基于 AOP 的数据库应用安全控制的设计与实现①

董 源[1,2], 李培军[2], 许舒人[2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院软件研究所 软件工程技术研究开发中心, 北京 100190)

摘 要: 数据库的安全访问控制作为系统层级的模块需要在各个模块中通用. 传统的实现方式有两种: 一是采用组件接口调用的方式将安全访问模块编入每个所需的业务逻辑当中, 二是通过 AOP 配置的方式将模块切入. 这两种方式中, 无论是首次调用还是修改调用, 开发人员都需要编写大量的函数调用代码或配置信息. 针对配置信息代码量大、修改处理代价高这一问题, 本文提出一种可视化的 AOP 配置方式, 通过 Dom4j 自动配置方式完成安全访问控制模块的注入, 减少开发人员编写冗余的配置信息, 更加灵活地更改注入的内容.

关键词: AOP; 数据访问; 安全控制; 可视化

Design and Implementation of a Database Application Security Controls Based on AOP

DONG Yuan[1,2], LI Pei-Jun[2], XU Shu-Ren[2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: The security of the database access control needs common used in various modules as a system-level modules. There are two traditional ways to achieve it, the first one is by the way of the component interface call to join the secure access module into each required, and the second one is through the AOP configuration mode. The two ways require developers to write a lot of function code or configuration information. According to the problem of the configuration information for a large quantity of code and the high cost of modify the processing, this paper proposes a visualization of the AOP configuration, access control injection module automatically through Dom4j configuration mode to complete the security access, reduce the configuration information that developers have to written redundancy, as while as make it more flexible to change content.

Key words: AOP; data access; safety control; visualization

1 引言

随着计算机软件广泛应用于各个行业, 软件系统涉及的领域越来越广, 规模越来越大, 复杂性越来越高. 与此同时, 软件系统的安全性也成为人们密切关注的问题. 从代码实现角度讲, 安全模块属于系统的通用模块, 系统各个业务均有涉及. 作为开发人员, 编写这些模块的代码具有很高的冗余性和复杂性[1].

首先, 对于具体业务逻辑点不同但是领域相近的软件系统而言, 除了需要实现各自的功能和领域专业模块外, 还需要单独实现一套自用的安全模块, 如日志管理、权限控制、异常处理和事务管理等. 这些模块虽然处于完全不同的软件系统中, 但由于领域知识的共通性, 其模块功能、应用场景和架构设计在本质上是相似甚至相同的. 在这种情况下, 不同的开发人员仍然需要针对各自系统开发一套独立的安全模块, 大大降低了开发效率.

其次, 对于某个单一系统而言, 其内部的业务逻辑需要频繁调用安全模块来实现系统安全检测. 以最简单的日志管理为例, 为了完成系统完整的日志操作记录, 开发人员几乎需要在系统撰写的每个方法中均

① 基金项目:新闻出版重大科技工程项目 GXTC-CZ-1015004/0;国家科技支撑课题(2012BAH14B02)
收稿时间:2015-03-27;收到修改稿时间:2015-04-26

© 中国科学院软件研究所 http://www.c-s-a.org.cn

调用日志模块的相关函数. 这种实现方式在一定程度上保证了系统较严密的日志管理, 但是其实现成本很高, 也不利于系统的后期维护. 传统的数据库安全访问控制本质上是对用户请求访问的 URL 或数据库 SQL 语句进行拦截, 在其中添加用户筛选等过滤条件, 这种方式需要开发人员频繁调用安全模块.

在目前的研究和解决方案中, 通过 AOP(Aspect Oriented-Programming, 面向切面编程)的方式将诸如系统安全模块这样的通用模块作为横向关注点从系统中提取出来, 再通过切面配置的方式将其织入到所需的业务逻辑中, 即可减少开发人员对通用模块函数调用的依赖[2,3]. 然而这种解决方式同样带来了新的问题: 开发人员在进行切面配置的时候需要编写配置信息代码, 面对庞大而复杂的系统时, 系统的开发者就需要编写大量的配置代码, 这无形中又增加了开发者的负担.

针对上述问题, 本文采用可视化配置的方式, 以数据库安全访问控制模块为例, 在 AOP 配置的基础上进行改进和优化, 旨在设计和完成一套领域可复用的安全模块原型, 同时减少开发人员在 AOP 配置过程中的代码编写, 将配置信息以持久化的形式存储. 在减少配置代码的同时, 能够灵活、便捷的对横切模块的切入策略进行变更, 方便系统使用者和开发者管理和维护安全模块.

2 相关技术介绍

2.1 AOP 介绍

AOP(Aspect Oriented-Programming), 即面向切面编程的主要思想是将业务逻辑中共通的部分从中剥离, 再通过切入的方式统一编织到不同的方法中. 比如日志管理、权限判断等各个方法都需要的内容就可以利用 AOP 的方式实现. 这样做可以使业务逻辑方面的代码更加关注自身的逻辑部分, 同时松耦合的关系也使得横向关注点模块在需求变更时更易修改. AOP 编程思想如图 1 所示.

有关 AOP 的相关概念如下:

(1)切面(Aspect): 一个切面即一个横向关注点模块, 如日志模块、权限管理、异常处理等, 它包括通知和切入点两部分;

(2)通知(Advice): 在特定的联结点执行 AOP 的动作, 是 point cut 执行的代码, 是“方面”执行的具体逻辑;

(3)切入点(Point Cut): 捕获联结点的结构, 指定一个通知引发一系列联结点的集合. 在 AOP 中, point cut 一般用来捕获相关方法的调用;

(4)联结点(Join Point): 在程序执行中明确定义的切入点, 正是联结点提供了描述 AOP 程序的构架. 一般的联结点可以是方法调用前后、类成员的访问或者异常处理的执行等;

(5)引入(Introduce): 添加字段或方法到目标对象, 或者引入新的接口来改变其类结构;

(6)目标对象(Target): 即横向关注点织入的业务逻辑对象. 如果不使用 AOP 进行织入, 其逻辑将要考虑通用的功能需求, 如日志、事物等;

(7)代理对象(proxy): 即 AOP 框架创建的对象, 将通知应用到目标对象后被动态创建;

(8)织入(Weaving): 将切面应用到目标对象后创建新的代理对象的过程;

AOP 的应用场景十分广泛, 可以在权限控制(Authentication)、缓存(Caching)、调试(Debugging)、懒加载(Lazy loading)、持久化(Persistence)、内容传递(Context passing)、异常处理(Error handing)、性能优化(Performance optimization)、事务(Transactions)等情境下使用[4-6]. 本文通过 AOP 的机制对日志管理、权限控制等方面进行优化.

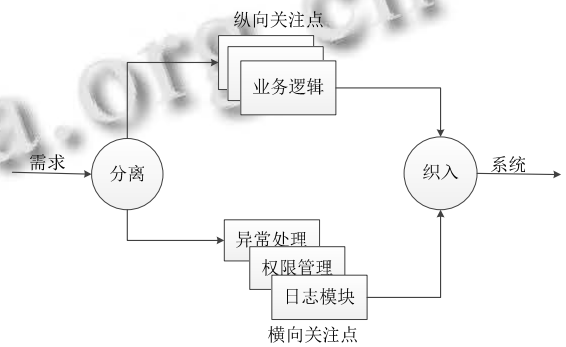

图 1 AOP 编程思想

下面介绍这些横向关注点的模块.

2.2 横向关注点模块

对于任何一个系统而言, 安全性都是需要考虑的因素. 与安全性相关的模块为日志管理、权限控制等. 作为横向关注点模块, 这些模块具有系统通用的特性. 从广义的角度上来讲, 任何能够为系统通用的模块都可以作为横向关注点, 利用面向切面编程的思想实现.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

对于功能众多、业务繁杂的系统而言, 通过 AOP 的方式对这些模块进行编写显得更为便捷. 下面举例介绍这些模块:

日志管理: 任何系统的后台都需要记录日志. 日志模块是典型的利用横向关注点完成的模块, 各个不同的业务逻辑方法中均需要切入日志的记录.

权限管理: 针对拥有用户管理的系统, 权限管理主要负责用户在使用系统中按照系统赋予用户的权限访问有限的操作、界面和数据. 模块操作或数据访问在后台都对应着相应类的方法调用, 这需要开发者在系统编程中对每个方法都设定访问权限, 并且要求访问权限可以根据需求变更. 对大型系统而言, 需要单独的切面类模块来配置方法的使用权限.

2.3 利用 xml 方式进行 Spring AOP 配置

本系统基于 Spring AOP 的机制进行横向关注点的注入. Spring AOP 是一个使用标准的 Java 语音编写的框架, 它的实现途径不同于其他大部分的面向切面编程框架. Spring AOP 的目标是提供一个和 Spring IOC 紧密结合的 AOP 框架, 从而帮助解决企业级应用中出现的一些问题, 因此它并不提供完善的 AOP 实现[8].

在 SSH 框架系统中, Spring 通过两种方式对 AOP 进行配置. 基于 Annotation 的“零配置”方式: 使用 @Aspect、@Pointcut 等 Annotation 来标注切入点和增强处理. 基于 XML 配置文件的管理方式: 使用 Spring 配置文件来定义切入点和增强点. 由于 Annotation 的方式需要对多文件进行标签标注, 因此 XML 配置文件的方式更适合界面化的实现.

Spring AOP 的 XML 配置方式通过在 Spring 的配置文件中编写配置代码来完成 AOP 的实现. 一个完整的 XML 配置方式的配置代码如下:

```
<bean id=”ccBean” class=”aaa.bb.ccBean”>
<aop:config>
    <aop:pointcut id="pc" expression="execution(*
package.class.method(…))"/>
    <aop:advisor/>
    <aop:aspect ref=”ccBean”>
        <aop:pointcut/>
        <aop:before method="before" pointcut-ref="pc"/>
        <aop:after method="after" pointcut-ref=""/>
        <aop:after-returning method="returning"
pointcut-ref=""/>
        <aop:after-throwing method="throwing"
pointcut-ref=""/>
        <aop:around method="around" pointcut-ref=""/>
    </aop:aspect>
</aop:config>
```

<aop:config>: 一段 AOP 程序开始结束符; 配置<aop:config>元素时, 实质是将已有的 Spring Bean 转换成切面 Bean, 所以需要先定义一个普通的 Spring Bean. 因为切面 Bean 可以当成一个普通的 Spring Bean 来配置, 所以我们完全可以为该切面 Bean 配置依赖注入. 当切面 Bean 的定义完成后, 通过<aop:config>元素中 ref 属性来引用该 Bean, 就可以将该 Bean 转换成切面 Bean 了;

<aop:pointcut>: 定义切入点, 即在什么位置切入. Pointcut 的作用是明确地为切入的位置进行重命名, 帮助切面中的通知便捷的找到切入点. 切入点可以为零个至多个; 当把<aop:pointcut../>元素作为<aop:config../>的子元素时, 表明该切点可以被多个切面共享; 当把<aop:pointcut../>元素作为<aop:aspect../>的子元素时, 表明该切点只能在这个切面内使用;

<aop:advisor/>: 定义只有一个切入点和通知的切面, 在点对点配置时使用;

<aop:aspect>: 定义一个完整的切面, 其中包括了切入点 pointcut、切入方式和切入内容 ref; 一个切面的切入内容可以同时被多个切入点调用, 这也是面向切面编程的核心所在;

<aop:before>: 前置切入方式, 即切入内容在切入点函数调用执行前切入;

<aop:after>: 后置切入方式, 即切入内容在切入点函数调用执行完毕切入;

<aop:after-returning>: 返回通知切入方式, 在调用函数返回 return 值后切入;

<aop:after-throwing >: 异常处理切入方式, 在切入点抛出异常时切入;

<aop:around>: 环绕切入方式, 即在切入方法前后均执行切入代码;

限于篇幅, 还有一些诸如优先级等其它使用场景较小的属性, 本文在此便不再列出.

2.4 通过 Dom4j 完成配置文件生成

Dom4j 是一个 Java 的 XML API, 和 jdom 一样可以用来读写 XML 文件. 通过 dom4j 的解析和 XML 转换模块可以快速将数据库中的数据生成到 XML 配置

© 中国科学院软件研究所 http://www.c-s-a.org.cn

文件中, 也可以对已有的配置信息进行修改, 从而完成系统的最后一步.

3 系统架构设计

对于一个复杂的软件工程系统而言, 通用模块横切入大量的业务逻辑模块需要开发人员手动编写许多相似的配置代码. 基于 AOP 的数据库应用安全控制系统旨在通过可视化、界面化的配置方式为开发人员提供 AOP 编程上的帮助. 开发人员无需了解面向切面编程的具体实现细节, 也无需手动编写 AOP 的配置文件, 在可视化的配置界面中进行选择和简单的编写即可完成整个 AOP 的配置工作. 同时, 对于日志管理、权限控制等通用模块的运用也做到了灵活性、可复用性.

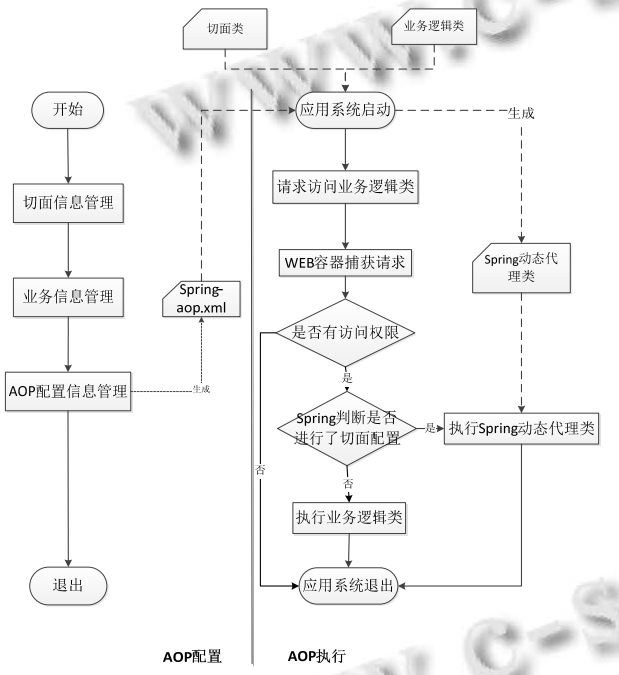

图 2 系统流程图

如图 2 所示为系统的流程图, 分为 AOP 配置和 AOP 执行两阶段. 在根据配置文件的信息进行数据安全访问控制之前, 需要进行相应的准备工作.

在 AOP 配置阶段, 开发人员可以根据系统需要进行切面信息管理、业务信息管理和配置信息管理. 前文提到, 切面和业务信息本质上也是类文件, 进行切面信息管理和业务信息管理实质上是指定系统中哪些类文件作为横向关注点、哪些作为纵向关注点. 在本系统中, 以权限管理中的数据安全访问策略部分作为切面类.

AOP 配置信息管理的过程则是将这些关注点切到系统需要的业务逻辑中. AOP 配置信息可随时更改、启用、失效, 这是通过传统的代码编写方式所做不到的, 而在本系统中通过修改配置文件操作即可自动化完成这一功能.

进行切面信息管理和业务信息管理分别需要系统切面类和业务逻辑类的支持, 最终生成切面信息和业务信息到数据库供 AOP 配置信息管理时使用. AOP 配置信息管理最终生成 Spring-aop.xml 文件供应用系统使用. 通过以上几项准备, 系统的 AOP 配置工作完毕.

在 AOP 执行阶段, 应用系统在启动时会根据切面类、业务逻辑类和 Spring-aop.xml 文件生成 Spring 动态代理类. 该动态代理类实质上是将切面方法和业务方法组合成一个新的类文件供系统使用. 用户在请求访问业务逻辑类时, WEB 容器首先捕获到请求访问, 对请求进行访问权限判断, 判断用户有权访问后便询问 Spring 访问的函数是否进行了切面配置. 如果访问请求设置了切面配置, 则执行系统启动时生成的 Spring 动态代理类, 否则仍然执行业务逻辑类.

切面类包括数据安全访问策略、日志管理等, 和访问权限判断共同构成了数据库安全访问的控制.

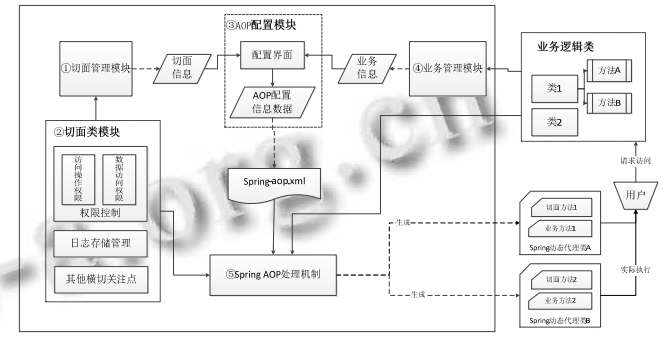

图 3 系统架构图

根据系统流程图, 图 3 所示为系统架构图. 本系统主要由 5 个部分组成:

①切面管理模块: 负责切面信息管理, 管理系统中的切面类模块, 将其文件名、方法参数持久化到数据库, 供配置模块配置时使用.

②切面类模块: 系统通用模块, 通过模块注入的方式应用于系统各个模块. 是 AOP 配置部分中切面机制设置的组成部分. 在应用系统启动时, 切面类根据 Spring-aop.xml 的信息将代码和业务逻辑类代码组合生成动态代理类. 本系统以安全访问相关的日志管

© 中国科学院软件研究所 http://www.c-s-a.org.cn

理、权限控制为例作为系统实现.

③AOP 配置模块: 包括系统表示层的可视化配置, 业务逻辑层的自动化生成 XML 配置文件等内容. 主要完成 AOP 配置流程的部分.

④业务管理模块: 负责业务信息管理, 管理各个系统本身的业务逻辑部分, 将其文件名、方法参数持久化到数据库, 供开发人员配置 AOP 信息.

⑤Spring AOP 处理机制: 根据 Spring-aop.xml 文件和切面类模块、业务逻辑类的输入信息处理生成 Spring 动态代理类, 供用户在访问业务逻辑类时实际调用执行.

4 系统模块实现

4.1 切面管理模块

切面管理模块主要负责持久化切面类模块的类名、方法名和参数信息, 生成切面信息数据. 这些数据将供 AOP 配置模块配置时使用. 目前为止, 系统仅支持权限管理、日志管理等涉及数据库安全访问控制的切面类模块调用. 从可扩展性角度上讲, 还有许多可以作为切面的性能模块没有被引入到系统中来. 这些模块并非适用于每个应用系统, 因此切面管理模块会根据不同的应用系统进行管理, 持久化需要的切面类模块即可.

4.2 切面类模块

作为系统中的通用模块, 我们以数据库的安全访问切面类模块作为样例. 其实任何可以从业务逻辑中提取出来的模块都可以作为横向关注点的切面类模块, 而不仅仅是传统意义上局限的日志管理、权限控制、事务处理等等.

安全访问控制模块中包含日志管理、权限控制管理、异常处理、性能监控等多项涉及数据库安全的部分. 本系统以日志存储管理和权限管理为例实现[7].

4.2.1 日志存储管理

与一般意义上的日志管理不同, 这里说的日志是面向最终用户设计的访问、操作记录日志. 因此并不是简简单单记录下系统内部的函数调用和数据传输过程, 而要详细的记录下用户登录后进行的一系列操作和动作, 这些操作和动作背后实际上就是已经注入了日志管理的业务逻辑模块.

系统的日志管理模块 LogService.java 类中有 addLog 和 deleteLog 两个方法, 系统的日志数据表包括日志编号、日志等级、日志类型、操作人、所属单位、IP 地址、操作细节(XX 单位的 XX 在什么时间做了什么操作)、操作时间. 其中除日志编号是数据库自动加载以外, 其余项均需要业务逻辑获取或生成. 当 addLog 方法执行时, 程序会获取当前登录的用户信息, 根据日志数据表的设计完成一次记录操作.

4.2.2 权限控制

数据库的安全访问控制重点就在于权限的控制, 它包括功能操作级的权限和数据访问行列级的权限. 其中功能操作级权限主要指用户是否具有访问某个界面和操作某项数据的权利; 数据访问行列级权限则指对于某一项具体的数据, 用户是否具有访问的权限, 具有哪些数据、哪几项数据的访问权限.

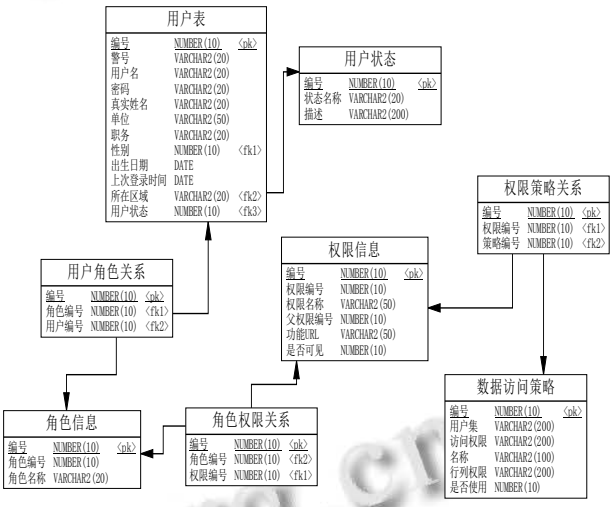

图 4　权限管理模块

如图 4 所示为权限控制模块的数据库设计表. 图中包括用户表、角色信息表、权限信息表、数据访问策略表等主表, 和用户状态表、用户角色关系表、角色权限关系表和权限策略关系表等附属表或关系表.

用户表用来记录系统注册的用户基本信息. 角色信息表记录系统中可能的不同角色, 如管理员、高级用户、普通用户等. 权限信息表记录访问某个功能模块所需要的最低权限. 通过用户状态表、用户角色关系和角色权限关系三个中间表便可构成完整的功能操作级权限判断. 通过 AOP 机制注入到业务逻辑组件中后, 一旦登录用户访问某个功能界面而调用了组件中的函数, 权限控制模块就会根据所访问的 URL 权限信息和用户角色的权限进行判断来决定用户是否成功访问或操作.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

对于数据访问行列级权限同样可以通过 AOP 的方式注入到业务逻辑当中. 数据访问策略表记录了行列级数据访问的权限, 和权限信息表一同完成数据访问行列级权限的控制. 当登录用户访问数据列表时, 系统会根据用户的身份和权限以及预先设定的行列访问权限判断数据列表中哪些数据项可以被访问.

权限控制渗透在系统关键业务模块当中, 利用 AOP 的方式配置实现动态的权限配给和收回, 而不再需要开发人员每次更改大量的代码来完成. 用户、角色和权限之间通过关联表连接后, 系统读取登录用户编号就可以根据编号得知该用户的权限和角色, 也基于此对用户的访问权限进行判断, 从而决定访问成功与否, 完成数据库访问安全控制中的用户访问权限控制.

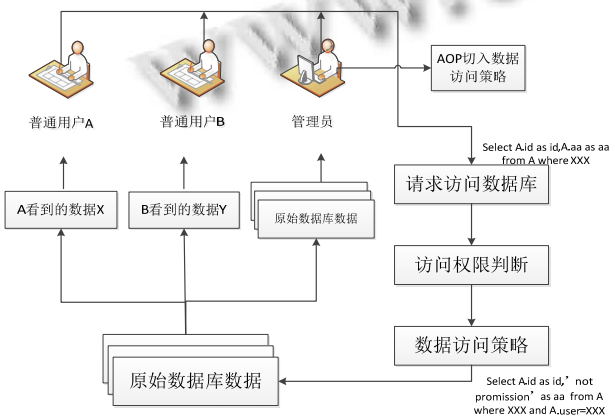

图 5 数据安全控制

图 5 所示为通过 AOP 切入机制设置数据访问行列级权限的数据库访问示意图. 系统管理员对角色权限和数据访问策略进行设定. 当用户发起一个数据库数据请求时, 根据用户的角色和权限进行判断, 在系统的 SQL 请求语句后添加角色和权限信息如"where data.create_user=_$User{User.getId}" 或者 "where data.create_user.location='XXX'", 筛选出只有自己创建的信息或者同一地区用户创建的信息. 数据经过初次筛选后匹配数据访问策略, 同样基于 SQL 注入的方式筛选出部分可见字段和数据, 最终展现给不同用户不同的数据, 如图中普通用户 A 只能看到 X 部分数据, 而普通用户 B 只能看到 Y 部分数据, 从而完成数据库安全访问的控制.

4.2.3 其它扩展模块

安全访问控制不仅仅是日志存储管理和权限管理, 还包括性能监控、攻击防范等一些列内容. 同样, 作为横向关注点的切面类模块不仅仅限于安全访问控制. 如前文所述, 在缓存、调试、懒加载、内容传递、异常处理、事务等情境下均可以作为横向关注点完善切面类模块. 在这种思想的引导下, 系统的可复用内容也会越来越广泛.

4.3 AOP 配置模块

4.3.1 AOP 的可视化配置

在安全访问控制系统中, 通过可视化的界面进行 AOP 配置是系统关键的模块之一.

如图 6 所示为 AOP 配置模块的数据库设计图, 通过持久化的存储自动生成 XML 配置文件信息, 从而替代手动编写的麻烦. 图中共包含 5 张数据库表, 通知表记录每一项配置的具体信息, 包括切入方式、切入位置和切入内容等主要信息和添加人等辅助信息; ASPECT 切面表则记录下一个具体的切面信息, 它和 BEAN 表一同完整记录了切面的信息; POINTCUT 切入点表和 POINTTYPE 切入类型表则反映了在什么业务逻辑模块中切入所需的内容, 记录了包括切入的方式和位置等内容. 有了这些数据, 系统就可以根据数据库中的数据进行动态生成配置文件的过程了.

图 6 AOP 配置模块

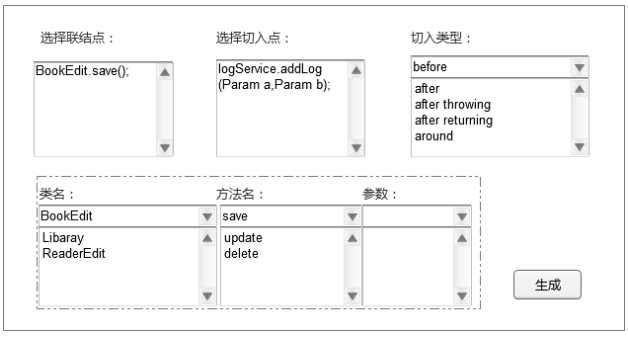

图 7 AOP 配置界面示意图

图 7 所示的 AOP 配置界面示意图显示了开发人员需要在配置时需要填入的信息, 包括选取切入点

© 中国科学院软件研究所 http://www.c-s-a.org.cn

(pointcut)、联结点(BEAN 表和通知表中的“切面方法”)和切入类型(pointType). 在用户填写好这些信息后, 后台逻辑会根据参数的有无和切入类型将 pointcut 信息拆分成方法名和参数名, 并自动添加上相应的附属信息.

4.3.2 自动生成 XML 的配置处理部分

在完成安全访问控制的 AOP 配置之后, 开发人员或领域专家可以根据配置好的信息自行设定想要执行的部分, 对于那些暂时不需要或者已经无效的配置信息可以选择不生成配置信息或者删除. 这些灵活的操作不需要开发者重新编写代码, 极大降低了系统的维护成本.

4.4 业务管理模块

业务管理模块主要负责持久化业务逻辑类的类名、方法名和参数信息生成应用信息数据. 这些数据记录了业务逻辑类中相应的信息, 将供 AOP 配置模块配置时使用. 配置过程中, 将业务信息中业务逻辑类的参数信息和切面信息相结合, 从而将业务信息的参数传递给切面.

4.5 Spring AOP 处理模块

Spring AOP 使用动态代理的机制实行横向关注点的切入. 在系统运行时, Spring 会根据 Spring-aop.xml 文件读取相应的切面类、业务逻辑类, 将其组合成各自的 Spring 动态代理类. 当用户访问某个具有切面配置的业务逻辑类时, 实际上系统执行的是已经切入了切面代码的代理类, 从而完成切面的织入, 但这一切对用户而言是不透明的.

5 总结与展望

数据库应用的安全访问关系到整个系统的安全. 作为系统通用的功能, 安全访问控制模块会被业务逻辑中的各个模块调用, 这种客观存在的业务需求使得开发人员必须通过有效的方式解决. 无论是传统的函数调用方式还是通过 AOP 的配置方式, 都需要开发人员进行大量的代码编写工作. 而这种编写很多都是冗余的工作, 降低了开发效率. 通过可视化的 AOP 配置, 将安全访问控制模块灵活、可定制、简便的注入到各个模块中. 同时, 通过可视化的系统, 开发人员和领域专家可以共同制定系统的安全访问控制机制, 并且随着时间的推移随时改变已有的安全机制, 从而使得安全访问控制更加灵活、简洁和多变.

在目前的系统中还存在不足, 例如切面类模块的设计还较为简单, 其它可以作为横向关注点的模块还没有完善到系统中来, 在未来的设计和实现工作中, 这将是努力完善的方向.

参考文献

1 高海洋,陈平.AOP 综述.计算机科学,2002,29(10):133–135.
2 张国平,万仲保,刘高原.Spring AOP 框架在 J2EE 中的应用.微计算机信息,2007,23(36):254–256.
3 古全友,王恩波,胥昌胜.AOP 技术在 J2EE 系统构建中的应用.计算机技术与发展,2006,16(4):150–152.
4 Nusayr A, Cook J. Using AOP for detailed runtime monitoring instrumentation. Woda the Sixth International, 2009.
5 Kim DK, Bohner S. Dynamic reconfiguration for Java applications using AOP. Southeastcon, IEEE. IEEE, 2008: 210–215.
6 张献,董威,齐治昌.基于 AOP 的运行时验证中的冲突检测.软件学报,2011,22(6):1224–1235.
7 唐建.农产品供应链管理平台的应用安全设计与实现[学位论文].中国科学院大学,2013.

© 中国科学院软件研究所 http://www.c-s-a.org.cn