



当前位置: Java 技术驿站 (<http://cmsblogs.com>) > 死磕Java (<http://cmsblogs.com/?cat=189>) > 死磕 Spring (<http://cmsblogs.com/?cat=206>) > 正文

【死磕 Spring】—— IOC 之解析 bean 标签：解析自定义标签 (<http://cmsblogs.com/?p=2756>)

2018-09-26 分类: 死磕 Spring (<http://cmsblogs.com/?cat=206>) 阅读(5995) 评论(2)

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

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，**勿忘初心**。processBeanDefinition() 负责 Bean 标签的解析，在解析过程中首先调用 BeanDefinitionParserDelegate.parseBeanDefinitionElement() 完成默认标签的解析，如果解析成功（返回的 bdHolder != null），则首先调用 BeanDefinitionParserDelegate.decorateBeanDefinitionIfRequired() 完成自定义标签元素解析，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。


```
public BeanDefinitionHolder decorateBeanDefinitionIfRequired(Element ele, BeanDefinitionHolder definitionHolder) {  
    return decorateBeanDefinitionIfRequired(ele, definitionHolder, null);  
}
```

调用 decorateBeanDefinitionIfRequired()：



```
public BeanDefinitionHolder decorateBeanDefinitionIfRequired(  
    Element ele, BeanDefinitionHolder definitionHolder, @Nullable BeanDefinition containingBd) {  
  
    BeanDefinitionHolder finalDefinition = definitionHolder;  
  
    // 遍历节点，查看是否有适用于装饰的属性  
    NamedNodeMap attributes = ele.getAttributes();  
    for (int i = 0; i < attributes.getLength(); i++) {  
        Node node = attributes.item(i);  
        finalDefinition = decorateIfRequired(node, finalDefinition, containingBd);  
    }  
  
    // 遍历子节点，查看是否有适用于修饰的子元素  
    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;  
}
```

遍历节点（子节点），调用 `decorateIfRequired()` 装饰节点（子节点）。

public BeanDefinitionHolder decorateIfRequired(Node node, BeanDefinitionHolder originalDef, @Nullable BeanDefinition containingBd) {

```

    Node node, BeanDefinitionHolder originalDef, @Nullable BeanDefinition containingBd) {
        // 获取自定义标签的命名空间

```

```

        String namespaceUri = getNamespaceUri(node);

```

```

        // 过滤掉默认命名空间

```

```

        if (namespaceUri != null && !isDefaultNamespace(namespaceUri)) {

```

```

            // 获取相应的处理器

```

```

            NamespaceHandler handler = this.readerContext.getNamespaceHandlerResolver().resolve(namespace

```

```

            Uri);

```

```

            if (handler != null) {

```

```

                // 进行装饰处理

```

```

                BeanDefinitionHolder decorated =

```

```

                    handler.decorate(node, originalDef, new ParserContext(this.readerContext, this, c

```

```

                    ontainingBd));

```

```

                if (decorated != null) {

```

```

                    return decorated;

```

```

                }

```

```

            }

```

```

            else if (namespaceUri.startsWith("http://www.springframework.org/")) {

```

```

                error("Unable to locate Spring NamespaceHandler for XML schema namespace [" + namespaceUr
                i + "]", node);

```

```

            }

```

```

            else {

```

```

                if (logger.isDebugEnabled()) {

```

```

                    logger.debug("No Spring NamespaceHandler found for XML schema namespace [" + namespace

```

```

                    eUri + "]);

```

```

                }

```

```

            }

```

```

        }

```

```

        return originalDef;

```

```

    }

```

首先获取自定义标签的命名空间，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，最后调用处理器的 `decorate()` 进行装饰处理。具体的装饰过程这里不进行讲述，在后面分析自定义标签时会做详细说明。至此，Bean 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.parseBeanDefinitions()`。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()` 实现，自定义标签由 `parseCustomElement()` 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()` 方法实现。`processBeanDefinition()` 开始进入解析核心工作，分为三步：

1. 解析默认标签： `BeanDefinitionParserDelegate.parseBeanDefinitionElement()`
2. 解析默认标签下的自定义标签： `BeanDefinitionParserDelegate.decorateBeanDefinitionIfRequired()`
3. 注册解析的 `BeanDefinition`： `BeanDefinitionReaderUtils.registerBeanDefinition`

在默认标签解析过程中，核心工作由 `parseBeanDefinitionElement()` 方法实现，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。