1. JAVA 设计模式

开闭原则：对扩展开放对修改关闭。理解：开发新需求使用其扩展功能，而不是修改其源代码完成。

1. 里氏替换原则: 子类可以扩展父类功能，但不能修改父类原有功能。 如果修改其父类方法可能造成复用性差，运行多态比较频繁时程序抱错几率较大。

修正方法为：取消继承关系，重新设计

1. 依赖倒置原则 高层模块不应依赖底层模块，两者均应该抽象，抽象不应该依赖细节，细节应该依赖抽象，即：面向接口编程，而不是面向对象编程。

是实现开闭原则途径之一,他降低了客户与实现中间的耦合。

作用：

1. 降低系统件的耦合性
2. 提升系统的运行稳定性
3. 减少并发引起的风险
4. 提升系统的可读性和可维护性

实现方法

1. 每个类提供接口或抽象类，或者二者具备
2. 变量声明尽量是接口或抽象类
3. 任何类不应该从具体类派生
4. 使用里氏替换原则
5. 依赖倒置原则
6. 单一职责 一个类应该有且仅有一个引起他变化的原因，否则类应该被拆分

优点：

1. 降低类的复杂度，一个类只负责一项职责，逻辑肯定要比负责多职责简单的多。
2. 提数类的可读性
3. 提升系统的可运维性。
4. 变更引起的风险降低
5. 接口隔离 要为个各类建立他们需要的接口，而不是试图建立一个庞大接口供所有类调用。单一职责原则和接口隔离职责都是为了高内聚、低耦合、体现了封装的思想。

单一职责原则注重约束类，针对程序的实现和细节，接口隔离约束接口，主要针对抽象和整体框架设计。

实现方法

1. 接口尽量小，但要有限度，一个接口只服务一个子模块或业务逻辑
2. 为依赖接口实现者定制服务，只需要提供接口需要的方法，屏蔽不需要的方法。
3. 了解环境，按照需求标准定制接口
4. 提高内聚，减少对外交互
5. 迪米特法则 两个实体类无需直接通信，那么就不应当发生直接调用，可以通过第三方转发该调用。

优点:降低类之间的耦合度，提高模块的独立性，提高了类的可复用性和系统的扩展性。

缺点：使用迪米特法则会产生大量的中间类，从而增加系统的复杂性，使模块通信效率降低。

案例:经纪人与明星

1. JAVA 设计模式

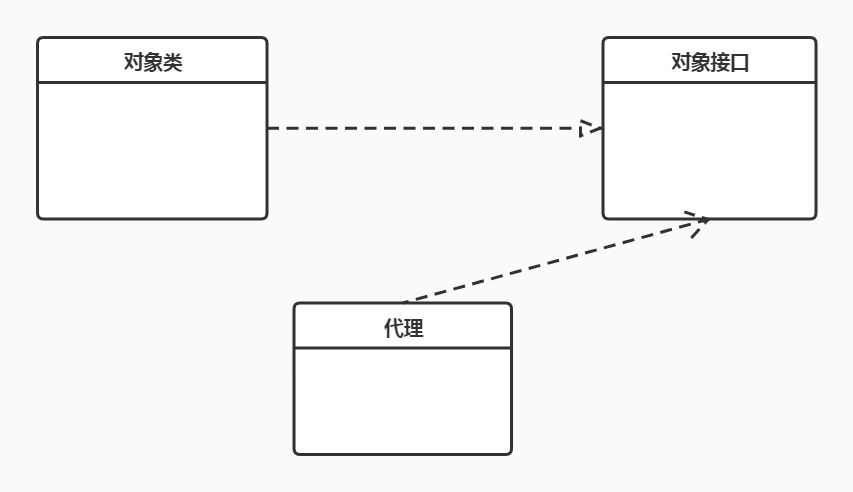
创建型模式：关注怎样创建对象，将对象的创建于使用分离。包含以下模式，抽象工厂、工厂模式、原型模式、单例模式、建造者模式

1. 代理模式 通过代理对象访问被代理对象的方法

角色

1. 被代理者
2. 被代理者接口
3. 代理

类图

****

1. 静态代理 代理与被代理实现相同接口或继承父类
2. JDK实现动态代理，机制为被代理对象实现InvocationHandler接口
3. 利用CGlib操作字节码在运行时给被代理对象生成子类，让其产生继承关系便可以调用被代理类方法，不支持final 方法
4. 单例模式
5. 懒汉模式 类加载时没有生成单例，但是在调用初始化方法后会生成。

缺点：在多线程下需要同步，消耗资源，影响性能。

1. 饿汉模式 在类加载时候创建单例
2. 原形模式 用一个已经创建的对象通过复制原对象来创建新的对象，java中Cloneable进行浅拷贝

深拷贝与浅拷贝区别

深拷贝会拷贝所有对象的属性，并拷贝属性指向的内存。速度慢，花销大。浅拷贝复制对象的引用，拷贝的对象和原对象之间共享。

类似指针与内存关系，深拷贝拷贝对账内存到新对象，而浅拷贝拷贝对象的指针引用。

角色

1. 抽象原型类：原型对象必须实现的接口
2. 具体原型类：实现抽象原型类clone方法，他是可复制的
3. 访问类：具体原型类的clone中方法创建新对象。
4. 工厂模式 定义一个创建产品对象的工厂接口，将产品对象的实际创建工作推迟到具体子类中。创建于使用分离。

优点

1） 用户只需要知道工厂名称就可以得到产品，无需知道产品的具体创建过程

2） 在系统中增加新产品时只需要添加产品和对应的具体工程类，无需对原工厂修改，满足开闭原则

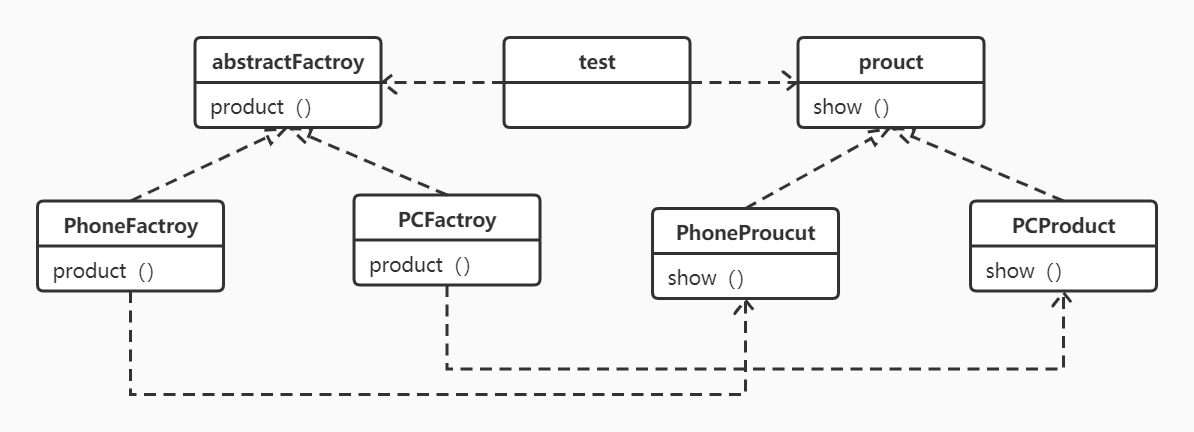
缺点

1. 每增加一个产品就要增加一个对应的工厂类，增加了系统复杂度

角色

1. 抽象工厂 提供创建产品的接口
2. 具体工厂 实现抽象工厂，完成创建产品的具体实现
3. 抽象产品 定义了产品的规范
4. 具体产品 实现了抽象产品所定义的接口

UML



1. 建造者模式 将一个复杂对象的构建和表示分离，使同样的构建过程可以创建不同的表示。

优点:

1. 各个具体的构建者相互独立，有利于系统的扩展
2. 客户端不必知道产品内部组成细节，便于控制细节。

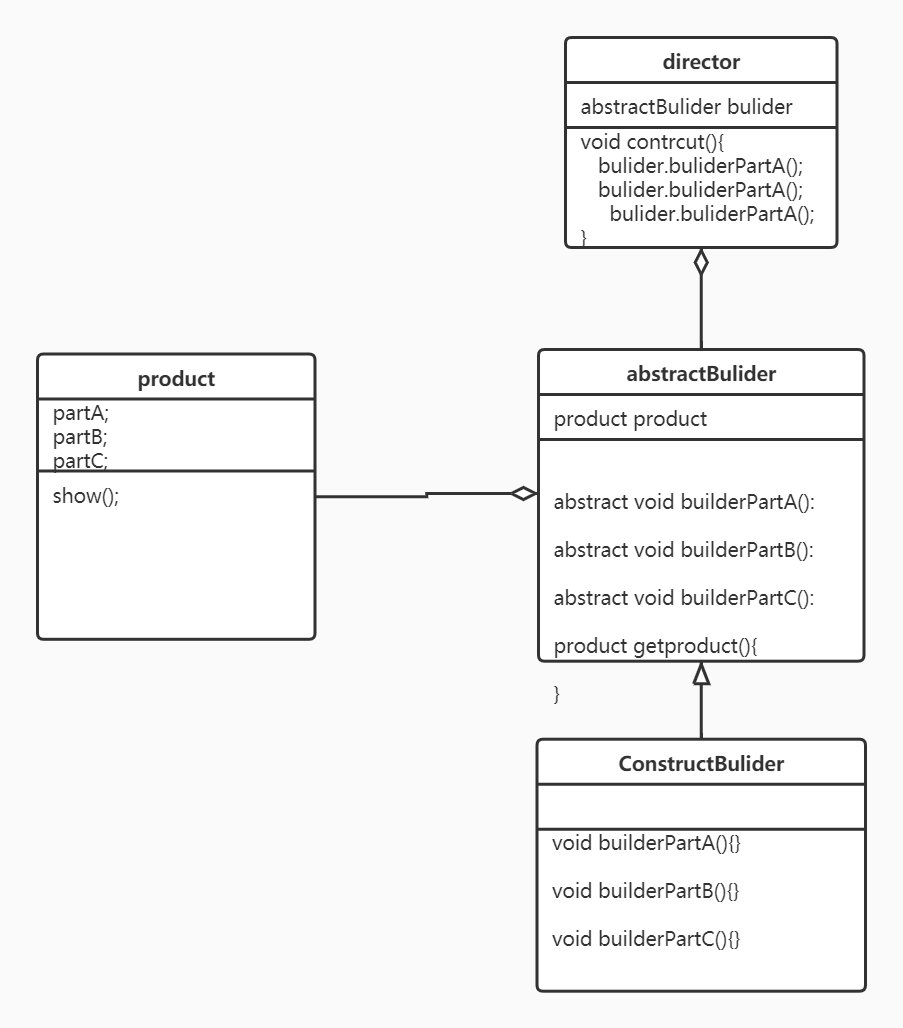
缺点

1. 产品的组成部分必须相同，限制了使用范围
2. 产品内部变化、会增加很多构建者

角色

1. 产品角色
2. 抽象建造者
3. 具体建造者
4. 指挥者

类图



结构型模式 描述将类获对象按某种布局组合更大的结构，类结构模式采取集成方式来组织类和接口，皱着采用聚合或组合来组合对象

代理模式，桥接模式，适配器模式，外观模式，组合模式，装饰模式，享元模元（代桥外适组装享）

1. 代理模式 不能直接访问对象，而是通过对象的代理来完成对对象的访问。

优点：

1. 代理模式在客户端与目标对之间起到一个中介和保护的作用
2. 代理对象可以扩展目标对象的功能
3. 代理模式能将客户端与目标对象分离，在一定程度降低了系统的耦合度

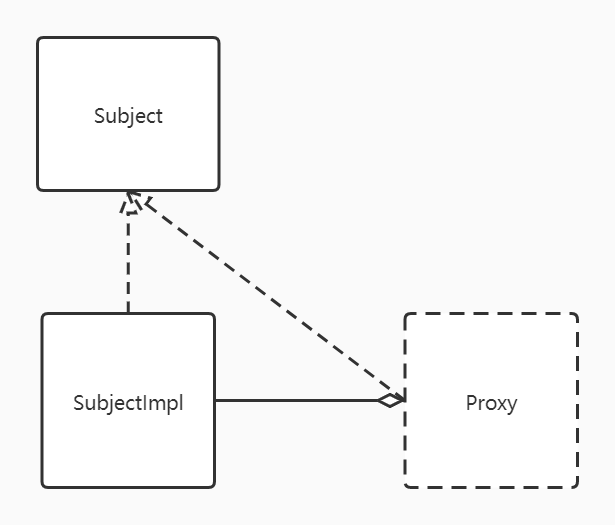
缺点

1. 增加了代理对象会造成请求速度变慢
2. 增加了系统的复杂度

角色

1. 抽象主题 接口或抽象类申明真是的主题和代理对的实现
2. 真是主题 实现了抽象主题中的具体业务
3. 代理类 提供了与真实主题相同的接口

类图



1. 适配器模式 讲一个接口转换成另外一个客户希望的接口，是的原本不能在一起不兼容的类一起工作

优点

1. 客户端通过适配器可以透明的调用目标接口
2. 复用了现有的类，不需要修改原有的代码
3. 将目标和适配类解耦，解决了目标类和适配接口不一致的问题

缺点：对类适配器来说更换适配器实现过程较为复杂

角色

1. 目标接口 业务所期待的接口
2. 适配者 被访问和适配的现存组件接口
3. 适配器 转换器，通过继承或引用适配的对象，把适配接口转换为目标接口

类图

