**3.3.6  Advice通知的实现**

经过前面的分析，我们看到在AopProxy代理对象的生成的时候，AopProxy代理对象的拦截器也同样建立起来了，对于拦截器的拦截调用和最终目标对象的方法调用，也都看到了它们相应的实现原理。但是，对于AOP实现的重要部分，Spring AOP定义的通知是怎样实现对目标对象的增强的呢？本节将探讨这个问题。在为AopProxy代理对象配置拦截器的实现中，有一个取得拦截器的配置过程，这个过程是由DefaultAdvisorChainFactory实现的，而在这个工厂类中，它负责生成拦截器链，在它的getInterceptorsAndDynamicInterceptionAdvice方法中，有一个适配和注册过程，就是在这个适配和注册过程中，通过配置Spring预先设计好的拦截器，Spring加入了它对AOP实现的处理。为了解这个详细的过程，我们先从DefaultAdvisor-

ChainFactory的实现开始，如代码清单3-25所示。在DefaultAdvisorChainFactory的实现中，首先构造了一个GlobalAdvisorAdapterRegistry单件，然后，对配置的Advisor通知器逐个进行遍历，这些通知器链都是配置在interceptorNames中的，这点我们已经不陌生；从getInterceptorsAndDynamicInterceptionAdvice传递进来的advised参数对象中可以方便地取到配置的通知器，有了这些通知器，接着就是一个由GlobalAdvisorAdapter-

Registry来完成的拦截器的适配和注册过程。

代码清单3-25  DefaultAdvisorChainFactory使用GlobalAdvisorAdapterRegistry得到AOP拦截器

1. //得到注册器GlobalAdvisorAdapterRegistry，这是一个单件模式的
2. 实现。
3. AdvisorAdapterRegistry registry =
4. GlobalAdvisorAdapterRegistry.getInstance();
5. **for** (Advisor advisor : config.getAdvisors()) {
6. **if** (advisor **instanceof** PointcutAdvisor) {
7. // Add it conditionally.
8. PointcutAdvisor pointcutAdvisor =
9. (PointcutAdvisor) advisor;
10. **if** (config.isPreFiltered() ||
11. pointcutAdvisor.getPointcut().getClass
12. Filter().matches(targetClass)) {
14. //从GlobalAdvisorAdapterRegistry中取得MethodInterceptor的实
15. 现。
16. MethodInterceptor[] interceptors =
17. registry.getInterceptors(advisor);
18. MethodMatcher mm =
19. pointcutAdvisor.getPointcut().getMethodMatcher();
20. **if** (MethodMatchers.matches(mm,
21. method, targetClass, hasIntroductions)) {
22. **if** (mm.isRuntime()) {
23. /\*\*
24. \* Creating a new
25. object instance in the getInterceptors() method
26. \* isn't a problem as
27. we normally cache created chains.
28. \*/
29. **for**
30. (MethodInterceptor interceptor : interceptors) {
32. interceptorList.add(**new** InterceptorAndDynamicMethod
33. Matcher(interceptor,
34. mm));
36. }
37. }
38. **else** {
39. interceptorList.addAll(Arrays.asList(interceptors));
40. }
41. }
42. }
43. }

仔细揣摩了以上代码的读者一定会注意到，在这个GlobalAdvisorAdapterRegistry中隐藏着不少AOP实现的重要细节，它的getInterceptors方法为AOP实现作出了很大的贡献，就是这个方法封装着advice织入实现的入口，我们先从GlobalAdvisorAdapterRegistry的实现入手，如代码清单3-26所示。从代码上看，GlobalAdvisorAdapterRegistry的实现很简洁，起到的基本上是一个适配器的作用，但同时也是一个单件模式的应用，它为Spring AOP模块提供了一个DefaultAdvisorAdapterRegistry单件，这个DefaultAdvisorAdapter

Registry是我们后面要分析的重点，像它的名字一样，由它来完成各种通知的适配和注册工作。

代码清单3-26  GlobalAdvisorAdapterRegistry的实现

1. **public** **abstract** **class** GlobalAdvisorAdapterRegistry {
2. /\*\*
3. \* Keep track of a single instance so we can
4. return it to classes that request it.
5. \*/
6. **private** **static** **final** AdvisorAdapterRegistry instance =
7. **new** DefaultAdvisor
8. AdapterRegistry();
10. // Return the singleton DefaultAdvisorAdapterRegistry instance.
11. **public** **static** AdvisorAdapterRegistry getInstance() {
12. **return** instance;
13. }
14. }

到这里，神秘的面纱慢慢地开始被揭开了，在DefaultAdvisorAdapterRegistry中设置了一系列的adapter适配器，正是这些adapter适配器的实现，为Spring AOP的advice提供编织能力，让我们先到DefaultAdvisorAdapterRegistry中去看一看究竟在这里发生了什么，如代码清单3-27所示。首先，我们看到了一系列在我们的AOP应用中，与使用到的Spring AOP的advice通知相对应的adapter适配实现，并看到了对这些adapter的具体使用。具体地说，对它们的使用主要体现在两个方面，一是调用adapter的support方法，通过这个方法来判断取得的advice属于什么类型的advice通知，从而根据不同的advice类型来注册不同的AdviceInterceopter，也就是我们前面看到的那些拦截器；而另一方面，对于这些AdviceInterceopter，不需要我们操心，都是Spring AOP框架设计好了的，它们是为实现不同的advice功能提供服务的。有了这些AdviceInterceopter，我们可以方便地使用由Spring提供的各种不同的advice来设计AOP应用。也就是说，正是这些AdviceInterceptor最终实现了advice通知在AopProxy代理对象中的织入功能。

代码清单3-27  DefaultAdvisorAdapterRegistry的实现

1. **public** **class** DefaultAdvisorAdapterRegistry **implements**
2. AdvisorAdapterRegistry,
3. Serializable {
4. //持有AdvisorAdaptor的List，它的Adapter是与实现Sprin
5. g AOP的advice增强功能相对应的。
6. **private** **final** List<AdvisorAdapter> adapters = **new**
7. ArrayList<AdvisorAdapter>(3);
9. // Create a new DefaultAdvisorAdapterRegistry,
10. registering well-known adapters.
11. /\*\*
12. \*这里把已有的advice实现的Adapter加入进来，有我们非常
13. 熟悉的
14. \*MethodBeforeAdvice、AfterReturningAdvice、ThrowsAdv
15. ice这些AOP的advice封装实现。
16. \*/
17. **public** DefaultAdvisorAdapterRegistry() {
18. registerAdvisorAdapter(**new**
19. MethodBeforeAdviceAdapter());
20. registerAdvisorAdapter(**new**
21. AfterReturningAdviceAdapter());
22. registerAdvisorAdapter(**new**
23. ThrowsAdviceAdapter());
24. }
25. **public** Advisor wrap(Object adviceObject) **throws**
26. UnknownAdviceTypeException {
27. **if** (adviceObject **instanceof** Advisor) {
28. **return** (Advisor) adviceObject;
29. }
30. **if** (!(adviceObject **instanceof** Advice)) {
31. **throw** **new**
32. UnknownAdviceTypeException(adviceObject);
33. }
34. Advice advice = (Advice) adviceObject;
35. **if** (advice **instanceof** MethodInterceptor) {
36. // So well-known it doesn't even
37. need an adapter.
38. **return** **new**
39. DefaultPointcutAdvisor(advice);
40. }
41. **for** (AdvisorAdapter adapter : **this**.adapters)
42. {
43. // Check that it is supported.
44. **if** (adapter.supportsAdvice(advice))
45. {
46. **return** **new**
47. DefaultPointcutAdvisor(advice);
48. }
49. }
50. **throw** **new**
51. UnknownAdviceTypeException(advice);
52. }
53. //这里是在DefaultAdvisorChainFactory中启动的getIntec
54. eptors方法。
55. **public** MethodInterceptor[] getInterceptors(Advisor
56. advisor) throwsUnknownAdvice
57. TypeException {
58. List<MethodInterceptor> interceptors = **new**
59. ArrayList<MethodInterceptor>(3);
60. //从Advisor通知器配置中取得advice通知。
61. Advice advice = advisor.getAdvice();
63. //如果通知是MethodInterceptor类型的通知，直接加入interceptor
64. s的List中，不需要适配。
65. **if** (advice **instanceof** MethodInterceptor) {
66. interceptors.add((MethodInterceptor)
67. advice);
68. }
69. /\*\*
71. \*对通知进行适配，使用已经配置好的Adapter：MethodBeforeAdvice
72. Adapter，
74. \*AfterReturningAdviceAdapter以及ThrowsAdviceAdapter。
76. \*然后从对应的adapter中取出封装好AOP编织功能的拦截器。
77. \*/
78. **for** (AdvisorAdapter adapter : **this**.adapters)
79. {
80. **if** (adapter.supportsAdvice(advice))
81. {
83. interceptors.add(adapter.getInterceptor(advisor));
84. }
85. }
86. **if** (interceptors.isEmpty()) {
87. **throw** **new**
88. UnknownAdviceTypeException(advisor.getAdvice());
89. }
90. **return** interceptors.toArray(**new**
91. MethodInterceptor[interceptors.size()]);
92. }
93. **public** **void** registerAdvisorAdapter(AdvisorAdapter
94. adapter) {
95. **this**.adapters.add(adapter);
96. }
97. }