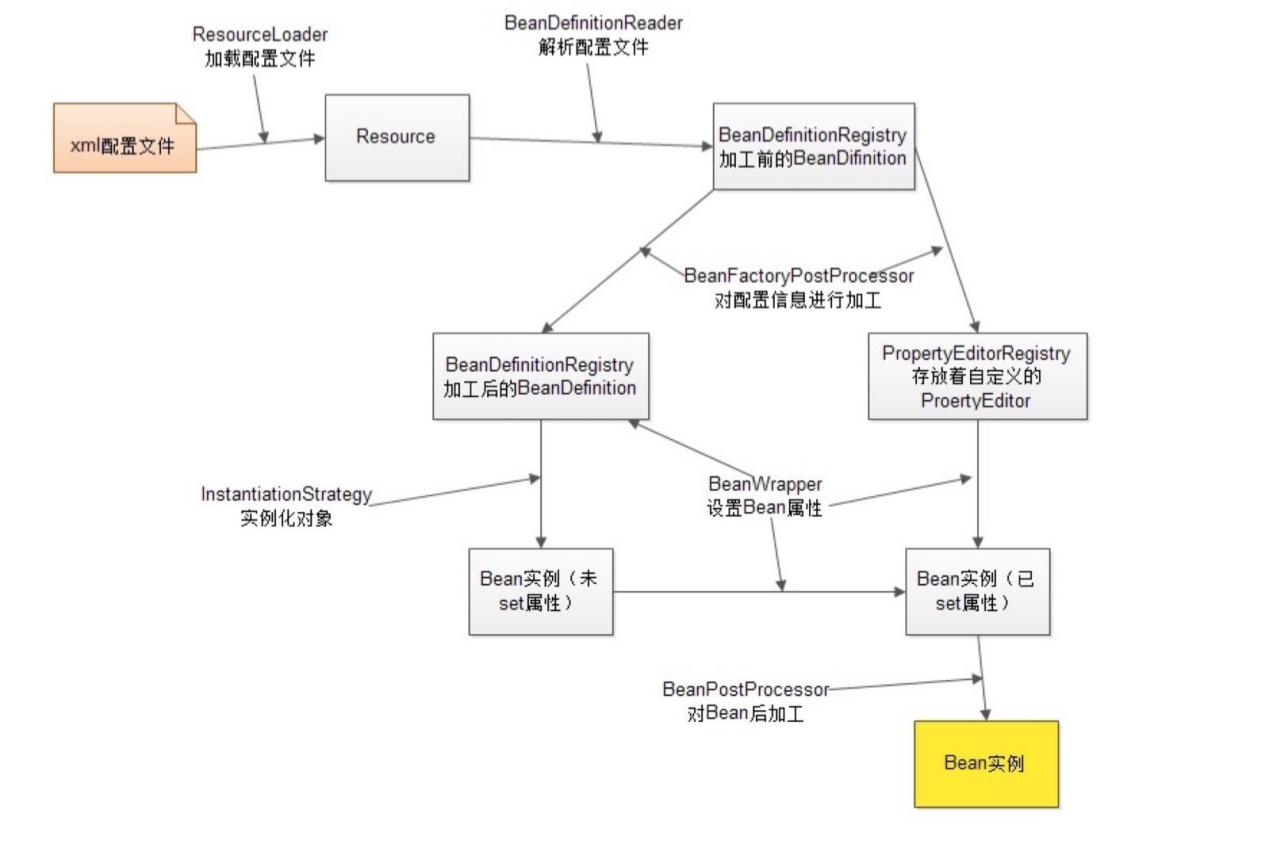
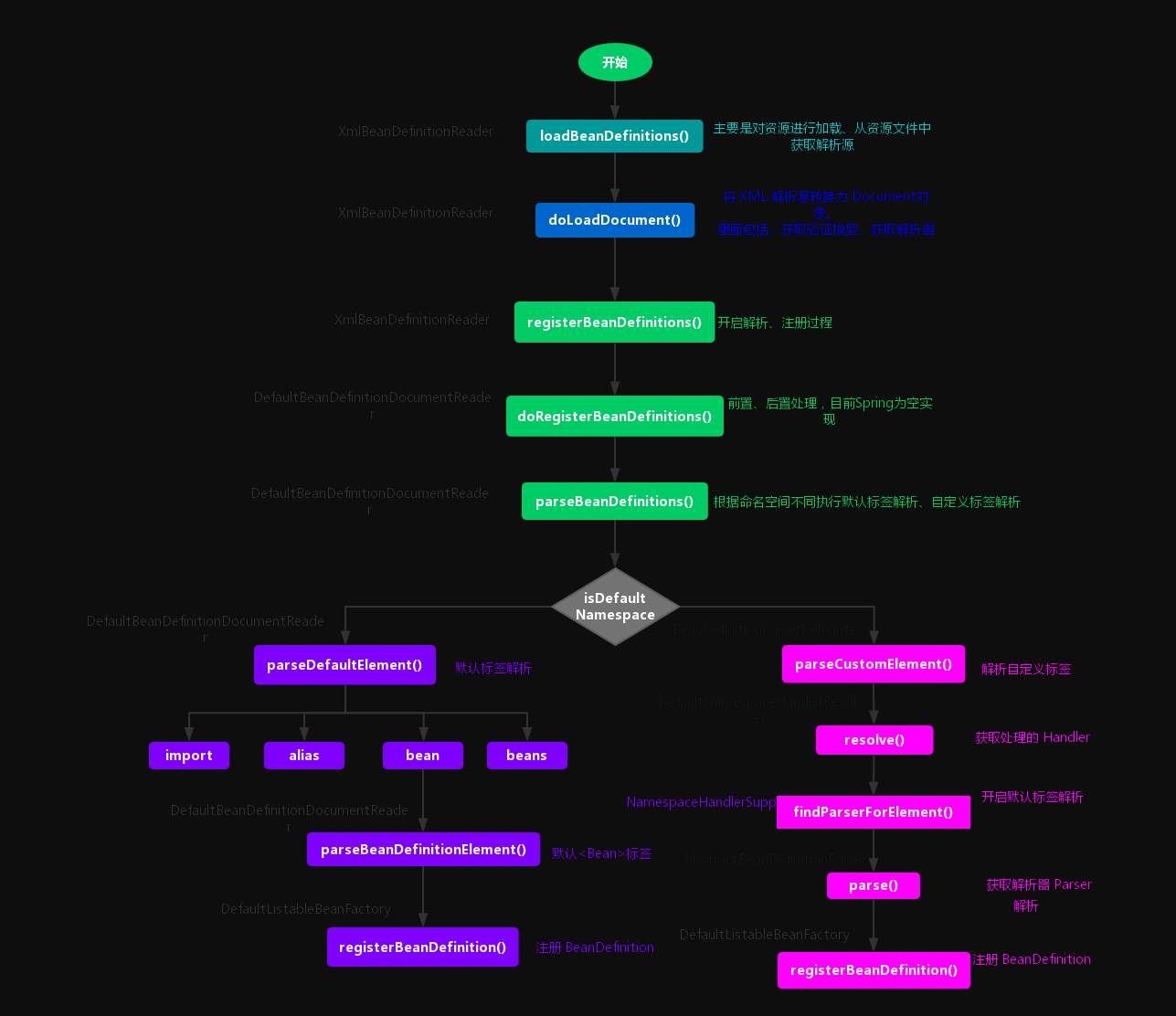
### SpringIOC加载过程图解



### SpringIOC转载Bean到BeanDefinition过程



### SpringIOC实例化bean过程(getBean(XXX.class))

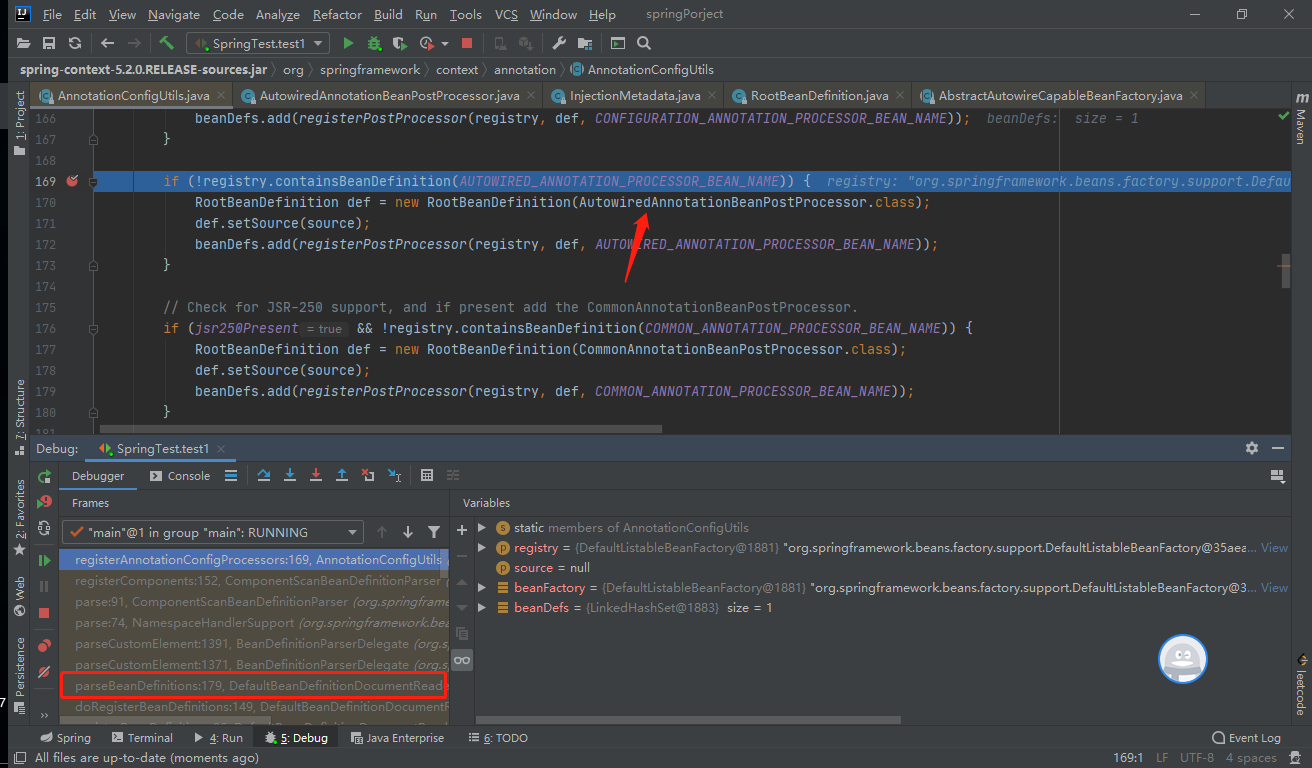


### Spring中@Autowried自动装配是如何实现的

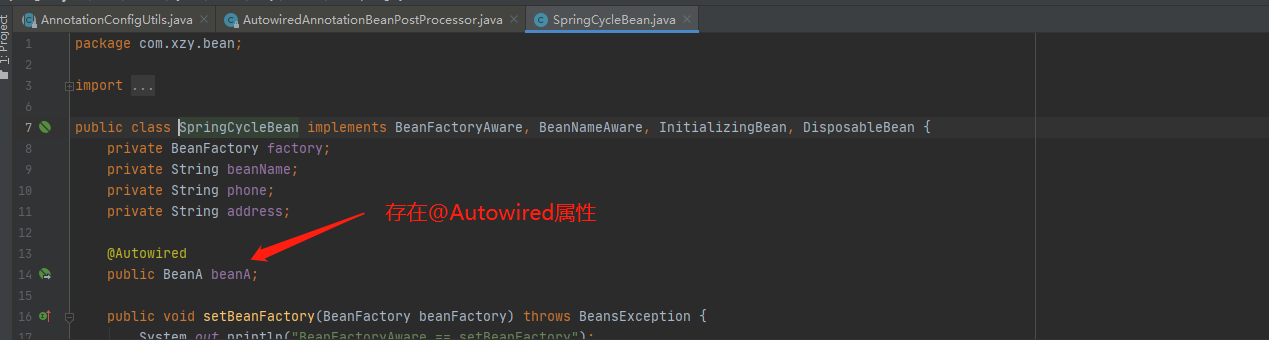
<https://zhuanlan.zhihu.com/p/28775709>

首先，@Autowried是通过后处理器实现的

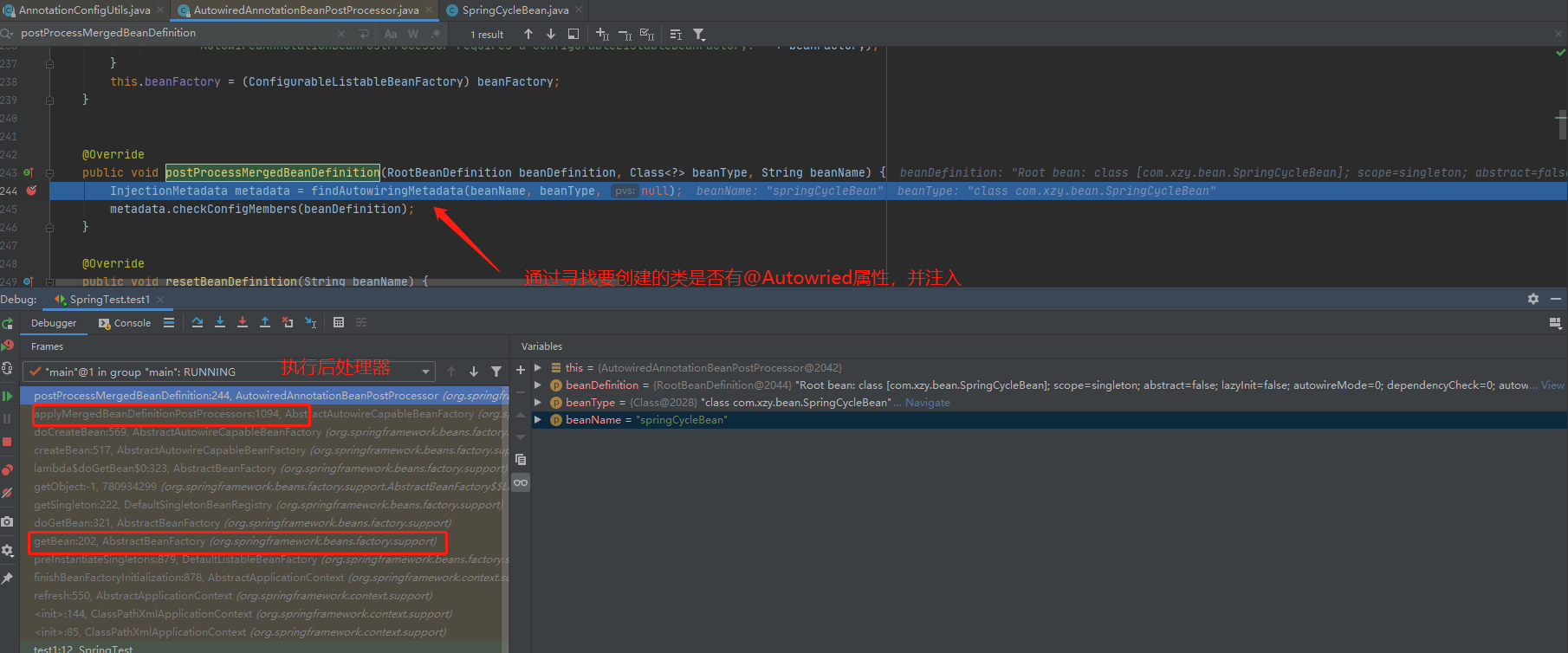
当我们在配置文件里面添加注释<context:component-scan/>时。Spring通过parseBeanDefinitions解析标签，会自动注入AutowiredAnnotationBeanPostProcessor后处理器。

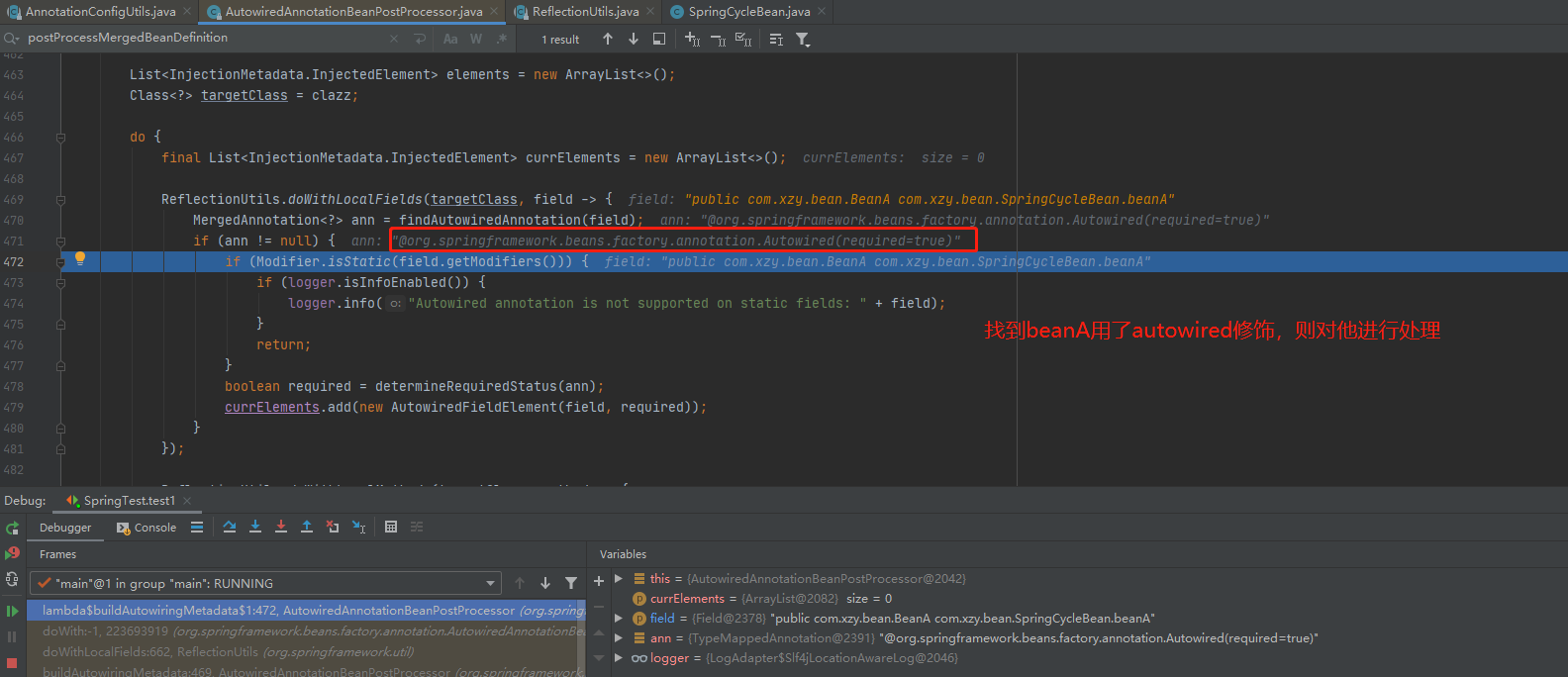


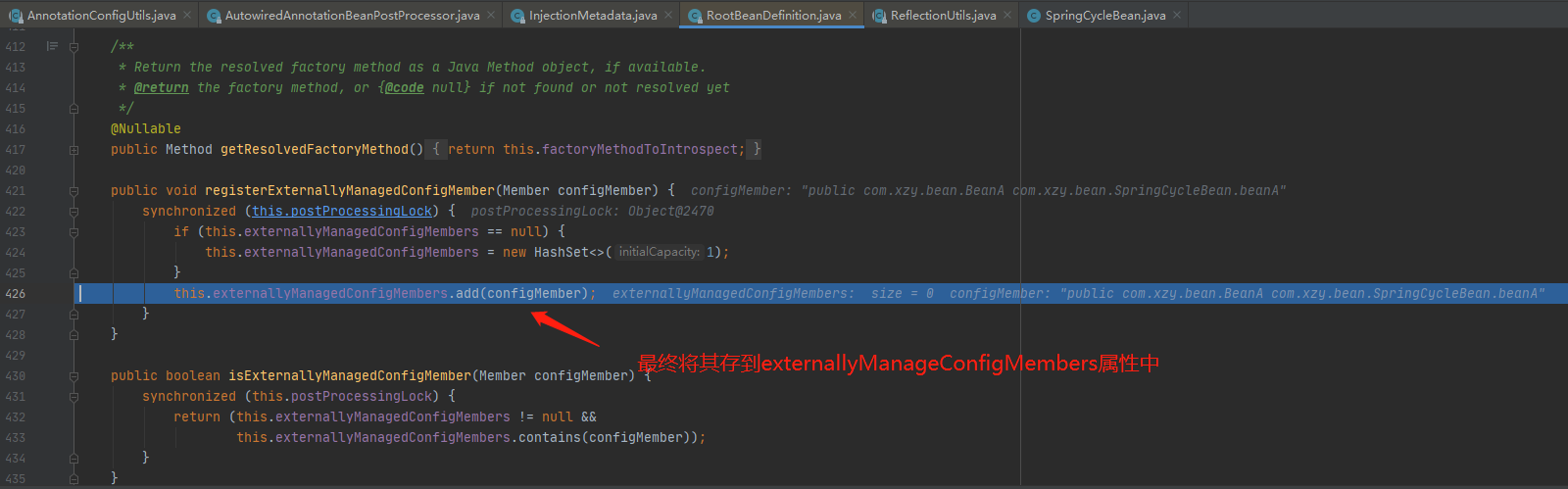
假如一个对象有@Autowried属性，则在getBean时，会通过AutowiredAnnotationBeanPostProcessor来解析@Autowried



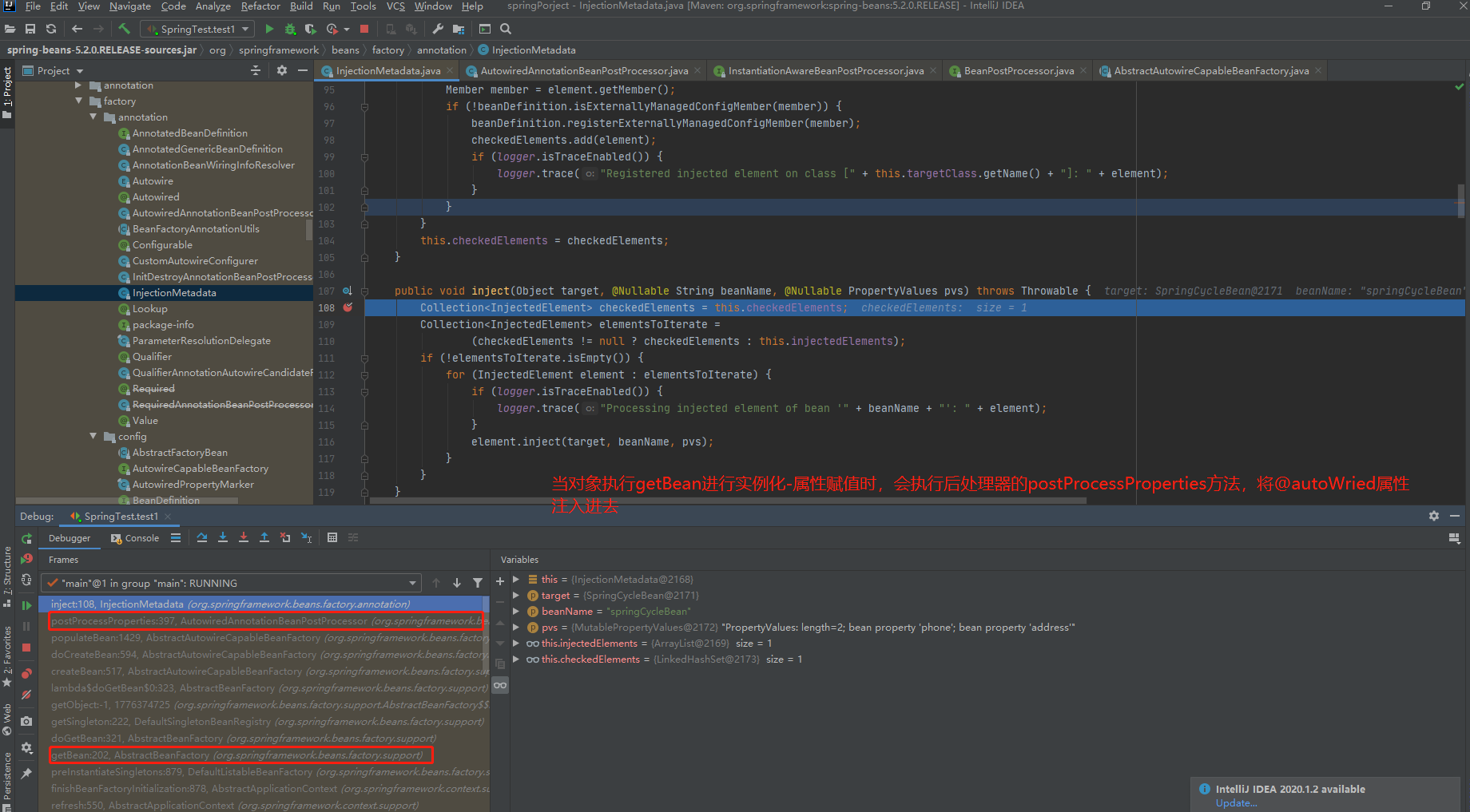
下图，在实例化之后，设置属性之前，由于实现了MergedBeanDefinitionPostProcessor，则会调用postProcessMergedBeanDefinition[这个方法在实例化后属性赋值之前调用]，找到对应的Autowired







当创建的对象要设置属性时，触发了AutowiredAnnotationBeanPostProcessor实现的后处理赋值方法postProcessPropertyValues(此方法是在InstantiationAwareBeanPostProcessorAdapter类中，需要子类重写)

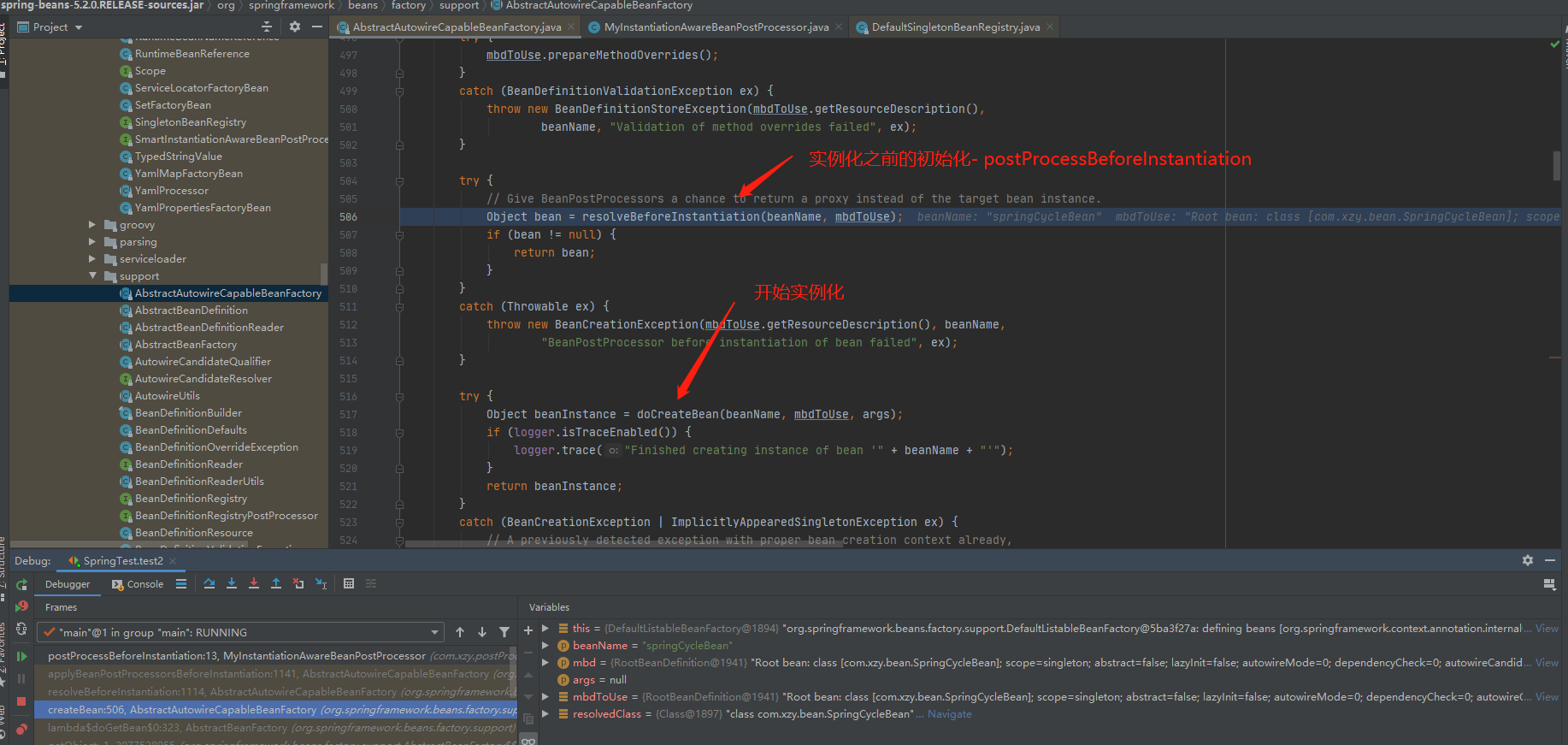


### Spring各种后处理器执行的代码

#### postProcessBeforeInstantiation实例化之前调用

org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory

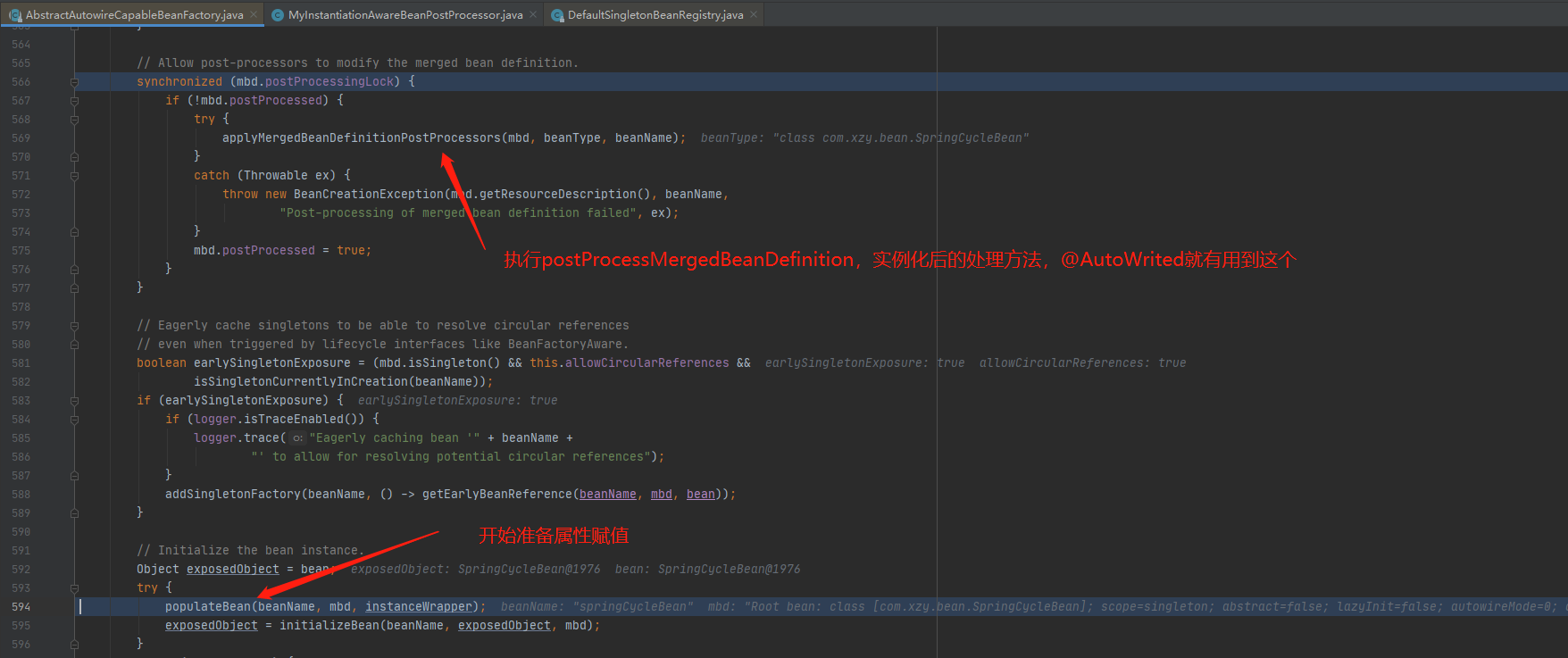
AOP的原理就是基于这个处理器方法，在他之前做了一层代理



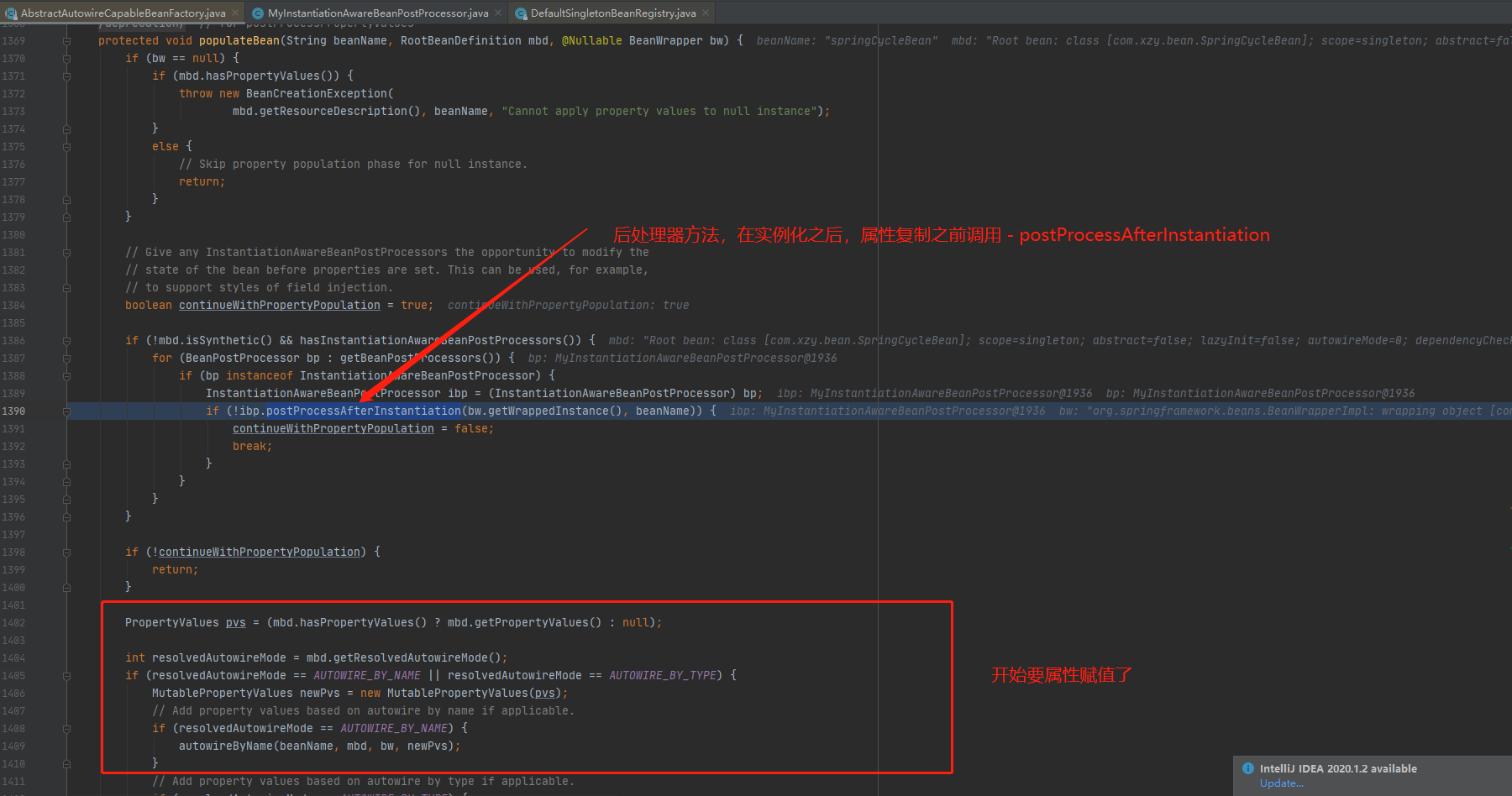
#### postProcessMergedBeanDefinition

org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory

postProcessMergedBeanDefinition和postProcessAfterInstantiation比对，两个都是在实例化之后，属性赋值之前调用，且前者早于后者，目前不清楚之间的区别



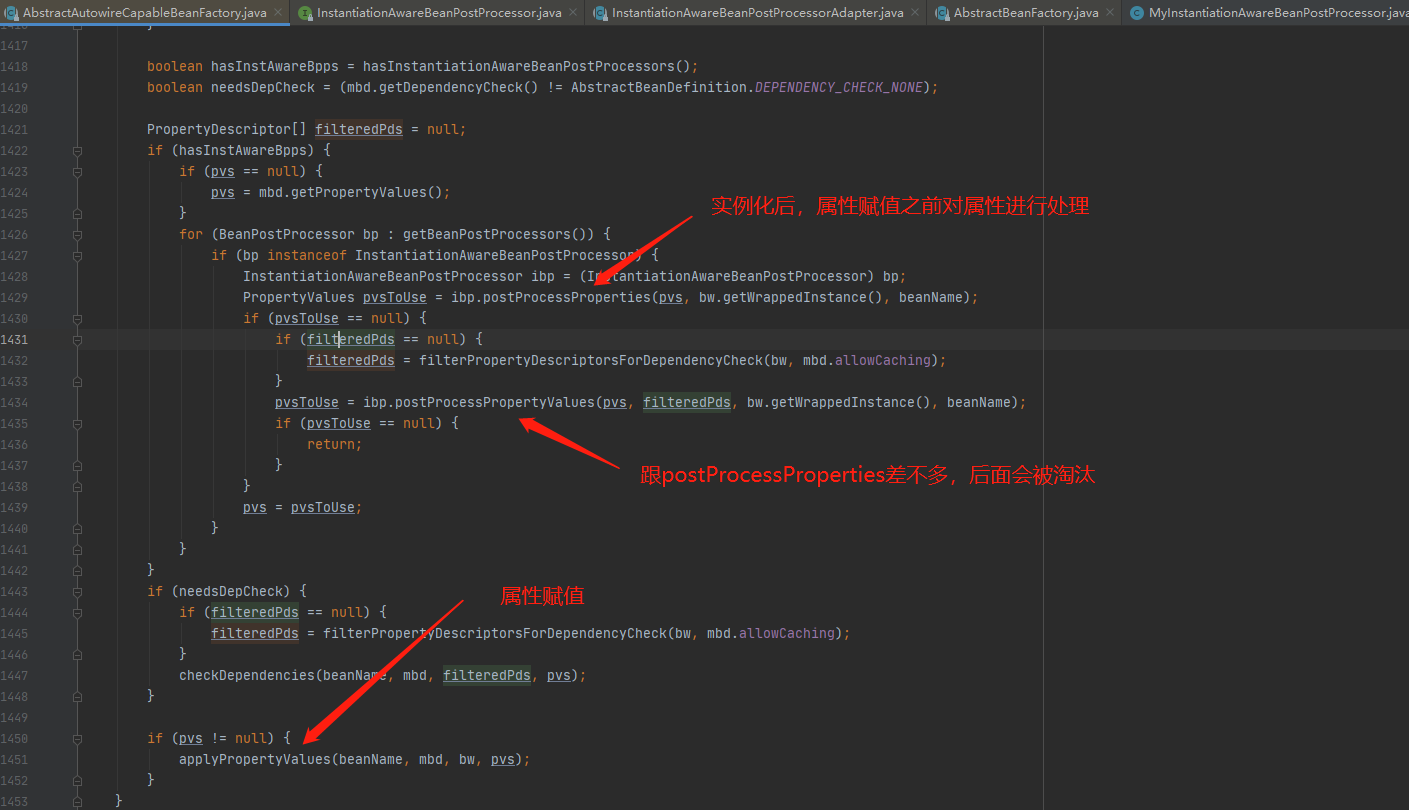
#### postProcessAfterInstantiation

org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory

#### postProcessProperties

org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory

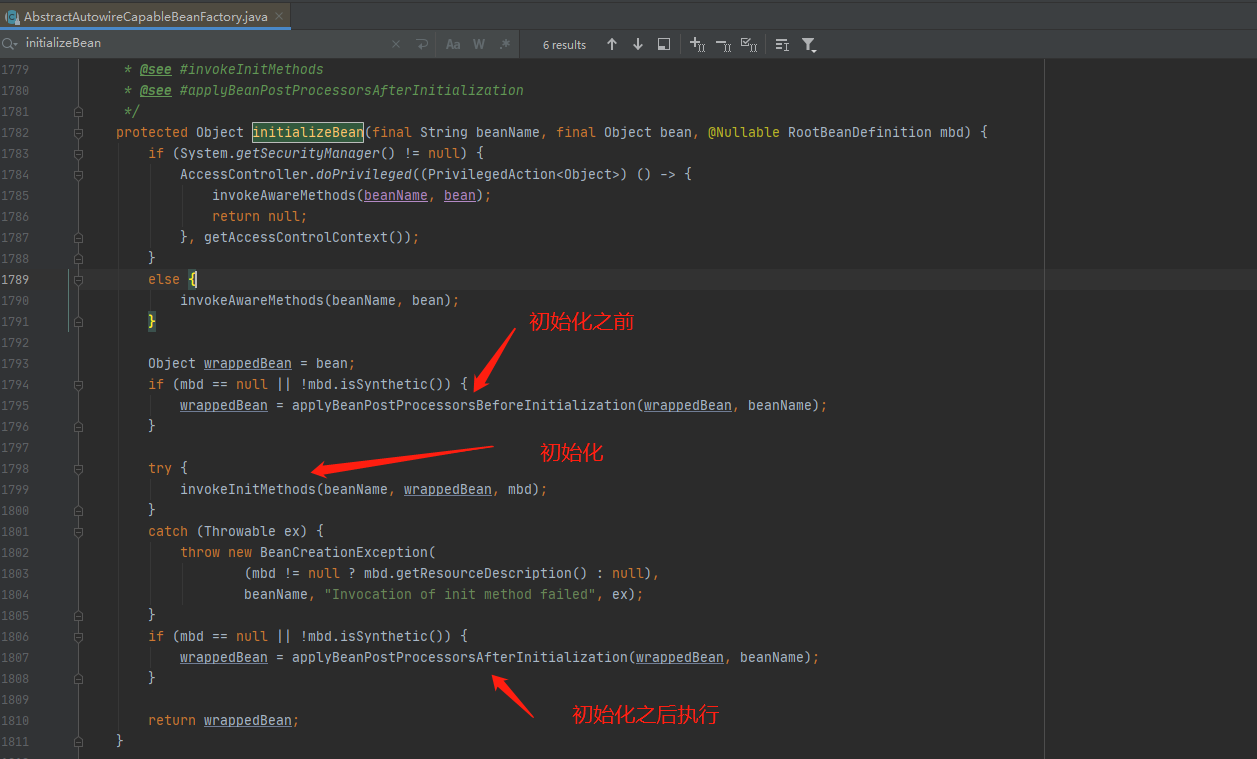
用于在属性赋值之前对属性做一些改动



#### postProcessBeforeInitialization

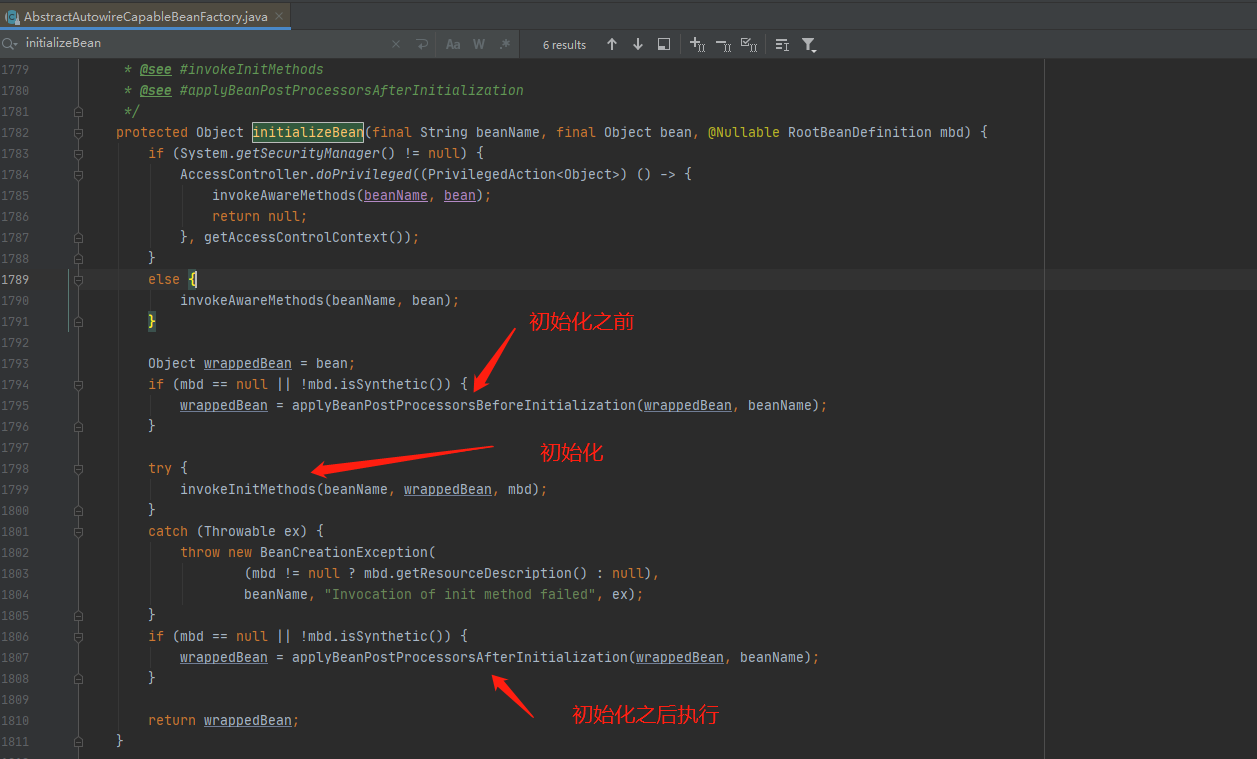
org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory

在初始化之前执行



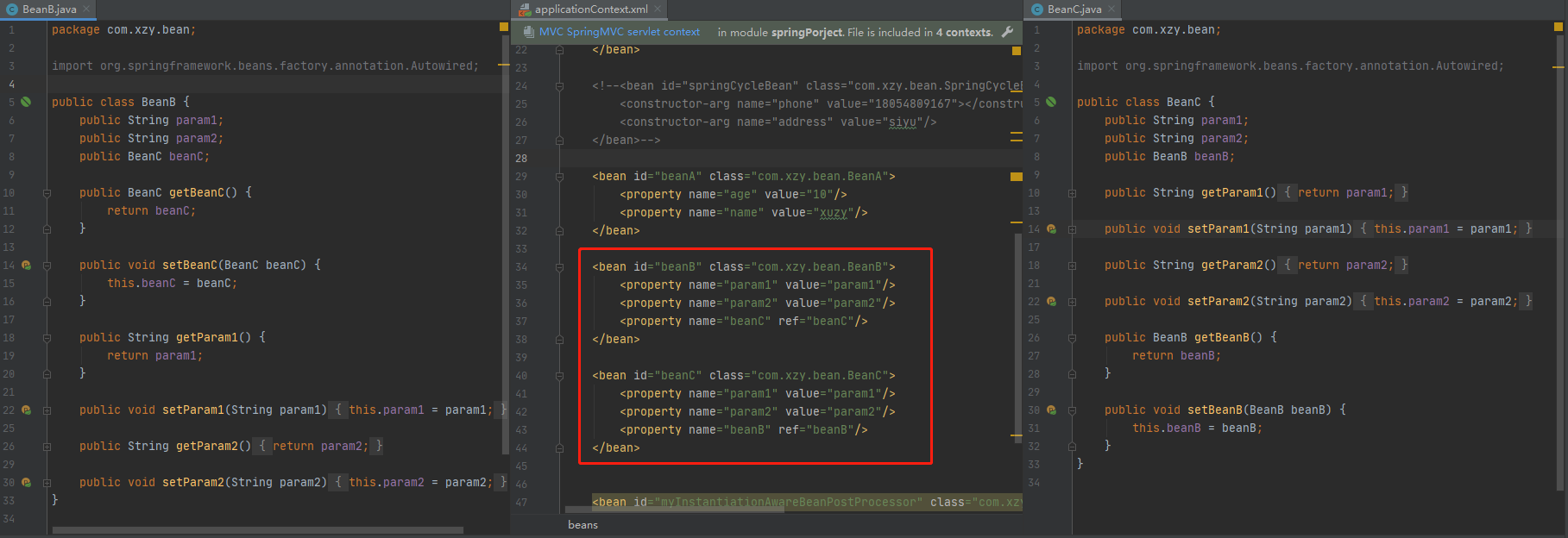
#### postProcessAfterInitialization

org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory

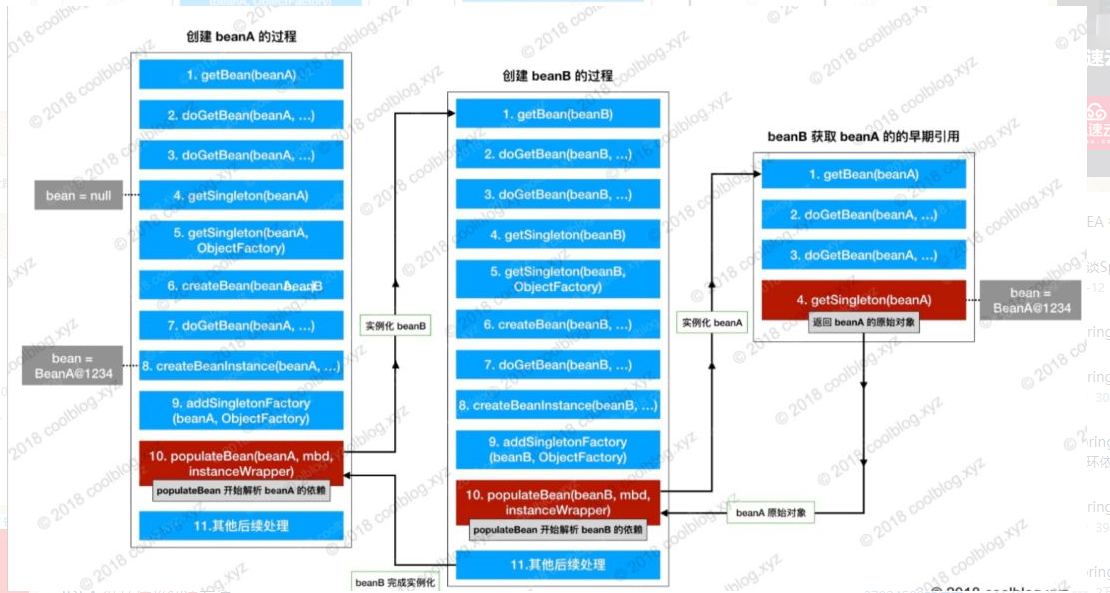


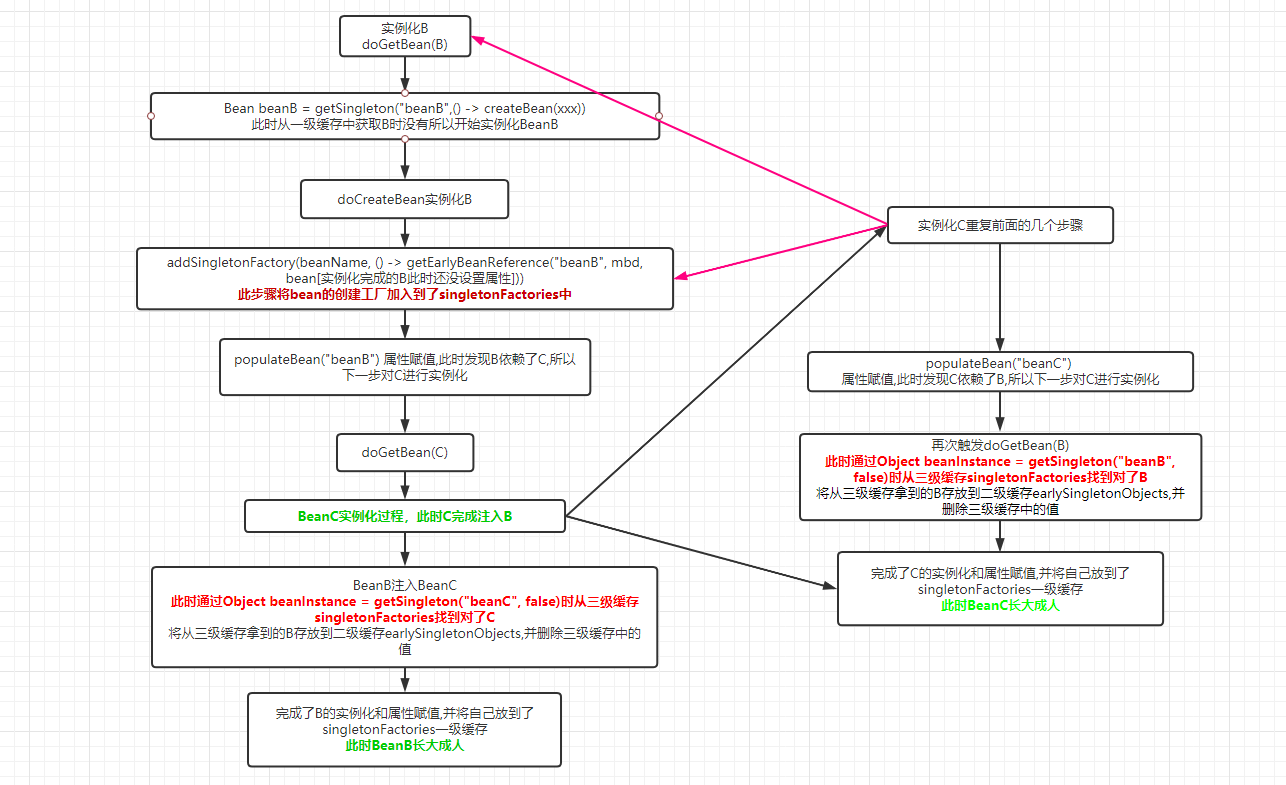
### Spring如何解决循环依赖

#### 循环依赖类设置



#### 相关流程图



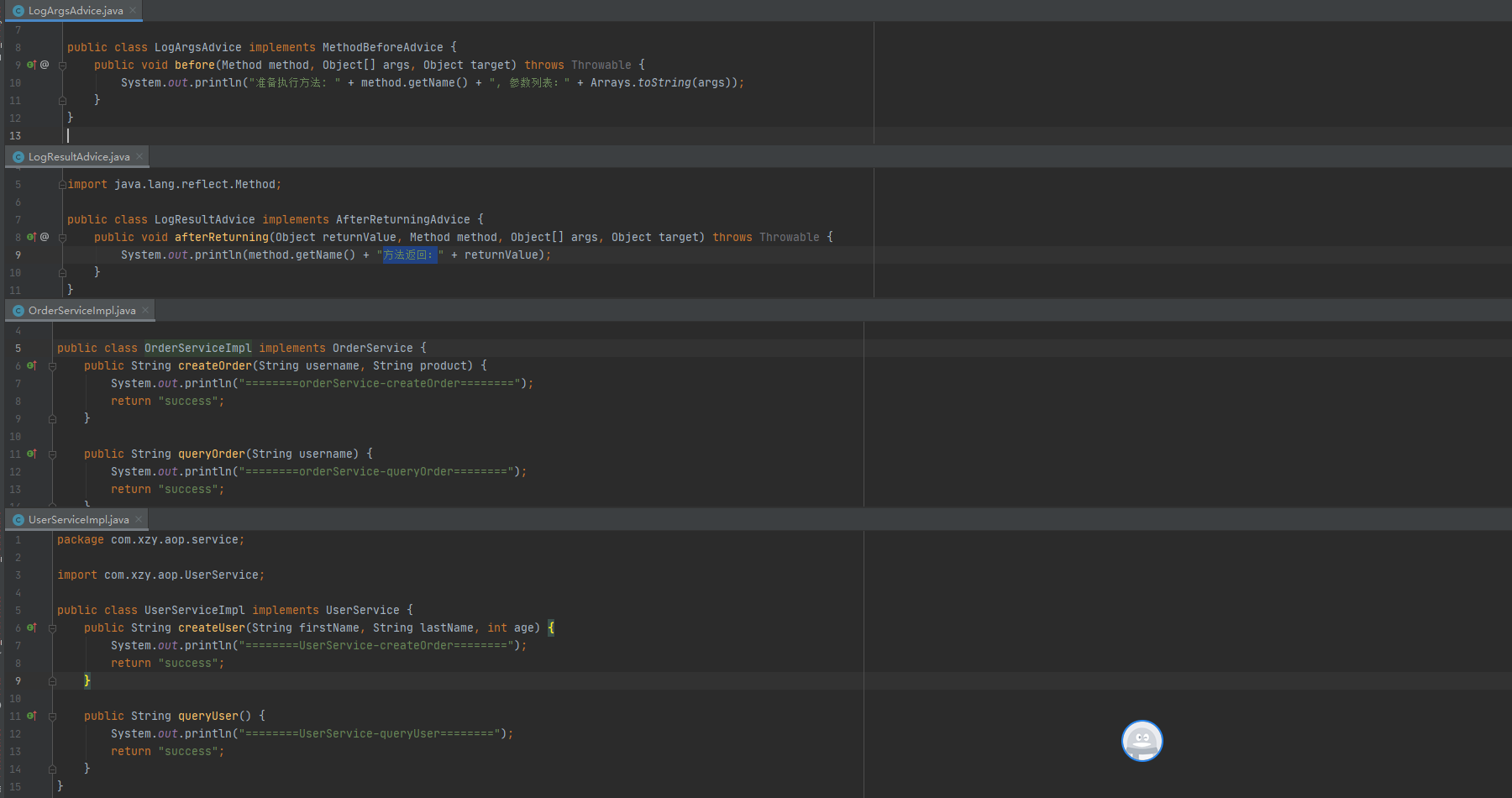


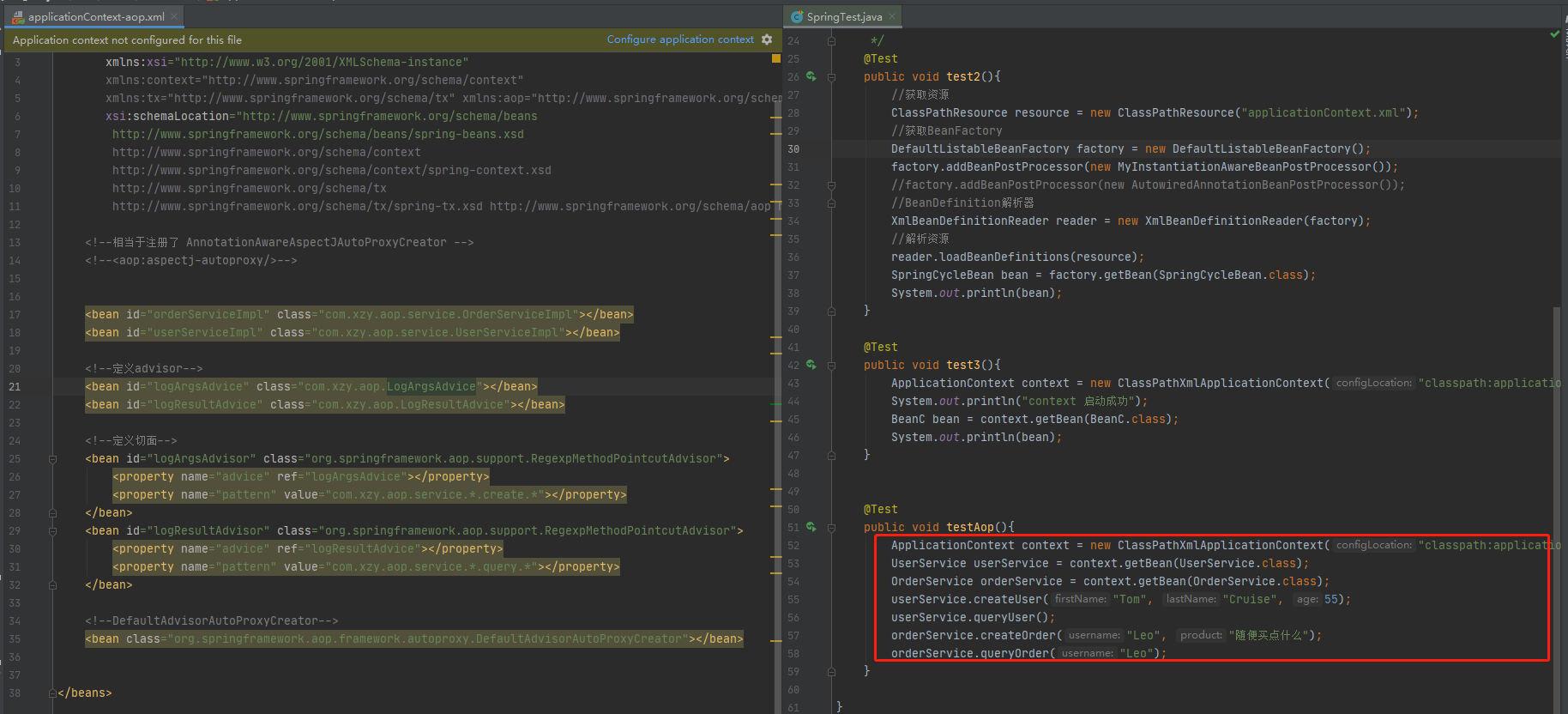
### SrpingAOP原理

#### SrpingAOP的原理是什么？

AOP(面向切面编程)实现的目标就是可以在方法执行前后插入自己的逻辑。他的实现原理它基于动态代理来实现。默认地，如果使用接口的，用 JDK 提供的动态代理实现，如果没有接口，使用 CGLIB 实现。

#### 具体测试代码类





#### 具体原理解析

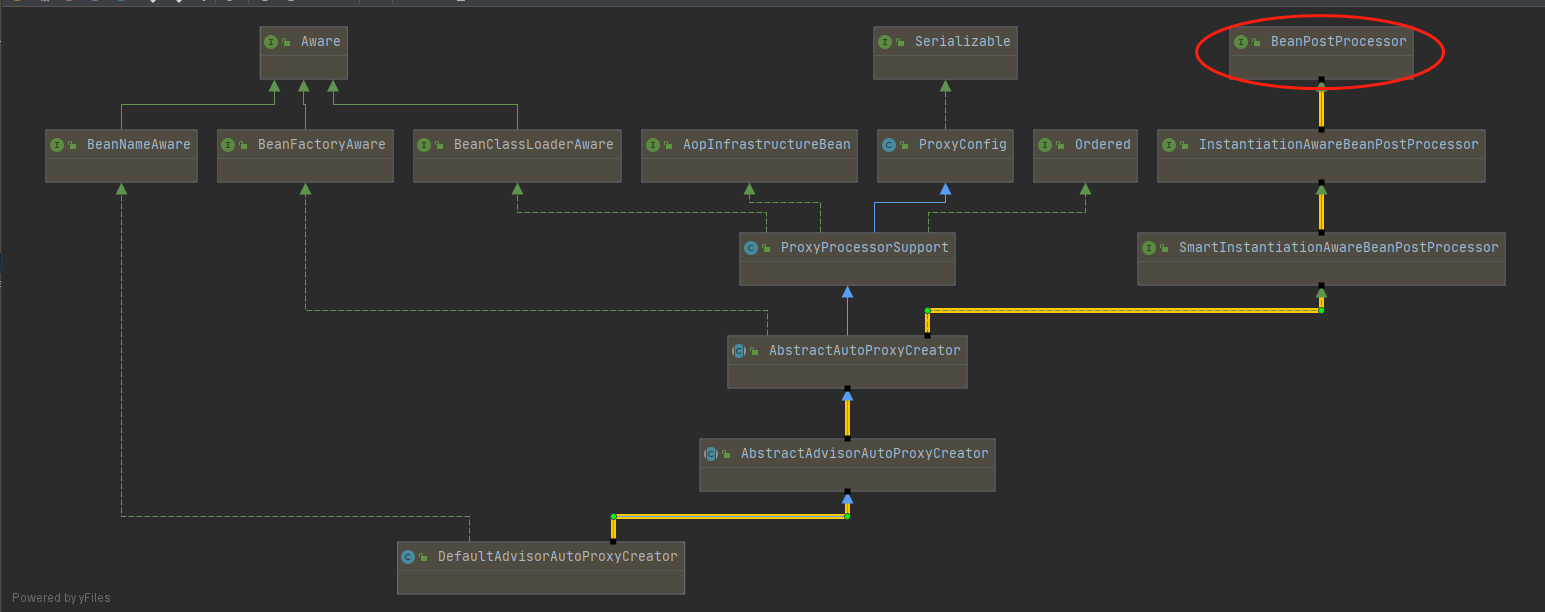
我们知道在执行createUser方法前后被植入了方法是用了代理的方式，如果没有AOP，在Spring获取生成orderServiceImpl实例时只是普通的实例，那到底SpringAOP是如何将orderServiceImpl生成的普通对象转成代理对象呢？

原理在于配置文件上的

**<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"></bean>**

我们先看下这个类的继承关系

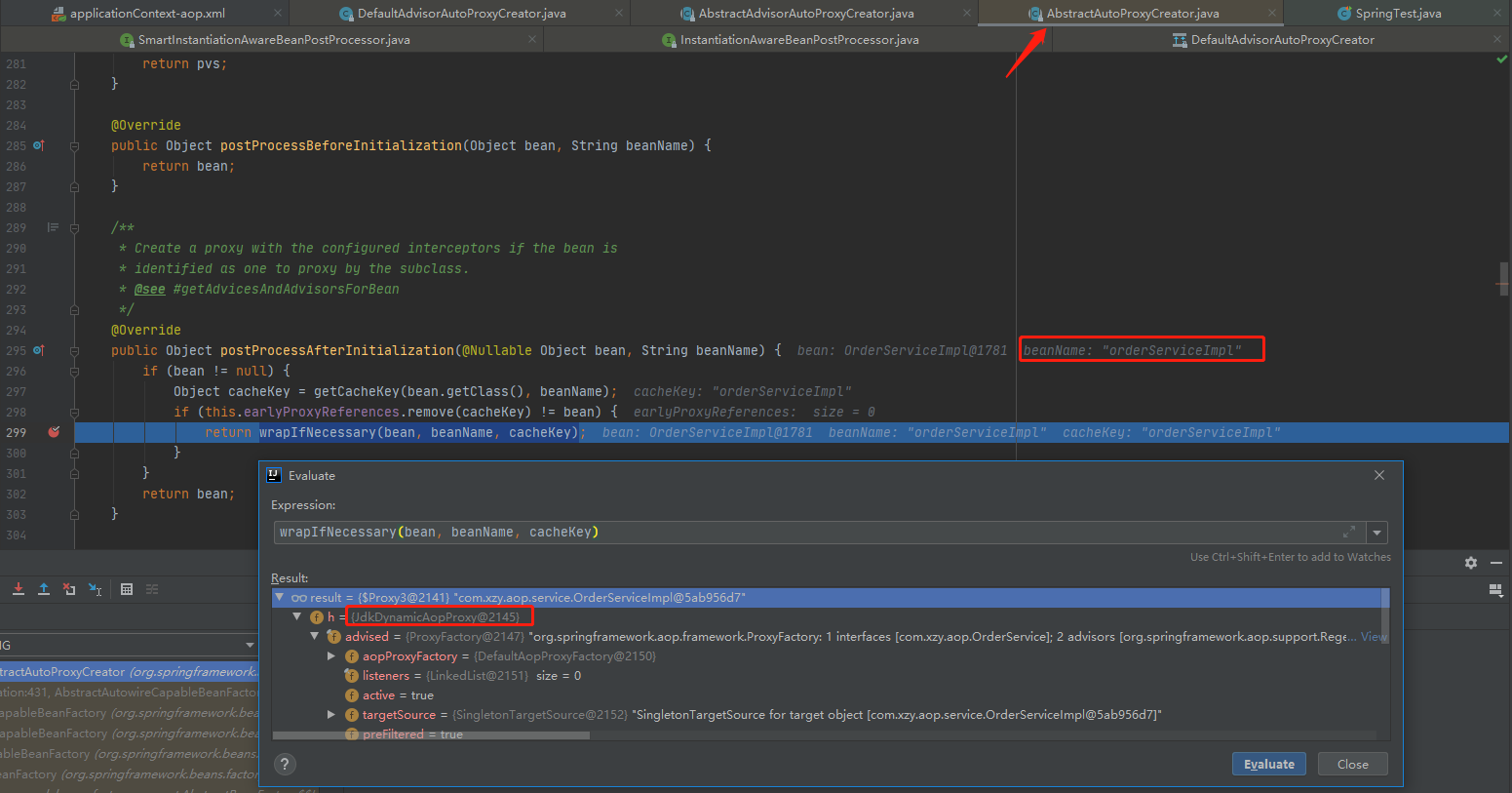
可以看到这个类实现了接口BeanPostProcessor后处理器，实现了这处理器后，就可以在每次bean初始化之前做一些操作，**所以可以猜想他是通过postProcessAfterInitialization方法在每个bean初始化之后将其用JDK代理包装了**。



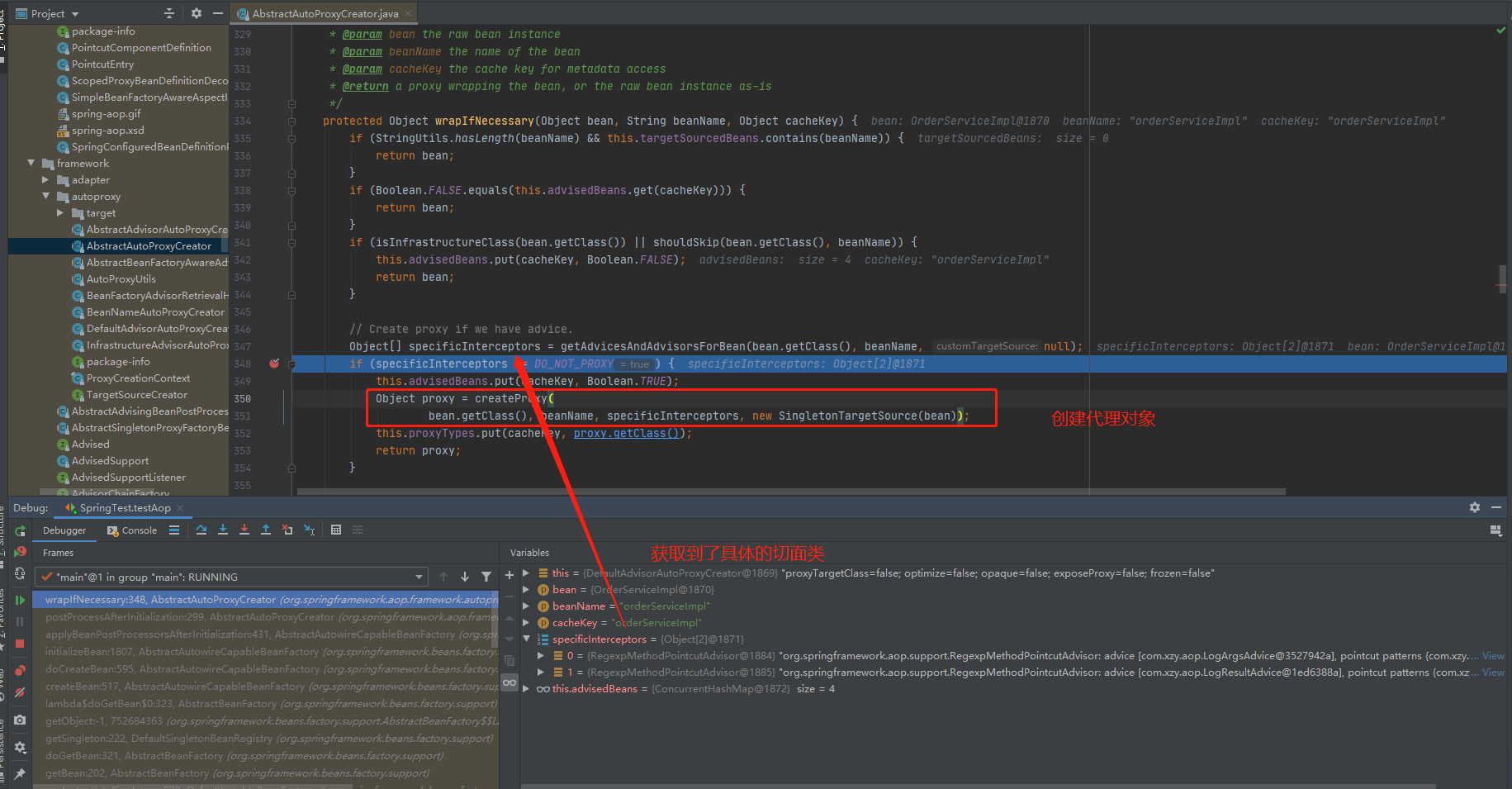
具体我们找到了，在其父类

org.springframework.aop.framework.autoproxy.AbstractAutoProxyCreator.java中，通过

postProcessAfterInitialization方法将初始化后的bean转成代理对象

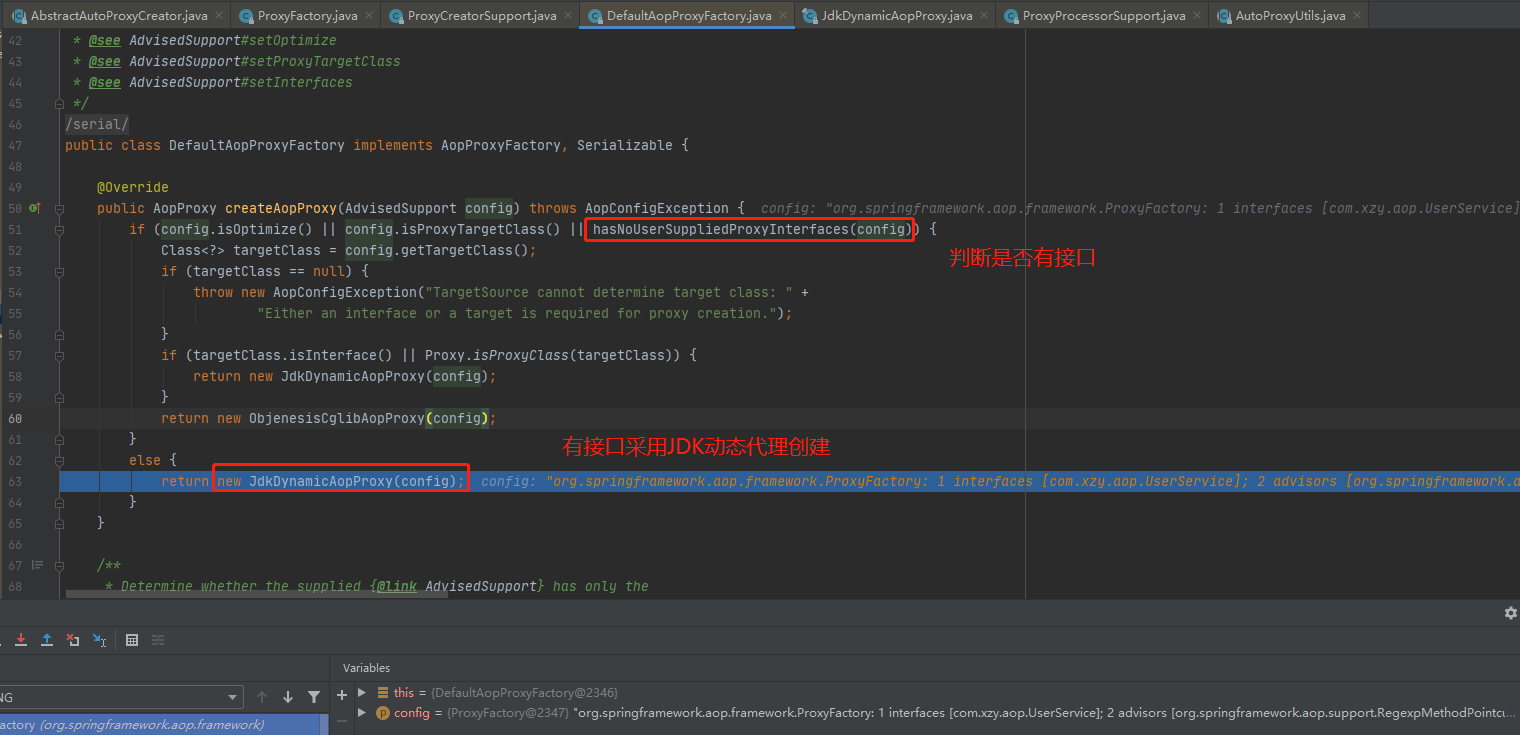


具体分析wrapIfNecessary方法



最后到了是使用cglib还是JDK创建代理的方法中，通过判断是否有接口来决定按照那种方式创建代理

org\springframework\aop\framework\DefaultAopProxyFactory.java



#### <aop:aspectj-autoproxy/>是如何实现将其转成代理的呢?

类似我们上面讲的

**<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"></bean>**

这个类，<aop:aspectj-autoproxy/>的原理也是通过这个自定义的标签帮我们注入了AnnotationAwareAspectJAutoProxyCreator类，功能与DefaultAdvisorAutoProxyCreator类似，只是他是通过注解来扫描。