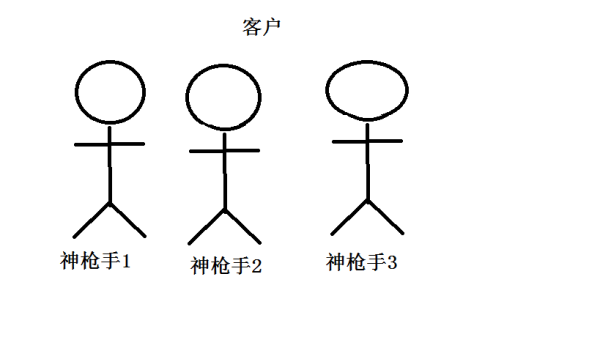
作者：caoglish  
链接：https://www.zhihu.com/question/20367734/answer/82361745  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

今天正好在学习抽象工厂和工厂方式，我就把我的理解写一写。  
  
工厂一般理解就是减少new创建对象的方式，用接口的方式来返回一个对象，而new创建的方式被封装了。然而，这个是初级认识，这不足以理解到工厂方式的真正目的。  
  
所以我要用我的方式帮助大家理解。  
  
**第一部分： 抽象工厂**  
一般来说，抽象工厂最简单形态也至少有4个元素：

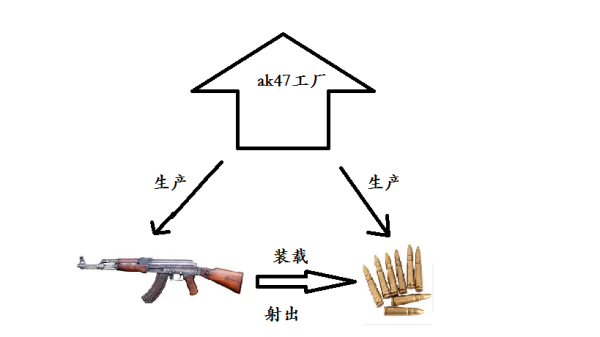
* 客户端（client）
* 工厂（factory）
* 产品A（product A）
* 产品B（product B）

我先用一个例子来实体说明抽象工厂是什么  
<img data-rawheight="460" data-rawwidth="819" src="https://pic4.zhimg.com/f01fefc49d87e52046b151ed2105bd9b\_b.png" class="origin\_image zh-lightbox-thumb" width="819" data-original="https://pic4.zhimg.com/f01fefc49d87e52046b151ed2105bd9b\_r.png">  
我现在有三个神枪手，他们聚在一起讨论他们玩枪的经历，

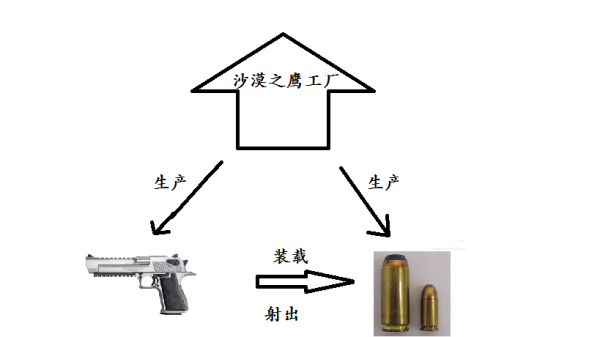
* 神枪手1：AK47打枪最好，杀伤力大。
* 神枪手2：沙漠之鹰最好，准。
* 神枪手3：连弩才好，上古兵器。

他们争论相持不下，所以决定比试一下，然后约定一个月后来鄙视。然后我们看到三个人分别去各自的工厂去购买武器。

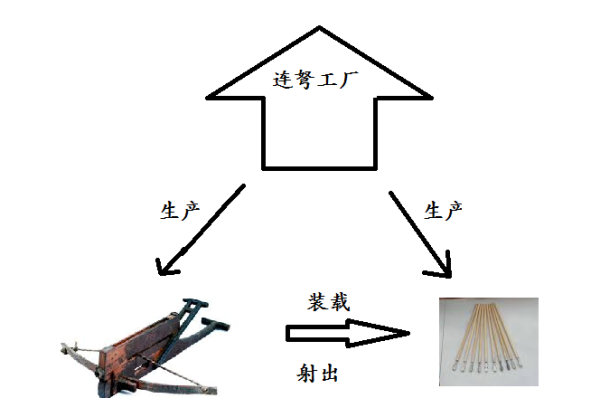
神射手1 去了AK47工厂,购买他们生产的武器和子弹

<img data-rawheight="460" data-rawwidth="819" src="https://pic1.zhimg.com/1e886da7830429f560618cc7827f5a3c\_b.png" class="origin\_image zh-lightbox-thumb" width="819" data-original="https://pic1.zhimg.com/1e886da7830429f560618cc7827f5a3c\_r.png">

神射手2 去了沙漠之鹰工厂，购买他们生产的武器和子弹

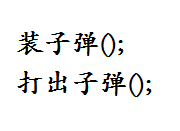
<img data-rawheight="460" data-rawwidth="819" src="https://pic1.zhimg.com/041908fde7964962890e6e50b6464abc\_b.png" class="origin\_image zh-lightbox-thumb" width="819" data-original="https://pic1.zhimg.com/041908fde7964962890e6e50b6464abc\_r.png">

神射手3，去了连弩工厂，购买他们生产的武器和“子弹”

<img data-rawheight="440" data-rawwidth="664" src="https://pic3.zhimg.com/0b3a2357020f0ab86278e5dd0854ad9a\_b.png" class="origin\_image zh-lightbox-thumb" width="664" data-original="https://pic3.zhimg.com/0b3a2357020f0ab86278e5dd0854ad9a\_r.png">

一个月后，我们看到了3个人分别拿出来以下的武器组合

<img data-rawheight="405" data-rawwidth="559" src="https://pic1.zhimg.com/9300f71eee33a18bea442b55faf6c6b0\_b.png" class="origin\_image zh-lightbox-thumb" width="559" data-original="https://pic1.zhimg.com/9300f71eee33a18bea442b55faf6c6b0\_r.png">

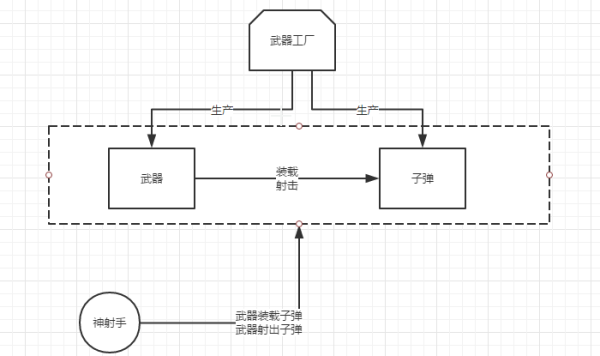
这三个人虽然用的武器不一样，但是每件武器都要做两件事<img data-rawheight="131" data-rawwidth="180" src="https://pic1.zhimg.com/0f95269350036ec0c54a341a9c32adf4\_b.png" class="content\_image" width="180">

最后，他们通过比赛谁，看哪种武器是最优秀的。

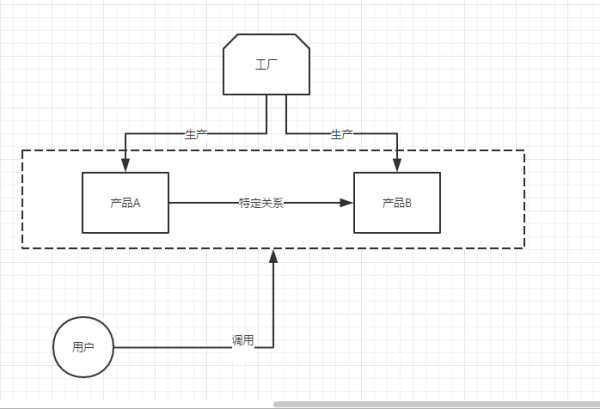
我们回头去看看，发现虽然三个人去了不同的工厂，用了不同的武器，用了不同的子弹，但是他们的相似之处太多，可以抽象出来。抽象出来的关系框架就是抽象工厂模式

* 武器工厂生产武器
* 武器工厂生产武器所用的子弹
* 武器可以装载子弹
* 武器可以射击子弹
* 神射手装载子弹
* 神射手射击子弹

那么把关系画出来就是：

<img data-rawheight="415" data-rawwidth="700" src="https://pic4.zhimg.com/467788de75805d3a715d2c2293dc4cbf\_b.png" class="origin\_image zh-lightbox-thumb" width="700" data-original="https://pic4.zhimg.com/467788de75805d3a715d2c2293dc4cbf\_r.png">

然后抽象一下成工厂模式就是：

<img data-rawheight="476" data-rawwidth="698" src="https://pic2.zhimg.com/5add7f54e4def71102eb70a602d7321d\_b.png" class="origin\_image zh-lightbox-thumb" width="698" data-original="https://pic2.zhimg.com/5add7f54e4def71102eb70a602d7321d\_r.png">

图画出来了，然后我们要如何理解抽象工厂模式. 过去人们都是从工厂开始解释，其实我个人觉得，倒过来讲反而更好理解。而理解抽象工厂模式的关键在于如何理解“**产品之间特定关系**”。

1. 用户要调用产品之间的这个特定关系
2. 这个特定关系只有产品A和产品B之间才有，所以我们需要产品A和产品B
3. 要获得产品A和产品B，我们要去生产这个产品A和产品B的那个工厂，叫工厂生产这个产品A和产品B

那么用例子来说，就是：

1. 神枪手喜欢装载子弹和射击的感觉（他们就是喜欢这个，只要有武器符合这个条件就行）
2. 那么他们就是需要武器（枪或者连弩）和子弹，这个武器是可以装载相应子弹和射击相应子弹的
3. 他们可以去特定工厂（ak47工厂或者连弩工厂）要求购买武器和相应子弹、

所以理顺抽象工厂的特点是什么?就是如下几个特点：

* 工厂是独立的（独立的类）
* 工厂是生产一整套有产品的（至少要生产两个产品)，这些产品**必须相互是有关系或有依赖的**
* 工厂是可以抽象的，工厂生产是可以抽象的
* 产品是可以抽象的，产品关系是可以抽象的
* 客户端是用来调用并理顺这些产品之间的关系（或指定工作流程）
* 不同工厂生产出的产品实例之间是不接触的，这个是靠客户端来封装实现的。

用一个例子来解释一下最后一条“不同工厂生产出的产品实例之间是不接触的，这个是考客户端来封装实现的”。

一个射击学员刚入门，听到射击老师说射击的几个要素: 武器，子弹，武器装载子弹，武器打出子弹。这个学员跃跃欲试，就跑到Ak47工厂买了枪，然后跑到沙漠之鹰工厂买了子弹，AK47装载沙鹰子弹，然后打出。学员卒。  
  
老师听说后，为了避免这个悲剧发生，承包了武器和子弹购买，要用AK47就必须在AK47工厂购买AK47和AK47子弹，保证了AK47加载沙漠之鹰子弹这样的悲剧发生了。

这就是客户端加载工厂实例后，保证只使用这个工厂的生产的产品和产品之间的关系，确保不和其他工厂的产品实例进行接触。

最终当我们调用客户端的行为时候，只要让客户端“加载”实例化的特定工厂，返回结果就是这个**“特定工厂**”所加工出来的“**特定产品**”的**“特定关系”**方法的结果了。

所以，当产品非常多的时候，产品之间关系又非常复杂，但却又可以进行抽象的时候，就是使用抽象工厂模式最好的时候了。

以上就是抽象工厂模式的个人理解

--------

**第二部分： 工厂方法**

工厂方法就两个元素：

* creator（创建者）
* product（产品）

而工厂方法就是一个创建者这个类的一个方法而已，这个方法就是用来封装产品的创建。

（留坑）

**第三部分： 抽象工厂 和工厂方法的不同点**

我从一下几个方面来理解抽象工厂和工厂方法不同点

* 抽象工程关键在于产品之间的抽象关系，所以至少要两个产品；工厂方法在于生成产品，不关注产品间的关系，所以可以只生成一个产品。
* 抽象工厂中客户端把产品的抽象关系理清楚，在最终使用的时候，一般使用客户端（和其接口），产品之间的关系是被封装固定的；而工厂方法是在最终使用的时候，使用产品本身（和其接口）。

抽象工厂更像一个复杂版本的策略模式，策略模式通过更换策略来改变处理方式或者结果；而抽象工厂的客户端，通过更改工厂还改变结果。所以在使用的时候，就使用客户端和更换工厂，而看不到产品本身。  
  
工厂方法目的是生产产品，所以能看到产品，而且还要使用产品。当然，如果产品在创建者内部使用，那么工厂方法就是为了完善创建者，从而可以使用创建者。另外创建者本身是不能更换所生产产品的。

* 抽象工厂的工厂是类；工厂方法的工厂是方法。

抽象工厂的工厂类就做一件事情生产产品。生产的产品给客户端使用，绝不给自己用。  
工厂方法生产产品，可以给系统用，可以给客户端用，也可以自己这个类使用。自己这个类除了这个工厂方法外，还能有其他功能性的方法  
  
  
其实仔细想想，这个两个模式是有交集的，在极端的情况下，这两个模式其实是一样的。所以可以这样理解

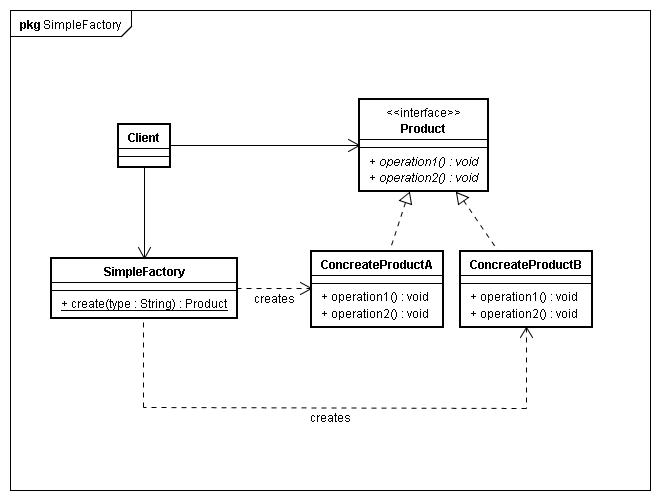
* 给工厂方法模式加一个客户端，除了客户端都不用这个创建者。这个时候创建者就是工厂类了。（单一产品的特定关系这个时候就是没有关系）
* 抽象工厂模式中，在客户端内部编程时候，就可以把工厂类当作创建者。

简单工厂，工厂方法，抽象工厂都属于设计模式中的创建型模式。其主要功能都是帮助我们把对象的实例化部分抽取了出来，优化了系统的架构，并且增强了系统的扩展性。

本文是本人对这三种模式学习后的一个小结以及对他们之间的区别的理解。

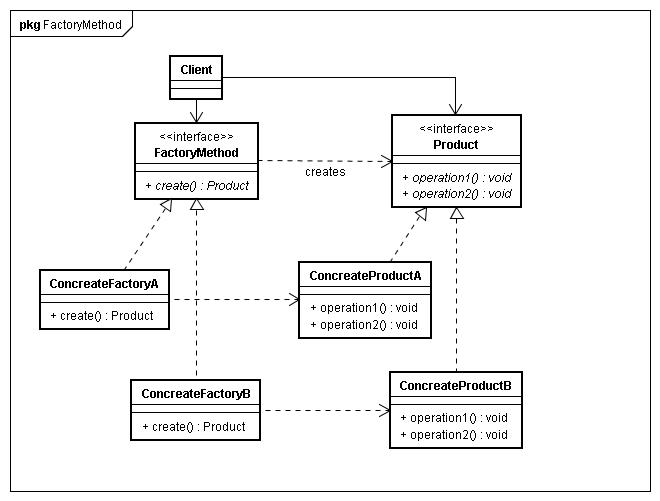
**简单工厂**

简单工厂模式的工厂类一般是使用静态方法，通过接收的参数的不同来返回不同的对象实例。

不修改代码的话，是无法扩展的。  


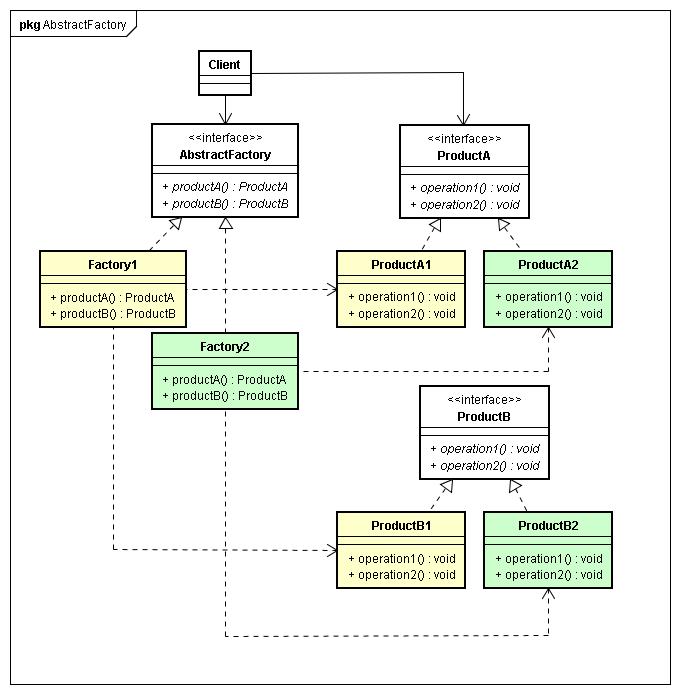
**工厂方法**

工厂方法是针对每一种产品提供一个工厂类。通过不同的工厂实例来创建不同的产品实例。

在同一等级结构中，支持增加任意产品。  


**抽象工厂**

抽象工厂是应对产品族概念的。比如说，每个汽车公司可能要同时生产轿车，货车，客车，那么每一个工厂都要有创建轿车，货车和客车的方法。

应对产品族概念而生，增加新的产品线很容易，但是无法增加新的产品。  


**小结**

★工厂模式中，重要的是工厂类，而不是产品类。产品类可以是多种形式，多层继承或者是单个类都是可以的。但要明确的，工厂模式的接口只会返回一种类型的实例，这是在设计产品类的时候需要注意的，最好是有父类或者共同实现的接口。

★使用工厂模式，返回的实例一定是工厂创建的，而不是从其他对象中获取的。

★工厂模式返回的实例可以不是新创建的，返回由工厂创建好的实例也是可以的。

**区别**

简单工厂 ： 用来生产同一等级结构中的任意产品。（对于增加新的产品，无能为力）

工厂方法 ：用来生产同一等级结构中的固定产品。（支持增加任意产品）     
抽象工厂 ：用来生产不同产品族的全部产品。（对于增加新的产品，无能为力；支持增加产品族）

以上三种工厂 方法在等级结构和产品族这两个方向上的支持程度不同。所以要根据情况考虑应该使用哪种方法。

**简单工厂优点：**客户端可以免除直接创建产品对象的责任，而仅仅是“消费”产品。简单工厂模式通过这种做法实现了对责任的分割。

**工厂方法有点：**允许系统在不修改具体工厂角色的情况下引进新产品。

**抽象工厂优点：**向客户端提供一个接口，使得客户端在不必指定产品具体类型的情况下，创建多个产品族中的产品对象

**第二种表述方式**

简单工厂、工厂方法和抽象工厂都是创建型的设计模式。

三者的**共同点**是：

1、都有两种作用不同的类：产品类和工厂类。其中，工厂类在自己的方法中实例化产品类（即使用new命令生成产品类的对象），并将生成的产品类的对象提供给外部使用。

2、创建的过程，都是类似的：工厂类得到一个标志（可以由程序输入，也可以读取配置文件中的内容），返回一个产品对象。

三者的**不同点**：

1、简单工厂

简单工厂方法中，包括一个“抽象产品类”（该类可以是接口Interface，也可以是实际的类Class），所有需要的产品类都是该“抽象产品类”的子类（如果是接口的话，那么就是说所有产品类都继承了该接口）。

简单工厂一般只包含一个具体的工厂类，由该工厂类生成所有的产品类的对象。生成产品类的方法，其内部一般是类似于switch的结构，根据输入的标志，选择创建不同类型的对象。由于不知道创建的对象到底是哪个类的，所以方法的返回值的类型是“抽象产品类”。

2、工厂方法

抽象工厂中，包括“抽象工厂类”和“抽象产品类”，同时包含不只一个工厂类。所有的工厂类都必须是“抽象工厂类”的子类，所有的产品都必须是“抽象产品类”的子类。

和简单工厂比起来，工厂方法一般是从抽象工厂开始的。一般都是在抽象工厂类中提供一个静态方法，由该方法根据输入的标志，生成不同的具体工厂类，然后由具体的产品类生成具体的产品。注意，一个具体工厂类只能生成一种具体的产品类的对象，不同的具体工厂生成不同的产品，而不是像简单工厂中那样，一个工厂类可以生成多种不同产品类的对象。可以这么理解，在选择不同的具体工厂类的时候，就选择了生成的产品，相对于简单工厂，相当于将选择产品的动作提前了。

因为不知道创建的具体工厂类到底是哪一个，所以生成具体工厂类的静态方法的返回值的类型是“抽象工厂类”。具体工厂类生成产品类的方法，返回值的类型也要求是“抽象产品类”（因为前端调用的时候，需要使用同样的代码来访问）。

3、抽象工厂

抽象工厂和工厂方法很类似，区别如下：

工厂方法模式：  
一个抽象产品类，可以派生出多个具体产品类。     
一个抽象工厂类，可以派生出多个具体工厂类。     
每个具体工厂类只能创建一个具体产品类的实例。

抽象工厂模式：  
多个抽象产品类，每个抽象产品类可以派生出多个具体产品类。     
一个抽象工厂类，可以派生出多个具体工厂类。     
每个具体工厂类可以创建多个具体产品类的实例。         
区别：  
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。     
工厂方法模式的具体工厂类只能创建一个具体产品类的实例，而抽象工厂模式可以创建多个