



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 之注册解析的 BeanDefinition (<http://cmsblogs.com/?p=2763>)


2018-09-28 分类: 死磕 Spring (<http://cmsblogs.com/?cat=206>) 阅读(10873) 评论(2)

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

DefaultBeanDefinitionDocumentReader.processBeanDefinition() 完成 Bean 标签解析的核心工作, 如下:

```
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));
    }
}
```

解析工作分为三步: 1、解析默认标签; 2、解析默认标签后下得自定义标签; 3、注册解析后的 BeanDefinition。经过前面两个步骤的解析, 这时的 BeanDefinition 已经可以满足后续的使用要求了, 那么接下来的工作就是将这些 BeanDefinition 进行注册, 也就是完成第三步。注册 BeanDefinition 由 BeanDefinitionReaderUtils.registerBeanDefinition() 完成。如下:

public static void **registerBeanDefinition**( Java技术驿站

BeanDefinitionHolder definitionHolder, BeanDefinitionRegistry registry) throws

BeanDefinitionStoreException { //registry的实现类有DefaultListableBeanFactory，就是把bd注册到该类内部map属性中



// 注册 beanName

String beanName = definitionHolder.getBeanName();

//以key=beanName, value=beanDefinition形式，存放到DefaultListableBeanFactory内部的currentHashMap中beanDefinitionMap，同时把beanName注册到DefaultListableBeanFactory内部的ArrayList中beanDefinitionNames

registry.registerBeanDefinition(beanName, definitionHolder.getBeanDefinition());

// 注册 alias

String[] aliases = definitionHolder.getAliases();

if (aliases != null) {

for (String alias : aliases) {

registry.registerAlias(beanName, alias);

}

}

}

首先通过 beanName 注册 BeanDefinition，然后再注册别名 alias。BeanDefinition 的注册由接口 BeanDefinitionRegistry 定义。通过 **beanName 注册** BeanDefinitionRegistry.registerBeanDefinition() 实现通过 beanName 注册 BeanDefinition，如下：



```
public void registerBeanDefinition(String beanName, BeanDefinition beanDefinition)
    throws BeanDefinitionStoreException {
```



```
// 校验 beanName 与 beanDefinition
Assert.hasText(beanName, "Bean name must not be empty");
Assert.notNull(beanDefinition, "BeanDefinition must not be null");

if (beanDefinition instanceof AbstractBeanDefinition) {
    try {
        // 校验 BeanDefinition
        // 这是注册前的最后一次校验了，主要是对属性 methodOverrides 进行校验
        ((AbstractBeanDefinition) beanDefinition).validate();
    }
    catch (BeanDefinitionValidationException ex) {
        throw new BeanDefinitionStoreException(beanDefinition.getResourceDescription(), beanName,
            "Validation of bean definition failed", ex);
    }
}

BeanDefinition oldBeanDefinition;

// 从缓存中获取指定 beanName 的 BeanDefinition
oldBeanDefinition = this.beanDefinitionMap.get(beanName);
/**
 * 如果存在
 */
if (oldBeanDefinition != null) {
    // 如果存在但是不允许覆盖，抛出异常
    if (!isAllowBeanDefinitionOverriding()) {
        throw new BeanDefinitionStoreException(beanDefinition.getResourceDescription(), beanName,
            "Cannot register bean definition [" + beanDefinition + "] for bean '" + beanName
            + "' : There is already [" + oldBeanDefinition + "] bound.");
    }
    //
    else if (oldBeanDefinition.getRole() < beanDefinition.getRole()) {
        // e.g. was ROLE_APPLICATION, now overriding with ROLE_SUPPORT or ROLE_INFRASTRUCTURE
        if (this.logger.isWarnEnabled()) {
            this.logger.warn("Overriding user-defined bean definition for bean '" + beanName +
                "' with a framework-generated bean definition: replacing [" +
                oldBeanDefinition + "] with [" + beanDefinition + "]");
        }
    }
    // 覆盖 beanDefinition 与 被覆盖的 beanDefinition 不是同类
    else if (!beanDefinition.equals(oldBeanDefinition)) {
        if (this.logger.isInfoEnabled()) {
            this.logger.info("Overriding bean definition for bean '" + beanName +
                "' with a different definition: replacing [" + oldBeanDefinition +
                "] with [" + beanDefinition + "]");
        }
    }
}
```



```

else {
    if (this.logger.isDebugEnabled()) {
        this.logger.debug("Overriding bean definition for bean '" + beanName +
            "' with an equivalent definition: replacing [" + oldBeanDefinition +
            "] with [" + beanDefinition + "]");
    }
}

// 允许覆盖，直接覆盖原有的 BeanDefinition
this.beanDefinitionMap.put(beanName, beanDefinition);
}
/**
 * 不存在
 */
else {

    // 检测创建 Bean 阶段是否已经开启，如果开启了则需要对 beanDefinitionMap 进行并发控制
    if (hasBeanCreationStarted()) {
        // beanDefinitionMap 为全局变量，避免并发情况
        synchronized (this.beanDefinitionMap) {
            //
            this.beanDefinitionMap.put(beanName, beanDefinition);
            List<String> updatedDefinitions = new ArrayList<>(this.beanDefinitionNames.size() + 1);

            updatedDefinitions.addAll(this.beanDefinitionNames);
            updatedDefinitions.add(beanName);
            this.beanDefinitionNames = updatedDefinitions;
            if (this.manualSingletonNames.contains(beanName)) {
                Set<String> updatedSingletons = new LinkedHashSet<>(this.manualSingletonNames);
                updatedSingletons.remove(beanName);
                this.manualSingletonNames = updatedSingletons;
            }
        }
    }
    else {
        // 不会存在并发情况，直接设置
        this.beanDefinitionMap.put(beanName, beanDefinition);
        this.beanDefinitionNames.add(beanName);
        this.manualSingletonNames.remove(beanName);
    }
    this.frozenBeanDefinitionNames = null;
}

if (oldBeanDefinition != null || containsSingleton(beanName)) {
    // 重新设置 beanName 对应的缓存
    resetBeanDefinition(beanName);
}
}
}

```

处理过程如下：

- 首先 BeanDefinition 进行校验，该校验也是注册过程中的最后一次校验了，主要是对 AbstractBeanDefinition 的 methodOverrides 属性进行校验
- 根据 beanName 从缓存中获取 BeanDefinition，如果缓存中存在，则根据 allowBeanDefinitionOverriding 标志来判断是否允许覆盖，如果允许则直接覆盖，否则抛出 BeanDefinitionStoreException 异常
- 若缓存中没有指定 beanName 的 BeanDefinition，则判断当前阶段是否已经开始了 Bean 的创建阶段 ()，如果是，则需要对 beanDefinitionMap 进行加锁控制并发问题，否则直接设置即可。对于 hasBeanCreationStarted() 方法后续做详细介绍，这里不过多阐述。
- 若缓存中存在该 beanName 或者 单利 bean 集合中存在该 beanName，则调用 resetBeanDefinition() 重置 BeanDefinition 缓存。

其实整段代码的核心就在于 `this.beanDefinitionMap.put(beanName, beanDefinition);`。BeanDefinition 的缓存也不是神奇的东西，就是定义 map，key 为 beanName，value 为 BeanDefinition。注册 alias BeanDefinitionRegistry.registerAlias 完成 alias 的注册。

`<bean id="person2" name="person1Alias1 person2Alias2" class="XXX">`

person2有两个别名person1Alias1 和2

```
public void registerAlias(String name, String alias) {
    // 校验 name、alias
    Assert.hasText(name, "'name' must not be empty");
    Assert.hasText(alias, "'alias' must not be empty");
    synchronized (this.aliasMap) {
        // name == alias 则去掉alias
        if (alias.equals(name)) {
            this.aliasMap.remove(alias);
        }
        else {
            // 缓存缓存记录
            String registeredName = this.aliasMap.get(alias);
            if (registeredName != null) {
                // 缓存中的相等，则直接返回
                if (registeredName.equals(name)) {
                    // An existing alias - no need to re-register
                    return;
                }
                // 不允许则抛出异常
                if (!allowAliasOverriding()) {
                    throw new IllegalStateException("Cannot register alias '" + alias + "' for name '" +
                        name + "': It is already registered for name '" + registeredName + "'");
                }
            }
            // 当 A --> B 存在时，如果再次出现 A --> B --> C 则抛出异常
            checkForAliasCircle(name, alias);
            // 注册 alias
            this.aliasMap.put(alias, name);
        }
    }
    // 别名存放在BeanDefinition的内部
    // ConcurrentHashMap中aliasMap, key=别名, value等于别名对应的真实bean的id。
}
```

```
> this = (DefaultListableBeanFactory@1504) *org.springframework.beans.factory.support.DefaultListableBeanFactory@37e547da:
> name = "person2"
> alias = "person2Alias2"
> registeredName = null
- this.aliasMap = {ConcurrentHashMap@1710} size = 2
  > "person2Alias2" -> "person2"
  > "person1Alias1" -> "person2"
```

注册 alias 和注册 BeanDefinition 的过程差不多。在最好调用了 `checkForAliasCircle()` 来对别名进行了检测。

```
public boolean hasAlias(String name, String alias) {
    for (Map.Entry<String, String> entry : this.aliasMap.entrySet()) {
        String registeredName = entry.getValue();
        if (registeredName.equals(name)) {
            String registeredAlias = entry.getKey();
            return (registeredAlias.equals(alias) || hasAlias(registeredAlias, alias));
        }
    }
    return false;
}
```

如果 name、alias 为 1、3，则构成 (1,3)，加入集合中存在 (A,1)、(3,A) 的情况则会出错。到这里 BeanDefinition、alias 都已经注入到缓存中，下一步则是等待初始化使用了。

👍 赞(15)

¥ 打赏

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

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

一个致命的 Redis 命令，导致公司损失 400 万！！
(<http://cmsblogs.com/?p=2765>)

下一篇

RabbitMQ 实战教程（一）Hello World
(<http://cmsblogs.com/?p=2768>)

- 【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>)