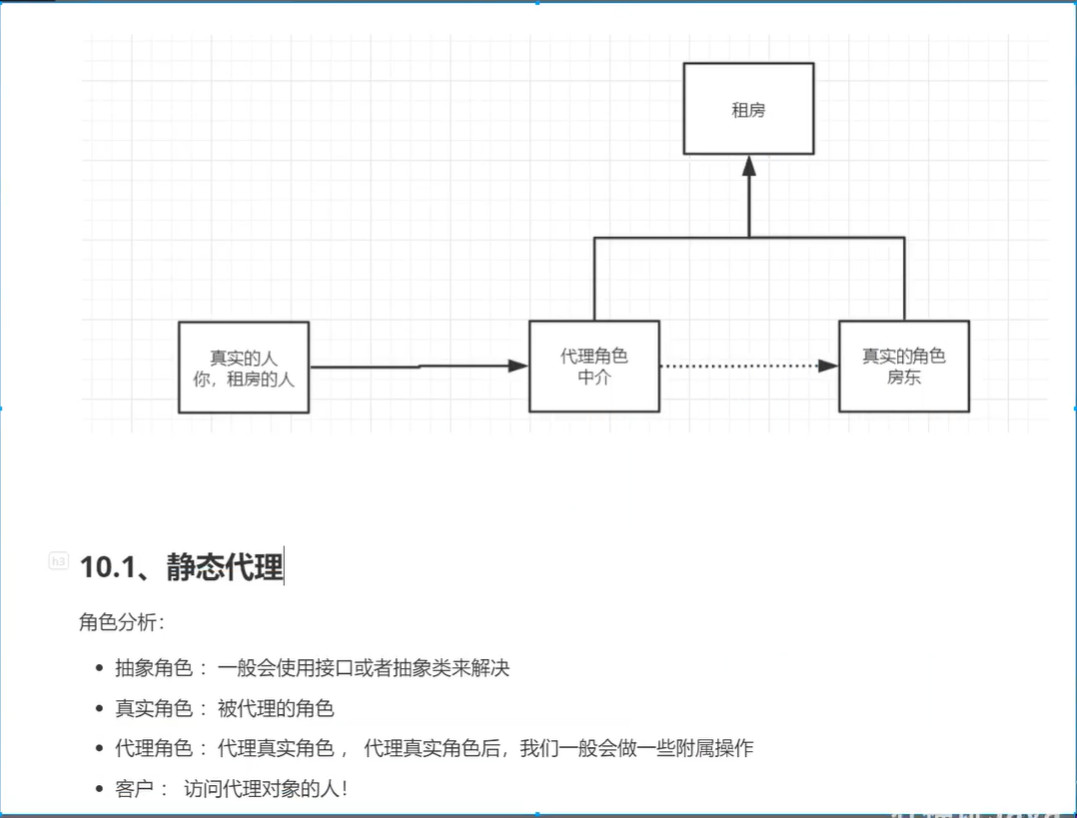
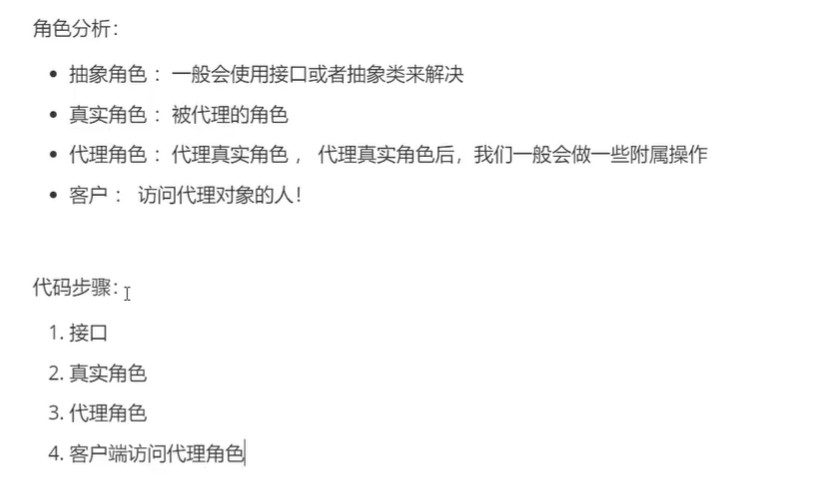
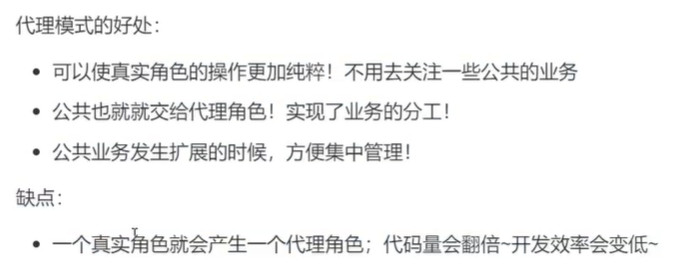
1、静态代理模式

SpringAOP的底层基于代理模式实现。

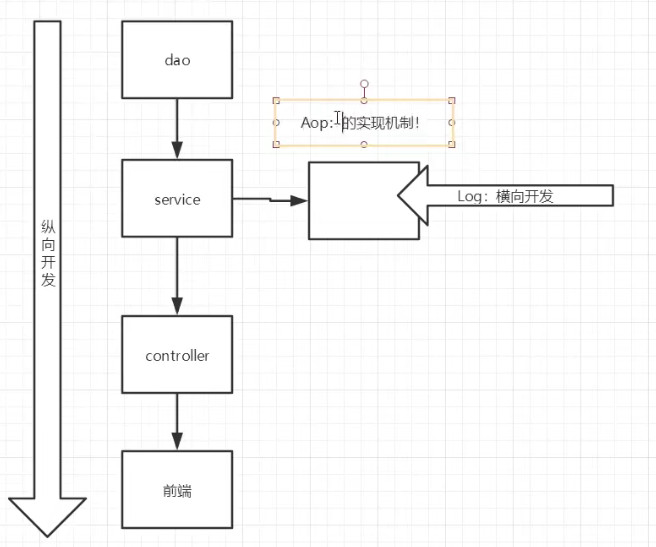




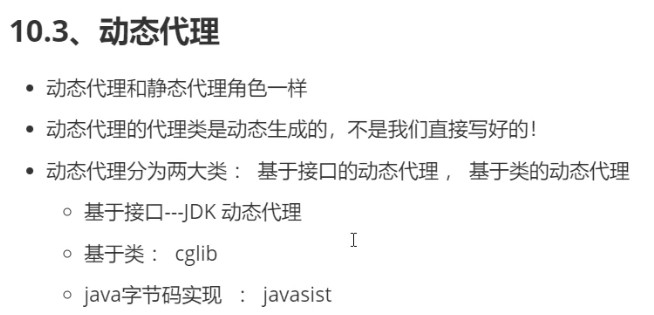
感觉使用代理模式做测试的驱动模块挺不错的。



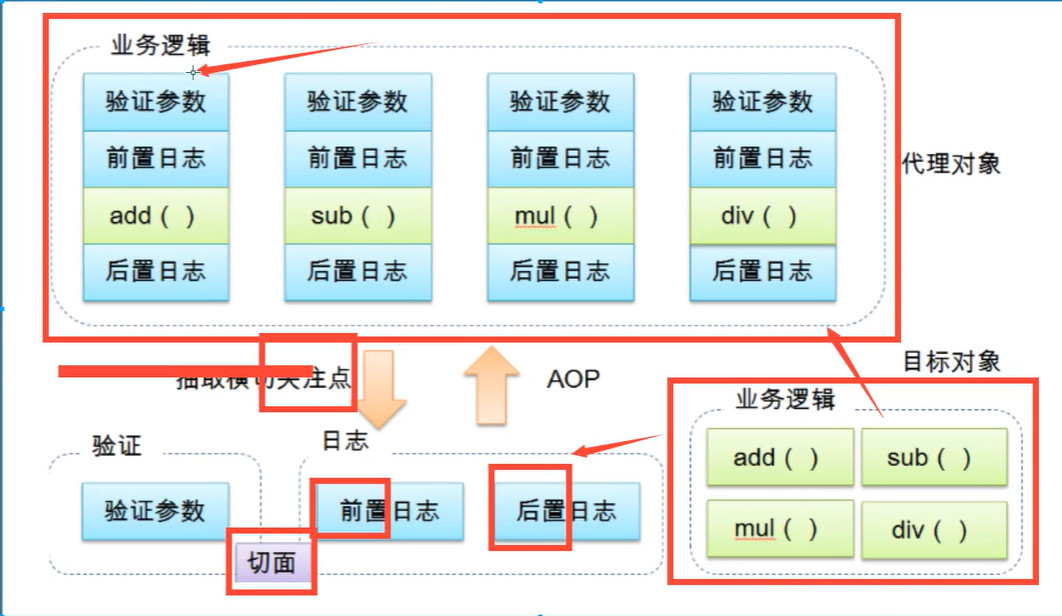
AOP：



动态代理：







切面就是：我们的需要增强的功能所封装为的类。（对应代理模式中的增强器：Invocation、MethodInterceptor）

通知就是：就是待增强的功能点。即是切面中的方法。（对应invoke（）、interceptor()方法）

目标：就是想要做增强的对象。（对应代理对象中的target成员）

代理：就是代理实例。（对应Proxy.newInstance或者enhancer.create()获取到的对象。）

切入点：就是待增强的位置（对应目标对象中的方法，通过切入点表达式进行定位。所以每个方法以及整个类都可以作为切入点）

Join Point（连接点）：连接点是一个虚拟的概念，可以理解为所有满足切点扫描条件的所有的时机。method="beforeAdvice"表示被扫描到目标方法后确定增强的时机，也就是连接点，这个表示前置一个增强（这里翻译Advice更好一点，而不是通知）

Advice（增强）：增强是一个具体的函数，例如，日志记录，权限验证，事务控制，性能检测，错误信息检测等。

Introduction（引入）: 添加方法或字段到被增强的类，Spring允许引入新的接口到任何被增强的对象。例如，你可以使用一个引入使任何对象实现 IsModified接口，来简化缓存。Spring中要使用Introduction, 可有通过DelegatingIntroductionInterceptor来实现增强，通过DefaultIntroductionAdvisor来配置Advice和代理类要实现的接口，比如我们自定义实现了

Weaving（织入）：组装切面来创建一个被通知对象。这可以在编译时完成（例如使用AspectJ编译器），也可以在运行时完成。Spring和其他纯Java AOP框架一样，在运行时完成织入。等候时机（连接点）。也就是采用代理技术，动态代理还是静态代理。（就是动态代理织入-插入整合出一个代理类，我们使用jdk代理实现InvocationHandler接口的类就可以称作一个织入器）

（下面这张图讲述的意思是：现在有定义好的切面以及通知若干，以及目标对象若干，目标对象有自己关注的切面及通知，现在需要将切面横切入到相关的目标对象中，生成各自的代理对象。

所以重点在于这些通知如何准确的定位到目标位置，这里使用的是目标被动得接收，通过扫描通知的切入点，定位所有符合的切入点，然后将通知通过织入的方式（其实就是切入）切入到目标对象中，但是我们我们不可能每次将通知的代码直接拷贝一份到对应的位置，这样代码量可想而知的，所以我们应该只需要在目标位置-切入点声明一个通知-连接点的位置，知道在什么时候应该转入到相应的通知的位置。这样就生成一个代理实例，通过代理实例在执行时，当遇见连接点时我们就跳转到连接点位置进行执行。

）

（可以这么说：切入点指向目标对象，在切面中，连接点指向切面，在目标对象中-准确的说应该是在代理对象中。）

（我们往往是在切面中声明切入点即可，连接点会在生成代理对象时自己创建。为什么不是在目标对象中声明连接点呢？

我认为理论上是可行的。

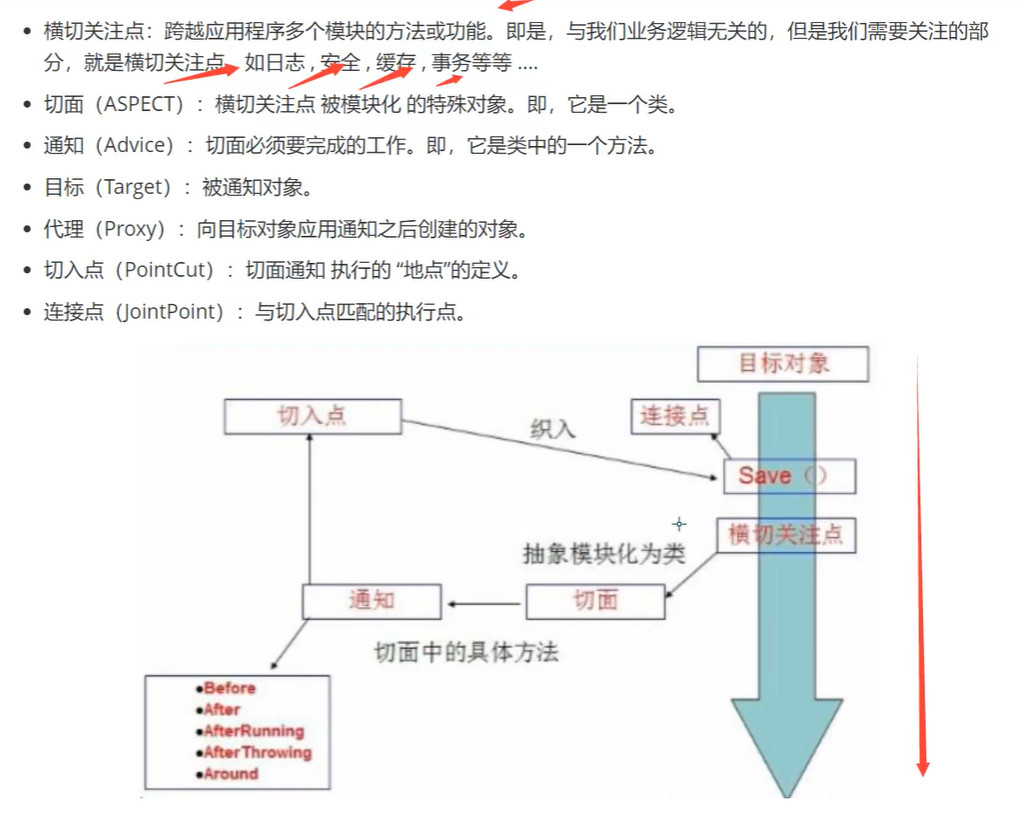
但是为什么会没使用，我认为原因有以下几点：  
切面往往是公共的执行资源，数量不多且尽量少改变，为某一类符合条件的对象进行服务，而目标对象往往是多、易变。如果我们直接在目标对象中进行连接点申明：

1、工作量大，每一个方法我们需要使用到某一个通知时都需要进行申明。

2、目标对象源码容易改变，当我需要使用新的通知时我们需要频繁的改变源码，而目标对象往往与我们的业务逻辑相关，而切面者往往是公共使用的重复为了减缓冗余代码而存在的。这样就会违背开闭原则且让业务维护难度增加。而使用织入，使得公共资源与业务逻辑进行分离，耦合度降低。

3、使用代理模式无法使用，没有意义，因为我们的目标对象中已经确定了所有的连接点。其实使用动态代理就是解决第二点问题的方式。通过动态的生成代理对象，不用改变业务相关代码，需求的改变通过代理实例进行增强就行。

）



切入点表达式以及切入点重用：[spring相关—AOP编程—切入点、连接点 - 康星悦 - 博客园 (cnblogs.com)](https://www.cnblogs.com/kangxingyue-210/p/7449924.html)

[(3条消息) aop前置通知，后置通知，返回通知，环绕通知执行顺序\_luo\_yu\_1106的博客-CSDN博客\_环绕通知什么时候执行](https://blog.csdn.net/luo_yu_1106/article/details/89513076)

