链滴

工厂方法模式

作者：kevin2020
原文链接：https://ld246.com/article/1513780866981
来源网站：链滴

当系统中需要引入新产品时，由于静态工厂方法通过所传入参数的不同来创建不同的产品，这必定要改工厂类的源代码，将违背“开闭原则”，如何实现增加新产品而不影响已有代码？

工厂方法模式便应运而生

定义

工厂方法(Factory Method)模式的意义是定义一个创建产品对象的工厂接口，将实际创建工作推迟到类当中。核心工厂类不再负责产品的创建，这样核心类成为一个抽象工厂角色，仅负责具体工厂子类须实现的接口，这样进一步抽象化的好处是使得工厂方法模式可以使系统在不修改具体工厂角色的情下引进新的产品。

工厂方法模式是简单工厂模式的延伸，它继承了简单工厂模式的优点，同时还弥补了简单工厂模式的足。工厂方法模式是使用频率最高的设计模式之一，是很多开源框架和API类库的核心模式。

示例

```
定义产品接口
public interface Phone {
    void chat();
    void watch();
}

public class ApplePhone implements Phone {

    @Override
    public void chat() {
        System.out.println("使用ApplePhone进行通话");
    }

    @Override
    public void watch() {
        System.out.println("使用ApplePhone看视频");
    }

}

public class SamsungPhone implements Phone{

    @Override
    public void chat() {
        System.out.println("使用SamPhone进行通话");
    }

    @Override
    public void watch() {
        System.out.println("使用SamPhone看视频");
    }

}

定义工厂接口
public abstract class PhoneFactory {
```

```
    abstract Phone createPhone();
}

public class ApplePhoneFactory extends PhoneFactory{

    @Override
    Phone createPhone() {
        return new ApplePhone();
    }

}

public class SamsungPhoneFactory extends PhoneFactory{

    @Override
    Phone createPhone() {
        return new SamsungPhone();
    }

}
```

客户端调用

```
public class PhoneClient {
    public static void main(String[] args) {
        PhoneFactory factory=new ApplePhoneFactory();
        Phone phone=factory.createPhone();
        phone.chat();
        phone.watch();
    }
}
```

我们发现，如果要创建不同的手机，客户端还得修改代码，这样还是没有完全遵循开闭原则

我们可以简单优化一下啊，通过配置文件的方式控制生成的产品，这样就不用修改源码了

```
public class XmlConfigUtils {
public static String getConfigPhone() throws Exception{
        File file=new File("./src/factorymethod/phoneFactory.xml");
        DocumentBuilder builder=DocumentBuilderFactory.newInstance().newDocumentBuilder()

        Document document=builder.parse(file);
        String phone=document.getElementsByTagName("className").item(0).getFirstChild().ge
NodeValue();
        return phone;

    }
```

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<className>factorymethod.ApplePhoneFactory</className>
</config>
```

```
}

public class PhoneClient {
```

```
    public static void main(String[] args) throws ClassNotFoundException, Exception {
        PhoneFactory factory=(PhoneFactory) Class.forName(XmlConfigUtils.getConfigPhone()).
ewInstance();
        Phone phone=factory.createPhone();
        phone.chat();
        phone.watch();
    }
}
```

这里还牵扯到了绝对路径的问题，有兴趣可以去看我Tip(Java)下的关于绝对路径的讲解。

总结

1. 主要优点

(1) 在工厂方法模式中，工厂方法用来创建客户所需要的产品，同时还向客户隐藏了哪种具体产品类被实例化这一细节，用户只需要关心所需产品对应的工厂，无须关心创建细节，甚至无须知道具体产类的类名。

(2) 基于工厂角色和产品角色的多态性设计是工厂方法模式的关键。它能够让工厂可以自主确定创建种产品对象，而如何创建这个对象的细节则完全封装在具体工厂内部。工厂方法模式之所以又被称为态工厂模式，就正是因为所有的具体工厂类都具有同一抽象父类。

(3) 使用工厂方法模式的另一个优点是在系统中加入新产品时，无须修改抽象工厂和抽象产品提供的口，无须修改客户端，也无须修改其他的具体工厂和具体产品，而只要添加一个具体工厂和具体产品可以了，这样，系统的可扩展性也就变得非常好，完全符合“开闭原则”。

2. 主要缺点

(1) 在添加新产品时，需要编写新的具体产品类，而且还要提供与之对应的具体工厂类，系统中类的数将成对增加，在一定程度上增加了系统的复杂度，有更多的类需要编译和运行，会给系统带来一些外的开销。

(2) 由于考虑到系统的可扩展性，需要引入抽象层，在客户端代码中均使用抽象层进行定义，增加了统的抽象性和理解难度，且在实现时可能需要用到DOM、反射等技术，增加了系统的实现难度。

3. 适用场景

(1) 客户端不知道它所需要的对象的类。在工厂方法模式中，客户端不需要知道具体产品类的类名，需要知道所对应的工厂即可，具体的产品对象由具体工厂类创建，可将具体工厂类的类名存储在配置件或数据库中。

(2) 抽象工厂类通过其子类来指定创建哪个对象。在工厂方法模式中，对于抽象工厂类只需要提供一创建产品的接口，而由其子类来确定具体要创建的对象，利用面向对象的多态性和里氏代换原则，在序运行时，子类对象将覆盖父类对象，从而使得系统更容易扩展。