第十三章 工厂方法模式

工厂方法模式（别名：虚拟构造）

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。Factory Method使一个类的实例化延迟到其子类。

Mediator Pattern(Another Name: Virtual Constructor)

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

一、概述

当系统准备为用户提供某个类的子类的实例，又不想让用户代码和该子类形成耦合时，就可以使用工厂方法模式来设计系统。工厂方法模式的关键是在一个接口或抽象类中定义一个抽象方法，该方法返回某个类的子类的实例，该抽象类或接口让其子类或实现该接口的类通过重写这个抽象方法返回某个子类的实例。

二、工厂方法模式的结构与使用

模式的结构中包括四种角色：

- 抽象产品（Product）
- 具体产品（ConcreteProduct）
- 构造者（Creator）
- 具体构造者（ConcreteCreator）

模式的UML类图

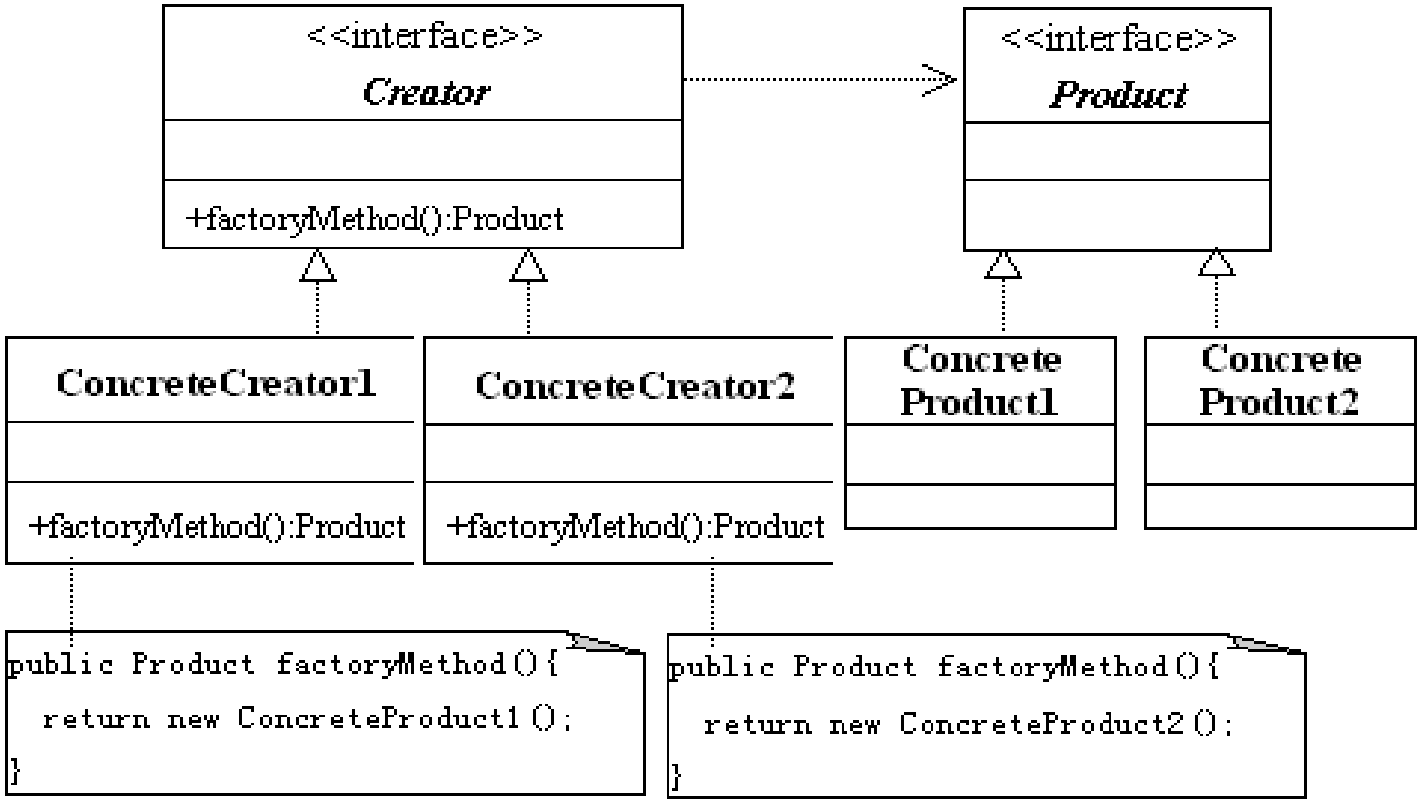


图 13.2　工厂方法模式的类图

模式的结构的描述与使用

1. 抽象产品（Product） ：PenCore.java

```
public abstract class PenCore{
    String color;
    public abstract void writeWord(String s);
}
```

模式的结构的描述与使用

2. 具体产品（ConcreteProduct）_1 ： RedPenCore.java

```
public class RedPenCore extends PenCore{
    RedPenCore(){
      color="红色";
    }
    public void writeWord(String s){
       System.out.println("写出"+color+"的字:"+s);
    }
}
```

模式的结构的描述与使用

2. 具体产品（ConcreteProduct）_2 ： BluePenCore.java

```
public class BluePenCore extends PenCore{
    BluePenCore(){
      color="蓝色";
    }
    public void writeWord(String s){
       System.out.println("写出"+color+"的字:"+s);
    }
}
```

模式的结构的描述与使用

2. 具体产品（ConcreteProduct）_3: BlackPenCore.java

```
public class BlackPenCore extends PenCore{
    BlackPenCore(){
      color="黑色";
    }
    public void writeWord(String s){
       System.out.println("写出"+color+"的字:"+s);
    }
}
```

模式的结构的描述与使用

3．构造者（Creator）:BallPen.java

```
public abstract class BallPen{
    BallPen(){
        System.out.println("生产了一只装有"+getPenCore().color+"笔芯的圆珠笔");
    }
    public abstract PenCore getPenCore(); //工厂方法
}
```

4. 具体构造者（ConcreteCreator）：

RedBallPen.java

```
public class RedBallPen extends BallPen{
    public PenCore getPenCore(){
       return new RedPenCore();
    }
}
```

BlueBallPen.java

```
public class BlueBallPen extends BallPen{
    public PenCore getPenCore(){
       return new BluePenCore();
    }
}
```

BlackBallPen.java

```
public class BlackBallPen extends BallPen{
    public PenCore getPenCore(){
       return new BlackPenCore();
    }
}
```

模式的结构的描述与使用

5. 应用 Application.java

```
public class Application{
    public static void main(String args[]){
        PenCore penCore;
        BallPen ballPen=new BlueBallPen();
        penCore=ballPen.getPenCore();
        penCore.writeWord("你好,很高兴认识你");
        ballPen=new RedBallPen();
        penCore=ballPen.getPenCore();
        penCore.writeWord("How are you");
        ballPen=new BlackBallPen();
        penCore=ballPen.getPenCore();
        penCore.writeWord("nice to meet you");
    }
}
```

三、工厂方法模式的优点

- 使用工厂方法可以让用户的代码和某个特定类的子类的代码解耦。
- 工厂方法使用户不必知道它所使用的对象是怎样被创建的，只需知道该对象有哪些方法即可。

感谢王良芳大神在校内的分享，新增20种模式的形象比喻。缺：简单工厂模式、缺省适配模式和不变模式。

创建模式：简单工厂、工厂方法、抽象工厂、单例、建造、模型；
结构模式：适配器、缺省适配、合成、装饰、代理、享元、门面、桥梁；
行为模式：不变、策略、模板方法、观察者、迭代子、责任链、命令
备忘录、状态、访问者、解释器、调停者。(最后三种不讲)

1、工厂方法模式

主讲：田旭园
程序：奚亮亮
ppt：叶良波
答问：陈才国

FACTORY METHOD—请MM去麦当劳吃汉堡，不同的MM有不同的口味，要每个都记住是一件烦人的事情，我一般采用Factory Method模式，带着MM到服务员那儿，说"要一个汉堡"，具体要什么样的汉堡呢，让MM直接跟服务员说就行了。

工厂方法模式：核心工厂类不再负责所有产品的创建，而是将具体创建的工作交给子类去做，成为一个抽象工厂角色，仅负责给出具体工厂类必须实现的接口，而不接触哪一个产品类应当被实例化这种细节。

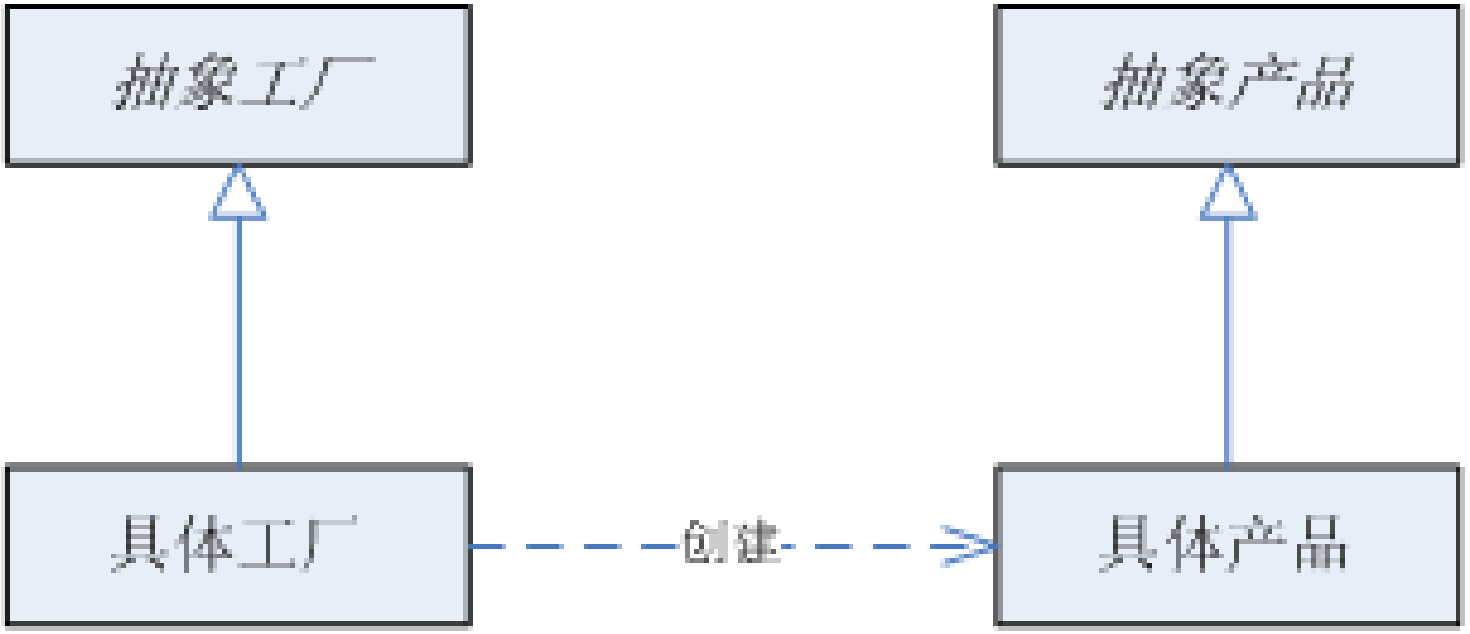
工厂方法模式是类的创建模式，又叫虚拟构造子（Virtual Constructor）模式或者多态性工厂（Polymorphic Factory）模式。

工厂方法模式的用意是定义一个创建产品对象的工厂接口，将实际工作推迟到子类中。

工厂方法解决问题：

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。

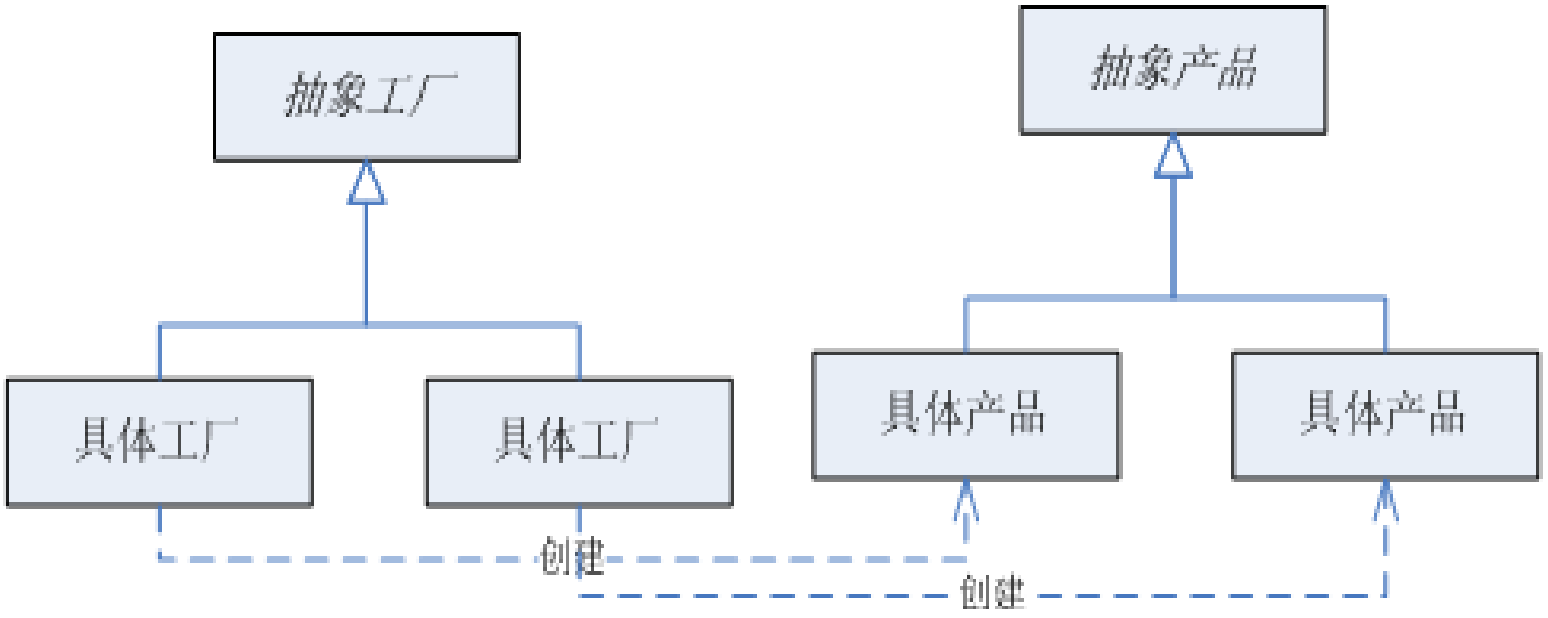
工厂方法缩略图



该模式的优点：

这种抽象的结果，使这种工厂方法模式可以用来允许系统不修改具体工厂角色的情况下引进新产品，这一特点无疑使得工厂模式具有超过简单工厂模式的优越性。

在工厂方法模式中，一般都有一个平行的等级结构，也就是说工厂和产品是对应的的。抽象工厂对应抽象产品，具体工厂对应具体产品。简单的示意图如下：



各种角色分类

抽象工厂角色:

具体工厂角色:

抽象产品角色:

具体产品角色:

2、简单工厂模式

主讲人：陈儒
组员：韩政高、戴鹏军、陈群

简单的介绍

- 简单工厂模式是创建型模式，用于对象的创建，它不属于23种gof设计模式。它是工厂模式家族中最简单实用的模式，可以理解为是不同工厂模式的一个特殊实现。

- 设计模式描述了软件设计过程中某一类常见问题的一般性的解决方案，是解决某个方向上的变动需求的问题。而工厂设计模式的存在是为了解决哪一方面的问题呢？或者说它的动机是什么呢？

动机

- 在软件系统中，经常面临着“某个对象”的创建工作；由于需求的变化，这个对象经常面临着剧烈的变化，但是它却拥有比较稳定的接口。

- 如何应对这种变化？如何提供一种“封装机制”来隔离出“这个易变对象”的变化，从而保持系统中“其他依赖该对象的对象”不随着需求改变而改变？

优缺点

- 优点：简单工厂模式主要用于隔离类对象的使用者和具体类型之间的耦合关系。面对一个经常变化的具体类型，紧耦合关系会导致软件的脆弱。通过使用工厂类,外界可以从直接创建具体产品对象的尴尬局面摆脱出来,仅仅需要负责“消费”对象就可以了。而不必管这些对象究竟如何创建及如何组织的．明确了各自的职责和权利，有利于整个软件体系结构的优化。

- 缺点：由于工厂类集中了所有实例的创建逻辑，违反了高内聚责任分配原则，将全部创建逻辑集中到了一个工厂类中；它所能创建的类只能是事先考虑到的，如果需要添加新的类，则就需要改变工厂类了。

3、抽象工厂模式 —

by: 缪丹权
柳敏乾
李青振

FACTORY—追MM少不了请吃饭了，麦当劳的鸡翅和肯德基的鸡翅都是MM爱吃的东西，虽然口味有所不同，但不管你带MM去麦当劳或肯德基，只管向服务员说"来四个鸡翅"就行了。麦当劳和肯德基就是生产鸡翅的Factory

工厂模式：客户类和工厂类分开。消费者任何时候需要某种产品，只需向工厂请求即可。消费者无须修改就可以接纳新产品。缺点是当产品修改时，工厂类也要做相应的修改。如：如何创建及如何向客户端提供。

抽象工厂模式定义

对象创建型模式，又称Kit模式。

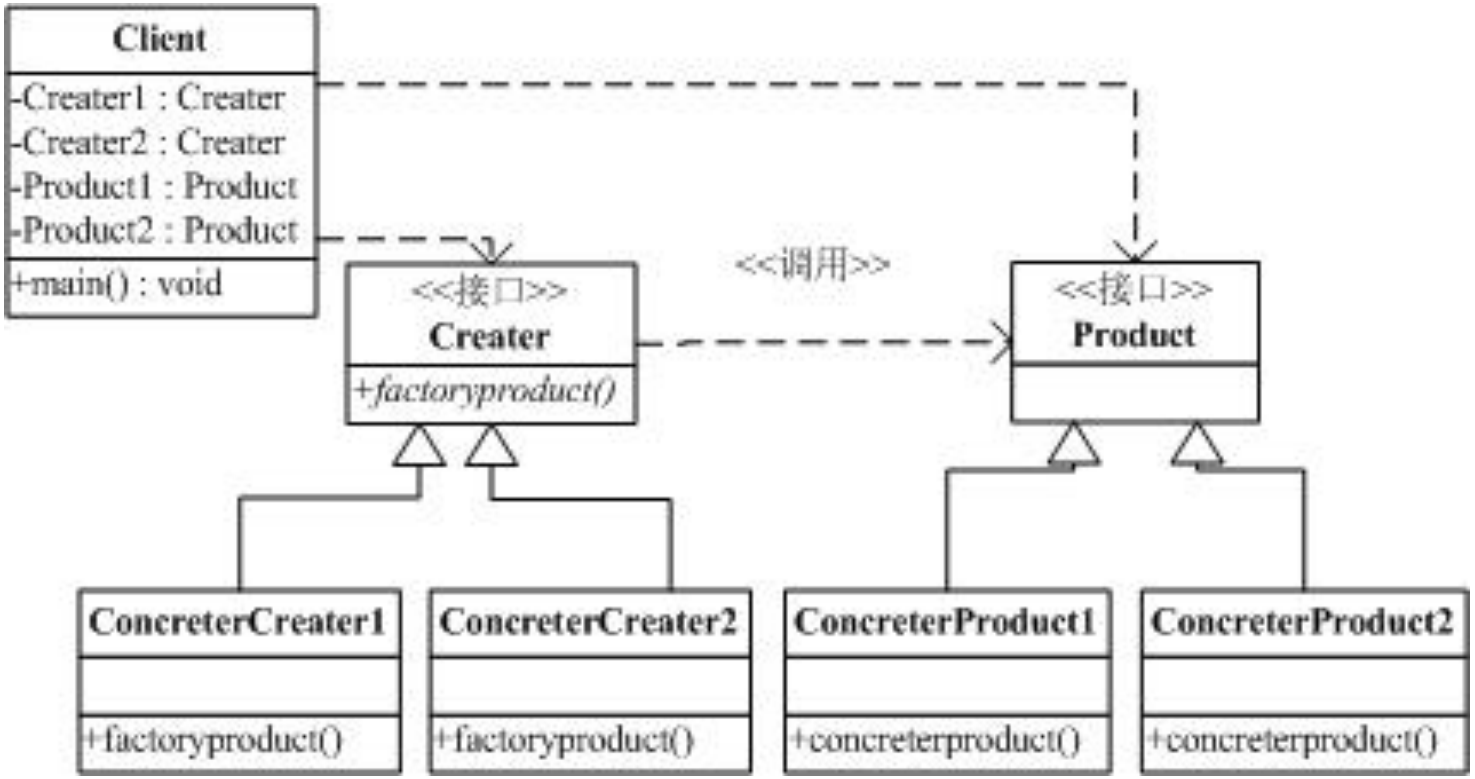
在<<设计模式>>中对Abstract Factory的意图是这样描述的：为了创建一组相关或相互依赖的对象提供一个接口，而且无需指定它们的具体类。

有时候，几个对象需要一种协调的方式实例化。例如，在处理用户界面时，系统可能在一个操作系统上用一组对象，在另一个系统上用另一组对象。Abstract Factory 确保系统根据情况获得正确的对象。

产品族

产品族，是指位于不同产品等级结构中，功能相关联的产品组成的家族。比如下图中，箭头所指就是三个功能相关联的产品。它们位于三个不同的等级结构中的相同位置上，组成了一个产品族。

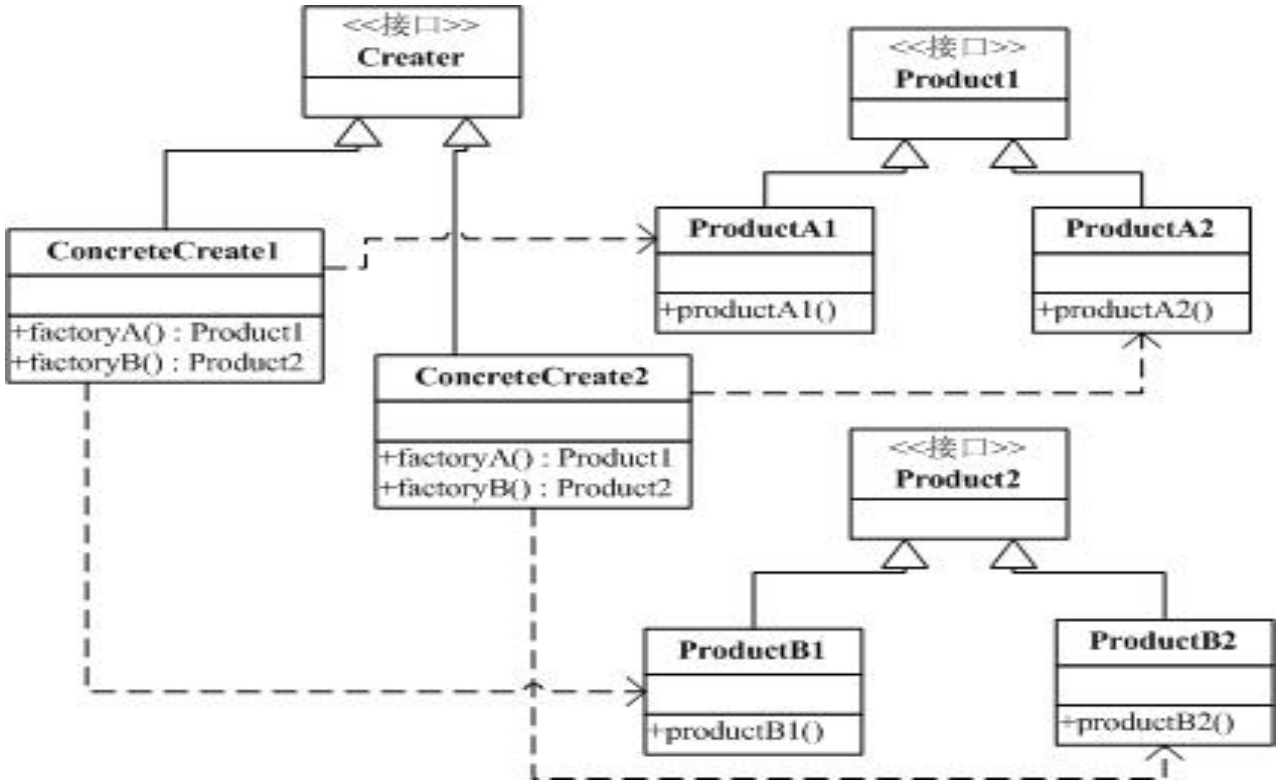
为什么需要AF


Client
-Creater1 : Creater
-Creater2 : Creater
-Product1 : Product
-Product2 : Product
+main() : void
<<接口>>
Creater
+factoryproduct()
<<调用>>
<<接口>>
Product
ConcreterCreater1
+factoryproduct()
ConcreterCreater2
+factoryproduct()
ConcreterProduct1
+concreterproduct()
ConcreterProduct2
+concreterproduct()

为什么需要AF

如果，现在有三个相似的工厂等级结构，那么采用工厂方法模式就势必要使用三个独立的工厂等级结构。由于三个等级结构相似性，会导致三个平行的等级结构。随着产品等级结构的数目增加，工厂方法模式所给出的工厂等级结构的数目也会增加。

抽象工厂模式结构图


<<接口>>
Creater
<<接口>>
Product1
ConcreteCreate1
+factoryA() : Product1
+factoryB() : Product2
ProductA1
+productA1()
ProductA2
+productA2()
ConcreteCreate2
+factoryA() : Product1
+factoryB() : Product2
<<接口>>
Product2
ProductB1
+productB1()
ProductB2
+productB2()

结构与角色

- 抽象工厂 (AbstractFactory)角色：
  担任这个角色的是工厂方法模式的核心，它是与应用系统的商业逻辑无关的。通常使用JAVA接口或者抽象JAVA类来实现，而所有的具体工厂类必须实现该接口或者继承抽象类。

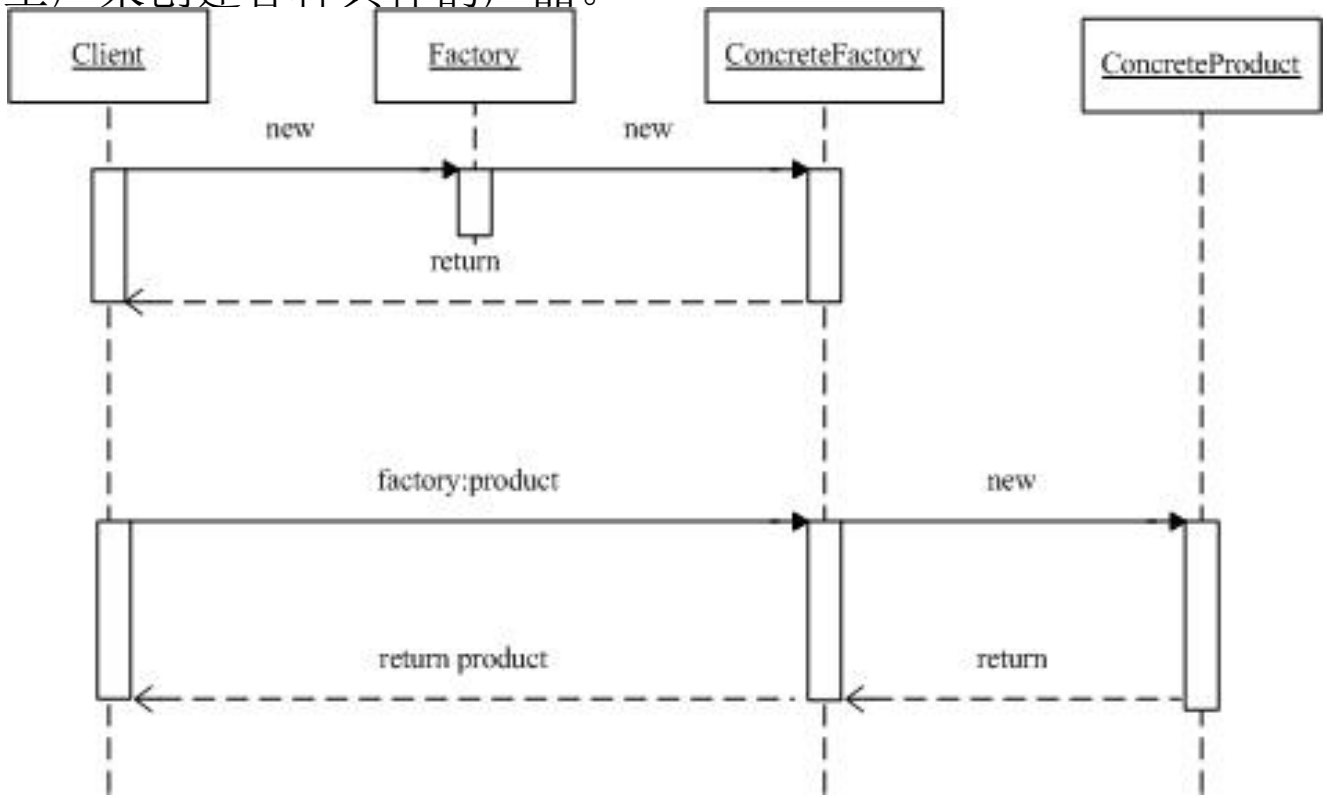
- 具体工厂类 (Conrete Factory)角色：
  这个角色直接在客户端的调用下创建产品的实例。这个角色含有选择合适的产品对象的逻辑。通常使用具体类来实现这个角色。

- 抽象产品(Abstract Product)角色：
  担任这个角色的类是工厂方法模式所创建的对象的父类，或它们共同拥有的接口。通常使用接口或者抽象类来实现这一角色。

- 具体产品(Concrete Product)角色:
  抽象工厂模式所创建的任何产品对象都是某一个具体产品类的实例。通常使用具体JAVA类来实现。

抽象工厂模式时序图

在客户新建一个抽象工厂的时候，其实是新建了一个具体的工厂，然后再通过该具体工厂来创建各种具体的产品。



抽象工厂模式优缺点

优点：

- 程序设计中有三种耦合：零耦合、抽象耦合、具体耦合。抽象工厂设计可以很好的把具体耦合转换到抽象耦合来减少耦合程度。
- 具体产品从客户代码中被分离出来。
- 容易改变产品的系列。
- 将一个系列的产品族统一到一起创建。

抽象工厂模式优缺点

缺点：

- 由于每个类的产生都要继承抽象类（或接口），并由工厂来创建，这样就增加了代码长度和工作量。
- 在产品族中扩展新的产品是很困难的，它需要修改抽象工厂的接口。
- 使软件结构更复杂。

与其他模式的区别和联系

与工厂模式的区别：

- 工厂方法模式是一种极端情况的抽象工厂模式，而抽象工厂模式可以看成是工厂方法模式的推广。

- 工厂方法模式用来创建一个产品的等级结构的，而抽象工厂模式是用来创建多个产品的等级结构的。

- 工厂方法模式只有一个抽象产品类，而抽象工厂模式有多个抽象产品类。

总结

- 工厂的实现通常使用Singleton模式。一个应用中一般每个系列产品只需要使用一个具体工厂的实例。

- 抽象工厂模式提供了一个创建一系列相关或相互依赖的对象的接口，关键点在于应对”多系列对象创建”的需求变化。

- 学会抽象工厂模式，可以更好地理解面向对象中的原则：面向接口编程，而不要面向实现编程。