

Spring注解驱动开发第26讲——总有人让我给他讲讲@EnableAspectJAutoProxy注解

@EnableAspectJAutoProxy注解

在配置类上添加@EnableAspectJAutoProxy注解，便能够开启注解版的AOP功能。也就是说，如果要使注解版的AOP功能起作用的话，那么就得需要在配置类上添加@EnableAspectJAutoProxy注解。我们先来看下@EnableAspectJAutoProxy注解的源码，如下所示。

```
16
17 package org.springframework.context.annotation;
18
19 import java.lang.annotation.Documented;
20 import java.lang.annotation.ElementType;
21 import java.lang.annotation.Retention;
22 import java.lang.annotation.RetentionPolicy;
23 import java.lang.annotation.Target;
24
25 * Enables support for handling components marked with AspectJ's {@code @Aspect} annotation,
109 @Target(ElementType.TYPE)
110 @Retention(RetentionPolicy.RUNTIME)
111 @Documented
112 @Import(AspectJAutoProxyRegistrar.class)
113 public @interface EnableAspectJAutoProxy {
114
115     /**
116      * Indicate whether subclass-based (CGLIB) proxies are to be created as opposed
117      * to standard Java interface-based proxies. The default is {@code false}.
118      */
119     boolean proxyTargetClass() default false;
120
121     /**
122      * Indicate that the proxy should be exposed by the AOP framework as a {@code ThreadLocal}
123      * for retrieval via the {@link org.springframework.aop.framework.AopContext} class.
124      * Off by default, i.e. no guarantees that {@code AopContext} access will work.
125      * @since 4.3.1
126      */
127     boolean exposeProxy() default false;
128
129 }
130
```

从源码中可以看出，@EnableAspectJAutoProxy注解使用@Import注解给容器中引入了AspectJAutoProxyRegistrar组件。那么，这个AspectJAutoProxyRegistrar组件又是什么呢？我们继续点进去到AspectJAutoProxyRegistrar类的源码中，如下所示。

```
2+ * Copyright 2002-2016 the original author or authors.
16
17 package org.springframework.context.annotation;
18
19 import org.springframework.aop.config.AopConfigUtils;
20 import org.springframework.beans.factory.support.BeanDefinitionRegistry;
21 import org.springframework.core.annotation.AnnotationAttributes;
22 import org.springframework.core.type.AnnotationMetadata;
23
24+ * Registers an {@link org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator}
34 class AspectJAutoProxyRegistrar implements ImportBeanDefinitionRegistrar {
35
36+ * Register, escalate, and configure the AspectJ auto proxy creator based on the value
41+ @Override
42 public void registerBeanDefinitions(
43     AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {
44
45     AopConfigUtils.registerAspectJAnnotationAutoProxyCreatorIfNecessary(registry);
46
47     AnnotationAttributes enableAspectJAutoProxy =
48         AnnotationConfigUtils.attributesFor(importingClassMetadata, EnableAspectJAutoProxy.class);
49     if (enableAspectJAutoProxy.getBoolean("proxyTargetClass")) {
50         AopConfigUtils.forceAutoProxyCreatorToUseClassProxying(registry);
51     }
52     if (enableAspectJAutoProxy.getBoolean("exposeProxy")) {
53         AopConfigUtils.forceAutoProxyCreatorToExposeProxy(registry);
54     }
55 }
56
57 }
58
```

可以看到AspectJAutoProxyRegistrar类实现了ImportBeanDefinitionRegistrar接口。我们再点击进入到ImportBeanDefinitionRegistrar接口的源码中，如下所示。

```
2+ * Copyright 2002-2013 the original author or authors.
16
17 package org.springframework.context.annotation;
18
19 import org.springframework.beans.factory.support.BeanDefinitionRegistry;
20 import org.springframework.beans.factory.support.BeanDefinitionRegistryPostProcessor;
21 import org.springframework.core.type.AnnotationMetadata;
22
24+ * Interface to be implemented by types that register additional bean definitions when
50 public interface ImportBeanDefinitionRegistrar {
51
52+ /**
53+ * Register bean definitions as necessary based on the given annotation metadata of
54+ * the importing {@code @Configuration} class.
55+ * <p>Note that {@link BeanDefinitionRegistryPostProcessor} types may <em>not</em> be
56+ * registered here, due to lifecycle constraints related to {@code @Configuration}
57+ * class processing.
58+ * @param importingClassMetadata annotation metadata of the importing class
59+ * @param registry current bean definition registry
60+ */
61 public void registerBeanDefinitions(
62     AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry);
63
64 }
```

看到ImportBeanDefinitionRegistrar这个接口，小伙伴们是不是觉得很熟悉呢！没错，我们在《Spring注解驱动开发第10讲——在@Import注解中使用ImportBeanDefinitionRegistrar向容器中注册bean》这篇文章中就介绍过。也就是说，我们可以通过ImportBeanDefinitionRegistrar接口实现将自定义的组件添加到IOC容器中。

也就是说，@EnableAspectJAutoProxy注解使用AspectJAutoProxyRegistrar对象自定义组件，并将相应的组件添加到了IOC容器中。

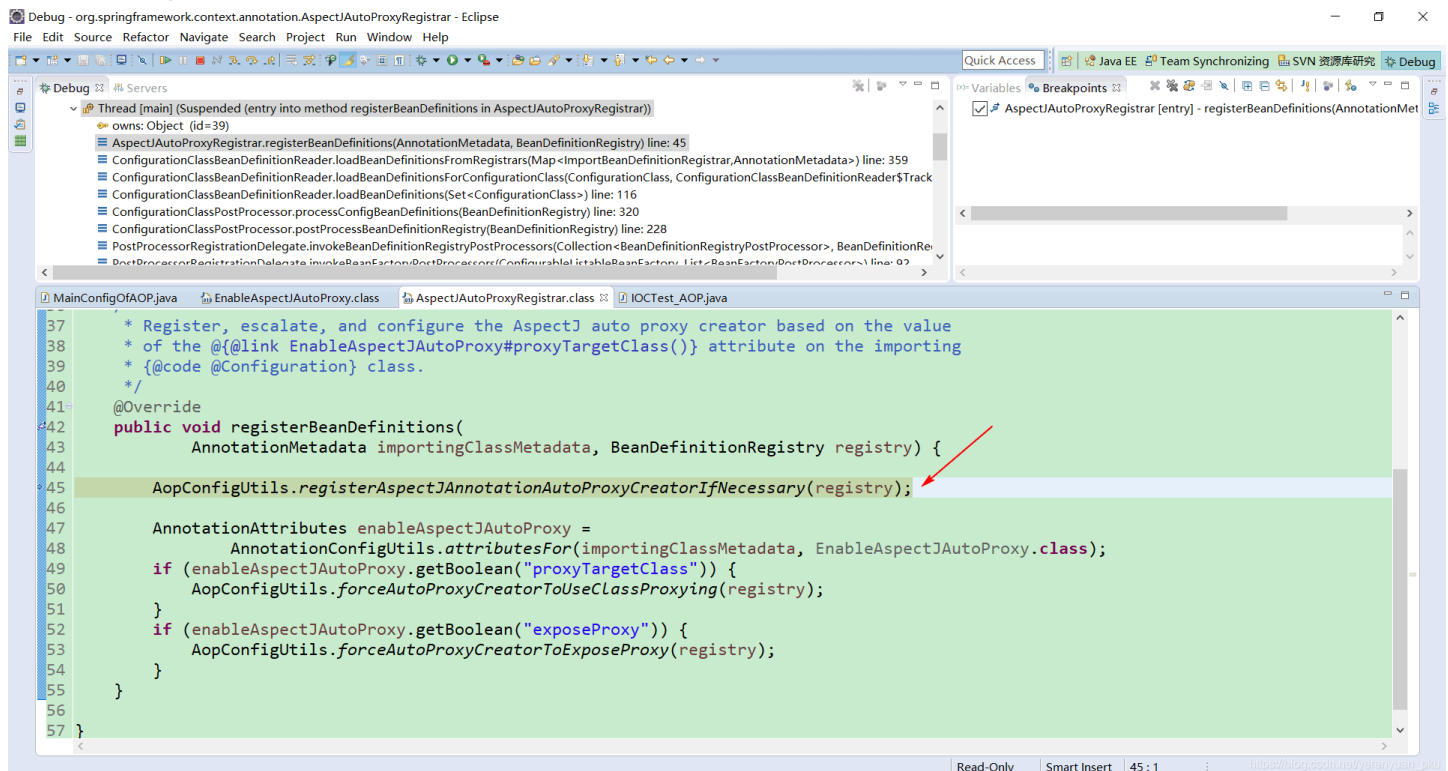
那么，@EnableAspectJAutoProxy注解使用AspectJAutoProxyRegistrar对象向容器中注册了一个什么bean呢？接下来咱们就要说道说道了。

调试Spring源码

首先，我们需要给这个AspectJAutoProxyRegistrar类打一个断点，断点就打在该类的registerBeanDefinitions()方法处，如下所示。

```
2+ * Copyright 2002-2016 the original author or authors.
16
17 package org.springframework.context.annotation;
18
19 import org.springframework.aop.config.AopConfigUtils;
20 import org.springframework.beans.factory.support.BeanDefinitionRegistry;
21 import org.springframework.core.annotation.AnnotationAttributes;
22 import org.springframework.core.type.AnnotationMetadata;
23
25+ * Registers an {@link org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreat
34 class AspectJAutoProxyRegistrar implements ImportBeanDefinitionRegistrar {
35
37+ * Register, escalate, and configure the AspectJ auto proxy creator based on the value
41+ @Override
42 public void registerBeanDefinitions(
43     AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {
44
45     AopConfigUtils.registerAspectJAnnotationAutoProxyCreatorIfNecessary(registry);
46
47     AnnotationAttributes enableAspectJAutoProxy =
48         AnnotationConfigUtils.attributesFor(importingClassMetadata, EnableAspectJAutoProxy.class);
49     if (enableAspectJAutoProxy.getBoolean("proxyTargetClass")) {
50         AopConfigUtils.forceAutoProxyCreatorToUseClassProxying(registry);
51     }
52     if (enableAspectJAutoProxy.getBoolean("exposeProxy")) {
53         AopConfigUtils.forceAutoProxyCreatorToExposeProxy(registry);
54     }
55 }
56
57 }
58
```

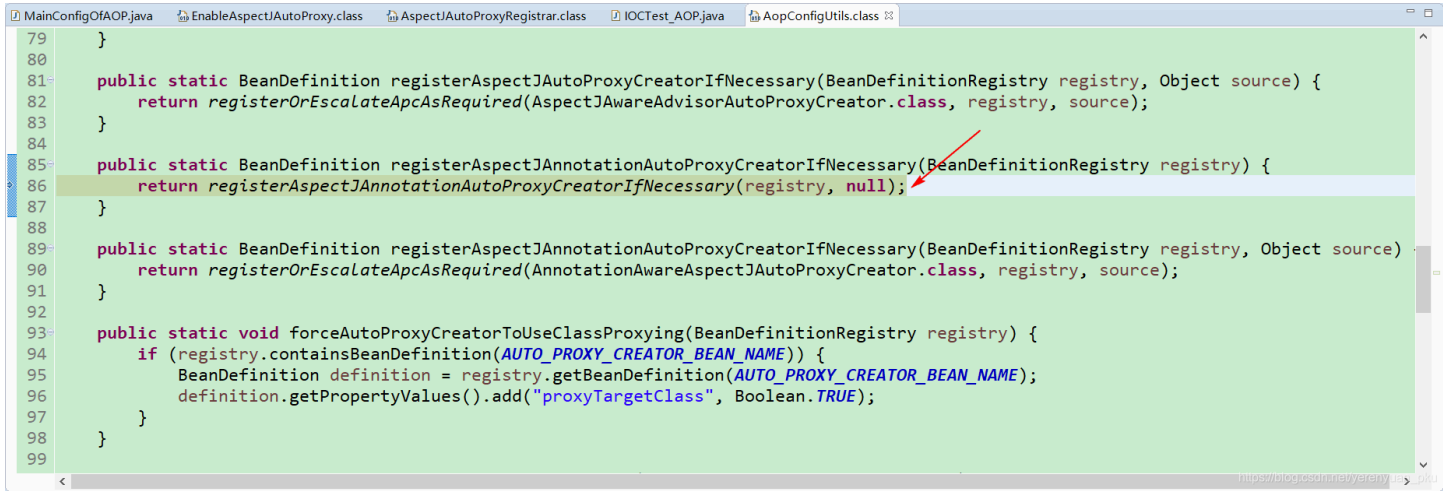
然后，我们以debug的方式来运行IOCTest_AOP类中的test01()方法。运行后程序进入到断点位置，如下所示。



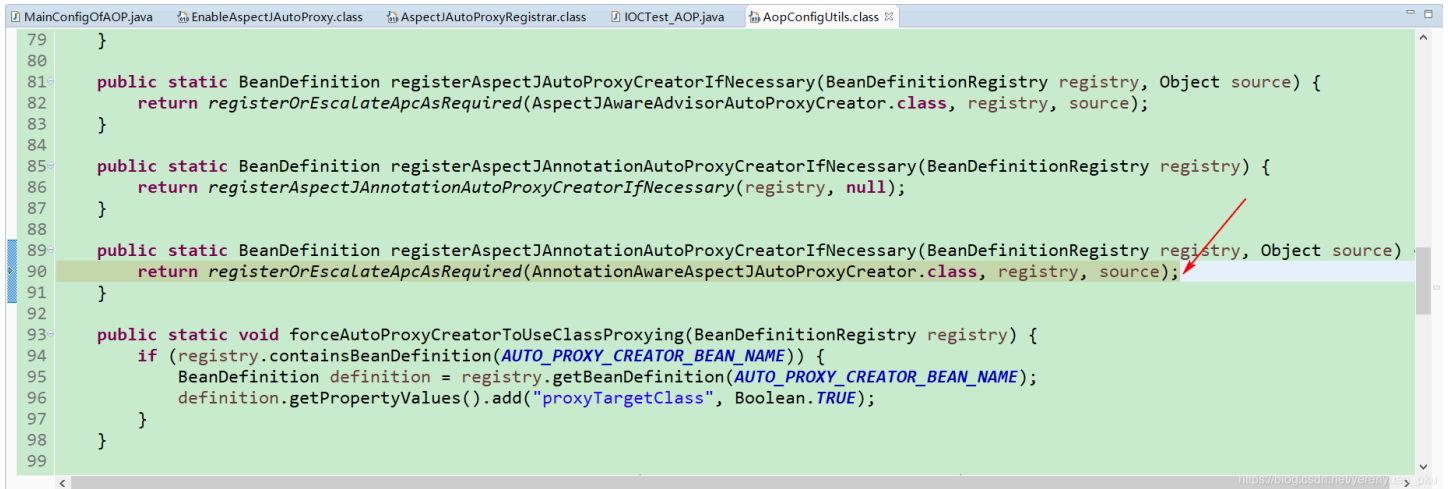
可以看到，程序已经暂停在断点位置了，而且在Eclipse 的左上角显示出了方法的调用栈。

我们还可以看到，在AspectJAutoProxyRegistrar类的registerBeanDefinitions()方法里面，首先调用了AopConfigUtils类的registerAspectJAnnotationAutoProxyCreatorIfNecessary()方法来注册registry，单看registerAspectJAnnotationAutoProxyCreatorIfNecessary()方法也不难理解，字面含义就是：如果需要的话，那么就注册一个AspectJAnnotationAutoProxyCreator组件。

接着，我们进入到AopConfigUtils类的registerAspectJAnnotationAutoProxyCreatorIfNecessary()方法中，如下所示。

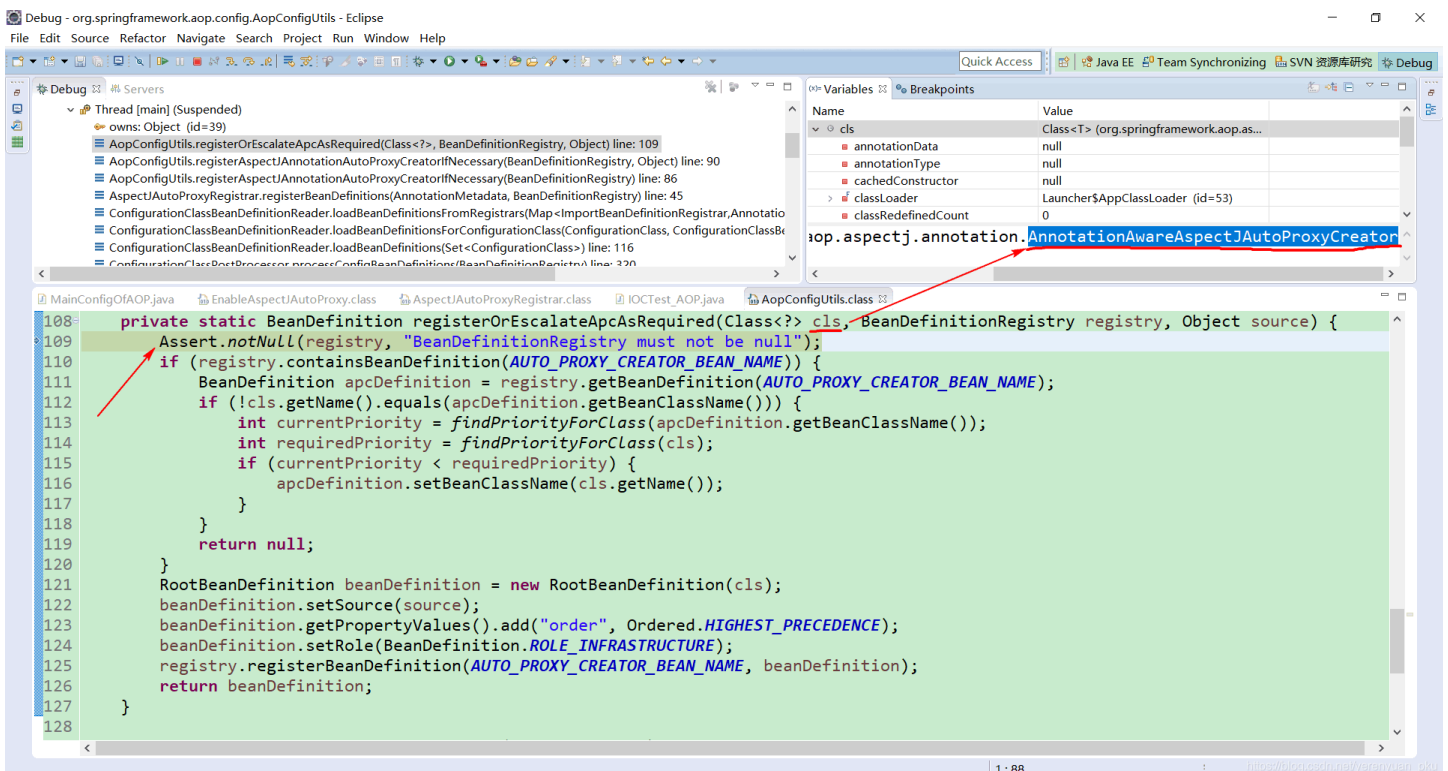


在AopConfigUtils类的registerAspectJAnnotationAutoProxyCreatorIfNecessary()方法中，直接调用了重载的registerAspectJAnnotationAutoProxyCreatorIfNecessary()方法。我们继续跟进代码，如下所示。



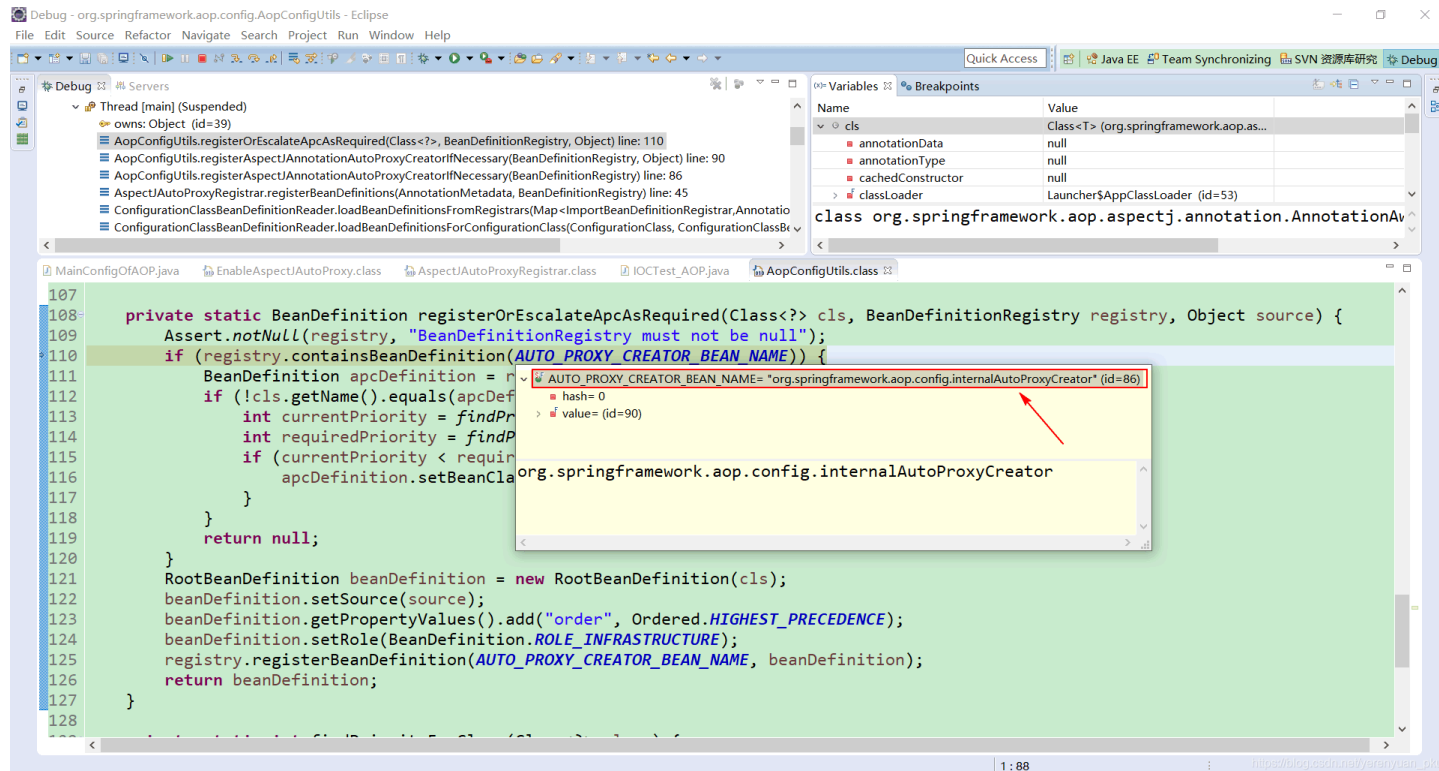
可以看到在重载的registerAspectJAnnotationAutoProxyCreatorIfNecessary()方法中直接调用了registerOrEscalateApcAsRequired()方法，并且在registerOrEscalateApcAsRequired()方法中，传入了AnnotationAwareAspectJAutoProxyCreator.class对象。

我们继续跟进代码，如下所示。



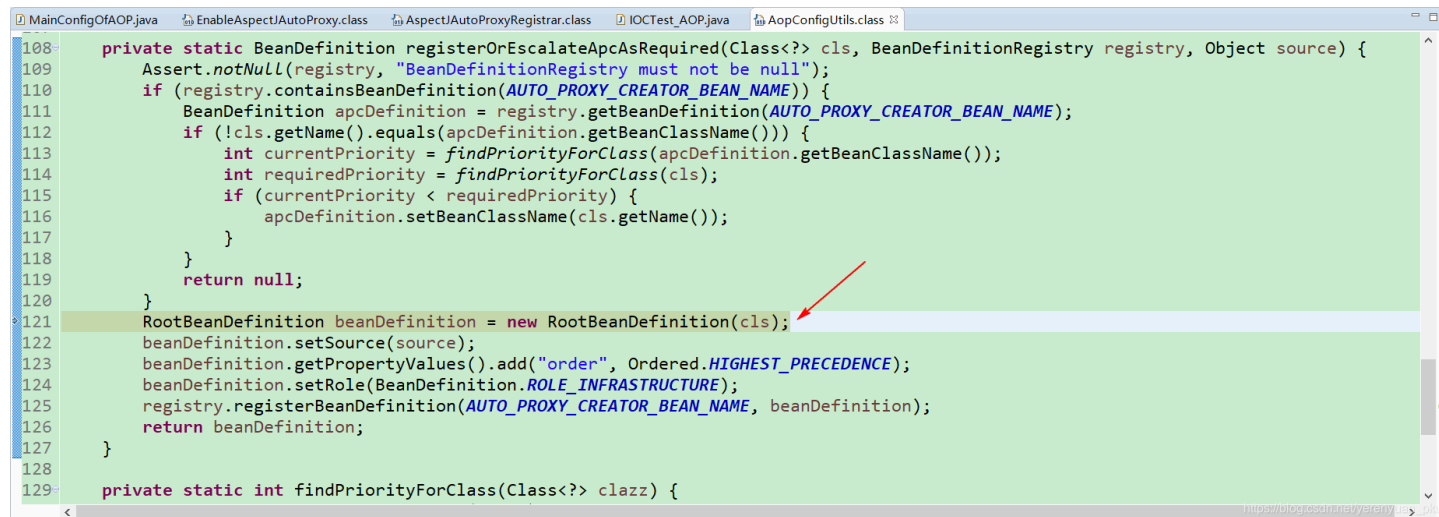
我们可以看到，在registerOrEscalateApcAsRequired()方法中，接收到的Class对象的类型为org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator。

除此之外，我们还可以看到，在registerOrEscalateApcAsRequired()方法中会做一个判断，即首先判断registry（也就是IOC容器）是否包含名称为org.springframework.aop.config.internalAutoProxyCreator的bean，如下所示。



如果registry中包含名称为org.springframework.aop.config.internalAutoProxyCreator的bean，那么就进行相应的处理。从Spring的源码来看，就是将名称为org.springframework.aop.config.internalAutoProxyCreator的bean从容器中取出，并且判断cls对象的名字值和apcDefinition的beanClassName值是否相等，若不相等，则获取apcDefinition和cls它俩的优先级，如果apcDefinition的优先级小于cls的优先级，那么将apcDefinition的beanClassName设置为cls的名字值。相对来说，理解起来还是比较简单的。

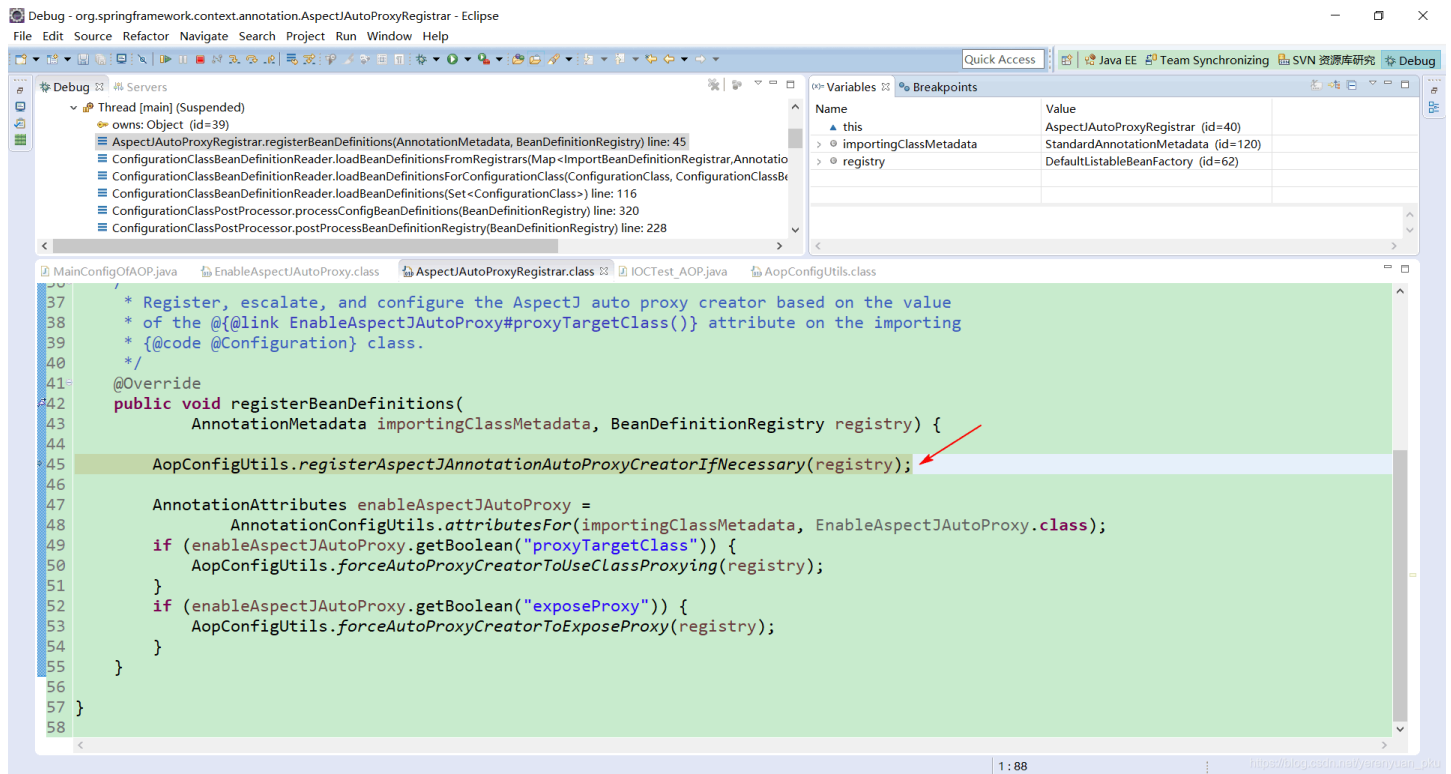
由于我们这里是第一次运行程序，容器中应该还没有包含名称为org.springframework.aop.config.internalAutoProxyCreator的bean，所以此时并不会进入到if判断条件中。我们继续往下看代码，如下所示。



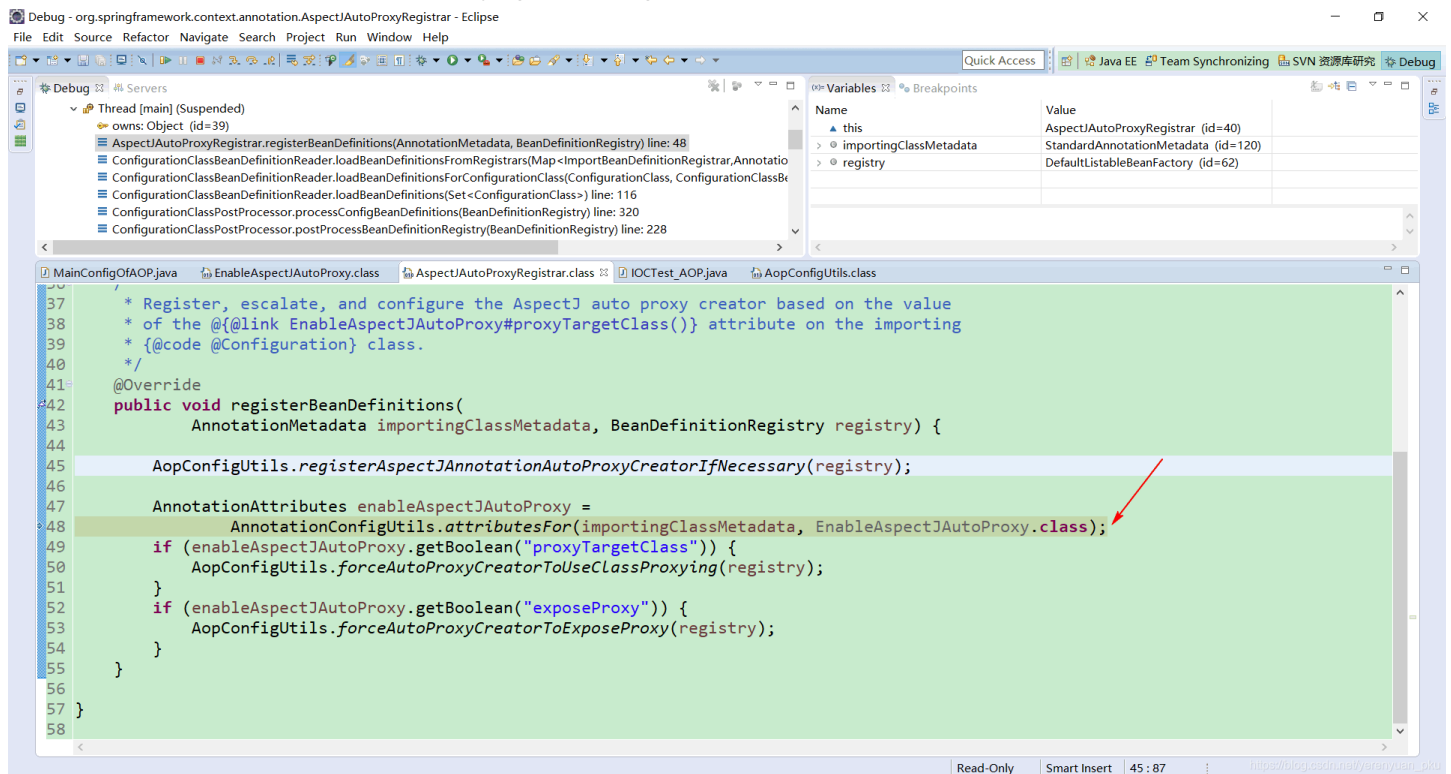
这儿，会使用RootBeanDefinition来创建一个bean的定义信息（即beanDefinition），并且将org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator的Class对象作为参数传递进来。



我们继续往下看代码，最终会回到AspectJAutoProxyRegistrar类的registerBeanDefinitions()方法中。



接下来，我们继续往下看代码，即查看AspectJAutoProxyRegistrar类中的registerBeanDefinitions()方法的源码，如下所示。

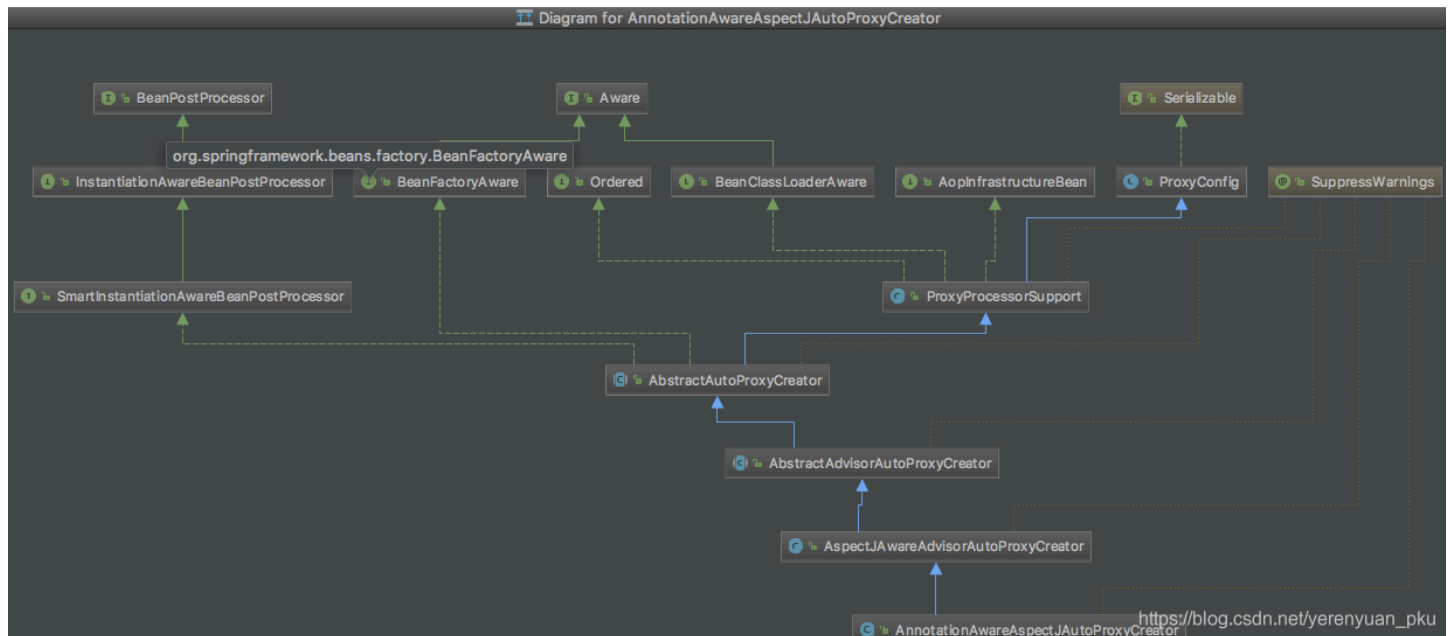


可以看到，通过AnnotationConfigUtils类的attributesFor()方法来获取@EnableAspectJAutoProxy注解的信息。接着，就是判断proxyTargetClass属性的值是否为true，若为true则调用AopConfigUtils类的forceAutoProxyCreatorToUseClassProxying()方法；继续判断exposeProxy属性的值是否为true，若为true则调用AopConfigUtils类的forceAutoProxyCreatorToExposeProxy()方法，其实就是暴露一些代理的这些bean，这个以后我们可以再说。

综上，向Spring的配置类上添加@EnableAspectJAutoProxy注解之后，会向IOC容器中注册AnnotationAwareAspectJAutoProxyCreator，翻译过来就叫注解装配模式的AspectJ切面自动代理创建器。

这个AnnotationAwareAspectJAutoProxyCreator又是什么呢？别急，后面我们会核心研究它，现在我们只要知道在容器中注册了这样一个AnnotationAwareAspectJAutoProxyCreator组件就行了，至于注册的这个组件有什么功能呢？后面我们会继续研究它！其实，只要我们研究出来了，那么这个AOP的原理我们就彻底地知道了。

在研究它之前，我们来看下AnnotationAwareAspectJAutoProxyCreator类的结构图。



可以看到，它继承了很多很多东东，我们简单梳理下`AnnotationAwareAspectJAutoProxyCreator`类的核心继承关系，如下所示。

```

1 | AnnotationAwareAspectJAutoProxyCreator
2 |     -> AspectJAwareAdvisorAutoProxyCreator (父类)
3 |     -> AbstractAdvisorAutoProxyCreator (父类)
4 |     -> AbstractAutoProxyCreator (父类)
5 |     implements SmartInstantiationAwareBeanPostProcessor, BeanFactoryAware (两个接口)
    |
    | AI写代码java运行
  
```

查看继承关系可以发现，此类实现了`Aware`与`BeanPostProcessor`接口，这两个接口都和Spring bean的初始化有关，由此可以推测此类的主要处理方法都来自于这两个接口中的实现方法。

好了，有关`AnnotationAwareAspectJAutoProxyCreator`类的详细代码和执行流程我们后面再讲，估计有些小伙伴有点消化不了了。