# 细节拼凑

## 1.@EnableAspectJAutoProxy

这个是在全局配置文件中的，作用是让动态代理**切面类**真正起作用，调用的方法主要是**boolean** proxyTargetClass() **default** **false，当值是false时用的是CGLIB（继承），true是JDK（接口）**

## 2. AspectJAutoProxyRegistrar

**public** **void** registerBeanDefinitions(

AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {

//如果有必要的话注册一个切面注解自动代理类创建器

AopConfigUtils.*registerAspectJAnnotationAutoProxyCreatorIfNecessary*(registry);

AnnotationAttributes enableAspectJAutoProxy =

AnnotationConfigUtils.*attributesFor*(importingClassMetadata, EnableAspectJAutoProxy.**class**);

**if** (enableAspectJAutoProxy != **null**) {

**if** (enableAspectJAutoProxy.getBoolean("proxyTargetClass")) {

AopConfigUtils.*forceAutoProxyCreatorToUseClassProxying*(registry);

}

**if** (enableAspectJAutoProxy.getBoolean("exposeProxy")) {

AopConfigUtils.*forceAutoProxyCreatorToExposeProxy*(registry);

}

}

}

## 3.registerOrEscalateApcAsRequired *registerOrEscalateApcAsRequired*(AnnotationAwareAspectJAutoProxyCreator.class, registry, source)

source==null

## 4．registerOrEscalateApcAsRequired

@Nullable

**private** **static** BeanDefinition registerOrEscalateApcAsRequired(Class<?> cls, BeanDefinitionRegistry registry,

@Nullable Object source) {

Assert.*notNull*(registry, "BeanDefinitionRegistry must not be null");

// ***AUTO\_PROXY\_CREATOR\_BEAN\_NAME=*** **org.springframework.aop.config.internalAutoProxyCreator**：作用是赋值beanId

**if** (registry.containsBeanDefinition(***AUTO\_PROXY\_CREATOR\_BEAN\_NAME***)) {

BeanDefinition apcDefinition = registry.getBeanDefinition(***AUTO\_PROXY\_CREATOR\_BEAN\_NAME***);

**if** (!cls.getName().equals(apcDefinition.getBeanClassName())) {

**int** currentPriority = *findPriorityForClass*(apcDefinition.getBeanClassName());

**int** requiredPriority = *findPriorityForClass*(cls);

**if** (currentPriority < requiredPriority) {

apcDefinition.setBeanClassName(cls.getName());

}

}

**return** **null**;

}

RootBeanDefinition beanDefinition = **new** RootBeanDefinition(cls);

beanDefinition.setSource(source);

beanDefinition.getPropertyValues().add("order", Ordered.***HIGHEST\_PRECEDENCE***);

beanDefinition.setRole(BeanDefinition.***ROLE\_INFRASTRUCTURE***);

registry.registerBeanDefinition(***AUTO\_PROXY\_CREATOR\_BEAN\_NAME***, beanDefinition);//注册bean

**return** beanDefinition;

}

## 5. AnnotationAwareAspectJAutoProxyCreator

AnnotationAwareAspectJAutoProxyCreator：

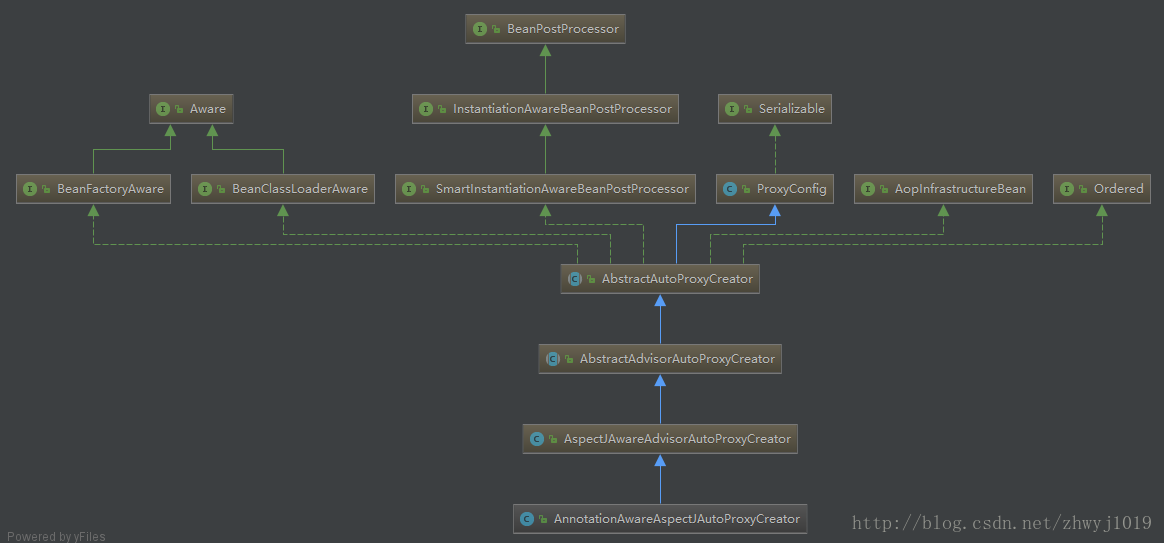
\* AnnotationAwareAspectJAutoProxyCreator

\* ->AspectJAwareAdvisorAutoProxyCreator

\* ->AbstractAdvisorAutoProxyCreator

\* ->AbstractAutoProxyCreator **extends** ProxyProcessorSupport

\* implements SmartInstantiationAwareBeanPostProcessor, BeanFactoryAware



**这个类的创建过程：//这个类的作用是去拦截被通知类的方法**

## 执行流程：

**new** AnnotationConfigApplicationContext(com.spring.aop.config.AopConfig.**class**)-- refresh()注:在@EnableAspectJAutoProxy下面的AspectJAutoProxyRegistrar这个类中的*registerAspectJAnnotationAutoProxyCreatorIfNecessary*(registry)方法加断点-- registerBeanPostProcessors(beanFactory)--registerBeanDefinitions--invokeBeanFactoryPostProcessors(beanFactory)-- *registerAspectJAnnotationAutoProxyCreatorIfNecessary*(registry)--*registerOrEscalateApcAsRequired*(AnnotationAwareAspectJAutoProxyCreator.**class**, registry, source)注：到此为止，都是再给bean定义

### 5.1 ProxyProcessorSupport

ProxyProcessorSupport **extends** ProxyConfig **implements** Ordered, BeanClassLoaderAware, AopInfrastructureBean

**IOC容器在创建bean时与优先级，先处理实现了PrivoityOrdered接口的类再处理Ordered接口的，再处理其他的**

### 5.2 SmartInstantiationAwareBeanPostProcessor

这是bean的后置处理器

# AOP-bean的创建

1. **new** AnnotationConfigApplicationContext(com.spring.aop.config.AopConfig.**class**)
2. refresh()
3. 3.1 invokeBeanFactoryPostProcessors(beanFactory)
4. 3.1.1
5. 3.2