spring mvc 扫描与注解

2017-06-06 IT哈哈

在spring mvc中扫描注解机制是我们理解javabean是怎么被加载，是如何被spring进行管理的第一步。那spring mvc 是如何扫描所有的编译文件并对注解进行操作的呢，下面我们来看下：

在spring-mvc中我们都会配置一个web.xml文件，内容如下：

```
<servlet>
    <servlet-name>springServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

我们知道只要servlet中的load-on-startup配置了大于1的数字，类就会在应用启动的时候被加载，在加载时候是调用了DispatcherServlet 的init()方法。
但我们进入dispatcherServlet并没有找到init()方法，此时我们继续查找父类，在其父类的父类HttpServletBean这个类中找到了init()方法:

```
125
126         // Let subclasses do whatever initialization they like.
127         initServletBean();
128
129         if (logger.isDebugEnabled()) {
130             logger.debug("Servlet '" + getServletName() + "' configured successfully");
131         }
```

init方法中有一个initServletBean()方法，这个方法就是用来初始化javabean的。
在其子类，FrameworkServlet中我们看到了initServletBean方法的实现方法，其方法中有这样一行代码：

```
305         try {
306             this.webApplicationContext = initWebApplicationContext();
307             initFrameworkServlet();
308         }
```

其实，只要获取到ApplicationContext就能获取到javabean信息，此时我们进去看下，其是如何初始化ApplicationContext的：

```
330     * @see #setContextClass
331     * @see #setContextConfigLocation
332     */
333    protected WebApplicationContext initWebApplicationContext() {
334        WebApplicationContext wac = findWebApplicationContext();
335        if (wac == null) {
336            // No fixed context defined for this servlet - create a local one.
337            WebApplicationContext parent =
338                    WebApplicationContextUtils.getWebApplicationContext(getServletContext());
339            wac = createWebApplicationContext(parent);
340        }
341
342        if (!this.refreshEventReceived) {
343            // Apparently not a ConfigurableApplicationContext with refresh support:
344            // triggering initial onRefresh manually here.
345            onRefresh(wac);
346        }
```

在FrameworkServlet中，标红代码创建了ApplicationContext，关于其中的parent参数，以后再说，继续跟进，在createWebApplicationContext方法中：

```
434        wac.setParent(parent);
435        wac.setServletContext(getServletContext());
436        wac.setServletConfig(getServletConfig());
437        wac.setNamespace(getNamespace());
438        wac.setConfigLocation(getContextConfigLocation());
439        wac.addApplicationListener(new SourceFilteringListener(wac, new ContextRefreshListener()));
440
441        postProcessWebApplicationContext(wac);
442        wac.refresh();
```

其实，不论是ClassPathXmlApplicationContext还是WEB装载都会调用这段方法，区别只是在于，web容器中，用servlet装载时，servlet包装了一个XmlWebApplicationContext而已，无论是ClassPathXmlApplicationContext还是XmlWebApplicationContext他们都继承自抽象类：AbstractApplicationContext，他们共用了AbstractApplicationContext的refresh()方法：

```
391    public void refresh() throws BeansException, IllegalStateException {
392        synchronized (this.startupShutdownMonitor) {
393            // Prepare this context for refreshing.
394            prepareRefresh();
395
396            // Tell the subclass to refresh the internal bean factory.
397            ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();
398
399            // Prepare the bean factory for use in this context.
400            prepareBeanFactory(beanFactory);
401
```

虽然这些一层调用一层看起来很烦，但还是希望大家有时间看下去，因为在spring-mvc装载bean的时候不仅要读取解析xml文件还要对class文件进行bean读取加载，并且在xml解析，类加载，实例化的过程中有很多需求，并且spring-mvc采用了分离设计，下面我们来看看obtainFreshBeanFactory()方法：

```
466    protected ConfigurableListableBeanFactory obtainFreshBeanFactory() {
467        refreshBeanFactory();
468        ConfigurableListableBeanFactory beanFactory = getBeanFactory();
469        if (logger.isDebugEnabled()) {
```

在obtainFreshBeanFactory()方法中，加载bean的是第一行refreshBeanFactory()方法，而不是getBeanFactory()方法。

关于refreshBeanFactory()的实现是在AbstractRefreshableApplicationContext类中实现了，它继承自AbstractApplicationContext同时，

ClassPathXmlApplicationContext与XmlWebApplicationContext也继承了

AbstractRefreshableApplicationContext这个类

```
127             DefaultListableBeanFactory beanFactory = createBeanFactory();
128             beanFactory.setSerializationId(getId());
129             customizeBeanFactory(beanFactory);
130             loadBeanDefinitions(beanFactory);
131             synchronized (this.beanFactoryMonitor) {
```

在第一行，他createBeanFactory，同时在loadBeanDefinitions(beanFactory)中加载bean并将bean放入了BeanFactory中，继续跟踪loadBeanDefinitions方法，它是由AbstractXmlApplicationContext类中的方法实现，web项目中会由XmlWebApplicationContext来实现，loadBeanDefinitions方法如下：

```
119     protected void loadBeanDefinitions(XmlBeanDefinitionReader reader) throws BeansException, IOException {
120         Resource[] configResources = getConfigResources();
121         if (configResources != null) {
122             reader.loadBeanDefinitions(configResources);
123         }
124         String[] configLocations = getConfigLocations();
125         if (configLocations != null) {
126             reader.loadBeanDefinitions(configLocations);
127         }
128     }
```

它通过XmlBeanDefinitionReader类去读取springXML中的信息(也就是解析SpringContext.xml)并加载bean，接下来会调用多层，并在XMLBeanDefinitionReader类中去调用doLoadBeanDefinitions、regiseterBeanDefinitions操作，在regiseterBeanDefinitions的时候就开始解析XML了。它调用了DefaultBeanDefinitionDocumentReader类的registerBeanDefinitions方法，如下图所示：

```
84      public void registerBeanDefinitions(Document doc, XmlReaderContext readerContext) {
85          this.readerContext = readerContext;
86
87          logger.debug("Loading bean definitions");
88          Element root = doc.getDocumentElement();
89
90          BeanDefinitionParserDelegate delegate = createHelper(readerContext, root);
91
92          preProcessXml(root);
93          parseBeanDefinitions(root, delegate);
94          postProcessXml(root);
```

在这个方法中做了解析XML中的操作，看标红代码，这段代码做了bean的解析工作。跟踪进去会发现里面解析了XML的信息，其中NamespaceHanderSupport的parse方法会根据节点的类型，找到合适的解析（BeanDefinitionParse）方式，他们预先已经被注册号了，放在一个HashMap中，例如在Spring的annotation的注解扫描中，我们通常会配置：<context:component-scan base-package="com.xxx">,此时根据名称“component-scan”就会找到对应的解析器来解析，此时解析注解是用的ComponentScanBeanDefinitionParser的parse方法。

在parse获取到后，有一个关键的步骤就是定义了ClassPathBeanDefinitionScanner来扫描类信息来对class文件进行扫描：

```
78      public BeanDefinition parse(Element element, ParserContext parserContext) {
79          String[] basePackages = StringUtils.tokenizeToStringArray(element.getAttribute(BASE_PACKAGE_ATTRIBUTE),
80                  ConfigurableApplicationContext.CONFIG_LOCATION_DELIMITERS);
81
82          // Actually scan for bean definitions and register them.
83          ClassPathBeanDefinitionScanner scanner = configureScanner(parserContext, element);
84          Set<BeanDefinitionHolder> beanDefinitions = scanner.doScan(basePackages);
85          registerComponents(parserContext.getReaderContext(), beanDefinitions, element);
86
87          return null;
88      }
```

这个方法中最重要的就是doScan方法的作用，那doScan方法是如何扫描的呢，这里可以跟踪进去一起看看：

我们先跟进去找到findCandidateComponents方法：

```
199    public Set<BeanDefinition> findCandidateComponents(String basePackage) {
200        Set<BeanDefinition> candidates = new LinkedHashSet<BeanDefinition>();
201        try {
202            String packageSearchPath = ResourcePatternResolver.CLASSPATH_ALL_URL_PREFIX +
203                    resolveBasePackage(basePackage) + "/" + this.resourcePattern;
204            Resource[] resources = this.resourcePatternResolver.getResources(packageSearchPath);
205            boolean traceEnabled = logger.isTraceEnabled();
206            boolean debugEnabled = logger.isDebugEnabled();
207            for (Resource resource : resources) {
208                if (traceEnabled) {
209                    logger.trace("Scanning " + resource);
210                }
211                if (resource.isReadable()) {
212                    try {
213                        MetadataReader metadataReader = this.metadataReaderFactory.getMetadataReader(resource);
214                        if (isCandidateComponent(metadataReader)) {
215                            ScannedGenericBeanDefinition sbd = new ScannedGenericBeanDefinition(metadataReader);
216                            sbd.setResource(resource);
217                            sbd.setSource(resource);
218                            if (isCandidateComponent(sbd)) {
219                                if (debugEnabled) {
220                                    logger.debug("Identified candidate component class: " + resource);
221                                }
222                                candidates.add(sbd);
223                            }
```

在此方法中通过标红代码获取路径，并取得了class文件，那么是怎么获取到class文件的呢，有兴趣的朋友可以将classpath路径写为：自己项目编译文件路径例如：classpath*:org/sinovate/**/*.class,并用如下操作：

```
204            Resource[] resources = this.resourcePatternResolver.getResources(packageSearchPath);
```

在控制台输出resources信息便可以看到class文件路径名称信息等。在获取到.class路径以及文件名称后，我们就可以通过classLoaderListenser加载.class文件了，此时在根据Annotation类的方法就可以对@Controller,@Service,@Resource,等一系列注解进行控制与装载了。

阅读原文