BeanDefinition的Resource定位（2）

<http://book.51cto.com>  2009-12-21 15:21  计文柯  机械工业出版社  [我要评论(0)](http://www.51cto.com/php/feedbackt.php?id=171399)

* 摘要：《Spring技术内幕：深入解析Spring架构与设计原理》从源代码的角度对Spring的内核和各个主要功能模块的架构、设计和实现原理进行了深入剖析。第2章讲述Spring Framework的核心：IoC容器的实现。本节说的是BeanDefinition的Resource定位。
* 标签：[Spring](http://www.51cto.com/php/search.php?keyword=Spring)  [IoC容器](http://www.51cto.com/php/search.php?keyword=IoC%C8%DD%C6%F7)  [Spring技术内幕：深入解析Spring架构与设计原理](http://www.51cto.com/php/search.php?keyword=Spring%BC%BC%CA%F5%C4%DA%C4%BB%A3%BA%C9%EE%C8%EB%BD%E2%CE%F6Spring%BC%DC%B9%B9%D3%EB%C9%E8%BC%C6%D4%AD%C0%ED)

[Oracle帮您准确洞察各个物流环节](http://ad.cn.doubleclick.net/clk;202930372;27012168;u)

\*这里是使用BeanDefinitionReader载入Bean定义的地方，因为允许有多种载入方式，

\*虽然用得最多的是XML定义的形式，这里通过一个抽象函数把具体的实现委托给子类来完成。

1. \*/
2. protected abstract void
3. loadBeanDefinitions(DefaultListableBeanFactory beanFactory)
4. throws IOException, BeansException;
5. **public** **int** loadBeanDefinitions(String location, **Set**
6. actualResources) throws
7. BeanDefinitionStoreException {
8. //这里取得
9. ResourceLoader,使用的是DefaultResourceLoader。
10. ResourceLoader resourceLoader = getResourceLoader();
11. if (resourceLoader == null) {
12. throw new BeanDefinitionStoreException(
13. "Cannot import bean
14. definitions **from** location [" + location + "]:
15. **no** ResourceLoader available");
16. }
17. /\*\*

\*这里对Resource的路径模式进行解析，比如我们设定的各种Ant格式的路径定义，得到需要的

\*Resource集合，这些Resource集合指向我们已经定义好的BeanDefinition信息，可以是多个文件。

1. \*/
2. **if** (resourceLoader **instanceof**
3. ResourcePatternResolver) {
4. // Resource pattern matching available.
5. **try** {
6. //
7. 调用DefaultResourceLoader的getResource完成具体的Resource定位
8. 。
9. Resource[] resources =
10. ((ResourcePatternResolver) resourceLoader).
11. getResources(location);
12. **int** loadCount =
13. loadBeanDefinitions(resources);
14. **if** (actualResources != **null**) {
15. **for** (**int** i = 0; i <
16. resources.length; i++) {
18. actualResources.add(resources[i]);
19. }
20. }
21. **if** (logger.isDebugEnabled()) {
22. logger.debug("Loaded " +
23. loadCount + " bean definitions from
24. location pattern [" + location + "]");
25. }
26. **return** loadCount;
27. }
28. **catch** (IOException ex) {
29. **throw** **new**
30. BeanDefinitionStoreException(
31. "Could not resolve
32. bean definition resource pattern [" +
33. location + "]", ex);
34. }
35. }
36. **else** {
37. // Can only load single resources by
38. absolute URL.
39. //
40. 调用DefaultResourceLoader的getResource完成具体的Resource定位
41. 。
42. Resource resource =
43. resourceLoader.getResource(location);
44. **int** loadCount =
45. loadBeanDefinitions(resource);
46. **if** (actualResources != **null**) {
47. actualResources.add(resource);
48. }
49. **if** (logger.isDebugEnabled()) {
50. logger.debug("Loaded " + loadCount +
51. " bean definitions from
52. location [" + location + "]");
53. }
54. **return** loadCount;
55. }
56. }
57. //对于取得Resource的具体过程，我们可以看看DefaultResourceLoa
58. der是怎样完成的:
59. **public** Resource getResource(String location) {
60. Assert.notNull(location, "Location must not be
61. **null**");
62. //这里处理带有classpath标识的Resource。
63. **if** (location.startsWith(CLASSPATH\_URL\_PREFIX)) {
64. **return** **new**
65. ClassPathResource(location.substring(CLASSPATH\_URL\_PREFIX.
66. length()), getClassLoader());
67. }
68. **else** {
69. **try** {
70. // Try to parse the location as a
71. URL...
72. // 这里处理URL标识的Resource定位。
73. URL url = **new** URL(location);
74. **return** **new** UrlResource(url);
75. }
76. **catch** (MalformedURLException ex) {
77. // No URL -> resolve as resource
78. path.
79. /\*\*
81. \*如果既不是classpath，也不是URL标识的Resource定位，则把getRe
82. source的重任
83. \*交给
84. getResourceByPath，这个方法是一个protected方法，默认的实现是
85. 得到一
87. \*个ClassPathContextResource，这个方法常常会用子类来实现。
88. \*/
89. **return** getResourceByPath(location);
90. }
91. }
92. }

前面我们看到的getResourceByPath会被子类FileSystemXmlApplicationContext实现，这个方法返回的是一个 FileSystemResource对象，通过这个对象Spring可以进行相关的IO操作，完成BeanDefinition的定位。分析到这里已经一目了然，它实现的就是对path进行解析，然后生成一个FileSystemResource对象并返回，如代码清单2-6所示。

代码清单2-6  FileSystemXmlApplicationContext生成Resource对象

1. **protected** Resource getResourceByPath(String path) {
2. **if** (path != **null** && path.startsWith("/")) {
3. path = path.substring(1);
4. }
5. **return** **new** FileSystemResource(path);
6. }

如果是其他的ApplicationContext，那么对应地会生成其他种类的Resource，比如ClassPathResource、ServletContextResource等。关于Spring中Resource的种类，可以在图2-6中的Resource类的继承关系中了解。作为接口的Resource定义了许多与IO相关的操作，这些操作也都可以从图2-6中Resource的接口定义中看到。这些接口对不同的Resource实现代表着不同的意义，是Resource的实现需要考虑的。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091221/155850773.jpg> |
|  |

从图2-6中我们可以看到Resource的定义和它的继承关系，通过对前面的实现原理的分析，我们以FileSystemXmlApplicationContext的实现原理为例子，了解了Resource定位问题的解决方案，即以FileSystem方式存在的Resource的定位实现。在BeanDefinition定位完成的基础上，就可以通过返回的Resource对象来进行BeanDefinition的载入了。在定位过程完成以后，为BeanDefinition的载入创造了IO操作的条件，但是具体的数据还没有开始读入。这些数据的读入将在下面看到的BeanDefinition的载入和解析中来完成。仍然以水桶为例子，这里就像如果要用水桶去打水，那么先要找到水源。这里完成对Resource的定位，就类似于水源已经找到了，下面就是打水的过程了，类似于把找到的水装到水桶里的过程。找水不简单，但与打水相比，我们发现打水更需要技巧。