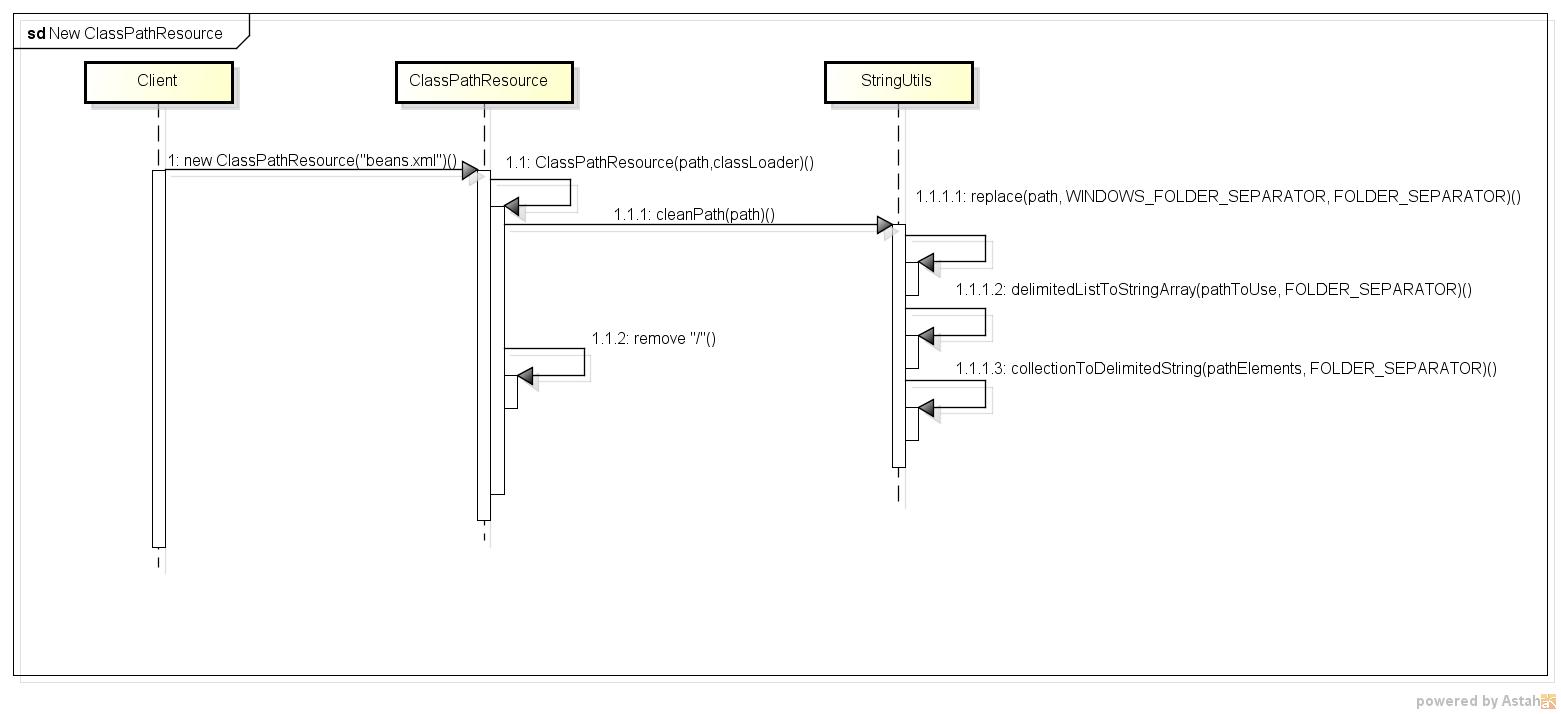
# IOC源代码分析

## ClassPathResource resource = new ClassPathResource("beans.xml");源代码分析：

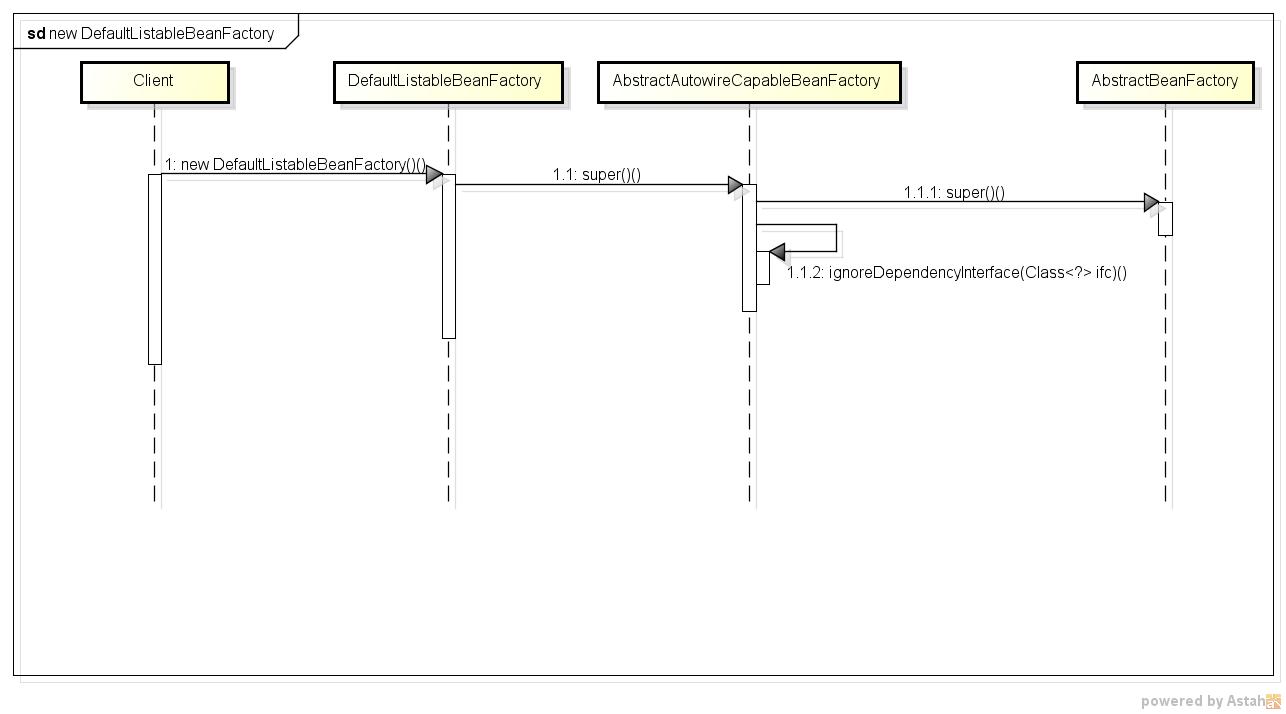


* 这一行语句只是将spring配置文件的路径给解析出来，并没有真正的去读取配置文件
* 针对cleanPath()的解析
* 将windows的目录分隔符”\”替换为unix下的目录分隔符"/”
* 使用delimitedListToStringArray方法来将路径拆分开，如a/b/beans.xml将被作为字符串数组[a,b,beans.xml]返回
* collectionToDelimitedString则是将上面的字符串数组再将组成/a/b/beans.xml的形式，为何要这样做，暂时不明。
* 最后如果文件路径以”/”开头，则将”/”删除，因为ClassLoader不允许这样的路径名称

总结：

在cleanPath()中，spring首先是把路径的第一个字符串”/” substring掉（如果以它开头的话），然后将substring后的字符串delimited和collection的操作，在返回的结果的时候，再将”/”拼在结果的前面返回，再最后在ClassPathResource的构造方法里再把”/”删除。cleanPath()这个方法有很多地方来调用，所以可以看出spring为了达到方法重用的目的，并不太在乎进行一些重复的操作或者一些内存上或者性能上的损耗。

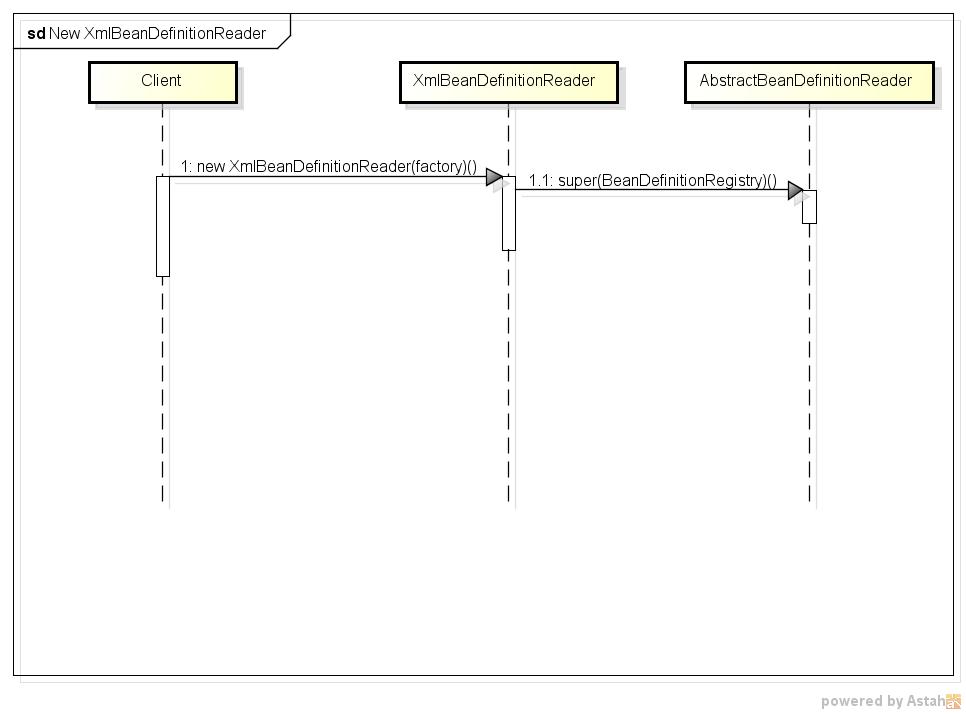
## DefaultListableBeanFactory factory = new DefaultListableBeanFactory();



总结：

这个过程没有什么特别的地方，都是显式的调用了父类的super构造方法，对一些属性方面的东西进行了初始化，最后调用了ignoreDependencyInterface这个方法，专门为autowiring服务的。

## XmlBeanDefinitionReader reader = new XmlBeanDefinitionReader(factory);

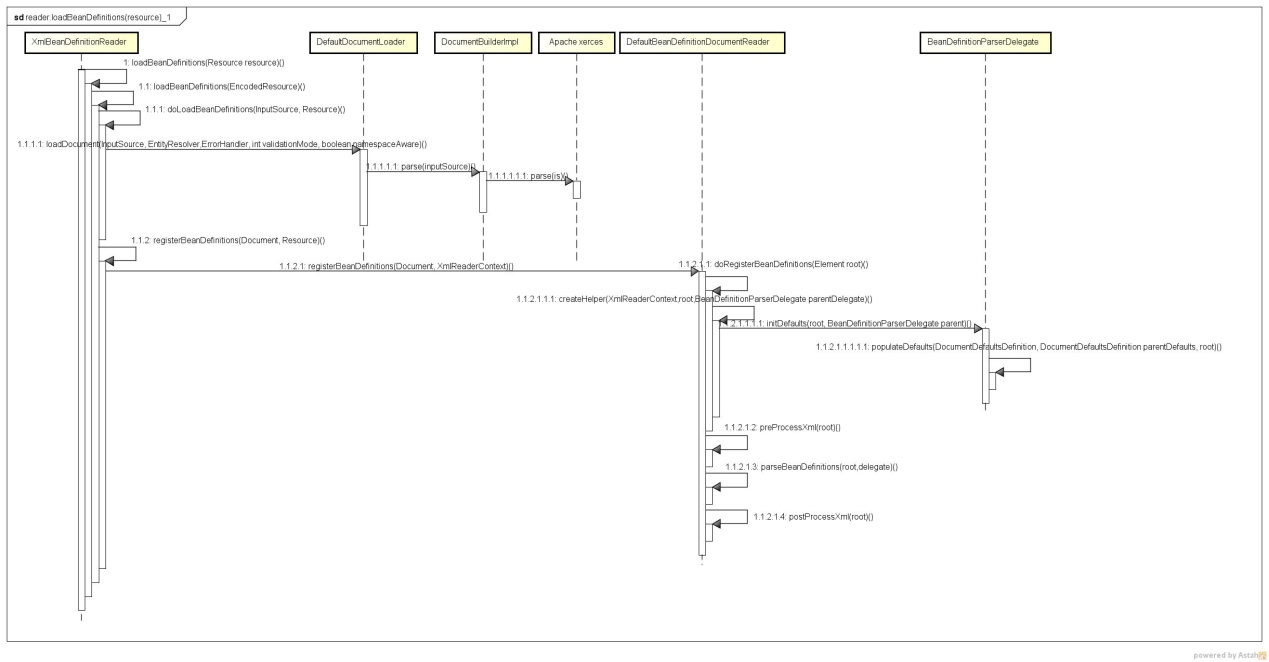


总结：

通过上图可以看出，DefaultListableBeanFactory其实就是一个BeanDefinitionRegistry。

真正的初始化是在AbstractBeanDefinitionReader中的，但也只是初始化了ResourceLoader和Environment这两个属性。前者在我们的demo时是被初始化为PathMatchingResourcePatternResolver，这个是路径解析器，用来解析绪如“classpath\*:”这样的配置的，后者会被初始化为StandardEnvironment，这是用来指明当前是应用是处于什么环境中的（具体还不是太明白其作用，貌似与”profile”这些配置有关）

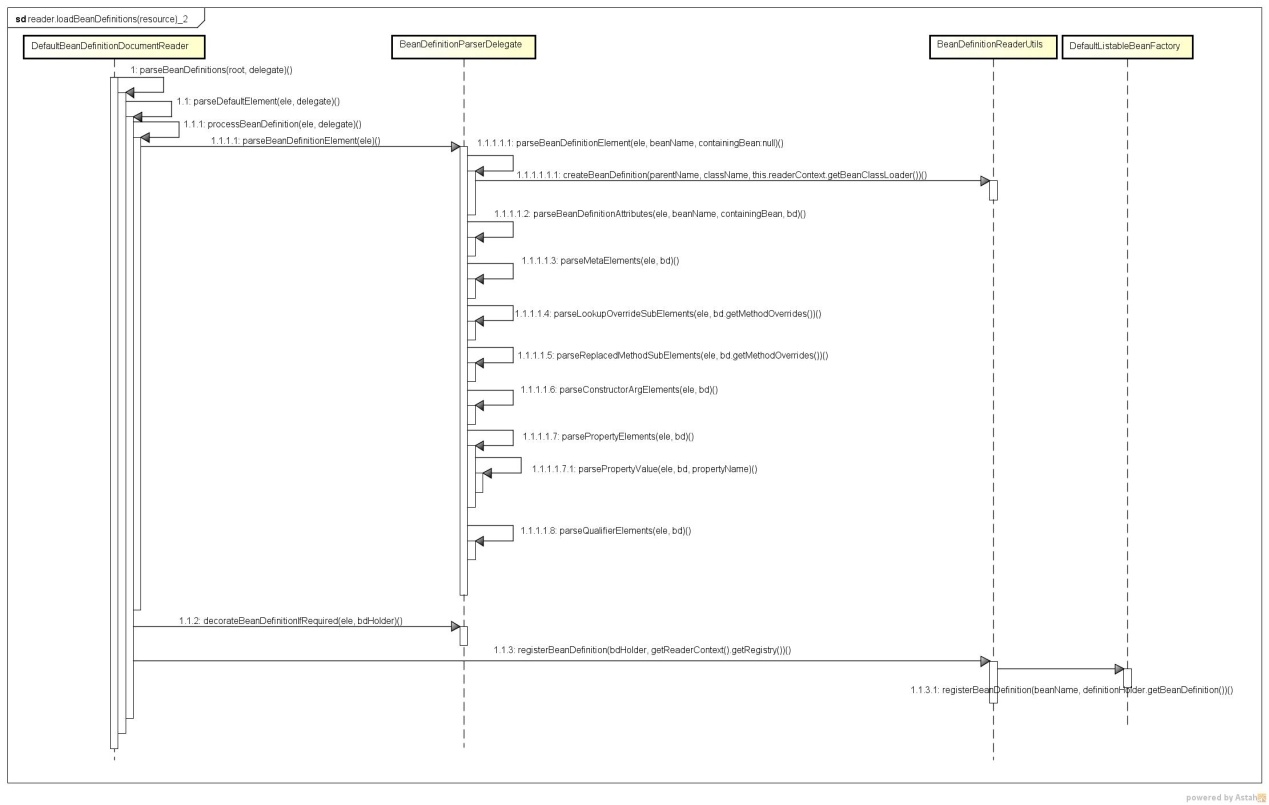
## reader.loadBeanDefinitions(resource)详解



解析：

* spring将传入的resource对象封装成EncodedResource（这时它的encoding和charset都是null）
* loadBeandefinitions中会做一些安全的检查来看当前的配置文件是不是已经正在加载中，如果是则报错，最后通过resource将InputStream读取出来并封装在InputResource对象里。
* 在doLoadBeanDefinitions里：
* 调用DefaultDocumentLoader来加载配置文件，再协助Apache的Xerces项目将其内容封装成Document对象
* 再调用registerBeanDefinitions等一系列方法之后，在DefaultBeanDefinitionDocumentReader中调用createHelper方法，在这个方法中会将一个已经初始化如"default-lazy-init","default-merge","default-destroy-method","default-init-method"等这些值的BeanDefinitionParserDelegate返回
* 再调用prePreocessXml(root),parseBeanDefinitions(root,delegate),postProcessXml(root)这三个方法，其中前后两个是模板方法，默认是空实现，到目前为止spring也没有任何类来继承并重写它。真正的配置内容封装是在parseBeanDefinitions()这个方法中

## reader.parseBeanDefinitions详解



下载完spring的源代码后，如果在./gradlew build 时出错，提示下载一个zip包里连接超时的话，编辑gradle.properties文件，在里面添加如下内容：

|  |
| --- |
| systemProp.http.proxyHost=127.0.0.1  systemProp.http.proxyPort=8087  systemProp.http.proxyUser=  systemProp.http.proxyPassword= |

spring的源代码在github上面默认的是下载master这个branch，这个branch需要JDK1.8的支持才能正确编译，所以我们只需要下载3.2.x这个branch就可以了：

|  |
| --- |
| git clone -b 3.2.x git://github.com/SpringSource/spring-framework.git |

Spring加载资源并装配对象的过程：

ClassPathResource resource = new ClassPathResource("beans.xml");

DefaultListableBeanFactory factory = new DefaultListableBeanFactory();

XmlBeanDefinitionReader reader = new XmlBeanDefinitionReader(factory);

reader.loadBeanDefinitions(resource);

1. 定义好Spring的配置文件。

2. 通过Resource对象将Spring配置文件进行抽象，抽象成一个Resource对象。

3. 定义好Bean工厂（各种BeanFactory）。

4. 定义好XmlBeanDefinitionReader对象，并将工厂作为参数传递进去供后续回调使用。

5. 通过XmlBeanDefinitionReader对象读取之前抽象出的Resource对象（包含了XML文件的解析过程）。

6. IoC容器创建完毕，用户可以通过容器获取到所需的对象信息。

Spring解析下面的bean定义的方式

<Beans>

<Beans/>

</Beans>

DocumentDefaultsDefinition.java描述的是beans tag层次的属性

在DefaultBeanDefinitionDocumentReader类中的doRegisterBeanDefinitions方法使用了经典的模板设计模式，子类可以重写preProcessXml()和postProcessXml()方法，实现对XML配置文件的自定义扩展。类似于Junit的setUp，testXXX与tearDown方法。

解析XML文件的时候，有一个地方使用了门面模式。

Spring默认没有在初始化容器后马上创建对象（可以通过配置修改这种行为），而是在getBean方法调用时才初始它对应的对象。

bean先被创建好，然后再set它的属性

* Spring的bean实际上是缓存在ConcurrentHashMap对象中
* 在创建Bean之前，首先需要将该bean的创建标识设定好，表示该bean已经或是即将被创建，为的是增强缓存的效率
* 根据bean的scope属性来确定是singleton还是prototype等范围，然后创建相应的bean对象
* 通过Java反射来创建Bean实例时，在创建之前首先检查访问修饰符，如果不是public的，则调用setAccesible(true)来突破Java的语法限制，使得可以通过私有的构造方法来创建实例
* 接下来，寻找bean的属性值，完成属性的注入
* 将所创建出的singleton对象添加到缓存当中，供下次使用

有可能是因为一开始的时候spring只支持singleton和prototype两个scope，后来才支持的request等类型，所以在根据scope的类型来创建bean实例的时候并没有使用策略模式。

当应用有多个bean定义的XML文件的时候，后加载的配置文件会覆盖前面的配置

Struts2也有一个Ioc容器，但它缓存的是创建方式的本身？？而spring缓存的是对象本身。

Spring的scope默认是singleton，找出在那里定义的。

当使用构造参数来初始化bean时，spring是怎样区别出不同的参数类型的？

DocumentBuilderFactory🡪DocumentBuilder

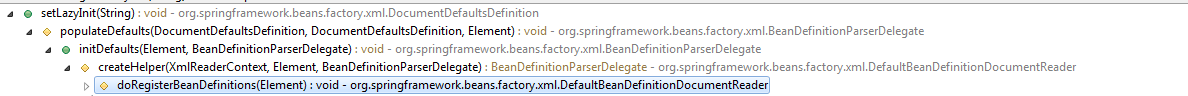
com.sun.org.apache.xerces.internal.impl.XMLEntityManager.setupCurrentEntity()这个方法可以看到spring是怎么判断配置文件的encoding的

core/src/main/java/org/springframework/util/ClassUtils.java 在这个类里可以看到spring的static final常量命名并不符合JAVA的编程规范，如：

|  |
| --- |
| **private** **static** **final** Map<String, Class<?>> ***primitiveTypeNameMap*** = **new** HashMap<String, Class<?>>(32); |

在创建BeanDefinitionParserDelegate时，就已经将一个bean的默认属性设置好了：

**this**.delegate = createHelper(**this**.readerContext, root, parent);



BeanDefinitionParserDelegate.parseMapElement写得太长了。。

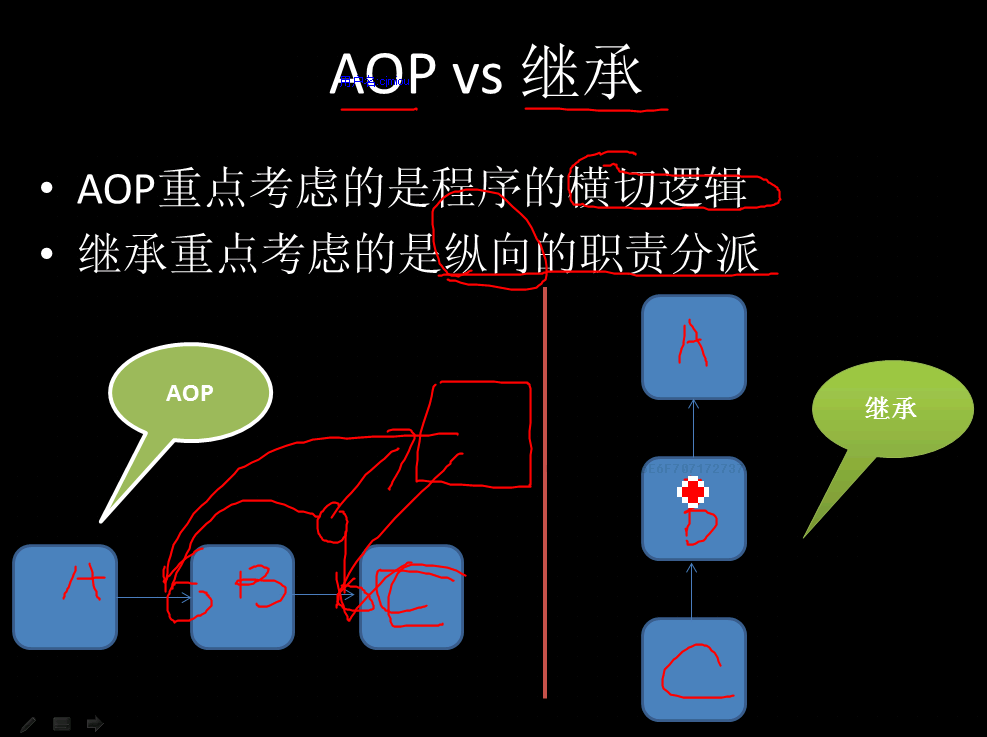
在创建bean的对象实例时，spring使用了一个prototypesCurrentlyInCreation类型为NameThreadLocal（其实就是继承自TheadLocal且增加了一个name属性）来判断是否spring的配置文件出现了环配置：先拿出当前prototypesCurrentlyInCreation中的name，再和当前要创建的类的实例的ID或name来比较，如果相同，则抛BeanCurrentlyInCreationException

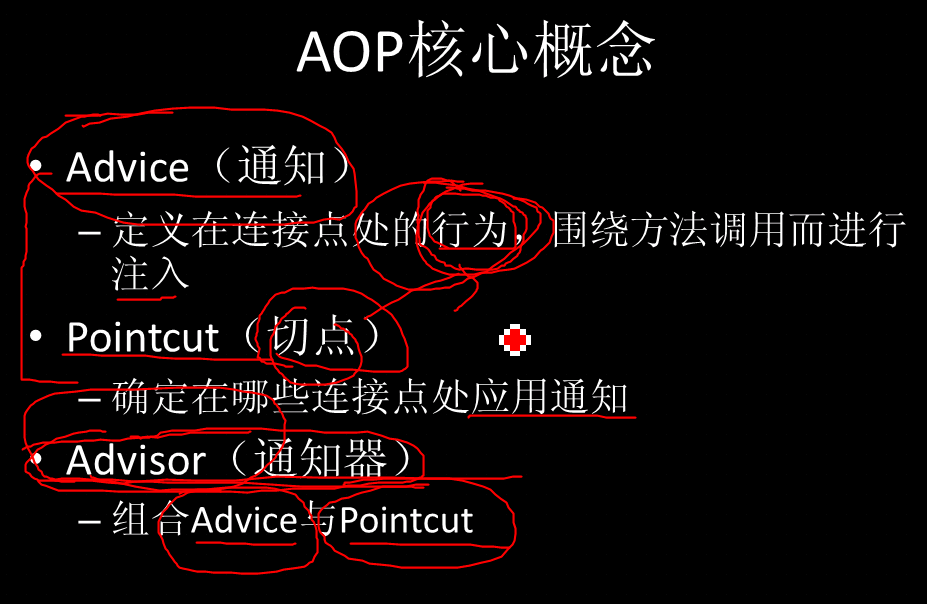
# AOP

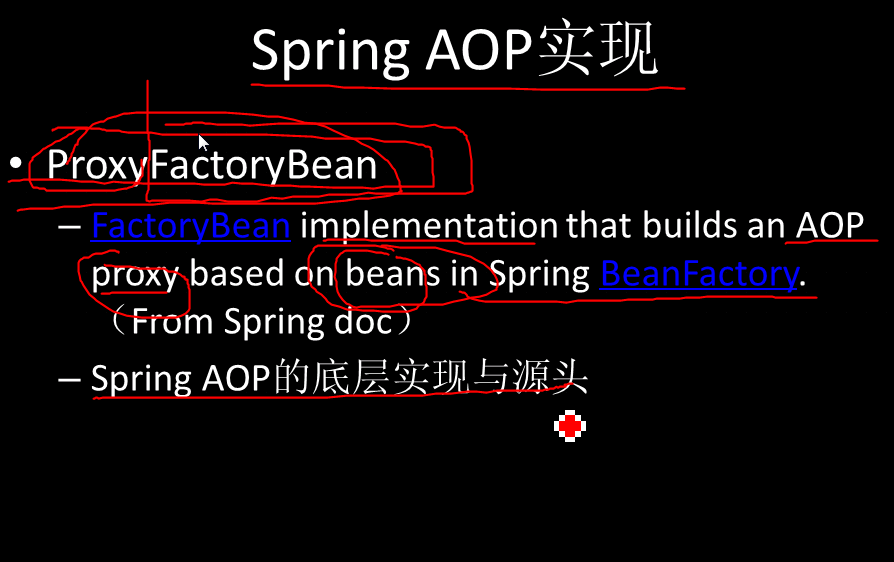
## AOP的目标

将分散在程序各处的横切关注点剥离出来并以集中的方式进行表达

使得开发人员能够专注于业务逻辑的实现而非繁杂的非功能代码简化了程序编写与单元测试

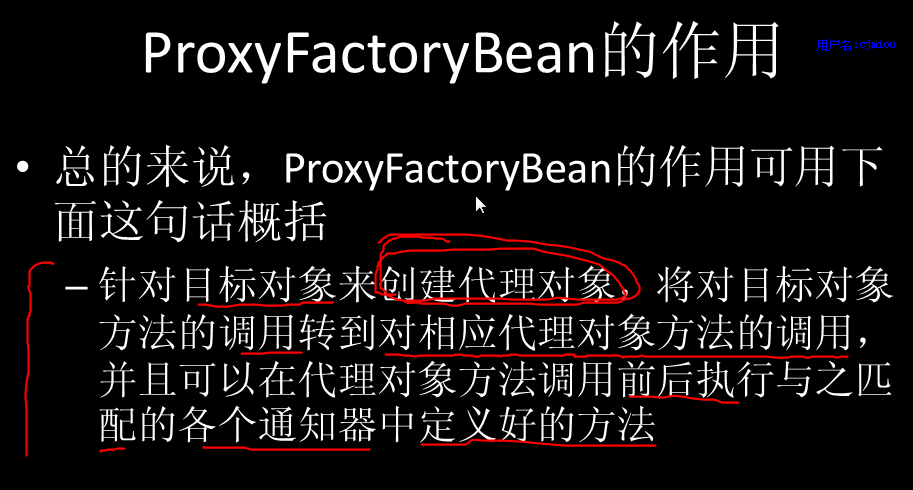


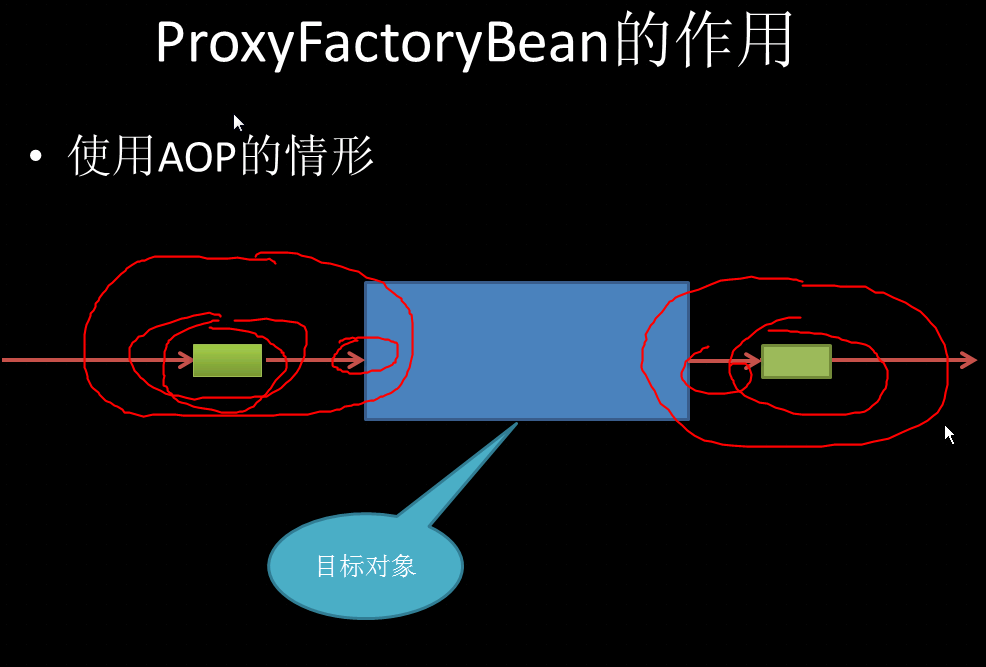


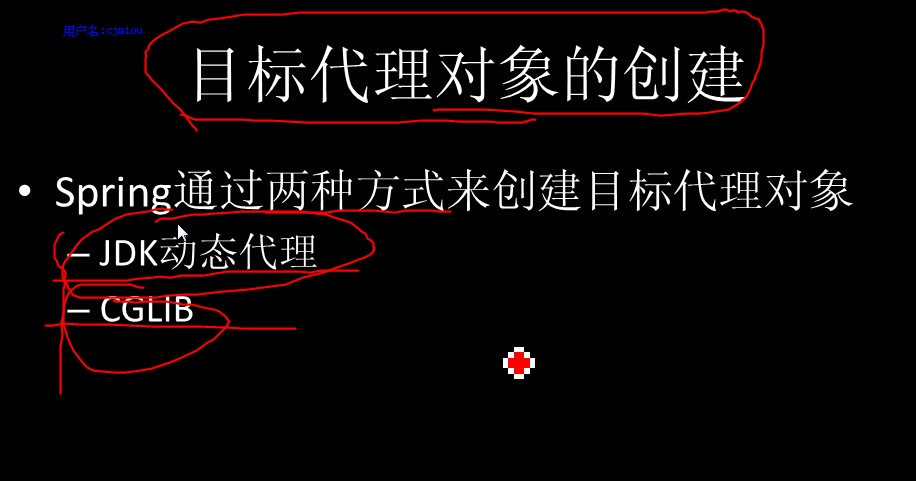


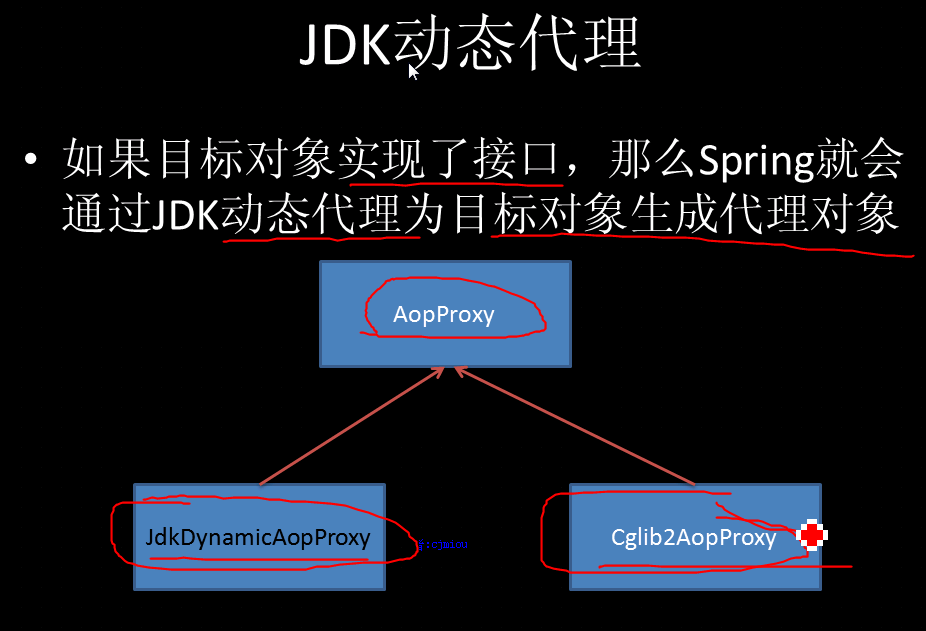




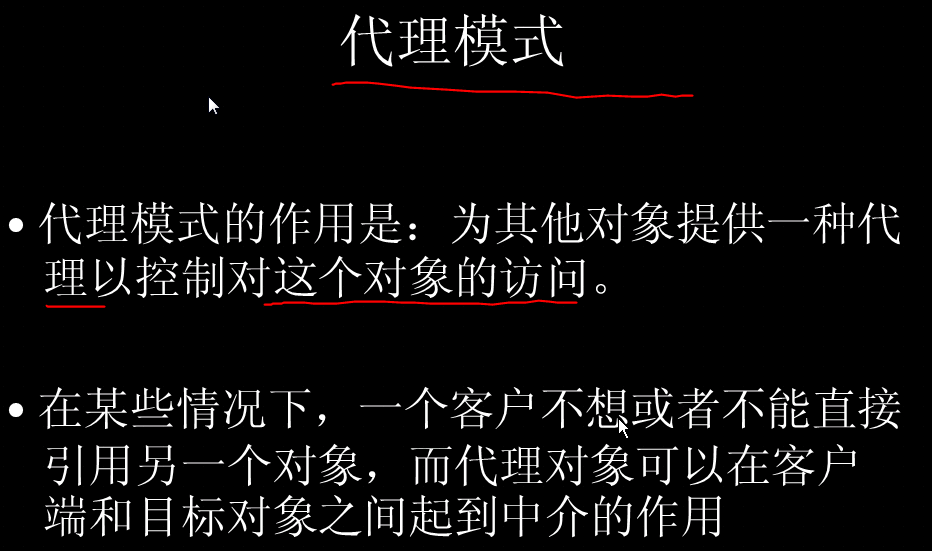


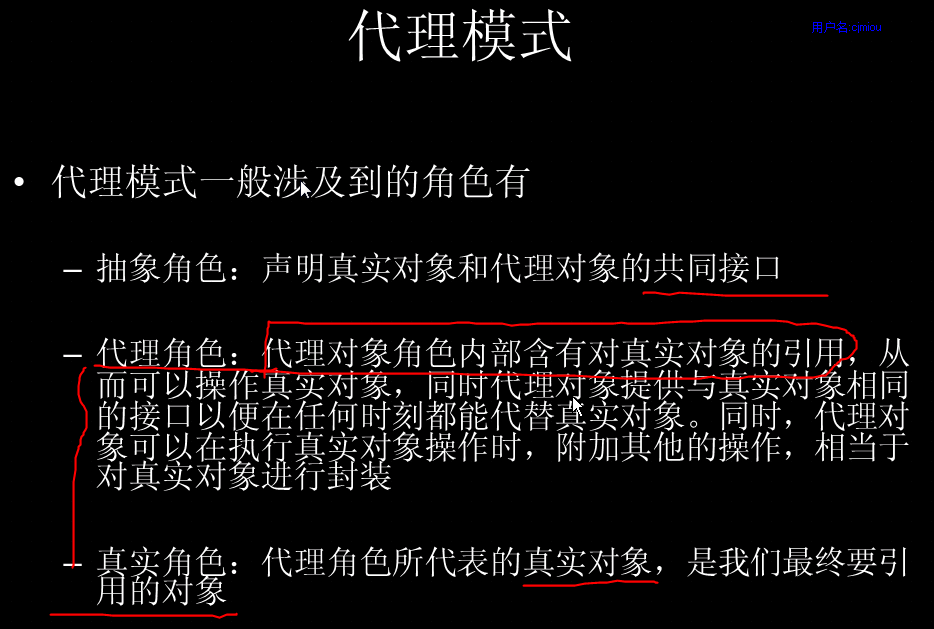


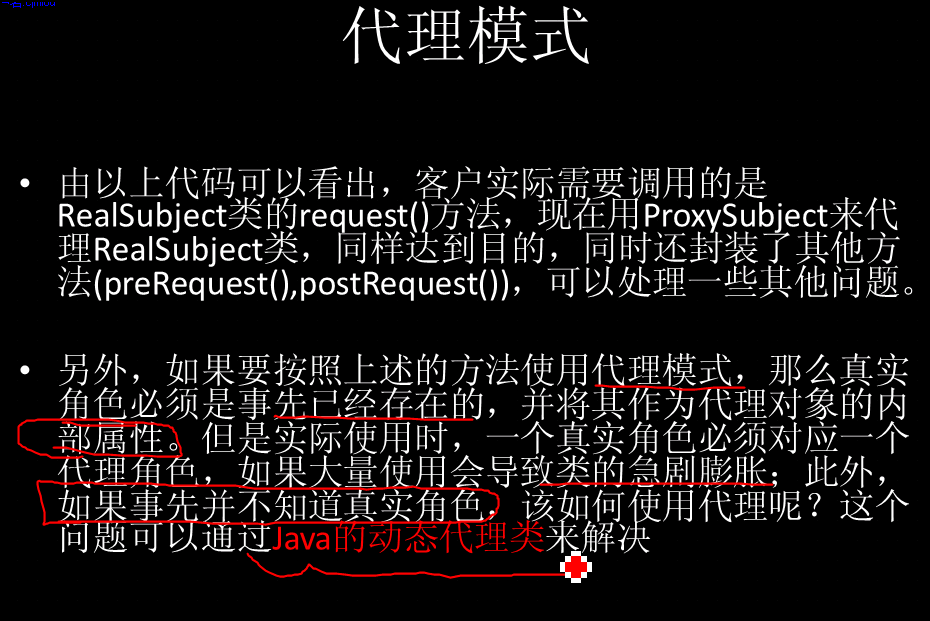


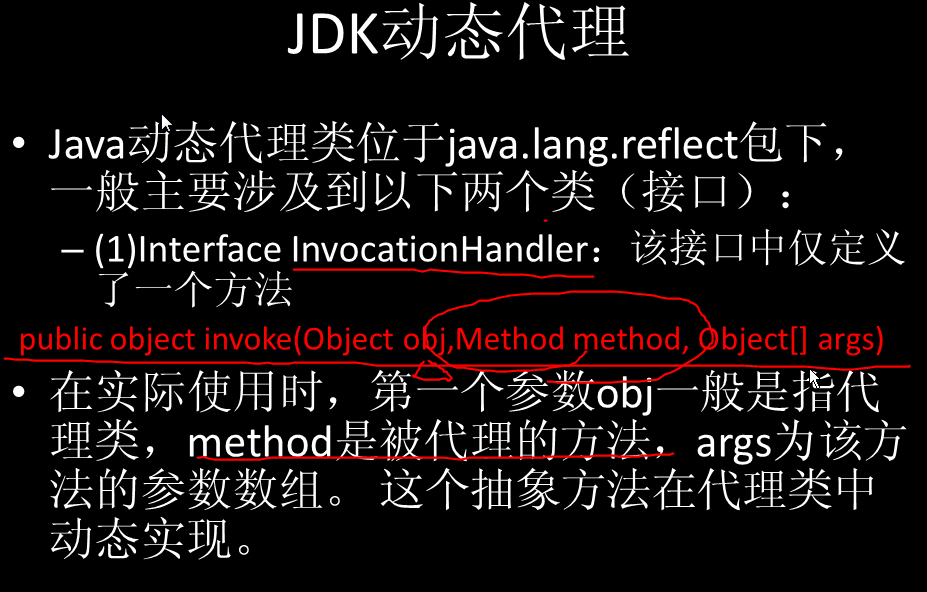


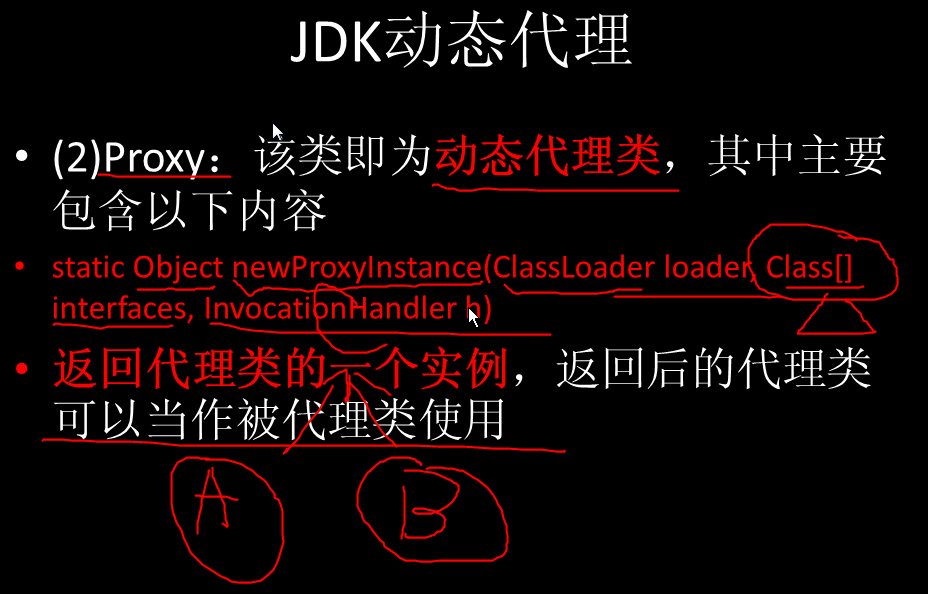


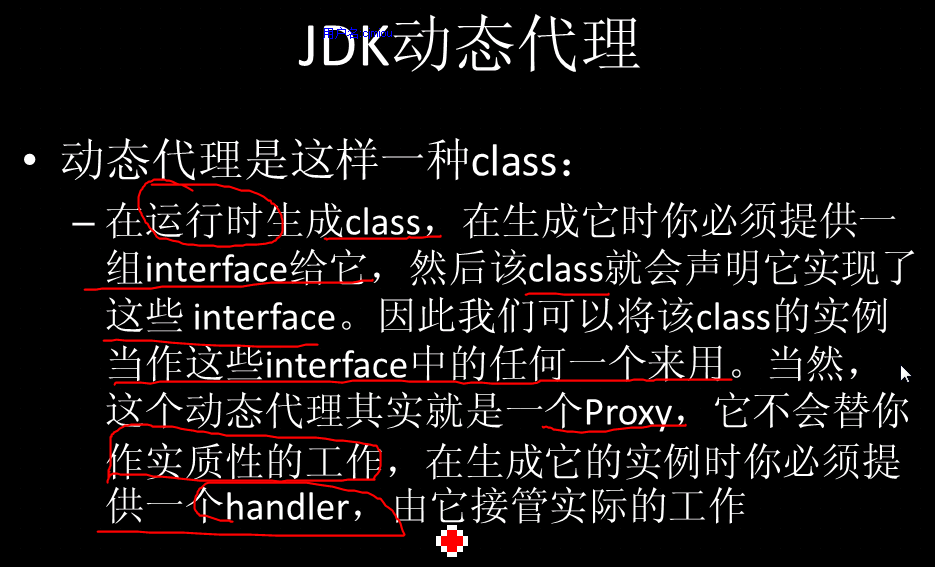






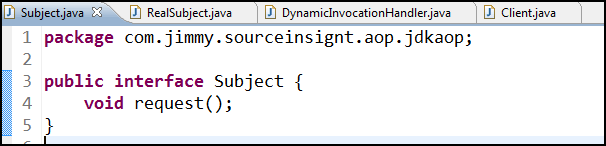


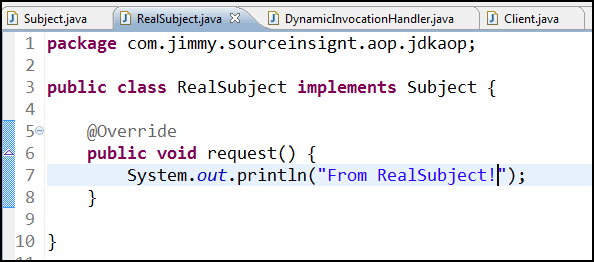


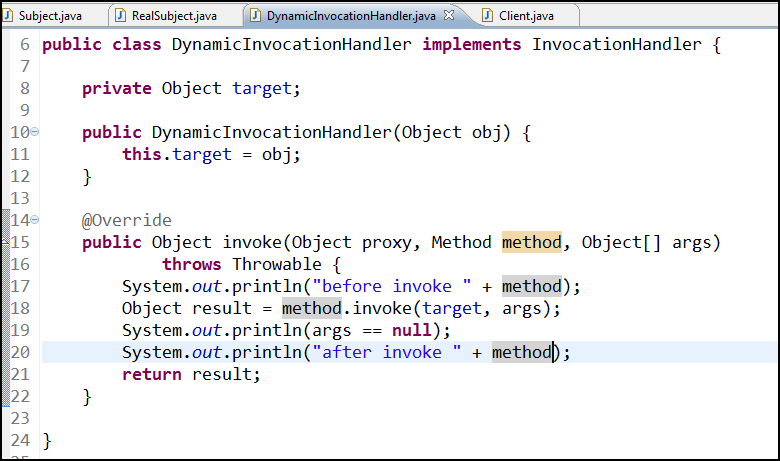


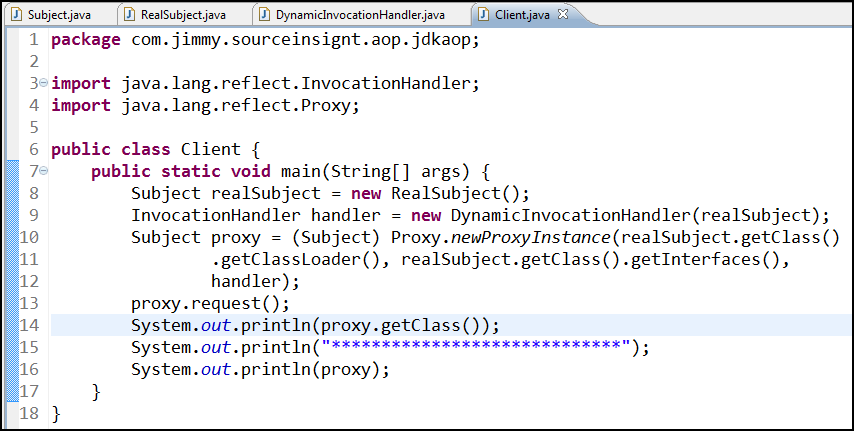
JDK的动态代理只能代理接口不能代理类，在Spring中，对AOP的实现有两种，一种是使用JDK代理的方式用来实现代理接口，而对于类的代理则使用CGLIB的方式来实现

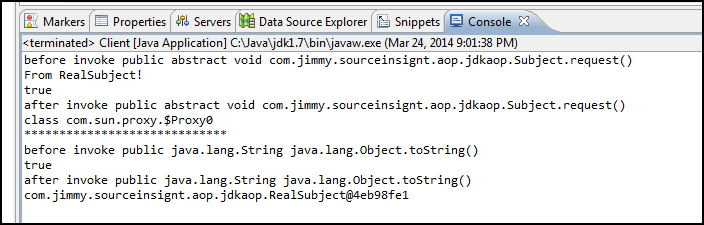
JDK动态代理的例子



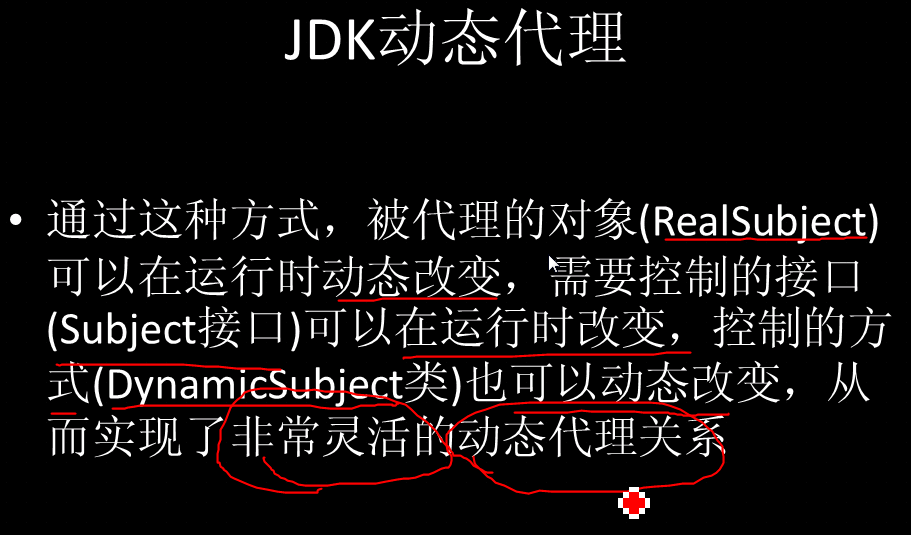


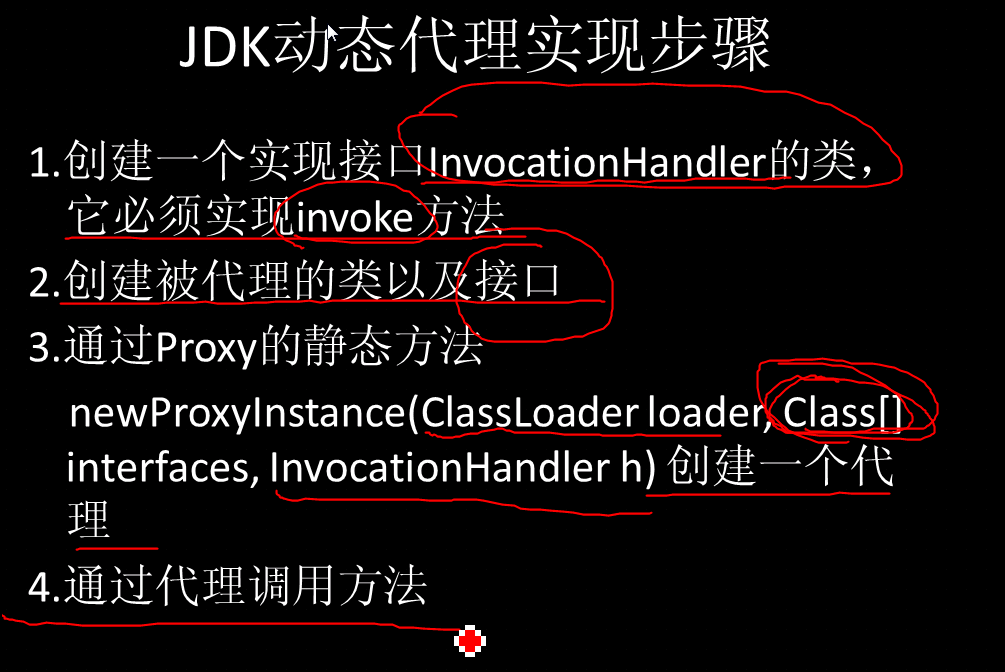




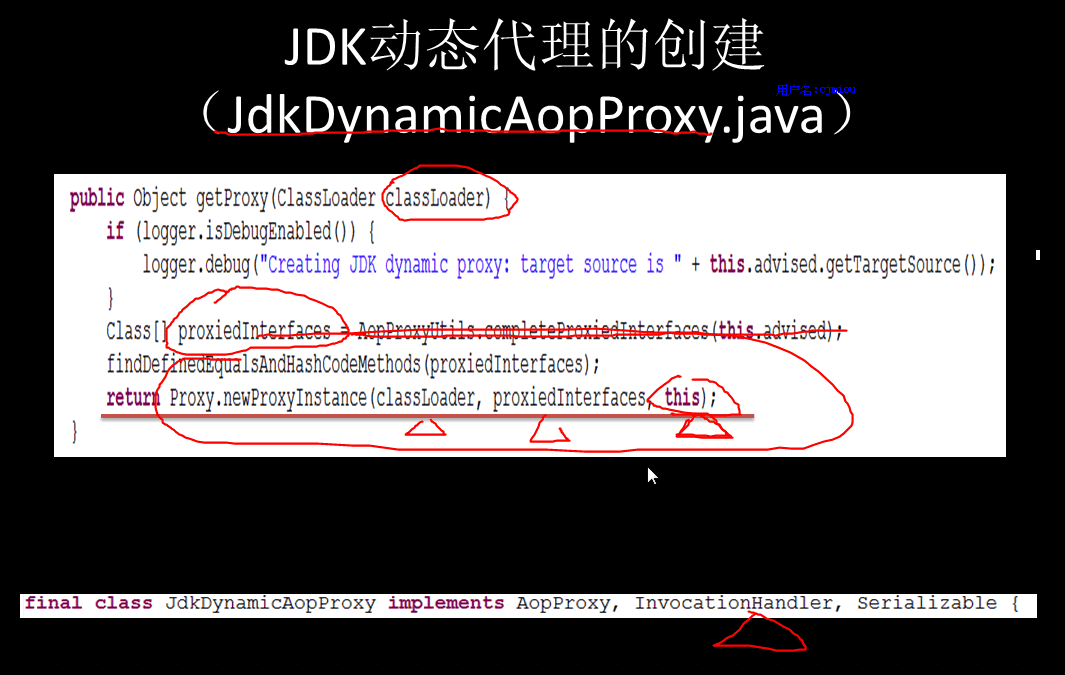


红框中的输出为toString的输出，暂时不明白为何。

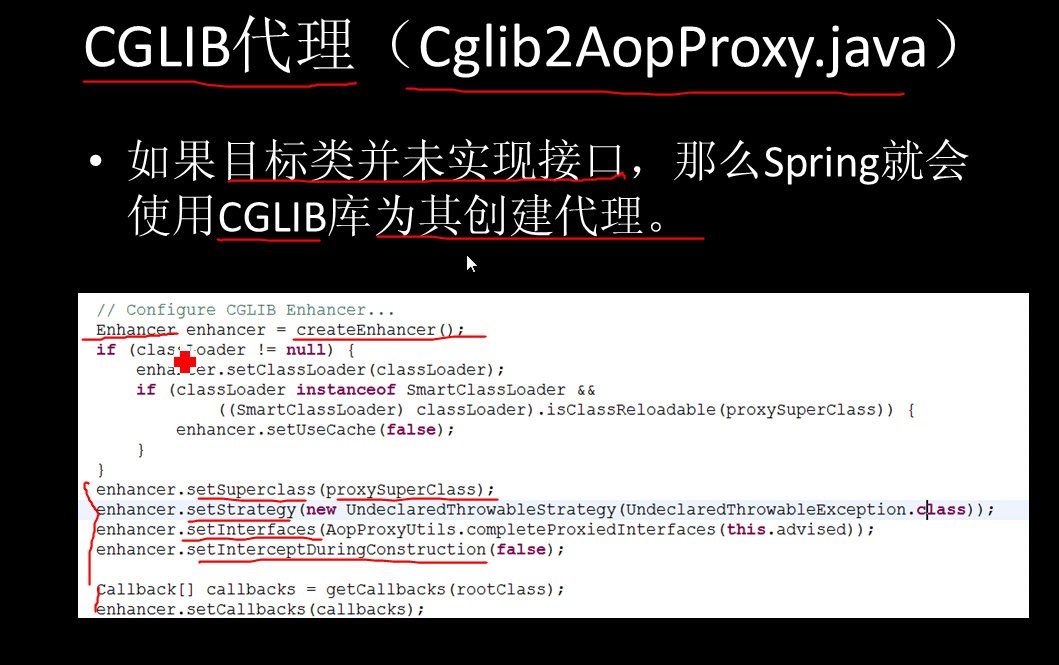


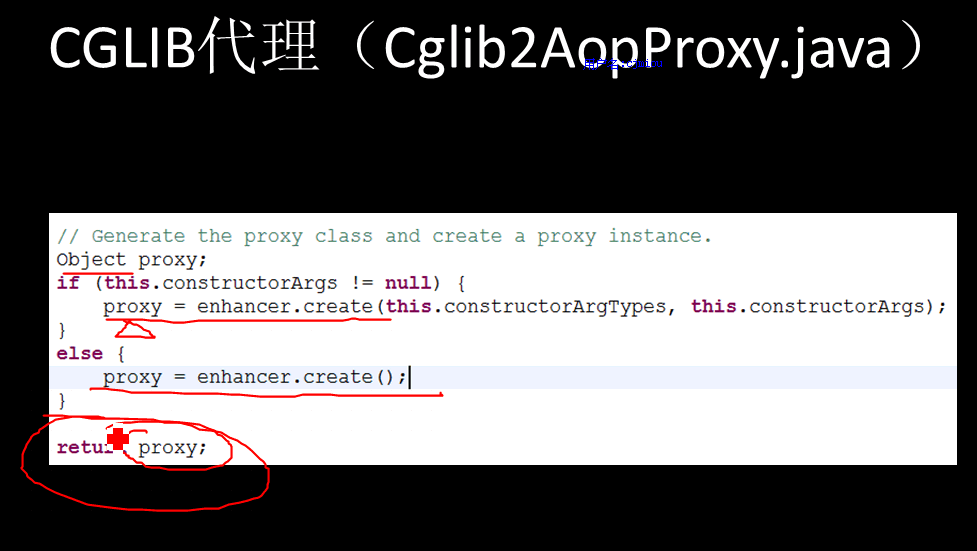


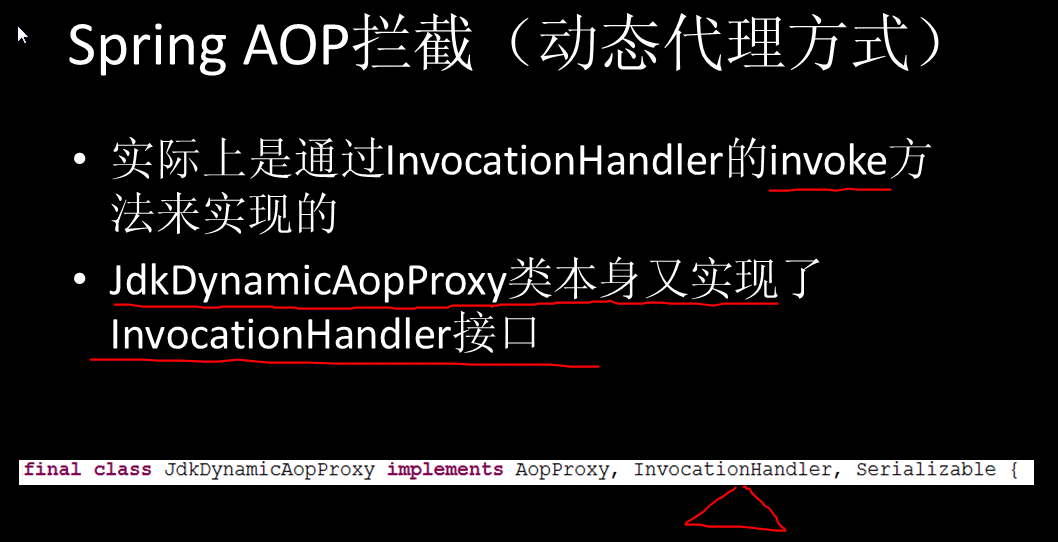
Spring中JDK代理的实现方法：

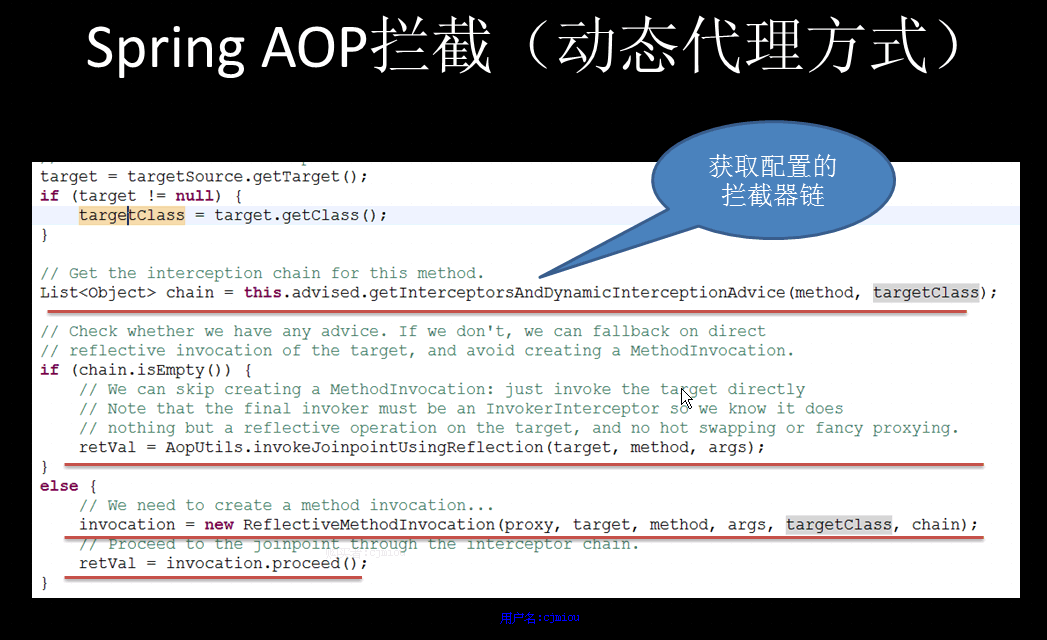


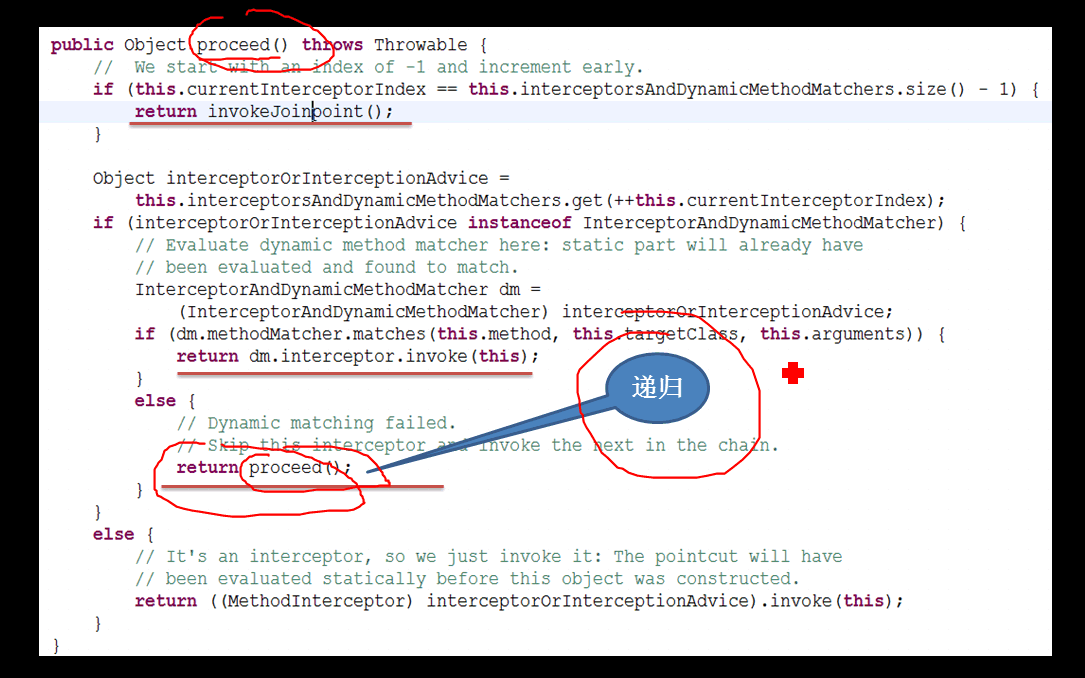
Spring中的CGLIB代理实现方式：

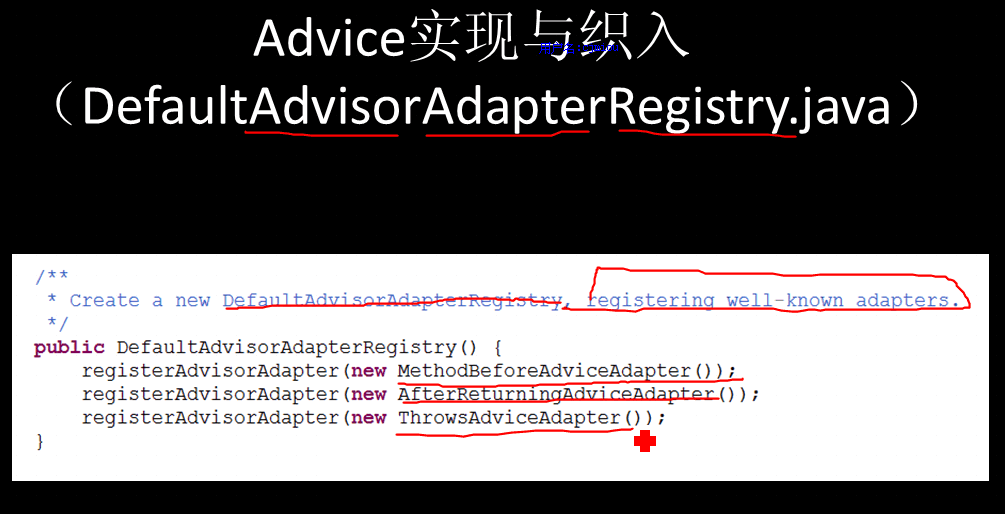


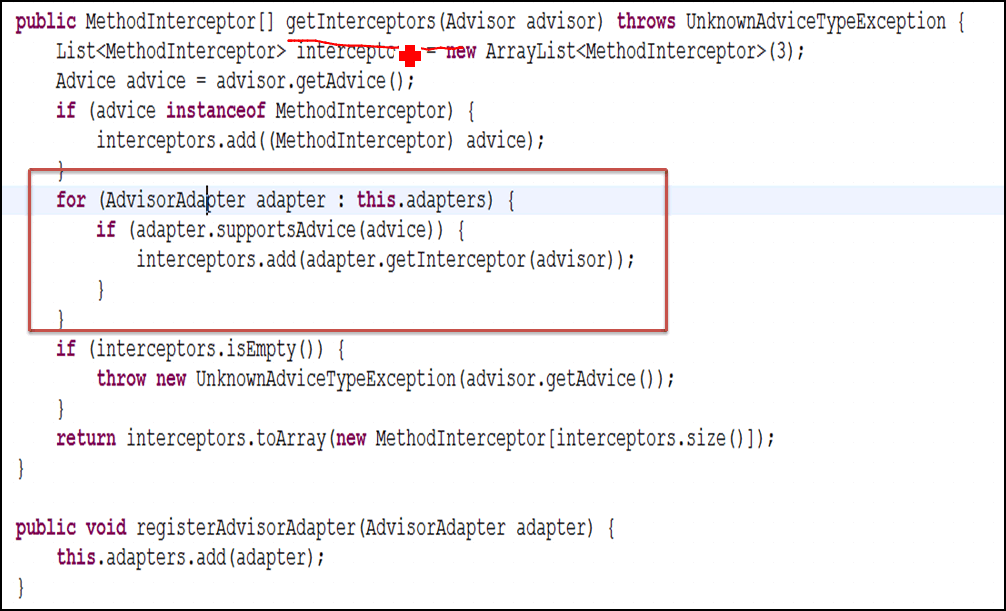


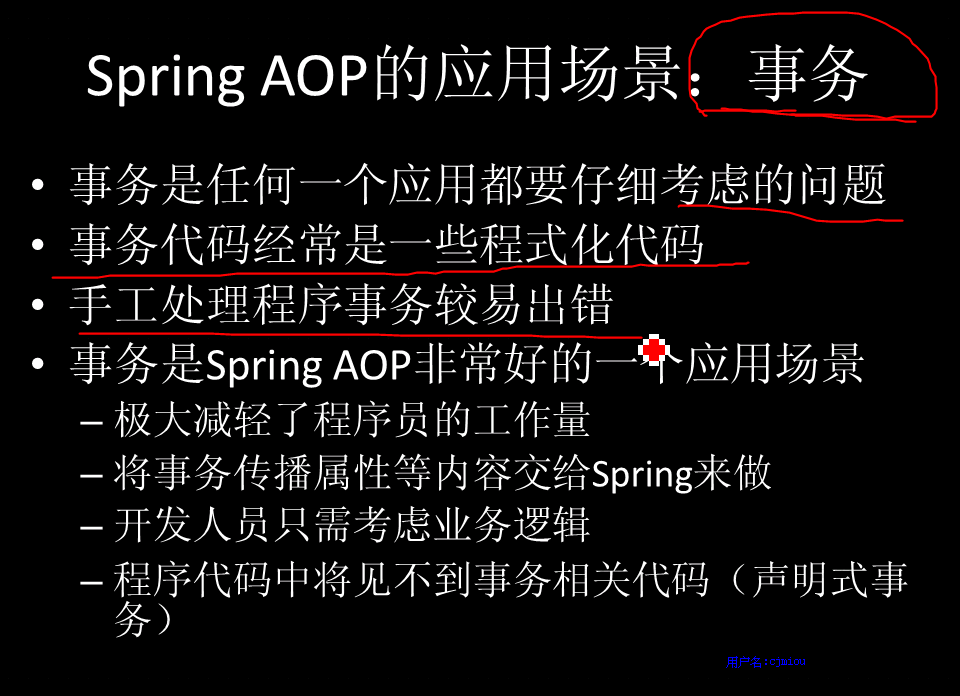


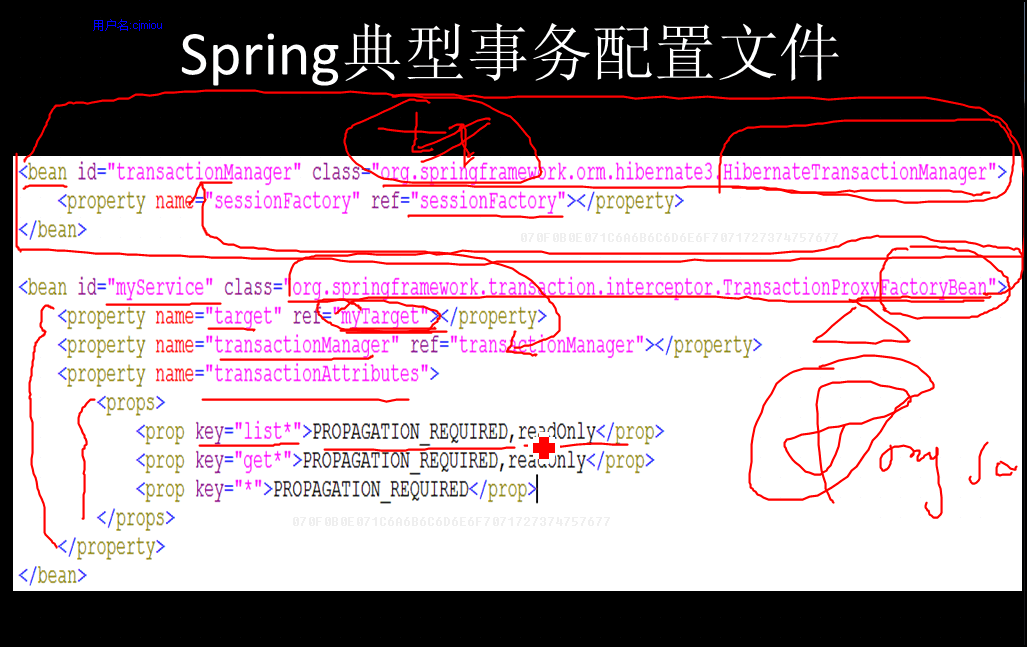






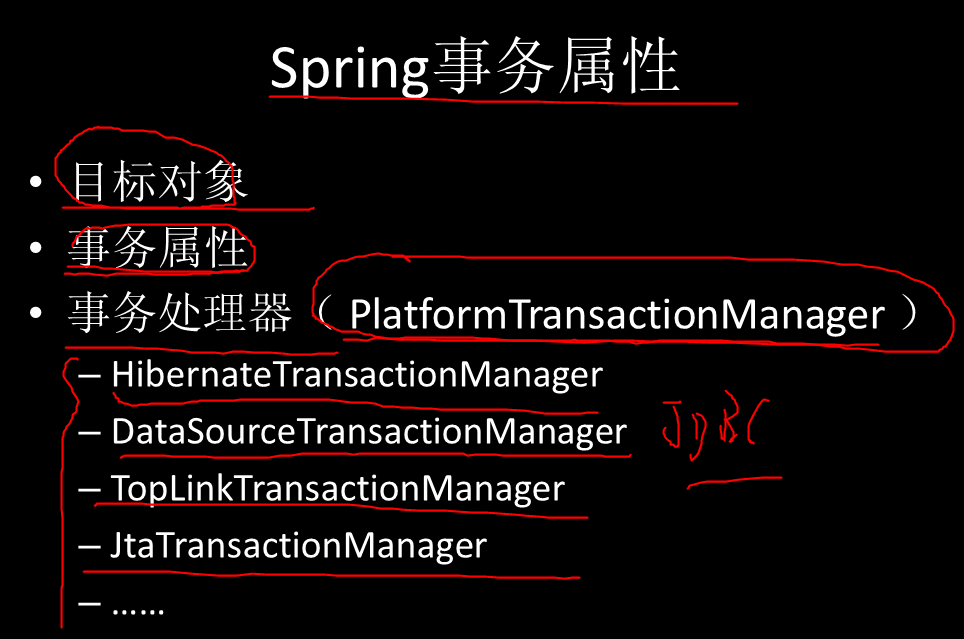




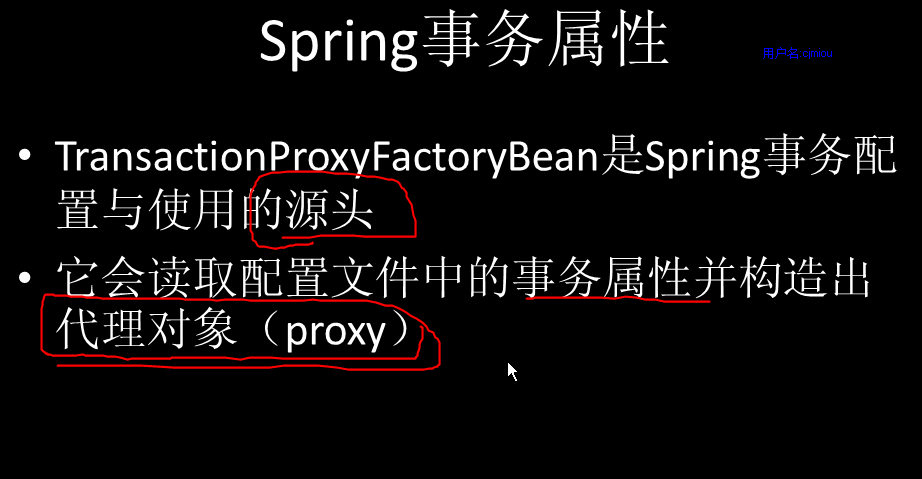


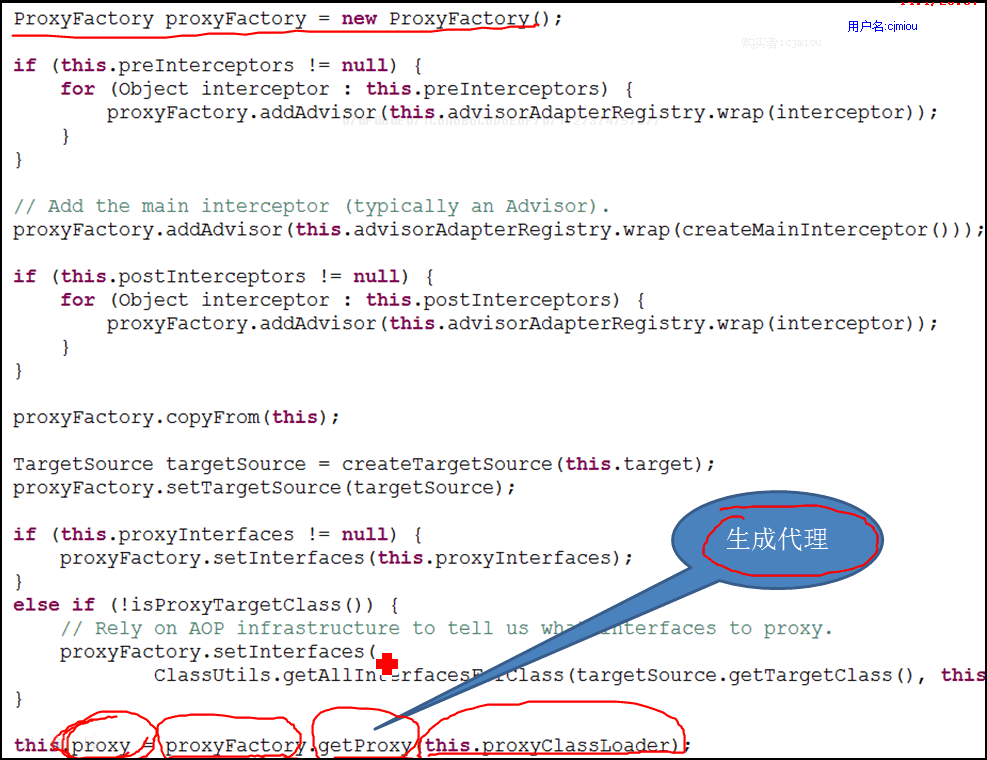
1. 定义一个由spring提供的transaction manager的bean实例
2. 定义一个由spring提供的transaction proxy factory bean实例

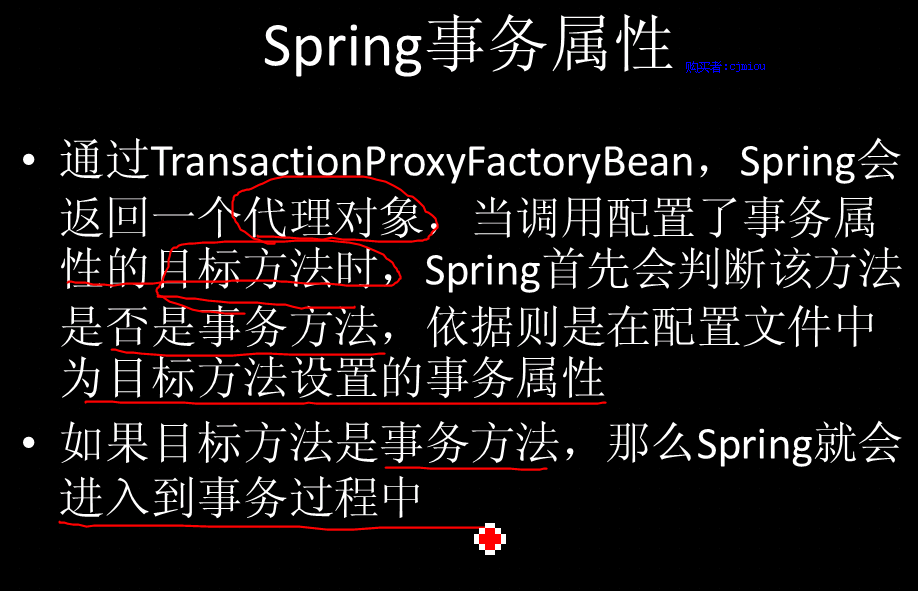
指定它的target，transaction manager和transaction attributes属性

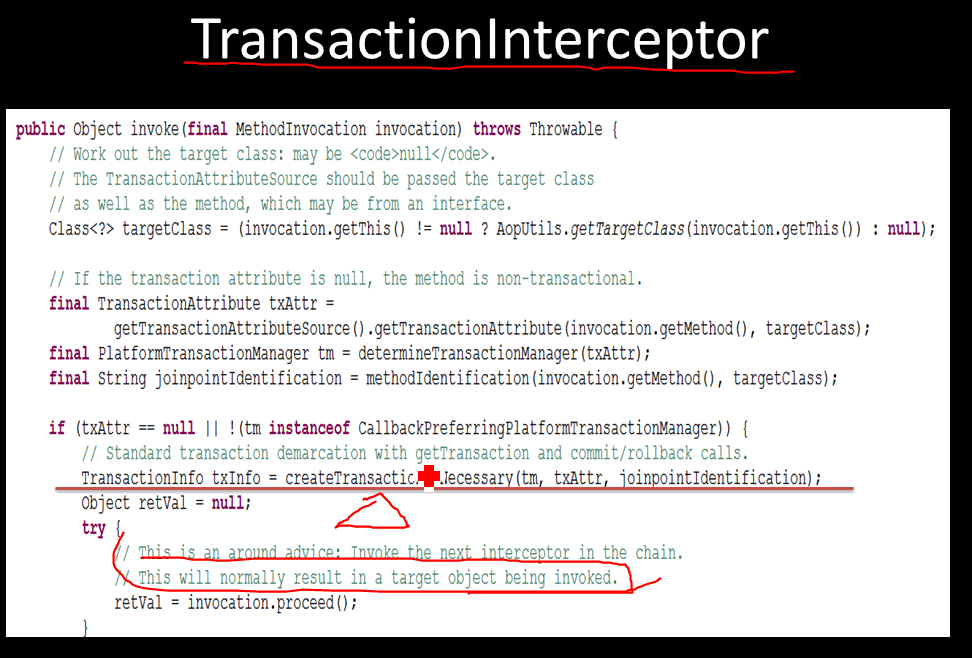


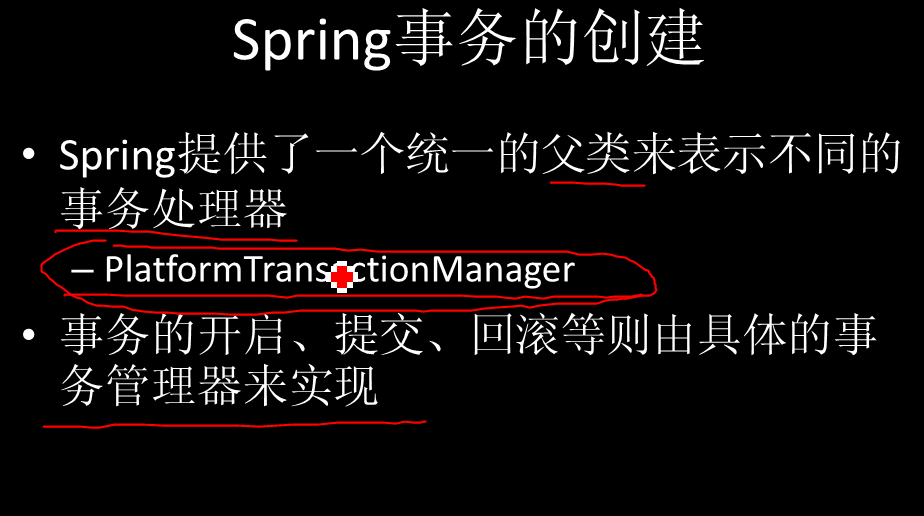
事务处理器都是继承自PlatformTransactionManager

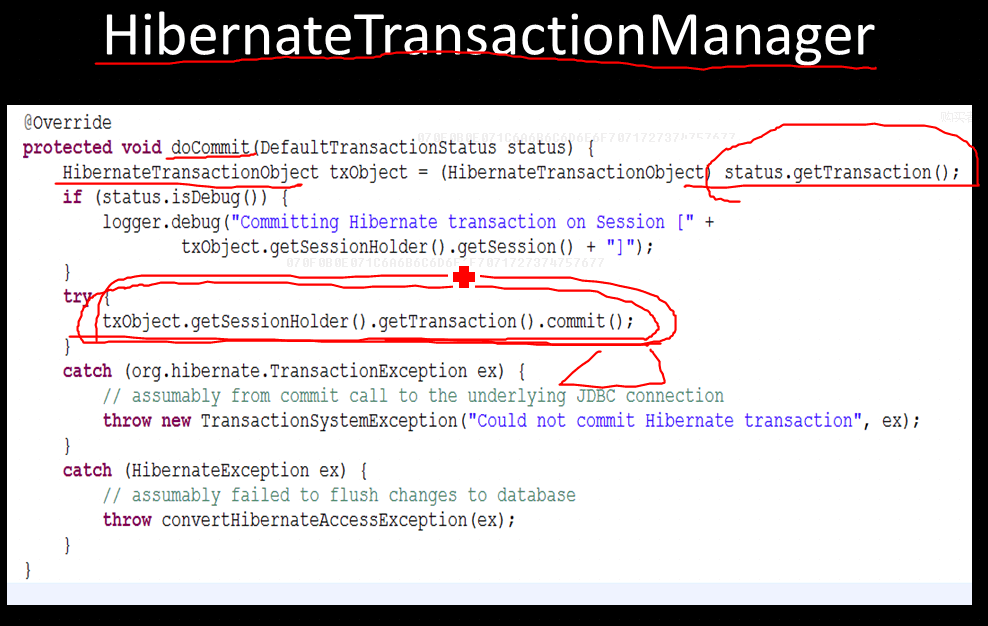


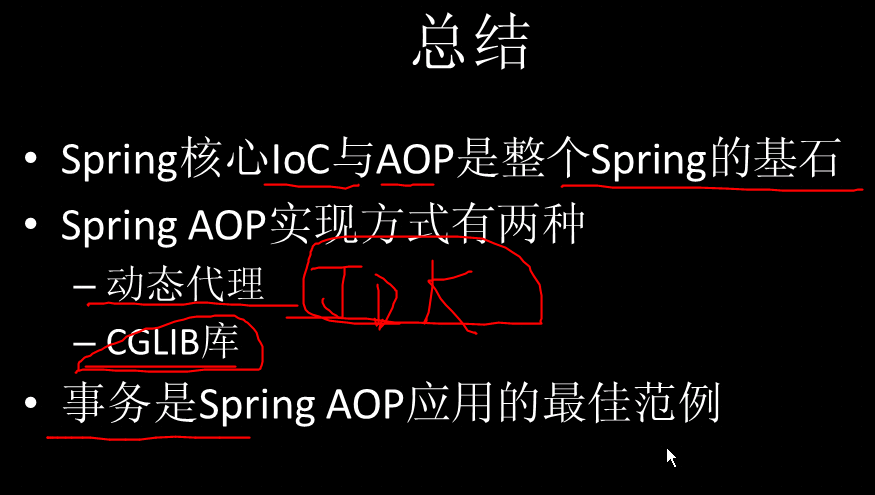










****