

工厂方法模式 (Factory Method) - 最易懂的设计模式解析



Carson_Ho (/u/383970bef0a0)   已关注

2016.08.16 19:35* 字数 2291 阅读 5595 评论 0 喜欢 37

(/u/383970bef0a0)



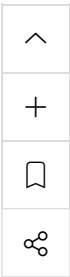
前言

在上文提到的最易懂的设计模式系列解析：简单工厂模式
(<https://www.jianshu.com/p/e55fbddc071c>)，发现简单工厂模式存在一系列问题：

- 工厂类集中了所有实例（产品）的创建逻辑，一旦这个工厂不能正常工作，整个系统都会受到影响；
- 违背“开放 - 关闭原则”，一旦添加新产品就不得不修改工厂类的逻辑，这样就会造成工厂逻辑过于复杂。
- 简单工厂模式由于使用了静态工厂方法，静态方法不能被继承和重写，会造成工厂角色无法形成基于继承的等级结构。

为了解决上述的问题，我们又使用了一种新的设计模式：工厂方法模式。

在阅读本文前建议先阅读：



- 1. 1分钟全面了解“设计模式” (<https://www.jianshu.com/p/6e5eda3a51af>)
- 2. 最易懂的设计模式系列解析：简单工厂模式 (<https://www.jianshu.com/p/e55fbddc071c>)
- 3. 其他设计模式介绍
 - 1分钟全面了解“设计模式” (<https://www.jianshu.com/p/6e5eda3a51af>)
 - 单例模式（Singleton） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/b8c578b07fbc>)
 - 简单工厂模式（SimpleFactoryPattern） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/e55fbddc071c>)
 - 工厂方法模式（Factory Method） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/d0c444275827>)
 - 抽象工厂模式（Abstract Factory） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/7deb64f902db>)
 - 策略模式（Strategy Pattern） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/0c62bf587b9c>)
 - 适配器模式（Adapter Pattern） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/9d0575311214>)
 - 代理模式（Proxy Pattern） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/a8aa6851e09e>)
 - 模板方法模式（Template Method） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/a3474f4fee57>)
 - 建造者模式（Builder Pattern） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/be290ccea05a>)
 - 外观模式（Facade Pattern） - 最易懂的设计模式解析 (<https://www.jianshu.com/p/1b027d9fc005>)

目录



工厂方法模式.jpg

1. 介绍

1.1 定义

工厂方法模式，又称工厂模式、多态工厂模式和虚拟构造器模式，通过定义工厂父类负责定义创建对象的公共接口，而子类则负责生成具体的对象。

^

+

□

⌘

1.2 主要作用

将类的实例化（具体产品的创建）延迟到工厂类的子类（具体工厂）中完成，即由子类来决定应该实例化（创建）哪一个类。

1.3 解决的问题

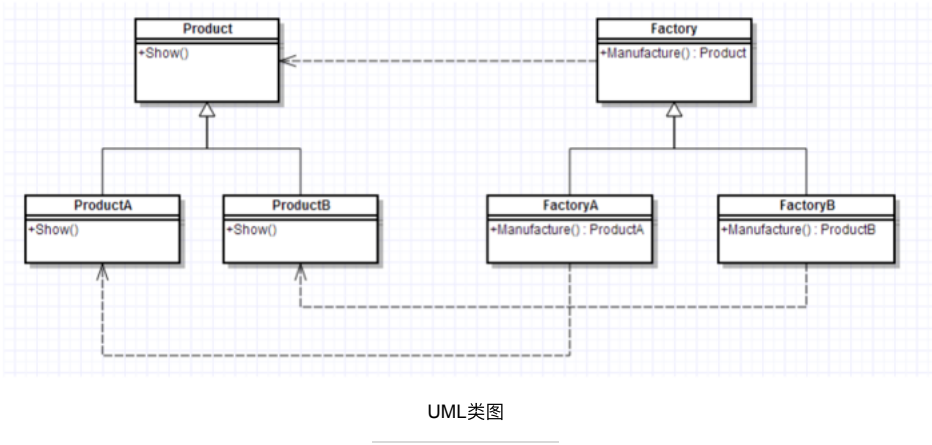
工厂一旦需要生产新产品就需要修改工厂类的方法逻辑，违背了“开放 - 关闭原则”

1. 即简单工厂模式的缺点

2. 之所以可以解决简单工厂的问题，是因为工厂方法模式把具体产品的创建推迟到工厂类的子类（具体工厂）中，此时工厂类不再负责所有产品的创建，而只是给出具体工厂必须实现的接口，这样工厂方法模式在添加新产品的时候就不修改工厂类逻辑而是添加新的工厂子类，符合开放封闭原则，克服了简单工厂模式中缺点

2. 模式原理

2.1 UML类图



UML类图

2.2 模式组成

组成（角色）	关系	作用
抽象产品 (Product)	具体产品的父类	描述具体产品的公共接口
具体产品 (Concrete Product)	抽象产品的子类；工厂类创建的目标类	描述生产的具体产品
抽象工厂 (Creator)	具体工厂的父类	描述具体工厂的公共接口
具体工厂 (Concrete Creator)	抽象工厂的子类；被外界调用	描述具体工厂；实现FactoryMethod工厂方法创建产品的实例

2.3 使用步骤

- 步骤1：创建抽象工厂类，定义具体工厂的公共接口；
- 步骤2：创建抽象产品类，定义具体产品的公共接口；
- 步骤3：创建具体产品类（继承抽象产品类） & 定义生产的具体产品；
- 步骤4：创建具体工厂类（继承抽象工厂类），定义创建对应具体产品实例的方法；
- 步骤5：外界通过调用具体工厂类的方法，从而创建不同具体产品类的实例

^

+

□

⌘

3. 实例讲解

接下来我用一个实例来对工厂方法模式进行更深一步的介绍。

3.1 实例概况

- 背景：小成有一间塑料加工厂（仅生产A类产品）；随着客户需求的变化，客户需要生产B类产品；
- 冲突：改变原有塑料加工厂的配置和变化非常困难，假设下一次客户需要再发生变化，再次改变将增大非常大的成本；
- 解决方案：小成决定置办**塑料分厂B**来生产B类产品；

即工厂方法模式

3.2 使用步骤

步骤1： 创建**抽象工厂类**，定义具体工厂的公共接口

```
abstract class Factory{
    public abstract Product Manufacture();
}
```

步骤2： 创建**抽象产品类**，定义具体产品的公共接口；

```
abstract class Product{
    public abstract void Show();
}
```

步骤3： 创建**具体产品类**（继承抽象产品类），定义生产的具体产品；

```
//具体产品A类
class ProductA extends Product{
    @Override
    public void Show() {
        System.out.println("生产出了产品A");
    }
}

//具体产品B类
class ProductB extends Product{
    @Override
    public void Show() {
        System.out.println("生产出了产品B");
    }
}
```

步骤4： 创建**具体工厂类**（继承抽象工厂类），定义创建对应具体产品实例的方法；

```
//工厂A类 - 生产A类产品
class FactoryA extends Factory{
    @Override
    public Product Manufacture() {
        return new ProductA();
    }
}

//工厂B类 - 生产B类产品
class FactoryB extends Factory{
    @Override
    public Product Manufacture() {
        return new ProductB();
    }
}
```



步骤5：外界通过调用具体工厂类的方法，从而创建不同具体产品类的实例

```
//生产工作流程
public class FactoryPattern {
    public static void main(String[] args){
        //客户要产品A
        FactoryA mFactoryA = new FactoryA();
        mFactoryA.Manufacture().Show();

        //客户要产品B
        FactoryB mFactoryB = new FactoryB();
        mFactoryB.Manufacture().Show();
    }
}
```

结果：

生产出了产品A
生产出了产品C

4. 优点

- 更符合开-闭原则
新增一种产品时，只需要增加相应的具体产品类和相应的工厂子类即可

简单工厂模式需要修改工厂类的判断逻辑

- 符合单一职责原则
每个具体工厂类只负责创建对应的产品

简单工厂中的工厂类存在复杂的switch逻辑判断

- 不使用静态工厂方法，可以形成基于继承的等级结构。

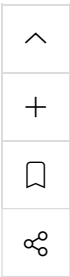
简单工厂模式的工厂类使用静态工厂方法

总结：工厂模式可以说是简单工厂模式的进一步抽象和拓展，在保留了简单工厂的封装优点的同时，让扩展变得简单，让继承变得可行，增加了多态性的体现。

5. 缺点

- 添加新产品时，除了增加新产品类外，还要提供与之对应的具体工厂类，系统类的个数将成对增加，在一定程度上增加了系统的复杂度；同时，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。
- 虽然保证了工厂方法内的对修改关闭，但对于使用工厂方法的类，如果要更换另外一种产品，仍然需要修改实例化的具体工厂类；
- 一个具体工厂只能创建一种具体产品

6. 应用场景



在了解了优缺点后，我总结了工厂方法模式的应用场景：

- 当一个类不知道它所需要的对象的类时
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；
- 当一个类希望通过其子类来指定创建对象时
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。

7. 总结

本文主要对工厂方法模式进行了全面介绍，接下来将介绍其他设计模式，有兴趣可以继续关注Carson_Ho的安卓开发笔记

(https://www.jianshu.com/users/383970bef0a0/latest_articles)!!!

请点赞！因为你的鼓励是我写作的最大动力！

相关文章阅读

单例模式 (Singleton) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/b8c578b07fbc>)

简单工厂模式 (SimpleFactoryPattern) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/e55fbddc071c>)

工厂方法模式 (Factory Method) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/d0c444275827>)

抽象工厂模式 (Abstract Factory) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/7deb64f902db>)

策略模式 (Strategy Pattern) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/0c62bf587b9c>)

适配器模式 (Adapter Pattern) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/9d0575311214>)

代理模式 (Proxy Pattern) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/a8aa6851e09e>)

模板方法模式 (Template Method) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/a3474f4fee57>)

建造者模式 (Builder Pattern) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/be290ccea05a>)

外观模式 (Facade Pattern) - 最易懂的设计模式解析

(<https://www.jianshu.com/p/1b027d9fc005>)

欢迎关注Carson_Ho

(https://www.jianshu.com/users/383970bef0a0/latest_articles)的简书！

不定期分享关于安卓开发的干货，追求短、平、快，但却不缺深度。




小礼物走一走，来简书关注我

赞赏支持

设计模式 (/nb/5752111)

举报文章 © 著作权归作者所有





Carson_Ho (/u/383970bef0a0) 

写了 285959 字，被 24487 人关注，获得了 17159 个喜欢
(/u/383970bef0a0)

✓ 已关注

简书认证作者、CSDN签约作者、稀土掘金专栏作者 定位：分享 Android开发 干货 Github：https://github.c...

喜欢 | 37





更多分享

(http://cwb.assets.jianshu.io/notes/images/5300129

被以下专题收入，发现更多相似内容

+ 收入我的专题

 Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)





 互联网科技 (/c/93d58e9169cb?utm_source=desktop&utm_medium=notes-included-collection)

^

+

🔖

🔗

-  Android开发 (/c/0dc880a2c73c?utm_source=desktop&utm_medium=notes-included-collection)
-  Android... (/c/58b4c20abf2f?utm_source=desktop&utm_medium=notes-included-collection)
-  Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)
-  架构算法设计模... (/c/c568ddab391a?utm_source=desktop&utm_medium=notes-included-collection)
-  知识 | 解析 (/c/c820977479ef?utm_source=desktop&utm_medium=notes-included-collection)


展开更多

(/p/7deb64f902db?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
抽象工厂模式（Abstract Factory） - 最易懂的设计模式解析 (/p/7deb64f902db?)

前言 在上文提到的最易懂的设计模式系列解析：工厂方法模式，发现工厂方法模式存在一个严重的问题：一个具体工厂只能创建一类产品 而在实际过程中，一个工厂往往需要生产多类产品。为了解决上述的问题，...


 Carson_Ho (/u/383970bef0a0?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/e55fbddc071c?)

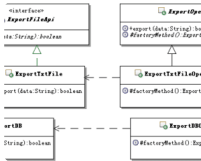


utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
简单工厂模式（SimpleFactoryPattern） - 最易懂的设计模式解析 (/p/e55fbddc071c?)

前言 今天我来全面总结一下Android开发中最常用的设计模式 - 简单工厂模式。其他设计模式介绍1分钟全面了解“设计模式”单例模式（Singleton） - 最易懂的设计模式解析简单工厂模式（SimpleFactoryPattern） - ...


 Carson_Ho (/u/383970bef0a0?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/f1960652b64b?)




utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
【创建型模式二】工厂方法(Factory Method) (/p/f1960652b64b?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

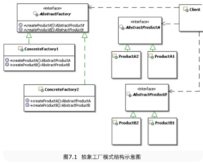
1 场景问题# 1.1 导出数据的应用框架## 考虑这样一个实际应用：实现一个导出数据的应用框架，来让客户选择数据的导出方式，并真正执行数据导出。 在一些实际的企业应用中，一个公司的系统往往分散在很多...

 猿码道 (/u/657c611b2e07?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

设计模式汇总 (/p/f7f9553ba2ba?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)


设计模式汇总 一、基础知识 1. 设计模式概述 定义：设计模式（Design Pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结，使用设计模式是为了可重用代码、让代码更容易被他人...

 MinoyJet (/u/9c0529da1861?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
【创建型模式三】抽象工厂 (Abstract Factory) (/p/e873855e88a0?utm_ca...


1 场景问题# 1.1 选择组装电脑的配件## 举个生活中常见的例子——组装电脑，我们在组装电脑的时候，通常需要选择一系列的配件，比如：CPU、硬盘、内存、主板、电源、机箱等等。为了使讨论简单点，只考...

 猿码道 (/u/657c611b2e07?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

傻笑练习19 (/p/249a8260a7bf?utm_campaign=maleskine&utm_content...


我们没有加班工资，但是中秋节前一天老板给我们发了300的过节费，让我们自己买点东西，哈哈，老板真好。前几天下楼买饭的时候正好遇到大雨，就问保安大叔借伞，大叔借了我一把伞 老板隔三差五就请我们...

 团丫团丫团 (/u/f928ce6035d7?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

写材料的痛，有谁懂？！ (/p/23ec4d0ffad7?utm_campaign=maleskine&...

不知从何起，加班就一直伴随着我，不，是我们这个群体。白加黑，5加2，换句话说就是每天都是上班，加班也就无从谈起了。那总要给自己一个加班的理由吧。理由就是领导交办的各项任务，在正常的时间内是...

 爱茉莉 (/u/69ee178b0eb2?)


utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/d77e58833ab2?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
可能我们都不真的懂“白” (/p/d77e58833ab2?utm_campaign=maleskine&...

这个双休读了一本好书，是MUJI艺术总监原研哉先生撰写的《白》，这本书原研哉并不仅仅企图讲述一个叫白的颜色，而是借由白找到人们自身文化设定的那些感觉之源。试图找到那个通过“白”的概念营造的简洁和...

 右小新 (/u/4e4206c5ba86?)


utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

(/p/d883b39e0b98?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)
小清新系列（五）——抱猫取暖 (/p/d883b39e0b98?utm_campaign=male...

迎风，没有肩膀，我还有我的猫咪..... 其实我已经很困了，但是，涂不完，我睡不着.....

 夏末夏墨 (/u/b3260b73cb20?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

^

+

🔖

🔗