Spring AOP原理

1. AOP基本概念

通知（Advice）：想要执行的功能，比如说验证、日志等，Spring使用一系列继承自Advice类的子类来对这样的行为进行封装，比如BeforeAdvice、AfterAdvice、AroundAdvice、ThrowsAdvice等。

连接点（JoinPoint）：spring允许用户进行通知的地方，比如说某个方法的前后、或者抛出异常的地方等，如果一类有十个函数，那么它就至少有二十个连接点。

切点（PointCut）：切点就是被用户选中的连接点，它是连接点的一个子集。用户通过配置，选择自己想要使用特定通知的地方，这些被选中的地方就叫做切点。

切面（Aspect）：切点与通知的结合就叫做切面。切点告诉在什么地方，通知指定了干什么，两者结合就可以让spring知道在什么地方干什么，这就叫切面。

引入（Introduction）：把切面作用于某一个类，告诉spring哪个类的哪个方法的哪个地方需要干什么。

目标（Target）：引入中所提到的目标类，也就是被通知的对象。在目标类没有知觉的情况下，被代理类替换，完成用户指定的AOP任务。

织入（Weaving）：根据用户指定的切面与目标，创建代理对象的过程。使用Java.lang.reflect.proxy或者使用CGLib完成这一过程。

AOP运行过程：根据用户的AOP配置，动态地使用jvm动态代理或者CGLib的方式，创建目标类的代理类，在指定特定方法时，先执行通知中的动作，再执行目标类的方法。

2. Advice类

Spring AOP提供BeforeAdvice，AfterAdvice，ThrowsAdvice这三大接口，以及一系列实现这三个接口的更具体的子类，比如CountingBeforeAdvice等，用户统计方法的调用次数、异常的抛出次数等等。

3. Pointcut类

Pointcut决定Advice应该作用于哪个连接点。在Spring中，基础的Pointcut接口，包含有ClassFilter对象和MethodMatcher对象，MethodMatcher类用于匹配方法，决定这个切点所匹配的类与方法。从Pointcut接口衍生出去，Spring提供了许多子类，使用不同的匹配方法，比如JdkRegexpMethodPointcut是使用jdk的正则匹配方式，来匹配方法；还有NameMatchMethodPointcut类是简单地使用名称来匹配方法等等。

4. Advisor通知器

Advisor通知器就是Advice和Pointcut的结合，一个Advisor可以定义应该再哪个切点执行什么行为。以DefaultPointcutAdvisor类为例，通过持有Advice和Pointcut对象，完成他的任务。

5. JVM动态代理

使用jdk的反射功能和Java.reflect.Proxy类库，在运行时动态地构建出代理类，完成代理操作。使用proxy类库提供的静态方法，Proxy.newInstance方法，传入类加载器、对应的接口、需要扩展的目标类的方法等，动态地生成代理类。

\*静态代理jvm动态代理都需要目标类实现过一个接口才能使用，对于单独的，没有实现接口的类，就需要使用CGLib的方法创建代理类。

6. 建立AopProxy代理对象

通过xml配置文件，来进行AOP的实现在ProxyFactoryBean这个类中。大致流程是通过读取xml的配置，生成对应的ProxyFactoryBean以及所有的Advisor，然后根据这些Advisor的内容，决定使用JDK动态代理的方式，还是CGLib的方式，生成对应的proxy类。

\*详细的流程比较复杂，细节很多，目前还没有详细看

7. AOP拦截器调用的实现