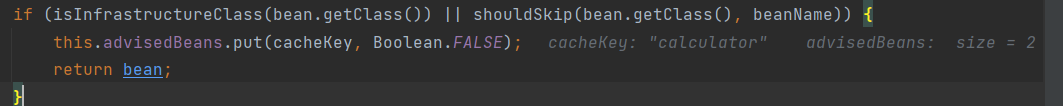
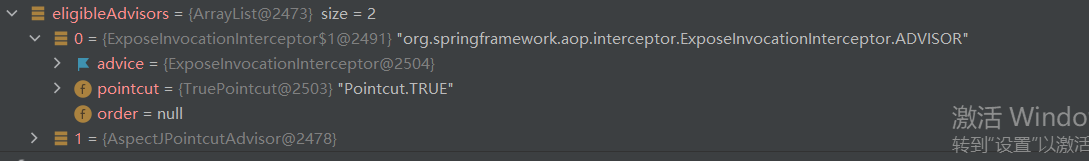
1. postProcessBeforeInstantiation创建所有的Advisor
2. 判断是否代理
3. 基础类不用代理：Advisor.class、Advice.class、AopInfrastructureBean.class
4. shouldSkip：创建所有AspectJPointCutAdvisor（根据存储的Advisor名字），如果bean为其中一个切面aspect则跳过，否则不能跳过

20210810005234

1. 如果不需要代理，则记录下来



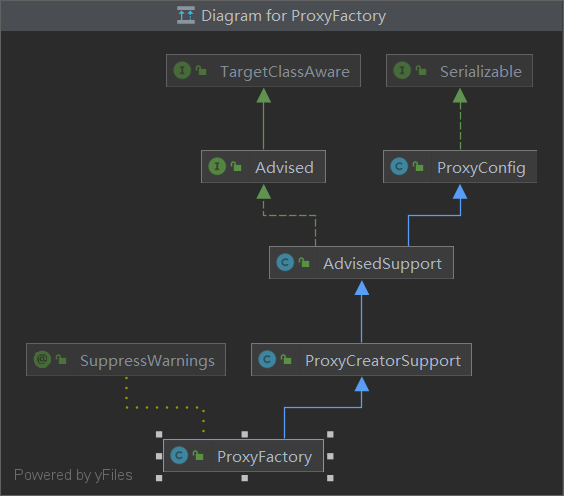
1. wrapIfNecessary创建代理对象
2. 如果不需要代理、或者可以跳过，返回普通对象
3. 如果需要代理
4. 准备所需的Advisor：除了当前类所需的AspectJPointCutAdvisor，还需要ExposeInvocationInterceptor，为了后续责任链拦截器的实现；然后记录当前对象需要被代理



即一个ExposeInvocationInterceptor + n个用户定义的AspectJPointCutAdvisor

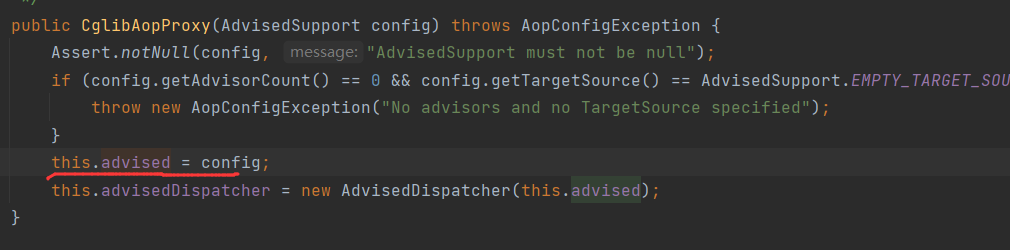
1. 创建代理对象：createProxy(bean.getClass(), beanName, specificInterceptors, new SingletonTargetSource(bean))
2. createProxy创建代理对象
3. 实例化ProxyFactory（继承AdvisedSupport）对象

填充参数、设置Advisors、设置TargetSource（被代理的对象）、

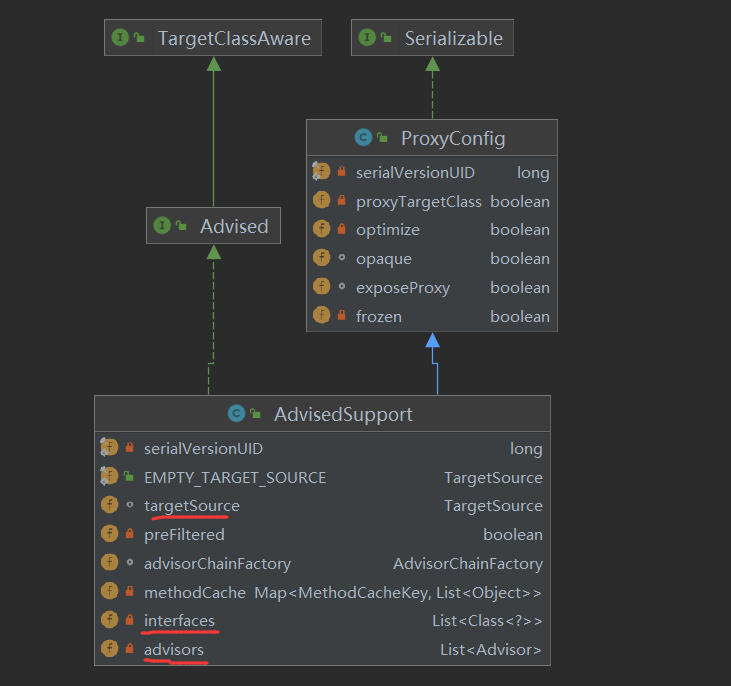


1. 通过proxyFactory选择jdk代理（JdkDynamicAopProxy）还是cglib代理 （ObjenesisCglibAopProxy），并将proxyFactory属性设置到代理类JdkDynamicAopProxy或者ObjenesisCglibAopProxy的adviced字段

由于proxyFactory继承AdvisedSupport，所以直接传this即可



Advice的参数proxyFactory都设置了

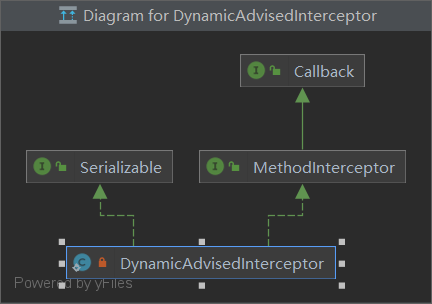


1. jdk代理或者cglib代理，cglib代理

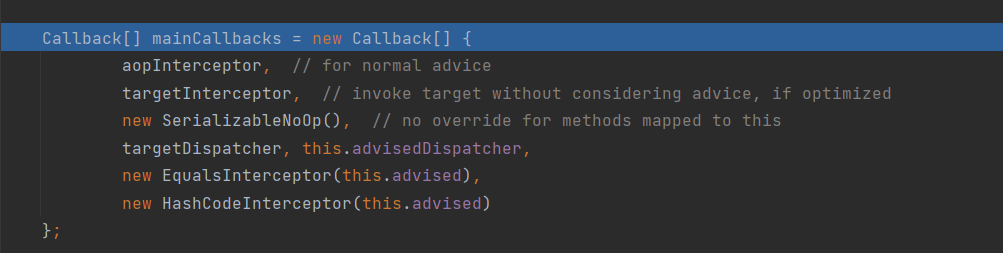
CglibAopProxy：getProxy(@Nullable ClassLoader classLoader)

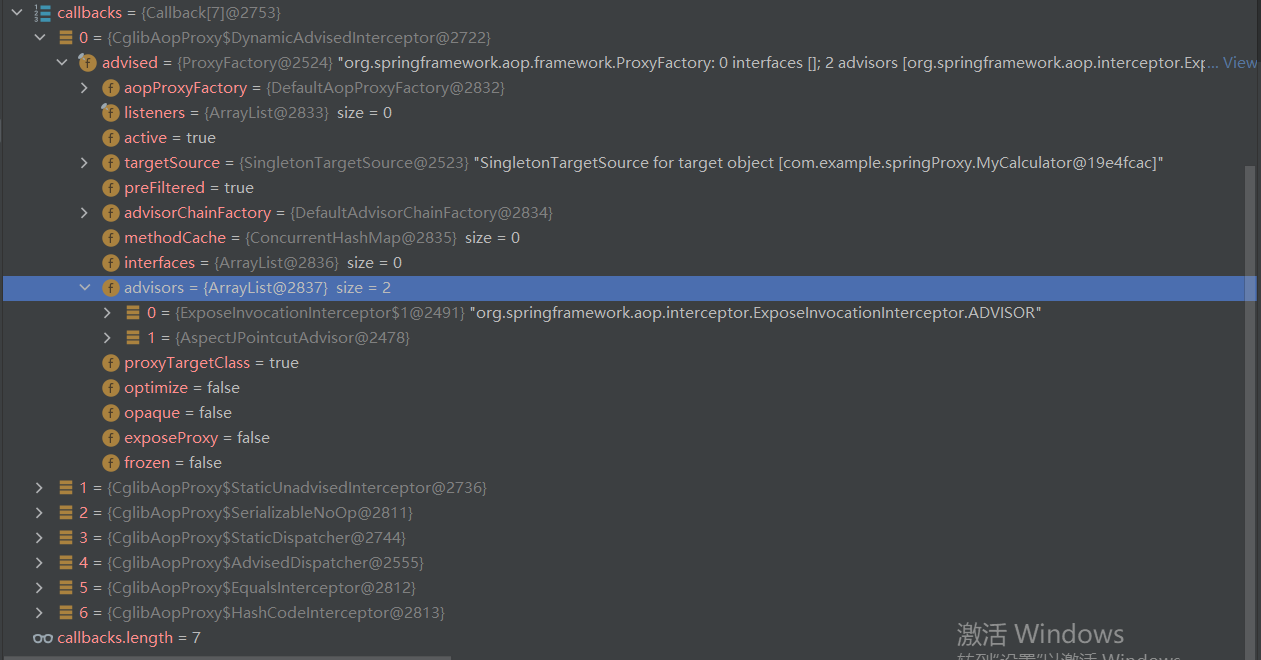
1. cglib创建代理对象
2. 通过advice获取被代理对象的Class对象
3. 创建Enhancer
4. 设置父类（被代理对象）
5. 设置接口
6. 设置Filter
7. 创建CallBack

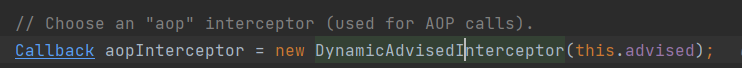
（接口MethodInceptor继承它）

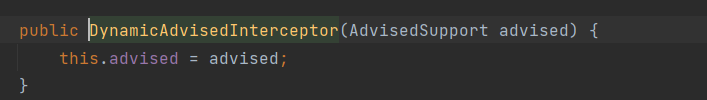


主要是第一个，第一个CallBack是包含了代理对象所需的所有Advisor（因为adviced就包含了前面获取的所需的Advisor）







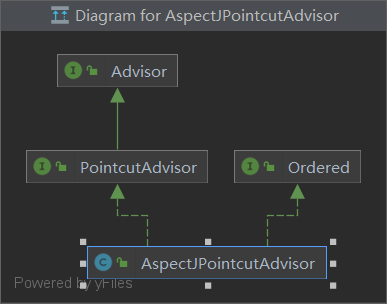


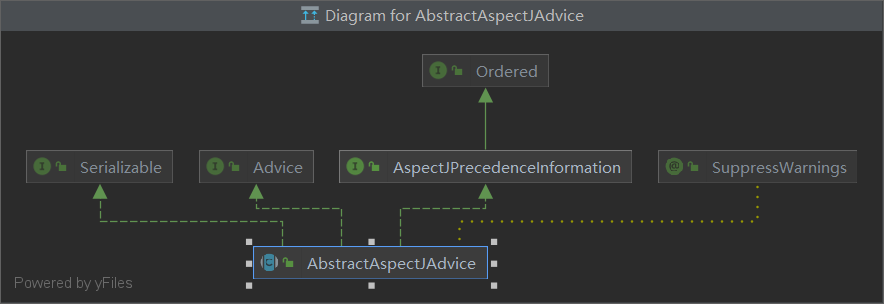
1. 通过Enhancer创建代理对象proxyInstance
2. 将CallBacks设置到proxyInstance中

方法调用

1. DynamicAdvisedInterceptor的intercept
2. 获取TargetSource
3. 根据匹配当前方法的Advisors，创建拦截链

遍历Advisors，获得可以匹配当前方法的Advisor（通过表达式match），并把它转化为MethodInceptor（需要有interruptor方法）





1. 通过CglibMethodInvocation.proceed()开始执行拦截链
2. 创建CglibMethodInvocation，设置TargetSource、拦截链

