使用过Spring的童鞋知道Spring使用的简单工厂模式，那我们一块看了解下什么是工厂模式吧，先看几个问题

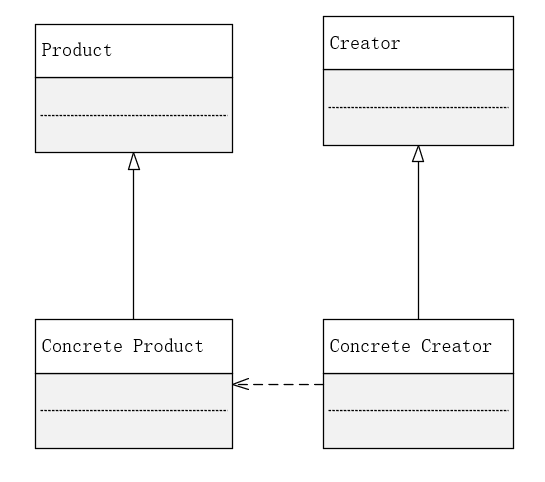
工厂模式有什么作用？

简单工厂模式、工厂方法模式和抽象工厂模式分别有什么区别？

一、工厂模式的定义

定义一个对象的接口，让其子类自己决定实例化哪一个工厂类，工厂模式使其创建过程延迟到子类。

工厂模式的通用类类图，如下



二、工厂模式的优缺点

使用工厂模式有什么好处呢？

1、调用者创建一个对象，不需要自己了解创建过程，只需要知道获取工厂类的参数

2、高扩展性，增加产品，新增一个工厂类即可

3、封装性比较好，结构也比较清晰

工厂模式又会带来什么问题呢？

1、每增加一个产品，需要新增一个产品类和一个工厂类，有较大的开发量

2、产品修改的时候，很可能需要修改对应的工厂类

工厂模式在什么情况下使用呢

1、替代new的创建过程

2、可扩展的框架中

三、实现工厂模式

让我们先来实现一个工厂模式，来体会一下怎么用工厂模式生产产品吧

1、首先需要创建一个产品的抽象类和两个产品

public abstract class Product {  
 public void doSomeThing(){  
 }  
}

public class ConcreteProduct1 extends Product {  
 @Override  
 public void doSomeThing() {  
 System.*out*.println("i am ConcreteProduct1");  
 }  
}

public class ConcreteProduct2 extends Product{  
 @Override  
 public void doSomeThing() {  
 System.*out*.println("i am ConcreteProduct2");  
 }  
}

2、然后创建一个工厂

public class SimpleFactory {  
 public Product createProduce(Class<? extends Product> cl) {  
 try {  
 return cl.newInstance();  
 } catch (InstantiationException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 }  
  
 return null;  
 }  
}

3、创建一个main函数，传递不同参数，获取不同的产品

public static void main(String[] args) {  
 SimpleFactory factory = new SimpleFactory();  
  
 factory.createProduce(ConcreteProduct1.class).doSomeThing();  
  
 factory.createProduce(ConcreteProduct2.class).doSomeThing();  
}

输出结果

i am ConcreteProduct1

i am ConcreteProduct2

四、工厂模式的三种实现方式

工厂模式的实现分为三种：简单工厂模式、工厂方法模式 和抽象工厂模式

一般的文章都是将抽象工厂模式作为一种单独的设计模式来讲解，不过个人认为三种一起比较能更好的理解工厂模式的使用。

1、简单工厂模式

上文代码中实现的就是一个简单工厂模式，那么简单工厂模式有什么好处呢

优点：

1、能够根据外界传递的参数，来控制创建的产品

2、低耦合，将用户和具体的产品类分隔开

看起来挺不错的，实现也简单，不过缺点是什么呢？

缺点：

1、扩展性差，所有生产逻辑都集成到工厂中了，有产品增加或删除的时候，需要重新修改工厂的逻辑

2、不适应多层次的工厂结构

现在了解了优点和缺点，那我们什么时候用简单工厂模式呢

应用场景：

1、用户不关心产品的创建过程

2、产品种类比较少，且比较稳定，不会经常增加或删除

2、方法工厂模式

方法工厂模式，有一个父工厂抽象类，父类不负责生产产品，只控制生产规则。具体的生产工作由其子类来完成，一个产品会有一个具体的工厂，通过切换子工厂来生产不同产品。

具体代码实现

首先还是需要一个产品的抽象类和两个产品（代码可以参考上文）

然后创建一个工厂的抽象类和两个生产工厂

public abstract class ConcreteFactory {  
   
 public abstract Product create();  
}

public class ConcreteFactory1 extends Factory {  
 @Override  
 public Product create() {  
 return new ConcreteProduct1();  
 }  
}

public class ConcreteFactory2 extends Factory {  
 @Override  
 public Product create() {  
 return new ConcreteProduct2();  
 }  
}

运行main函数

public static void main(String[] args) {  
 Factory factory = new ConcreteFactory1();  
 factory.create().doSomeThing();  
  
 factory = new ConcreteFactory2();  
 factory.create().doSomeThing();  
}

运行结果

i am ConcreteProduct1

i am ConcreteProduct2

现在基本了解什么是方法工厂模式了，那么这个和简单工厂模式比又有什么优缺点呢？

优点：

1、需要切换不同工厂就能生产不同产品

2、扩展性好，每增加一个产品，只需要增加对应的工厂即可

缺点：

1、代码量大，每个产品都需要有自己对应的工厂

2、如果大量产品有代码变动，很可能顺带修改大量的工厂代码

那我们什么情况下使用方法工厂模式呢？

使用场景：

1. 对高度可扩展比较依赖的生产环境和框架中

3、抽象工厂模式

抽象工厂模式，一个工厂可以生产多个不同产品，向用户提供一个接口，不指定具体类的情况下，可以生产多个产品组的产品对象。

字面理解起来优点抽象，和名字有点像，先来看下具体实现吧

首先创建两种产品的抽象类，和两个产品的具体实现类

抽象产品类 AbstractProductA

public abstract class AbstractProductA {  
 public abstract void doSomething();  
}

它的两个具体产品类

public class ProductA1 extends AbstractProductA {  
 @Override  
 public void doSomething() {  
 System.*out*.println("A1 实例对象！");  
 }  
}

public class ProductA2 extends AbstractProductA {  
 @Override  
 public void doSomething() {  
 System.*out*.println("A2 实例对象！");  
 }  
}

抽象产品类 AbstractProductB

public abstract class AbstractProductB {  
  
 public abstract void doSomething();  
}

它的两个具体产品类

public class ProductB1 extends AbstractProductB {  
 @Override  
 public void doSomething() {  
 System.*out*.println("B1 实例对象！");  
 }  
}

public class ProductB2 extends AbstractProductB {  
 @Override  
 public void doSomething() {  
 System.*out*.println("B2 实例对象！");  
 }  
}

然后创建工厂的抽象类AbstractFactory

public abstract class AbstractFactory {  
 public abstract AbstractProductA productA();  
  
 public abstract AbstractProductB productB();  
}

两个具体工厂类

public class Factory1 extends AbstractFactory {  
 @Override  
 public AbstractProductA productA() {  
 return new ProductA1();  
 }  
  
 @Override  
 public AbstractProductB productB() {  
 return new ProductB1();  
 }  
}

public class Factory2 extends AbstractFactory {  
 @Override  
 public AbstractProductA productA() {  
 return new ProductA2();  
 }  
  
 @Override  
 public AbstractProductB productB() {  
 return new ProductB2();  
 }  
}

运行main函数

Factory1 factory1 = new Factory1 ();  
  
factory1.productA().doSomething();;  
factory1.productB().doSomething();  
  
  
Factory2 factory2 = new Factory2 ();  
  
  
factory2.productA().doSomething();  
factory2.productB().doSomething();

输出结果

A1 实例对象！

B1 实例对象！

A2 实例对象！

B2 实例对象！

看完代码，抽象工厂模式和其他的工厂模式最大的区别就比较明显了，工厂中有多个方法，每个方法生产不同的产品。

那抽象工厂模式有什么优点和缺点呢

优点：

1、当多个产品组成产品组需要一起生产工作的时候，能保证客户端能使用组合好的产品组

缺点：

扩展性差，如果增加一个新的产品系列，需要从父工厂到子工厂都修改一遍

应用场景：

1、多个产品组合使用的情况，例如网页主题的修改