Spring 常见问题汇总

作者：AlanSune
原文链接：https://ld246.com/article/1591530110444
来源网站：链滴

本文收集了一些在面试时被经常问及的关于Spring的主要问题，这些问题有可能在下次面试时就会被到。

AOP原理

AOP的家庭成员

PointCut

即在哪个地方进行切入,它可以指定某一个点，也可以指定多个点。比如类A的methord函数，当然一的AOP与语言(AOL)会采用多用方式来定义PointCut,比如说利用正则表达式，可以同时指定多个类的个函数。

Advice

在切入点干什么，指定在PointCut地方做什么事情(增强)，打日志、执行缓存、处理异常等等。

Advisor/Aspect

PointCut + Advice 形成了切面Aspect，这个概念本身即代表切面的所有元素。但到这一地步并不是整的，因为还不知道如何将切面植入到代码中，解决此问题的技术就是PROXY。

Proxy

Proxy 即代理，其不能算做AOP的家庭成员，更相当于一个管理部门，它管理了AOP的如何融入OO。之所以将其放在这里，是因为Aspect虽然是面向切面核心思想的重要组成部分，但其思想的践行者是Proxy,也是实现AOP的难点与核心据在。

AOP代理

AOP代理主要分为静态代理和动态代理，静态代理的代表为AspectJ；而动态代理则以Spring AOP为表。AspectJ在编译时就增强了目标对象，Spring AOP的动态代理则是在每次运行时动态的增强，生AOP代理对象，区别在于生成AOP代理对象的时机不同，相对来说AspectJ的静态代理方式具有更好性能，但是AspectJ需要特定的编译器进行处理，而Spring AOP则无需特定的编译器处理。

AspectJ

AspectJ是静态代理的增强，所谓的静态代理就是AOP框架会在编译阶段生成AOP代理类，因此也称编译时增强。实质为修改字节码。

Spring AOP

与 AspectJ 相同的是，Spring AOP 同样需要对目标类进行增强，也就是生成新的 AOP 代理类；与 AspectJ 不同的是，Spring AOP 无需使用任何特殊命令对 Java 源代码进行编译，它采用运行时动态地在内存中临时生成“代理类”的方式来生成 AOP 代理。

-

```
before
-- sayHello() --
class com.sun.proxy.$Proxy53
```

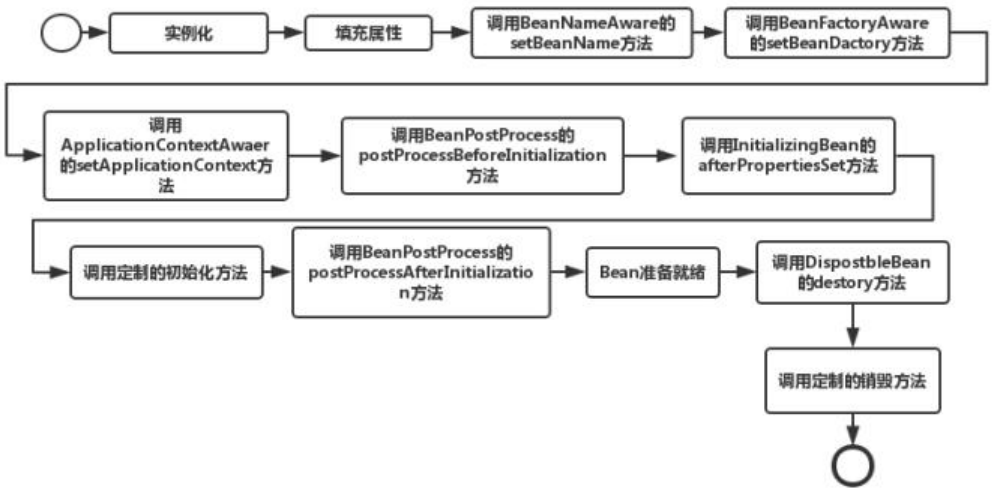
可以看到类型是**com.sun.proxy.$Proxy53**，也就是前面提到的Proxy类，因此这里Spring AOP使了JDK的动态代理。

-

```
before
-- sayHello() --
class com.listenzhangbin.aop.Chinese$$EnhancerBySpringCGLIB$$56b89168
```

可以看到类被CGLIB增强了，也就是动态代理。这里的CGLIB代理就是Spring AOP的代理，这个类也是所谓的AOP代理，AOP代理类在切点动态地织入了增强处理。

Bean的生命周期



1. Spring对Bean进行实例化(相当于程序中的new Xx())

2. Spring将值和Bean的引用注入进Bean对应的属性中

3. 如果Bean实现了BeanNameAware接口，Spring将Bean的ID传递给setBeanName()方法(实现BeaNameAware清主要是为了通过Bean的引用来获得Bean的ID，一般业务中是很少有用到Bean的ID的)

4. 如果Bean实现了BeanFactoryAware接口，Spring将调用setBeanDactory(BeanFactory bf)方法把BeanFactory容器实例作为参数传入。(实现BeanFactoryAware 主要目的是为了获取Spring容器如Bean通过Spring容器发布事件等)

5. 如果Bean实现了ApplicationContextAwaer接口，Spring容器将调用setApplicationContext(ApplcationContext ctx)方法，把y应用上下文作为参数传入.(作用与BeanFactory类似都是为了获取Sprin容器，不同的是Spring容器在调用setApplicationContext方法时会把它自己作为setApplicationConext 的参数传入，而Spring容器在调用setBeanDactory前需要程序员自己指定(注入)setBeanDactor里的参数BeanFactory)

6. 如果Bean实现了BeanPostProcess接口，Spring将调用它们的postProcessBeforeInitialization(初始化)方法(作用是在Bean实例创建成功后对进行增强处理，如对Bean进行修改，增加某个功能)

7. 如果Bean实现了InitializingBean接口，Spring将调用它们的afterPropertiesSet方法，作用与在置文件中对Bean使用init-method声明初始化的作用一样，都是在Bean的全部属性设置成功后执行初始化方法。(为同一个 bean 配置多个生命周期机制，创建顺序为:@postconstruct -> afterPropertesSet() -> init-method, 销毁顺序为：@PreDestroy -> destroy() -> destroy-method)

8. 如果Bean实现了BeanPostProcess接口，Spring将调用它们的postProcessAfterInitialization(后始化)方法(作用与6的一样，只不过6是在Bean初始化前执行的，而这个是在Bean初始化后执行的，机不同)

9. 经过以上的工作后，Bean将一直驻留在应用上下文中给应用使用，直到应用上下文被销毁

10. 如果Bean实现了DispostbleBean接口，Spring将调用它的destory方法，作用与在配置文件中对ean使用destory-method属性的作用一样，都是在Bean实例销毁前执行的方法