



[返回首页](#)

芋道源码 —— 知识星球

我是一段不羁的公告！

记得给芳芳这 3 个项目加油，添加一个 STAR 噢。

<https://github.com/YunaiV/SpringBoot-Labs>

<https://github.com/YunaiV/onemall>

<https://github.com/YunaiV/ruoyi-vue-pro>

2019-03-10

[Spring](#)

【死磕 Spring】—— IoC 之解析 <bean> 标签： meta、lookup-method、replace-method

本文主要基于 Spring 5.0.6.RELEASE

摘要：原创出处 <http://cmsblogs.com/?p=2736> 「小明哥」，谢谢！

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

在上篇博客[【死磕 Spring】—— IoC 之解析 标签: BeanDefinition](#) 中，已经完成了对 <bean> 标签属性的解析工作。这篇博文开始，分析子元素的解析。

完成 bean 标签的基本属性解析后，会依次调用 BeanDefinitionParserDelegate 的 #parseMetaElements(Element ele, BeanMetadataAttributeAccessor attributeAccessor)、#parseLookupOverrideSubElements(Element beanEle, MethodOverrides overrides)、#parseReplacedMethodSubElements(Element beanEle, MethodOverrides overrides) 方法，分别对子元素 meta、lookup-method、replace-method 元素完成解析。三个子元素的作用如下：

<meta>：元数据。

<lookup-method>：Spring 动态改变 bean 里方法的实现。方法执行返回的对象，使用 Spring 内原有的这类对象替换，通过改变方法返回值来动态改变方法。内部实现为使用 cglib 方法，重新生成子类，重写配置的方法和返回对象，达到动态改变的效果。

<replace-method>：Spring 动态改变 bean 里方法的实现。需要改变的方法，使用 Spring 内原有其他类（需要继承接口 org.springframework.beans.factory.support.MethodReplacer）的逻辑，替换这个方法。通过改变方法执行逻辑来动态改变方法。

1. meta 子元素

meta：元数据。当需要使用里面的信息时可以通过 key 获取。

meta 所声明的 key 并不会在 Bean 中体现，只是一个额外的声明，当我们需要使用里面的信息时，通过调用 BeanDefinition 的 #getAttribute(String name) 方法来获取。该子元素的解析过程，代码如下：

```
// BeanDefinitionParserDelegate.java

public void parseMetaElements(Element ele, BeanMetadataAttributeAccessor attributeAccessor) {
    NodeList nl = ele.getChildNodes();
    // 遍历子节点
    for (int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
        // <meta key="special-data" value="special stragey" />
        if (isCandidateElement(node) && nodeNameEquals(node, META_ELEMENT)) { // 标签名为 meta
            Element metaElement = (Element) node;
            String key = metaElement.getAttribute(KEY_ATTRIBUTE); // key
            String value = metaElement.getAttribute(VALUE_ATTRIBUTE); // value
            // 创建 BeanMetadataAttribute 对象
            BeanMetadataAttribute attribute = new BeanMetadataAttribute(key, value);
            attribute.setSource(extractSource(metaElement));
            // 添加到 BeanMetadataAttributeAccessor 中
            attributeAccessor.addMetadataAttribute(attribute);
        }
    }
}
```

解析过程较为简单，获取相应的 key - value 构建 BeanMetadataAttribute 对象，然后调用 BeanMetadataAttributeAccessor#addMetadataAttribute(BeanMetadataAttribute) 方法，添加 BeanMetadataAttribute 加入到 AbstractBeanDefinition 中。

友情提示：

AbstractBeanDefinition 继承 BeanMetadataAttributeAccessor 类
BeanMetadataAttributeAccessor 继承 AttributeAccessorSupport 类。

1.1 addMetadataAttribute

调用 BeanMetadataAttributeAccessor#addMetadataAttribute(BeanMetadataAttribute) 方法，添加 BeanMetadataAttribute 加入到 AbstractBeanDefinition 中。代码如下：

```
// BeanMetadataAttributeAccessor.java

public void addMetadataAttribute(BeanMetadataAttribute attribute) {
    super.setAttribute(attribute.getName(), attribute);
}
```

委托 AttributeAccessorSupport 实现，如下：

```
// AttributeAccessorSupport.java

/** Map with String keys and Object values. */
private final Map<String, Object> attributes = new LinkedHashMap<>();

@Override
public void setAttribute(String name, @Nullable Object value) {
    Assert.notNull(name, "Name must not be null");
    if (value != null) {
        this.attributes.put(name, value);
    } else {

```

```

        removeAttribute(name);
    }
}

```

`org.springframework.core.AttributeAccessorSupport`，是接口 `AttributeAccessor` 的实现者。
`AttributeAccessor` 接口定义了与其他对象的元数据进行连接和访问的约定，可以通过该接口对属性进行获取、设置、删除操作。

1.2 getAttribute

设置元数据后，则可以通过调用 `BeanDefinition` 的 `#getAttribute(String name)` 方法来获取属性。代码如下：

```

// AttributeAccessorSupport.java

/** Map with String keys and Object values. */
private final Map<String, Object> attributes = new LinkedHashMap<>();

@Override
@Nullable
public Object getAttribute(String name) {
    Assert.notNull(name, "Name must not be null");
    return this.attributes.get(name);
}

```

2. lookup-method 子元素

`lookup-method`：获取器注入，是把一个方法声明为返回某种类型的 `bean` 但实际要返回的 `bean` 是在配置文件里面配置的。该方法可以用于设计一些可插拔的功能上，解除程序依赖。

2.1 示例

直接上例子：

```

public interface Car {

    void display();

}

public class Bmw implements Car{

    @Override
    public void display() {
        System.out.println("我是 BMW");
    }

}

```

```

public class Hongqi implements Car {

    @Override
    public void display() {
        System.out.println("我是 hongqi");
    }

}

public abstract class Display {

    public void display() {
        getCar().display();
    }

    public abstract Car getCar();

}

public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("classpath:spring.xml");
    Display display = (Display) context.getBean("display");
    display.display();
}

```

XML 配置内容如下:

```

<bean id="display" class="org.springframework.core.test1.Display">
    <lookup-method name="getCar" bean="hongqi"/>
</bean>

```

运行结果为:

```
我是 hongqi
```

如果将 bean="hongqi" 替换为 bean="bmw", 则运行结果变成:

```
我是 BMW
```

2.2 parseLookupOverrideSubElements

看了这个示例, 我们初步了解了 lookup-method 子元素提供的功能了。其解析通过 #parseLookupOverrideSubElements(Element beanEle, MethodOverrides overrides) 方法, 代码如下:

```

// BeanDefinitionParserDelegate.java

public void parseLookupOverrideSubElements(Element beanEle, MethodOverrides overrides) {
    NodeList nl = beanEle.getChildNodes();
    // 遍历子节点
    for (int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
    }
}

```

```

        if (isCandidateElement(node) && nodeNameEquals(node, LOOKUP_METHOD_ELEMENT)) { // 标签名为 lookup-method
            Element ele = (Element) node;
            String methodName = ele.getAttribute(NAME_ATTRIBUTE); // name
            String beanRef = ele.getAttribute(BEAN_ELEMENT); // bean
            // 创建 LookupOverride 对象
            LookupOverride override = new LookupOverride(methodName, beanRef);
            override.setSource(extractSource(ele));
            // 添加到 MethodOverrides 中
            overrides.addOverride(override);
        }
    }
}

```

解析过程和 meta 子元素没有多大区别，同样是解析 methodName、beanRef 构造一个 LookupOverride 对象，然后记录到 AbstractBeanDefinition 中的 methodOverrides 属性中。

在实例化 Bean 的时候，再详细阐述具体的实现过程，这里仅仅只是一个标记作用。

3. replace-method 子元素

replaced-method：可以在运行时调用新的方法替换现有的方法，还能动态的更新原有方法的逻辑。

3.1 示例

该标签使用方法和 lookup-method 标签差不多，只不过替代方法的类需要实现 org.springframework.beans.factory.support.MethodReplacer 接口。如下：

```

public class Method {

    public void display() {
        System.out.println("我是原始方法");
    }

}

public class MethodReplace implements MethodReplacer {

    @Override
    public Object reimplement(Object obj, Method method, Object[] args) throws Throwable {
        System.out.println("我是替换方法");
        return null;
    }

}

public static void main(String[] args) {
    ApplicationContext context = new ClassPathXmlApplicationContext("classpath:spring.xml");
    Method method = (Method) context.getBean("method");
    method.display();
}

```

如果 spring.xml 文件如下：

```

<bean id="methodReplace" class="org.springframework.core.test1.MethodReplace"/>

<bean id="method" class="org.springframework.core.test1.Method"/>

```

则运行结果为：

我是原始方法

增加 `replaced-method` 子元素：

```

<bean id="methodReplace" class="org.springframework.core.test1.MethodReplace"/>

<bean id="method" class="org.springframework.core.test1.Method">

    <replaced-method name="display" replacer="methodReplace"/>

</bean>

```

运行结果为：

我是替换方法

3.2 parseReplacedMethodSubElements

上面代码演示了 `replaced-method` 子元素的用法，其解析通过 `#parseReplacedMethodSubElements(Element beanEle, MethodOverrides overrides)` 方法，代码如下：

```

/**
 * Parse replaced-method sub-elements of the given bean element.
 */
public void parseReplacedMethodSubElements(Element beanEle, MethodOverrides overrides) {
    NodeList nl = beanEle.getChildNodes();
    // 遍历子节点
    for (int i = 0; i < nl.getLength(); i++) {
        Node node = nl.item(i);
        if (isCandidateElement(node) && nodeNameEquals(node, REPLACED_METHOD_ELEMENT)) { // 标签名为 replace-method
            Element replacedMethodEle = (Element) node;
            String name = replacedMethodEle.getAttribute(NAME_ATTRIBUTE); // name
            String callback = replacedMethodEle.getAttribute(REPLACER_ATTRIBUTE); // replacer
            // 创建 ReplaceOverride 对象
            ReplaceOverride replaceOverride = new ReplaceOverride(name, callback);
            // Look for arg-type match elements. 参见 《spring bean中lookup-method属性 replaced-method属性》 http://l
            List<Element> argTypeEles = DomUtils.getChildElementsByTagName(replacedMethodEle, ARG_TYPE_ELEMENT); // a
            for (Element argTypeEle : argTypeEles) {
                String match = argTypeEle.getAttribute(ARG_TYPE_MATCH_ATTRIBUTE); // arg-type 子标签的 match 属性
                match = (StringUtils.hasText(match) ? match : DomUtils.getTextValue(argTypeEle));
                if (StringUtils.hasText(match)) {
                    replaceOverride.addTypeIdIdentifier(match);
                }
            }
        }
    }
}

```

```
        replaceOverride.setSource(extractSource(replacedMethodEle));
        // 添加到 MethodOverrides 中
        overrides.addOverride(replaceOverride);
    }
}
```

该子元素和 `lookup-method` 标签的解析过程差不多，同样是提取 `name` 和 `replacer` 属性构建 `ReplaceOverride` 对象，然后记录到 `AbstractBeanDefinition` 中的 `methodOverrides` 属性中。

在实例化 `Bean` 的时候，再详细阐述具体的实现过程，这里仅仅只是一个标记作用。

4. 小结

对于 `lookup-method` 和 `replaced-method` 两个子元素是如何使用以完成他们所提供的功能，在后续实例化 `Bean` 的时候会做详细说明。

老芳芳：貌似，实际 `Spring` 使用场景中，也很少用这两个标签。

文章目录

1. [1. meta 子元素](#)
 1. [1.1. 1.1 addMetadataAttribute](#)
 2. [1.2. 1.2 getAttribute](#)
2. [2. lookup-method 子元素](#)
 1. [2.1. 2.1 示例](#)
 2. [2.2. 2.2 parseLookupOverrideSubElements](#)
3. [3. replace-method 子元素](#)
 1. [3.1. 3.1 示例](#)
 2. [3.2. 3.2 parseReplacedMethodSubElements](#)
4. [4. 4. 小结](#)