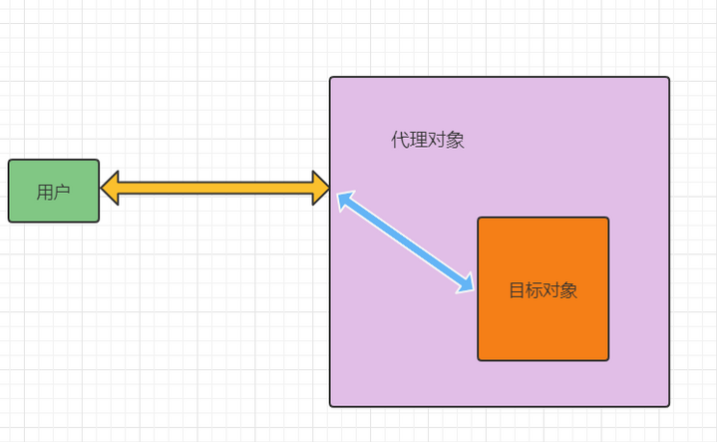
# 基本概念

代理(Proxy)模式是一种**结构型**设计模式，提供了对目标对象另外的访问方式；即通过代理对象访问目标对象。



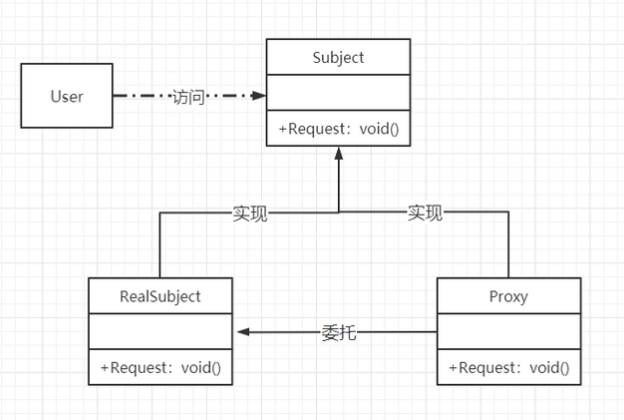
这样做的好处是：可以在目标对象实现的基础上，增强额外的功能操作，即扩展目标对象的功能。这里使用到编程中的一个思想：不要随意去修改别人已经写好的代码或者方法，如果需要修改，可以通过代理的方式来扩展该方法。

代理模式大致有三种角色：

1、Real Subject：真实类，也就是被代理类、委托类。用来真正完成业务服务功能。

2、Proxy：代理类，将自身的请求用 Real Subject 对应的功能来实现，代理类对象并不真正的去实现其业务功能。

3、Subject：定义 RealSubject 和 Proxy 角色都应该实现的接口。

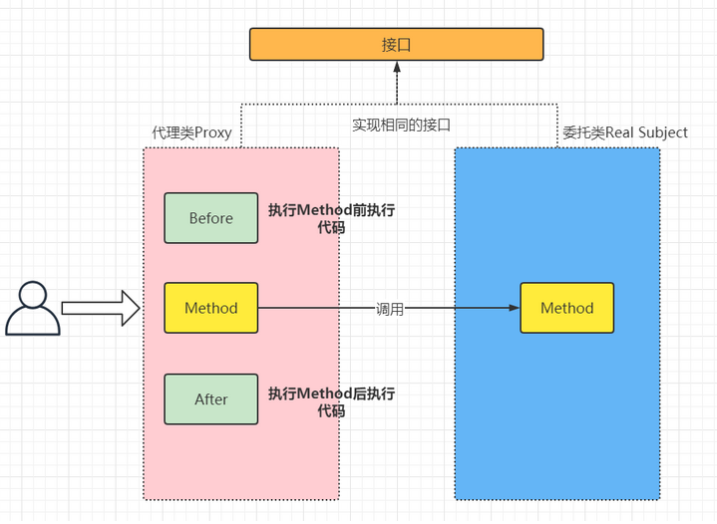


代理模式有三种类型，静态代理，动态代理（JDK代理，接口代理）、Cglib代理（在内存中动态的创建目标对象的子类）

# 静态代理

## 基本概念

静态代理需要先定义接口，被代理对象与代理对象一起实现相同的接口，然后通过调用相同的方法来调用目标对象的方法。

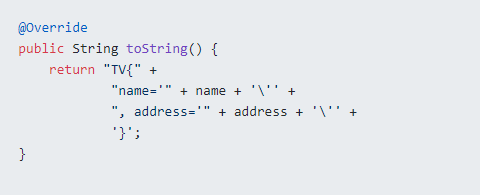


可以看见，代理类无非是在调用委托类方法的前后增加了一些操作。委托类的不同，也就导致代理类的不同。某公司生产电视机，在当地销售需要找到一个代理销售商。那么客户需要购买电视机的时候，就直接通过代理商购买就可以。

## 代码示例

### 1、创建电视机类





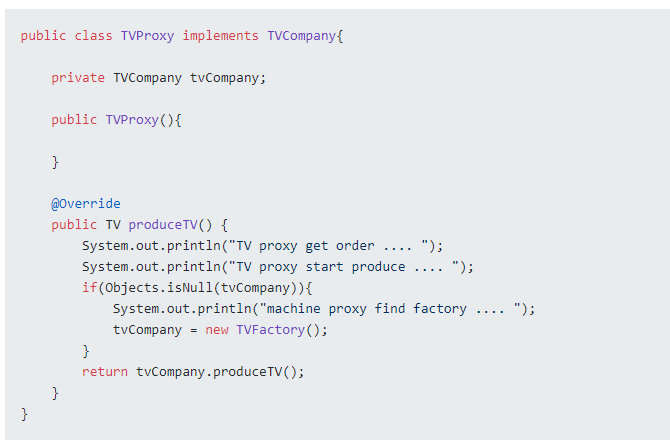
### 2、创建公司接口



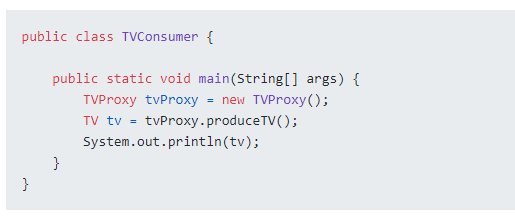
### 3、公司的工厂生产电视机



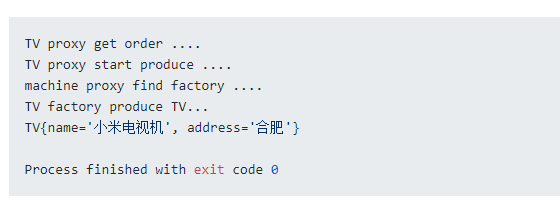
### 4、代理商去下单拿货（静态代理类）:



### 5、消费者通过代理商拿货（代理类的使用）



### 6、输出结果



## 优点

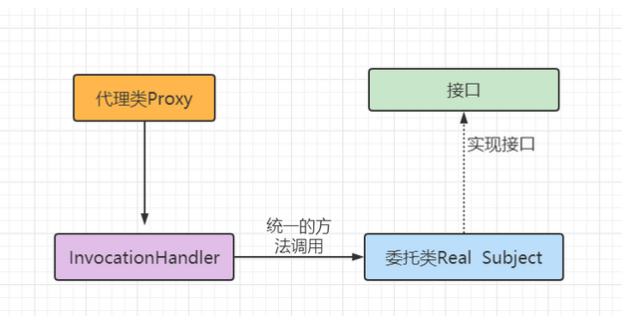
静态代理模式在不改变目标对象的前提下，实现了对目标对象的功能扩展。

## 缺点

静态代理实现了目标对象的所有方法，一旦目标接口增加方法，代理对象和目标对象都要进行相应的修改，增加维护成本。如何解决静态代理中的缺点呢？答案是可以使用动态代理方式

# 动态代理

## 基本概念

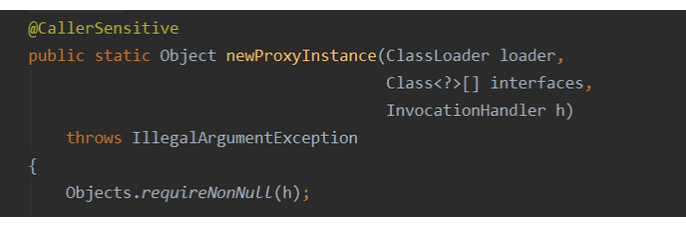


## 特点

1、JDK动态代理对象不需要实现接口，只有目标对象需要实现接口。

2、实现基于接口的动态代理需要利用JDK中的API，在JVM内存中动态的构建Proxy对象。

3、需要使用到 java.lang.reflect.Proxy和其newProxyInstance方法，但是该方法需要接收三个参数。



（1）ClassLoader loader：指定当前目标对象使用的类加载器，获取加载器的方法是固定的。

（2）Class<?>[] interfaces：目标对象实现的接口的类型，使用泛型方式确认类型。

（3）InvocationHandler h：事件处理，执行目标对象的方法时，会触发事件处理器的方法，会把当前执行目标对象的方法作为参数传入。

## 底层原理

由Java内部的反射机制来实现的。

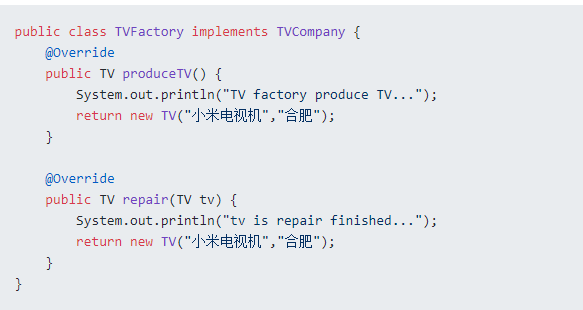
## 代码示例

有一天公司增加了业务，出售的商品越来越多，售后也需要更上。但是公司发现原来的代理商，还要再培训才能完成全部的业务，于是就找了另外的动态代理商B 。 代理商B 承诺无缝对接公司所有的业务，不管新增什么业务，均不需要额外的培训即可完成。

1、公司增加了维修业务



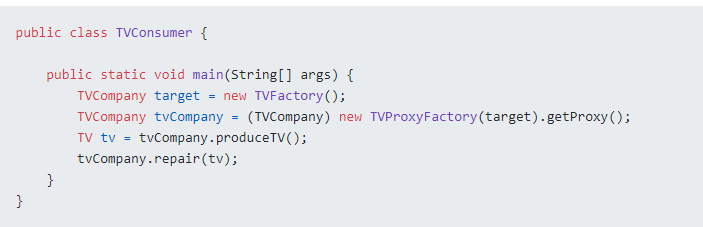
2、工厂也得把维修业务搞起来



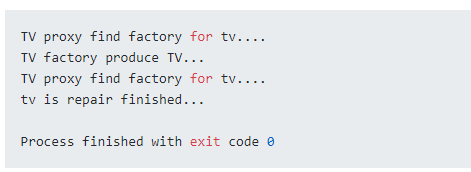
3、**B代理商** 全面代理公司所有的业务。使用Proxy.newProxyInstance方法生成代理对象，实现InvocationHandler中的 invoke方法，在invoke方法中通过反射调用代理类的方法，并提供增强方法。



**购买、维修这两个业务** B代理就可以直接搞定了。后面公司再增加业务，B代理也可以一样搞定。



输出结果：



## 总结

1、代理对象不需要实现接口，但是目标对象一定要实现接口，否则不能用动态代理。

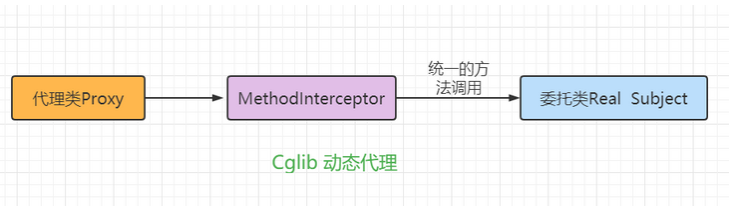
2、动态代理的方式中，所有的函数调用最终都会经过 invoke 函数的转发，因此我们就可以在这里做一些自己想做的操作，比如日志系统、事务、拦截器、权限控制等。

3、JDK 动态代理有一个最致命的问题是它只能代理实现了某个接口的实现类，并且代理类也只能代理接口中实现的方法，要是实现类中有自己私有的方法，而接口中没有的话，该方法不能进行代理调用。

# CGLIB代理

## 基本概念

静态代理和JDK代理都需要某个对象实现一个接口，有时候代理对象只是一个单独对象，此时可以使用Cglib代理。



Cglib代理可以称为子类代理，是在内存中构建一个子类对象，从而实现对目标对象功能的扩展。Cglib通过Enhancer 来生成代理类，通过实现MethodInterceptor接口，并实现其中的intercept方法，在此方法中可以添加增强方法，并可以利用反射Method或者MethodProxy继承类来调用原方法。

## 代码示例

看到 B代理商承接了公司（接口）的多种业务，那么此时C代理商又从中发现新的商机， B 只能代理某个公司的产品，而我不仅想要代理公司产品，而且对接不同的工厂，拿货渠道更广，赚钱更爽快。于是Cglib就用上了。

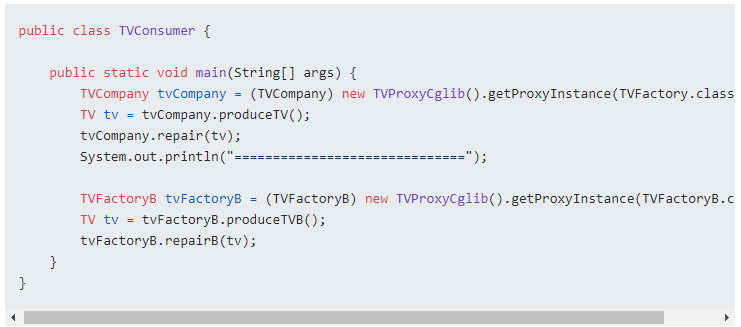
1、创建代理对象类



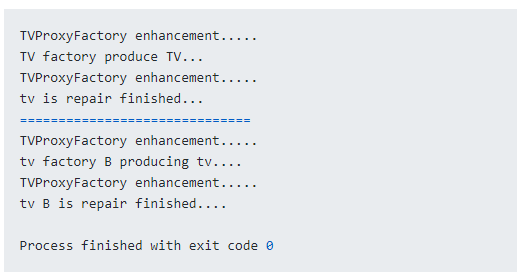
2、新代理的B工厂



3、C代理可以直接和公司合作，也可以和工厂打交道。并且可以代理任何工厂的产品。

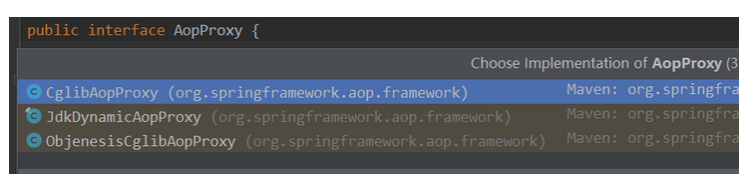


4、输出结果



# Spring AOP使用代理

Spring中AOP的实现有JDK和Cglib两种，如下图：



如果目标对象需要实现接口，则使用JDK代理。如果目标对象不需要实现接口，则使用Cglib代理。

# 总结

1、静态代理：需要代理类和目标类都实现接口的方法，从而达到代理，增强其功能。

2、JDK动态代理：需要代理类实现某个接口，使用Proxy.newProxyInstance方法生成代理类，并实现InvocationHandler中的invoke方法，实现增强功能。

3、Cglib动态代理：无需代理类实现接口，使用Cblib中的Enhancer来生成代理对象子类，并实现MethodInterceptor中的intercept方法，在此方法中可以实现增强功能。

# InvocationHandler接口

## 基本概念

每一个动态代理类都必须要实现InvocationHandler这个接口，并且每个代理类的实例都关联到了一个handler，当我们通过代理对象调用一个方法的时候，这个方法的调用就会被转发为由InvocationHandler这个接口的 invoke 方法来进行调用。

## 方法

### public Object invoke(Object proxy, Method method, Object[] args)

proxy：指我们代理的那个真实对象

method：指我们所要调用真实对象的某个方法的Method对象

args：指代的是调用真实对象某个方法时接受的参数

# Proxy类

## 基本概念

Proxy这个类的作用就是用来动态创建一个代理对象的类，它提供了许多的方法，但是我们用的最多的就是 newProxyInstance 这个方法。

## 方法

### public static Object newProxyInstance(ClassLoader loader,Class<?>[] interfaces,InvocationHandler h)

这个方法的作用就是得到一个动态的代理对象。

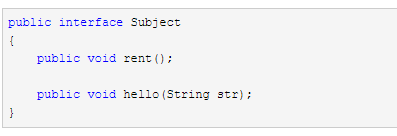
loader：一个ClassLoader对象，定义了由哪个ClassLoader对象来对生成的代理对象进行加载。

interface：一个Interface对象的数组，表示的是我将要给我需要代理的对象提供一组什么接口，如果我提供了一组接口给它，那么这个代理对象就宣称实现了该接口(多态)，这样我就能调用这组接口中的方法了

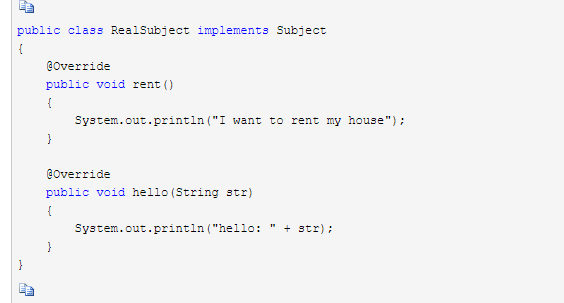
h：一个InvocationHandler对象，表示的是当我这个动态代理对象在调用方法的时候，会关联到哪一个InvocationHandler对象上。

## 代码示例

1、我们定义了一个Subject类型的接口



2、定义了一个类来实现这个接口，这个类就是我们的真实对象，RealSubject类：



3、我们就要定义一个动态代理类了，前面说个，每一个动态代理类都必须要实现 InvocationHandler 这个接口，因此我们这个动态代理类也不例外：



4、最后来看看我们的Client类：



5、输出结果

