代理模式

什么是代理?

代理模式，就是给一个对象提供一种代理对象以控制对该对象的访问。在这个模式中，我们通过创建代理对象作为“替身”替代了原有对象，从而达到我们控制对象访问的目的。

通俗来说，代理=代替处理。是由另一个对象来代替原对象来处理某些逻辑。

举个例子：房产中介，代跑腿业务，送外卖...

代理模式能带给我们什么?

代理模式能带给我们控制访问某个对象的能力，在某些情况下，一个对象的某些方法想要进行屏蔽或者某种逻辑的控制，则我们可以通过代理的方式进行。再此能力上，引申出来的作用，也是目前在开发中经常使用的一个作用，就是——“**在不修改原对象代码的基础上，对原对象的功能进行修改或者增强**”。

解耦

什么是耦合?

耦合本来是一个工业上的概念，指的就是齿轮像上图一样卡咬到一起，一个齿轮运转的时候，其他咬死的齿轮也会跟着发生运转。而此概念平移到我们的开发领域，就是项目中的各个模块或者各个组件，当一个发生变动或者调用的时候，其他有一些模块或组件也会跟着发生变动或者调用，产生了强烈的联系。这样的情况就说两个模块或组件耦合到一起了。

什么是解耦?

大家看这些齿轮，如果在运行过程中我发现一个齿轮大小不对要更换或者要移动位置，其他的齿轮会如何?

当模块或组件耦合度过高，会带来难以扩展、维护性差、纠错困难等各种问题，所以我们在设计的时候，要尽可能的避免模块或组件之间的耦合度过高。在设计过程中，削减耦合度或者消除耦合度的操作，就是解耦。

当我们想要给某个对象添加一些额外的逻辑时（例如访问权限的控制，日志的记录...），使用代理模式。我们可以不修改原代码，只针对这些额外的功能进行编码。在整个过程中，原对象的逻辑和额外增加的逻辑完全解耦，互不干扰。

高扩展性

由于模块间是解耦的，所以我们随时可以添加任意的功能或者修改之前的功能而不回影响到原模块的正常执行。相当于我们的代码像钢铁侠的战衣一样，可以随时添加各种战斗模组以适应不同的作战环境。这就是高扩展性。

Java中的代理模式

Java中代理有三种方式来创建代理对象：静态代理、基于JDK（接口）的动态代理、基于CGLIB（父类）的动态代理。

相关概念tips

目标类：原对象，我们需要通过代理对象控制它的访问，扩展其功能。

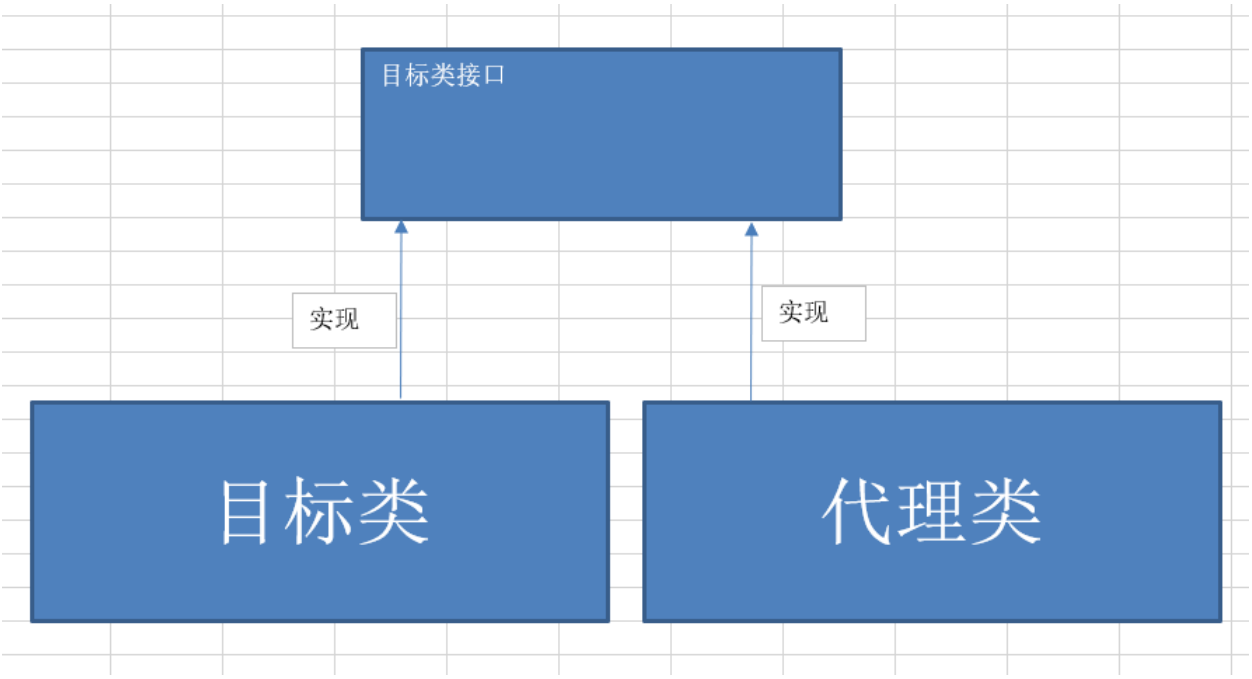
代理类：代理模式产生的对象，是原对象的“替身”，已经在原有基础上修改了逻辑。

静态代理

静态代理，也就是我们会手写代理类的代码，工程中有代理类的源码，代理类会编译后执行。

编写代理类，实现目标类的接口或者直接继承目标类，完成逻辑的修改。

接口实现方式：



继承方式：

分析：静态代理虽然能够实现我们所说的代理模式，完成了解耦，但是静态代理类的代码维护依然非常复杂。一旦接口或者父类发生了变动，则代理类的代码就得随之修改，代理类多的时候维护比较麻烦。所以在实际开发的时候，一般使用动态代理的方式。

动态代理

动态代理技术，是在内存中生成代理对象的一种技术。也就是整个代理过程在内存中进行，我们不需要手写代理类的代码，也不会存在代理类编译的过程，而是直接在运行期，在JVM中“凭空”造出一个代理类对象供我们使用。一般使用的动态代理技术有以下两种：

基于JDK（接口）的动态代理

JDK自带的动态代理技术，需要使用一个静态方法来创建代理对象。

它要求被代理对象，也就是目标类，必须实现接口。

生成的代理对象和原对象都实现相同的接口，是兄弟关系。

```
Proxy.newProxyInstance(ClassLoader loader,Class<?>[] interfaces,InvocationHandler h)
```

主要关注参数列表：

ClassLoader loader：固定写法，指定目标类对象的类加载器即可。用于加载目标类及其接口的字节码文件

通常，使用目标类的字节码对象调用getClassLoader()方法即可得到

Class<?>[] interfaces：固定写法，指定目标类的实现的所有接口的字节码对象的数组

通常，使用目标类的字节码对象调用getInterfaces()方法即可得到

InvocationHandler h：

这个参数是一个接口，主要关注它里面唯一一个方法，invoke方法。它会在代理类对象调用方法时执行，也就是说，我们在代理类对象中调用任何接口中的方法时，都会执行到invoke中。所以，我们在此方法中完成对增强或者扩展代码逻辑的编写。

```
Object invoke(Object proxy, Method method, Object[] args)
```

proxy：就是代理类对象的一个引用，也就是Proxy.newProxyInstance的返回值，此引用几乎不回用到，忽略即可。

method：对应的是触发invoke执行的方法的Method对象。假如我们调用了xxx方法，该方法触发了invoke的执行，那么，method就是xxx方法对应的反射对象（Method对象）

args：代理对象调用方法时，传递的实际参数

总结：

基于接口的动态代理，实际上是在内存中生成了一个对象，该对象实现了指定的目标类对象拥有的接口。所以代理类对象和目标类对象是兄弟关系。

兄弟关系：并列的关系，不能互相转换，包容性比较差。在后续会学习Spring框架，如果配置JDK的动态代理方式，一定要用接口类型接收代理类。

基于CGLIB（父类）的动态代理

第三方CGLIB的动态代理技术，也是可以使用一个静态方法来创建代理对象。

它不要求目标类实现接口，但是要求目标类不能是最终类，也就是不能被final修饰。

因为CGLIB是基于目标类生成改类的一个子类作为代理类，所以目标类必须可被继承。

```
Enhancer.create(Class type, Callback callback)
```

主要参数列表：

Class type：指定我们要代理的目标类的字节码对象，也就是指定目标类的类型

Callback callback：此单词的意思叫做回调，意思就是我们提供一个方法，它会在合适的时候帮我们调用它。回来调用的意思。

Callback是一个接口，由于该接口只是一个名称定义的作用，并不包含方法的声明。所以我们使用时通常使用它的一个子接口MethodInterceptor，此单词的意思叫做方法拦截器。

MethodInterceptor接口中也只有一个方法，叫做intercept

```
Object intercept(Object proxy, Method method, Object[] args, MethodProxy methodProxy)
```

proxy：就是代理类对象的一个引用，也就是Enhancer.create的返回值，此引用几乎不回用到，忽略即可。

method：对应的是触发intercept执行的方法的Method对象。假如我们调用了xxx方法，该方法触发了intercept的执行，那么，method就是xxx方法对应的反射对象（Method对象）

args：代理对象调用方法时，传递的实际参数

methodProxy：方法的代理对象，一般也不作处理，可以暂时忽略

总结：

基于父类的动态代理，是在内存中生成了一个对象，该对象继承了原对象（目标类对象）。所以代理类对象实际上是目标类对象的儿子。

父子关系：父子关系，代理类对象是可以用父类的引用接收的。

总结

1. 代理模式在Java开发中是广泛应用的，特别是在框架中底层原理经常涉及到代理模式（尤其是动态代理）
2. 静态代理和动态代理，实际使用时还是动态代理使用的比较多。原因就是静态代理需要自行手写代码，维护、修改非常繁琐，会额外引入很多工作量。也不能很好的使用配置完成逻辑的指定，所以使用较少。
3. 基于JDK和基于CGLIB，实际使用时两个都会用到。
    1. 在spring中，默认情况下它就支持了两种动态代理方式。如果你指定的目标类实现了接口，spring就会自动选择jdk的动态代理。而如果目标类没有实现接口，则spring会使用CGLIB。
    2. 我们在开发时，由于基于JDK的动态代理要求比较多，更不容易实现，所以很多人习惯于统一配置为使用CGLIB进行代理。也就是CGLIB更通用。
    3. 如果使用dubbo+zookeeper，底层进行代理时，最好配置定死使用CGLIB的方式进行代理。因为dubbo会使用基于包名的扫描方式进行类的处理，而JDK的代理类生成的包名类似于com.sun.proxy....格式。我们实际需要让代理类和目标类保持一样的包名，所以只有CGLIB能保持原包名不变生成代理类。