代码清单3-24  在拦截器链的初始化中获取advisor通知器

1. **private** **synchronized** **void** initializeAdvisorChain() **throws**
2. AopConfigException,
3. BeansException {
4. **if** (**this**.advisorChainInitialized) {
5. **return**;
6. }
7. **if** (!ObjectUtils.isEmpty(**this**.interceptorNames)) {
8. **if** (**this**.beanFactory == **null**) {
9. **throw** **new** IllegalStateException("No
10. BeanFactory available anymore
11. (probably due to serialization) " +
12. "- cannot resolve interceptor names
13. " + Arrays.asList(**this**.
14. interceptorNames));
15. }
16. // Globals can't be last unless we specified
17. a targetSource using the
18. property...
19. **if**
20. (**this**.interceptorNames[**this**.interceptorNames.length -
21. 1].endsWith
22. (GLOBAL\_SUFFIX) &&
23. **this**.targetName == **null** &&
24. **this**.targetSource == EMPTY\_TARGET\_
25. SOURCE) {
26. **throw** **new** AopConfigException("Target
27. required after globals");
28. }
29. // Materialize interceptor chain from bean
30. names.
31. **for** (String name : **this**.interceptorNames) {
32. **if** (logger.isTraceEnabled()) {
33. logger.trace("Configuring
34. advisor or advice '" + name + "'");
35. }
36. **if** (name.endsWith(GLOBAL\_SUFFIX)) {
37. **if** (!(**this**.beanFactory
38. **instanceof** ListableBeanFactory)) {
39. **throw** **new**
40. AopConfigException(
41. "Can
42. only use global advisors or interceptors with
43. a
44. ListableBeanFactory");
45. }
47. addGlobalAdvisor((ListableBeanFactory) **this**.beanFactory,
49. name.substring(0, name.length() - GLOBAL\_SUFFIX.length()));
50. }
51. **else** {
52. // If we get here, we need
53. to add a named interceptor.
54. // We must check if it's a
55. singleton or prototype.
56. Object advice;
58. **if** (**this**.singleton ||
59. **this**.beanFactory.isSingleton(name)) {
60. // Add the real
61. Advisor/Advice to the chain.
62. /\*\*
63. \*
64. 这里是取得advisor的地方，是通过beanFactory取得的，
66. \*把interceptorNames这个List中的interceptor名字交给
68. \*beanFactory就可以了，然后通过调用BeanFactory的getBean去获取
69. 。
70. \*/
71. advice =
72. **this**.beanFactory.getBean(name);
73. }
74. **else** {
75. // It's a prototype
76. Advice or Advisor: replace with a prototype.
77. /\*\*
78. \* Avoid unnecessary
79. creation of prototype bean just for
80. \*advisor
81. chain initialization.
82. \*/
83. advice = **new**
84. PrototypePlaceholderAdvisor(name);
85. }
87. addAdvisorOnChainCreation(advice, name);
88. }
89. }
90. }
91. **this**.advisorChainInitialized = **true**;
92. }

advisor通知器的取得是由IoC容器完成的，但是在ProxyFactoryBean中是如何获得IoC容器，然后通过回调IoC容器的getBean方法来得到需要的通知器advisor呢？这涉及IoC容器的实现原理，在我们使用DefaultListableBeanFactory作为IoC容器使用的时候，由于它的基类是AbstractAutowireCapableBeanFactory，在这个AbstractAutowireCapable-

BeanFactory中，可以看到一个对Bean进行初始化的initializeBean方法；在这个Bean的初始化过程中，对IoC容器在Bean中的回调进行了设置，首先，判断这个Bean的类型是不是实现了BeanFactoryAware接口，如果它实现了BeanFactoryAware接口，那么它就一定实现了BeanFactoryAware定义的接口方法，通过这个接口方法，可以把IoC容器设置到Bean自身定义的一个属性中去。这样，在这个Bean的自身实现中，就能够得到它所在的IoC容器，从而调用IoC容器的getBean方法，完成对IoC容器的回调，就像一个有特异功能的Bean一样, 除了使用为自己设计的功能之外，还可以去调用它所在的容器的功能，如下所示。

1. **if** (bean **instanceof** BeanFactoryAware) {
2. ((BeanFactoryAware) bean).setBeanFactory(**this**);
3. }

对IoC容器的使用，如果需要回调容器，前提是当前的Bean需要实现BeanFactoryAware接口，这个接口只需要实现一个接口方法setBeanFactory，同时设置一个属性来持有BeanFactory的IoC容器，就可以在Bean中取得IoC容器进行回调了。在IoC容器对Bean进行初始化的时候，会对Bean的类型进行判断，如果这是一个BeanFactoryAware的Bean类型，那么IoC容器会调用这个Bean的setBeanFactory方法，完成对这个BeanFactory在Bean中的设置。具体来说，对于ProxyFactoryBean，它实现了这个接口，所以在它初始化完成以后，可以在Bean中使用容器进行回调。这里设置进去的this对象就是Bean所在的IoC容器，一般而言是DefaultListableBeanFactory对象。通过这个设置，在得到这个设置好的BeanFactory以后，ProxyFactoryBean就可以通过回调容器的getBean去获取配置在Bean定义文件中的通知器了，获取通知器就是向IoC容器getBean的过程。了解IoC容器的实现原理的读者都知道，这个getBean是IoC容器一个非常基本的方法。在调用时，ProxyFactoryBean需要给出通知器的名字，而这些名字都是在interceptorNames的List中已经配置好的，在IoC对FactoryBean进行依赖注入时，会直接注入到FactoryBean的interceptorNames属性中。完成这个过程以后，ProxyFactoryBean就获得了配置的通知器，为切面增强做好准备。