#### Spring-aop源码解析

Xml配置文件中：<aop:aspectj-autoproxy />开启，该标签使用AopNameSpaceHandler进行自定义标签的解析，如下标红处即解析该标签：

**public class AopNamespaceHandler extends** NamespaceHandlerSupport { **public void** *init*() {  
 registerBeanDefinitionParser("config", **new** ConfigBeanDefinitionParser());  
 registerBeanDefinitionParser("aspectj-autoproxy", **new** AspectJAutoProxyBeanDefinitionParser());  
 registerBeanDefinitionDecorator("scoped-proxy", **new** ScopedProxyBeanDefinitionDecorator());  
 registerBeanDefinitionParser("spring-configured", **new** SpringConfiguredBeanDefinitionParser());  
 }  
}

调用AspectJAutoProxyBeanDefinitionParser()的parse()方法解析，依次往下调用如下：

AopNamespaceUtils.*registerAspectJAnnotationAutoProxyCreatorIfNecessary*(parserContext, element);

**BeanDefinition** beanDefinition = AopConfigUtils.*registerAspectJAnnotationAutoProxyCreatorIfNecessary*(  
 parserContext.getRegistry(), parserContext.extractSource(sourceElement));

**public static BeanDefinition** *registerAspectJAnnotationAutoProxyCreatorIfNecessary*(**BeanDefinitionRegistry** registry, **Object** source) {  
 **return** *registerOrEscalateApcAsRequired*(**AnnotationAwareAspectJAutoProxyCreator**.**class**, registry, source);  
}

expose-proxy属性使用：目标对象（将要使用aop增强的对象，如service层）的内部调用this.b()不会使用增强，即没有事务增强，该属性设置为true后可使用AopContext.currentProxy().b()来使用代理对象进行事务增强。

##### AspectJAwareAdvisorAutoProxyCreator.class

**public Object** *postProcessAfterInitialization*(**Object** bean, **String** beanName) **throws** BeansException {  
 **if** (bean != **null**) {  
 **Object** cacheKey = getCacheKey(bean.getClass(), beanName);  
 **if** (!**this**.earlyProxyReferences.containsKey(cacheKey)) {  
 **return** wrapIfNecessary(bean, beanName, cacheKey);  
 }  
 }  
 **return** bean;  
**}**

**postProcessAfterInitialization方法会在bean初始化之后调用.....wrapIfNecessary方法!!**

// 如果有增强，那么创建代理  
**Object**[] specificInterceptors = getAdvicesAndAdvisorsForBean(bean.getClass(), beanName, **null**);  
**if** (specificInterceptors != *DO\_NOT\_PROXY*) {  
 **this**.advisedBeans.put(cacheKey, **Boolean**.*TRUE*);  
 **Object** proxy = createProxy(bean.getClass(), beanName, specificInterceptors, **new** SingletonTargetSource(bean));  
 **this**.proxyTypes.put(cacheKey, proxy.getClass());  
 **return** proxy;  
}