**2.3.3  BeanDefinition在IoC容器中的注册**

我们已经分析过BeanDefinition在IoC容器中载入和解析的过程。在这些动作完成以后，用户定义的BeanDefinition信息已经在IoC容器内建立起了自己的数据结构以及相应的数据表示，但此时这些数据还不能让IoC容器直接使用，需要在IoC容器中对这些BeanDefinition数据进行注册。这个注册为IoC容器了提供更友好的使用方式，在DefaultListableBeanFactory中，是通过一个HashMap来持有载入的BeanDefinition的，这个HashMap的定义在DefaultListableBeanFactory可以看到，如下所示。

1. /\*\* Map of bean definition objects, keyed by bean name \*/
2. **private** **final** Map<STRING,&NBSP;BEANDEFINITION> beanDefinitionMap
3. = **new** ConcurrentHashMap
4. <STRING,&NBSP;BEANDEFINITION>();

将解析得到的BeanDefinition向IoC容器中的beanDefinitionMap注册的过程是在载入BeanDefinition完成后进行的，注册的调用过程如图2-8所示。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091221/163853612.jpg> |
|  |

我们跟踪以上的代码调用去看一下具体的注册实现，在DefaultListableBeanFactory中实现了BeanDefinitionRegistry的接口，这个接口的实现完成BeanDefinition向容器的注册。这个注册过程不复杂，就是把解析得到的BeanDefinition设置到hashMap中去。需要注意的是，如果遇到同名的BeanDefinition的情况，进行处理的时候需要依据allowBeanDefinitionOverriding的配置来完成。具体的是实现如代码清单2-21所示。

代码清单2-21  BeanDefinitio注册的实现

1. //----------------------------------------------------------
2. -----------
3. // Implementation of BeanDefinitionRegistry interface.
4. //----------------------------------------------------------
5. -----------
7. **public** **void** registerBeanDefinition(String beanName,
8. BeanDefinition beanDefinition)
9. **throws** BeanDefinitionStoreException {
10. Assert.hasText(beanName, "'beanName' must not be
11. empty");
12. Assert.notNull(beanDefinition, "BeanDefinition must
13. not be **null**");
14. **if** (beanDefinition **instanceof**
15. AbstractBeanDefinition) {
16. **try** {
17. ((AbstractBeanDefinition)
18. beanDefinition).validate();
19. }
20. **catch** (BeanDefinitionValidationException ex)
21. {
22. **throw** **new**
23. BeanDefinitionStoreException(
25. beanDefinition.getResourceDescription(), beanName,
26. "Validation of bean
27. definition failed", ex);
28. }
29. }
30. //注册的过程需要synchronized，保证数据的一致性。
31. **synchronized** (**this**.beanDefinitionMap) {
32. /\*\*
33. \*这里检查是不是有相同名字的BeanDefinition已经在IoC容
34. 器中注册了，如果有
35. \*相同名字的BeanDefinition，但又不允许覆盖，那么抛出
36. 异常。
37. \*/
38. Object oldBeanDefinition =
39. **this**.beanDefinitionMap.get(beanName);
40. **if** (oldBeanDefinition != **null**) {
41. **if**
42. (!**this**.allowBeanDefinitionOverriding) {
43. **throw** **new**
44. BeanDefinitionStoreException(
45. beanDefinition.getResource
46. Description(), beanName,
47. "Cannot
48. register bean definition [" + beanDefinition + "]
49. **for** bean '" +
50. beanName +
51. "': There is
52. already [" + oldBeanDefinition + "] bound.");
53. }
54. **else** {
55. **if**
56. (**this**.logger.isInfoEnabled()) {
58. **this**.logger.info("Overriding bean definition for bean '" +
59. beanName +"': replacing [" +
60. oldBeanDefinition + "] with [" +
61. beanDefinition + "]");
62. }
63. }
64. }
65. /\*\*
66. \*这是正常注册BeanDefinition的过程，把Bean的名字存入到
67. beanDefinitionNames的同时，
68. \*把beanName作为Map的key，把beanDefinition作为value存入到IoC
69. 容器持有的
70. \*beanDefinitionMap中去。
71. \*/
72. **else** {
74. **this**.beanDefinitionNames.add(beanName);
75. **this**.frozenBeanDefinitionNames =
76. **null**;
77. }
78. **this**.beanDefinitionMap.put(beanName,
79. beanDefinition);
80. resetBeanDefinition(beanName);
81. }
82. }

完成了BeanDefinition的注册，就完成了IoC容器的初始化过程。此时，在我们使用的IoC容器DefaultListableBeanFactory中已经建立了整个Bean的配置信息，而且这些BeanDefinition已经可以被容器使用了，它们都可在beanDefinitionMap里检索和使用。容器的作用就是对这些信息进行处理和维护。这些信息是容器建立依赖反转的基础，有了这些基础数据，下面我们接着看看在IoC容器中，依赖注入是怎样完成的。