27_精选面试题：Spring AOP 和 AspectJ AOP 存在哪些区别？

1、开篇

上节课聚焦在 Spring AOP CGLIB 的动态代理的源码分析，首先通过一个例子回顾 CGLIB 的应用是如何实现的，然后通过源码分析将 CGLIB 中 Class 对象创建过程做了讲解，包括生成指定类的 Class 对象字节数组，将 Class 对象字节数组转换为 Class 对象，通过 Class.forName 方法将 Class 对象装载到 JVM。本节课介绍 Spring AOP 和 AspectJ AOP 存的区别。

2、AOP 的基本概念

AOP（Aspect Orient Programming），它是面向对象编程的一种补充，主要应用于处理一些具有横切性质的系统级服务，如日志收集、事务管理、安全检查、缓存、对象池管理等。

AOP 实现的关键就在于 AOP 框架自动创建的 AOP 代理，AOP 代理则可分为静态代理和动态代理两大类，其中静态代理是指使用 AOP 框架提供的命令进行编译，从而在编译阶段就可生成 AOP 代理类，因此也称为编译时增强；而动态代理则在运行时借助于 JDK 动态代理、CGLIB 等在内存中“临时”生成 AOP 动态代理类，因此也被称为运行时增强。

面向切面的编程（AOP） 是一种编程范式，旨在通过允许横切关注点的分离，提高模块化。AOP 提供切面来将跨越对象关注点模块化。

AOP 要实现的是在我们写的代码的基础上进行一定的包装，如在方法执行前、或执行后、或是在执行中出现异常后这些地方进行拦截处理或叫做增强处理

AOP 的几个要素：

- Pointcut：是一个（组）基于正则表达式的表达式，有点绕，就是说他本身是一个表达式，但是他是基于正则语法的。通常一个 pointcut，会选取程序中的某些我们感兴趣的执行点，或者说是程序执行点的集合。

- JoinPoint：通过 pointcut 选取出来的集合中的具体的一个执行点，我们就叫 JoinPoint.
- Advice：在选取出来的 JoinPoint 上要执行的操作、逻辑。关于 5 种类型，我不多说，不懂的同学自己补基础。
- Aspect：就是我们关注点的模块化。这个关注点可能会横切多个对象和模块，事务管理是横切关注点的很好的例子。它是一个抽象的概念，从软件的角度来说是指在应用程序不同模块中的某一个领域或方面。又 pointcut 和 advice 组成。
- Weaving：把切面应用到目标对象来创建新的 advised 对象的过程。

3、Spring AOP 和 AspectJ 的区别

相信作为 Java 开发者我们都很熟悉 Spring 这个框架，在 spring 框架中有一个主要的功能就是 AOP，提到 AOP 就往往会想到 AspectJ，下面对 AspectJ 和 Spring AOP 作一个简单的比较：

（1）Spring AOP

- 基于动态代理来实现，默认如果使用接口的，用 JDK 提供的动态代理实现，如果是方法则使用 CGLIB 实现
- Spring AOP 需要依赖 IOC 容器来管理，并且只能作用于 Spring 容器，使用纯 Java 代码实现
- 在性能上，由于 Spring AOP 是基于动态代理来实现的，在容器启动时需要生成代理实例，在方法调用上也会增加栈的深度，使得 Spring AOP 的性能不如 AspectJ 的那么好。

（2）AspectJ

AspectJ 来自于 Eclipse 基金会，其实现机制属于静态织入，通过修改代码来实现，有如下几个织入的时机：

- 编译期织入（Compile-time weaving）： 如类 A 使用 AspectJ 添加了一个属性，类 B 引用了它，这个场景就需要编译期的时候就进行织入，否则没法编译类 B。
- 编译后织入（Post-compile weaving）： 也就是已经生成了 .class 文件，或已经打成 jar 包了，这种情况我们需要增强处理的话，就要用到编译后织入。

- 类加载后织入（Load-time weaving）： 指的是在加载类的时候进行织入，要实现这个时期的织入，有几种常见的方法。1、自定义类加载器来干这个，这个应该是最容易想到的办法，在被织入类加载到 JVM 前去对它进行加载，这样就可以在加载的时候定义行为了。2、在 JVM 启动的时候指定 AspectJ 提供的 agent：-javaagent:xxx/xxx/aspectjweaver.jar。

  AspectJ 可以做 Spring AOP 干不了的事情，它是 AOP 编程的完全解决方案，Spring AOP 则致力于解决企业级开发中最普遍的 AOP（方法织入）。而不是成为像 AspectJ 一样的 AOP 方案

  因为 AspectJ 在实际运行之前就完成了织入，所以说它生成的类是没有额外运行时开销的

3、Spring AOP 和 AspectJ 对照表

在说完 Spring AOP 和 AspectJ 的区别之后，如图 1 所示，我们将它们的对照表放到这里，供大家参考。

Spring AOP	AspectJ
在纯 Java 中实现	使用 Java 编程语言的扩展实现
不需要单独的编译过程	除非设置 LTW，否则需要 AspectJ 编译器 (ajc)
只能使用运行时织入	运行时织入不可用。支持编译时、编译后和加载时织入
功能不强-仅支持方法级编织	更强大 – 可以编织字段、方法、构造函数、静态初始值设定项、最终类/方法等……。
只能在由 Spring 容器管理的 bean 上实现	可以在所有域对象上实现
仅支持方法执行切入点	支持所有切入点
代理是由目标对象创建的，并且切面应用在这些代理上	在执行应用程序之前 (在运行时) 前，各方面直接在代码中进行织入
比 AspectJ 慢多了	更好的性能
易于学习和应用	相对于 Spring AOP 来说更复杂

图 1 Spring AOP 和 AspectJ 的对照表

4、总结

本节课介绍了 AOP 的基本概念，然后对 Spring AOP 和 AspectJ 分别做了描述，并且将两种技术的区别进行了讲解，最后提供一张对照表给大家参考。本节课是这周的最后一节课，这里对本周的课程做一个总结。

本周从 AOP 原理说起，先介绍了 Spring AOP 的两种实现模式：JDK 动态代理和 CGLib 动态代理。并且针对 AOP 的几个基本元素 Aspect、Joint Pints、Piontcuts 和 Advice 进行了介绍。顺势将 JDK 动态代理和 CGLIB 动态代理的代码进行了刨析，最后将 Spring AOP 和 AspectJ AOP 通过对照表的形式进行了区别。

下周课程会专注于 Spring AOP 在系统中的应用，课程会涉及到日志和异常处理的部分。下期见，拜拜。