链滴

Spring AOP

作者：someone33881
原文链接：https://ld246.com/article/1542542887784
来源网站：链滴

    本文主要是记录在学习
pring AOP过程中的一些知识点记录！

关键词：**AOP代理、AspectJ静态代理、SpringAOP动态代理**

问题

1、什么时候使用过JDK动态代理和CGLib动态代理?

一、AOP - Aspect Orient Programming

AOP面向切面编程，是对面向对象编程的一种补充，常用于处理系统中各模块中的公共横切关注，如事务管理、日志、缓存等；

AOP实现的关键在于AOP框架自动创建的AOP代理（静态代理、动态代理），静态代理以AspectJ代表且动态代理SpringAOP为代表；

AspectJ是在编译时增强了目标对象，SpringAOP则是在每次运行时动态地增加，区别在于二者生AOP代理对象的时机不同（相对来说AspectJ静态代理性能更好些，但需要特定编译器处理而SpringOP无需特定编译器）

调用者类 -> **伪装类**（ -> 目标类），所谓aop就是将对目标类的用用伪装类进行替换，即**代理对象的方法=增强处理+被代理对象的方法**

二、AspectJ

实现原理：静态代理 - 编译时增强

所谓静态代理，就是AOP框架在编译阶段生成AOP代理类，并且AspectJ则是静态代理的增加（->以也称为编译时增强）

三、Spring AOP

** 实现原理：动态代理 - JDK动态代理、CGLIB动态代理**

所谓动态代理，就是AOP框架不会去修改字节码，而是在内存中为方法临时生成一个AOP对象（含了目标对象的全部方法）并且在特定的切点做了增强处理和回调原对象的方法（->所以也称为运行增强）

SpringAOP的动态代理有两种方式 - JDK动态代理和CGLIB动态代理

JDK动态代理:

JDK动态代理通过**反射**来接收被代理的类，并且要求被代理的类必须实现一个接口；

JDK动态代理的核心是InvocationHandler接口和Proxy类

CGLIB动态代理:

如果目标类没有实现接口，那么SpringAOP则会选择CGLIB动态代理目标类；

CGLIB（Code Generation Library）即一个代码生成的类库，实现在运行时动态地为某个类生成子类，也即是通过**继承**实现动态代理（->因此final类时无法使用CGLIB动态代理的）

如果目标类没有实现接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code eneration Library），是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLI是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代的。

四、核心概念

通知Advice:

需要在连接点处所执行的代码，分为前置、后置、异常、最终、环绕通知；；；**也即，想要的功能安全、事务、日志等，先定义好才能仔想用的地方用一下；**

连接点JoinPoint:

定义的可被拦截到的点（Spring只支持方法连接点，而AspectJ则也支持构造器或属性字段注入）；；**也即允许使用通知的地方，如方法的前、后、前后、异常；**

切入点PointCut:

连接点的集合，由切入点表达式进行指定的需要被拦截的哪些类的哪些方法，在类运行时动态织入知；；；**也即是用切点筛选出方法（连接点）的，选中想要使用通知的那几个方法的；**

切面Aspect:

通知 + 切入点，是对横切关注点的抽象，抽取重复的功能代码；；；**也即切面是知和切入点的结合，前者说明了做什么，后者说明在哪个地方执行，这就是一个完整的切面；**

引入introduction:

在不修改代码的前提下，在运行期为类动态地添加一些方法或属性；；；**也即把切面（通知定义的方法属性）用到目标类中；**

织入weaving：

把切面应用到目标对象和创建新的代理对象的过程；

目标target：

被织入切面的目标类，也即需要被通知的对象，只关注于其业务本身的逻辑；

五、SpringAOP注解

1、@Aspect： 指定一个类为切面类；

2、@Pointcut("execution(* cn.zorke.e_aop_
nno..(..))")： 指定切入点表达式；

3、@Before("pointCut_()")： 前置通知，在目标方法执行之前执行；

4、@After("pointCut_()")： 后置通知，在目标方法执行之后执行（始终执行）；

5、@AfterReturning("pointCut_()")： 返回后通知，目标方法执行结束前执行（异常不执行）；

6、@AfterThrowing("pointCut_()")： 前置通知，在目标方法出现异常时候执行；

7、@Around("pointCut_()")： 前置通知，环绕目标方法执行；

参考：

1、SpringAOP概念理解（通俗易懂）