**2.2  IoC容器系列的实现：BeanFactory和ApplicationContext**

**2.2.1  BeanFactory对IoC容器的功能定义**

从前面的介绍，我们知道BeanFactory定义了IoC容器的基本功能规范，所以，下面我们就从BeanFactory这个最基本的容器定义来进入Spring的IoC容器体系，去了解IoC容器的实现原理。IoC容器的基本接口是由BeanFactory来定义的，也就是说，BeanFactory定义了IoC容器的最基本的形式，并且提供了IoC容器所应该遵守的最基本的服务契约。同时，这也是我们使用IoC容器所应遵守的最底层和最基本的编程规范，这些接口定义勾画出了IoC的基本轮廓。很显然，在Spring的代码实现中，BeanFactory只是一个接口类，并没有给出容器的具体实现，而我们在图2-2中看到的各种具体类，比如DefaultListableBeanFactory、XmlBeanFactory、ApplicationContext等都可以看成是容器的附加了某种功能的具体实现，也就是容器体系中的具体容器产品。下面我们来看看BeanFactory是怎样定义IoC容器的基本接口的。下面介绍这个基本接口为用户提供的基本功能。

用户使用容器时，可以使用转义符"&"来得到FactoryBean本身，用来区分通过容器来获取FactoryBean产生的对象和获取FactoryBean本身。举例来说，如果myJndiObject是一个FactoryBean，那么使用&myJndiObject得到的是FactoryBean，而不是myJndiObject这个FactoryBean产生出来的对象。

注意  理解上面这段话需要很好地区分FactoryBean和BeanFactory这两个在Spring中使用频率很高的类，它们在拼写上非常相似。一个是Factory，也就是IoC容器或对象工厂；一个是Bean。在Spring中，所有Bean都是由BeanFactory（也就是IoC容器）来进行管理的。但对FactoryBean而言，这个Bean不是简单的Bean，而是一个能产生或者修饰对象生成的工厂Bean，它的实现与设计模式中的工厂模式和修饰器模式类似。

BeanFactory接口设计了getBean方法，这个方法是使用IoC容器API的主要方法，通过这个方法，可以取得IoC容器中管理的Bean，Bean的取得是通过指定名字来进行索引的。如果需要在获取Bean时对Bean的类型进行检查，BeanFactory接口定义了带有参数的getBean方法，这个方法的使用与getBean方法类似，不同的是增加了对Bean检索的类型的要求。

用户可以通过BeanFactory接口方法getBean来使用Bean名字，从而当获取Bean时，如果需要获取的Bean是prototype类型的，用户还可以为这个prototype类型的Bean生成指定构造函数的对应参数。这使得在一定程度上可以控制生成prototype类型的Bean。有了BeanFactory的定义，用户可以执行以下操作：

\* 通过接口方法containsBean让用户能够判断容器是否含有指定名字的Bean。

\* 通过接口方法isSingleton来查询指定了名字的Bean是否是Singleton类型的Bean。对于Singleton属性，用户可以在BeanDefinition中指定。

\* 通过接口方法isPrototype来查询指定了名字的Bean是否是prototype类型的。与Singleton属性一样，这个属性也可以由用户在BeanDefinition中指定。

\* 通过接口方法isTypeMatch来查询指定了名字的Bean的Class类型是否是特定的Class类型。这个Class类型可以由用户来指定。

\* 通过接口方法getType来查询指定了名字的Bean的Class类型。

\* 通过接口方法getAliases来查询指定了名字的Bean的所有别名，这些别名都是用户在BeanDefinition中定义的。

这些定义的接口方法勾画出了IoC容器的基本特性，因为BeanFactory接口定义了IoC容器，所以下面给出它定义的全部内容来让大家参考，如代码清单2-1所示。

代码清单2-1  BeanFactory接口

1. **public** **interface** BeanFactory {
2. /\*\*
3. \* Used to dereference a {@link FactoryBean}
4. instance and distinguish it from
5. \* beans <i>created</i> by the FactoryBean. For
6. example, if the bean named
7. \* <code>myJndiObject</code> is a FactoryBean,
8. getting <code>&myJndiObject </code>
9. \* will return the factory, not the instance
10. returned by the factory.
11. \*/
12. String FACTORY\_BEAN\_PREFIX = "&";
13. /\*\*
14. \* Return an instance, which may be shared or
15. independent, of the specified bean.
16. \* <p>This method allows a Spring BeanFactory to be used as
17. a replacement for the
18. \* Singleton or Prototype design pattern. Callers may retain
19. references to
20. \* returned objects in the case of Singleton beans.
21. \* <p>Translates aliases back to the corresponding canonical
22. bean name.
23. \* Will ask the parent factory if the bean cannot be found
24. in this factory instance.
25. \* \*/
26. Object getBean(String name) **throws** BeansException;
27. /\*\*
28. \* Return an instance, which may be shared or
29. independent, of the specified bean.
30. \* <p>Behaves the same as {@link #getBean(String)}, but
31. provides a measure of type
32. \* safety by throwing a BeanNotOfRequiredTypeException if
33. the bean is not of the
34. \* required type. This means that ClassCastException can't
35. be thrown on casting
36. \* the result correctly, as can happen with {@link
37. #getBean(String)}.
38. \* <p>Translates aliases back to the corresponding canonical
39. bean name.
40. \* Will ask the parent factory if the bean cannot be
41. found in this factory instance.
42. \*  \*/
43. <T> T getBean(String name, Class<T> requiredType)
44. **throws** BeansException;
45. /\*\*
46. \* Return an instance, which may be shared or
47. independent, of the specified bean.
48. \* <p>Allows for specifying explicit constructor
49. arguments / factory method arguments,
50. \* overriding the specified default arguments (if any) in
51. the bean definition.
52. \*  \*/
53. Object getBean(String name, Object... args) **throws**
54. BeansException;
55. /\*\*
56. \* Does this bean factory contain a bean with the
57. given name? More specifically,
58. \* is {@link #getBean} able to obtain a bean
59. instance for the given name?
60. \* <p>Translates aliases back to the corresponding
61. canonical bean name.
62. \* Will ask the parent factory if the bean cannot be
63. found in this factory instance.
64. \*/
65. **boolean** containsBean(String name);
66. /\*\*
67. \* Is this bean a shared singleton? That is, will
68. {@link #getBean} always
69. \* return the same instance?
70. \* <p>Note: This method returning <code>false</code>
71. does not clearly indicate
72. \* independent instances. It indicates non-singleton
73. instances, which may correspond
74. \* to a scoped bean as well. Use the {@link
75. #isPrototype} operation to explicitly
76. \* check for independent instances.
77. \* <p>Translates aliases back to the corresponding canonical
78. bean name.
79. \* Will ask the parent factory if the bean cannot be found
80. in this factory instance.
81. \*/
82. **boolean** isSingleton(String name) **throws**
83. NoSuchBeanDefinitionException;
84. /\*\*
85. \* Is this bean a prototype? That is, will {@link #getBean}
86. always return
87. \* independent instances?
88. \* <p>Note: This method returning <code>false</code> does
89. not clearly indicate
90. \* a singleton object. It indicates non-independent
91. instances, which may correspond
92. \* to a scoped bean as well. Use the {@link #isSingleton}
93. operation to explicitly
94. \* check for a shared singleton instance.
95. \* <p>Translates aliases back to the corresponding
96. canonical bean name.
97. \* Will ask the parent factory if the bean cannot be found
98. in this factory instance.
99. \*/
100. **boolean** isPrototype(String name) **throws**
101. NoSuchBeanDefinitionException;
102. /\*\*
103. \* Check whether the bean with the given name matches the
104. specified type.
105. \* More specifically, check whether a {@link #getBean} call
106. for the given name
107. \* would return an object that is assignable to the specified
108. target type.
109. \* <p>Translates aliases back to the corresponding canonical
110. bean name.
111. \* Will ask the parent factory if the bean cannot be found in
112. this factory instance.
113. \*/
114. **boolean** isTypeMatch(String name, Class targetType)
115. **throws** NoSuchBeanDefinitionException;
116. /\*\*
117. \* Determine the type of the bean with the given name. More
118. specifically,
119. \* determine the type of object that {@link #getBean} would
120. return for the given name.
121. \* <p>For a {@link FactoryBean}, return the type of object
122. that the FactoryBean creates,
123. \* as exposed by {@link FactoryBean#getObjectType()}.
124. \* <p>Translates aliases back to the corresponding canonical
125. bean name.
126. \* Will ask the parent factory if the bean cannot be found in
127. this factory instance.
128. \*/
129. Class getType(String name) **throws**
130. NoSuchBeanDefinitionException;
131. /\*\*
132. \* Return the aliases for the given bean name, if any.
133. \* All of those aliases point to the same bean when used in a
134. {@link #getBean} call.
135. \* <p>If the given name is an alias, the corresponding
136. original bean name
137. \* and other aliases (if any) will be returned, with the
138. original bean name
139. \* being the first element in the array.
140. \* <p>Will ask the parent factory if the bean cannot be found
141. in this factory instance.
142. \*/
143. String[] getAliases(String name);
144. }