代码清单2-28  对Bean Reference的解析

1. **private** Object resolveReference(Object argName,
2. RuntimeBeanReference ref) {  **try** {  /\*\*
3. \*从RuntimeBeanReference取得reference的名字，这个RuntimeBeanR
4. eference是在载入  \*BeanDefinition时根据配置生成的。  \*/
5. String refName = ref.getBeanName();   refName =
6. String.valueOf(evaluate(refName));
7. //如果ref是在双亲IoC容器中，那就到双亲IoC容器中去取。   if
8. (ref.isToParent()) {    **if**
9. (**this**.beanFactory.getParentBeanFactory() == **null**) {
10. **throw** **new** BeanCreationException(
11. **this**.beanDefinition.getResourceDescription(), **this**.beanName,
12. "Can't resolve reference to bean '" + refName +       "' in
13. parent factory: no parent factory available");    }
14. **return**
15. **this**.beanFactory.getParentBeanFactory().getBean(refName);
16. }  /\*\*
17. \*在当前IoC容器中去取bean，这里会触发一个getBean的过程，如果
18. 依赖注入没有发生， \*这里会触发相应的依赖注入的发生。 \*/
19. **else** {   Object bean = **this**.beanFactory.getBean(refName);
20. **this**.beanFactory.registerDependentBean(refName,
21. **this**.beanName);   **return** bean;   }  **catch** (BeansException
22. ex) {  **throw** **new** BeanCreationException(
23. **this**.beanDefinition.getResourceDescription(), **this**.beanName,
24. "Cannot resolve reference to bean '" + ref.getBeanName() +
25. "'    while setting " + argName, ex);  }
26. 我们看看对其他类型的属性进行注入的例子，比如array和list等：
27. ivate Object resolveManagedArray(Object argName, List<?> ml,
28. Class elementType) {Object resolved =
29. Array.newInstance(elementType, ml.size()); **for** (**int** i = 0; i
30. < ml.size(); i++) {   Array.set(resolved, i,
31. resolveValueIfNecessary(      argName + " with key " +
32. BeanWrapper.PROPERTY\_KEY\_      PREFIX + i +
33. BeanWrapper.PROPERTY\_KEY\_SUFFIX,       ml.get(i)));  } eturn
34. resolved; / For each element in the managed list, resolve
35. reference **if** necessary. ivate List resolveManagedList(Object
36. argName, List<?> ml) {   List<Object> resolved = **new**
37. ArrayList<Object>(ml.size()); **for** (**int** i = 0; i < ml.size();
38. i++) {   resolved.add(      resolveValueIfNecessary(
39. argName + " with key " + BeanWrapper.PROPERTY\_KEY\_
40. PREFIX + i + BeanWrapper.PROPERTY\_KEY\_SUFFIX,
41. ml.get(i)));  }  **return** resolved; }

这两种属性的注入都调用了resolveValueIfNecessary，这个方法包含了所有对注入类型的处理，这个resolveValueIfNecessary的实现，如代码清单2-29所示。

代码清单2-29  resolveValueIfNecessary的实现

1. **public** Object resolveValueIfNecessary(Object argName, Object
2. value) {  /\*\*  \* We must check each value to see whether it
3. requires a runtime reference  \* to another bean to be
4. resolved.  \*/  /\*\*  \*
5. 这里对RuntimeBeanReference进行解析，RuntimeBeanReference
6. \*是在对BeanDefinition进行解析是生成的数据对象。  \*/  **if**
7. (value **instanceof** RuntimeBeanReference) {
8. RuntimeBeanReference ref = (RuntimeBeanReference) value;
9. **return** resolveReference(argName, ref);  }  **else** **if** (value
10. **instanceof** RuntimeBeanNameReference) {   String refName =
11. ((RuntimeBeanNameReference) value).getBeanName();   refName
12. = String.valueOf(evaluate(refName));   **if**
13. (!**this**.beanFactory.containsBean(refName)) { **throw** **new**
14. BeanDefinitionStoreException(      "Invalid bean name '" +
15. refName + "' in bean reference for " +       argName);   }
16. **return** refName;  }  **else** **if** (value **instanceof**
17. BeanDefinitionHolder) {   /\*\*   \* Resolve
18. BeanDefinitionHolder: contains BeanDefinition with name
19. \*and aliases.   \*/   BeanDefinitionHolder bdHolder =
20. (BeanDefinitionHolder) value;   **return**
21. resolveInnerBean(argName, bdHolder.getBeanName(),
22. bdHolder.get BeanDefinition());  }  **else** **if** (value
23. **instanceof** BeanDefinition) {   // Resolve plain
24. BeanDefinition, without contained name: use dummy name.
25. BeanDefinition bd = (BeanDefinition) value;   **return**
26. resolveInnerBean(argName, "(inner bean)", bd);  }
27. //这里对ManageArray进行解析。  else if (value instanceof
28. ManagedArray) {   // May need to resolve contained runtime
29. references.   ManagedArray array = (ManagedArray) value;
30. Class elementType = array.resolvedElementType;   **if**
31. (elementType == **null**) {    String elementTypeName =
32. array.getElementTypeName();    **if**
33. (StringUtils.hasText(elementTypeName)) {     **try** {
34. elementType = ClassUtils.forName(elementTypeName,
35. **this**.beanFactory.     array.resolvedElementType =
36. elementType;     }     **catch** (Throwable ex) {      //
37. Improve the message by showing the context.      **throw** **new**
38. BeanCreationException(
39. **this**.beanDefinition.getResourceDescription(),
40. **this**.beanName,        "Error resolving array type for " +
41. argName, ex);     }    }    **else** {     elementType =
42. Object.**class**;    }   }   **return** resolveManagedArray(argName,
43. (List<?>) value, elementType);  }
44. //这里对ManageList进行解析。  else if (value instanceof
45. ManagedList) {   // May need to resolve contained runtime
46. references.   **return** resolveManagedList(argName, (List<?>)
47. value);  } //这里对ManageSet进行解析。  else if (value
48. **instanceof** ManagedSet) {   // May need to resolve contained
49. runtime references.   **return** resolveManagedSet(argName,
50. (Set<?>) value);  } //这里对ManageMap进行解析。 else if
51. (value **instanceof** ManagedMap) {   // May need to resolve
52. contained runtime references.   **return**
53. resolveManagedMap(argName, (Map<?, ?>) value);  }
54. //这里对ManageProperties进行解析。  else if (value
55. **instanceof** ManagedProperties) {   Properties original =
56. (Properties) value;   Properties copy = **new** Properties();
57. **for** (Map.Entry propEntry : original.entrySet()) {    Object
58. propKey = propEntry.getKey();    Object propValue =
59. propEntry.getValue();    **if** (propKey **instanceof**
60. TypedStringValue) {     propKey = ((TypedStringValue)
61. propKey).getValue();    }    **if** (propValue **instanceof**
62. TypedStringValue) {     propValue = ((TypedStringValue)
63. propValue).getValue();    }    copy.put(propKey, propValue);
64. }   **return** copy;  }  //这里对TypedStringValue进行解析。
65. **else** **if** (value **instanceof** TypedStringValue) {   // Convert
66. value to target type here.   TypedStringValue
67. typedStringValue = (TypedStringValue) value;   Object
68. valueObject = evaluate(typedStringValue.getValue());   **try** {
69. Class resolvedTargetType =
70. resolveTargetType(typedStringValue);    **if**
71. (resolvedTargetType != **null**) {     **return**
72. **this**.typeConverter.convertIfNecessary(valueObject,
73. resolvedTargetType);    }    **else** {     **return** valueObject;
74. }   }   **catch** (Throwable ex) {    // Improve the message by
75. showing the context.    **throw** **new** BeanCreationException(
76. **this**.beanDefinition.getResourceDescription(),
77. **this**.beanName,      "Error converting typed String value **for**
78. " + argName, ex);   }  }  **else** {   **return** evaluate(value);
79. } } //对runtimeBeanReference类型的注入在resolveReference中：
80. **private** Object resolveReference(Object argName,
81. RuntimeBeanReference ref) {  **try** { /\*\*
82. \*从RuntimeBeanReference取得reference的名字，这个RuntimeBeanR
83. eference \*是在载入BeanDefinition时根据配置生成的。 \*/
84. String refName = ref.getBeanName();   refName =
85. String.valueOf(evaluate(refName));
86. //如果ref是在双亲IoC容器中，那就到双亲IoC容器中去取。if
87. (ref.isToParent()) {    **if**
88. (**this**.beanFactory.getParentBeanFactory() == **null**) {
89. **throw** **new** BeanCreationException(
90. **this**.beanDefinition.getResourceDescription(),
91. **this**.beanName,       "Can't resolve reference to bean '" +
92. refName +       "' in parent factory: no parent factory
93. available");    }    **return**
94. **this**.beanFactory.getParentBeanFactory().getBean(refName);
95. }  /\*\*
96. \*在当前IoC容器中去取bean，这里会触发一个getBean的过程，如果
97. 依赖注入没有发生， \*这里会触发相应的依赖注入的发生。 \*/
98. **else** {    Object bean = **this**.beanFactory.getBean(refName);
99. **this**.beanFactory.registerDependentBean(refName,
100. **this**.beanName);    **return** bean;   }  }  **catch**
101. (BeansException ex) {   **throw** **new** BeanCreationException(
102. **this**.beanDefinition.getResourceDescription(), **this**.beanName,
103. "Cannot resolve reference to bean '" + ref.getBeanName() +
104. "'     while setting " + argName, ex);  } }
105. //对manageList的处理过程在resolveManagedList中： private
106. List resolveManagedList(Object argName, List<?> ml) {
107. List<Object> resolved = **new** ArrayList<Object>(ml.size());
108. **for** (**int** i = 0; i < ml.size(); i++) {
109. //递归的对List的元素进行resolve。   resolved.add(
110. resolveValueIfNecessary(       argName + " with key " +
111. BeanWrapper.PROPERTY\_KEY\_      PREFIX + i +
112. BeanWrapper.PROPERTY\_KEY\_SUFFIX,       ml.get(i)));  }
113. **return** resolved; }

在完成这个 resolve 过程后，就已经为依赖注入准备好了条件，这是真正把 Bean 对象设置到它所依赖的另一个 Bean 的属性中去的地方，其中处理的属性是各种各样的。依赖注入是在BeanWrapper 的 setPropertyValues 中实现的，而具体的完成却是在 BeanWrapper 的子类BeanWrapperImpl中，如代码清单2-30所示。