

1. 跟着@EnableAspectJAutoProxy的执行流程，可以发现最后注册了一个AnnotationAwareAspectJAutoProxyCreator 类型的BeanDefinition; 根据右图的 AnnotationAwareAspectJAutoProxyCreator 的继承关系，可以发现这样三个顶层接口：BeanPostProcessor，InstantiationAwareBeanPostProcessor，SmartInstantiationAwareBeanPostProcessor；追踪过SpringIOC的执行流程可以知道，在实例化的时候，Spring底层会有9个拓展点，其中就牵涉到了这三个接口！  
注意：AOP前面的解析会在实例化前完成，一般情况下，AOP动态代理创建是在初始化后，但是如果存在循环依赖问题的话，将会提前到实例化后创建！

2. Spring AOP四种实现方式：  
1. 基于接口的配置--spring1.2  
2. 纯POJO切面(XML)--spring2.0  
3. @Aspect注解驱动的切面(注解)  
4. 注入式AspectJ切面 (其实与Spring并无多大的关系，这个就是使用AspectJ这个框架实现Aop编程，需要用到AspectJ的编译器)

给容器中注册一个名称为org.springframework.aop.config.internalAutoProxyCreator  
类型为AnnotationAwareAspectJAutoProxyCreator的组件

注册bean定义的名称为： 'org.springframework.aop.config.internalAutoProxyCreator'

在配置类上，加上此注解，可以开启注解AOP

@EnableAspectJAutoProxy

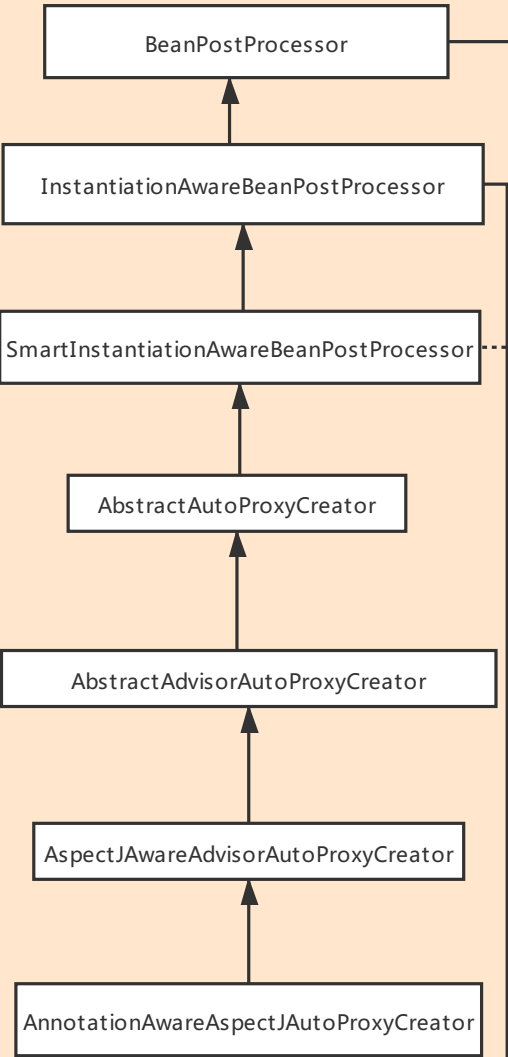
@Import(AspectJAutoProxyRegistrar.class)

AopConfigUtils.registerAspectJAnnotationAutoProxyCreatorIfNecessary(registry);

registerOrEscalateApcAsRequired(AnnotationAwareAspectJAutoProxyCreator.class, registry, source);

registry.registerBeanDefinition(AUT\_PROXY\_CREATOR\_BEAN\_NAME, beanDefinition);

AnnotationAwareAspectJAutoProxyCreator依赖关系



解决循环依赖问题 @Override  
public Object getEarlyBeanReference(Object bean, String beanName) {  
 Object cacheKey = getCacheKey(bean.getClass(), beanName);  
 this.earlyProxyReferences.put(cacheKey, bean);  
 return wrapIfNecessary(bean, beanName, cacheKey);  
}

第一步

第二步

解析切面

postProcessBeforeInstantiation()

advisedBeans  
是否解析过

否

isInfrastructureClass()

是否实现了Advisor，PointCut或者  
AopInfrastructureBean接口

否

shouSkip()

找到候选的 advisor (通知，前置，后置等)  
findCandidateAdvisors()

Advisors的相关信息后封装为一个advisors  
buildAspectJAdvisors()

根据class 对象判断是不是切面  
isAspect()

提取Aspect类中所有的Advice方法  
getAdvisors()

将解析的结果放入缓存  
this.advisorsCache.put(beanName, classAdvisors);

解析切面  
的方法

创建动态代理

postProcessAfterInitialization()

wrapIfNecessary()

getAdvicesAndAdvisorsForBean()

createProxy()

找到了Advisor 【增强点】

创建代理对象，对目标方法进行增强