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

2018-09-19 分类：死磕 Spring (http://cmsblogs.com/?cat=206) 阅读(9864) 评论(4)

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

前面历经千辛万苦终于到达解析 bean 标签步骤来了，解析 bean 标签的过程其实就是构造一个 BeanDefinition 对象的过程，`<bean>` 元素标签拥有的配置属性，BeanDefinition 均提供了相应的属性，与之一一对应。所以我们有必要对 BeanDefinition 有一个整体的认识。

BeanDefinition

BeanDefinition 是一个接口，它描述了一个 Bean 实例，包括属性值、构造方法值和继承自它的类的更多信息。它继承 AttributeAccessor 和 BeanMetadataElement 接口。两个接口定义如下：

- AttributeAccessor ： 定义了与其它对象的（元数据）进行连接和访问的约定，即对属性的修改，包括获取、设置、删除。
- BeanMetadataElement：Bean 元对象持有的配置元素可以通过 `getSource()` 方法来获取。

BeanDefinition 整个结构如下图：

spring-201806201001 (http://cmsblogs.qiniudn.com/spring-201806201001.png) 我们常用的三个实现类有：ChildBeanDefinition 、GenericBeanDefinition 、RootBeanDefinition ，三者都继承 AbstractBeanDefinition 。如果配置文件中定义了父 `<bean>` 和子 `<bean>` ，则父 `<bean>` 用 RootBeanDefinition 表示，子 `<bean>` 用 ChildBeanDefinition 表示，而没有父 `<bean>` 的就使用 RootBeanDefinition 表示。GenericBeanDefinition 为一站式服务类。AbstractBeanDefinition对三个子类共同的类信息进行抽象。

解析 Bean 标签

在 `BeanDefinitionParserDelegate.parseBeanDefinitionElement()` 中完成 Bean 的解析，返回的是一个已经完成对 `<bean>` 标签解析的 BeanDefinition 实例。在该方法内部，首先调用 `createBeanDefinition()` 方法创建一个用于承载属性的 GenericBeanDefinition 实例，如下：

```
    protected AbstractBeanDefinition createBeanDefinition(@Nullable String className, @Nullable String pa
rentName)
                                                    1、创建出一个空BeanDefinition对象
            throws ClassNotFoundException {
        return BeanDefinitionReaderUtils.createBeanDefinition(
                parentName, className, this.readerContext.getBeanClassLoader());
    }
```

委托 BeanDefinitionReaderUtils 创建，如下：

```
    public static AbstractBeanDefinition createBeanDefinition(
            @Nullable String parentName, @Nullable String className, @Nullable ClassLoader classLoader) t
hrows ClassNotFoundException {

        GenericBeanDefinition bd = new GenericBeanDefinition();
        bd.setParentName(parentName);
        if (className != null) {
            if (classLoader != null) {
                bd.setBeanClass(ClassUtils.forName(className, classLoader));
            }
            else {
                bd.setBeanClassName(className);
            }
        }
        return bd;
    }
```

该方法主要是设置 parentName 、className、classLoader。 创建完 GenericBeanDefinition 实例后，再调用 `parseBeanDefinitionAttributes()` ，该方法将创建好的 GenericBeanDefinition 实例当做参数，对 Bean 标签的所有属性进行解析，如下：

```
public AbstractBeanDefinition parseBeanDefinitionAttributes(Element ele, String beanName,
                                                            @Nullable BeanDefinition containingBean,
AbstractBeanDefinition bd) {
        // 解析 scope 标签
        if (ele.hasAttribute(SINGLETON_ATTRIBUTE)) {
            error("Old 1.x 'singleton' attribute in use - upgrade to 'scope' declaration", ele);
        }
        else if (ele.hasAttribute(SCOPE_ATTRIBUTE)) {
            bd.setScope(ele.getAttribute(SCOPE_ATTRIBUTE));
        }
        else if (containingBean != null) {
            // Take default from containing bean in case of an inner bean definition.
            bd.setScope(containingBean.getScope());
        }

        // 解析 abstract 标签
        if (ele.hasAttribute(ABSTRACT_ATTRIBUTE)) {
            bd.setAbstract(TRUE_VALUE.equals(ele.getAttribute(ABSTRACT_ATTRIBUTE)));
        }

        // 解析 lazy-init 标签
        String lazyInit = ele.getAttribute(LAZY_INIT_ATTRIBUTE);
        if (DEFAULT_VALUE.equals(lazyInit)) {
            lazyInit = this.defaults.getLazyInit();
        }
        bd.setLazyInit(TRUE_VALUE.equals(lazyInit));

        // 解析 autowire 标签
        String autowire = ele.getAttribute(AUTOWIRE_ATTRIBUTE);
        bd.setAutowireMode(getAutowireMode(autowire));

        // 解析 depends-on 标签
        if (ele.hasAttribute(DEPENDS_ON_ATTRIBUTE)) {
            String dependsOn = ele.getAttribute(DEPENDS_ON_ATTRIBUTE);
             bd.setDependsOn(StringUtils.tokenizeToStringArray(dependsOn, MULTI_VALUE_ATTRIBUTE_DELIMITERS
));
        }

        // 解析 autowire-candidate 标签
        String autowireCandidate = ele.getAttribute(AUTOWIRE_CANDIDATE_ATTRIBUTE);
        if ("".equals(autowireCandidate) || DEFAULT_VALUE.equals(autowireCandidate)) {
            String candidatePattern = this.defaults.getAutowireCandidates();
            if (candidatePattern != null) {
                String[] patterns = StringUtils.commaDelimitedListToStringArray(candidatePattern);
                bd.setAutowireCandidate(PatternMatchUtils.simpleMatch(patterns, beanName));
            }
        }
        else {
            bd.setAutowireCandidate(TRUE_VALUE.equals(autowireCandidate));
        }
```

```
        // 解析 primay 标签
        if (ele.hasAttribute(PRIMARY_ATTRIBUTE)) {
            bd.setPrimary(TRUE_VALUE.equals(ele.getAttribute(PRIMARY_ATTRIBUTE)));
        }

        // 解析 init-method 标签
        if (ele.hasAttribute(INIT_METHOD_ATTRIBUTE)) {
            String initMethodName = ele.getAttribute(INIT_METHOD_ATTRIBUTE);
            bd.setInitMethodName(initMethodName);
        }
        else if (this.defaults.getInitMethod() != null) {
            bd.setInitMethodName(this.defaults.getInitMethod());
            bd.setEnforceInitMethod(false);
        }

        // 解析 destroy-mothod 标签
        if (ele.hasAttribute(DESTROY_METHOD_ATTRIBUTE)) {
            String destroyMethodName = ele.getAttribute(DESTROY_METHOD_ATTRIBUTE);
            bd.setDestroyMethodName(destroyMethodName);
        }
        else if (this.defaults.getDestroyMethod() != null) {
            bd.setDestroyMethodName(this.defaults.getDestroyMethod());
            bd.setEnforceDestroyMethod(false);
        }

        // 解析 factory-method 标签
        if (ele.hasAttribute(FACTORY_METHOD_ATTRIBUTE)) {
            bd.setFactoryMethodName(ele.getAttribute(FACTORY_METHOD_ATTRIBUTE));
        }
        if (ele.hasAttribute(FACTORY_BEAN_ATTRIBUTE)) {
            bd.setFactoryBeanName(ele.getAttribute(FACTORY_BEAN_ATTRIBUTE));
        }

        return bd;
    }
```

从上面代码我们可以清晰地看到对 Bean 标签属性的解析，这些属性我们在工作中都或多或少用到过。 完成 Bean 标签基本属性解析后，会依次调用 `parseMetaElements()` 、 `parseLookupOverrideSubElements()` 、 `parseReplacedMethodSubElements()` 对子元素 meta、lookup-method、replace-method 完成解析。下篇博文将会对这三个子元素进行详细说明。

- 【死磕 Spring】—— IOC 之解析Bean：解析 import 标签 (http://cmsblogs.com/?p=2724)
- 【死磕 Spring】----- IOC 之解析 bean 标签：开启解析进程 (http://cmsblogs.com/?p=2731)

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

下一篇
【死磕 Spring】—— IOC 之解析 bean 标签：meta、lookup-method、replace-method (http://cmsblogs.com/?p=2736)

- 【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)