



[回到首页](#)

芋道源码 —— 知识星球

我是一段不羁的公告！

记得给芳芳这 3 个项目加油，添加一个 STAR 噢。

<https://github.com/YunaiV/SpringBoot-Labs>

<https://github.com/YunaiV/onemall>

<https://github.com/YunaiV/ruoyi-vue-pro>

2019-03-16

[Spring](#)

【死磕 Spring】—— IoC 之解析 <bean> 标签： 解析自定义标签

本文主要基于 Spring 5.0.6.RELEASE

摘要：原创出处 <http://cmsblogs.com/?p=2754> 「小明哥」，谢谢！

作为「小明哥」的忠实读者，「老芳芳」略作修改，记录在理解过程中，参考的资料。

前面四篇文章都是分析 Bean 默认标签的解析过程，包括 [基本属性](#)、六个子元素（[meta](#)、[lookup-method](#)、[replaced-method](#)、[constructor-arg](#)、[property](#)、[qualifier](#)），涉及内容较多，拆分成了四篇文章，导致我们已经忘记从哪里出发的了。所以，我们先来回顾下。

DefaultBeanDefinitionDocumentReader 的 #processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate) 方法，负责 <bean> 标签的解析：

在解析过程中，首先调用 BeanDefinitionParserDelegate#parseBeanDefinitionElement(Element ele) 方法，完成默认标签的解析。

如果解析成功（返回的 bdHolder != null），则调用

BeanDefinitionParserDelegate#decorateBeanDefinitionIfRequired(Element ele, BeanDefinitionHolder definitionHolder) 方法，完成自定义标签元素的解析。

1. decorateBeanDefinitionIfRequired

前面四篇文章已经分析了默认标签的解析，所以这篇文章分析自定义标签的解析。代码如下：

```
// BeanDefinitionParserDelegate.java

public BeanDefinitionHolder decorateBeanDefinitionIfRequired(
    Element ele, BeanDefinitionHolder definitionHolder, @Nullable BeanDefinition containingBd) {

    BeanDefinitionHolder finalDefinition = definitionHolder;

    // <1> 遍历属性，查看是否有适用于装饰的【属性】
    // Decorate based on custom attributes first.
    NamedNodeMap attributes = ele.getAttributes();
```

```

for (int i = 0; i < attributes.getLength(); i++) {
    Node node = attributes.item(i);
    finalDefinition = decorateIfRequired(node, finalDefinition, containingBd);
}

// <2> 遍历子节点，查看是否有适用于修饰的【子节点】
// Decorate based on custom nested elements.
NodeList children = ele.getChildNodes();
for (int i = 0; i < children.getLength(); i++) {
    Node node = children.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        finalDefinition = decorateIfRequired(node, finalDefinition, containingBd);
    }
}
return finalDefinition;
}

```

<1> 和 <2> 处，都是遍历，前者遍历的是属性（`attributes`），后者遍历的是子节点（`childNodes`），最终调用的都是 `#decorateIfRequired(Node node, BeanDefinitionHolder originalDef, BeanDefinition containingBd)` 方法，装饰对应的节点（`Node`）。详细解析，见 [\[2. decorateIfRequired\]](#)。

2. decorateIfRequired

`#decorateIfRequired(Node node, BeanDefinitionHolder originalDef, BeanDefinition containingBd)` 方法，装饰对应的节点（`Node`）。代码如下：

```

// BeanDefinitionParserDelegate.java

public BeanDefinitionHolder decorateIfRequired(
    Node node, BeanDefinitionHolder originalDef, @Nullable BeanDefinition containingBd) {
    // <1> 获取自定义标签的命名空间
    String namespaceUri = getNamespaceURI(node);
    // <2> 过滤掉默认命名标签
    if (namespaceUri != null && !isDefaultNamespace(namespaceUri)) {
        // <2> 获取相应的处理器
        NamespaceHandler handler = this.readerContext.getNamespaceHandlerResolver().resolve(namespaceUri);
        if (handler != null) {
            // <3> 进行装饰处理
            BeanDefinitionHolder decorated =
                handler.decorate(node, originalDef, new ParserContext(this.readerContext, this, containingBd));
            if (decorated != null) {
                return decorated;
            }
        } else if (namespaceUri.startsWith("http://www.springframework.org/")) {
            error("Unable to locate Spring NamespaceHandler for XML schema namespace [" + namespaceUri + "]", node);
        } else {
            // A custom namespace, not to be handled by Spring - maybe "xml:...".
            if (logger.isDebugEnabled()) {
                logger.debug("No Spring NamespaceHandler found for XML schema namespace [" + namespaceUri + "]);
            }
        }
    }
    return originalDef;
}

```

在 <1> 处，首先获取自定义标签的命名空间。
在 <2> 处，如果不是默认的命名空间，则根据该命名空间获取相应的处理器。
在 <3> 处，如果处理器存在，则进行装饰处理。

上述过程的详细解析，见 [《【死磕 Spring】—— IoC 之解析 标签：解析自定义标签》](#) 一文。

3. 小结

至此，BeanDefinition 的解析过程已经全部完成了，下面做一个简要的总结：

解析 BeanDefinition 的入口在 DefaultBeanDefinitionDocumentReader 的 #parseBeanDefinitions(Element root, BeanDefinitionParserDelegate delegate) 方法。该方法会根据命名空间来判断标签是默认标签还是自定义标签，其中：

默认标签，由 #parseDefaultElement(Element ele, BeanDefinitionParserDelegate delegate) 方法来实现
自定义标签，由 BeanDefinitionParserDelegate 的 #parseCustomElement(Element ele, @Nullable BeanDefinition containingBd) 方法来实现。

在默认标签解析中，会根据标签名称的不同进行 import、alias、bean、beans 四大标签进行处理。其中 bean 标签的解析为核心，它由 processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate) 方法实现。

processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate) 方法，开始进入解析核心工作，分为三步：

1. 解析默认标签的默认标签：BeanDefinitionParserDelegate#parseBeanDefinitionElement(Element ele, ...) 方法。该方法会依次解析 <bean> 标签的属性、各个子元素，解析完成后返回一个 GenericBeanDefinition 实例对象。
2. 解析默认标签下的自定义标签：BeanDefinitionParserDelegate#decorateBeanDefinitionIfRequired(Element ele, BeanDefinitionHolder definitionHolder) 方法。
3. 注册解析的 BeanDefinition：BeanDefinitionReaderUtils#registerBeanDefinition(BeanDefinitionHolder definitionHolder, BeanDefinitionRegistry registry) 方法。

文章目录

1. [1. 1. decorateBeanDefinitionIfRequired](#)
2. [2. 2. decoratelfRequired](#)
3. [3. 3. 小结](#)