# Spring注解驱动开发第42讲——Spring IOC容器创建源码解析(二)之执行 BeanFactoryPostProcessor

#### 文章目录

写在前面

BeanFactoryPostProcessor的执行过程

先执行BeanDefinitionRegistryPostProcessor的方法

根据优先级,分别执行BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

执行实现了PriorityOrdered优先级接口的BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

执行实现了Ordered顺序接口的BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

执行没有实现任何优先级或者是顺序接口的BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

执行BeanDefinitionRegistryPostProcessor的postProcessBeanFactory方法

再执行BeanFactoryPostProcessor的方法

小结

### 写在前面

在上一讲中,我们详细地分析了一下BeanFactory的创建以及预准备工作的流程。紧接上一讲,我们就要来看看接下来又做了哪些工作。

现在,程序已经运行到了下面这行代码处了。

```
🛿 IOCTest_Ext.java 🔝 AnnotationConfigApplicationContext.class 🛣 AbstractApplicationContext.class 🗵
 516
 517
                   // Prepare the bean factory for use in this context.
 518
                   prepareBeanFactory(beanFactory);
 519
 520
                   try {
    // Allows post-processing of the bean factory in context subclasses.
 521
 522
                       postProcessBeanFactory(beanFactory);
 523
                        // Invoke factory processors registered as beans in the context.
 524
                       invokeBeanFactoryPostProcessors(beanFactory);
 525
 526
 527
                        // Register bean processors that intercept bean creation.
 528
                        registerBeanPostProcessors(beanFactory);
 529
                        // Initialize message source for this context.
 530
 531
                       initMessageSource();
 532
 533
                        // Initialize event multicaster for this context.
 534
                       initApplicationEventMulticaster();
 535
```

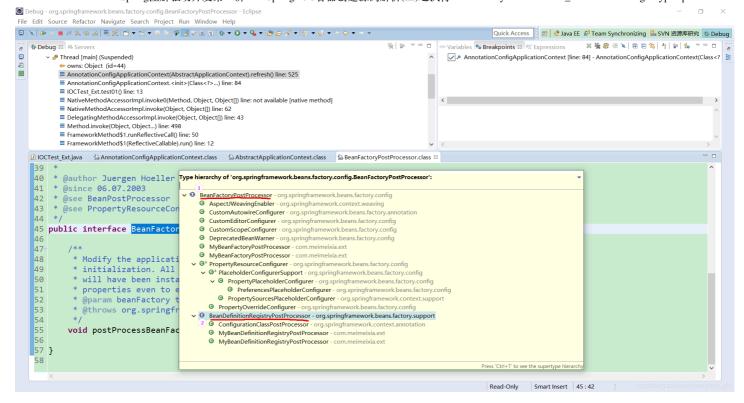
可以看到这儿会执行一个叫invokeBeanFactoryPostProcessors的方法,这个方法我们之前也看过,它就是来执行BeanFactoryPostProcessor的。而这个BeanFactoryPostProcessor,我们之前也介绍过,如果你要是忘记了,那么可以回顾回顾我写的《Spring注解驱动开发第36讲——或许,这是你以前没看过的从源码角度理解BeanFactoryPostProcessor的原理》这篇文章。

现在,你是不是想起来了,它就是BeanFactory的后置处理器。那么,它是什么时候来执行的呢?我们不妨看一下它的源码,如下图所示。

```
🖟 IOCTest_Ext.java 🖟 AnnotationConfigApplicationContext.class 🚵 AbstractApplicationContext.class 🛣 BeanFactoryPostProcessor.class 🗯
     * @author Juergen Hoeller
       @since 06.07.2003
    * @see BeanPostProcessor
       @see PropertyResourceConfigurer
                                                           描述说, BeanFactoryPostProcessor(的方法)是在BeanFactory标准初始化之后执行的。而
                                                           BeanFactory标准初始化就是我们上
                                                                                            一讲所阐述的内容
45 public interface BeanFactoryPostProcessor {
46
47
48
49
50
          * Modify the application context's internal bean factory
            initialization. All bean definitions will have been loaded, but no beans will have been instantiated yet. This allows for overriding or adding
            properties even to eager-initializing beans
            @param beanFactory the bean factory used by the application context
            @throws org.springframework.beans.BeansException in case of errors
        void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException;
57 }
```

从它内部方法的描述上来看,BeanFactoryPostProcessor(也可以说它里面的那个方法)是在BeanFactory标准 初始化 之后执行的。而BeanFactory标准初始化正是我们上一讲所阐述的内容。

我们之前也看过BeanFactoryPostProcessor接口的继承树,如下图所示。



可以看到,BeanFactoryPostProcessor接口下还有一个子接口,即BeanDefinitionRegistryPostProcessor。以前,我们还用过BeanDefinitionRegistryPostProcessor这个接口给IOC容器中额外添加过组件,不知你还记不记得?

接下来,我们就来看看invokeBeanFactoryPostProcessors这个方法里面到底做了哪些事,也就是看一下BeanFactoryPostProcessor的整个执行过程。

## BeanFactoryPostProcessor的执行过程

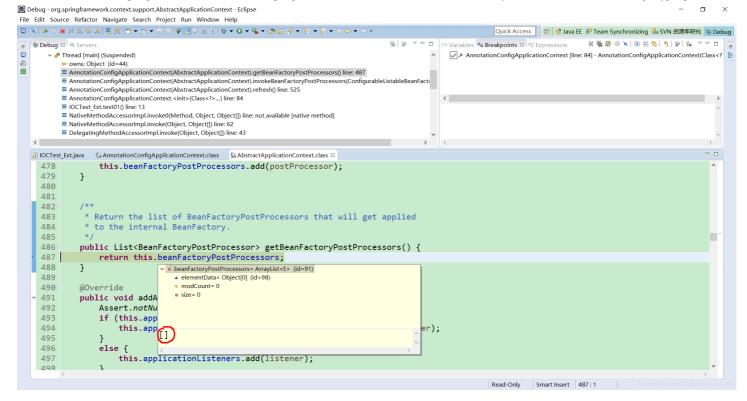
其实,当你看完这篇文章之后,你就知道了在invokeBeanFactoryPostProcessors方法里面主要就是执行了BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry和postProcessBeanFactory这俩方法,以及BeanFactoryPostProcessors的postProcessBeanFactory方法。

### 先执行BeanDefinitionRegistryPostProcessor的方法

我们可以按下 F5 快捷键进入invokeBeanFactoryPostProcessors方法里面去瞧一瞧、如下图所示,可以看到现在程序来到了如下这行代码处。

```
☑ IOCTest_Ext.java  
☐ AnnotationConfigApplicationContext.class  
☐ AbstractApplicationContext.class 
☐ AnnotationContext.class 
☐ AnnotationContext.cl
         681
                                                       ^{st} Instantiate and invoke all registered BeanFactoryPostProcessor beans,
         682
                                                                respecting explicit order if given.
         683
                                                                 Must be called before singleton instantiation.
         684
         685
         686
                                                 \label{protected} \textbf{protected void} \ \ \textbf{invokeBeanFactoryPostProcessors} \\ \textbf{(ConfigurableListableBeanFactory beanFactory)} \ \ \{\textbf{(Softward of the protected void invokeBeanFactory beanFactory)} \} \\ \textbf{(Softward of the protected void invokeBeanFactory beanFactory)} \\ \textbf{(Softward of the protected void invokeBeanFactory)} \\ \textbf{(Softward of the protected void invok
                                                                      PostProcessor Registration Delegate. {\it invokeBeanFactoryPostProcessors} (bean Factory, \ getBean FactoryPostProcessors()); \\
         687
         688
         689
                                                                      // Detect a LoadTimeWeaver and prepare for weaving, if found in the meantime
                                                                                                                   through an @Bean method registered by ConfigurationClassPostProcessor
         691
                                                                       \textbf{if (beanFactory.getTempClassLoader()} == \textbf{null \& beanFactory.containsBean} \\ (\textbf{\textit{LOAD\_TIME\_WEAVER\_BEAN\_NAME}}) \  \  \{ \textbf{\textit{NAME}} \} 
         692
                                                                                          beanFactory.addBeanPostProcessor(new LoadTimeWeaverAwareProcessor(beanFactory));
         693
                                                                                          bean Factory. set Temp Class Loader (new Context Type Match Class Loader (bean Factory. get Bean Class Loader ())); \\
         694
                                                                    }
         695
                                                 }
         696
         697
                                                      * Instantiate and invoke all registered BeanPostProcessor beans,
         698
                                                                respecting explicit order if given.
         699
```

以上这个invokeBeanFactoryPostProcessors方法,看名字就知道了,同样是来执行BeanFactoryPostProcessor的方法的,那怎么来执行呢?我们可以按下 F5 快捷键来跟 踪源码看看,此时你会发现进入到了getBeanFactoryPostProcessors方法中,如下图所示,该方法仅仅只是返回了一个空的 List<BeanFactoryPostProcessor> 集合,该集合是用于存放所有的BeanFactoryPostProcessor的,只不过它现在默认是空的而已,也就是说该集合里面还没存储任何BeanFactoryPostProcessor。



不过,我们可以通过以下addBeanFactoryPostProcessor方法向该集合中添加BeanFactoryPostProcessor。

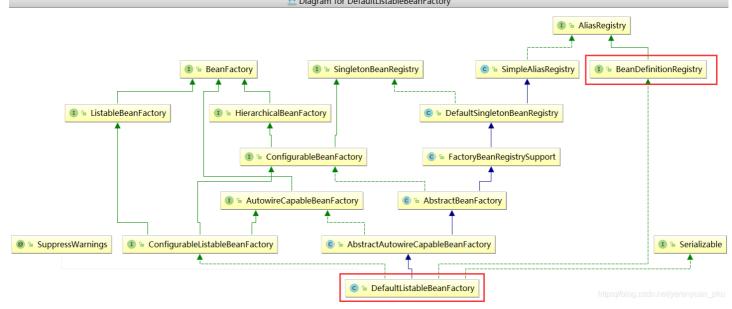
```
onConfigApplicationContext.class AbstractApplicationContext.class &
475
         @Override
         public void addBeanFactoryPostProcessor (BeanFactoryPostProcessor postProcessor) {
476
477
             Assert.notNull(postProcessor, "BeanFactoryPostProcessor must not be null");
478
             this.beanFactoryPostProcessors.add(postProcessor);
479
480
481
482
483
          * Return the list of BeanFactoryPostProcessors that will get applied
484
          * to the internal BeanFactory
485
486
         public List<BeanFactoryPostProcessor> getBeanFactoryPostProcessors() {
487
             return this.beanFactoryPostProcessors;
488
489
490
491
         public void addApplicationListener(ApplicationListener<?> listener) {
492
             Assert.notNull(listener, "ApplicationListener must not be null");
```

于是,按下 F7 快捷键退出getBeanFactoryPostProcessors方法,返回到调用层,然后按下 F5 快捷键进入invokeBeanFactoryPostProcessors方法里面去一探究竟,如下图 所示,是不是来到了我们熟悉的地方啊❤

```
51
 52
        public static void invokeBeanFactoryPostProcessors(
 53
               54
            // Invoke BeanDefinitionRegistryPostProcessors first, if any.
 55
 56
           Set<String> processedBeans = new HashSet<String>();
 57
 58
           if (beanFactory instanceof BeanDefinitionRegistry) {
 59
               BeanDefinitionRegistry registry = (BeanDefinitionRegistry) beanFactory;
               List<BeanFactoryPostProcessor> regularPostProcessors = new LinkedList<BeanFactoryPostProcessor>();
 60
               List<BeanDefinitionRegistryPostProcessor> registryProcessors = new LinkedList<BeanDefinitionRegistryPostProcessor>();
 61
 62
 63
               for (BeanFactoryPostProcessor postProcessor : beanFactoryPostProcessors) {
 64
                  if (postProcessor instanceof BeanDefinitionRegistryPostProcessor) {
                      BeanDefinitionRegistryPostProcessor registryProcessor = (BeanDefinitionRegistryPostProcessor) postProcessor;
 65
 66
 67
                      registryProcessor.postProcessBeanDefinitionRegistry(registry);
 68
                      registryProcessors.add(registryProcessor);
 69
 70
                  else {
                         1] anDoc+Drocessons add/noc+Drocesson).
```

其中,一开始的注释就告诉了我们,无论什么时候都会先调用实现了BeanDefinitionRegistryPostProcessor接口的类。

大家一定要注意哟! 紧接着会先来判断我们这个beanFactory是不是BeanDefinitionRegistry。之前我们在上一讲中就已经说过了,我们生成的BeanFactory对象是 DefaultListableBeanFactory类型的,而且还使用了ConfigurableListableBeanFactory接口进行接收。这里我们就来看下DefaultListableBeanFactory类是不是实现了 BeanDefinitionRegistry接口,看下图,很显然是实现了。



自然地,程序就会进入到if判断语句中,进来以后呢,我们来大致地分析一下下面的流程。首先,映入眼帘的是一个for循环,它是来循环遍历 invokeBeanFactoryPostProcessors方法中的第二个参数的,即beanFactoryPostProcessors。其实呢,就是拿到所有的BeanFactoryPostProcessor,再挨个遍历出来。然后,再来以遍历出来的每一个BeanFactoryPostProcessor是否实现了BeanDefinitionRegistryPostProcessor接口为依据将其分别存放于以下两个箭头所指向的LinkedList中,其中实现了BeanDefinitionRegistryPostProcessor接口的还会被直接调用。

```
52
                            public static void invokeBeanFactoryPostProcessors(
      53
                                                       Configurable \texttt{ListableBeanFactory} \ \ \texttt{L} \underline{\textbf{ist}} \\ \textbf{BeanFactoryPostProcessor} \\ \textbf{beanFactoryPostProcessors}) \ \ \{ \\ \textbf{beanFactoryPostProcessors} \\ \textbf{beanFactoryPostProcessors}
      54
      55
                                            // Invoke BeanDefinitionRegistryPostProcessors first, if any.
      56
                                          Set<String> processedBeans = new HashSet<String>();
      57
      58
                                          if (beanFactory instanceof BeanDefinitionRegistry) {
                                                        BeanDefinitionRegistry registry = (BeanDefinitionRegistry) beanFactory;
      59
      60
                                                        List (Bean Factory Post Processor) regular Post Processors = new Linked List (Bean Factory Post Processor)():
                                                       List<BeanDefinitionRegistryPostProcessor> registryProcessors = new LinkedList<BeanDefinitionRegistryPostProcessor>();
      61
      62
      63
                                                        \textbf{for (BeanFactoryPostProcessor postProcessor : beanFactoryPostProcessors) } \{
                                                                    if (postProcessor instanceof BeanDefinitionRegistryPostProcessor) {
                                                                                 BeanDefinitionRegistryPostProcessor registryProcessor = (BeanDefinitionRegistryPostProcessor) postProcessor;
      65
      66
      67
                                                                                    registryProcessor.postProcessBeanDefinitionRegistry(registry);
      68
                                                                               registryProcessor .add(registryProcessor);
      69
      70
                                                                    else
      71
                                                                                regularPostProcessors.add(postProcessor);
      72
       73
                                                      }
      74
```

根据优先级,分别执行BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

继续按下 F6 快捷键让程序往下运行,直至运行到下面这行代码处,可以看到现在是会拿到所有BeanDefinitionRegistryPostProcessor的这些bean的名字。

```
AnnotationConfigApplicationContext.class PostProcessorRegistrationDelegate.class
                // Do not initialize FactoryBeans here: We need to leave all regular beans
 76
                   uninitialized to let the bean factory post-processors apply to them!
 77
                // Separate between BeanDefinitionRegistryPostProcessors that implement
 78
                   PriorityOrdered, Ordered, and the rest
                List<BeanDefinitionRegistryPostProcessor> currentRegistryProcessors = new ArrayList<BeanDefinitionRegistryPostProcessor>()
 80
                   81
                String[] postProcessorNames =
 82
                         peanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false);
 83
 84
                for (String ppName : postProcessorNames) {
                    if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
    currentRegistryProcessors.add(beanFactory.getBean(ppName, BeanDefinitionRegistryPostProcessor.class));
 85
 86
 87
                        processedBeans.add(ppName);
 88
 89
 90
                sortPostProcessors ({\tt currentRegistryProcessors, beanFactory});\\
 91
                registryProcessors.addAll(currentRegistryProcessors);
 92
                invokeBeanDefinitionRegistryPostProcessors(currentRegistryProcessors, registry);
 93
                currentRegistryProcessors.clear();
 94
```

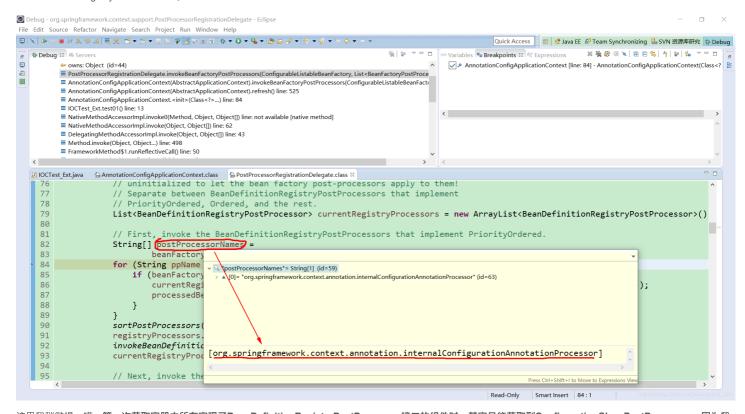
有意思的是,如果说你留心的话,那么你会发现每次执行前,都会运行完这么一行代码:

1 | beanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false); AI写代码java运行

2025/9/16 08:27 Spring注解驱动开发第42讲——Spring IOC容器创建源码解析(二)之执行BeanFactoryPostProcessor\_beandefinitionregistrypostprocessor ... 这行代码的意思,我上面已经说过了,就是来获取容器中所有实现了BeanDefinitionRegistryPostProcessor接口的组件。**那么,为什么每次执行前,都会运行这样一行代码** 呢?这是因为我们每次执行可能会加载进来新的BeanDefinition,所以每次都要重新获取。

执行实现了PriorityOrdered优先级接口的BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

继续按下 F6 快捷键让程序往下运行,往下运行一步即可,Inspect一下postProcessorNames变量的值,你会发现从IOC容器中拿到的只有一个名字为 org.springframework.context.annotation.internalConfigurationAnnotationProcessor的组件,即默认拿到的是ConfigurationClassPostProcessor这样一个 BeanDefinitionRegistryPostProcessor。



这里我稍微提一嘴,第一次获取容器中所有实现了BeanDefinitionRegistryPostProcessor接口的组件时,其实只能获取到ConfigurationClassPostProcessor,因为我们手工加的只是BeanDefinition,等ConfigurationClassPostProcessor把对应的Definition加载后,下面才能获取到我们手工加载的BeanDefinition。

不扯远了,我们还是回到程序中来。获取到容器中所有BeanDefinitionRegistryPostProcessor组件之后,接下来,就得遍历所有这些BeanDefinitionRegistryPostProcessor组件了,接个遍历出来之后,会判断每一个BeanDefinitionRegistryPostProcessor组件是不是实现了PriorityOrdered这个优先级接口,若是,则会先按照优先级排个序,然后再调用该组件的postProcessBeanDefinitionRegistry方法。

```
// Do not initialize FactoryBeans here: We need to leave all regular beans
 76
                 // uninitialized to let the bean factory post-processors apply to them!
 77
                 // Separate between BeanDefinitionRegistryPostProcessors that implement
 78
79
                 // PriorityOrdered, Ordered, and the rest
                List<BeanDefinitionRegistryPostProcessor> currentRegistryProcessors = new ArrayList<BeanDefinitionRegistryPostProcessor>()
 80
 81
                 // First, invoke the BeanDefinitionRegistryPostProcessors that implement PriorityOrdered.
 82
                String[] postProcessorNames
 83
                        beanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false);
 84
                 for (String ppName : postProcessorNames) {
                    if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
    currentRegistryProcessors.add(beanFactory.getBean(ppName, BeanDefinitionRegistryPostProcessor.class));
 85
 86
 87
                        processedBeans.add(ppName);
 88
                                       按照优先级排个序
 89
                cortPostProcessors(currentRegistryProcessors, beanFactory);
 90
 91
                 registryProcessors.addAll(currentRegistryProcessors);
 92
                invokeBeanDefinitionRegistryPostProcessors currentRegistryProcessors, registry);
 93
                    rentRegistryProcessors.clear();
                                                                    ► 执行每一个实现了PriorityOrdered优先级接口的
 94
                                                                      BeanDefinitionRegistryPostProcessor组件的方法
```

继续按下 F6 快捷键让程序往下运行,直至运行到下面这行代码处,这儿就是来执行每一个实现了PriorityOrdered优先级接口的BeanDefinitionRegistryPostProcessor组件的方法的。

```
// Do not initialize FactoryBeans here: We need to leave all regular beans
 76
                 // uninitialized to let the bean factory post-processors apply to them!
                 // Separate between BeanDefinitionRegistryPostProcessors that implement
 78
                 // PriorityOrdered, Ordered, and the rest
 79
                List<BeanDefinitionRegistryPostProcessor> currentRegistryProcessors = new ArrayList<BeanDefinitionRegistryPostProcessor>()
 80
                 // First, invoke the BeanDefinitionRegistryPostProcessors that implement PriorityOrdered.
 81
 82
                String[] postProcessorNames
                         beanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false);
 83
 84
                 for (String ppName :
                                      postProcessorNames) {
                     if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
 85
                         current Registry Processors. {\tt add(beanFactory.getBean(ppName, BeanDefinitionRegistry PostProcessor.class))}; \\
 86
 87
                         processedBeans.add(ppName);
 88
                    }
 89
                 sortPostProcessors(currentRegistryProcessors, beanFactory);
registryProcessors.addAll(currentRegistryProcessors);
 90
 91
                 invokeBeanDefinitionRegistryPostProcessors(currentRegistryProcessors, registry);
 92
 93
                 currentRegistryProcessors.clear();
 94
```

我们不妨按下 F5 快捷键进入该方法中去看一看,如下图所示,可以看到这儿是来执行BeanDefinitionRegistryPostProcessor组件的postProcessBeanDefinitionRegistry方法的。

```
IOCTest_Ext.java
             ⅓ AnnotationConfigApplicationContext.class ⅓ PostProcessorRegistrationDelegate.class ⋈
 263
 264
 265
 266
           * Invoke the given BeanDefinitionRegistryPostProcessor beans.
 267
 268
         private static void invokeBeanDefinitionRegistryPostProcessors(
                  Collection<? extends BeanDefinitionRegistryPostProcessor> postProcessors, BeanDefinitionRegistry registry) {
 270
 271
272
273
              for (BeanDefinitionRegistryPostProcessor postProcessor : postProcessors) {
                  postProcessor. \underline{postProcessBeanDefinitionRegistry} (registry);\\
 274
         }
 275
 276
           * Invoke the given BeanFactoryPostProcessor beans
 278
         private static void invokeBeanFactoryPostProcessors(
 280
                  Collection<? extends BeanFactoryPostProcessor> postProcessors, ConfigurableListableBeanFactory beanFactory) {
 281
 282
              for (BeanFactorvPostProcessor postProcessor : postProcessors) {
```

执行实现了Ordered顺序接口的BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

继续按下 F6 快捷键让程序往下运行,直至运行到下面这行代码处,可以看到在每次执行前都会执行下面一行代码,这是因为我们每次执行可能会加载进来新的
BeanDefinition,所以每次都要重新获取所有实现了BeanDefinitionRegistryPostProcessor接口的组件。其实,我在上面已经讲过一遍了,这里再讲一遍,大家可一定要注 意幽⊖

```
95
                 // Next, invoke the BeanDefinitionRegistryPostProcessors that implement Ordered
                 postProcessorNames = beanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false);
 97
                 for (String ppName : postProcessorNames) {
                     if (!processedBeans.contains(ppName) && beanFactory.isTypeMatch(ppName, Ordered.class)) {
 98
 99
                         current Registry Processors. {\tt add(beanFactory.getBean(ppName, BeanDefinitionRegistry PostProcessor.class))}; \\
100
                         processedBeans.add(ppName);
101
102
103
                 sortPostProcessors(currentRegistryProcessors, beanFactory);
                registryProcessors.addAll(currentRegistryProcessors);
invokeBeanDefinitionRegistryPostProcessors(currentRegistryProcessors, registry);
105
                 currentRegistryProcessors.clear();
```

很明显,这儿是来执行实现了Ordered顺序接口的BeanDefinitionRegistryPostProcessor组件的方法的。

原理同上面都是一模一样的,都是获取到容器中所有BeanDefinitionRegistryPostProcessor组件,紧接着再来遍历所有这些BeanDefinitionRegistryPostProcessor组件,挨个遍历出来之后,会判断每一个BeanDefinitionRegistryPostProcessor组件是不是实现了Ordered这个顺序接口,若是,则会先按照指定顺序来排个序,然后再调用该组件的postProcessBeanDefinitionRegistry方法。

```
// Next, invoke the BeanDefinitionRegistryPostProcessors that implement Ordered.
 95
                postProcessorNames = bean Factory. {\tt getBeanNamesForType} ({\tt BeanDefinitionRegistryPostProcessor.class, true, false);} \\
 96
                for (String ppName : postProcessorNames) {
   if (!processedBeans.contains(ppName) && beanFactory.isTypeMatch(ppName, Ordered.class)) {
 97
 98
 99
                        currentRegistryProcessors.add(beanFactory.getBean(ppName, BeanDefinitionRegistryPostProcessor.class));
100
                        processedBeans.add(ppName);
                    }
101

▼ 按照指定顺序排个序

102
103
                sortPostProcessors(currentRegistryProcessors, beanFactory);
                 registryProcessors.addAll(currentRegistryProcessors);
104
105
                invokeBeanDefinitionRegistryPostProcessors(currentRegistryProcessors, registry);
106
                currentRegistryProcessors.clear();
                                                                ►执行每一个实现了Ordered顺序接口的BeanDefinitionRegistryPostProcessor组件的方法
107
```

执行没有实现任何优先级或者是顺序接口的BeanDefinitionRegistryPostProcessor的postProcessBeanDefinitionRegistry方法

```
108
                // Finally, invoke all other BeanDefinitionRegistryPostProcessors until no further ones appear.
109
                boolean reiterate = true;
110
                while (reiterate) {
                    reiterate = false;
112
                    postProcessorNames = beanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false);
113
                    for (String ppName : postProcessorNames)
                        if (!processedBeans.contains(ppName)) {
114
115
                           currentRegistryProcessors.add(beanFactory.getBean(ppName, BeanDefinitionRegistryPostProcessor.class));
                            processedBeans.add(ppName);
117
                            reiterate = true;
118
                        }
119
                    sortPostProcessors(currentRegistryProcessors, beanFactory);
120
                    registryProcessors.addAll(currentRegistryProcessors);
                    invoke \textit{BeanDefinitionRegistryPostProcessors} (\textit{currentRegistryProcessors}, \textit{registry});
123
                    currentRegistryProcessors.clear();
124
                }
```

很明显,这块是来执行没有实现任何优先级或者是顺序接口的BeanDefinitionRegistryPostProcessor组件的方法的。

原理基本同上,首先获取到容器中所有BeanDefinitionRegistryPostProcessor组件,然后遍历所有这些BeanDefinitionRegistryPostProcessor组件,挨个遍历出来之后,接着再调用该组件的postProcessBeanDefinitionRegistry方法。

```
108
                // Finally, invoke all other BeanDefinitionRegistryPostProcessors until no further ones appear.
109
               boolean reiterate = true;
110
               while (reiterate) {
                   reiterate = false:
                   postProcessorNames = beanFactorv.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false):
112
                   for (String ppName : postProcessorNames) {
113
                       if (!processedBeans.contains(ppName))
114
115
                           current Registry Processors. add (bean Factory. get Bean (pp Name, Bean Definition Registry Post Processor. class)); \\
                           processedBeans.add(ppName);
117
                           reiterate = true:
118
                       }
119
120
                   sortPostProcessors(currentRegistryProcessors, beanFactory);
                                     addAll(currentRegistryProcessors);
                  (currentRegistryProcessors) (currentRegistryProcessors, registry);
123
                   currentRegistryProcessors.clear();
               }
125
```

继续按下 F6 快捷键让程序往下运行,直至运行到下面这行代码处,这时,你会发现Eclipse 控制台有内容输出。

```
Debug - org.springframework.context.support.PostProcessorRegistrationDelegate - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access 🔛 😤 Java EE 😜 Team Synchronizing 🔓 SVN 资源库研究 🂠 Debug
      松椒
                                                                                                                                                                                                                                                                                                                                                        (x)=
        108
                                               // Finally, invoke all other BeanDefinitionRegistryPostProcessors until no further ones appear.
                                              boolean reiterate = true;
        110
                                              while (reiterate) {
                                                                                                                                                                                                                                                                                                                                                        8 8
                                                        reiterate = false;
                                                        postProcessorNames = beanFactory.getBeanNamesForType(BeanDefinitionRegistryPostProcessor.class, true, false);
                                                        for (String ppName : postProcessorNames) {
                                                                 if (!processedBeans.contains(ppName)) {
        115
                                                                           current Registry Processors. {\tt add(beanFactory.getBean(ppName, BeanDefinitionRegistry PostProcessor.class));}
        116
                                                                          processedBeans.add(ppName);
        117
                                                                           reiterate = true;
        118
                                                                 }
        119
                                                       sortPostProcessors(currentRegistryProcessors, beanFactory);
registryProcessors.addAll(currentRegistryProcessors);
        120
                                                        invokeBeanDefinitionRegistryPostProcessors(currentRegistryProcessors, registry);
                                                        currentRegistryProcessors.clear();
        124
                                              }
                                              // Now, invoke the postProcessBeanFactory callback of all processors handled so far.
invokeBeanFactoryPostProcessors(registryProcessors, beanFactory);
        127
                                               invokeBeanFactoryPostProcessors(regularPostProcessors, beanFactory);
        128
        129
                                    }
       130
                                                                                                                                                                                                                                                                                      IOCTest_Ext.test01 (1) [JUnit] D:\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2021年2月26日 下午5:13:33)
         Reference of Chromosome Properties of Continuary Action (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) (1997) 
       postProcessBeanDefinitionRegistry...bean的数量: 12_
                                                                                                                                                                                                                                    Read-Only Smart Insert 127:1
```

很明显,这是咱们自己编写的MyBeanDefinitionRegistryPostProcessor类中的postProcessBeanDefinitionRegistry方法执行之后所输出的信息。

### 执行BeanDefinitionRegistryPostProcessor的postProcessBeanFactory方法

因为BeanDefinitionRegistryPostProcessor是BeanFactoryPostProcessor的子接口,所以,接下来还得执行BeanDefinitionRegistryPostProcessor组件里面的postProcessBeanFactory方法。

按下 F6 快捷键让程序往下运行,往下运行一步即可,这时,你同样会发现Eclipse控制台有内容输出。

```
Debug - org.springframework.context.support.PostProcessorRegistrationDelegate - Eclipse
                                                                                                                                                                                            σ
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access 🔛 🙁 Java EE 🖆 Team Synchronizing 🤮 SVN 资源库研究 🏠 Debug
                                                                                                                                                             x % 2 4 x | B B 5 | 1 | 9 | % V B B
                                                                                                                     Variables • Breakpoints 🛭 🎋 Expressions
No. 10
                                                                                                                     AnnotationConfigApplicationContext [line: 84] - AnnotationConfigApplicationContext(Class<?>.
          owns: Object (id=44
        PostProcessorRegistrationDelegate.invokeBeanFactoryPostProcessors(ConfigurableListableBeanFactory, List<BeanFactoryPostProcessors)
        AnnotationConfigApplicationContext(AbstractApplicationContext).invokeBeanFactoryPostProcessors(ConfigurableListaten AnnotationConfigApplicationContext(AbstractApplicationContext).refresh() line: 525
        ■ AnnotationConfigApplicationContext, <init>(Class <?>...) line: 84
        ■ IOCTest_Ext.test010 line: 13
■ NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
<
                                                       ♣ PostProcessorRegistrationDelegate.class 

                                                                                                                                                                                              - -
                 AnnotationConfigApplicationContext.class
                            registryProcessors.addAll(currentRegistryProcessors);
 122
123
                            invokeBeanDefinitionRegistryPostProcessors(currentRegistryProcessors, registry);
                            currentRegistryProcessors.clear();
 124
                      }
                                                            ▼执行BeanDefinitionRegistryPostProcessor组件里面的postProcessBeanFactory方法
 125
                                  invoke the postProcessBeanFactory callback of all processors handled so far.
                      invokeBeanFactoryPostProcessor3(registryProcessors, beanFactory);
invokeBeanFactoryPostProcessors(regularPostProcessors, beanFactory);
 128
 129
 130
                 else {
                          Invoke factory processors registered with the context instance
                       invoke \textit{BeanFactoryPostProcessors} (bean \textit{FactoryPostProcessors}, \ bean \textit{Factory}); \\
 134
                 }
 135
                                                                                                                                                              © Console ⋈ 🗷 Tasks 🛗 JUnit
IOCTest_Ext.test01 (1) [Unit] D\Developer\Java\jdk1.8.0_181\bin\javaw.exe (2021年2月26日 下午5:13:33)
二月 26, 2021 5:13:39 下午 org.springframework.context.annotation.AnnotationConfigApplicationContext prepareRefresh
      Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@51e2adc7: startup date [Fri Feb 26 17:13:39 CST 2021]
postProcessBeanDefinitionRegistry...bean的数量: 12
MyBeanDefinitionRegistryPostProcessor...bean的数量: 13
```

很明显,这是咱们自己编写的MyBeanDefinitionRegistryPostProcessor类中的postProcessBeanFactory方法执行之后所输出的信息。

也就是说,对于BeanDefinitionRegistryPostProcessor组件来说,它里面postProcessBeanDefinitionRegistry方法会先被调用,postProcessBeanFactory方法会后被调用。

## 再执行BeanFactoryPostProcessor的方法

在《Spring注解驱动开发第37讲——你知道Spring中BeanDefinitionRegistryPostProcessor是如何执行的吗?》这一讲中,我们就知道了,

BeanDefinitionRegistryPostProcessor是要优先于BeanFactoryPostProcessor执行的。在上面已经执行完了BeanDefinitionRegistryPostProcessor的方法,接下来就得来执行BeanFactoryPostProcessor的方法了。

执行的流程是怎样的呢?我们可以大致地来看一下,按下 F6 快捷键让程序往下运行,直至程序运行到以下这行代码处,可以看到现在是来从beanFactory中按照类型获取 所有BeanFactoryPostProcessor组件的名字。

```
🖟 AnnotationConfigApplicationContext.class 🖟 PostProcessorRegistrationDelegate.class 🛭
⚠ IOCTest Ext.java
136
             // Do not initialize FactoryBeans here: We need to leave all regular beans
137
             // uninitialized to let the bean factory post-processors apply to them!
138
             String[] postProcessorNames
139
                     beanFactory.getBeanNamesForType(BeanFactoryPostProcessor.class, true, false);
140
             // Separate between BeanFactoryPostProcessors that implement PriorityOrdered
141
                                                                 ►从beanFactory中按照类型来获取所有BeanFactoryPostProcessor组件(的名字)
142
             // Ordered. and the rest.
             List<BeanFactoryPostProcessor> priorityOrderedPostProcessors = new ArrayList<BeanFactoryPostProcessor>();
144
             List<String> orderedPostProcessorNames = new ArrayList<String>();
145
             List<String> nonOrderedPostProcessorNames = new ArrayList<String>();
146
             for (String ppName : postProcessorNames) {
147
                 if (processedBeans.contains(ppName)) {
148
                     // skip - already processed in first phase above
149
150
                 else if (beanFactory.isTypeMatch(ppName, PriorityOrdered.class)) {
151
                     priorityOrderedPostProcessors.add(beanFactory.getBean(ppName, BeanFactoryPostProcessor.class));
                 else if (beanFactory.isTypeMatch(ppName, Ordered.class)) {
153
                     orderedPostProcessorNames.add(ppName);
                 else {
                     nonOrderedPostProcessorNames.add(ppName);
158
             }
159
```

获取到所有BeanFactoryPostProcessor组件之后,接下来,就得遍历所有这些BeanFactoryPostProcessor组件了,挨个遍历出来之后,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,将其分别存储于三个ArrayList中。

紧接着,按照顺序依次执行BeanFactoryPostProcessors组件对应的postProcessBeanFactory方法。

nonOrderedPostProcessorNames.add(ppName);

else {

}

2025/9/16 08:27

158

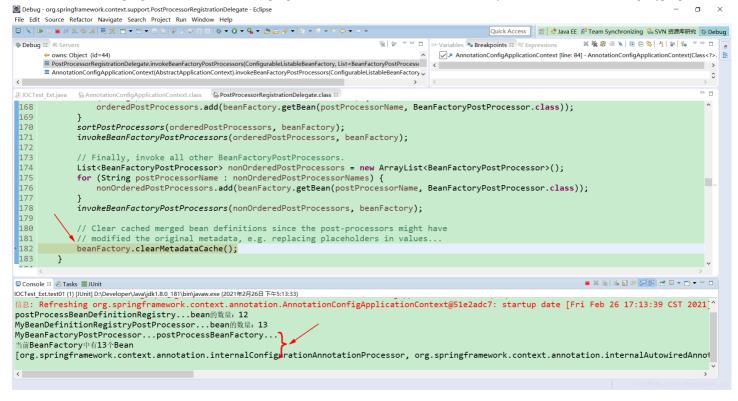
159

```
// First, invoke the BeanFactoryPostProcessors that implement PriorityOrdered.
                                  sortPostProcessors(priorityOrderedPostProcessors, beanFactory);
                                 invokeBeanFactoryPostProcessors(priorityOrderedPostProcessors, beanFactory);
  164
                                  // Next, invoke the BeanFactoryPostProcessors that implement Ordered.
                                  List<BeanFactoryPostProcessor> orderedPostProcessors = new ArrayList<BeanFactoryPostProcessor>();
                                  for (String postProcessorName : orderedPostProcessorNames) {
  167
  168
                                            ordered Post Processors. add (bean Factory. {\tt getBean} (post Processor Name, {\tt Bean Factory Post Processor. class})); and the processor of the processor of
                                 sortPostProcessors(orderedPostProcessors, beanFactory);
invokeBeanFactoryPostProcessors(orderedPostProcessors, beanFactory);
  170
  171
  173
                                  // Finally, invoke all other BeanFactoryPostProcessors.
  174
175
                                  {\tt List < Bean Factory Post Processor > non Ordered Post Processor = new Array List < Bean Factory Post Processor > ();}
                                  for (String postProcessorName : nonOrderedPostProcessorNames) {
  176
                                            nonOrderedPostProcessors.add(beanFactory.getBean(postProcessorName, BeanFactoryPostProcessor.class));
  178
                                  invokeBeanFactoryPostProcessors(nonOrderedPostProcessors, beanFactory);
  179
                                  // Clear cached merged bean definitions since the post-processors might have // modified the original metadata, e.g. replacing placeholders in values...
  180
  181
  182
                                  beanFactory.clearMetadataCache();
  183
  184
```

也就是说,先来执行实现了PriorityOrdered优先级接口的BeanFactoryPostProcessor组件的postProcessBeanFactory方法,再来执行实现了Ordered顺序接口的BeanFactoryPostProcessor组件的postProcessBeanFactory方法,最后再来执行没有实现任何优先级或者是顺序接口的BeanFactoryPostProcessor组件的postProcessBeanFactory方法。

你有没有发现,程序直至到这儿,才是来执行所有BeanFactoryPostProcessor组件的postProcessBeanFactory方法的呢?

继续按下 F6 快捷键让程序往下运行,直至程序运行到以下这行代码处,这时,你会发现Eclipse控制台有内容输出。



很明显,这是咱们自己编写的MyBeanFactoryPostProcessor类中的postProcessBeanFactory方法执行之后所输出的信息。

#### 小结

继续按下 F6 快捷键让程序往下运行,直至程序运行到以下这行代码处,这时,invokeBeanFactoryPostProcessors方法才总算是执行完了。

```
508
         @Override
         public void refresh() throws BeansException, IllegalStateException {
 509
 510
             synchronized (this.startupShutdownMonitor) {
 511
                 // Prepare this context for refreshing.
 512
                 prepareRefresh();
 513
 514
                 // Tell the subclass to refresh the internal bean factory
                ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();
 516
                 // Prepare the bean factory for use in this context.
 518
                prepareBeanFactory(beanFactory);
 519
                try {
    // Allows post-processing of the bean factory in context subclasses.
 520
 521
 522
                    postProcessBeanFactory(beanFactory);
 523
 524
                     // Invoke factory processors registered as beans in the context.
                     invokeBeanFactoryPostProcessors(beanFactory);
 525
 526
                     // Register bean processors that intercept bean creation.
 528
                    registerBeanPostProcessors(beanFactory);
 529
 530
                     // Initialize message source for this context.
 531
                    initMessageSource();
 532
                    // Initialize event multicaster for this context.
 533
```

至此,我们知道了一点,那就是invokeBeanFactoryPostProcessors方法最主要的核心作用就是执行了BeanDefinitionRegistryPostProcessor的 postProcessBeanDefinitionRegistry和postProcessBeanFactory这俩方法,以及BeanFactoryPostProcessors的postProcessBeanFactory方法。而这正呼应了开头,你说是不是呢?

而且,还有一点我们需要知道,那就是BeanDefinitionRegistryPostProcessor是要优先于BeanFactoryPostProcessor执行的。