**Spring-AOP切面编程指导手册**

# 第一部分 概述

面向方面编程(AOP)补充了面向对象编程(OOP)，提供了另一种思考程序结构的方式。OOP中模块化的关键单元是类，而AOP中模块化的单元是方面。方面支持跨多个类型和对象的关注点(例如事务管理)的模块化。(在AOP文献中，这种关注通常被称为“横切”关注。)。

Spring的关键组件之一是AOP框架。虽然Spring IoC容器不依赖于AOP(也就是说，如果不想使用AOP，就不需要使用AOP)，但AOP补充了Spring IoC，提供了一个非常强大的中间件解决方案。

Spring AOP 2.0 +

Spring 2.0通过使用基于模式的方法或@AspectJ注释样式，引入了一种更简单、更强大的方式来编写定制方面。这两种样式都提供了完全类型化的建议和AspectJ切入点语言的使用，同时仍然使用Spring AOP进行编织。

本章讨论基于Spring 2.0+ schema和@ aspectj的AOP支持。下一章将讨论在Spring 1.2应用程序中常见的较低级别的AOP支持。

在Spring框架中使用AOP:

1. 提供声明性企业服务，特别是作为EJB声明性服务的替代。这种服务最重要的是声明性事务管理。
2. 让用户实现自定义方面，用AOP补充他们对OOP的使用。

注意，如果您只对泛型声明性服务或其他预打包声明性中间件服务(如数据池)感兴趣，则不需要直接使用Spring AOP，并且可以跳过本章的大部分内容。

# 第二部分 概念和术语

## 2.1 AOP Concepts

让我们从定义一些核心的AOP概念和术语开始。这些术语不是特定于spring的。不幸的是，AOP术语不是特别直观。然而，如果Spring使用它自己的术语，那就更令人困惑了。

* Aspect:

跨多个类的关注点的模块化。事务管理是企业Java应用程序中横切关注点的一个很好的例子。在Spring AOP中，方面是通过使用常规类(基于模式的方法)或使用@Aspect注释(@AspectJ样式)的常规类实现的。

* Join point:

程序执行过程中的一个点，如方法的执行或异常的处理。在Spring AOP中，连接点总是表示方法的执行。

* Advice:

方面在特定连接点上采取的动作。不同类型的建议包括“前后”、“前后”和“前后”。(通知类型将在后面讨论。)许多AOP框架，包括Spring，将通知建模为拦截器，并在连接点周围维护拦截器链

* Pointcut:

匹配连接点的谓词。通知与切入点表达式关联，并在与切入点匹配的任何连接点上运行(例如，具有特定名称的方法的执行)。连接点与切入点表达式匹配的概念是AOP的核心，Spring默认使用AspectJ切入点表达式语言。

* Introduction:

代表类型声明其他方法或字段。Spring AOP允许向任何建议的对象引入新的接口(以及相应的实现)。例如，您可以使用介绍来让bean实现IsModified接口，从而简化缓存。(在AspectJ社区中，引入被称为类型间声明。)

* Target object:

被一个或多个方面通知的对象。也称为“建议对象”。因为Spring AOP是通过运行时代理实现的，所以这个对象总是一个代理对象。

* AOP proxy:

AOP框架为实现方面契约(通知方法执行等等)而创建的对象。在Spring框架中，AOP代理是JDK动态代理或CGLIB代理。

* Weaving:

将方面与其他应用程序类型或对象链接起来以创建建议对象。这可以在编译时(例如使用AspectJ编译器)、加载时或运行时完成。与其他纯Java AOP框架一样，Spring AOP在运行时执行编织。

Spring AOP包括以下类型的通知

Before advice:在连接点之前运行的advice，但是它不能阻止执行流继续到连接点(除非它抛出一个异常)。

After returning advice:在连接点正常完成之后运行的通知(例如，如果方法返回而没有抛出异常)

After throwing advice: 如果方法通过抛出异常而退出，则执行通知。

After (finally) advice: 无论连接点以何种方式退出(正常或异常返回)，都要执行通知。

Around advice: 围绕连接点(如方法调用)的通知。这是最有力的建议。Around通知可以在方法调用前后执行自定义行为。它还负责选择是继续到连接点，还是通过返回自己的返回值或抛出异常来缩短建议的方法执行。

环绕通知是最普遍的通知。因为Spring AOP，比如AspectJ，提供了各种各样的通知类型，所以我们建议您使用能够实现所需行为的功能最少的通知类型。例如，如果您只需要用方法的返回值更新缓存，那么您最好实现after返回通知而不是around通知，尽管around通知可以完成相同的任务。使用最特定的通知类型可以提供更简单的编程模型，出错的可能性更小。例如，您不需要在用于around通知的连接点上调用proceed()方法，因此，您不能不调用它.

在Spring 2.0中，所有通知参数都是静态类型的，这样您就可以使用适当类型的通知参数(例如，方法执行返回值的类型)，而不是对象数组。

连接点与切入点匹配的概念是AOP的关键，它区别于只提供拦截的旧技术。切入点使通知能够独立于面向对象的层次结构成为目标。例如，您可以将提供声明性事务管理的around通知应用到一组跨多个对象的方法(例如服务层中的所有业务操作)。

## Spring AOP功能和目标

Spring AOP是在纯Java中实现的。不需要特殊的编译过程。Spring AOP不需要控制类装入器层次结构，因此适合在servlet容器或应用服务器中使用。

Spring AOP目前只支持方法执行连接点(建议在Spring bean上执行方法)。虽然可以在不破坏Spring AOP核心api的情况下添加对字段拦截的支持，但是没有实现字段拦截。如果需要建议字段访问和更新连接点，请考虑使用诸如AspectJ之类的语言。

Spring AOP对AOP的方法不同于大多数其他AOP框架。目标不是提供最完整的AOP实现(尽管Spring AOP很有能力)。相反，其目标是提供AOP实现与Spring IoC之间的紧密集成，以帮助解决企业应用程序中的常见问题。

因此，例如，Spring框架的AOP功能通常与Spring IoC容器一起使用。方面是通过使用普通bean定义语法来配置的(尽管这允许强大的“自动代理”功能)。这是与其他AOP实现的一个关键区别。在Spring AOP中，您无法轻松或高效地完成一些事情，比如建议非常细粒度的对象(通常是域对象)。在这种情况下，AspectJ是最好的选择。然而，我们的经验是，Spring AOP为符合AOP的企业Java应用程序中的大多数问题提供了优秀的解决方案.

Spring AOP从未试图与AspectJ竞争，以提供全面的AOP解决方案。我们认为基于代理的框架(如Spring AOP)和成熟的框架(如AspectJ)都是有价值的，它们是互补的，而不是竞争的。Spring无缝地将Spring AOP和IoC与AspectJ集成在一起，以支持在一致的基于Spring的应用程序体系结构中使用AOP。这种集成不会影响Spring AOP API或AOP Alliance API。Spring AOP仍然是向后兼容的。有关Spring AOP api的讨论，请参阅下一章。

注意：

Spring框架的核心原则之一是非入侵性。您不应该被迫将特定于框架的类和接口引入业务模型或域模型。但是，在某些地方，Spring框架确实为您提供了在代码基中引入特定于Spring框架的依赖项的选项。提供这样的选项的基本原理是，在某些场景中，以这种方式读取或编码某些特定的功能可能更简单。然而，Spring框架(几乎)总是为您提供选择:您可以自由地做出明智的决定，知道哪个选项最适合您的特定用例或场景。

与本章相关的一种选择是选择哪个AOP框架(以及哪个AOP风格)。您可以选择AspectJ、Spring AOP，或者两者都选择。您还可以选择@AspectJ注释样式的方法或Spring XML配置样式的方法。本章选择先介绍@AspectJ风格的方法，这一事实不应该被认为表明Spring团队更喜欢@AspectJ注释风格的方法，而不是Spring XML配置风格。

请参阅选择使用哪种AOP声明样式，以更完整地讨论每种样式的“为什么和哪里”。

## AOP Proxies