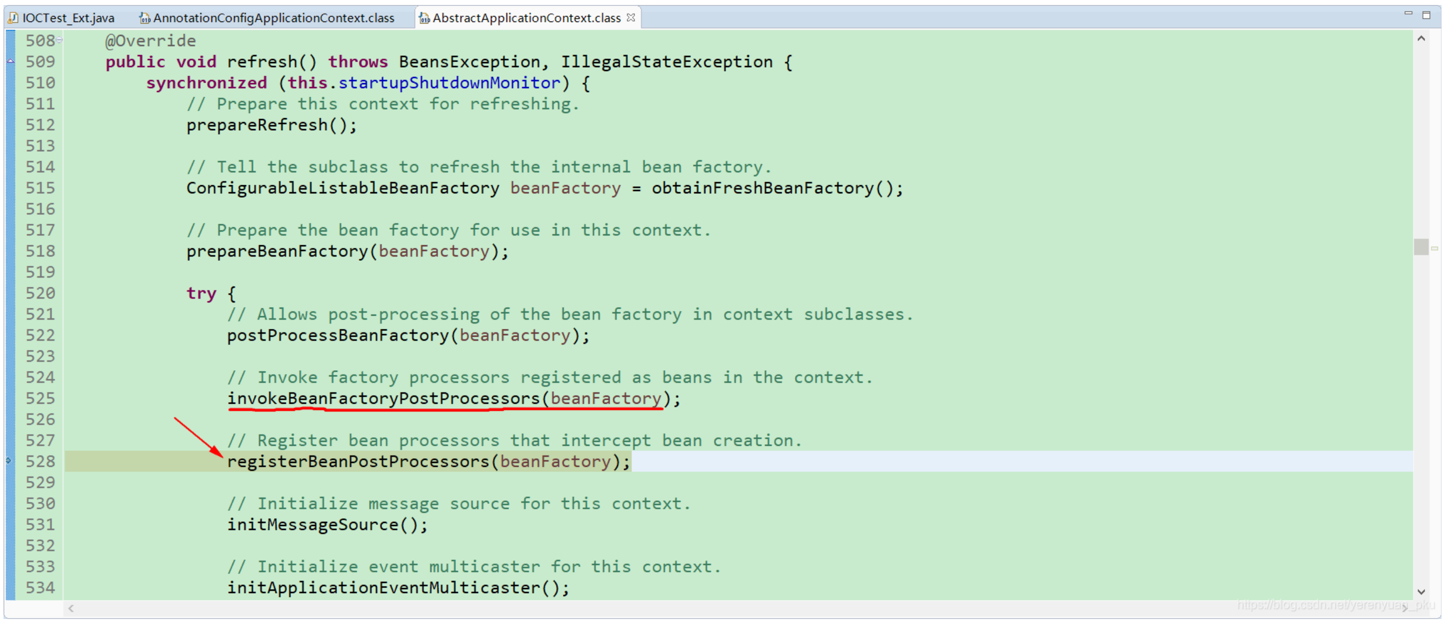
Spring注解驱动开发第43讲——Spring IOC容器创建源码解析(三)之注册BeanPostProcessor

文章目录

写在前面

在上一讲中，我们让程序停留在了下面这行代码处。



IOCTest_Ext.java | AnnotationConfigApplicationContext.class | AbstractApplicationContext.class

```
	@Override
	public void refresh() throws BeansException, IllegalStateException {
		synchronized (this.startupShutdownMonitor) {
			// Prepare this context for refreshing.
			prepareRefresh();

			// Tell the subclass to refresh the internal bean factory.
			ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

			// Prepare the bean factory for use in this context.
			prepareBeanFactory(beanFactory);

			try {
				// Allows post-processing of the bean factory in context subclasses.
				postProcessBeanFactory(beanFactory);

				// Invoke factory processors registered as beans in the context.
				invokeBeanFactoryPostProcessors(beanFactory);

				// Register bean processors that intercept bean creation.
				registerBeanPostProcessors(beanFactory);

				// Initialize message source for this context.
				initMessageSource();

				// Initialize event multicaster for this context.
				initApplicationEventMulticaster();
```

我们刚好讲完以上invokeBeanFactoryPostProcessors方法，该方法所做的事情无非就是在BeanFactory准备好以后，执行BeanFactoryPostProcessor的方法。

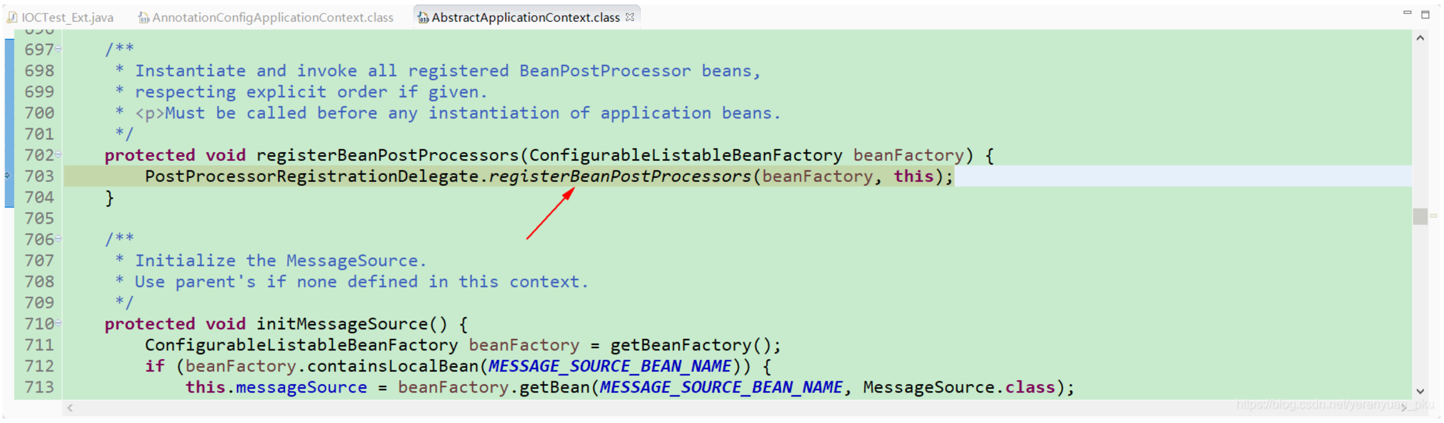
接下来，我们就得来说道说道registerBeanPostProcessors方法了。顾名思义，该方法就是来注册BeanPostProcessor的，即注册bean的后置处理器。其实，从该方法上的描述上，我们也能知道其作用就是注册bean的后置处理器，拦截bean的创建过程。

其实，我们之前在深扒 AOP 的原理时，就已经debug跟踪过该方法了。我说得更具体点，在创建AOP的核心类时，就是调用这个方法来进行处理的。不记得的同学，可以翻阅我之前写的文章哟😀！

注册BeanPostProcessor

获取所有 的BeanPostProcessor

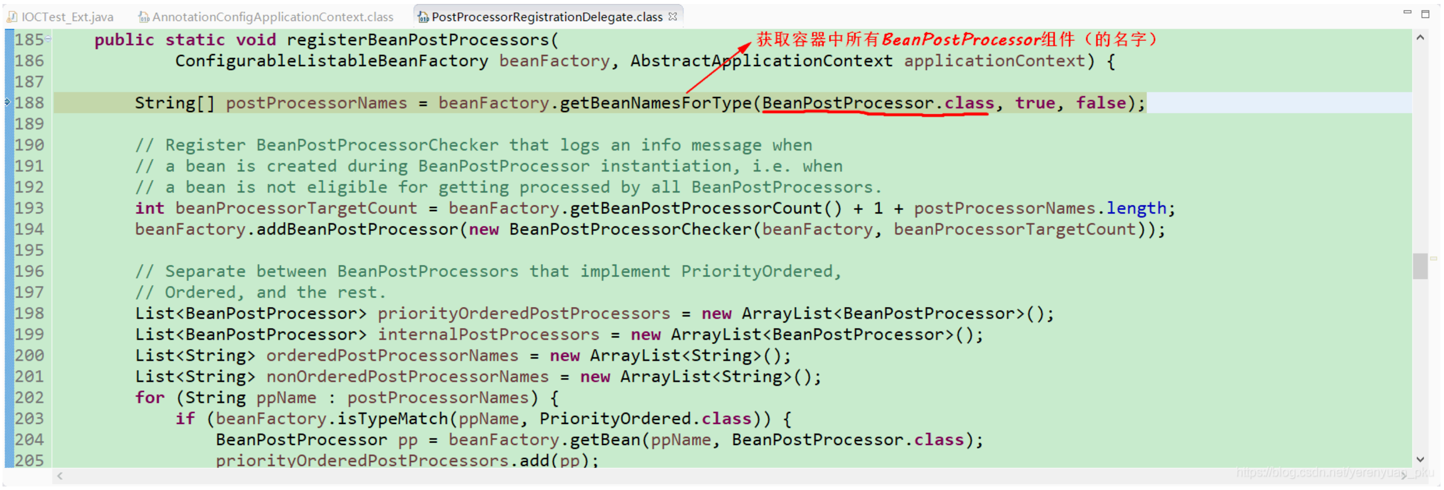
按下 F5 快捷键进入registerBeanPostProcessors方法里面，如下图所示，可以看到在该方法里面会调用PostProcessorRegistrationDelegate类的registerBeanPostProcessors方法。



IOCTest_Ext.java | AnnotationConfigApplicationContext.class | AbstractApplicationContext.class

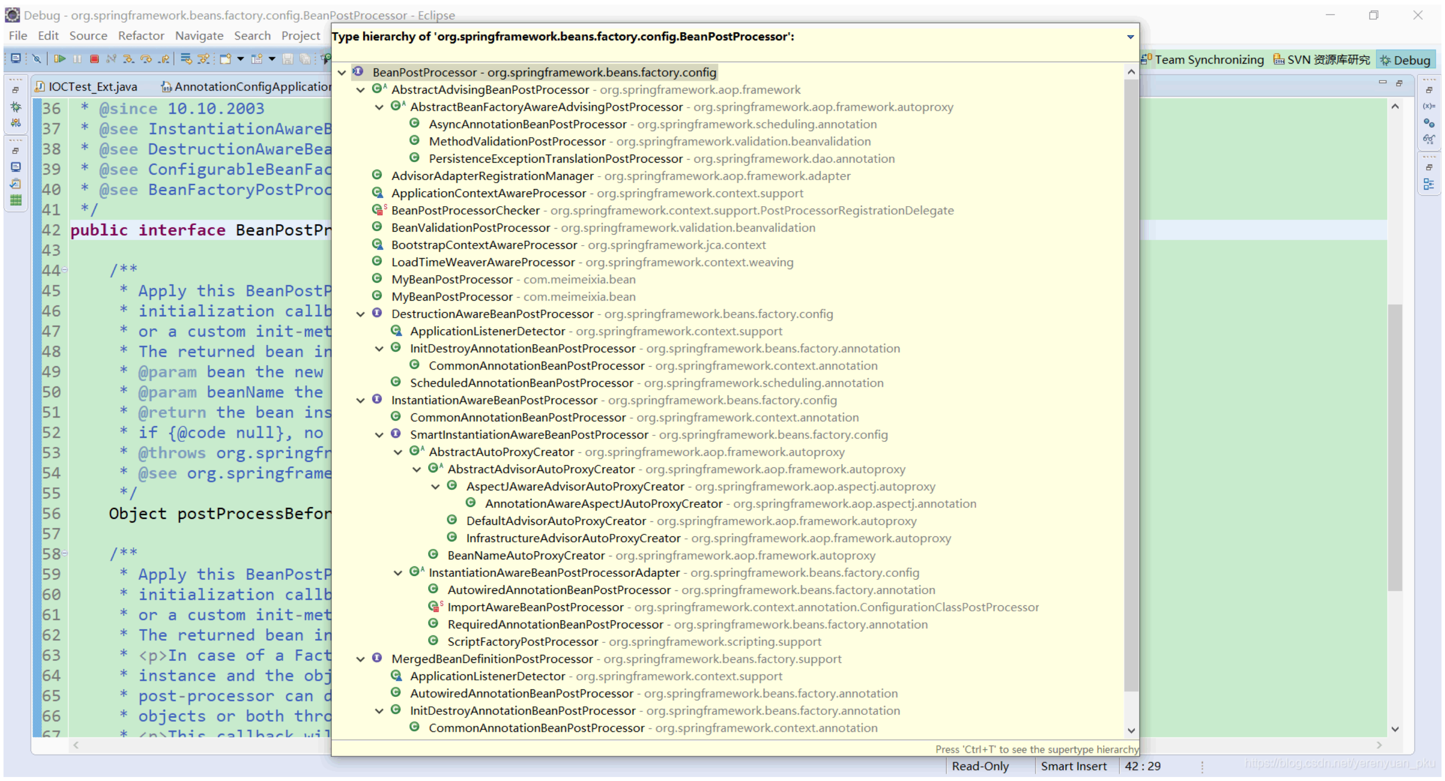
```
	/**
	 * Instantiate and invoke all registered BeanPostProcessor beans,
	 * respecting explicit order if given.
	 * <p>Must be called before any instantiation of application beans.
	 */
	protected void registerBeanPostProcessors(ConfigurableListableBeanFactory beanFactory) {
		PostProcessorRegistrationDelegate.registerBeanPostProcessors(beanFactory, this);
	}

	/**
	 * Initialize the MessageSource.
	 * Use parent's if none defined in this context.
	 */
	protected void initMessageSource() {
		ConfigurableListableBeanFactory beanFactory = getBeanFactory();
		if (beanFactory.containsLocalBean(MESSAGE_SOURCE_BEAN_NAME)) {
			this.messageSource = beanFactory.getBean(MESSAGE_SOURCE_BEAN_NAME, MessageSource.class);
```

于是，我们再次按下 F5 快捷键进入以上方法中，如下图所示，可以看到一开始就会获取所有BeanPostProcessor组件的名字。


IOCTest_Ext.java
AnnotationConfigApplicationContext.class
PostProcessorRegistrationDelegate.class
获取容器中所有BeanPostProcessor组件（的名字）

```java
    public static void registerBeanPostProcessors(
            ConfigurableListableBeanFactory beanFactory, AbstractApplicationContext applicationContext) {

        String[] postProcessorNames = beanFactory.getBeanNamesForType(BeanPostProcessor.class, true, false);

        // Register BeanPostProcessorChecker that logs an info message when
        // a bean is created during BeanPostProcessor instantiation, i.e. when
        // a bean is not eligible for getting processed by all BeanPostProcessors.
        int beanProcessorTargetCount = beanFactory.getBeanPostProcessorCount() + 1 + postProcessorNames.length;
        beanFactory.addBeanPostProcessor(new BeanPostProcessorChecker(beanFactory, beanProcessorTargetCount));

        // Separate between BeanPostProcessors that implement PriorityOrdered,
        // Ordered, and the rest.
        List<BeanPostProcessor> priorityOrderedPostProcessors = new ArrayList<BeanPostProcessor>();
        List<BeanPostProcessor> internalPostProcessors = new ArrayList<BeanPostProcessor>();
        List<String> orderedPostProcessorNames = new ArrayList<String>();
        List<String> nonOrderedPostProcessorNames = new ArrayList<String>();
        for (String ppName : postProcessorNames) {
            if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
                BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
                priorityOrderedPostProcessors.add(pp);
```

这里，我得提醒大家的一点是BeanPostProcessor接口旗下有非常多的子接口，这一点你查看一下BeanPostProcessor接口的继承树就知道了，如下图所示。


Debug - org.springframework.beans.factory.config.BeanPostProcessor - Eclipse
File Edit Source Refactor Navigate Search Project
Team Synchronizing
SVN 资源库研究
Debug
IOCTest_Ext.java
AnnotationConfigApplicatio
Read-Only
Smart Insert
42 : 29

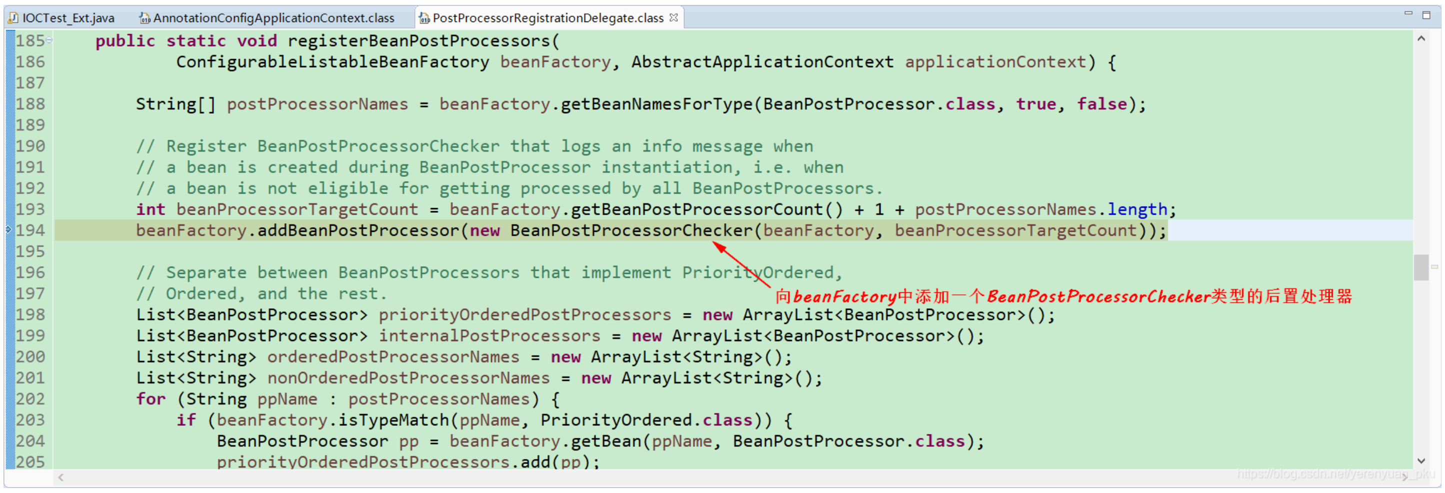
```java
 * @since 10.10.2003
 * @see InstantiationAwareB
 * @see DestructionAwareBea
 * @see ConfigurableBeanFac
 * @see BeanFactoryPostProc
 */
public interface BeanPostPr

    /**
     * Apply this BeanPostP
     * initialization callb
     * or a custom init-met
     * The returned bean in
     * @param bean the new
     * @param beanName the
     * @return the bean ins
     * if {@code null}, no
     * @throws org.springfr
     * @see org.springframe
     */
    Object postProcessBefor

    /**
     * Apply this BeanPostP
     * initialization callb
     * or a custom init-met
     * The returned bean in
     * <p>In case of a Fact
     * instance and the obj
     * post-processor can d
     * objects or both thro
```

```
Type hierarchy of 'org.springframework.beans.factory.config.BeanPostProcessor':

BeanPostProcessor - org.springframework.beans.factory.config
  AbstractAdvisingBeanPostProcessor - org.springframework.aop.framework
    AbstractBeanFactoryAwareAdvisingPostProcessor - org.springframework.aop.framework.autoproxy
      AsyncAnnotationBeanPostProcessor - org.springframework.scheduling.annotation
      MethodValidationPostProcessor - org.springframework.validation.beanvalidation
      PersistenceExceptionTranslationPostProcessor - org.springframework.dao.annotation
  AdvisorAdapterRegistrationManager - org.springframework.aop.framework.adapter
  ApplicationContextAwareProcessor - org.springframework.context.support
  BeanPostProcessorChecker - org.springframework.context.support.PostProcessorRegistrationDelegate
  BeanValidationPostProcessor - org.springframework.validation.beanvalidation
  BootstrapContextAwareProcessor - org.springframework.jca.context
  LoadTimeWeaverAwareProcessor - org.springframework.context.weaving
  MyBeanPostProcessor - com.meimeixia.bean
  MyBeanPostProcessor - com.meimeixia.bean
  DestructionAwareBeanPostProcessor - org.springframework.beans.factory.config
    ApplicationListenerDetector - org.springframework.context.support
    InitDestroyAnnotationBeanPostProcessor - org.springframework.beans.factory.annotation
      CommonAnnotationBeanPostProcessor - org.springframework.context.annotation
    ScheduledAnnotationBeanPostProcessor - org.springframework.scheduling.annotation
  InstantiationAwareBeanPostProcessor - org.springframework.beans.factory.config
    CommonAnnotationBeanPostProcessor - org.springframework.context.annotation
    SmartInstantiationAwareBeanPostProcessor - org.springframework.beans.factory.config
      AbstractAutoProxyCreator - org.springframework.aop.framework.autoproxy
        AbstractAdvisorAutoProxyCreator - org.springframework.aop.framework.autoproxy
          AspectJAwareAdvisorAutoProxyCreator - org.springframework.aop.aspectj.autoproxy
            AnnotationAwareAspectJAutoProxyCreator - org.springframework.aop.aspectj.annotation
          DefaultAdvisorAutoProxyCreator - org.springframework.aop.framework.autoproxy
          InfrastructureAdvisorAutoProxyCreator - org.springframework.aop.framework.autoproxy
        BeanNameAutoProxyCreator - org.springframework.aop.framework.autoproxy
      InstantiationAwareBeanPostProcessorAdapter - org.springframework.beans.factory.config
        AutowiredAnnotationBeanPostProcessor - org.springframework.beans.factory.annotation
        ImportAwareBeanPostProcessor - org.springframework.context.annotation.ConfigurationClassPostProcessor
        RequiredAnnotationBeanPostProcessor - org.springframework.beans.factory.annotation
        ScriptFactoryPostProcessor - org.springframework.scripting.support
  MergedBeanDefinitionPostProcessor - org.springframework.beans.factory.support
    ApplicationListenerDetector - org.springframework.context.support
    AutowiredAnnotationBeanPostProcessor - org.springframework.beans.factory.annotation
    InitDestroyAnnotationBeanPostProcessor - org.springframework.beans.factory.annotation
      CommonAnnotationBeanPostProcessor - org.springframework.context.annotation

Press 'Ctrl+T' to see the supertype hierarchy
```

看到了吗，BeanPostProcessor接口旗下是不是有很多子接口啊，而且每一个子接口，还有点不一样。这里，我也只会挑出如下的几个子接口将其罗列出来，目的是为了告诉大家BeanPostProcessor接口旗下确实是有非常多的子接口，而且这些不同 接口类型 的BeanPostProcessor在bean创建前后的执行时机是不一样的，虽然它们都是后置处理器。

- DestructionAwareBeanPostProcessor：该接口我们之前是不是说过啊？它是销毁bean的后置处理器
- InstantiationAwareBeanPostProcessor
- SmartInstantiationAwareBeanPostProcessor
- MergedBeanDefinitionPostProcessor

获取到所有的BeanPostProcessor组件之后，我们按下 F6 快捷键让程序往下运行，直至程序运行到下面这行代码处，可以看到现在向beanFactory中添加了一个BeanPostProcessorChecker类型的后置处理器，它是来检查所有BeanPostProcessor组件的。



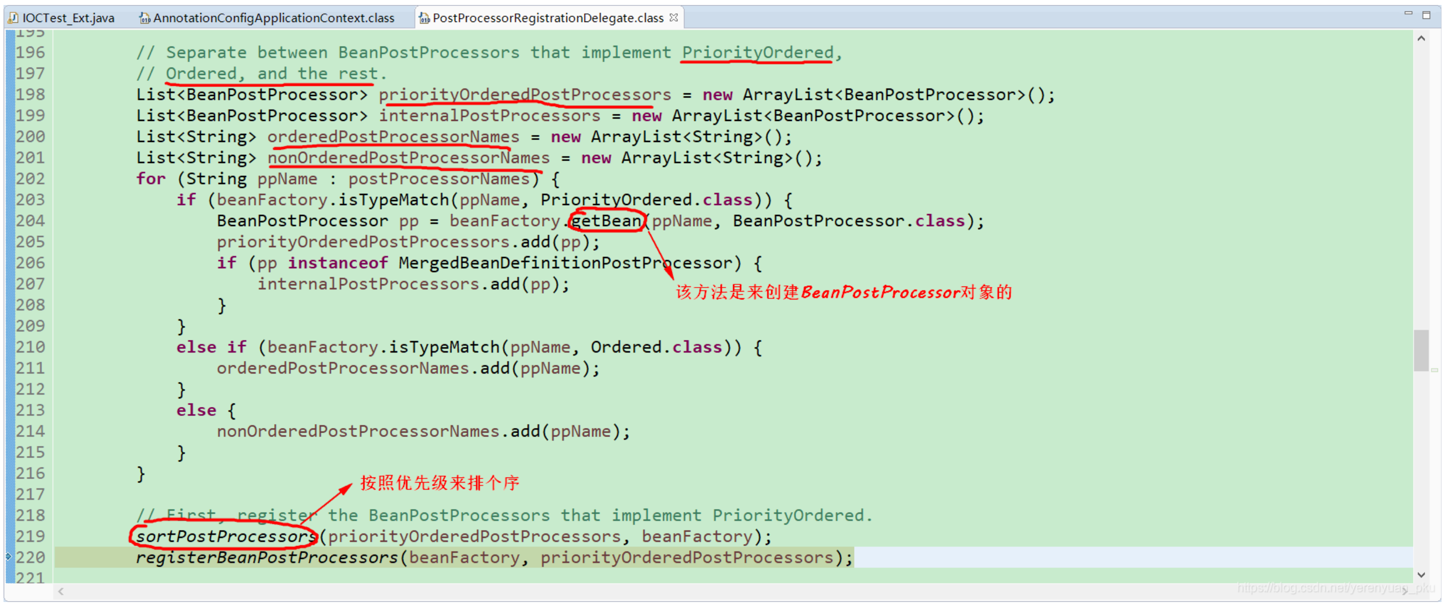
IOCTest_Ext.java AnnotationConfigApplicationContext.class PostProcessorRegistrationDelegate.class

```java
    public static void registerBeanPostProcessors(
            ConfigurableListableBeanFactory beanFactory, AbstractApplicationContext applicationContext) {

        String[] postProcessorNames = beanFactory.getBeanNamesForType(BeanPostProcessor.class, true, false);

        // Register BeanPostProcessorChecker that logs an info message when
        // a bean is created during BeanPostProcessor instantiation, i.e. when
        // a bean is not eligible for getting processed by all BeanPostProcessors.
        int beanProcessorTargetCount = beanFactory.getBeanPostProcessorCount() + 1 + postProcessorNames.length;
        beanFactory.addBeanPostProcessor(new BeanPostProcessorChecker(beanFactory, beanProcessorTargetCount));

        // Separate between BeanPostProcessors that implement PriorityOrdered,
        // Ordered, and the rest.
        List<BeanPostProcessor> priorityOrderedPostProcessors = new ArrayList<BeanPostProcessor>();
        List<BeanPostProcessor> internalPostProcessors = new ArrayList<BeanPostProcessor>();
        List<String> orderedPostProcessorNames = new ArrayList<String>();
        List<String> nonOrderedPostProcessorNames = new ArrayList<String>();
        for (String ppName : postProcessorNames) {
            if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
                BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
                priorityOrderedPostProcessors.add(pp);
```

向beanFactory中添加一个BeanPostProcessorChecker类型的后置处理器

按分好类的优先级顺序来注册BeanPostProcessor

继续按下 F6 快捷键让程序往下运行，在这一过程中，可以看到后置处理器也可以按照是否实现了PriorityOrdered接口、Ordered接口以及没有实现这两个接口这三种情况进行分类。



IOCTest_Ext.java AnnotationConfigApplicationContext.class PostProcessorRegistrationDelegate.class

```java
        // Separate between BeanPostProcessors that implement PriorityOrdered,
        // Ordered, and the rest.
        List<BeanPostProcessor> priorityOrderedPostProcessors = new ArrayList<BeanPostProcessor>();
        List<BeanPostProcessor> internalPostProcessors = new ArrayList<BeanPostProcessor>();
        List<String> orderedPostProcessorNames = new ArrayList<String>();
        List<String> nonOrderedPostProcessorNames = new ArrayList<String>();
        for (String ppName : postProcessorNames) {
            if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
                BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
                priorityOrderedPostProcessors.add(pp);
                if (pp instanceof MergedBeanDefinitionPostProcessor) {
                    internalPostProcessors.add(pp);
                }
            }
            else if (beanFactory.isTypeMatch(ppName, Ordered.class)) {
                orderedPostProcessorNames.add(ppName);
            }
            else {
                nonOrderedPostProcessorNames.add(ppName);
            }
        }

        // First, register the BeanPostProcessors that implement PriorityOrdered.
        sortPostProcessors(priorityOrderedPostProcessors, beanFactory);
        registerBeanPostProcessors(beanFactory, priorityOrderedPostProcessors);
```

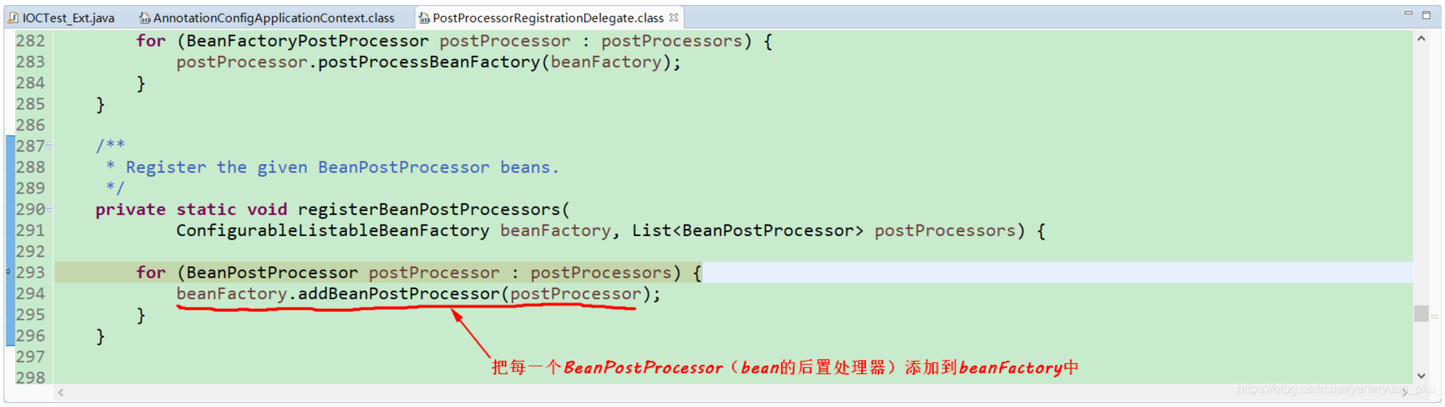
将所有的BeanPostProcessor组件分门别类之后，依次存储在不同的ArrayList集合中。

其实，我们会发现不止有三个ArrayList集合，还有一个名字为internalPostProcessors的ArrayList集合。如果后置处理器是MergedBeanDefinitionPostProcessor这种类型的，那么它就会被存放在名字为internalPostProcessors的ArrayList集合中。

由于BeanPostProcessor还是挺多的（除了IOC容器自己拥有的以外，还有咱们自己编写的），因此你得不停地按下 F6 快捷键让程序往下运行，直至程序运行到第220行代码处。

当程序运行到第220行代码处时，可以看到这是来注册实现了PriorityOrdered优先级接口的BeanPostProcessor的。因为这儿调用了一个叫registerBeanPostProcessors的方法，该方法就是来注册bean的后置处理器的，而所谓的注册就是向beanFactory中添加进去这些BeanPostProcessor。

我为何会这么说呢？按下 F5 快捷键进入到registerBeanPostProcessors方法中，你就一目了然了，勿须我再多说。

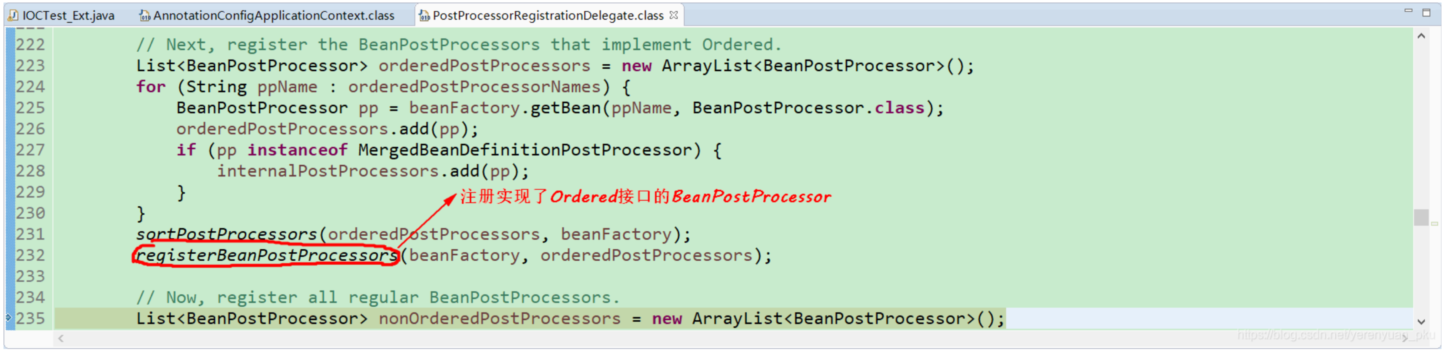


IOCTest_Ext.java AnnotationConfigApplicationContext.class PostProcessorRegistrationDelegate.class

```java
        for (BeanFactoryPostProcessor postProcessor : postProcessors) {
            postProcessor.postProcessBeanFactory(beanFactory);
        }
    }

    /**
     * Register the given BeanPostProcessor beans.
     */
    private static void registerBeanPostProcessors(
            ConfigurableListableBeanFactory beanFactory, List<BeanPostProcessor> postProcessors) {

        for (BeanPostProcessor postProcessor : postProcessors) {
            beanFactory.addBeanPostProcessor(postProcessor);
        }
    }
```

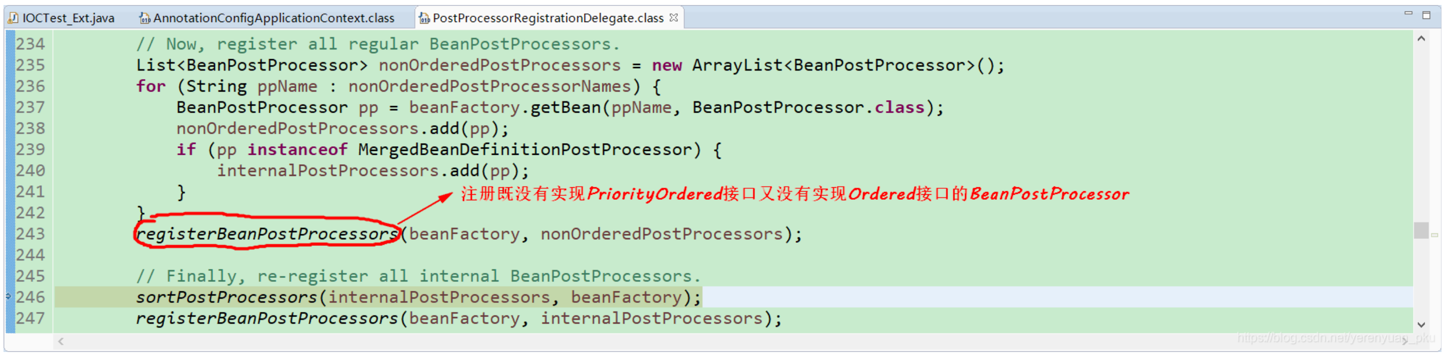
把每一个BeanPostProcessor（bean的后置处理器）添加到beanFactory中

然后，注册实现了Ordered接口的BeanPostProcessor，如下图所示。


IOCTest_Ext.java
AnnotationConfigApplicationContext.class
PostProcessorRegistrationDelegate.class
注册实现了Ordered接口的BeanPostProcessor

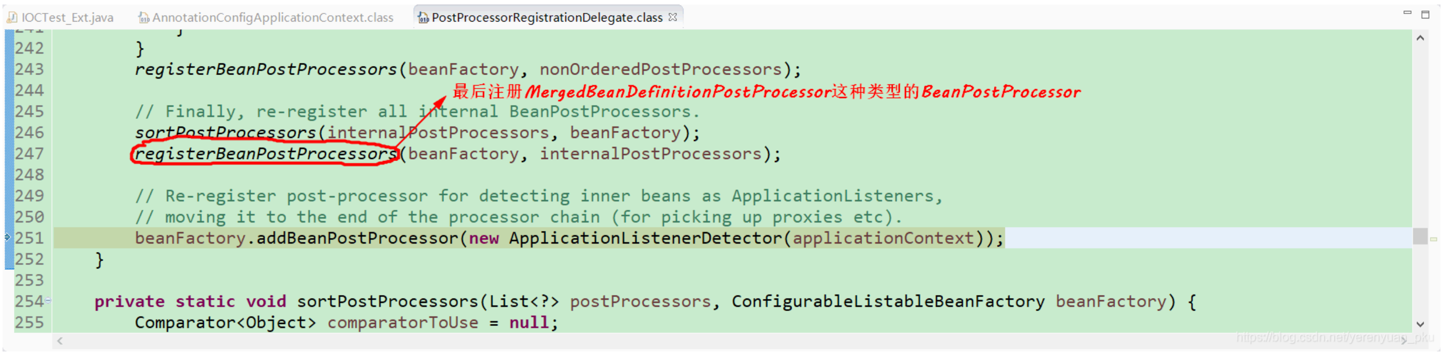
```java
        // Next, register the BeanPostProcessors that implement Ordered.
        List<BeanPostProcessor> orderedPostProcessors = new ArrayList<BeanPostProcessor>();
        for (String ppName : orderedPostProcessorNames) {
            BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
            orderedPostProcessors.add(pp);
            if (pp instanceof MergedBeanDefinitionPostProcessor) {
                internalPostProcessors.add(pp);
            }
        }
        sortPostProcessors(orderedPostProcessors, beanFactory);
        registerBeanPostProcessors(beanFactory, orderedPostProcessors);

        // Now, register all regular BeanPostProcessors.
        List<BeanPostProcessor> nonOrderedPostProcessors = new ArrayList<BeanPostProcessor>();
```

接着，再来注册既没有实现PriorityOrdered接口又没有实现Ordered接口的BeanPostProcessor，如下图所示。


IOCTest_Ext.java
AnnotationConfigApplicationContext.class
PostProcessorRegistrationDelegate.class
注册既没有实现PriorityOrdered接口又没有实现Ordered接口的BeanPostProcessor

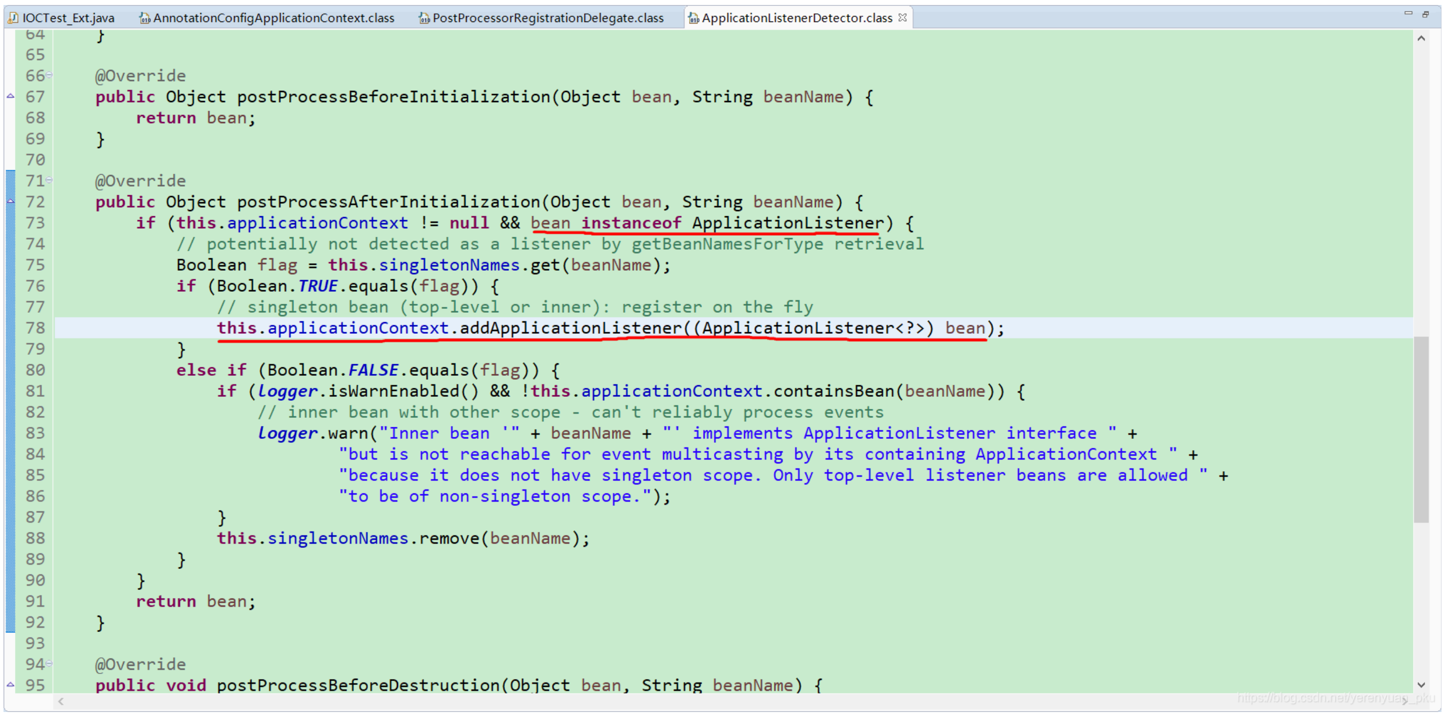
```java
        // Now, register all regular BeanPostProcessors.
        List<BeanPostProcessor> nonOrderedPostProcessors = new ArrayList<BeanPostProcessor>();
        for (String ppName : nonOrderedPostProcessorNames) {
            BeanPostProcessor pp = beanFactory.getBean(ppName, BeanPostProcessor.class);
            nonOrderedPostProcessors.add(pp);
            if (pp instanceof MergedBeanDefinitionPostProcessor) {
                internalPostProcessors.add(pp);
            }
        }
        registerBeanPostProcessors(beanFactory, nonOrderedPostProcessors);

        // Finally, re-register all internal BeanPostProcessors.
        sortPostProcessors(internalPostProcessors, beanFactory);
        registerBeanPostProcessors(beanFactory, internalPostProcessors);
```

最后，再来注册MergedBeanDefinitionPostProcessor这种类型的BeanPostProcessor，因为名字为internalPostProcessors的ArrayList集合中存放的就是这种类型的BeanPostProcessor。


IOCTest_Ext.java
AnnotationConfigApplicationContext.class
PostProcessorRegistrationDelegate.class
最后注册MergedBeanDefinitionPostProcessor这种类型的BeanPostProcessor

```java
        }
        registerBeanPostProcessors(beanFactory, nonOrderedPostProcessors);

        // Finally, re-register all internal BeanPostProcessors.
        sortPostProcessors(internalPostProcessors, beanFactory);
        registerBeanPostProcessors(beanFactory, internalPostProcessors);

        // Re-register post-processor for detecting inner beans as ApplicationListeners,
        // moving it to the end of the processor chain (for picking up proxies etc).
        beanFactory.addBeanPostProcessor(new ApplicationListenerDetector(applicationContext));
    }

    private static void sortPostProcessors(List<?> postProcessors, ConfigurableListableBeanFactory beanFactory) {
        Comparator<Object> comparatorToUse = null;
```

除此之外，还会向beanFactory中添加一个ApplicationListenerDetector类型的BeanPostProcessor。我们不妨点进ApplicationListenerDetector类里面去看一看，如下图所示，它里面有一个postProcessAfterInitialization方法，该方法是在bean创建初始化之后，探测该bean是不是ApplicationListener的。


IOCTest_Ext.java
AnnotationConfigApplicationContext.class
PostProcessorRegistrationDelegate.class
ApplicationListenerDetector.class

```java
    }

    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName) {
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName) {
        if (this.applicationContext != null && bean instanceof ApplicationListener) {
            // potentially not detected as a listener by getBeanNamesForType retrieval
            Boolean flag = this.singletonNames.get(beanName);
            if (Boolean.TRUE.equals(flag)) {
                // singleton bean (top-level or inner): register on the fly
                this.applicationContext.addApplicationListener((ApplicationListener<?>) bean);
            }
            else if (Boolean.FALSE.equals(flag)) {
                if (logger.isWarnEnabled() && !this.applicationContext.containsBean(beanName)) {
                    // inner bean with other scope - can't reliably process events
                    logger.warn("Inner bean '" + beanName + "' implements ApplicationListener interface " +
                            "but is not reachable for event multicasting by its containing ApplicationContext " +
                            "because it does not have singleton scope. Only top-level listener beans are allowed " +
                            "to be of non-singleton scope.");
                }
                this.singletonNames.remove(beanName);
            }
        }
        return bean;
    }

    @Override
    public void postProcessBeforeDestruction(Object bean, String beanName) {
```

也就是说，该方法的作用是检查哪些bean是监听器的。如果是，那么会将该bean放在容器中保存起来。

最最后，我得多提一嘴，以上只是来注册bean的后置处理器，即只是向beanFactory中添加了所有这些bean的后置处理器，而并不会执行它们。