**3.3.7  ProxyFactory实现AOP**

我们回到3.2.1节中提到的Spring AOP的类层次关系，从中可以看到除了使用ProxyFactoryBean实现AOP应用之外，还可以使用ProxyFactory来实现Spring AOP的功能，只是在使用ProxyFactory的时候，需要编程式的完成AOP应用的设置。我们举一个使用ProxyFactory的例子，如代码清单3-32所示。

代码清单3-32  ProxyFactory的使用

1. TargetImpl target = **new** TargetImpl();
2. ProxyFactory aopFactory = **new** ProxyFactory(target);
3. aopFactory.addAdvisor(yourAdvisor);
4. aopFactory.addAdvice(yourAdvice);
5. TargetImpl targetProxy = (TargetImpl)aopFactory.getProxy();

对于使用ProxyFactory实现AOP功能，它的实现原理与ProxyFactoryBean的实现原理是一样的，只是在最外层的表现形式上有所不同；ProxyFactory没有使用FactoryBean的IoC封装，而是通过直接继承ProxyCreatorSupport的功能来完成AOP的属性配置。至于其他ProxyCreatorSupport的子类，ProxyFactory取得AopProxy代理对象其实是和ProxyFactoryBean是一样的。一般来说，也是通过getProxy为入口，由DefaultAopProxy-

Factory来完成的，关于取得AopProxy的详细分析和以后对拦截器调用的实现原理，前面我们都分析过了，这里就不再重复了。对ProxyFactory实现感兴趣的读者，可以从以下的源代码中看到它与ProxyFactoryBean实现上不同的地方，ProxyFactory的实现代码如代码清单3-33所示。从代码清单上可以看到在ProxyFactory的getProxy方法，由这个方法去取得AopProxy代理对象，这个getProxy方法的实现使用了ProxyFactory的基类ProxyCreatorSupport的createProxy方法来生成AopProxy代理对象，而这个AopProxy代理对象的生成是由AopProxyFactory来完成的，它会生成JDK或者CGLIB的代理对象。从这里的getProxy的实现开始，ProxyFactory和ProxyFactoryBean在AOP的功能实现上基本上都是一样的，包括以后拦截器的调用等。

代码清单3-33  ProxyFactory的实现

1. **public** **class** ProxyFactory **extends** ProxyCreatorSupport {
2. **public** ProxyFactory() {
3. }
4. **public** ProxyFactory(Object target) {
5. Assert.notNull(target, "Target object must
6. not be **null**");
8. setInterfaces(ClassUtils.getAllInterfaces(target));
9. setTarget(target);
10. }
11. **public** ProxyFactory(Class[] proxyInterfaces) {
12. setInterfaces(proxyInterfaces);
13. }
14. **public** ProxyFactory(Class proxyInterface, Interceptor
15. interceptor) {
16. addInterface(proxyInterface);
17. addAdvice(interceptor);
18. }
19. **public** ProxyFactory(Class proxyInterface, TargetSource
20. targetSource) {
21. addInterface(proxyInterface);
22. setTargetSource(targetSource);
23. }
24. **public** <T> T getProxy() {
25. **return** (T) createAopProxy().getProxy();
26. }
27. **public** <T> T  getProxy(ClassLoader classLoader) {
28. **return** (T)
29. createAopProxy().getProxy(classLoader);
30. }
31. **public** **static** Object getProxy(Class proxyInterface,
32. Interceptor interceptor) {
33. **return** **new** ProxyFactory(proxyInterface,
34. interceptor).getProxy();
35. }
36. **public** **static** Object getProxy(Class proxyInterface,
37. TargetSource targetSource) {
38. **return** **new** ProxyFactory(proxyInterface,
39. targetSource).getProxy();
40. }
41. **public** **static** Object getProxy(TargetSource targetSource) {
42. **if** (targetSource.getTargetClass() == **null**) {
43. **throw** **new**
44. IllegalArgumentException("Cannot create **class** proxy **for**
45. TargetSource with **null** target **class**");
46. }
47. ProxyFactory proxyFactory = **new**
48. ProxyFactory();
49. proxyFactory.setTargetSource(targetSource);
50. proxyFactory.setProxyTargetClass(**true**);
51. **return** proxyFactory.getProxy();
52. }
53. }