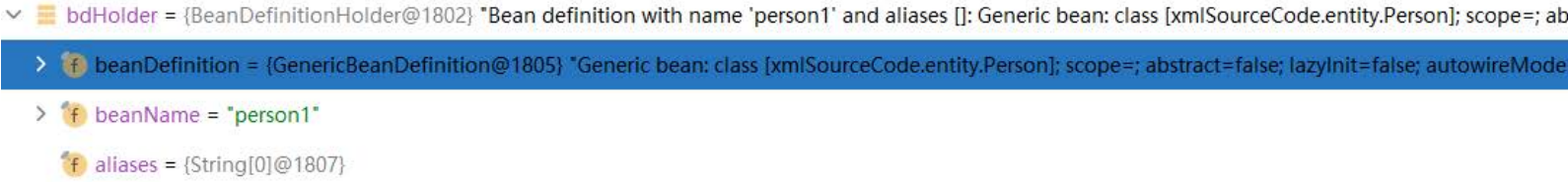
【死磕 Spring】—— IOC 之解析 bean 标签：开启解析进程 (http://cmsblogs.com/?p=2731)

2018-09-19 分类：死磕 Spring (http://cmsblogs.com/?cat=206) 阅读(9017) 评论(0)

import 标签解析完毕了，再看 Spring 中最复杂也是最重要的标签 bean 标签的解析过程。 在方法 `parseDefaultElement()` 中，如果遇到标签 为 bean 则调用 `processBeanDefinition()` 方法进行 bean 标签解析，如下：

解析Element到BeanDefinitionHolder对象：内部的beanDefinition的scope和depentOn为空null




```
    protected void processBeanDefinition(Element ele, BeanDefinitionParserDelegate delegate) {
        BeanDefinitionHolder bdHolder = delegate.parseBeanDefinitionElement(ele);
        if (bdHolder != null) {
            bdHolder = delegate.decorateBeanDefinitionIfRequired(ele, bdHolder);
            try {
                // Register the final decorated instance.
                BeanDefinitionReaderUtils.registerBeanDefinition(bdHolder, getReaderContext().getRegistry
());
            }
            catch (BeanDefinitionStoreException ex) {
                getReaderContext().error("Failed to register bean definition with name '" +
                        bdHolder.getBeanName() + "'", ele, ex);
            }
            // Send registration event.
            getReaderContext().fireComponentRegistered(new BeanComponentDefinition(bdHolder));
        }
    }
```

整个过程分为四个步骤

1. 调用 `BeanDefinitionParserDelegate.parseBeanDefinitionElement()` 进行元素解析，解析过程中如果失败，返回 null，错误由 `ProblemReporter` 处理。如果解析成功则返回 BeanDefinitionHolder 实例 bdHolder。BeanDefinitionHolder 为持有 beanName 和 alias 的 BeanDefinition。
2. 若实例 bdHolder 不为空，则调用 `BeanDefinitionParserDelegate.decorateBeanDefinitionIfRequired()` 进行自定义标签处理
3. 解析完成后，则调用 `BeanDefinitionReaderUtils.registerBeanDefinition()` 对 bdHolder 进行注册
4. 发出响应事件，通知相关的监听器，完成 Bean 标签解析

先看方法 `parseBeanDefinitionElement()`，如下：

```
public BeanDefinitionHolder parseBeanDefinitionElement(Element ele, @Nullable BeanDefinition containingBean) {
        // 解析 ID 属性
        String id = ele.getAttribute(ID_ATTRIBUTE);
        // 解析 name 属性
        String nameAttr = ele.getAttribute(NAME_ATTRIBUTE);

        // 分割 name 属性
        List<String> aliases = new ArrayList<>();
        if (StringUtils.hasLength(nameAttr)) {
            String[] nameArr = StringUtils.tokenizeToStringArray(nameAttr, MULTI_VALUE_ATTRIBUTE_DELIMITERS);
            aliases.addAll(Arrays.asList(nameArr));
        }

        String beanName = id;
        if (!StringUtils.hasText(beanName) && !aliases.isEmpty()) {
            beanName = aliases.remove(0);
            if (logger.isDebugEnabled()) {
                logger.debug("No XML 'id' specified - using '" + beanName +
                        "' as bean name and " + aliases + " as aliases");
            }
        }

        // 检查 name 的唯一性
        if (containingBean == null) {
            checkNameUniqueness(beanName, aliases, ele);
        }

        // 解析 属性，构造 AbstractBeanDefinition
        AbstractBeanDefinition beanDefinition = parseBeanDefinitionElement(ele, beanName, containingBean);
        if (beanDefinition != null) {
            // 如果 beanName 不存在，则根据条件构造一个 beanName
            if (!StringUtils.hasText(beanName)) {
                try {
                    if (containingBean != null) {
                        beanName = BeanDefinitionReaderUtils.generateBeanName(
                                beanDefinition, this.readerContext.getRegistry(), true);
                    }
                    else {
                        beanName = this.readerContext.generateBeanName(beanDefinition);
                        String beanClassName = beanDefinition.getBeanClassName();
                        if (beanClassName != null &&
                                beanName.startsWith(beanClassName) && beanName.length() > beanClassName.length() &&
                                !this.readerContext.getRegistry().isBeanNameInUse(beanClassName)) {
                            aliases.add(beanClassName);
                        }
                    }
                    if (logger.isDebugEnabled()) {
```

```
                    logger.debug("Neither XML 'id' nor 'name' specified - " +
                            "using generated bean name [" + beanName + "]");
                }
            }
            catch (Exception ex) {
                error(ex.getMessage(), ele);
                return null;
            }
        }
        String[] aliasesArray = StringUtils.toStringArray(aliases);

        // 封装 BeanDefinitionHolder
        return new BeanDefinitionHolder(beanDefinition, beanName, aliasesArray);
    }

    return null;
}
```

这个方法还没有对 Bean 标签进行解析，只是在解析动作之前做了一些功能架构，主要的工作有：

- 解析 id、name 属性，确定 alias 集合，检测 beanName 是否唯一
- 调用方法 `parseBeanDefinitionElement()` 对属性进行解析并封装成 GenericBeanDefinition 实例 beanDefinition
- 根据所获取的信息（beanName、aliases、beanDefinition）构造 BeanDefinitionHolder 实例对象并返回。

这里有必要说下 beanName 的命名规则：如果 id 不为空，则 beanName = id；如果 id 为空，但是 alias 不空，则 beanName 为 alias 的第一个元素，如果两者都为空，则根据默认规则来设置 beanName。 上面三个步骤第二个步骤为核心方法，它主要承担解析 Bean 标签中所有的属性值。如下：

```
public AbstractBeanDefinition parseBeanDefinitionElement(
        Element ele, String beanName, @Nullable BeanDefinition containingBean) {

    this.parseState.push(new BeanEntry(beanName));

    String className = null;
    // 解析 class 属性
    if (ele.hasAttribute(CLASS_ATTRIBUTE)) {
        className = ele.getAttribute(CLASS_ATTRIBUTE).trim();
    }
    String parent = null;

    // 解析 parent 属性
    if (ele.hasAttribute(PARENT_ATTRIBUTE)) {
        parent = ele.getAttribute(PARENT_ATTRIBUTE);
    }

    try {

        // 创建用于承载属性的 GenericBeanDefinition 实例
        AbstractBeanDefinition bd = createBeanDefinition(className, parent);

        // 解析默认 bean 的各种属性
        parseBeanDefinitionAttributes(ele, beanName, containingBean, bd);

        // 提取 description
        bd.setDescription(DomUtils.getChildElementValueByTagName(ele, DESCRIPTION_ELEMENT));

        // 解析元数据
        parseMetaElements(ele, bd);

        // 解析 lookup-method 属性
        parseLookupOverrideSubElements(ele, bd.getMethodOverrides());

        // 解析 replaced-method 属性
        parseReplacedMethodSubElements(ele, bd.getMethodOverrides());

        // 解析构造函数参数
        parseConstructorArgElements(ele, bd);

        // 解析 property 子元素
        parsePropertyElements(ele, bd);

        // 解析 qualifier 子元素
        parseQualifierElements(ele, bd);

        bd.setResource(this.readerContext.getResource());
        bd.setSource(extractSource(ele));

        return bd;
    }
```

```
        catch (ClassNotFoundException ex) {
            error("Bean class [" + className + "] not found", ele, ex);
        }
        catch (NoClassDefFoundError err) {
            error("Class that bean class [" + className + "] depends on not found", ele, err);
        }
        catch (Throwable ex) {
            error("Unexpected failure during bean definition parsing", ele, ex);
        }
        finally {
            this.parseState.pop();
        }

        return null;
    }
```

到这里，Bean 标签的所有属性我们都可以看到其解析的过程，也就说到这里我们已经解析一个基本可用的 BeanDefinition。 由于解析过程较为漫长，篇幅较大，为了更好的观看体验，将这篇博文进行拆分。下篇博客主要介绍 BeanDefinition，以及解析默认 Bean 的过程（ `parseBeanDefinitionAttributes()` ） -- - 【死磕 Spring】—— IOC 之解析Bean：解析 import 标签 (http://cmsblogs.com/?p=2724)

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)

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
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)

下一篇
【死磕 Spring】—— IOC 之解析 bean 标签：BeanDefinition (http://cmsblogs.com/?p=2734)

- 【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)