**第3章 Spring AOP的实现**

好雨知时节，当春乃发生。

随风潜入夜，润物细无声。

野径云俱黑，江船火独明。

晓看红湿处，花重锦官城。

--【唐】杜甫《春夜喜雨》

**3.1  Spring AOP概述**

**3.1.1  AOP概念回顾**

AOP是Aspect-Oriented Programming（面向方面编程）的简称，维基百科对它的解释如下所示。

维基百科对"AOP"相关概念的叙述

Aspect是一种新的模块化机制，用来描述分散在对象、类或函数中的横切关注点（crosscutting concern）。从关注点中分离出横切关注点是面向侧面的程序设计的核心概念。分离关注点使得解决特定领域问题的代码从业务逻辑中独立出来，业务逻辑的代码中不再含有针对特定领域问题代码的调用，业务逻辑同特定领域问题的关系通过侧面来封装、维护，这样原本分散在整个应用程序中的变动就可以很好地管理起来。

这里提到的概念是从模块化出发的，面向对象设计其实也是一种模块化的方法，它把相关的数据及其处理方法放在了一起，与单纯的使用子函数进行封装相比，面向对象的模块化特性更完备，它体现了计算的一个基本原则--让计算尽可能地靠近数据。这样一来，代码组织起来就更加整齐和清晰，一个类就是一个基本的模块。很多程序的功能还可以通过设计类的继承关系而得到重用，进一步提高了开发效率。后来又出现了各种各样的设计模式供我们使用，让我们在设计程序功能时更加得心应手。

虽然我们利用面向对象的方法可以很好地组织代码，也可以通过继承关系实现代码重用，但是程序中总是会出现一些重复的代码，而且不太方便使用继承的方法把它们重用和管理起来。它们功能重复并且需要作用在不同的地方，虽然可以对这些代码做一些简单的封装，使之成为一些公共函数，但是在这种显式的调用中，使用它们并不是很方便。例如，这个公共函数在什么情况下可以使用，能不能更灵活地使用，等等。

在使用这些公共函数时，往往也需要进行一些逻辑设计。也就是需要代码实现来支持，而这些逻辑代码也是需要维护的。这时就是AOP大显身手的时候了，使用AOP后，不仅可以将这些重复的代码抽取出来单独维护，在需要使用时统一调用这些公共代码，还可以为如何使用这些公共代码提供丰富灵活的手段。这虽然与设计公共子模块有几分相似，但在传统的公共子模块调用中，除了直接硬调用之外并没有其他的手段，而AOP为处理这一类问题提供了一套完整的理论和灵活多样的实现方法。也就是说，通过AOP提出横切的概念以后，在把模块功能正交化的同时，也在此基础上提供了一系列横切的灵活实现。比如通过使用Proxy代理对象、拦截器、字节码翻译技术等，通过这一系列已有的AOP或者AOP实现技术，来实现切面应用的各种编织实现和环绕增强；为了更好地应用AOP技术，技术专家们还成立了AOP联盟来探讨AOP的标准化，有了这些支持，AOP的发展就更快了。关于AOP技术，可以到AOP联盟的文档里找到一些相关的介绍，从而提高我们对AOP的理解。比如在AOP联盟的网站上提到的以下AOP技术：

AspectJ：源代码和字节码级别的编织器，用户需要使用不同于Java的新语言。

AspectWerkz：AOP框架，使用字节码动态编织器和XML配置。

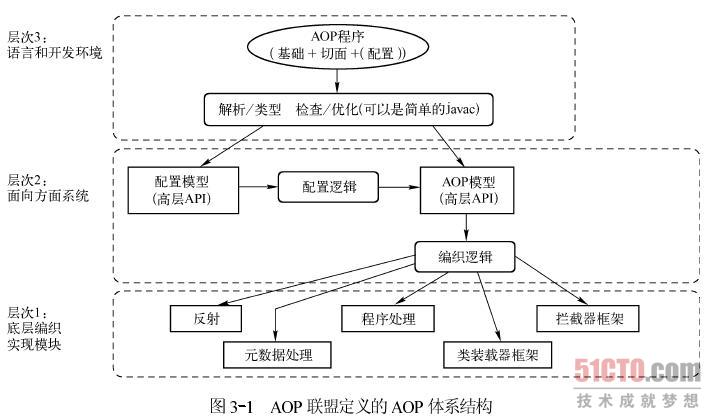
JBoss-AOP：基于拦截器和元数据的AOP框架，运行在JBoss应用服务器上。

以及在AOP中使用到的一些相关的技术实现：

BCEL（Byte-Code Engineering Library）：Java字节码操作类库，具体的信息可以参见项目网站：<http://jakarta.apache.org/bcel/>。

Javassist：Java字节码操作类库，JBoss的一个子项目，项目信息可以参见项目网站：<http://jboss.org/javassist/>。

对应于现有的AOP实现方案，AOP联盟对它们进行了一定程度的抽象，从而定义出AOP体系结构，结合这个体系结构去了解AOP技术，对我们理解AOP的概念是非常有帮助的，这个AOP体系结构如图3-1所示。

[](http://images.51cto.com/files/uploadimg/20091222/0957100.jpg)

AOP联盟定义的AOP体系结构把与AOP相关的概念大致分为了由高到低、从使用到实现的三个层次。从上往下，最高层是语言和开发环境，在这个环境中可以看到几个重要的概念：基础可以视为待增强对象或者说目标对象；切面通常包含对于基础的增强应用；配置可以看成是一种编织或者说配置，通过在AOP体系中提供这个配置环境，可以把基础和切面结合起来，从而完成切面对目标对象的编织实现。

在Spring AOP实现中，是使用Java语言来实现增强对象与切面增强应用的，并为这两者的结合提供了配置环境。对于编织配置，可以使用IoC容器来完成；对于POJO对象的配置，本来就是Spring的核心IoC容器的强项。因此，对于使用Spring的AOP开发而言，使用POJO就能完成AOP任务。但是，对于其他的AOP实现方案，可能需要使用特定的实现语言、配置环境甚至是特定的编译环境。例如在AspectJ中，尽管切面增强的对象是Java对象，但却需要使用特定的Aspect语言和AspectJ编译器。AOP体系结构的第二个层次是为语言和开发环境提支持的，在这个层次中可以看到AOP框架的高层实现，主要包括配置和编织实现两部分内容。例如配置逻辑和编织逻辑实现本身，以及对这些实现进行抽象的一些高层API封装。这些实现和API封装，为前面提到的语言和开发环境的实现提供了有力的支持。

最底层是编织的具体实现模块，图3-1中看到的各种技术，都可以作为编织逻辑的具体实现方法，比如反射、程序处理、拦截器框架、类装载框架、元数据处理等。阅读完本章对Spring AOP实现原理的分析，我们会了解到，在Spring AOP中，使用的是Java本身的语言特性，比如Java Proxy代理类、拦截器这些技术，来完成AOP编织的实现。

对Spring平台或者说生态系统来说，AOP是Spring框架的核心功能模块之一。AOP与IoC容器的结合使用, 为应用开发或者Spring自身功能的扩展都提供了许多便利。Spring AOP的实现和其他特性的实现一样，除了可以使用Spring本身提供的AOP实现之外，还封装了业界优秀的AOP解决方案AspectJ来让应用使用。在本章中，我们主要对Spring自身的AOP实现原理进行分析；在这个AOP实现中，Spring充分利用了IoC容器Proxy代理对象以及AOP拦截器的功能特性，通过这些对AOP基本功能的封装机制，为用户提供了AOP的实现框架。所以，要了解这些AOP的基本实现，需要我们对Java 的Proxy机制有一些基本了解。在Spring中，有一些相关的概念与AOP设计相对应；在本章的内容中，按照笔者个人的粗浅理解，我们会结合Spring的AOP实现，温故而知新，在以下的小节中对一些相关的AOP概念，先简单地回顾一下；然后再逐步展开对AOP实现原理的分析，包在这些实现原理的分析中，包括了代理对象的生成，AOP拦截器的实现，等等。这里以ProxyFactoryBean和ProxyFactory为例子来进行说明。通过分析它们背后隐藏的实现原理来了解Spring的AOP模块。和前面一样，我们需要在Eclipse环境中导入对应的AOP代码模块来辅助这些原理实现的分析，具体需要导入的包是：org.springframework.aop，这个代码包里包含了Spring AOP的实现代码。