**3.4  Spring AOP的高级特性**

了解了Spring AOP的基本实现，下面我们来看一个关于使用Spring AOP高级特性的例子，来了解它的实现原理。在使用Spring AOP时，对目标对象的增强是通过拦截器来完成的。对于一些应用场合，需要对目标对象本身进行一些处理，比如，如何从一个对象池或对象工厂中获得目标对象等。这样，我们需要使用Spring的TargetSource接口特性，在这里，我们把这类AOP特性当作高级特性的一种，从这些AOP特性的实现原理的了解上，可以看到对AOP基本特性的灵活运用。

在Spring中，提供了许多现成的TargetSource实现，比如下面的HotSwappableTarget-

Source，这个HotSwappableTargetSource使得用户可以以线程安全的方式切换目标对象，提供所谓的热交换功能。这个特性是很有用的，尽管它的开启需要AOP应用进行显式的配置。但这个配置并不复杂，在使用时只需要把这个HotSwappableTargetSource配置到ProxyFactoryBean的target属性就可以了，在需要更换真正的目标对象时，调用HotSwappableTargetSource的swap方法就可以完成。由此可见，对HotSwappableTarget

Source的热交换功能的使用，是需要触发swap方法调用的。这个swap方法的实现很简单，它完成target对象的替换，也就是说，它使用新的target对象来替换原有的target对象。为了保证线程安全，需要把这个替换方法设为synchronized方法，如代码清单3-34所示。

代码清单3-34  HotSwappableTargetSource的swap方法

1. **public** **synchronized** Object swap(Object newTarget)
2. **throws** IllegalArgumentException {
3. Assert.notNull(newTarget, "Target object must not be null");
4. Object old = **this**.target;
5. **this**.target = newTarget;
6. **return** old;
7. }
8. **public** **synchronized** Object getTarget() {
9. **return** **this**.target;
10. }

这个target是怎样在AOP中起作用的呢？了解一下对getTarget的调用就很清楚了，这个HotSwappableTargetSource只是对真正的target做了一个简单的封装，以提供热交换的能力，并没有其他特别之处。对getTarget的方法调用关系，如图3-12所示。

[](http://images.51cto.com/files/uploadimg/20091222/1536560.jpg)

我们以JdkDynamicAopProxy的实现为例子，可以看到在AOP对Proxy代理对象进行invoke方法调用的时候，会使用这个getTarget调用取得真正的目标对象，如果已经调用过swap方法完成目标对象的热交换，那么交给AOP的已经是交换后的目标对象了，如代码清单3-35所示。具体来说，在invoke方法中，我们看到代理对象的取得，是在AopProxy代理对象的拦截器起作用之前，通过targetSource.getTarget()的调用来取得的，而这个代理对象是否被更换过，是由对swap方法的调用来负责的。因而，在invoke方法中，可以看到对于使用了什么样的代理对象，都不会对拦截器的行为做任何的改变。

代码清单3-35  invoke获取目标对象

1. target = targetSource.getTarget();
2. **if** (target != **null**) {
3. targetClass = target.getClass();
4. }
5. // Get the interception chain for this method.
6. // 这里获得定义好的拦截器链。
7. List<Object> chain =
8. **this**.advised.getInterceptorsAndDynamicInterceptionAdvice
9. (method, targetClass);
10. /\*\*
11. \* Check whether we have any advice. If we don't, we can
12. fallback on direct
13. \* reflective invocation of the target, and avoid creating a
14. MethodInvocation.
15. \*/
16. // 如果没有设定拦截器，那么就直接调用target的对应方法。
17. **if** (chain.isEmpty()) {
18. /\*\*
19. \* We can skip creating a MethodInvocation: just
20. invoke the target directly
21. \* Note that the final invoker must be an
22. InvokerInterceptor so we know it does
23. \* nothing but a reflective operation on the target,
24. and no hot swapping or
25. \* fancy proxying.
26. \*/
27. retVal =
28. AopUtils.invokeJoinpointUsingReflection(target, method,
29. args);
30. }
31. **else** {
32. // We need to create a method invocation...
33. //
34. 如果有拦截器的设定，那么需要调用拦截器之后才调用目标对象的相
35. 应方法。
36. //通过构造一个ReflectiveMethodInvocation来实现。
37. invocation = **new** ReflectiveMethodInvocation(proxy, target,
38. method, args, targetClass,
39. chain);
40. // Proceed to the joinpoint through the interceptor
41. chain.
42. retVal = invocation.proceed();
43. }
44. // Massage return value if necessary.
45. **if** (retVal != **null** && retVal == target &&
46. method.getReturnType().isInstance(proxy) &&
47. !RawTargetAccess.**class**.isAssignableFrom(method.getDe
48. claringClass())) {
49. /\*\*
50. \* Special case: it returned "this" and the return
51. type of the method
52. \* is type-compatible. Note that we can't help if the
53. target sets
54. \* a reference to itself in another returned object.
55. \*/
56. retVal = proxy;
57. }
58. **return** retVal;

通过这个getTarget方法，完成了HotSwappableTargetSource与AOP的集成。这个热交换功能为AOP的使用提供了更多便利，对构建应用的基础服务是非常有帮助的，比如可以在运行时支持经常改变的对象的重新配置。对于其他AOP的高级特性，有兴趣的读者可以结合自己的需要进行分析。