**3.1.2  Advice通知**

Advice（通知）：定义在连接点做什么，为切面增强提供织入接口。在Spring AOP中，它主要描述Spring AOP围绕方法调用而注入的切面行为。Advice是AOP联盟定义的一个接口，具体的接口定义在org.aopalliance.aop.Advice中。在Spring AOP的实现中，使用了这个统一接口，并通过这个接口为AOP切面增强的织入功能做了更多的细化和扩展，比如提供了更具体的通知类型，像BeforeAdvice、AfterAdvice、ThrowAdvice等。作为Spring AOP定义的接口类，具体的切面增强可以通过这些接口集成到AOP框架中去发挥作用。对于这些接口类，我们下面会进行逐一详细地讨论，我们从接口BeforeAdvice开始，首先了解它的类层次关系，如图3-2所示。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091222/142230122.jpg> |
|  |

在BeforeAdvice的继承关系中，定义了为待增强的目标方法设置的前置增强接口MethodBeforeAdvice，使用这个前置接口需要实现一个回调函数：

1. **void** before(Method method, Object[] args,
2. Object target) **throws** Throwable;

作为回调函数，before方法的实现在advice中被配置到目标方法后，会在调用目标方法时被回调。具体的调用参数有：Method对象，这个参数是目标方法的反射对象；object[]对象数组，这个对象数组中包含的是目标方法的输入参数。以CountingBeforeAdvice为例，用它来说明BeforeAdvice的具体使用，CountingBeforeAdvice是接口MethodBeforeAdvice的具体实现，如代码清单3-1所示，它的实现比较简单，完成的工作是统计被调用的方法次数。作为切面增强实现，它会根据调用方法的方法名进行统计，把统计结果根据方法名和调用次数作为键值对放入一个map中。

代码清单3-1  CountingBeforeAdvice的实现

1. **public** **class** CountingBeforeAdvice **extends** MethodCounter **implements** MethodBeforeAdvice {
2. //实现before回调接口，这是接口MethodBeforeAdvice的要求。
3. **public** **void** before(Method m, Object[] args, Object target)
4. **throws** Throwable {
5. count(m);
6. }
7. }

这里调用了count方法，使用了目标方法的反射对象作为参数，完成对调用方法名的统计工作。count方法在CountingBeforeAdvice的基类MethodCounter中实现，如代码清单3-2所示。这个切面增强完成的统计实现并不复杂，它在对象中维护一个哈希表，用来存储统计数据。在统计过程中，首先通过目标方法的反射对象得到方法名，然后进行累加，把统计结果放到维护的哈希表中。如果需要统计数据，就到这个哈希表中根据key来获取。

代码清单3-2  MethodCounter实现统计目标方法调用次数

1. **public** **class** MethodCounter **implements** Serializable {
2. // 这个Hashmap用来存储方法名和调用次数的键值对。
3. **private** HashMap<String, Integer> map = **new** HashMap<String,
4. Integer>();
5. //所有的调用次数，不管是什么方法名。
6. **private** **int** allCount;
7. //CountingBeforeAdvice的调用入口。
8. **protected** **void** count(Method m) {
9. count(m.getName());
10. }
11. //根据目标方法的方法名统计调用次数。
12. **protected** **void** count(String methodName) {
13. Integer i = map.get(methodName);
14. i = (i != **null**) ? **new** Integer(i.intValue() + 1) :
15. **new** Integer(1);
16. map.put(methodName, i);
17. ++allCount;
18. }
19. //根据方法名取得调用的次数。
20. **public** **int** getCalls(String methodName) {
21. Integer i = map.get(methodName);
22. **return** (i != **null** ? i.intValue() : 0);
23. }
24. //取得所有的方法调用次数。
25. **public** **int** getCalls() {
26. **return** allCount;
27. }
28. **public** **boolean** equals(Object other) {
29. **return** (other != **null** && other.getClass() ==
30. **this**.getClass());
31. }
33. **public** **int** hashCode() {
34. **return** getClass().hashCode();
35. }
36. }

在Advice的实现体系中，Spring还提供了AfterAdvice的这种通知类型，它的类接口关系如图3-3所示。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091222/142526527.jpg> |
|  |

在如图3-3所示的AfterAdvice类接口关系中，有一系列对AfterAdvice的实现和接口扩展，比如AfterReturnningAdvice。我们以AfterReturningAdvice通知的实现为例，分析一下AfterAdvice通知类型的实现原理。在AfterReturningAdvice接口中定义了接口方法，如下所示。

1. **void** afterReturning(Object returnValue, Method method,
2. Object[] args, Object
3. target) **throws** Throwable;

AfterReturing方法也是一个回调函数，AOP应用需要在这个接口实现中，提供切面增强的具体设计，在这个Advice通知被正确配置以后，在目标方法调用结束并成功返回的时候，接口实现会被Spring AOP回调。对于回调参数，有目标方法的返回结果、反射对象以及调用参数（AOP把这些参数都封装在一个对象数组中传递进来）等。与前面分析BeforeAdvice一样，在Spring AOP的包中，同样可以看到一个CountingAfterReturningAdvice，作为熟悉AfterRetur-

ningAdvice使用的例子；它的实现基本上与CountingBeforeAdvice是一样的，如代码清单3-3所示。

代码清单3-3  CountingAfterReturningAdvice的实现

1. **public** **class** CountingAfterReturningAdvice **extends**
2. MethodCounter **implements**
3. AfterReturningAdvice {
4. **public** **void** afterReturning(Object o, Method m,
5. Object[] args, Object target)
6. **throws** Throwable {
7. count(m);
8. }
9. }

在实现AfterReturningAdvice的接口方法AfterReturning中，可以调用MethodCounter的count方法，从而完成根据方法名来对目标方法调用的次数进行统计。count方法调用的实现与CountingBeforeAdvice基本上是一样的，所不同的是调用发生的时间。尽管增强逻辑相同，但是，如果它实现不同的AOP通知接口，就会被AOP编织到不同的调用场合中。尽管它们完成的增强行为是一样的，都是根据目标方法名对调用次数进行统计，但是它们的最终实现却有很大不同，一个是在目标方法调用前实现切面增强，一个是在目标方法成功调用返回结果后实现切面增强。由此可见，AOP技术给应用带来的灵活性，使得相同的代码完全可以根据应用的需要灵活地出现在不同的应用场合。

了解了BeforeAdvice和AfterAdvice，在Spring AOP中还可以看到另外一种Advice通知类型--ThrowsAdvice，它的类层次关系如图3-4所示。

|  |
| --- |
| <http://images.51cto.com/files/uploadimg/20091222/142639751.jpg> |
|  |

ThrowsAdvice并没有指定需要实现的接口方法，它在抛出异常时被回调，这个回调是AOP使用反射机制来完成的。我们可以通过CountingThrowsAdvice了解ThrowsAdvice的使用方法，如代码清单3-4所示。

代码清单3-4  CountingThrowsAdvice的实现

1. **public** **static** **class** CountingThrowsAdvice
2. **extends** MethodCounter **implements** ThrowsAdvice {
3. **public** **void** afterThrowing(IOException ex)
4. **throws** Throwable {
5. count(IOException.**class**.getName());
6. }
8. **public** **void** afterThrowing(UncheckedException ex)
9. **throws** Throwable {
10. count(UncheckedException.**class**.getName());
11. }
12. }

在AfterThrowing方法中，从输入的异常对象中得到异常的名字并进行统计。count方法同样是在MethodCounter中实现，与前面的两个Advice的实现相同，只是CountingBeforeAdvice和CountingAfterReturningAdvice统计的是目标方法的调用次数。在这里，count方法完成的是根据异常名称统计抛出异常的次数。