介绍AJ

该章主要包括：

写一个AJ的helloworld应用

熟悉aj语言

织入机制

与spring的集成

在第一章中，我们讨论了AOP的概念。在这之后，我们现在来看一种AOP的详细的实现：AJ。AJ是JAVA面向切面编程的扩展。像其他的AOP实现一样，AJ由两部分组成：语言规范，规定了语法和语义；语言的实现，包含了可以采用编译和连接两种方式来工作的织入者。一个织入者产生遵循JAVA字节码规范的字节码，允许任何JVM执行这些字节码。语言的实现也为IDE的集成提供了支持，可以简化建立和调试应用。

AJ最初开始发展时，目标是成长为一种继承java语言并具有一些关键字的语言。同时它也提供了一种特殊的编译器来支持这些扩展关键字。但是近年来，在语言层次上发生了很多的变化，织入者方面同样如此。首先，在横切点构造的表达方面，AJ基于java注解提供了一种可供选择的语法。这可以允许你使用普通的JAVA编译器而不再局限于专用的编译器。其次，AJ使用切面为织入过程提供了新的选择。最后，通过一些集成选项，在spring框架中，AJ有了很强的立足点。以上所有的这些变化，都让AJ比之前有了更强的易用性。

在这一章，我们将会讨论AJ较为重要的几个方面，从一个较高的视角来看，我们首先从语言的构造开始，分析语法和织入选择，窥探spring的集成，最后以讨论相关工具支持结束。在接下来的章节中，我们会深入探索每个部分。

2.1 编写第一个AJ程序。

通过写一个简单的应用来开始你的旅程。这段代码介绍了一些AJ的概念，并且会给予你AJ语言基本的感觉。

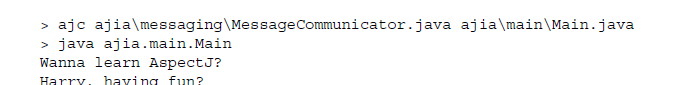
安装AJ

2.1.1 第一个例子。

如2.1那样，我们建立一个普通的java类，该类包含了两个可以打印信息的方法。稍后，你会创建一些切面，从而不需要修改这些类就可以为他们增加额外的行为。

MessageCommunicator类有两个方法，一个可以发送一段普通的信息，另一个会对特定的人发送信息。加下来，让我们写一个简单的类来执行这个类的功能。如2.2所示。

当你使用AJC编译器联合编译MessageCommunicator和Main类并且运行Main类时，输出如下（该章都是基于win平台的命令行，请根据你自己的平台进行适当的更改。）：



运行示例

进行命令行操作的主要原因是消除我们对AJ是如何工作的相关疑问。可供下载的源码包含了shell脚本，这些脚本可以匹配这些命令，也包含了必要的设置。但是很多开发者更喜欢通过ANT和Maven脚本来运行这些源码。附录B和C展示了如何使用这些工具来运行AJ的细节。

每个合法的JAVA程序都是一个合法的AJ程序。因此，你可以使用AJC编译器来替代JAVAC编译器编译这些类。

现在，基本的设置已经配置完成，接下来让我们来为这个系统增加一些切面来改善消息发送功能。

2.1.2 增加一个切面

考虑一个认证功能：在发送消息之前，你需要检查用户是否已被认证。如果不使用AOP，你不得不像下面这样编写代码：

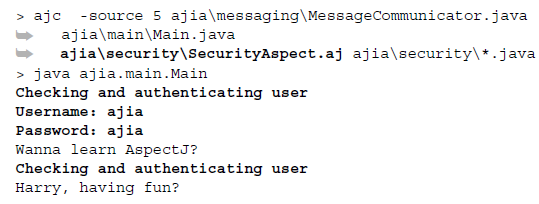
你不得不在每个需要认证的方法里增加对authenticate（）方法的调用，而这会导致代码缠绕。而相似的代码存在于所有的类中，这会导致代码分散。使用AOP，你可以做的更好一点。

不需要更改2.1中的任何一行代码，你可以通过向系统增加一个切面来增强他的功能。2.3展示了传统的语法。稍后，你会看见实现相同功能的但却不一样的语法。



当authenticate()方法在线程中首次调用的时候Authenticator类请求证书（包含用户名和密码）。如果认证成功，会在当前线程储存该用户，因此不会再在相同的线程中再次请求用户的证书信息。如果失败，会抛出一个运行时异常（为简单起见，在这里我并没有展示Authenticator类。你可以从下列网址下载该类的代码）。

编译该类。现在，当你运行该代码的时候，你会卡到如下的输出（使用Authenticator类需要指定-source 5给AJC编译器）。



SecurityAspect.aj文件声明了一个切面。记住，你也可以在SecurityAspect.java文件中声明切面，因为AJ对于输入的文件，接受java和aj两种扩展名。虽然编译器对于文件扩展名并不关心，切面一般都会只用.aj扩展名。JAVA代码一般使用.java扩展名。下面让我们来了解更多这方面的细节。

1. 类似于类是OOP概念中的一个模块，一个切面就是AOP中的一个模块。切面的声明与类的声明很相似。
2. 在一个系统中，一个pointcut会在被称为连接点的一系列点中选择它所感兴趣的执行点。在MessageCommunicator类中，切面定义了一个secureAccess（）切点，该切点会选择该类中所有名字为deliver的方法的执行点。\*表明该切点匹配所有类型的返回这，..表明在deliver后面的括号里允许匹配任意类型和个数的参数。在这个例子中，切点选择了MessageCommunicator类deliver（）方法的所有重载版本的执行点。在下一章，你会了解到有关加入点和切点的详细细节。
3. 增加定义了一段可执行的代码。这段代码会在程序到达切点定义的加入点处时得到执行。这里，你定义了一个在secureAccess()切点之前需要执行的代码。Before（）表明增强会在被增强点之前得到优先执行。在我们的例子中，即优先于MessageCommunicator.deliver()方法的执行。在增强逻辑中，你认证了当前的用户。现在，使用系统中已存在的切面逻辑，每当MessageCommunicator.deliver()方法被执行的时候，在它执行之前，增强代码都会执行认证逻辑。

现在，你已经感受到了AJ语言的味道，是时候对AJ的语言和它核心的部分进行一个全面的了解了。

2.2 AJ横切点设计

回顾在第一章中一般的AOP模型。AJ是该模型最完整的实现，对模型中的所有元素提供了支持。在这一节中，我们将会讨论AJ是如何将模型的每一个元素映射到构造程序当中的。记住，AJ提供了两种语法选择：传统方式和@AspectJ方式。这一节主要使用传统方式来学习模块的建造，在2.3节中，我们将会讨论aspectJ语法。

我们将AOP模型中横切点的设计分为如下几种：普通横切点设计（加入点，切点，切面），动态横切点设计（增强），静态横切点设计（内部类型声明和织入时机声明）。这些设计组成了AJ的建造部分。首先我们来学习一般的横切点设计。

2.2.1 普通横切点设计

AJ为AOP支持一些普通的设计，这些设计由连接点，切点和切面组成。你可以使用动态和静态横切来应用这些设计。

连接点

在AOP中，同样也是在AJ中，连接点是横切行为发生的地方。在2.1中，连接点对应着deliver()的执行，也对应着System.out对象的println()方法的执行。2.2中的连接点则对应着MessageCommunicator对象的创建与deliver()方法的调用。

当对于一个横切功能，你发现其可用的连接点之后，你需要使用切点设计来选择他们。

切点

一个切点就是一段程序设计，这段程序选择了一些连接点，并且拥有了这些点的上下文环境。例如，一个切点选择了一个连接点，而这个连接点是一个方法的执行处。切点也可以手机连接点的上下文环境，例如this对象和方法的参数。

下列切点就选取了一个系统中所有的public方法。



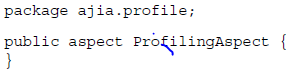
通配符\*和..表明切点会无视返回类型，声明类型，方法名称和方法的参数。在这里，唯一的限制条件是方法的访问限制符必须是Public.

命名一个切点是一个非常好的主意，因为这样其他的程序元素就可以使用这个切点（当然也包括其他程序员，包括你自己，这样你们就能够明白切点的意图）。例如，你可以提前命名一个如下的切点为publicOperation：



**切面**

切面是AJ的中心单元，就像JAVA中类的地位一样。切面包含了一些代码，而这些代码则表示了动态和静态的横切点的织入规则。此外，切面也包含了数据，方法和嵌套内部类成员，就像一个普通JAVA类一样。让我们来定义一个切面，当你学习到更多的元素之后，你会不断地更新对这个切面的分析认识。



学习普通的横切点设计将会让你在静态动态横切点设计中再次使用它们时，感到事半功倍。让我们从动态横切点设计开始，看看这一切是如何发生的。

2.2.2 动态横切点设计：增强。