【死磕 Spring】—— IOC 之 注册 BeanDefinition (http://cmsblogs.com/?p=2697)

2018-09-13 分类：死磕 Spring (http://cmsblogs.com/?cat=206) 阅读(14119) 评论(0)

> 原文出自：http://cmsblogs.com (http://cmsblogs.com)

获取 Document 对象后，会根据该对象和 Resource 资源对象调用 `registerBeanDefinitions()` 方法，开始注册 BeanDefinitions 之旅。如下：

```
    public int registerBeanDefinitions(Document doc, Resource resource) throws BeanDefinitionStoreExcepti
on {
        BeanDefinitionDocumentReader documentReader = createBeanDefinitionDocumentReader();
        int countBefore = getRegistry().getBeanDefinitionCount();
        documentReader.registerBeanDefinitions(doc, createReaderContext(resource));
        return getRegistry().getBeanDefinitionCount() - countBefore;
    }
```

第一步、创建BeanDefinitionReader对象

首先调用 `createBeanDefinitionDocumentReader()` 方法实例化 BeanDefinitionDocumentReader 对象，然后获取统计前 BeanDefinition 的个数，最后调用 `registerBeanDefinitions()` 注册 BeanDefinition。 实例化 BeanDefinitionDocumentReader 对象方法如下：

```
    protected BeanDefinitionDocumentReader createBeanDefinitionDocumentReader() {
        return BeanDefinitionDocumentReader.class.cast(BeanUtils.instantiateClass(this.documentReaderClas
s));
    }
```

注册 BeanDefinition 的方法 `registerBeanDefinitions()` 是在接口 BeanDefinitionDocumentReader 中定义，如下：

```
    void registerBeanDefinitions(Document doc, XmlReaderContext readerContext)
            throws BeanDefinitionStoreException;
```

从给定的 Document 对象中解析定义的 BeanDefinition 并将他们注册到注册表中。方法接收两个参数，待解析的 Document 对象，以及解析器的当前上下文，包括目标注册表和被解析的资源。其中 readerContext 是根据 Resource 来创建的，如下：

```
    public XmlReaderContext createReaderContext(Resource resource) {
        return new XmlReaderContext(resource, this.problemReporter, this.eventListener,
                this.sourceExtractor, this, getNamespaceHandlerResolver());
    }
```

第二步、BeanDefinitionReader对象注册Document对象到IOC容器中

DefaultBeanDefinitionDocumentReader 对该方法提供了实现：


```
public void registerBeanDefinitions(Document doc, XmlReaderContext readerContext) {
    this.readerContext = readerContext;
    logger.debug("Loading bean definitions");
    Element root = doc.getDocumentElement();
    doRegisterBeanDefinitions(root);
}
```

调用 doRegisterBeanDefinitions() 开启注册 BeanDefinition 之旅。

```
protected void doRegisterBeanDefinitions(Element root) {
    BeanDefinitionParserDelegate parent = this.delegate;
    this.delegate = createDelegate(getReaderContext(), root, parent);

    if (this.delegate.isDefaultNamespace(root)) {
          // 处理 profile
        String profileSpec = root.getAttribute(PROFILE_ATTRIBUTE);
        if (StringUtils.hasText(profileSpec)) {
            String[] specifiedProfiles = StringUtils.tokenizeToStringArray(
                    profileSpec, BeanDefinitionParserDelegate.MULTI_VALUE_ATTRIBUTE_DELIMITERS);
            if (!getReaderContext().getEnvironment().acceptsProfiles(specifiedProfiles)) {
                if (logger.isInfoEnabled()) {
                    logger.info("Skipped XML bean definition file due to specified profiles [" + prof
ileSpec +
                            "] not matching: " + getReaderContext().getResource());
                }
                return;
            }
        }
    }

    // 解析前处理
    preProcessXml(root);
    // 解析
    parseBeanDefinitions(root, this.delegate);
    // 解析后处理
    postProcessXml(root);

    this.delegate = parent;
}
```

当前类名字是DefaultBeanDefinitionDocumentReader

程序首先处理 profile属性，profile主要用于我们切换环境，比如切换开发、测试、生产环境，非常方便。然后调用 parseBeanDefinitions() 进行解析动作，不过在该方法之前之后分别调用 preProcessXml() 和 postProcessXml() 方法来进行前、后处理，目前这两个方法都是空实现，交由子类来实现。

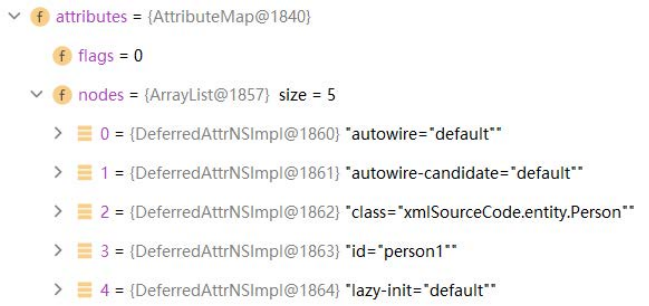
```
    protected void preProcessXml(Element root) {
    }

    protected void postProcessXml(Element root) {
    }
```

parseBeanDefinitions() 定义如下：对xml中根root级别的进行解析“import”，“alias”，“bean”

```
    protected void parseBeanDefinitions(Element root, BeanDefinitionParserDelegate delegate) {
        if (delegate.isDefaultNamespace(root)) {
            NodeList nl = root.getChildNodes();
            for (int i = 0; i < nl.getLength(); i++) {
                Node node = nl.item(i);
                if (node instanceof Element) {
                    Element ele = (Element) node;
                    if (delegate.isDefaultNamespace(ele)) {
                        parseDefaultElement(ele, delegate);
                    }
                    else {
                        delegate.parseCustomElement(ele);
                    }
                }
            }
        }
        else {
            delegate.parseCustomElement(root);
        }
    }
```

确定给定节点是否指示默认名称空间

ele是Element类型对象，内部有属性封装了xml中配置



最终解析动作落地在两个方法处：parseDefaultElement(ele, delegate) 和 delegate.parseCustomElement(root)。我们知道在 Spring 有两种 Bean 声明方式：

- 配置文件式声明：<bean id="studentService" class="org.springframework.core.StudentService"/>
- 自定义注解方式：<tx:annotation-driven>

两种方式的读取和解析都存在较大的差异，所以采用不同的解析方法，如果根节点或者子节点采用默认命名空间的话，则调用 parseDefaultElement() 进行解析，否则调用 delegate.parseCustomElement() 方法进行自定义解析。 至此， doLoadBeanDefinitions() 中做的三件事情已经全部分析完毕，下面将对 Bean 的解析过程做详细分析说明。

标签： Spring源码解析 (http://cmsblogs.com/?tag=spring%e6%ba%90%e7%a0%81%e8%a7%a3%e6%9e%90)

死磕Java (http://cmsblogs.com/?tag=%e6%ad%bb%e7%a3%95java)

死磕Spring (http://cmsblogs.com/?tag=%e6%ad%bb%e7%a3%95spring)