正文

在之间文章讲解到简单工厂模式，简单工厂模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对工厂类进行修改，这违背了闭包原则。

所以，从设计角度考虑，有一定的问题，如何解决？就用到工厂方法模式，创建一个工厂接口和创建多个工厂实现类，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码。

一、什么是工厂方法模式

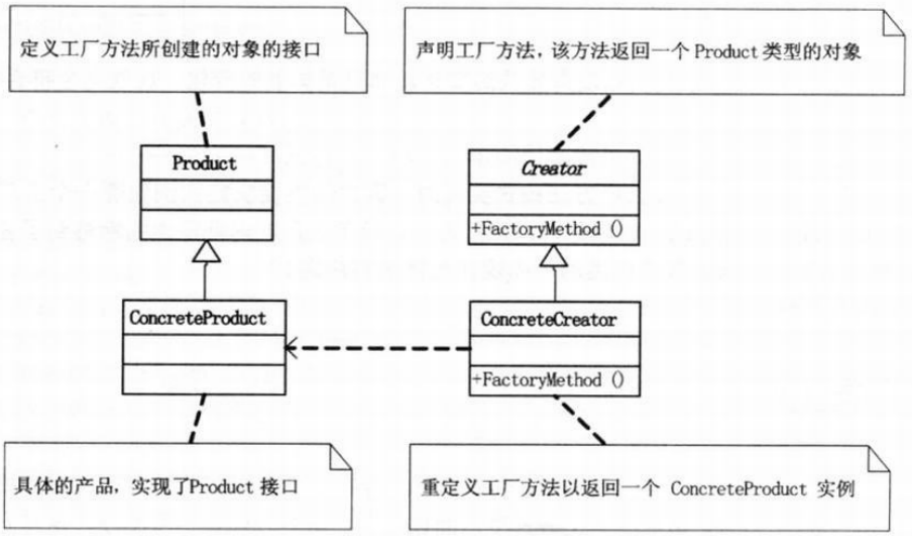
工厂方法模式(Factory Method Pattern)，也叫虚拟构造器(Virtual Constructor)模式或者多态工厂(Polymorphic Factory)模式，它属于类创建型模式。**在工厂方法模式中，工厂父类负责定义创建产品对象的公共接口，而工厂子类则负责生成具体的产品对象，这样做的目的是将产品类的实例化操作延迟到工厂子类中完成，即通过工厂子类来确定究竟应该实例化哪一个具体产品类。**

工厂方法模式是简单工厂模式的进一步抽象和推广。由于使用了面向对象的多态性，工厂方法模式保持了简单工厂模式的优点，而且克服了它的缺点。**在工厂方法模式中，核心的工厂类不再负责所有产品的创建，而是将具体创建工作交给子类去做。这个核心类仅仅负责给出具体工厂必须实现的接口，而不负责哪一个产品类被实例化这种细节，这使得工厂方法模式可以允许系统在不修改工厂角色的情况下引进新产品。**

抽象产品角色（Product）：定义产品的接口
具体产品角色（ConcreteProduct） ：实现接口Product的具体产品类
抽象工厂角色（Creator） ：声明工厂方法（FactoryMethod），返回一个产品
真实的(具体的)工厂角色（ConcreteCreator）：实现FactoryMethod工厂方法，由客户调用，返回一个产品的实例。

工厂方法模式（Factory Method），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。工厂方法使一个类的实例化延迟到其子类。[DP]

工厂方法模式（Factory Method）结构图



二、工厂方法模式的适用场景

在以下情况下可以使用工厂方法模式：

一个类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，只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；客户端需要知道创建具体产品的工厂类。**

一个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个创建产品的接口，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，在程序运行时，子类对象将覆盖父类对象，从而使得系统更容易扩展。将创建对象的任务委托给多个工厂子类中的某一个，**客户端在使用时可以无须关心是哪一个工厂子类创建产品子类，需要时再动态指定，可将具体工厂类的类名存储在配置文件或数据库中。**

三、工厂方法模式的优缺点

优点

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将被实例化这一细节，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品类的类名。

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。它能够使工厂可以自主确定创建何种产品对象，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多态工厂模式，是因为所有的具体工厂类都具有同一抽象父类。

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接口，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就可以了。这样，系统的可扩展性也就变得非常好，完全符合“开闭原则”。**

缺点
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，系统中类的个数将成对增加，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，会给系统带来一些额外的开销。

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，增加了系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。

四、工厂方法模式和简单工厂模式的区别

简单工厂模式：
（1）工厂类负责创建的对象比较少，由于创建的对象较少，不会造成工厂方法中的业务逻辑太过复杂。
（2）客户端只知道传入工厂类的参数，对于如何创建对象并不关心。

工厂方法模式：
（1）客户端不知道它所需要的对象的类。
（2）抽象工厂类通过其子类来指定创建哪个对象。

五、工厂方法模式的实现

先构建一个工厂接口

```
// 抽象工厂中声明了工厂方法，用于返回一个产品，它是工厂方法模式的核心，任何在模式中创建对象的工厂类都必须实现该接口；
public interface IFactory {
    public Iphone factoryMethod();
}
```

再创建产品接口

```
// 工厂接口
public interface Iphone {
    public void getSize();
}
```

创建具体产品类

```
// 具体的产品
public class IphoneX implements Iphone{
    @Override
    public void getSize() {
        System.out.println("IphoneX 屏幕：3.5英寸");
    }
}

public class IphoneXs implements Iphone{
    @Override
    public void getSize() {
        System.out.println("IphoneXs 屏幕：4.5英寸");
    }
}
```

工厂子类

```
public class IphoneXFactory implements IFactory{

    @Override
    public Iphone factoryMethod() {
        return new IphoneX();
    }
}

// 具体工厂是抽象工厂类的子类，实现了抽象工厂中定义的工厂方法，并可由客户调用，返回一个具体产品类的实例。
public class IphoneXsFactory implements IFactory{
    @Override
    public Iphone factoryMethod() {
        return new IphoneXs();
    }
}
```

客户端测试代码：

```
public class Client {
    public static void main(String[] args) {
```

```
        IFactory iphoneXFactory = new IphoneXFactory();
        Iphone iphoneX = iphoneXFactory.factoryMethod();
        iphoneX.getSize();

        IFactory iphoneXsFactory = new IphoneXsFactory();
        Iphone iphoneXs = iphoneXsFactory.factoryMethod();
        iphoneXs.getSize();
    }
}
```

六、总结

工厂方法模式又称为工厂模式，它属于类创建型模式。在工厂方法模式中，工厂父类负责定义创建产品对象的公共接口，而工厂子类则负责生成具体的产品对象，这样做的目的是将产品类的实例化操作延迟到工厂子类中完成，即通过工厂子类来确定究竟应该实例化哪一个具体产品类。

工厂方法模式包含四个角色：抽象产品是定义产品的接口，是工厂方法模式所创建对象的超类型，即产品对象的共同父类或接口；

具体产品实现了抽象产品接口，某种类型的具体产品由专门的具体工厂创建，它们之间往往一一对应；

抽象工厂中声明了工厂方法，用于返回一个产品，它是工厂方法模式的核心，任何在模式中创建对象的工厂类都必须实现该接口；

具体工厂是抽象工厂类的子类，实现了抽象工厂中定义的工厂方法，并可由客户调用，返回一个具体产品类的实例。

工厂方法模式是简单工厂模式的进一步抽象和推广。由于使用了面向对象的多态性，工厂方法模式保持了简单工厂模式的优点，而且克服了它的缺点。

在工厂方法模式中，核心的工厂类不再负责所有产品的创建，而是将具体创建工作交给子类去做。这个核心类仅仅负责给出具体工厂必须实现的接口，而不负责产品类被实例化这种细节，这使得工厂方法模式可以允许系统在不修改工厂角色的情况下引进新产品。

工厂方法模式的主要优点是增加新的产品类时无须修改现有系统，并封装了产品对象的创建细节，系统具有良好的灵活性和可扩展性；

其缺点在于增加新产品的同时需要增加新的工厂，导致系统类的个数成对增加，在一定程度上增加了系统的复杂性。

工厂方法模式适用情况包括：

一个类不知道它所需要的对象的类；

一个类通过其子类来指定创建哪个对象；将创建对象的任务委托给多个工厂子类中的某一个，客户端在使用时可以无须关心是哪一个工厂子类创建产品子类，需要时再动态指定。