**3.5  小结**

在本章中，我们对最基本Spring AOP的实现方式进行了解析。具体地说，通过使用ProxyFactoryBean/ProxyFactory的实现原理作为例子，对Spring AOP的基本实现和工作原理进行了一些梳理和分析。ProxyFactoryBean是在IoC环境中创建代理的一个很灵活的方法，与其他方法相比，虽然有些繁琐，但却并不妨碍我们从ProxyFactoryBean入手，去了解AOP在Spring中的基本实现。

在Spring AOP的基本实现中，可以了解Spring是如何得到AopProxy代理对象的，以及它是如何利用AopProxy代理对象来对拦截器进行处理的。Proxy代理对象的使用，在Spring AOP的实现过程中是非常重要的一个部分，Spring AOP充分利用了像Java的Proxy、反射技术以及第三方字节码技术实现CGLIB这些技术方案，通过这些技术完成了AOP需要的AopProxy代理对象的生成。回顾整个通过ProxyFactoryBean实现AOP的过程我们可以看到，在它的实现中，首先需要对目标对象进行正确的配置，以及对拦截器的正确配置，以便AopProxy代理对象得以顺利产生；这些配置既可以通过配置ProxyFactoryBean的属性来完成，也可以通过编程式的使用ProxyFactory来实现。这两种AOP的使用方式，只是在表面配置的方式上不同而已，对于内在的AOP实现原理它们是一样的。在生成AopProxy代理对象的时候，Spring AOP设计了专门的AopProxyFactory作为AopProxy代理对象的生产工厂，由它来负责产生相应的AopProxy代理对象，在使用ProxyFactoryBean来得到AopProxy代理对象的时候，它默认地使用的AopProxy代理对象的生产工厂是DefaultAopProxyFactory对象。这个对象是一个在AopProxy生产过程中比较重要的类，它定义了AopProxy代理对象的生成策略，从而决定使用哪一种AopProxy代理对象的生成技术，是使用JDK的Proxy类还是使用CGLIB来完成生产任务。而对于最终的AopProxy代理对象的产生，则是交给JdkDynamicAopProxy和Cglib2AopProxy这两个具体的工厂来完成AopProxy代理对象的生产的，它们使用了不同的生产技术，一种使用的是JDK的Proxy技术，它使用InvocationHandler对象的invoke完成回调，而另一种则是使用CGLIB的技术来生成AopProxy代理对象。

在得到AopProxy代理对象后，在代理的接口方法被调用执行的时候，也就是当AopProxy暴露代理的方法被调用的时候，前面定义的Proxy机制就起作用了。当Proxy对象暴露的方法被调用时，并不是直接地运行目标对象的调用方法，而是会根据Proxy的定义，改变了原有的目标对象方法调用的运行轨迹。这种改变体现在，首先会触发对这些方法调用进行拦截，这些拦截为对目标调用的功能增强提供了工作空间；拦截过程在JDK的Proxy代理对象中，是通过invoke方法来完成的，这个invoke方法是虚拟机触发的一个回调。而在CGLIB的Proxy代理对象中，拦截是由设置好的回调callback方法来完成的。有了这些拦截器的拦截作用，才会有AOP切面增强大显身手的舞台。

在ProxyFactoryBean的回调中，首先会根据配置来对拦截器是否与当前的调用方法相匹配来进行判断。如果当前的调用方法与配置的拦截器相匹配，那么相应的拦截器就会开始发挥作用，这个过程是一个遍历的过程，它会遍历在Proxy代理对象中设置的拦截器链中的所有拦截器。经过这个过程后，在代理对象中定义好的拦截器链里的拦截器会被逐一调用，直到整个拦截器的调用完成为止。在对拦截器的调用完成以后，才是我们最后看到的对目标对象（target）的方法调用。这样，一个普通的Java对象的功能就得到了增强，这种增强和现有的目标对象的设计是正交解耦的，这也是AOP需要达到的一个目标。

在拦截器的调用过程中，实际上已经封装了Spring对AOP的实现，比如对各种通知器的增强织入功能。尽管我们在使用Spring AOP的时候，看到的是一些advice的使用，但实际上这些AOP应用中接触到的这些advice通知是不能直接对目标对象完成增强的，为了完成AOP应用需要的对目标对象的增强，Spring AOP做了许多工作。这些工作包括，对应于每种advice通知，Spring设计了对应的AdviceAdapter通知适配器，正是这些AdviceAdapter通知适配器，实现了advice通知对目标对象的不同的增强方式。对于这些AdviceAdapter通知适配器，在AopProxy代理对象的回调方法中，需要有一个注册机制，它们才能发挥作用。完成这个注册过程之后，实际上在拦截器链中运行的拦截器，已经是经过这些AdviceAdapter适配过的拦截器了。有了这些拦截器，再去结合AopProxy代理对象的拦截回调机制，才能够让advice通知对目标对象的增强作用实实在在地发生。谁知盘中餐，粒粒皆辛苦，在软件开发的世界里，真是没有什么免费午餐。看起来简洁易用的AOP，和IoC容器的实现一样，背后同样蕴含着许多艰苦的努力。

熟悉AOP使用的读者还知道，除了提供AOP的一些基本功能之外，Spring还提供了许多其他高级特性让用户更加方便地使用AOP。对于这些高级特性，在本章中，我们选取了HotSwappableTargetSource来对它的实现原理进行分析，一叶知秋，一管窥豹，希望能够在这里为那些对AOP其他特性的实现感兴趣的读者打开一扇窗。

在本章中，我们提到了Proxy、反射等Java虚拟机特性的使用，CGLIB的使用以及在它们建立的Proxy对象的基础上，对拦截器特性的灵活运用，这些特性都是我们掌握本章内容的背景知识和重要基础。同时，不妨可以反过来看，通过了解本章中AOP的实现原理，也为我们使用这些Java虚拟机的特性以及CGLIB的技术，提供了生动而精彩的应用案例。在AOP的实现中，还有一个值得注意的地方就是，在ProxyFactoryBean得到advisor配置的实现过程中，是通过回调IoC容器的getBean方法来完成的，这个处理既简洁又巧妙，是灵活使用IoC容器功能的一个非常好的实例。以上这些，都是在本章中，除了Spring AOP实现原理本身之外，非常值得我们学习和研究的地方。