Java设计模式

1. 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。
2. 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借口，而不是试图建立一个庞大借口供所有类调用。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。

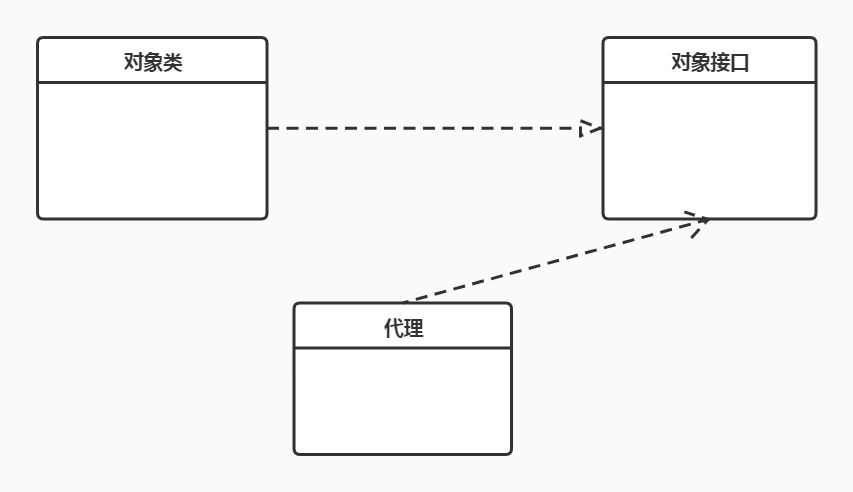
实现方法

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. 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

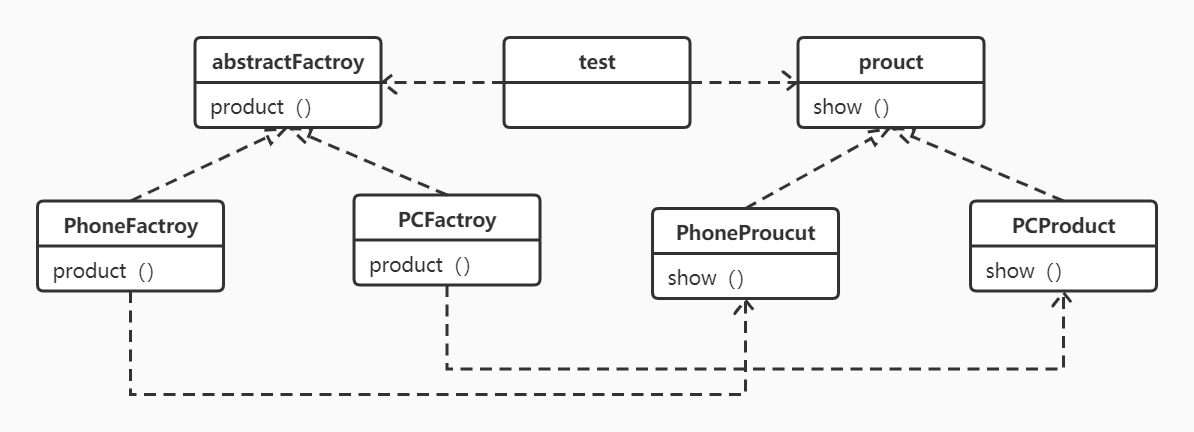
缺点

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. 构造者模式