



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：解析 import 标签 (<http://cmsblogs.com/?p=2724>)


2018-09-17 分类: 死磕 Spring (<http://cmsblogs.com/?cat=206>) 阅读(11535) 评论(3)

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

在博客【死磕Spring】----- IOC 之 注册 BeanDefinition ()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()` 进行默认标签解析，否则调用 `delegate.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，先从默认标签解析过程开始，源码如下：

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

// 解析xml1中发现有import xml2
// 解析xml1中发现有import xml2
```



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。

import 标签的处理 递归处理xml标签

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。Spring 利用 `importBeanDefinitionResource()` 方法完成对 import 标签的解析。



```

protected void importBeanDefinitionResource(Element ele) {
    // 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);
        return;
    }

    // 解析系统属性, 格式如 : "${user.dir}"
    location = getReaderContext().getEnvironment().resolveRequiredPlaceholders(location);

    Set<Resource> actualResources = new LinkedHashSet<>(4);

    // 判断 location 是相对路径还是绝对路径
    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*:"
    }

    // 绝对路径
    if (absoluteLocation) {
        try {
            // 直接根据地质加载相应的配置文件
            int importCount = getReaderContext().getReader().loadBeanDefinitions(location, actualResources);

            if (logger.isDebugEnabled()) {
                logger.debug("Imported " + importCount + " bean definitions from URL location [" + location + "]");
            }
        }
        catch (BeanDefinitionStoreException ex) {
            getReaderContext().error(
                "Failed to import bean definitions from URL location [" + location + "]", ele, ex
            );
        }
    }
    else {
        // 相对路径则根据相应的地质计算出绝对路径地址
        try {
            int importCount;
            Resource relativeResource = getReaderContext().getResource().createRelative(location);
            if (relativeResource.exists()) {
                importCount = getReaderContext().getReader().loadBeanDefinitions(relativeResource);
                actualResources.add(relativeResource);
            }
        }
    }
}

```



```

else { 件，对bean注册到DefaultListableBeanFactory中的map中
    String baseLocation = getReaderContext().getResource().getURL().toString();
    importCount = getReaderContext().getReader().loadBeanDefinitions(
        StringUtils.applyRelativePath(baseLocation, location), actualResources);
}
if (logger.isDebugEnabled()) {
    logger.debug("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 [" + l
ocation + "]",
        ele, ex);
}
}
// 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 过程较为清晰，整个过程如下：

1. 获取 source 属性的值，该值表示资源的路径
2. 解析路径中的系统属性，如"\${user.dir}"
3. 判断资源路径 location 是绝对路径还是相对路径
4. 如果是绝对路径，则递归调用 Bean 的解析过程，进行另一次的解析，注意解析会把bean注册到map中
5. 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
6. 通知监听器，完成解析

判断路径 方法通过以下方法来判断 location 是为相对路径还是绝对路径：

```
absoluteLocation = ResourcePatternUtils.isUrl(location) || ResourceUtils.toURI(location).isAbsolute();
```

判断绝对路径的规则如下：

- 以 classpath*: 或者 classpath: 开头为绝对路径
- 能够通过该 location 构建出 java.net.URL 为绝对路径
- 根据 location 构造 java.net.URI 判断调用 isAbsolute() 判断是否为绝对路径

绝对路径 如果 location 为绝对路径则调用 loadBeanDefinitions()，该方法在 AbstractBeanDefinitionReader 中定义。

```

public int loadBeanDefinitions(String location, @Nullable Set<Resource> actualResources) throws BeansException {
    ResourceLoader resourceLoader = getResourceLoader();
    if (resourceLoader == null) {
        throw new BeansException("Cannot import bean definitions from location [" + location + "]: no ResourceLoader available");
    }

    if (resourceLoader instanceof ResourcePatternResolver) {
        // Resource pattern matching available.
        try {
            Resource[] resources = ((ResourcePatternResolver) resourceLoader).getResources(location);
            int loadCount = loadBeanDefinitions(resources);
            if (actualResources != null) {
                for (Resource resource : resources) {
                    actualResources.add(resource);
                }
            }
            if (logger.isDebugEnabled()) {
                logger.debug("Loaded " + loadCount + " bean definitions from location pattern [" + location + "];");
            }
            return loadCount;
        } catch (IOException ex) {
            throw new BeansException("Could not resolve bean definition resource pattern [" + location + "]", ex);
        }
    } else {
        // Can only load single resources by absolute URL.
        Resource resource = resourceLoader.getResource(location);
        int loadCount = loadBeanDefinitions(resource);
        if (actualResources != null) {
            actualResources.add(resource);
        }
        if (logger.isDebugEnabled()) {
            logger.debug("Loaded " + loadCount + " bean definitions from location [" + location + "];");
        }
        return loadCount;
    }
}

```

整个逻辑比较简单，首先获取 ResourceLoader，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，但是最终都会回归到 XmlBeanDefinitionReader.loadBeanDefinitions()，所以这是一个递归的过程。**相对路径** 如果是相对路径则会根据相应的 Resource 计算出相应的绝对路径，然后根据该路径构造一个 Resource，若该 Resource 存在，则调用 XmlBeanDefinitionReader.loadBeanDefinitions() 进行 BeanDefinition 加载，否则构造一个绝对 location，调用

`AbstractBeanDefinitionReader.loadBeanDefinitions()` 方法，与绝对路径过程一样。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 `loadBeanDefinitions()` 方法进行递归的 `BeanDefinition` 加载

👍 赞(27)

¥ 打赏

【公告】版权声明 (http://cmsblogs.com/?page_id=1908)

标签： Spring源码研究 (<http://cmsblogs.com/?tag=spring%e6%ba%90%e7%a0%81%e7%a0%94%e7%a9%b6>)

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>)

👤 chenssy (<http://cmsblogs.com/?author=1>)

不想当厨师的程序员不是好的架构师....

上一篇

Java 中的 try catch 影响性能吗?
(<http://cmsblogs.com/?p=2718>)

下一篇

6个实例详解如何把if-else代码重构成高质量代码
(<http://cmsblogs.com/?p=2727>)

- 【死磕 Redis】—— 如何排查 Redis 中的慢查询 (<http://cmsblogs.com/?p=18352>)
- 【死磕 Redis】—— 发布与订阅 (<http://cmsblogs.com/?p=18348>)
- 【死磕 Redis】—— 布隆过滤器 (<http://cmsblogs.com/?p=18346>)
- 【死磕 Redis】—— 理解 pipeline 管道 (<http://cmsblogs.com/?p=18344>)
- 【死磕 Redis】—— 事务 (<http://cmsblogs.com/?p=18340>)
- 【死磕 Redis】—— Redis 的线程模型 (<http://cmsblogs.com/?p=18337>)
- 【死磕 Redis】—— Redis 通信协议 RESP (<http://cmsblogs.com/?p=18334>)
- 【死磕 Redis】—— 开篇 (<http://cmsblogs.com/?p=18332>)
- 【死磕 Spring】—— IOC 总结 (<http://cmsblogs.com/?p=4047>)
- 【死磕 Spring】—— 4 张图带你读懂 Spring IOC 的世界 (<http://cmsblogs.com/?p=4045>)
- 【死磕 Spring】—— 深入分析 ApplicationContext 的 refresh() (<http://cmsblogs.com/?p=4043>)
- 【死磕 Spring】—— ApplicationContext 相关接口架构分析 (<http://cmsblogs.com/?p=4036>)
- 【死磕 Spring】—— IOC 之 分析 bean 的生命周期 (<http://cmsblogs.com/?p=4034>)
- 【死磕 Spring】—— Spring 的环境&属性：PropertySource、Environment、Profile (<http://cmsblogs.com/?p=4032>)
- 【死磕 Spring】—— IOC 之 BeanDefinition 注册机：BeanDefinitionRegistry (<http://cmsblogs.com/?p=4026>)