

【死磕 Spring】—— loC 之解析Bean：解析 import 标签

本文主要基于 Spring 5.0.6.RELEASE

摘要: 原创出处 <http://cmsblogs.com/?p=2724> 「小明哥」，谢谢！

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

在博客【死磕 Spring】—— loC 之注册 BeanDefinitions 中分析到，Spring 中有两种解析 Bean 的方式：

- 如果根节点或者子节点采用默认命名空间的话，则调用 `#parseDefaultElement(...)` 方法，进行默认标签解析
- 否则，调用 `BeanDefinitionParserDelegate#parseCustomElement(...)` 方法，进行自定义解析。

所以，以下博客就这两个方法进行详细分析说明。而本文，先从默认标签解析过程开始。代码如下：

```
// DefaultBeanDefinitionDocumentReader.java

public static final String IMPORT_ELEMENT = "import";
public static final String ALIAS_ATTRIBUTE = "alias";
public static final String BEAN_ELEMENT =
    BeanDefinitionParserDelegate.BEAN_ELEMENT;
public static final String NESTED_BEANS_ELEMENT = "beans";

private void parseDefaultElement(Element ele, BeanDefinitionParserDelegate
delegate) {
    if (delegate.nodeNameEquals(ele, IMPORT_ELEMENT)) { // import
        importBeanDefinitionResource(ele);
    } else if (delegate.nodeNameEquals(ele, ALIAS_ELEMENT)) { // alias
        processAliasRegistration(ele);
    } else if (delegate.nodeNameEquals(ele, BEAN_ELEMENT)) { // bean
        processBeanDefinition(ele, delegate);
    } else if (delegate.nodeNameEquals(ele, NESTED_BEANS_ELEMENT)) { //
beans
        // recurse
        doRegisterBeanDefinitions(ele);
    }
}
```

该方法的功能一目了然，分别是对四种不同的标签进行解析，分别是 `import`、`alias`、`bean`、`beans`。咱们从第一个标签 `import` 开始。

1. import 示例

经历过 Spring 配置文件的小伙伴都知道，如果工程比较大，配置文件的维护会让人觉得恐怖，文件太多了，想象将所有的配置都放在一个 `spring.xml` 配置文件中，哪种后怕感是不是很明显？

所有针对这种情况 Spring 提供了一个分模块的思路，利用 `import` 标签，例如我们可以构造一个这样的 `spring.xml`。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">

    <import resource="spring-student.xml"/>

    <import resource="spring-student-dtd.xml"/>

</beans>
```

`spring.xml` 配置文件中，使用 `import` 标签的方式导入其他模块的配置文件。

- 如果有配置需要修改直接修改相应配置文件即可。
- 若有新的模块需要引入直接增加 `import` 即可。

这样大大简化了配置后期维护的复杂度，同时也易于管理。

2. importBeanDefinitionResource

Spring 使用 `#importBeanDefinitionResource(Element ele)` 方法，完成对 `import` 标签的解析。

```
// DefaultBeanDefinitionDocumentReader.java

/**
 * Parse an "import" element and load the bean definitions
 * from the given resource into the bean factory.
 */
protected void importBeanDefinitionResource(Element ele) {
    // <1> 获取 resource 的属性值
    String location = ele.getAttribute(RESOURCE_ATTRIBUTE);
    // 为空，直接退出
    if (!StringUtils.hasText(location)) {
        getReaderContext().error("Resource location must not be empty", ele);
    }
    // 使用 problemReporter 报错
    return;
}

// <2> 解析系统属性，格式如 : "${user.dir}"
// Resolve system properties: e.g. "${user.dir}"
location =
getReaderContext().getEnvironment().resolveRequiredPlaceholders(location);

// 实际 Resource 集合，即 import 的地址，有哪些 Resource 资源
Set<Resource> actualResources = new LinkedHashSet<>(4);

// <3> 判断 location 是相对路径还是绝对路径
// Discover whether the location is an absolute or relative URI
boolean absoluteLocation = false;
try {
    absoluteLocation = ResourcePatternUtils.isUrl(location) ||
ResourceUtils.toURI(location).isAbsolute();
} catch (URISyntaxException ex) {
```

```

        // cannot convert to an URI, considering the location relative
        // unless it is the well-known Spring prefix "classpath*:"
    }

    // Absolute or relative?
    // <4> 绝对路径
    if (absoluteLocation) {
        try {
            // 添加配置文件地址的 Resource 到 actualResources 中，并加载相应的
            BeanDefinition 们
            int importCount =
getReaderContext().getReader().loadBeanDefinitions(location, actualResources);
            if (logger.isTraceEnabled()) {
                logger.trace("Imported " + importCount + " bean definitions
from URL location [" + location + "]");
            }
        } catch (BeanDefinitionStoreException ex) {
            getReaderContext().error(
                "Failed to import bean definitions from URL location [" +
location + "]", ele, ex);
        }
        // <5> 相对路径
    } else {
        // No URL -> considering resource location as relative to the current
file.
        try {
            int importCount;
            // 创建相对地址的 Resource
            Resource relativeResource =
getReaderContext().getResource().createRelative(location);
            // 存在
            if (relativeResource.exists()) {
                // 加载 relativeResource 中的 BeanDefinition 们
                importCount =
getReaderContext().getReader().loadBeanDefinitions(relativeResource);
                // 添加到 actualResources 中
                actualResources.add(relativeResource);
            } // 不存在
        } else {
            // 获得根路径地址
            String baseLocation =
getReaderContext().getResource().getURL().toString();
            // 添加配置文件地址的 Resource 到 actualResources 中，并加载相应的
            BeanDefinition 们
            importCount =
getReaderContext().getReader().loadBeanDefinitions(
                StringUtils.applyRelativePath(baseLocation, location)
/* 计算绝对路径 */ , actualResources);
        }
        if (logger.isTraceEnabled()) {
            logger.trace("Imported " + importCount + " bean definitions
from relative location [" + location + "]");
        }
    } catch (IOException ex) {
        getReaderContext().error("Failed to resolve current resource
location", ele, ex);
    } catch (BeanDefinitionStoreException ex) {
        getReaderContext().error(
            "Failed to import bean definitions from relative location
[" + location + "]", ele, ex);
    }
}

// <6> 解析成功后，进行监听器激活处理
Resource[] actResArray = actualResources.toArray(new Resource[0]);

```

```

        getReaderContext().fireImportProcessed(location, actResArray,
        extractSource(ele));
    }

```

解析 `import` 标签的过程较为清晰，整个过程如下：

- <1> 处，获取 `source` 属性的值，该值表示资源的路径。
 - <2> 处，解析路径中的系统属性，如 `"${user.dir}"`。
- <3> 处，判断资源路径 `location` 是绝对路径还是相对路径。详细解析，见「[2.1 判断路径](#)」。
 - <4> 处，如果是绝对路径，则调递归调用 `Bean` 的解析过程，进行另一次的解析。详细解析，见「[2.2 处理绝对路径](#)」。
 - <5> 处，如果是相对路径，则先计算出绝对路径得到 `Resource`，然后进行解析。详细解析，见「[2.3 处理相对路径](#)」。
- <6> 处，通知监听器，完成解析。

2.1 判断路径

通过以下代码，来判断 `location` 是为相对路径还是绝对路径：

```

absoluteLocation = ResourcePatternUtils.isUrl(location) // <1>
|| ResourceUtils.toURI(location).isAbsolute(); // <2>

```

判断绝对路径的规则如下：

- <1> 以 `classpath*`：或者 `classpath`：开头的为绝对路径。
- <1> 能够通过该 `location` 构建出 `java.net.URL` 为绝对路径。
- <2> 根据 `location` 构造 `java.net.URI` 判断调用 `#isAbsolute()` 方法，判断是否为绝对路径。

2.2 处理绝对路径

如果 `location` 为绝对路径，则调用 `#loadBeanDefinitions(String location, Set<Resource> actualResources)`，方法。该方法在 `org.springframework.beans.factory.support.AbstractBeanDefinitionReader` 中定义，代码如下：

```

/**
 * Load bean definitions from the specified resource location.
 * <p>The location can also be a location pattern, provided that the
 * ResourceLoader of this bean definition reader is a ResourcePatternResolver.
 * @param location the resource location, to be loaded with the ResourceLoader
 * (or ResourcePatternResolver) of this bean definition reader
 * @param actualResources a Set to be filled with the actual Resource objects
 * that have been resolved during the loading process. May be {@code null}
 * to indicate that the caller is not interested in those Resource objects.
 * @return the number of bean definitions found
 * @throws BeanDefinitionStoreException in case of loading or parsing errors
 * @see #getResourceLoader()
 * @see #loadBeanDefinitions(org.springframework.core.io.Resource)
 * @see #loadBeanDefinitions(org.springframework.core.io.Resource[])
 */
public int loadBeanDefinitions(String location, @Nullable Set<Resource>

```

```

actualResources) throws BeanDefinitionStoreException {
    // 获得 ResourceLoader 对象
    ResourceLoader resourceLoader = getResourceLoader();
    if (resourceLoader == null) {
        throw new BeanDefinitionStoreException(
            "Cannot load bean definitions from location [" + location +
    "]: no ResourceLoader available");
    }

    if (resourceLoader instanceof ResourcePatternResolver) {
        // Resource pattern matching available.
        try {
            // 获得 Resource 数组，因为 Pattern 模式匹配下，可能有多个 Resource 。例
            如说，Ant 风格的 location
            Resource[] resources = ((ResourcePatternResolver)
            resourceLoader).getResources(location);
            // 加载 BeanDefinition 们
            int count = loadBeanDefinitions(resources);
            // 添加到 actualResources 中
            if (actualResources != null) {
                Collections.addAll(actualResources, resources);
            }
            if (logger.isTraceEnabled()) {
                logger.trace("Loaded " + count + " bean definitions from
            location pattern [" + location + "]");
            }
            return count;
        } catch (IOException ex) {
            throw new BeanDefinitionStoreException(
                "Could not resolve bean definition resource pattern [" +
            location + "]", ex);
        }
    } else {
        // Can only load single resources by absolute URL.
        // 获得 Resource 对象，
        Resource resource = resourceLoader.getResource(location);
        // 加载 BeanDefinition 们
        int count = loadBeanDefinitions(resource);
        // 添加到 actualResources 中
        if (actualResources != null) {
            actualResources.add(resource);
        }
        if (logger.isTraceEnabled()) {
            logger.trace("Loaded " + count + " bean definitions from location
        [" + location + "]");
        }
        return count;
    }
}

```

整个逻辑比较简单：

- 首先，获取 ResourceLoader 对象。
- 然后，根据不同的 ResourceLoader 执行不同的逻辑，主要是可能存在多个 Resource 。
- 最终，都会回归到 XmlBeanDefinitionReader#loadBeanDefinitions(Resource... resources) 方法，所以这是一个递归的过程。
- 另外，获得到的 Resource 的对象或数组，都会添加到 actualResources 中。

2.3 处理相对路径

如果 `location` 是相对路径，则会根据相应的 `Resource` 计算出相应的相对路径的 `Resource` 对象，然后：

- 若该 `Resource` 存在，则调用 `XmlBeanDefinitionReader#loadBeanDefinitions()` 方法，进行 `BeanDefinition` 加载。
- 否则，构造一个绝对 `location` (即 `StringUtils.applyRelativePath(baseLocation, location)` 处的代码)，并调用 `#loadBeanDefinitions(String location, Set<Resource> actualResources)` 方法，与绝对路径过程一样。

3. 小结

至此，`import` 标签解析完毕，整个过程比较清晰明了：获取 `source` 属性值，得到正确的资源路径，然后调用 `XmlBeanDefinitionReader#loadBeanDefinitions(Resource... resources)` 方法，进行递归的 `BeanDefinition` 加载。