设计模式之工厂模式

作者：xuyan1095
原文链接：https://ld246.com/article/1484013776368
来源网站：链滴

引言

上一篇中，我介绍了单例模式的关键点，掌握了这些关键点，面试被问到单例肯定就没太大问题了，们先来回顾一下：

单例设计模式的关键点

一.私有构造函数

二.声明静态单例对象

三.构造单例对象之前要加锁（lock一个静态的object对象，某些语言可以声明同步执行，其实是一个的）

四.需要两次检测单例实例是否已经被构造，分别在锁之前和锁之后

好了，本文将向大家来讨论一下工厂模式，并且归纳工厂模式的关键点。

我在面试的时候，有时候会问到候选人有没有熟悉的设计模式，一般大部分候选人会选择说熟悉单例工厂（也有部分人会说熟悉观察者），如果我进一步问候选人是如何应用工厂模式的，10个候选人有1个会举连接Sqlserver,Oracle,Mysql等不同数据库时会用工厂模式产生不同的连接的例子。

OMG，我知道大家都是爱学习的同学，但是大家为什么不仔细想想，网上一搜一大堆的例子，你能到，别人难道看不到？我不知道大家能不能感受，作为一个面试官，问到一个问题的时候，所有候选都用同一个例子来回答你的感受。。。

如果我进一步问，工厂模式中，简单工厂，工厂和抽象工厂的区别，以及每种工厂的优劣势时，候选基本都会蒙圈。如果自己没有使用过，甚至使用过没有认真思考过，肯定是回答不上来的。好吧，下我就来跟大家讲讲工厂模式。

工厂模式理论我不长篇大论了。其核心功能是根据“需求”生产“产品”，还记得我上一篇说的，设模式的核心是解耦吗？工厂模式就是为了解耦“需求”和“产品”，但是别忘了，工厂模式工厂模式还有一个重要元素，就是“工厂”，所以工厂模式的核心思想，就是解耦“需求”“工厂”和“产品。

工厂模式，实际上也会根据业务情景不同会有不同的实现方式。一般分为3种。简单工厂，工厂和抽工厂。顾名思义，这三种从简单到抽象，名称越来越高大上，实现方式肯定是越来越复杂，所以，我可以得到第一个结论，三种工厂的实现是越来越复杂的。

简单工厂

先来看看简单工厂。废话不多，撸代码：

```
public class SimpleFactory
{
  int prodNo;

  public SimpleFactory(int prodNo) //构造工厂时告知工厂产品标识
  {
    this.prodNo = prodNo;
  }

  public IProduct GetProduct()
  {
```

```
    switch (prodNo) //根据产品标识生产产品
    {
      case 1:
        return new ProductA();
      case 2:
        return new ProductB();
      default:
        return new ProductA();
    }
  }
}

//产品A
class ProductA: IProduct
{
  //产品属性

  //......
}

//产品B
class ProductB : IProduct
{
  //产品属性

  //......
}

//产品接口
interface IProduct
{
  //产品方法

  //......
}
```

注意，这段例子代码当然还可以写的简单点，我完全可以在简单工厂中直接返回字符串而避免写产品和产品接口。但是即便是真实的业务场景是这样（真的只需要返回字符串或者数字什么的），大家还把产品类和工厂类分开，这样才是使用工厂模式的初衷，也就是实现解耦。

那么大家看看这段简单工厂的例子，如果我现在问，这个会有什么问题，该如何回答呢？提示一下，果说来了一个需求，增加一个产品C，该如何办？没错，简单工厂的问题就在于swich case（或者if el e)。每当新增一种产品时，你都需要去维护工厂中的判断语句，造成的后果就是可能这个工厂类会非非常长，各种判断全部挤在一起，给扩展和维护带来很多麻烦。说白了，你的产品和工厂还是没有完解耦，绑定在一起的。所以，我们得到了第二个结论：简单工厂通过构造时传入的标识来生产产品，同产品都在同一个工厂中生产，这种判断会随着产品的增加而增加，给扩展和维护带来麻烦。那么，何解决这个问题呢？

工厂模式

你猜对了，工厂模式可以解决这个问题，代码撸上：

```
interface IFactory //工厂接口
{
```

```
  IProduct GetProduct();
}

//A工厂类
public class FactoryA: IFactory
{
  IProduct productA;

  public FactoryA()
  {
    this.productA = new ProductA();
  }

  public IProduct GetProduct() //A工厂生产A产品
  {
    return this.productA;
  }
}

//B工厂类
public class FactoryB : IFactory
{
  IProduct productB;

  public FactoryB()
  {
    this.productB = new ProductB();
  }

  public IProduct GetProduct() //B工厂生产B产品
  {
    return this.productB;
  }
}

//产品接口
public interface IProduct
{
  //产品方法

  //.....
}

//产品A
public class ProductA : IProduct
{
  //产品属性

  //.....
}

//产品B
  public class ProductB : IProduct
  {
```

```
    //产品属性

    //.....
}
```

仔细观察这段代码，在工厂模式中，已经将工厂类分开，不再将所有产品在同一工厂中生产，这样就决了简单工厂中不停的switch case的问题。如果说来了一个C产品，那么我们只需要写一个C工厂和产品，在调用时用C工厂生产C产品即可，A和B工厂和产品完全不受影响。OK，优化说完了，但是还有问题。

问题在哪里呢？当业务需求是需要生产产品族的时候，工厂就不再适合了。首先我们搞清楚何谓产品和产品等级结构。举个例子来说，比如三星是一个品牌，三星生产洗衣机，电视，冰箱；格力也是一品牌，格力也生产洗衣机，电视，冰箱。那么，三星工厂和格力工厂生产的2个品牌的洗衣机，就在衣机这种产品的产品等级结构中（当然洗衣机产品等级结构中还有LG，海尔，三菱等等不同的品牌的厂的产品），所以，洗衣机就是一个产品等级。那么三星生产的三星洗衣机，三星电视机，三星冰箱是三星这个工厂的产品族。可能还会有西门子工厂产品族，格力工厂产品族，美的工厂产品族等等。

抽象工厂

好了，搞清楚了产品等级结构和产品族，我们得到第三个结论：工厂模式无法解决产品族和产品等级构的问题。再回过头来看抽象工厂模式。如果如上所述，业务场景是需要实现不同的产品族，并且实产品等级结构，就要用到抽象工厂模式了。还是来看代码：

```
//工厂接口，即抽象工厂
interface IFactory
{
  IFridge CreateFridge();
  IAirCondition CreateAirCondition();
}

//三星的工厂，生产三星的产品族
public class SamsungFactory : IFactory
{
  public IAirCondition CreateAirCondition()
  {
    return new SamsungAirCondition(); //三星的工厂生产三星的空调
  }

  public IFridge CreateFridge()
  {
    return new SamsungFridge(); //三星的工厂生产三星的冰箱
  }
}

//格力的工厂，生产格力的产品族
public class GreeFactry : IFactory
{
  public IAirCondition CreateAirCondition()
  {
    return new GreeAirCondition(); //格力的工厂生产格力的空调
  }

  public IFridge CreateFridge()
  {
```

```
    return new GreeFridge(); //格力的工厂生产格力的冰箱
  }
}

//冰箱产品接口
public interface IFridge
{
  //冰箱产品接口

  //冰箱的action
}

//空调接口
public interface IAirCondition
{
  //空调产品接口

  //空调的action
}

//三星的冰箱
public class SamsungFridge: IFridge
{
  //三星冰箱的action和property
}

//格力的冰箱
public class GreeFridge : IFridge
{
  //格力冰箱的action和property
}

//三星的空调
public class SamsungAirCondition : IAirCondition
{
  //三星空调的action和property
}

//格力的空调
public class GreeAirCondition : IAirCondition
{
  //格力空调的action和property
}
```

我们可以看到，在工厂模式中，一个工厂生产一个产品，所有的具体产品是由同一个抽象产品派生的不存在产品等级结构和产品族的概念；而在抽象工厂中，同一个等级的产品是派生于一个抽象产品（产品接口），一个抽象工厂派生不同的具体工厂，每个具体工厂生产自己的产品族（包含不同产品等）。所以我们得到第四个结论，工厂模式中，一个工厂生产一个产品，所有产品派生于同一个抽象产（或产品接口）；而抽象工厂模式中，一个工厂生产多个产品，它们是一个产品族，不同的产品族的品派生于不同的抽象产品（或产品接口）。

总结

一. 好了我们归纳一下，工厂模式实际上包含了3中设计模式，简单工厂，工厂和抽象工厂，关键点如

：

1. 三种工厂的实现是越来越复杂的

2. 简单工厂通过构造时传入的标识来生产产品，不同产品都在同一个工厂中生产，这种判断会随着产的增加而增加，给扩展和维护带来麻烦

3. 工厂模式无法解决产品族和产品等级结构的问题

4. 抽象工厂模式中，一个工厂生产多个产品，它们是一个产品族，不同的产品族的产品派生于不同的象产品（或产品接口）。

好了，如果你能理解上面的关键点，说明你对工厂模式已经理解的很好了，基本上面试官问你工厂模，你可以昂头挺胸的说一番。但是，面试官怎么可能会放过每一次虐人的机会？你仍然可能面临下面问题：

在上面的代码中，都使用了接口来表达抽象工厂或者抽象产品，那么可以用抽象类吗？有何区别？

从功能上说，完全可以，甚至可以用接口来定义行为，用抽象类来抽象属性。抽象类更加偏向于属性抽象，而用接口更加偏向行为的规范与统一。使用接口有更好的可扩展性和可维护性，更加灵活实现散耦合，所以编程原则中有一条是针对接口编程而不是针对类编程。

二. 到底何时应该用工厂模式

根据具体业务需求。不要认为简单工厂是用switch case就觉得一无是处，也不要觉得抽象工厂比较大上就到处套。我们使用设计模式是为了解决问题而不是炫技，所以根据三种工厂模式的特质，以及未来扩展的预期，来确定使用哪种工厂模式。

三. 说说你在项目中工厂模式的应用

如果你看了这篇文章，被问到这个问题时，还傻乎乎的去举数据库连接的例子，是要被打板子的。。

所以在面试中，把工厂模式和自己做的东西联系起来，如何建立工厂，如何生产不同的产品，如何扩，如何维护等等。我想，把理论应用到实际，而且是真实业务逻辑中，给面试官的印象无论如何不会差，甚至会对你刮目相看。

当然，即便是你没有真的使用过，如果面试官问道了工厂模式，你仍然可以把你以往的经验和设计模联系起来回（hu）答（you）面试官，只要你理解了，把来龙去脉说清楚，并且可以回答问题，我想该是可以令面试官满意的。