射创建对象。
3. 通过 clone创建对象。
4. 通过 工厂模式创建对象。

通过new关键字来创建对象虽然很方便但是灵活性很差，考虑一个简单的例子：

```
class A {
    //声明一个B类型的对象，其拥有method方法
    private B b;
    //类初始化时创建一个对象
    public A (){
        //C为B的实现，B是一个接口
        this.b = new C();
    }
    //使用对象
    public void foo(){
        b.method();
    }
}
```

简单工厂

这段代码看上去没什么问题但是如果你想要替换一个B的实现就不得不修改A类的代码，违反了开闭原则那么有什么办法解决这个问题呢？在回答这个问题之前，首先来看一下上面这个例子中类A做了哪些情。第一步它创建了一个B的实现C，然后在foo方法中调用了对象方法method。简言之A类既负责象的创建也负责对象的使用。这是一个非常不好的现象，因为这样做导致了A类的职责过重。**与一个象相关的职责通常有三类：对象本身所具有的职责、创建对象的职责和使用对象的职责。对象本身的责比较容易理解，就是对象自身所具有的一些数据和行为，可通过一些公开的方法来实现它的职责。** 在A中替换实现实际上是对对象创建过程的修改。让替换不影响A类就需要把A类的对象创建职责分离出，这样A类只负责对象的使用，符合单一职责原则，有利于功能的复用和系统的维护。如何分离呢？里就用到了工厂模式，针对上面的例子我们使用简单工厂来分离对象的创建过程。

```
class Factory {
    public static B getB(String type) {
        if(StringUtil.equals(type,type1)){
                return new C();
```

```
        }else
            if(StringUtil.equals(type,type2)) {
                return new D();
            }
            ...
    }
}

class A {
    private B b;
    public A (String type){
        this.b = Factory.getB(type);
    }
    public void foo(){
        b.method();
    }
}
```

这样一来如果需要替换B的实现，只需要修改工厂类中的静态工厂方法，然后通过不同的参数来切换B实现。使用工厂模式还有一个好处就是如果在创建对象的过程中还需要执行一些初始化或者参数设置操作就可以把这些过程集中在工厂中，而不是让它散落在各处，不仅重复而且难以维护。

在Java中重载的构造函数的使用只能通过参数列表的不同来区分。如果需要使用不同的构造函数来创对象，工厂模式提供了一个方便，我们可以提供多个名字不同的工厂方法来对应不同的构造函数。这使用的方便性和代码的可读性会更好。

思考

在简单工厂又称静态工厂模式中，如果我们需要增加接口的具体实现，不得不修改工厂方法，这显然符合开闭原则，那么有什么解决方法呢?

总结

简单工厂解决了单一类职责过重的问题，将对象的创建过程分离到工厂类中，但是造成了工厂类的复性。

工厂方法中充斥这大量条件判断和初始化语句。随着具体实现种类的增加，工厂方法势必越来越复杂变的难以维护。还有一个很重要的问题，如果工厂方法不可用，那么所有涉及的对象使用都将发生错。这将是一个很大的风险。