基于领域驱动的MIS系统细粒度权限模型研究

张忠捷 喻 昕 王高才
(广西大学计算机与电子信息学院 南宁 530004)

摘 要 权限管理是管理信息系统(MIS)中非常重要的组成部分,现有部分系统中的权限管理过于粗糙或混乱,无法实现较为完整细致的权限管理;而其它的权限管理方案理论虽好,但与当今主流的软件开发方式和框架不相符,且不具备可实现性,无法应用于实际开发的信息系统。为此本文基于领域驱动设计的思想,对传统的基于角色访问控制(RBAC)模型进行细化和改进,在此基础上构建一个基于领域驱动的MIS系统细粒度的权限模型,然后再实现该模型,以展示权限系统构建的全过程,从而给目前权限系统的开发提供一个较完善的解决方案。本文给出了权限实现的详细方法和技术。

关键词 RBAC,权限管理,细粒度,领域驱动,MIS
中图法分类号 TP31 **文献标识码** A

Study on Finer-Grained Privilege Model for MIS Based on Domain-driven

ZHANG Zhong-jie YU Xin WANG Gao-cai
(School of Computer and Electronic Information, Guangxi University, Nanning 530004, China)

Abstract Privilege management is an very important part for management information system(MIS). Nowadays, the existing privilege management methods are coarse or confused, and they can't implement integrated and elaborate privilege management. On the other hand, others privilege management schemes maybe good, but they can't match current mainstream software development models and frameworks, and can't provide better exercisablility. Therefore, this paper improved and refined traditional RBAC model based on domain-driven to construct and implement a finer-grained privilege model for MIS. The paper also showed the whole privilege management process for constructing, and provided a whole solvable project for developing privilege management.

Keywords RBAC, Privilege management, Finer-grained, Domain-driven, MIS

1 引言

权限管理指根据系统设置的安全规则或者安全策略,用户可以访问而且只能访问自己被授权的资源。权限管理几乎出现在任何有用户和密码的系统中。在管理信息系统(MIS系统)中,权限的管理是必不可少且非常重要的一个组成部分。当前的MIS系统中的权限管理模块往往存在一些问题,如:(1)系统定位不合理,或与业务结合得过于紧密,或者过于独立;(2)权限的粒度过于粗糙,仅定位在大的功能模块一级,使得系统管理员无法在该系统上实现用户权限细粒度的调整,往往需要程序员修改代码实现;(3)用户权限分配不合理,在用户量多且关系复杂时,对用户进行增、删、改等操作时往往造成权限分配的混乱;(4)权限设置范围不合理,往往只涉及系统的一些重要功能模块,不能涵盖系统的各个方面,从而给系统管理员在需要给用户分配权限时带来困难;(5)权限的开发不合理,往往需要花费大量的时间和精力却无法达到满意的效果。

针对以上问题,本文在目前MIS系统中最常采用和最为流行的权限控制方法——基于角色的访问控制模型(RBAC)的基础上,采用领域驱动设计的方法对其进行扩展和细化,以此构建一个细粒度的权限模型,并实现之,从而构建一个高效的权限系统,作为MIS系统中构建权限模块的一个通用解决方案。

2 权限系统的构建理论

2.1 RBAC模型原理

基于角色的访问控制(RBAC)是目前公认的解决大型企业的统一资源访问控制的有效方法[1]。RBAC模型由4个部件模型组成,分别是基本模型RBAC0、角色分级模型RBAC1、角色限制模型RBAC2和统一模型RBAC3。RBAC0定义了能构成一个RBAC控制系统的最小元素集合,包含用户、角色、对象、操作、许可权5个基本元素。基于RBAC0模型,先后派生出如下模型:

(1)RBAC1:即角色分层模型,它引入了角色间的继承关系;

(2)RBAC2:即角色约束模型,它添加了约束,即规定了权限被赋予角色时,或角色被赋予用户时,以及当用户在某一时刻激活一个角色时所应遵循的强制性规则;

本文受国家自然科学基金项目(61063045)资助。
张忠捷(1982—),男,硕士生,主要研究方向为软件工程;喻 昕(1973—),男,副教授,主要研究方向为软件项目开发、项目管理,E-mail:yuxin21@126.com(通信作者);王高才(1976—),男,博士,教授,博士生导师,主要研究方向为计算机网络技术及系统性能评价。

(3)RBAC3:即统一模型,它包含 RBAC1 和 RBAC2,既提供了角色间的继承关系,又提供了约束。

RBAC 在用户和访问权限之间引入角色的概念[2],并且用户与特定的一个或多个角色相关联,角色同一个或多个访问权限相关联。在 RBAC 模型中,通过分配和取消角色来完成用户权限的授予和取消,实现了用户与访问权限的逻辑分离,这样极大地简化了权限管理。考虑到 MIS 系统的复杂性,本文的权限模型将基于 RBAC3 统一模型[3],如图 1 所示。

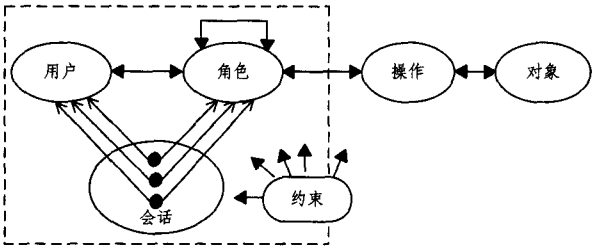

图 1 RBAC3 统一模型

2.2 基于领域驱动的权限系统构建方法

目前,绝大多数开发人员都把权限系统当作相应业务系统的一个模块,开发的思路和方向也就局限在该系统之内,这样,即使采用了 RBAC 模型,也未能实现权限和业务的相对独立。为了构建较为全面严谨的权限系统,从而能够灵活应对变化,本文将使用领域驱动设计的思想,从另一个较新的角度来分析解决此问题。

领域驱动设计是一种全新的软件开发方法[4],目的是在实现软件系统时能准确地基于对真实业务过程的建模并根据真实业务过程的调整进行相应调整。它把软件系统当作业务过程的一个影射,是使能动而不是驱动。领域驱动设计应深入到业务过程中,了解业务术语和实践方法。领域驱动设计认为专注于框架技术而忽略具体业务是错误的软件开发方法,对于复杂的业务领域必须分清重点和次要部分,抓住核心领域概念,实现重点突破[5,6]。领域驱动设计的前提是:(1)对于多数软件项目,主要的焦点应该在领域及领域逻辑方面;(2)复杂的领域设计应该基于一个模型来进行。

按照领域驱动设计的理论,如果说一般的 MIS 系统是以实际中的业务为研究对象,从中抽象出业务领域模型,那么权限系统则是以构建完成的 MIS 系统为研究对象,从中抽象出权限领域模型。在权限系统中,MIS 整个系统即是它所关注的"业务",MIS 系统里的类、属性、方法、函数等各种元素就构成了它的需求领域,当软件开发人员对这些元素进行增、删、改时,需求领域就会发生相应的变化;而对这些元素进行设置、判断、获取、删除权限,就是权限系统的功能需求。即权限系统不应看作对应的 MIS 系统的一部分,而应该从独立系统的角度上来分析,将其看作一种特殊的 MIS 系统,一个建构在对应的 MIS 系统之上的 MIS 系统,它们之间的关系如图 2 所示。

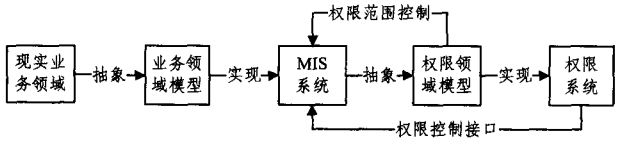

图 2 权限系统的实现流程

显然,相对传统从业务系统模块角度开发的模式,领域驱动下从独立系统角度开发的模式更有优势,也更能解决传统模式下的各种弊端,具体有:

1)保证权限系统和 MIS 系统的松散耦合。一方面对于权限系统的需求分析,权限领域模型设计不能脱离 MIS 系统,权限范围控制和权限控制接口也要作用在 MIS 系统上;另一方面权限系统又可以根据权限领域模型独立编程实现,而不受原 MIS 系统的约束,从而较好地实现了权限-业务的相互独立和相互关联。

2)明确权限系统的权限设定范围。设定 MIS 系统中所有元素为权限领域模型研究对象并通过模型来控制权限范围的方法不仅使得权限设定的对象更明确,也使得权限设定的粒度可控性更强,根据实际需要可以任意选取大至系统里的一个项目或一个模块,小至某个类中的一个属性或一个方法等元素来设定权限,从而避免旧模式权限可控范围小、粒度粗糙等不足之处。

3)扩展空间更强大。独立权限系统使得程序员可以根据用户需求对与权限相关的问题做任意的增删改操作,也可创建复杂的权限控制流程和权限控制接口,而不会影响到原 MIS 系统,其具有很强的自主性和扩展性。

故此,以下本文将根据图 2 的流程,展示构建权限模型并实现权限系统的全过程,从而彻底解决权限系统中的常见问题。

3 MIS 系统分析

要开发 MIS 的权限系统,则必须先构建 MIS 业务信息系统。为便于分析,假定实现一个简单的客户-订单系统,其主要系统逻辑用类图表示,如图 3 所示。

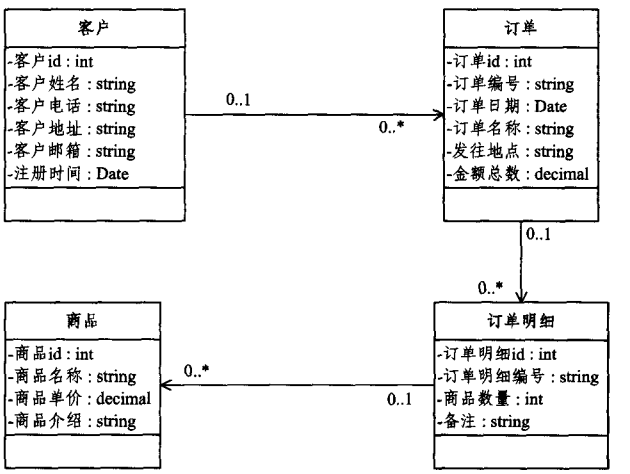

图 3 客户-订单系统业务逻辑图

一般来说,对于客户-订单系统的系统架构,可以有两种不同的实现方式:以数据库为核心和以领域层为核心。其中前者适用于数据库模式的开发思想,后者适用于领域驱动模式的开发思想,为论述方便,在此取后者作为例子。根据领域驱动设计理论,其分层架构为:用户界面层、应用层、领域层、基础设施层,如图 4 所示。

各层的功能如下:用户界面层负责与用户交互;应用层不涉及具体业务规则,它只是调用领域层,应用层的元素实现工作流程,起到协调任务、委托工作的作用;领域层是整个系统的核心,它包括所有业务数据和业务规则,业务领域在这一层中得到集中体现;基础设施层为以上各层功能的实现提供技

术支持，一切与技术有关、与业务无关的元素都可以放在这一层，比如发送 e-mail，绘制图像等。MIS 系统常用的、包含各种操纵数据库操作的数据访问层也属于基础设施层，从而 MIS 系统的存储工具数据库也自然位于基础设施层之下，由其直接调用。

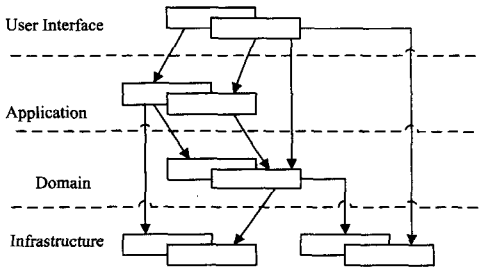

图 4　领域驱动设计标准分层架构图

采用这种架构的客户-订单系统，需要把图 3 中的类图放置在领域层，同时该层增加各种类、属性、方法等元素来实现下订单、发订单等系统的各种业务流程和业务规则，其它各层则围绕领域层提供各种支持来实现系统功能。在此设定该系统已采用. NET 平台下技术实现。

4　基于领域驱动的 MIS 权限系统领域模型

4.1　模型框架以及分析方法

根据图 2 流程，需求领域－客户－订单业务信息系统已成功实现之后，下一步就是构建权限领域模型。权限领域模型是实现权限功能最核心的部分，是实现权限系统的基础，以下将按顺序论述模型从构建、实现到发挥作用的全过程。首先是确定模型的整体框架，而这仍然是遵循传统的 RBAC3 模型逻辑，由用户模型、角色模型、会话模型、约束模型、对象模型和操作模型(对象模型和操作模型结合起来又可以称为权限模型)这 6 大模型相互关联而成。显然，RBAC3 模型是从整体上定位权限系统的领域逻辑，此时的领域模型仅具备整体架构，却无法编程实现。因此，需要进一步分析客户-订单系统里的类、属性、方法、函数等各种元素，从中抽象出 6 大模型的数据结构与关系，确定各模型包含的具体元素，才能构建出一个各模型可持久化紧密关联并可实现的权限领域模型(见图 5)。

由于 UML 图是构建系统模型最常用的手段，因此权限领域模型可采用 UML 类图来表示，以下即从类图角度分析各模型的具体构建。

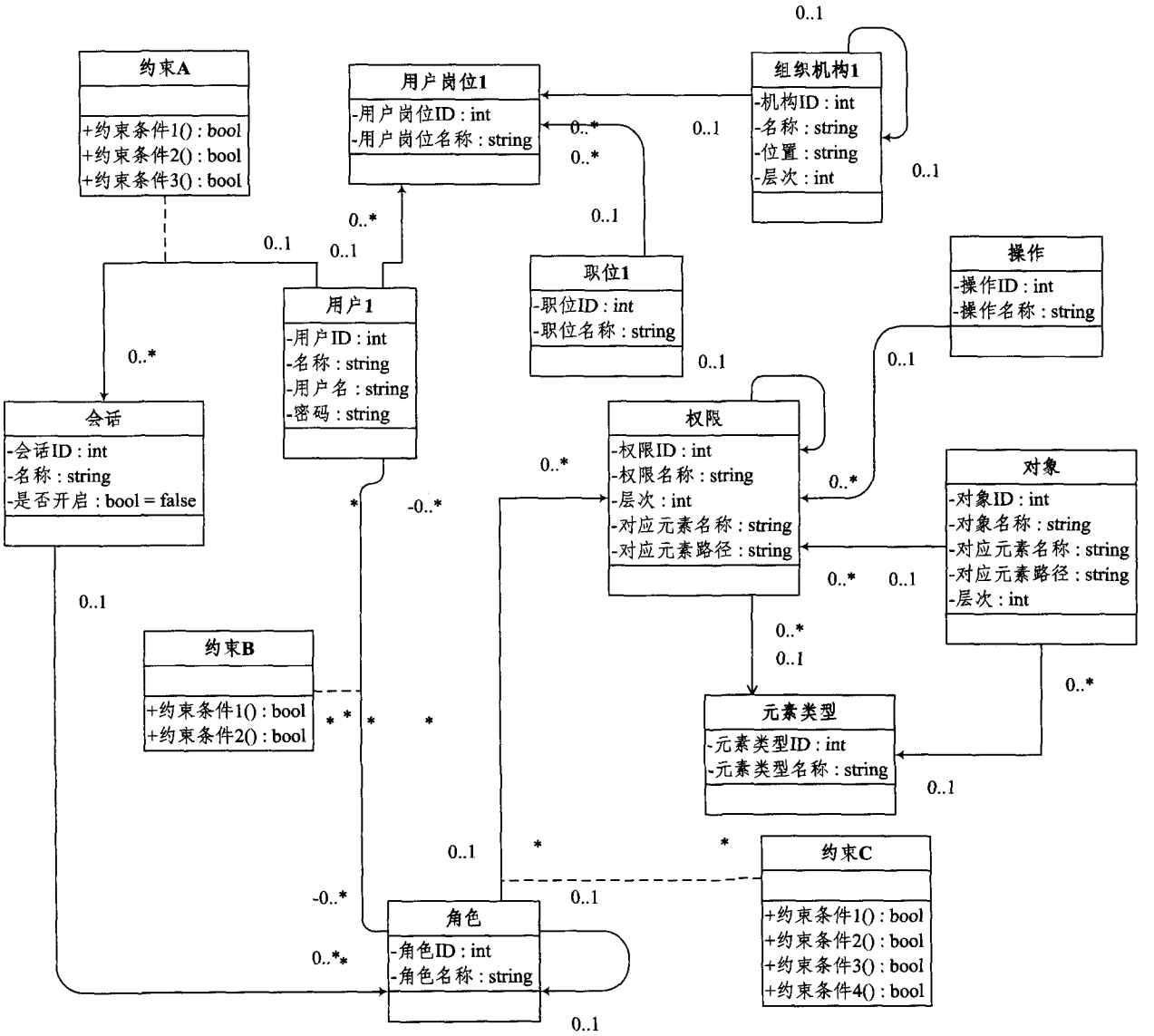

图 5　权限系统领域模型

4.2 模型分析

1. 用户模型

实际上对于用户模型来说，它既属于客户-订单业务系统，也属于相应的权限系统，它在两者之间的数据结构和具体元素都是一致的，因此权限系统的用户模型直接采用该系统的用户部分模型即可。在实际中，用户往往和用户所在的组织机构以及机构中的职位相关联，因此在权限系统中构建用户模型时，也同样必须把组织机构和职位设置考虑在内。用户、组织机构、职位三者的任意两者间都是多对多的关系，于是可增加用户岗位类实现关联。

2. 角色模型

角色模型和用户模型之间是多对一关系。角色模型是为提高权限管理效率而设，与业务系统元素并不存在关联对应关系，因此其数据结构只需要设定“角色 id”和“角色名称”两个属性即可。在实际中，角色元素往往和业务系统中职位元素相对应，角色对权限的许可权分配也依赖于职位在业务领域中的定位，例如：业务系统中存在经理和业务员两个职位，则角色模型中基本上也要增加经理角色和业务员角色两个元素。

角色模型和权限模型之间是多对多关系。角色模型还需要考虑在权限授予功能上的角色间的层次关系，其规则是向下继承，即若 A 角色继承了 B 角色，则 A 角色就拥有 B 角色所拥有的所有权限，反之则未必成立。例如：经理角色继承了业务员角色，则经理角色拥有业务员角色的全部权限。但业务员却不一定具备经理的所有权限。

3. 会话模型

会话模型起到精确控制某段时间用户激活角色的作用。会话模型与用户模型以及会话模型与角色模型间分别是多对一和一对多的关系。会话元素不与 MIS 系统元素相关联，可根据权限系统的业务逻辑自行给用户分配会话以及给会话分配角色。

4. 对象模型、操作模型和权限模型

权限模型和角色模型是多对多关系。与之前模型不同之处在于：权限模型中每一个元素都应该在业务系统中找到一个元素与之对应，因此，权限模型也与业务系统结合得最紧密，如何从业务系统元素中抽象构建权限模型也是权限领域模型的核心问题。

(1)业务系统元素的划分

由业务系统架构可知，系统中元素种类大致包括：层次、模块、类、类中的字段、类中属性、类的方法以及数据库中的表、字段、存储过程等，划分如下。

对象模型：比较明显，类中的字段、类中的属性以及数据库中的表、字段等能够存放数据的元素是对象模型的需求领域，可归属于对象模型。类中的方法和数据库中的存储过程等过程式元素，包含了对象和操作两种语义，理论上来说已经属于权限模型的需求领域，但为了方便模型的构建，同样可以把它们归入对象模型需求领域，限定操作模型中它对应的操作元素仅能取“访问”。层次、模块等大的框架性元素均由上述元素逐层构建组合而成，且自身并不包含操作语义，于是同样可归入对象模型需求领域。

操作模型：操作模型的需求领域，对于类、属性、表、字段等元素来说，在理论上是对应业务系统中操作类、表等数据结构的操作语句，例如 LINQ，SQL 等。实际上，这些语句并不会以单独某种元素的形式存在，而都会嵌入到方法、存储过程等过程式元素中。而层次、模块等框架性元素所对应的“访问”操作也同样没有单独的 MIS 系统元素承载。这就意味着业务系统中不存在操作模型的需求领域，从而操作模型也就不需要从业务系统中抽象构建。事实上，操作模型也基本上依赖于对象模型，可以根据对象模型的不同类型，设置相应的“虚拟”操作。

权限模型：由于已经把方法、函数、存储过程等元素归入对象模型，于是权限模型元素取对象模型中对应的 MIS 系统元素和相应的操作模型元素相互组合即可。例外的是用户界面层元素，之前所研究的系统元素均属于“类别”，界面层元素则属于“类别实例”。例如按钮是一个类别，界面层任一界面中就能包含多个按钮，对这些按钮进行权限控制就是对“类别实例”进行权限控制。对于按钮这样的“类别实例”，以及其它常见的界面元素如模块、页面、各种控件以及方法等，其中已经包含了对象和操作这两种语义，且不可再分割，例如一个按钮对应一个订单的删除功能，一个文本框对应一个客户姓名的访问或者显示功能，因此权限模型元素可以在界面层中找到对应的元素。

(2)模型的层次

权限设计中包含权限的分配和授予这一需求，从而要求权限模型必须要考虑层次关系，即父子关系。其规则定义为向上继承的，即若要拥有 A 权限，则必须先拥有 A 权限的父权限，反之则未必成立，则 3 个模型的层次创建规则如下：

对象模型：类、属性、表、字段等数据存储性质元素以及层次、模块等框架性元素在 MIS 系统中基本上是包含与被包含关系，于是被包含者为子对象，包含者为父对象。包含关系使得除最高对象元素外，每一个对象元素只存在唯一的父对象元素。调用关系由于涉及到具体操作，故不应该在对象模型内构建父子关系。

操作模型：操作模型由于依赖于对象模型，因此本身无需定义层次关系。

权限模型：权限模型基本上继承对象模型的层次关系。如果涉及到方法、存储过程等过程式元素的调用关系，则调用者为子权限，被调用者为父权限。方法、存储过程等过程式元素在权限模型中基本上可按照调用关系构建父子权限关系。例如，方法 A 的功能是获取订单内容，方法 B 的功能是把订单内容显示在屏幕上，方法 B 内调用方法 A 实现，即要想执行 B 就必须执行 A，显然 B 为子权限，A 为父权限。调用关系使得一个子权限可以存在多个父权限。

5. 约束模型

约束模型可在 3 种情况下：用户模型-会话模型之间、用户模型-角色模型之间和角色模型-权限模型之间，用于管理这些模型在构建相互关系时的条件控制，因此约束可以定义为关系的关联类，分别用约束 A、约束 B、约束 C 表示，其表现形式即为类中的各个方法，随模型间关联条件的变化而变化。约束模型主要用于实现权限系统内部的约束逻辑，例如某两个角色或某两个会话之间始终保持互斥，某两个权限在分配角色时始终保持关联等。

4.3 模型的构建和实现

根据上述模型分析，我们可得出权限系统领域模型，如图

5 所示。该模型主要从静态角度展示权限系统的整体架构以及系统中各个具体元素的构成和关联，从独立 MIS 系统的角度来看，可以说权限系统主体功能框架已经完成，而且采用类图表示使得该模型结构清晰且直接具备可实现性，即该模型已经能够转化为 C#、JAVA 等程序代码，直接实现权限功能。当然，此时实现的仅仅是权限系统的整体框架功能，还需要在权限系统中增加用户模型、角色模型、会话模型、约束模型、对象模型、操作模型这 6 大模型的增删改查逻辑、角色分配逻辑以及权限分配逻辑等权限系统的业务逻辑，将之转化为 C#、JAVA 等程序代码，并把数据持久化到数据库，这样才能构建一个完整的权限系统。考虑到实现该模型的代码较为庞大且具体逻辑已在模型中展现，在此就不再将其详细展开。

另外需要注意的是权限系统从业务系统元素上构建了权限领域模型，但同时实现该模型的权限系统必须能反作用于这些元素，即权限系统必须能够“嵌入”业务系统中，使业务系统在运行中能够通过权限系统控制各元素的权限。所以权限系统还必须提供一个可以“嵌入”业务系统的接口来调用权限系统的内部功能以实现权限控制功能，可以称之为嵌入接口。该接口可以作用于业务系统中任一元素之上，对其实施权限控制。所以在完整实现权限系统功能后，就必须根据实际需求实现嵌入接口的功能。

4.4 模型范围控制

从理论上来说，权限领域模型通过嵌入接口可以实现对所有业务系统元素的权限控制。但由于业务系统元素的庞大性和复杂性，把其全部纳入权限领域模型的控制范围并作用于其上实施权限控制既无必要也无可能。而对于业务系统而言，包含业务数据和业务规则的业务领域是其核心，因此，权限领域模型的控制领域着重于那些对应业务数据和实现了业务规则的业务元素即可。

参照业务系统的系统架构，具体来说，一般有如下 3 种方法来实现模型范围控制：界面层控制法、数据库层控制法和领域层控制法。其中界面层控制法比较简单直观，应用范围最广，适用于任何架构的 MIS 系统，现实中很多 MIS 系统的权限模块均采取类似方法来实现粗粒度的权限控制。文献[7]研究了 RBAC 在微软 ASP. NET MVC 框架下的应用，即权限模型实现界面层控制的一个较好的例子。数据库层控制法主要应用于业务逻辑采用数据库中的元素来处理实现的 MIS 系统。领域层控制法则适用于以领域模型为基础，构建了反映业务领域的领域模型，并且具备单独的领域层用独立的数据结构(一般是各种类聚集起来的复杂集合)实现该模型的 MIS 系统。本文的客户-订单系统采用了领域驱动设计架构，显然应该采用领域层控制法。

领域层控制法的实现方法：在领域层中，类、属性和方法等元素可构建完整的层次对象模型，以此为基础也能构建操作模型、权限模型。由于领域层是系统的核心，于是其它各层元素可以通过调用和依赖关系视具体情况有选择地纳入权限控制范围。对于数据库，领域驱动设计认为其只是业务模型的一个持久化工具，数据库中元素是业务元素的一个映射，对领域层实现了权限控制也就对数据库中元素实现了权限控制，而且业务流程也基本集中在领域层而不是数据库中，因此，数据库中元素不再纳入权限控制范围。

领域层控制法的优点：权限控制范围广，可以包含 MIS 系统中除数据库元素外的任意元素；权限控制粒度细，可达到属性级别；权限设置较灵活，不依赖于数据库之类具体存储介质数据结构，可根据需求随意设置 MIS 系统元素权限；采用面向对象技术使得对权限模型的分析、构建和实现比其它方式更易于理解且简单灵活，也能实现权限-业务的分离，大大降低实现权限系统的工作量。同时其与界面层层次距离不远，界面层不少元素直接对应领域层中的业务元素，可以较简单地实现界面层的权限控制。

领域层控制法的缺点：仅适用于以领域层为核心的 MIS 系统，其它架构的 MIS 系统无法采用，使得其使用范围较窄。

4.5 模型权限控制实现

确定了模型的权限控制范围之后，就可以使用程序代码具体实现权限控制功能。本节将采用领域层控制法研究在 .NET平台下实现客户-订单系统的权限控制。主要做法是采用 AOP 技术，把嵌入接口的权限功能植入到系统需要控制权限的各元素中，这种方法既能实现权限-业务的相对独立，保证系统的灵活性，又能组合实现客户-订单系统的权限控制功能，具体流程如下。

(1)嵌入接口的实现

嵌入接口在权限系统中实现并提供给客户-订单系统调用。嵌入接口可以用一个类来表示，类中每一个方法对应一个权限功能，可随业务功能变化不断添加，如下所示：

```
namespace 权限系统接口
{
  public class PrivilegeInterface
  {
    /// <summary>
    /// 在某个用户的某个会话下，判断某个元素是否具有权限
    /// </summary>
     /// <param name="用户 id">用户 id</param>
     /// <param name="会话 id">会话 id</param>
    /// <param name="strText">元素名称</param>
    /// <returns></returns>
      public bool JudgePrivilege(int 用户 id,int 会话 id,string strText)
      {
          //读权限系统数据库判断
      }
      /// <summary>
      /// 在某个用户的某个会话下，获取所有具有权限的元素
      /// </summary>
      /// <param name="用户 id">用户 id</param>
      /// <param name="会话 id">会话 id</param>
      /// <returns>元素名称集合</returns>
      public List<string>GetPrivilege(int 用户 id,int 会话 id)
      {
          //读权限系统数据库获取
      }
      //……
      //……
  }
}
```

(2)嵌入接口类转化为特性类

.NET平台下要实现AOP技术有多种方法可选，最常用的还是采用Microsoft Enterprise Library(微软企业库来实现)。首先实现AOP类，这个类用于判断系统中任意一个方法的权限，如下：

```
namespace PrivilegeJudge
{
[ConfigurationElementType(typeof(CustomCallHandlerData))]
  public class JudgePrivilegeHandler:ICallHandler
  {
      /// <summary>
      /// 构造函数,初始化AOP类
      /// </summary>
      /// <param name="用户 id">用户 id</param>
      /// <param name="会话 id">会话 id</param>
      /// <returns>元素名称集合</returns>
      public JudgePrivilegeHandler(int 用户 id,int 会话 id)
      {
        this.用户 id=用户 id;
        this.会话 id=会话 id;
      }
  public int 用户 id{ get;set;}
  public int 会话 id { get;set;}
  private int _order=0;
  public int Order
  {
        get{ return _order;}
        set{ _order=value;}
        }
        /// <summary>
        /// 实现ICallHandler.Invoke方法,用于对具体拦截方法做相应的处理
        /// </summary>
          public IMethodReturn Invoke(IMethodInvocation input,
          GetNextHandlerDelegate getNext)
  {
    //检查参数是否存在
    if(input== null) throw new ArgumentNullException("input");
    if(getNext== null) throw new ArgumentNullException("getNext");
    IMethodReturn result=null;
    //调用嵌入接口类判断权限,参数input.MethodBase.Name是要拦截的方法名称
    bool bolResult=new PrivilegeInterface().JudgePrivilege(this.用户 id,this.会话 id,input.MethodBase.Name);
    if(bolResult== true)//如果权限存在则继续执行要拦截的方法
    {
      result=getNext()(input,getNext);
    }
    //权限不存在则结束,不执行要调用的方法
    return result;
   }
  }
}
```

接着再把AOP类转化为特性类。

```
[AttributeUsage(AttributeTargets.Method)]
  public class JudgePrivilegeHandlerAttribute:HandlerAttribute
  {
    public JudgePrivilegeHandlerAttribute(int 用户 id,int 会话 id)
    {
      this.用户 id=用户 id;
      this.会话 id=会话 id;
    }
    public int 用户 id{ get;set;}
    public int 会话 id { get;set;}
public override ICallHandler CreateHandler(IUnityContainer container)
    { //创建具体AOP类,并调用JudgePrivilegeHandler handler=
      new JudgePrivilegeHandler(int 用户 id,int 会话 id);
return handler;
      }
        }
```

(3)在客户-订单系统中应用特性类

任意取系统中需要判断权限的一个方法GetCustomerName，该方法所在的类必须实现一个接口，如下所示，在这个方法的头上加上特性类的标记[JudgePrivilegeHandler(1,2)]，并输入相应的参数，1-代表用户id，2-代表会话id，参数可根据实际业务的变化而变化。

```
namespace Application
{
  using Domain;
  using Infrastructure;
  public class CustomerService:ICustomerService
  {
    [JudgePrivilegeHandler(1,2)]
    /// <summary>
    /// 获取一个客户的姓名
    /// </summary>
    /// <param name="客户 id">客户 id</param>
    /// <returns>客户姓名</returns>
    public string GetCustomerName(int 客户 id)
    {
      ///调用基础设施层的仓库功能从数据库中取出
    }
  }
  public interface ICustomerService
  {
    /// <summary>
    /// 获取一个客户的姓名
    /// </summary>
    /// <param name="客户 id">客户 id</param>
    /// <returns>客户姓名</returns>
    public string GetCustomerName(int 客户 id);
  }
}
```

(4)客户端调用方法判断权限

```
using Application;
class Program
{
```

(下转第76页)

[53] Argón P, Delzanno G, Mukhopadhyay S, et al. Model Checking Communication Protocols [C]//SOFSEM 2001: Theory and Practice of Informatics. 2001:160-170

[54] Faber J, Meyer R. Model checking data-dependent real-time properties of the European Train Control System[C]// Formal Methods in Computer Aided Design, FMCAD'06. 2006:76-77

[55] Clarke E, Fehnker A, Han Z, et al. Abstraction and counterexample-guided refinement in model checking of hybrid systems [M]. Citeseer, 2003:233-237

[56] Marrero W, Clarke E, Jha S. A model checker for authentication protocols[D]. Rutgers University, 1997:188-196

[57] Basin D, Modersheim S, Vigano L. An on-the-fly model-checker for security protocol analysis[D]. Computer Security-ESORICS, 2003:253-270

[58] Clarke E M, Emerson E A, Sistla A P. Automatic verification of finite-state concurrent systems using temporal logic[J]. ACM Transactions on Programming Languages and Systems, 1986, 8: 244-263

[59] Vardi M Y, Wolper P A. An automata-theoretic approach to automatic program verification[C]// Proceedings of the First Symposium on Logic in Computer Science. 1986:322-331

[60] Bradfield L, Stirling C. Modal logics and mu-calculi: an introduction[M]. Handbook of Process Algebra, 2001:293-330

[61] Emerson E A, Lei C L. Efficient model checking in fragments of the propositional mu-calculus[J]. 1986:267-278

[62] Stirling C, Walker D. Local model checking in the modal mu-calculus[C]// TAPSOFT'89. 1989:369-383

[63] Emerson E A, Jutla C S, Sistla A P. On model checking for the [mu]-calculus and its fragments[J]. Theoretical Computer Science, 2001, 258(1/2):491-522

[64] Clarke E M, Emerson E A, Sifakis J. Model checking: algorithmic verification and debugging [J]. Communications of the ACM, 2009, 52(11):74-84

[65] McMillan K L. Symbolic model checking: an approach to the state explosion problem [R]. Carnegie-Mellon University, Department of Computer Science, Report CMU-CS-92-131. 1992

[66] Holzmann G J. The model checker SPIN[J]. IEEE Transactions on software engineering, 1997, 23(5):279-295

[67] Holzmann G J, Smith M H. Software model checking: Extracting verification models from source code[J]. Software Testing, Verification and Reliability, 2001, 11(2):65-79

[68] Godefroid P. VeriSoft: A tool for the automatic analysis of concurrent reactive software [C]// Computer Aided Verification. 1997:476-479

[69] Telelogic T A U. A SDL tools for real time systems development[OL]. http://www.telelogic.com

[70] Cleavel R, Parrow J, Steffen B. The concurrency workbench: A semantics based verification tool for the verification of concurrent systems[J]. ACM Trans. on Programming Languages and Systems, 1993, 5(1):36-72

[71] 傅海仑. 定理机器证明思想的产生与发展[J]. 科技导报, 2001: 14-18

[72] 林作铨. 一个模态非单调逻辑[J]. 中国科学 E 辑, 1996, 26(3): 276-288

(上接第 49 页)

```
static void Main(string[] args)
{
  //必须使用微软企业库的功能模块来创建接口
  ICustomerService iCustomerService = PolicyInjection.Create
  <CustomerService, ICustomerService>();
  //通过接口调用方法
  string strName=iCustomerService.GetCustomerName(12);
  //如果不存在,则说明没有权限
  if(strName== null)
  {
    Console.Write("您没有权限查看此客户的姓名");
  }
  else//姓名存在则显示出来
  {
    Console.Write(strName);
  }
  }
}
```

客户端调用 GetCustomerName 方法时,会先调用方法上的特性类 JudgePrivilegeHandler 中的 CreateHandler 方法,而这也就是间接调用了 AOP 类中的 Invoke 方法,从而可以调用嵌入接口类通过权限系统判断该方法的权限。

结束语 本文采用领域驱动设计思想,结合 RBAC 模型,针对 MIS 系统,研究了基于领域驱动的权限模型的构建、实现以及实施权限控制的方法。本文所采取的方法可以较好地解决业务和权限无法分开以及系统权限控制粒度较粗等权限系统面对的难解问题,具备较强的通用性,在实际 MIS 系统开发和应用中,也具有较高的应用价值。当前该方法已应用于广西南宁市某事业单位的内部办公自动化系统开发中,取得了良好的应用效果。

参 考 文 献

[1] 王兴伟,王宇. Web 信息系统中基于 RBAC 模型的访问控制模块设计与实现[J]. 大连理工大学学报, 2005, 45(S1):284-286

[2] 范明虎,樊红,伍孝. ASP. NET 中基于 RBAC 的通用权限管理系统[J]. 计算机工程, 2010, 36(1):143-145

[3] 韩道军,高洁,翟浩良,等. 访问控制模型研究进展[J]. 计算机科学, 2010, 37(11):29-33

[4] What is Domain Driven Design? [EB/OL]. http://devlicio.us/blogs/casey/archive/2011/05/16/what-is-domain-driven-design.aspx

[5] 王忠,程磊. 基于领域驱动设计的软件开发[J]. 软件导刊, 2008, 7(2):37-39

[6] Evans E. Domain-Driven Design-Tacking Complexity in the Heart of Software [M]. Addison-Wesley Professional, 2004

[7] ASP. NET MVC 下基于 RBAC 权限认证的设计与实现[J]. 重庆理工大学学报:自然科学版, 2011, 9:75-80