文章编号: 1000-5013(2009)06-0642-04

构建大型货代系统的解决方案

秦 波, 余金山

(华侨大学 计算机科学与技术学院, 福建 泉州 362021)

摘要: 在构建大型货运代理系统时,针对所遇到的分公司间存在需求上的差异、权限和并发访问等关键问题,提出一种有效的,可行的解决方案.引入基于Java的规则引擎,解决各分公司需求差异的技术实现问题,把大量的if-else语句从应用程序中分离出来,简化了代码,并增强可读性和可维护性. 通过在服务器和客户端两方面的构建,系统可以很好地分配功能访问权限,既保证分公司间角色的多样性,又保证数据的安全性.选用Read Committed隔离级别,并采用对象持久化技术,可解决不可重复读和第二类丢失更新的并发问题.

关键词: 货代系统; 规则引擎; 对象持久化技术; 参数配置
中图分类号: TP 393.07; F 252 文献标识码: A

目前,国内绝大部分的物流系统还只是基于SQL数据库和NT操作系统平台的,对J2EE和.NET的支持不够全面.当物流系统的规模和用户数较小时,系统尚能应付,但当企业的业务发展达到一定的规模,对系统的应用需求达到一定的程度时,这种局限性马上就会暴露出来.大量的业务需求无法满足,甚至会出现系统崩溃[1].造成这种状况的主要原因就在于,大型的货代企业大都是总公司、分公司和经营部相结合的一个组织结构,如果在这样一个组织结构中共享同一套系统,会给其设计和实现带来极大的挑战.即使不同的分公司使用各自独立的系统,也要涉及到系统整合的问题,主要包括分公司间存在需求上的差异问题、权限问题和并发访问问题.基于此,本文提出一种有效、可行的解决方案.

1 利用规则引擎解决分公司需求差异

规则引擎的基本机制:对提交给引擎的数据对象进行检索,根据这些对象的当前属性值和它们之间的关系,从加载到引擎的规则集中发现符合条件的规则,并创建执行实例.这些实例将在引擎接到执行指令时,依照某种优先序依次执行.一般地,规则引擎内部由如下3个部分构成.(1) 工作内存. 用于存放被引擎引用的数据对象集合.(2) 规则执行队列.用于存放被激活的规则执行实例.(3) 静态规则区.用于存放所有被加载的业务规则,而这些规则将按照某种数据结构进行组织. 当工作区中的数据发生改变后,引擎需要迅速根据工作区中的对象现状,调整规则执行队列中的规则执行实例[2].

在货代系统的应用程序中,对于分公司间存在操作差异的方法体加一个拦截器,然后把前台传过来的数据和参数截获;接着,调用相应的规则文件,选取对应的规则进行判断.如果通过,则程序继续往下执行;否则,前台弹出对话框进行说明,同时阻止后台程序往下执行.规则文件的设计是按照分公司的功能需求来划分的,一个分公司独立拥有一个规则文件. 这样做最大的好处就是能够为各分公司定制不同的规则,并且修改一个规则文件不会影响到其他的分公司.如果新增一个分公司也很容易进行扩展,因为只需要增加一个新的规则文件就可以了.

规则引擎的算法有很多种,其中Rete算法是唯一的效率与执行规则数目无关的决策支持算法,是目前用于生产系统的效率最高的算法.其核心思想是,将分离的匹配项根据内容动态构造匹配树,以达到显著降低计算量的效果.研究和实践表明,使用基于Rete算法的规则引擎——Drools,可以在大型货

收稿日期: 2008-08-15

通信作者: 余金山(1952-),男,教授,主要从事软件工程及人工智能应用的研究. E-mail: yjs@hqu.edu.cn.

基金项目: 福建省自然科学基金资助项目(A0810013)

代系统中有效地解决分公司间需求差异的问题.

Drools 分为构建(Authoring) 和运行时(Runtime) 两个主要部分. 构建的过程涉及到规则文件(xml 文件或 drl 文件) 的创建, 它们被读入一个解析器, 使用 ANTLR3 语法进行解析. 解析器对语法进行正确性的检查, 产生一种中间结构" descr", descr 用 AST 来描述规则. AST 被传到 Package Builder, 由 Package Builder 来产生 Package 对象. Package Builder 还承担着一些代码产生和编译的工作, 这些对于产生 Package 对象都是必需的. Package 对象是一个可以配置的, 可序列化的, 由一个或多个规则组成的对象.

Rule Base 是一个运行时组件, 它包含了一个或多个 Package 对象, 并可以将一个 Package 对象加入或移出 RuleBase 对象. 一个 RuleBase 对象可以在任意时刻实例化一个或多个 Working Memory 对象, 在它的内部保持对这些 Working Memory 的弱引用. Working Memory 由一系列子组件组成. 当应用程序中的对象被 assert 进 Working Memory, 可能会导致一个或多个 Activation 的产生, 然后, 由 Agenda 负责安排这些 Activation 的执行[3].

由以上的设计方案可以看出, 规则引擎能够有效地解决分公司间的操作差异, 把大量的 if-else 语句从应用程序中分离出来, 简化了代码, 增强可读性和可维护性. 同时, 由于每个分公司有自己的规则文件, 进一步解耦合, 可增强系统的可扩展性.

2　利用配置分公司参数解决权限问题

权限是对系统数据访问和功能使用的许可, 分为功能访问权限和数据使用权限. 数据访问权限控制比较简单, 一般只需在查询时加一个过滤条件即可; 而功能访问权限的分配将引入分公司参数的概念. 一个分公司可分配多种角色, 同时具有该角色的权限模型, 可以操作该权限模型下的系统功能和业务数据. 权限管理涉及的角色多, 相互关系复杂[4].

(1) 服务器端. 服务器端设计的总体思路可以归结为如下 4 个概念的定义.

(a) Security Privilege. 对受保护资源的操作定义. 可以只针对功能(如菜单选项, 按钮等) , 也可以和 Variable 结合控制对资源的访问.

(b) Security Variable. 对 Privilege 具体化. 定义其具体的控制范围 Variable Area, 同时指明所针对的 Object (如 Office).

(c) Security Session. 用户登录后, 创建当前用户权限的一个实例, 并提供权限判断的接口. 定义用户的权限包括 Privilege 和 Variable 两个方面. 其中, Privilege 定义了用户所有的操作权限, Variable 定义了操作权限的适用范畴及控制范围.

(2) 客户端. 用户登录成功后, 从后台获取 User Session 信息, 从中读取 Operator(包括 Privilege 和 Variable) 的信息; 然后, 根据 Privilege 来判断该用户的相关功能权限.

通过在服务器和客户端两方面的构建, 系统可以很好地分配功能访问权限, 既保证了分公司间角色的多样性, 又保证了数据的安全性.

3　利用对象持久化技术解决并发控制问题

大型货代系统的并发访问是一个很普遍的问题, 这对数据库系统的并发性能有比较高的要求. 对于同时运行的多个事务, 当这些事务访问数据库中相同的数据时, 如果没有采取必要的隔离机制, 就会导致各种并发问题. 这些并发问题可归纳为以下 5 类.

(1) 第一类丢失更新. 撤销一个事务时, 把其他事务已提交的更新数据覆盖. (2) 第二类丢失更新. 这是不可重复读中的特例, 一个事务覆盖另一事务已提交的更新数据. (3) 脏读. 一个事务读到另一个事务未提交的更新数据. (4) 虚读. 一个事务读到另一个事务已提交的新插入的数据. (5) 不可重复读. 一个事务读到另一事务已提交的更新数据.

数据库系统采用锁来实现事务的隔离性, 以此来达成对并发事务的控制. 数据库系统允许用户在事务中显式地为数据资源加锁, 但是首先应该考虑让数据库系统自动管理锁, 因为它能自动为 SQL 语句

所操作的数据资源加上合适的锁,以提高系统性能.锁机制能有效地解决各种并发问题,但是它会影响并发性能.当一个事务锁定数据资源时,其他事务必须停下来等待,这就降低了数据库系统同时响应各种客户程序的速度.因此,数据库系统提供了由高到低的4种事务隔离级别供用户选择.隔离级别越高,越能保证数据的完整性和一致性,但是对并发性能的影响也越大.表1列出了各种隔离级别所能避免的并发问题.

表1 各种隔离级别所能避免的并发问题

Tab.1 Concurrent problems avoiding on the various shielding level

隔离级别	第一类丢失更新	脏读	虚读	不可重复读	第二类丢失更新
Serializable	否	否	否	否	否
Repeatable Read	否	否	是	否	否
Read Committed	否	否	是	是	是
Read Uncommitted	否	是	是	是	是

由于大型货代系统中并发访问的普遍性,数据库系统不能选用太高的隔离级别.研究和实际经验表明,选用Read Committed隔离级别最为合适.那么,对于选用此级别而导致的不可重复读和第二类丢失更新的并发问题,采用对象持久化技术予以解决.

所谓的对象持久化,就是使用一些持久的存储介质来保存对象的运行状态信息.譬如,利用Hibernate,能够建立面向对象的域模型和关系数据模型之间的映射.概念模型用来模拟问题域中的真实实体.关系数据模型是在概念模型的基础上建立起来的,用于描述这些关系数据的静态结构.域模型描述了具有状态和行为的域对象及域对象之间的关系.

这里,Hibernate是连接Java应用程序和关系数据库的中间件.在分层的软件架构中,逻辑上,它位于持久化层,封装了所有的数据访问细节,使业务规则层可以专注于实现业务逻辑;物理上,它位于业务规则层,因为它并不需要独立的服务器[5].

利用Hibernate的乐观锁机制,可以轻易地解决不可重复读和第二类丢失更新所产生的并发问题.从应用程序的角度讲,锁可以分为悲观锁和乐观锁.悲观锁指在应用程序中显式地为数据资源加锁.它假定当前事务操纵数据资源时,肯定还会有其他事务同时访问该数据资源,为了避免当前事务的操作受到干扰,先锁定资源.乐观锁则相反,它假定当前事务操纵资源时,不会有其他事务同时访问该数据资源,因此完全依靠数据库的隔离级别来自动管理锁的工作.

从这两种锁的实现机制来看,虽然悲观锁能够比较有效地防止出现并发问题,但是它会影响到并发性能;而乐观锁虽然能有效地保证并发性能,但同时可能会产生以下并发问题,需要采用一些控制手段.在应用程序中,可以利用Hibernate提供的版本控制功能来实现乐观锁.对象-关系映像文件中的〈version〉元素和〈timestamp〉元素都具有版本控制功能.〈version〉元素利用一个递增的整数来跟踪数据库表中记录的版本,而〈timestamp〉元素用时间戳来跟踪数据库表中记录的版本,并使用〈version〉元素的版本控制方法.映象文件如下:

```
〈hibernate-mapping package="example.entity" auto-import="true"〉
  〈class name="User" table="USER" optimistic="version"〉
    〈id name="user_id"
      〈column name="USER_ID" not-null="true" length="8"/〉
      〈generator class="increment"/〉
    〈/id〉
    〈version name="version" column="VERSION" type="Integer"/〉
    〈property name="user_name〉
      〈column name="USERNAME" length="20"/〉
    〈/property〉
    〈property name="password"〉
      〈column name="PASSWORD" length="100"/〉
```

```
    〈/property〉
  〈/class〉
〈/hibernate-mapping〉
```

首先, 在 Java 类里添加一个属性 version, 整数型(Integer 型) . 接着, 在该类对应的数据库表中, 添加一个字段 VERSION, 整数型(Integer 型) . 最后, 映射文件中, 添加一个〈version〉元素, 注意必须紧跟在〈id〉元素的后面, 同时在〈class〉元素中加入一个属性: optimistic-lock= “version”. 这样, 当加载该类的一个对象时, 它的 version 属性表示数据库表中相关记录的版本. 当用户更新该对象时, 会根据它的 id 与 version 属性到数据库表中去定位匹配的记录.

假定 A 和 B 两个操作员同时对同一数据资源 Data1(假定 Data1 的 version 值为 0) 进行操作, 操作员 A 首先更新 Data1 一次, 此时 Data1 的 version 值变成 0+ 1= 1; 接着, 如果操作员 B 也尝试去更新 Data1, 将会失败. 这是由于操作员 B 手上的 Data1 的 version 值仍然为原来的 0, 而数据库中 Data1 的 version 值已经变成 1, 所以操作员 B 试图更新的时候, 系统会提示找不到匹配的资源, 更新失败并抛出异常. 这就很好地解决了并发访问控制的问题.

4　结束语

给出的方案解决了各分公司需求差异的技术实现问题, 但终究没有实现业务流程与程序代码的完全剥离, 业务流程和代码的耦合度还比较高. 因此, 可以考虑引入工作流的思想, 以进行流程和代码的解耦合, 使系统能够具有较好的灵活性与扩充性.

参考文献:

[1] 王淑云. 现代物流[J]. 北京: 人民交通出版社, 2002.

[2] ERNEST F H. Jess in action: Rule-based system in Java[M]. Greenwich: Manning Publications, 2003.

[3] FU G, SHAO J, EMBURY S M, et al. A framwork for business rule presentation[C] ∥ Proceedings 12th International Workshop on Database and Expert Systems Applications Washington D C: IEEE Computer Society, 2001: 922-926

[4] DATTA A, SANG H S. A study of concurrency control in real-time acitive database systems[J]. IEEE Trans Knowledge and Engineering, 2002, 14(3): 465-484.

[5] 沈　锐. 基于 J2EE 物流系统持久层的 Hibernate 解决方案[J]. 电脑知识与技术, 2005(3): 13-15.

Solution for Constructing Large-Sized Freight Agent Systems

QIN Bo, YU Jin-shan

(College of Computer Science and Technology, Huaqiao University, Quanzhou 362021, China)

Abstract: This paper analyses the key questions on constructing large-sized freight agent systems, which are the different demands among branch companies, concurrency-control problems and authorization control problems, and then an efficient, feasible resolution is brought forward. Using rule engine based on java, the technology is realized for demand difference from branch companies, and many if-else statements are separated from application program, so codes are simplified, and readability and maintainability is enhanced. The system can assign the access privileges by the construction on the server side and client side which can ensure the diversity of roles among the branches and data safety. Choosing isolation level of read committed and adopting the object persistent technology can settle the concurrent problems between non-repeatable read and second losing the refreshment

Keywords: freight agent systems; rule engine; object persistent technology; parameter configuration

(责任编辑: 鲁　斌　　英文审校: 吴逢铁)