兰州大学

硕士学位论文

# 基于角色的权限管理访问控制系统平台研究与实践

姓名：李兰崇

申请学位级别：硕士

专业：计算机技术

指导教师：马义忠;孙勋

20090501

# 摘　要

在信息技术高速发展的当今社会，随着互联网的飞速发展，各种信息管理系统如雨后春笋般地不断涌现，为了维护信息管理系统的安全及网络安全，权限管理访问控制系统扮演着举足轻重的角色。构建强健的权限管理访问控制系统，保证管理信息系统的安全性是十分重要的，权限管理访问控制系统是管理信息系统中可代码重用性最高的模块之一。但几乎每个单位都有自己的权限管理系统，都要在这方面投入相当的人力、财力，为此我们有了构建统一的、通用的权限管理访问控制模型并赋予实践的念头。本文首先分析已有的基于角色的 RBAC 四种模型，RBAC0、RBAC1、RBAC2、RBAC3 的现状及实施问题；总结其优缺点并对其进行改良，提出了改良的 E-RBAC 模型，并从需求分析、模型架构、模型实施方法等方面对模型进行了详细阐述，并给出了模型实施图；文中还对模型的设计从设计任务、设计约定、对象设计、数据库设计、功能设计、测试设计等方面进行了详细阐述；最后还对模型实施、关键算法、进行了概括阐述，对本系统的特点及使用前景进行了概括总结。

**关键词：** RBAC；部门；用户；角色；权限

# Abstract

In the current society with information technology being developed rapidly, with the fast progress of Internet, big amount of different kinds of information management systems keep emerging. In the area of assuring the safety of the information management systems and the network, RBAC (Role-based access control) system plays an important role. It' s critical to construct a robust permission management access control system to make sure the safety of the management information system security. In the meantime, permission management system is one of the modules which code could be reused to the best degree. However, the current situation is that almost every unit has its own permission management system and invest considerable human and financial resources to it. Hence it dawns on us that we should build a unified, universal permission management model and subsequently put it into practice. This paper is based on the analysis of the status and implementation issues about the role of four RBAC models - RBAC0, RBAC1, RBAC2, RBAC3, summing up their strengths and weaknesses and making improvements to them by proposing new E-RBAC model; It also gives detailed description for the model in the aspects of needs analysis, model structure, and model implementation methods, etc. It proposes the implementation plans as well. Furthermore, it elaborates the model of the design tasks from the design, design agreement, object design, database design, functional design, and testing design aspects, .etc. Finally, the paper briefly explains the way of the implementation of the model and the key algorithm; also it summarizes the features of this system and its prospect in use.

**Key words:** RBAC; Department; User; Role; Permission

# 原 创 性 声 明

本人郑重声明：本人所呈交的学位论文，是在导师的指导下独立进行研究所取得的成果。学位论文中凡引用他人已经发表或未发表的成果、数据、观点等，均已明确注明出处。除文中已经注明引用的内容外，不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究成果做出重要贡献的个人和集体，均已在文中以明确方式标明。

本声明的法律责任由本人承担。

论文作者签名：李美荣　　日　期：2009.4.26

# 关于学位论文使用授权的声明

本人在导师指导下所完成的论文及相关的职务作品，知识产权归属兰州大学。本人完全了解兰州大学有关保存、使用学位论文的规定，同意学校保存或向国家有关部门或机构送交论文的纸质版和电子版，允许论文被查阅和借阅；本人授权兰州大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用任何复制手段保存和汇编本学位论文。本人离校后发表、使用学位论文或与该论文直接相关的学术论文或成果时，第一署名单位仍然为兰州大学。

保密论文在解密后应遵守此规定。

论文作者签名：李玉荣 导师签名：马文忠 日 期：2009.4

# 第1章 绪论

## 1.1. 课题研究的意义

在信息技术高速发展的当今社会，各种信息应用管理系统如雨后春笋般地不断涌现。信息管理系统是一个复杂的人机交互系统，其中每个具体环节都可能受到安全威胁。构建强健的权限管理访问控制系统，保证管理信息系统的安全性是十分重要的。权限管理访问控制系统是管理信息系统中可代码重用性最高的模块之一。任何多用户的系统都不可避免的涉及到相同的权限需求，都需要解决实体鉴别、数据保密性、数据完整性、防抵赖和访问控制等安全服务(据 ISO7498-2)。为此，国际标准化组织 ISO 在网络安全体系的设计标准( ISO7498-2)中，提出了层次型的安全体系结构，并定义了与其对应的五大安全服务功能：身份认证服务、访问控制服务、数据保密服务、数据完整性服务和不可否认服务。例如，访问控制服务要求系统根据操作者已经设定的操作权限，控制操作者可以访问哪些资源，以及确定对资源如何进行操作。

作为五大安全服务之一的访问控制服务目前已成为安全研究的重要方向。通过访问控制服务，既可限制关键资源的访问，也可防止非法用户的侵入或因合法用户的不慎操作所造成的破坏。基于角色的访问控制 RBAC(Role-Based Access Control)作为访问控制的新模式，已逐渐受到人们的认可和重视。基于角色的访问控制就是以角色为基础显式地准许或限制访问能力和范围的一种控制方法。也就是说，传统的访问控制直接将访问主体(发出访问操作、存取要求的主动方)和客体(被调用的程序或欲存取的数据)相联系，而 RBAC 在中间加入了角色，通过角色沟通主体与客体，真正决定访问权限的是用户所对应的角色。

目前，权限管理访问控制系统也是重复开发率最高的模块之一。在企业中，不同的应用系统都拥有一套独立的权限管理访问控制系统。每套权限管理访问控制系统只满足自身系统的权限管理需要，无论在数据存储、权限访问和权限控制机制等方面都可能不一样，这种不一致性存在很多弊端。

采用统一的安全管理设计思想，规范化设计和先进的技术架构体系，构建一个通用的、完善的、安全的、易于管理的、有良好的可移植性和扩展性的权限管

理访问控制系统，使得权限管理访问控制系统真正成为权限控制的核心，在维护系统安全方面发挥重要的作用，是十分必要的。

## 1.2. 课题的研究现状

基于角色的访问控制是美国 NIST 研究会 (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY) 20世纪90代初提出的一种新的访问控制技术。该技术主要研究将用户划分成与其在组织结构体系相一致的角色，以减少授权管理的复杂性，降低管理开销和为管理员提供一个比较好的实现复杂安全策略的环境而著称。目前对于基于角色的访问控制 RBAC(Role-Based Access Control)的研究较为深入的有美国 George Mason 大学 Ravi Sandhu 等人和以 NIST 研究会的 John F. Barkley 为首的DAC 研究小组。以 Sandhu 为代表的学院派主要提出了 RBAC 96 和 RBAC 97 模型(历史上也有很多其他比较成熟的 RBAC 模型，详细内容可参看2.2.1 节)；而 NIST 的 DAC 研究小组主要提出了 RBAC 在商业和政府中的应用架构，特别提出了 RBAC/WEB 模型，将 RBAC 独立于 WEB 服务器和浏览器，给 Intranet提供了一种方便灵活又安全可靠的访问控制策略。 RBAC 既可以嵌入到操作系统或者数据库系统中，也可以在应用级中实现。最近，RBAC 也有了一个最新的统一模型，即Proposed NIST Standard for Role-Based Access Control (ACM TISSEC 2001)。

RBAC 的核心含义是用户不能随意地访问企业对象。取而代之的是，访问许可只能与角色相联系，用户必须是相应角色的成员。这种方法大大简化了授权管理，同时又使指定和执行特定企业保护策略的过程更加灵活。用户可以根据他们的责任和权限成为角色成员，也很容易在不改变基本存取结构的前提下被重新指定新的角色。新的应用和操作产生新的角色许可，需要时角色许可会被取消。

许多用户和生产商不知道构建RBAC的确切定义,但承认其潜在价值。尽管还没有一个关于RBAC的功能组成和优势效力的参考框架,但一些 RBAC 特性也已在商业产品中得以实现。这方面的研究包括：深入调查,以便更好地理解商业和政府用户之安全要求,开发正式的RBAC模式、结构、原型和演示并以此实现其用途和可行性。通过这些努力， RBAC 系统出现了。

目前，RBAC系统多是以美国NIST研究会提出的RBAC建议和Sandhu RBAC 模型

标准作为基础规范而构建的。国外主要的信息网络公司，如Sun公司的Sun Microsystems Logo产品，IBM公司的Tivoli SecureWay等，均提出了自己实用性的类RBAC产品解决方案，国内的三〇卫士通公司以及台湾博询认证公司都有实现该项功能的相关系统产品。同时一些具有RBAC思想的独立产品，包括Informix、Sybase以及Oracle等数据库，Windows NT、2000等操作系统，对RBAC标准都提供了或大或小的支持及创造性的扩展，可见RBAC在信息安全领域中的重要地位。

接下来，我将对我们设计实现的RBAC系统相关的技术背景和原理逐一进行说明。

## 1.3. 论文的结构安排

第一章 绪论：阐述了课题研究的意义和现状，分析了构建强健的权限管理访问控制系统的必要性及迫切性,给出了本文的章节结构；

第二章 基于角色的访问控制原理：详尽分析总结了访问控制技术，重点对基于角色的访问控制（即RBAC）技术的概念、特点、相关算法规则进行了阐述，并给出了RBAC体系结构与设计目标。

第三章 RBAC系统需求分析：以一个企业的组织为例进行了系统需求分析，得出本权限系统的角色、用户、组织结构关系及用例关系，并给出本权限模型需要完成的功能及其操作流程；最后给出了改良后的E-RBAC模型图。

第四章 模型设计：对本模型从设计任务、设计约定、对象设计、数据库设计、功能设计、测试设计等方面进行了阐述。

第五章 模型实施：首先给出模型实施任务及表现形式，接着给出对象—数据库映射、对象复用考虑、编码约定、设计原则，并给出详细的程序结构图、主要类设计，并对各模块的功能进行了简述。

第六章 系统应用举例：对本权限管理系统在“在线考试系统系统”中的应用进行了简述。

第七章 结论与前景：对本系统的特点进行了概括总结，对基于角色的访问控制技术的未来进行了展望。

# 第2章 基于角色的访问控制原理

## 2.1. 相关访问控制技术

访问控制实质上是对资源使用的限制，决定主体是否被授权对客体执行某种操作。它依赖于鉴别使主体合法化，并将组成员关系和特权与主体联系起来。只有经授权的用户，才允许访问特定的系统资源。以用户认证作为访问控制的前提，将各种安全策略相互配合才能真正起到企业信息资源的保护作用。

访问控制的原始概念：是对进入系统的控制。

访问控制的一般概念：是针对越权使用资源的防御措施。

访问控制的目的：是为了限制访问主体(用户、进程、服务等)对访问客体(文件、系统等)的访问权限，从而使计算机系统在合法范围内使用；决定用户能做什么，也决定代表一定用户利益的程序能做什么。

访问控制的作用：是对需要访问系统及其数据的人进行鉴别并验证其合法身份，也是进行记账审计等的前提。

访问控制的基本问题：对于很多组织，系统的数量成百上千，用户数量也成千上万，要被保护的资源的数量常常超过百万，在这种情况下，访问控制的管理复杂性、管理成本和效率就成了访问控制最基本的问题。

目前主要有3 种不同类型的访问控制策略(如图2-1)：

- 自主访问控制策略(Discretionary Access Control Policies)
- 强制访问控制策略(Mandatory Access Control Policies)
- 基于角色的访问控制策略(Role-Based Access Control Policies)

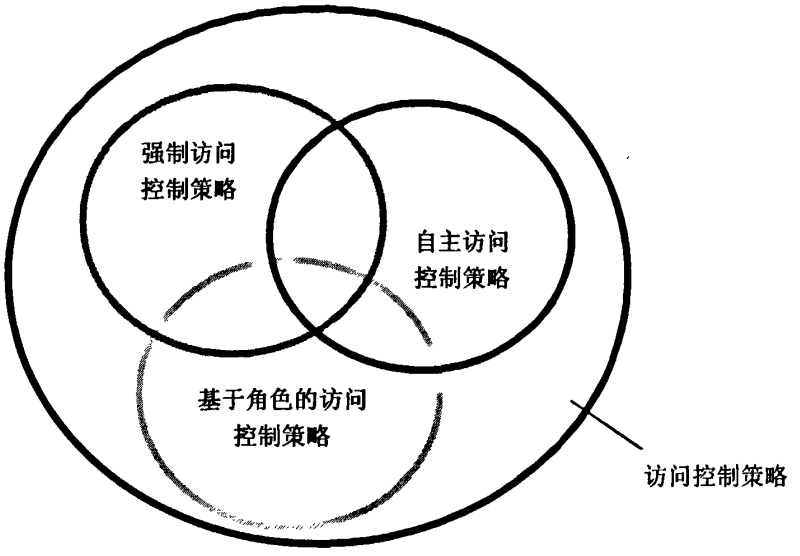



**图 2-1 访问控制策略**

### 2.1.1. 自主访问控制

在自主访问控制安全模型中，自主是指具有某种访问能力的主体能够自主地将访问权限的某个子集授予其它主体的特性，用户对信息的存取控制是基于用户的鉴别和存取访问规则。拥有权限的用户可以自主地将他所拥有的权限传授给其他用户，如图2-2 所示。

• 特点：根据主体的身份及允许访问的权限进行决策。灵活性高，被大量采用。

• 缺点：信息在移动过程中其访问权限关系会被改变。例如用户A可将其对目标O的访问权限传递给用户B，从而使不具备对O访问权限的B可访问O。

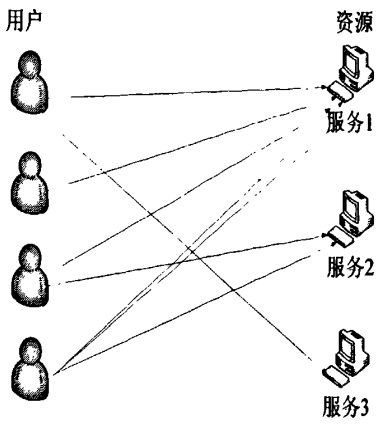



图 2-2 自主访问控制

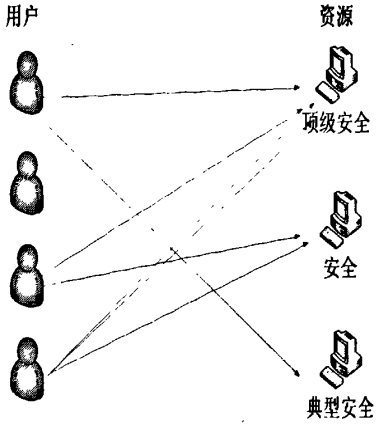



图 2-3 强制访问控制

### 2.1.2. 强制访问控制

在强制访问控制安全模型中，系统给主体和客体分配不同的安全属性，系统通过对安全属性的匹配比较来决定是否允许访问进行。安全属性在安全策略没有被改变之前是不能被改变的，所以用户无权将对系统资源的访问权传授给其他用户，如图2-3所示。

• 特点：强制访问控制的实施取决于能用算法表达的且能在计算机上可执行的策略。策略给出的只是资源受到的限制和实体的授权，而对资源的访问权限取决于实体的授权而非实体的身份。

### 2.1.3. 基于角色的访问控制

1、自主型的访问控制DAC(Discretionary Access Control)是目前计算机系统中实现最多的访问控制机制，它是在确认主体身份以及(或)它们所属组的基础上对访问进行限定的一种方法。其基本思想是：允许某个主体显式地指定其他主体对该主体所拥有的信息资源是否可以访问以及可执行的访问类型。如Windows和Unix就是这种类型。强制型的访问控制MAC (Mandatory Access Control)是“强加”给访问主体的，即系统强制主体服从既定的访问控制政策。它预先定义主体的可信任级别及客体(信息)的敏感程度(安全级别)。用户的访问必须遵守安全政策划分的安全级别的设定以及有关访问权限的设定。这种访问控制方式主要适合

于多层次安全级别的军事应用。这两种访问控制方式有其明显的不足，自主型的访问控制DAC将赋予或取消访问权限的一部分权力留给用户个人，这使得管理员难以确定哪些用户对哪些资源有访问权限，不利于实现统一的全局访问控制。而强制型的访问控制MAC由于过于偏重保密性，对其他方面如系统连续工作能力、授权的可管理性等考虑不足。

2、基于角色的访问控制(Role-Based Access Control)原理

① RBAC的基本思想是：分配给每个用户一个合适的角色，每一个角色都具有其对应的权限，角色是安全控制策略的核心。RBAC的实现机制为：用户与角色关联；角色与权限关联；一个用户拥有某个权限，当且仅当这个用户所拥有的某个角色同该权限相关联。角色控制与自主访问控制DAC、强制访问控制MAC的区别在于角色控制相对独立，根据配置可使某些角色为接近自主访问控制DAC，某些角色接近强制访问控制MAC。RBAC对访问权限的授权由管理员统一管理，而且授权规定是强加给用户的，用户只能被动接受，不能自主地决定。用户也不能自主地将访问权限传给他人。这是一种非自主型的访问控制。用户以什么样的角色对资源进行访问，决定了用户将拥有的权限及可执行何种操作。所以在RBAC中，访问的主体变成了角色。通过采用“角色继承”的概念，把角色组织起来，很自然地反映了组织内部人员之间的职权、责任关系。既提高了效率，又避免了相同权限的重复设置。

② RBAC的优势：

1）RBAC的最大优势在于:它对授权管理的支持。通常的访问控制实现方法是将用户与访问权限直接相联系，当组织内人员新增或有人离开时，或某个用户的职能发生变化时，需要进行大量授权更改工作。而在RBAC中，由于把角色作为用户与资源之间的“中间层”，有效实现了用户和资源之间的沟通。对用户的访问授权转变为对角色的授权，然后再将用户与特定的角色联系起来。一旦一个 RBAC 系统建立起来以后，主要的管理工作即为授权或取消用户的角色。

2）RBAC 的另一优势在于：系统管理员在一种比较抽象且与企业通常的业务管理相类似的层次上。同其它的访问控制方式相比，RBAC能为企业应用带来以下几个明显的好处：简化系统管理、提高组织生产力、减少新雇员停工期、增加系统安全和完整性和量化RBAC等优点。

③ 基本概念

1) 角色(Role)：

角色是建议策略的语义单元，是授权的集合，是一个可以完成一定事务的命名组，一个组织或任务中的工作或者位置，它代表了一种资格、权利和责任。角色既是用户的集合，也是权限的集合。角色将作为中间层，将用户和权限结合起来。不同的角色通过不同的事务来执行各自的功能。

角色与组的区别是：在很多访问控制系统中，用户组都被用来作为访问控制的单元。

对于绝大多数组的实现和角色的概念，其主要区别是：组被作为用户的集合，而角色既是用户的集合也是权限的集合。角色将用于中间层，将用户和权限结合起来。

组：用户集

角色：用户集＋权限集

2) 权限(Permission)：

权限是对计算机系统中的数据或者用数据表示的其它资源进行访问的许可。它允许用户在系统内执行权限许可范围内的操作。

3) 实体(Entity)：

实体表示一个计算机资源(物理设备 ，数据文件，内存或进程)或一个合法用户。

4) 主体(Subject)：

一个可以对其它实体施加动作的主动实体被称为主体，主体可以是用户或其它任何代理用户行为的实体(例如进程，作业和程序)，包括各种用户、其它与本系统有接口的应用程序、非法入侵者。一个主体可以拥有一个或多个角色。

5) 客体(Object)：

一个接受其它实体动作的被动实体被称为客体，客体可以是一个可识别的资源，一个客体可以包含另外一个客体。需要注意的是，一个实体可以在某一时刻是主体，而在另一时刻是客体，这取决于某一实体的功能是动作执行者还是被执行者。

6) 用户(User)：

用户就是一个可以独立访问计算机系统中的数据或者用数据表示的其它资源的主体，在一般情况下是指一个被授权使用计算机的人员。

7)用户标识UID(User Identifier)：

UID是一个用来识别用户的字符串。每个用户具有唯一的用户标识。当一个用户注册进入系统时，他必须提供其用户标识，然后系统执行一个可靠的审查来确信当前用户是对应用户标识的那个用户。

8)角色基数(Cardinality)：

角色基数是角色可以被分配的最大用户数。这是对角色的一个约束性条件。例如，如果一个组织只能有一个部门主任，那么可以用角色基数为“1”来进行约束：一旦已经有某个人被授权为部门主任，其他人就不能再获得部门主任这个角色。

9)角色继承(Inherits)：

角色继承是指一个角色集合中的偏序关系。如果一个角色继承了另一个角色，同时这个角色被授权给一个用户，那么被继承的角色也同时被授权给了这个用户。

## 2.2. RBAC 的技术与特点

### 2.2.1. RBAC 的模型和发展

**1、RBAC 技术的特点**

**① 抽象**

RBAC通过提供一种名为角色的抽象层，来对现实中的用户、操作和资源进行一对多的映射。以抽象的方式管理权限减少了复杂性，使管理结构清晰，并提供了实现复杂访问控制策略的条件。在现实的系统中，抽象层可以被用来对权限进行集中的管理。

RBAC通过抽象层(角色层)简化了访问控制的管理工作，提高了管理效能，改善了操作的直观性，并扩大了访问控制策略实施的范围。在RBAC中，权限与角色关联，用户是角色的成员，用户从相应的角色那里得到相应的权限。

根据组织中的各种职位的职责，角色被集中的创建，根据每个用户的职权，

为相应用户赋予相应的角色。因此，可以很容易地将用户的一个角色撤消，并赋予另一个角色。在新的应用和新的系统加入到组织中时，可能通过对相应角色添加和撤消权限，来使角色的成员用户增加或删除相应权限。例如在RBAC当中，一个用户从一个职位调离到另一个职位时，只需要被简单的撤消一个角色，并赋予另一个角色。如果没有RBAC，用户的过期权限则需要被逐一定位、撤消，并赋予新的权限。

② 用户、角色和权限

传统访问控制(Traditionally Access Control)

对特定数据库或特定应用中信息的访问控制，其实现方式是对组织中的每个用户都要建立相应的权限。如果用户要访问多个应用和数据库，那么对于每个资源，都要赋予该用户相应的权限。

自主访问控制与强制访问控制方式作为传统访问控制方式，其工作量大，又不便于管理。

例如：1000 主体访问10000 客体，需1000 万次配置。如每次配置需1 秒，每天工作8小时，就需要10,000,000/(3600*8) ≈ 347.2天才能完成一次完整的配置工作。

问题(Problem)

(1) 当组织中的用户进入、离开或更改其职权时，要更新每个用户的授权不但很困难、费时，而且容易出错。

(2) 因此，此种方式将导致信息和系统安全的潜在冲突。通过以用户的角色而不是用户的身份作为访问的关键，RBAC避免了这些问题。

在基于角色的模型中，每个用户可能被赋予多个角色，每个角色又可能有多个用户。用户被赋予的角色依据于他们的职权；并且每个角色被赋予访问权限。权限决定数据和应用是否能被访问；并且，每个角色被赋予相应的权限集，使得相应的用户可以完成所需的任务，如图2-4所示。

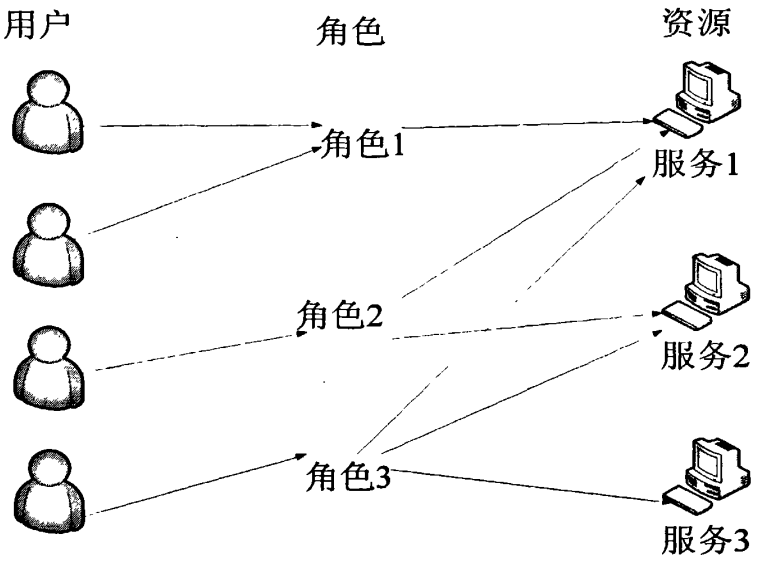



**图 2-4 基于角色的访问控制**

用户的角色可以属于特定的任务、地理位置或个体描述。

在多数情况下，组织里用户职位更改的频率比角色或职权更改的频率高。通过将角色与权限相关联，并只在角色的范围内对用户进行更改，这样管理的成本就降低了。

**③最少权限**

最少权限原则是指有选择地将权限赋予用户，并使用户被赋予的权限仅仅是用户完成其工作所需的且足够的权限。它提供了RBAC保护机制所要求的设置权限分离边界的原则。

最少权限的原则避免了用户超越自身职责既定权限范围，执行非法权限操作的潜在威胁。

**④职责分离**

职责分离就是在不同角色之间，将任务和其相关联的权限进行分离，以防止用户拥有超过他们职位所需的权力。其动机是为了保证欺骗和大的错误不会出生，除非多个用户蓄意勾结。

在RBAC系统中，职责分离的概念是由最小权限的原则所支持的。NIST

RBAC模型就支持静态和动态的职责分离。

(1) 静态职责分离(SSD)

如果两个角色是静态职责分离SSD (Static Separation of Duty)的关系，那么这两个角色不能分配给同一个用户。

在基于角色的系统中，可能出现由于某个用户同时拥有具有相互冲突关系的已经授权的多个角色，而导致出现的矛盾冲突。阻止出现这种冲突的一个方法就是静态职责分离，通过这种方法来约束对用户进行的角色赋予。

这种方法就是一个用户如果被授权给一个角色，(而这个角色同另一角色有静态职责分离关系)，那么这个用户就会被禁止成为另一个角色的成员。例如，一个用户如果被授权为票据文员，那么他就不能被授权为应收账文员，如图2-5所示。也就是说，角色“票据文员”与角色“应收账文员”互斥。

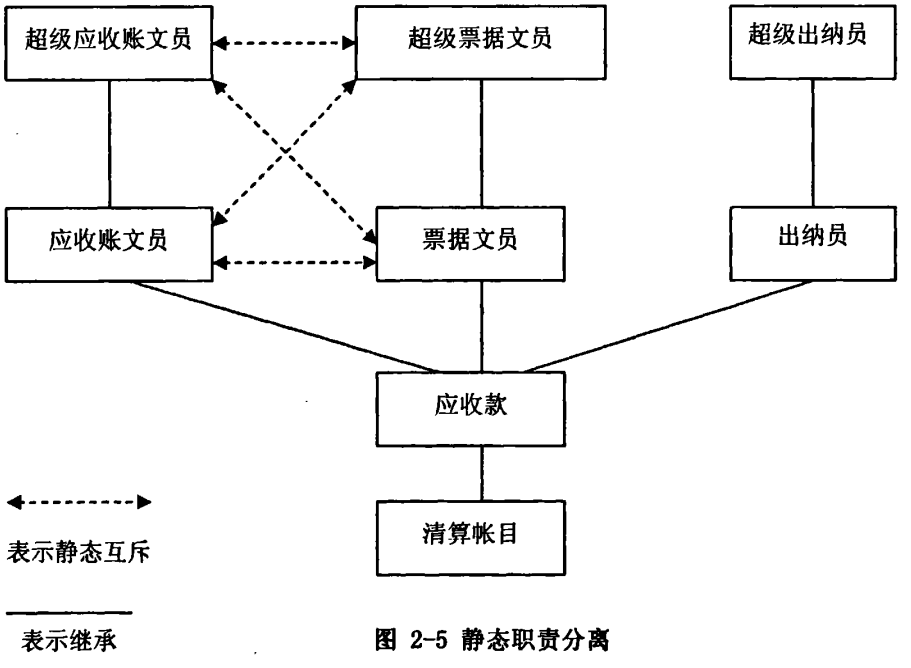



图 2-5 静态职责分离

SSD 策略可以被集中式定制，并被统一地强加在特定的角色上。

这种(职责分离)的约束关系可以被继承。例如，如果一个角色——“超级应收账文员”是由“应收账文员”继承而来的，而“应收账文员”与“票据文员”是静态职责分离的关系，那么“超级应收账文员”的与出纳员也是静态职责分离

的关系。从另一方面考虑，“超级应收账文员”的任何实例都可以被当作“应收账文员”的一个实例，因此，出纳员和“应收账文员”之间的静态职责分离关系也必须适用于“超级应收账文员”。

A. 动态职责分离(DSD)

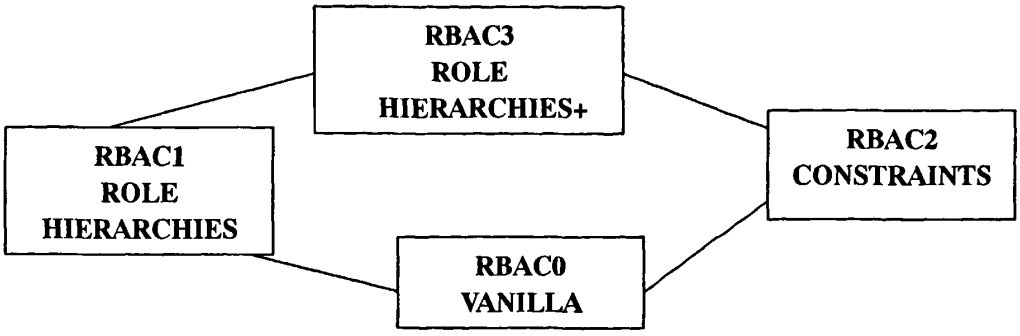
如果两个角色是动态职责分离DSD(Dynamic Separation of Duty)的关系，那么这两个角色可以同时分配给一个用户，但是用户的这两个角色在同一时刻不能都是处于活动状态。

具有动态职责分离DSD关系的角色可以被授权给同一个用户，只要用户在完成任务时分别单独使用其中一个角色时，不会出现冲突，但当用户同时要同时使用具有DSD 关系的两个或多个角色时，便会出现冲突。例如，一个用户可以被同时授权为出纳员和出纳主管；由出纳主管来负责监视出纳员打开收银柜的行为是否合法。如果该用户在使用出纳的角色时，想要转为使用出纳主管的角色，RBAC将要求该用户退出其出纳的角色，并因而要求该用户在行使出纳主管角色之前先关闭收银柜。只要用户不能在同一时刻拥有这两个角色，利害冲突就不会出现。

**2、RBAC 的模型和发展**

在授权管理中使用角色并不是一个新概念，但是在实际的网络管理策略中使用角色是新的。尽管基于角色的安全模型已经存在了20年，对于在计算机网络和安全社区里面实现RBAC的方法至今还没有形成一个一致的概念。结果，对RBAC包括其模型在内形成了一系列从简单到复杂的概念。

Sandhu 等人分析了各种RBAC 定义。他们定义了基本的RBAC模型，即RBAC0，其中包括了最少权限和职责分离。其后的RBAC模型就是建立在这个基本模型之上，并引入了新的继承和约束概念，以加强管理和安全上的利益。从图2-6 所示的RBAC模型家族中可以看出，各模型之间不是孤立的，而是存在一定的关系。




| Models | Hierarchies | Constraints |
|---|---|---|
| RBAC0 | No | No |
| RBAC1 | Yes | No |
| RBAC2 | No | Yes |
| RBAC3 | Yes | Yes |

**图 2-6 RBAC 模型家族**

1) RBAC0(FLAT RBAC，OR VANILLA RBAC)模型

- 用户通过角色拥有权限；
- 必须支持用户-角色授权的多对多映射；
- 必须支持授权-角色的多对多映射；
- 必须支持用户-角色授权的检索；
- 用户可以同时使用多个角色的权限。

2) RBAC1(Hierarchical RBAC)模型

- 基于RBAC0，并支持角色继承；
- 需要支持任意层级继承；
- 支持有限层级继承。

角色继承是在组织内部对角色的授权和责任的一种自然扩展。例如，在一个组织中可能既含有低级又有高级的角色，如果引进角色继承的概念，那么高级的角色就有权访问低级角色能够访问的信息，但是反之并不成立。这种方法可以增加网络管理的效率。高级角色通过继承低级角色的权限避免了对每个角色的一般

权限进行重复性地指定所造成地麻烦。随着组织的增大和权限的增加，角色继承机制所带来的好处就越多。

3) RBAC2(Constrained RBAC) 模型

•基于RBAC0，且必须实行职责分离；

•需要支持任意层级继承；

•支持有限层级继承。

RBAC2 同样是基于最初的RBAC0 模型，但它引入了约束概念。例如，可以约束一个用户是否能拥有某一角色。然而，约束也可以被用于许多其它情况下：约束可以用于建立一个特定角色的成员资格；约束也可以被用于作为授予一个角色的先决条件。例如，可以约束用户能被授权为角色X 的先决条件是他拥有角色Y。

4) RBAC3(对称RBAC，Symmetric RBAC)模型

•基于RBAC1+RBAC2，且必须支持授权-角色关系的检索，并且其效率要比用户-角色关系的检索高；

•需要支持任意层级继承；

•支持有限层级继承。

NIST RBAC模型就是RBAC3。这是最复杂的RBAC模型，它包括了角色继承和约束。在RBAC3中，约束被强加在角色继承关系上。例如，低级别角色可能被限定最大的派生角色数量；多个低级别角色可能被限定只能派生不同的高级别角色；或限定用户所能拥有的高级别角色的数量；在RBAC3中出现的这些继承和约束间敏感的交互使得它成为最复杂的RBAC模型。

### 2.2.2. RBAC 的相关算法规则

1、集合定义与约定

1）集合定义

**USERS**：用户集；

**ROLES**：表示角色集；

**SSD**：表示静态职责分离集合，简称为静态分集；

**DSD**：表示动态职责分离集合，简称为动态分集；

$A \rightarrow B$ ：表示 A 从 B 派生，派生层数是 1，即 B 是 A 的父角色，A 是 B的子角色；

$A \rightarrow^{+} B$ ：表示 A 从 B 直接或间接派生，派生层数是 1~N，即 B 是 A 的祖先角色，A 是 B 的派生角色；

$A \rightarrow^{*} B$：表示 A 从 B 派生，派生层数是 0~N，即 A 可以是 B 本身，也可以是 B 的派生角色；

**Assigned_users(R)**：表示 R 的直接分配用户集合，即直接拥有角色 R 的用户名的集合；

**Authorized_users(R)**：表示拥有角色 R 的用户集合，即直接拥有角色 R的用户集合，以及通过继承关系拥有 R 的派生角色的用户集合之并集；

**Cardinality(R)**：角色 R 的基数(容量)

2）约定说明

**Authorize**：授权

解释：直接指定给某用户的角色，以及通过继承关系所间接拥有的父角色，都属于该用户的授权角色；某角色直接分配的用户，以及通过继承关系，其子角色所分配的用户，都属于该角色的授权用户。

**Assign**：分配

解释：直接指定某角色给某用户，称之为角色分配；直接指定给某用户的角色，属于该用户的分配角色；某角色直接分配的用户，都属于该角色的分配用户。

在我们的RBAC 系统的设计过程中，采用了以下的规则：

**2、RBAC中的基本算法规则**

1)如果一个主体不具有任何角色，那么该主体无权执行任何事务。这一规则保证任何主体只有通过角色才能完成其功能，而不能绕过角色直接访问客体。

2)主体的活动角色必须是经过授权的。规则2在规则1的基础上确保用户只能扮演经过授权的角色。

3)如果某主体要执行某个事务，那么该事务必须是授权给该用户的当前活动角色的。在规则1和2的基础上，规则3确保用户只能执行经过授权的事务(操作)。

本文中我们所设计的RBAC系统ACDF 模块就是基于上面的三条规则进行判断的。

**3、数据库完整性与一致性规则**

尽管我们所设计的RBAC系统以NIST RBAC模型作为模型设计基础，但由于在系统设计开始时尚未有一个国际公认的RBAC标准(最近在NIST官方有一个供讨论的RBAC标准的提案)。而且NIST的RBAC模型过于复杂，为了控制系统的复杂性，本系统对其规则集进行了较大的删改。根据Sandhu所提出的RBAC四层模型，保存了RBAC中最关键的RBAC0，保存了RBAC1中的的权限继承的概念，保存了RBAC2中的静态互斥的概念，并删除了动态互斥和多继承，由此得到了一个复杂性可控的设计。

为使系统数据库完整性与一致性能够得到保障，我们必须遵循以下的完整性规则：

1) 每个角色的授权用户数不能超过其基数(Cardinality)。

$\forall r \in ROLES \Rightarrow authorized\_users(r) \leqslant cardinality(r)$

2)角色自己不能继承自己。

$\forall r \in ROLES \Rightarrow \neg(r \rightarrow^{+} r)$

3)两个静态互斥的角色不能被赋予给同一用户。

$\forall u \in USERS, \forall r_1, r_2 \in ROLERS, (r_1, r_2) \in ssd, r_1 \in assigned\_roles(u) \Rightarrow r_2 \notin assigned\_roles(u)$

4)每个角色不能与自己静态互斥。

$\forall r \in ROLES \Rightarrow (r, r) \notin ssd$

5)静态互斥是具有对称性的。

$\forall r_1, r_2 \in ROLERS, (r_1, r_2) \in ssd \Rightarrow (r_2, r_1) \in ssd$

6)一个角色只能拥有一个父角色，即禁止多继承。

$\forall r, r_1, r_2 \in ROLERS, r_1 \rightarrow\Rightarrow \neg(r_1 \rightarrow r)$

7)禁止循环继承，即一个角色不能通过多层继承到自己。

$\forall r \in ROLES \Rightarrow \neg(r \rightarrow^{+} r)$

8)角色定义不能重复。

9)继承关系定义不能重复。

10)互斥关系定义不能重复。

11)用户的活动角色只能是用户所拥有的角色的一个子集。

12)每个角色的派生角色基数之和加上该角色自身用户数之和不应超过该角

色基数。

$$\forall r \in Rolers, \forall r_i \in ROLERS, r_i \rightarrow r \Rightarrow \left\{ \sum cardinality(r_i) + |assigned\_users(r)| \right\}$$

13)任何两个具有(直接的，或间接的)继承关系的角色不能被赋予给同一个用户。

$$\forall u \in USERS, \forall r_1, r_2 \in ROLES, \forall r_1, r_2 \in \text{assigned\_r oles(u)} \Rightarrow \neg(r_1 \rightarrow + r_2)$$

注：规则13只与用户授权工具相关，在将角色1 r 赋予给用户u 时，实现策略如下：

$$IF \quad r_{1\rightarrow +}, r_2, r_2 \in assigned \quad \_ roles \quad (u)$$

THEN 询问是否将2 r 替换为1 r ，并根据用户的选择进行操作。

14)任何具有静态互斥关系的角色不能存在(直接或间接的)继承关系。

$$\forall r_1, r_2 \in ROLES \ , r_1 \rightarrow^{+} r_2 r_1 \Rightarrow (r_1, r_2) \notin ssd$$

15)如果一个角色从另一个角色(直接的，或间接的)派生，并且被派生角色与第三个角色互斥，则此角色同样与第三个角色互斥。

$$\forall r, r_1, r_2 \in ROLES \ , r \rightarrow^{+} r_1, (r_1, r_2) \in ssd \Rightarrow (r_1, r_2) \in ssd$$

16)用户不能拥有不存在的角色。

$$\forall u \in USERS \ , \forall r \in assigned \quad \_ roles \ (u) \Rightarrow r \in ROLES$$

17)具有授权信息的用户必须存在。

$$\forall r \in ROLES, u \in \text{authorized\_users(r)} \Rightarrow u \in USERS$$

## 2.3. RBAC 体系结构与设计目标

### 2.3.1. 总体结构

根据对RBAC原理和实际需求的分析，我们得到以下RBAC的总体架构模型如图2-1所示。我们将按照图2-1中所示将系统分成以下五个层次分别进行介绍。

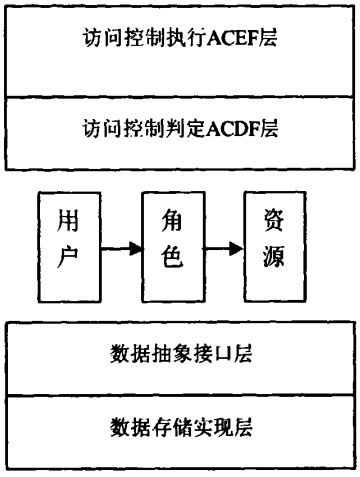



**图 2-7 RBAC 总体架构图**

1、访问控制判决（ACDF）层

这一层是RBAC的核心层，RBAC在定义好用户、角色及其关系和设定受控资源后，就可以对访问请求进行控制了，而这个控制就是ACDF。它根据传递的参数(用户、操作、资源)进行判断，并把判定结果告诉ACEF，ACEF根据判定结果作出相应的处理。

2、访问控制执行（ACEF）层

我们的系统设计主要是针对WEB应用。在B/S应用中，该层通常是由不同平台下的某种Server担负这一使命，如J2EE-based Application Server、Apache、IIS 等。据统计，面向这些Server的Web应用占到全部Web应用的90%以上，而这些Server虽然没有明确包括ACEF功能，但都以某种Filter机制提供了编程接口，允许对资源访问请求进行预处理，方便开发者对其进行扩展，如图2-8 所示。

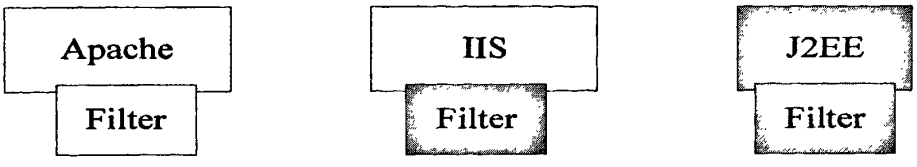



**图 2-8 ACEF 和 ACDF 的联结**

本系统主要通过Apache、IIS 与J2EE 的Filter中加入身份认证模块和访问控制模块，从而把ACDF和ACEF联结起来，通过对请求访问的资源的授权判断并处理，达到对系统资源的保护。

3、数据抽象接口层

这一层为上层提供了一致的数据存储接口，隐藏了不同数据和不同存储模式下的不同类型的接口。这种模式设计所带来的好处是不言而喻的，它使得系统结构清晰，降低上层实现的复杂度且不易出错，很好地体现了面向对象的设计思想。

4、数据存储实现层

这一层完成实际的数据存储功能，在系统中存在不同的数据，如角色及其关系，用户信息，资源和授权信息，这些数据类型不同而且数据量也差异很大，需要分别实现。此外，数据存储方式也可以根据需要选用。

5、访控逻辑业务层

角色、用户、资源这三者共同组成了访问控制业务逻辑层，分别对应RBAC模型中的相应部分。其中用户模块相对简单一些；角色模块在实现逻辑规则时较难，这是因为规则本身的复杂性，但实现后改动较小；至于资源模块，由于资源形式和表现差异很大，涉及到资源的描述、定位、与授权的绑定和同步等等诸多问题，目前我们虽然采用了基于URI的资源描述方式希望能更好的解决这些问题，但从前一版本和已知的实现过程来看，较之以前有所进步，但远远没有得到完美解决。如考虑要兼容未来更好的设计，这一块的变动较大。

### 2.3.2. 系统设计目标

1、主要功能

1）实现角色信息的图形可视化编辑操作功能；

2）实现对用户和目录的权限管理；

3）实现日志管理功能；

4）实现访问控制决定模块(ACDF)及测试工具；

5）实现相关应用所需的RBAC功能组件；

6）提供用户二次开发包(For Windows)。

2、技术指标

1）RBAC设计规则符合相关国际(NIST RBAC)模型标准；

2) 加入RBAC后对资源访问起到相关需求的控制；

3) RBAC系统成功应用于在线考试系统；

4) RBAC系统界面友好美观；

5) 提供集成化管理平台工具软件包；

6) 各种技术文档：需求规格说明书、软件设计、软件测试文档、验收报告。

## 2.4. 实施问题

RBAC 模型是一种纯理论的模型，在实施的过程中不易实现。其原因是：

（1）RBAC 模型是一种纯理论的模型，在实施的过程中不易实现。

（2）RBAC1、RBAC2、RBAC3 三种模型均提到了角色继承这个问题，在实际实施过程中，角色继承会导致模型不易理解，模型结构混乱，因此在实际实施过程中须对角色继承概念进行改良。

（3）RBAC0、RBAC1、RBAC2、RBAC3 四种模型对用户的权限范围定义均不够清晰，导致实施时权限范围不易定义，未达到“实体鉴别、数据保密性、数据完整性、防抵赖和访问控制等安全服务”的效果。

（4）现实世界中，用户均隶属于某一个组织结构，而在 RBAC0、RBAC1、RBAC2、RBAC3 四种模型中均未对用户所隶属的组织结构做出说明，因此对用户管理的实施没有指导意义。

（5）RBAC 模型只是从理论上说明了一个权限系统应该完成什么样的功能，并未对完成该模型所要求的数据结构，数据库做出说明，而现实世界中用户理解 RBAC 模型因知识背景等原因，均不相同，从而造成实现的系统千差万别，一套无岐义的权限管理系统是具有现实紧迫意义的。

# 第3章 RBAC系统需求分析

鉴于上述两章节的叙述，对于我们实际系统的体系结构应该是如图3-1所示：

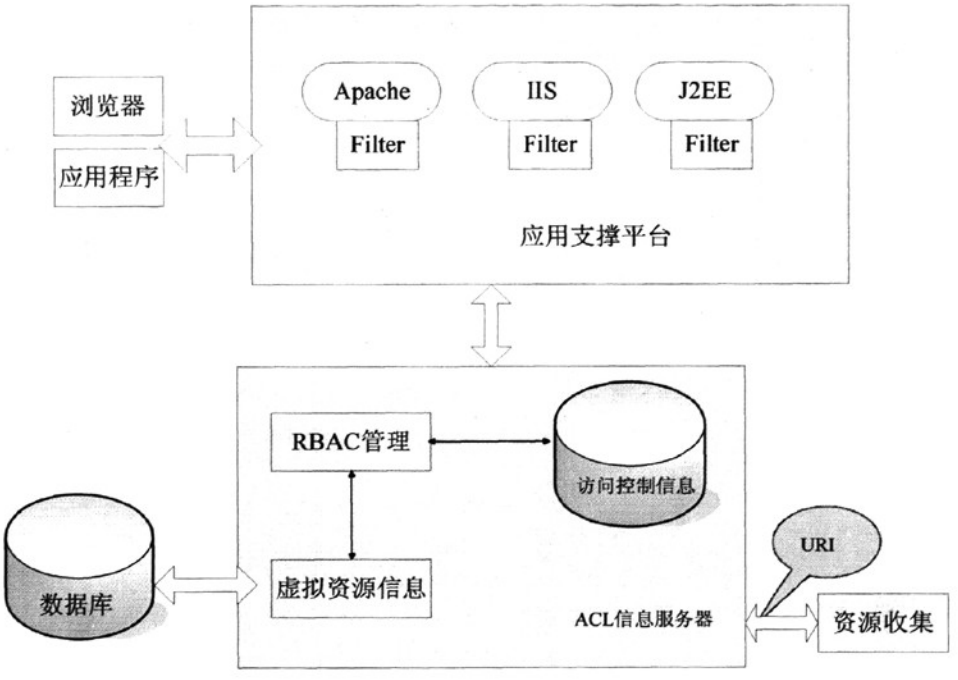



**图 3-1 实际的体系结构拓扑图**

根据RBAC模型，很容易将角色、用户、资源、ACDF判定和数据存储的需求分离开来，为清晰起见，这里我们就分别对其进行描述，在此基础上进行需求分析。

## 3.1 需求分析

### 3.1.1. 需求说明

在一个组织中，首先该组织具有一个组织结构，以一个企业为例，我们可以一个树型结构表达组织结构，如下图 3-2 所示：

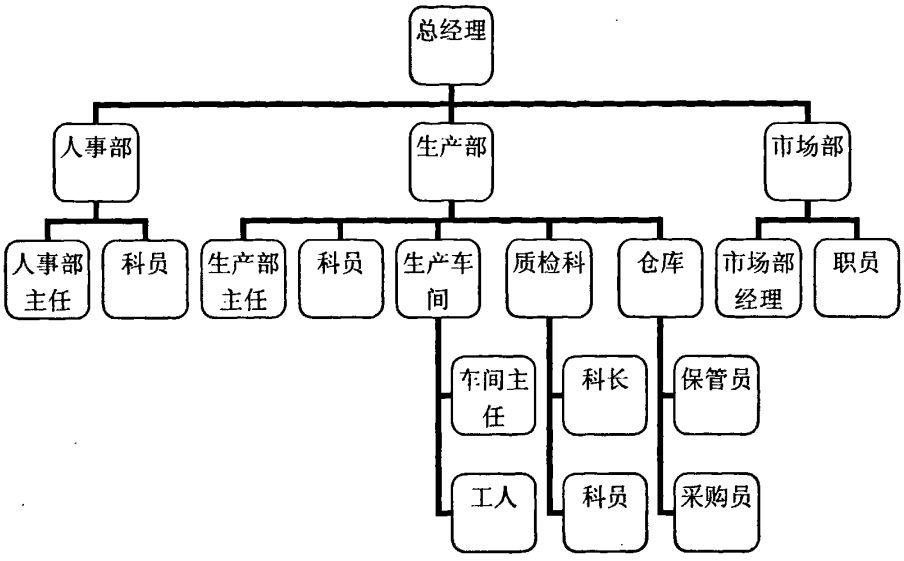



**图 3—2 一个企业树型结构的组织形式**

图 3-2 中显示了一个企业的部分组织结构，其中浅色的为部门，深色的为职位，在这个企业中，有人事部，生产部，市场部，每一个部门下有主任，也有职员，在生产部中下属还有生产车间，质检科和仓库。在组织结构的每一个层次上均有职位。

对应于每一个职位，因各种原因，任用职位的人员会发生变化，如人事部门的人员会因工作需要发生变动，今天人事部主任由张三担任，明天因工作需要人事部主任由李四担任等等；对应一个系统，每一个部门的职位所行使的功能各不相同，体现在本组织结构中，人事部和生产部，市场部均行使不同的功能，如人事部可操作人员的进出，调动，生产部负责产品的生产加工，原料的采购，发放，产品质量的检测等功能，市场部负责产品的销售及售后服务等，总经理负责整个企业的管理工作……

对应于以上的需要，正是权限管理模型所要解决的问题，首先在权限管理模型中要体现出组织结构层次，其次要体现出职位的归属，人员和职位的对应关系；每一个职位（角色）的权限及权限范围。

对应于以上的需求，我们分析如下：

### 3.1.2. 角色、用户、组织结构关系

在该模型中有组织结构、用户、角色、系统管理员、总系统管理员、分系统管理员这六种对象。其中用户隶属与某个组织结构;用户属于一定的角色；系统管理员属于一类特殊的用户，根据其拥有的权限可分为总系统管理员和分系统管理员，它们是特殊与一般的用例关系。

它们的具体关系如图 3-3 所示。

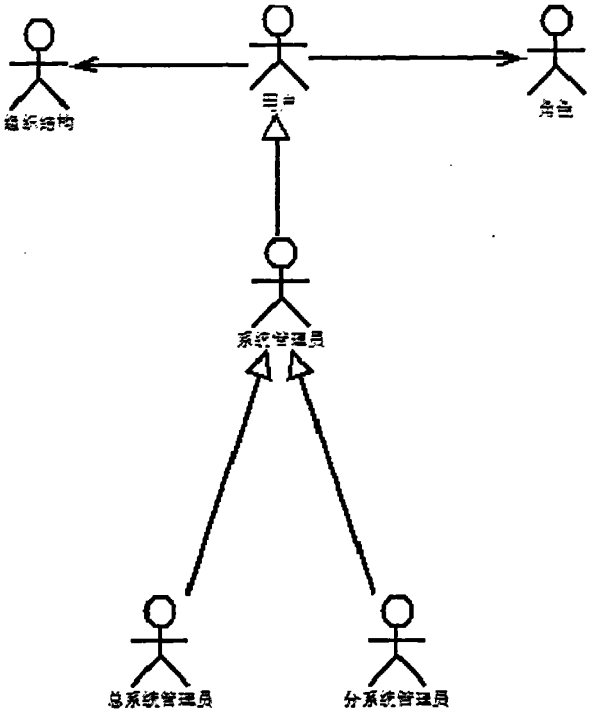



图 3—3 角色、用户、组织结构关系

### 3.1.3. 用例关系

对于本权限模型需要完成以下功能：

A. 登录系统：只有拥有合法帐号及口令的用户才能成功登录系统。

B. 建立角色：只有拥有建立角色权限的特殊用户（即管理员）才能建立角色。

C. 分配角角色权限：只有拥有分配角色权限的特殊用户（即管理员）才能进行角色权限的分配。

D. **建立组织结构：**只有拥有建立组织结构权限的特殊用户（即管理员）才能建立组织结构。

E. **建立用户:**只有拥有建立用户权限的特殊用户（即管理员）才能建立用户。

F. **设置用户权限范围:**只有拥有设置用户权限范的特殊用户（即管理员）才能进行用户权限范的设置。

G. **建立用户角色关系:**只有拥有建立用户角色关系的特殊用户才能建立用户角色关系。

用例关系如图 3-4 所示

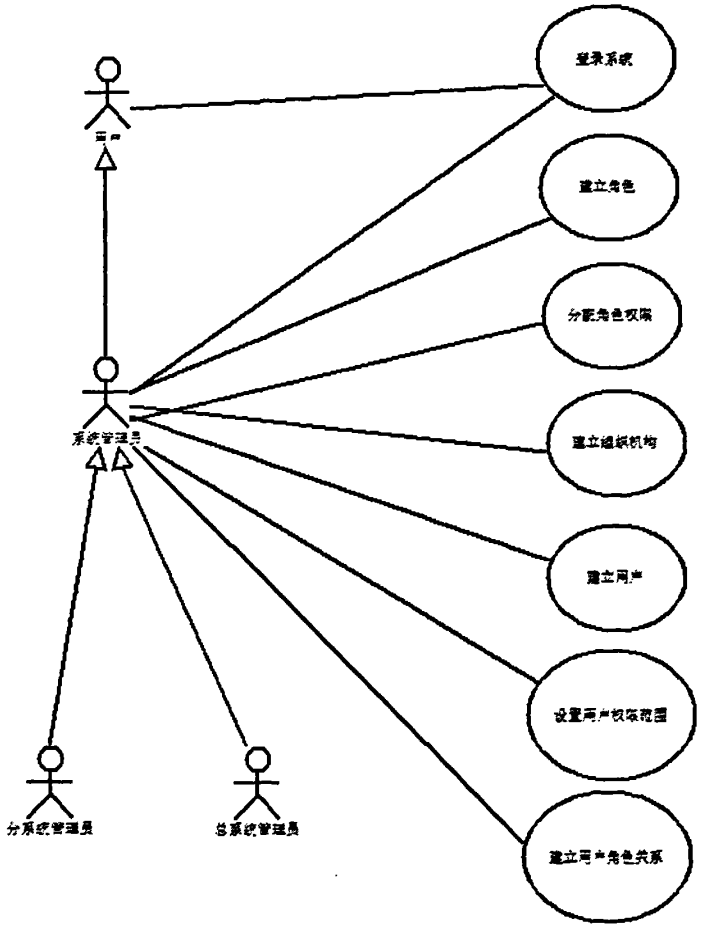



图 3—4 用例关系

### 3.1.4. 需求描述

**1、登录系统**

登录系统过程如图 3-5 所示。

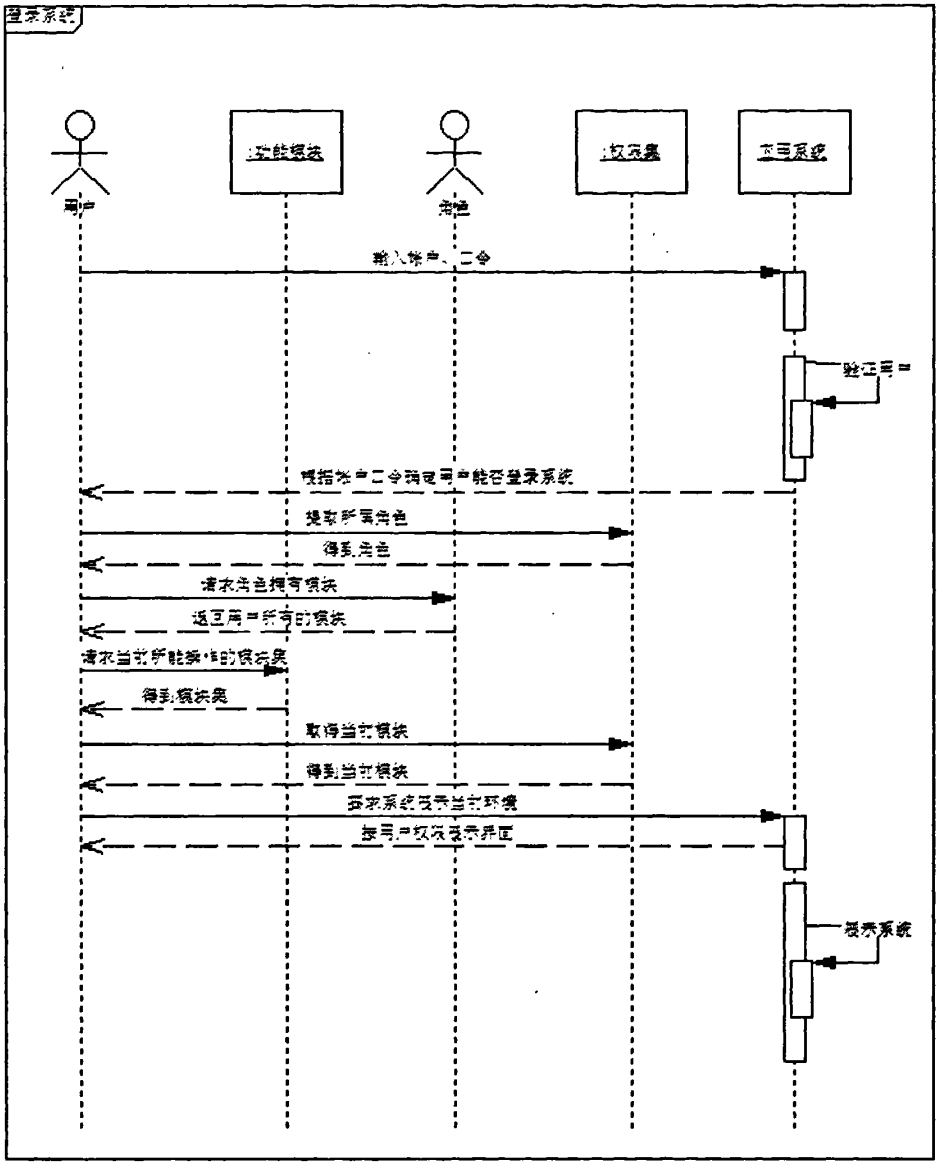


图 3-5 登录系统

1）基本流程

① 用户输入帐号、口令，发出登录请求;

② 应用系统验证用户帐号、口令，返回登录是否成功的信息;

③ 用户请求系统从权限集中提取角色;

④ 系统响应用户请求，用户得到相应角色;

⑤ 用户向系统请求角色拥有相应的模块;

⑥ 系统响应用户请求，返回用户所有的模块;

⑦ 用户向系统请求当前所能操作的模块集;

⑧ 系统响应用户需求，用户获得模块集;

⑨ 用户请求取得当前模块;

⑩ 系统响应用户的请求，用户得到当前模块;

⑪ 用户请求系统展示当前环境;

⑫ 系统响应用户请求，按用户权限展示用户界面。

2）例外流程

第二步如帐号或口令与数据库中的登录数据不符，则提示登录失败的信息并要求重新输入帐号和口令。

**2、 建立角色**

建立角色的基本过程如图 3-6 所示。

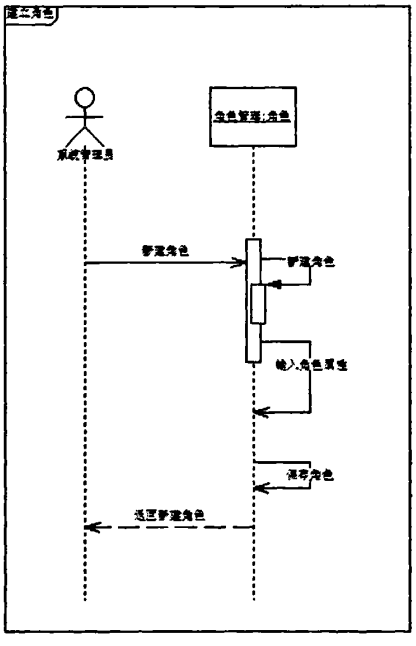


**图 3-6 建立角色**

**基本流程：**

1）系统管理员发出新建角色请求

2）系统建立角色，并给角色赋予初始值（如角色默认名称，排序，状态，描述，备注等信息）

3）输入角色的属性

4）保存属性

约束1：角色名的长度不能超25个字节(显示上的考虑)；

约束2：自动更新同角色关联的继承关系坐标；

约束3：自动更新同该角色关联的互斥关系坐标；

约束4：在角色被删除时，自动删除同角色关联的所有继承关系；

约束 5：在角色被删除时，自动删除同该角色关联的所有互斥关系。

3、 分配角色权限

分配角色权限的过程如图 3-7 所示。

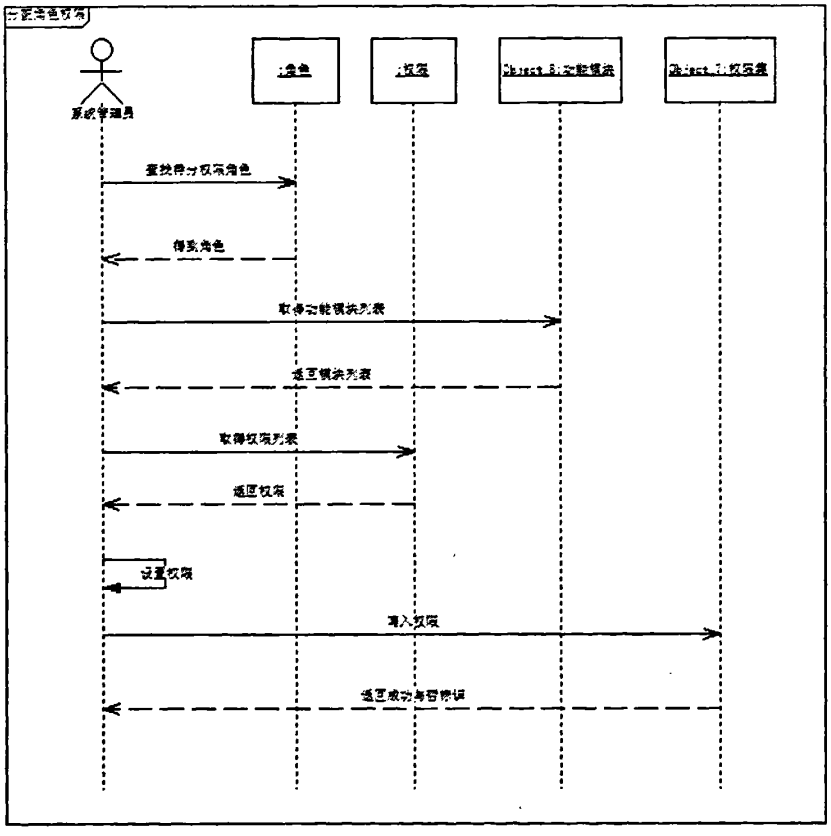



**图 3-7 分配角色权限**

**基本流程：**

1) 系统管理员查找待分配权限角色并得到相应角色；

2) 系统管理员取得功能模块列表并返回模块列表；

3) 系统管理员取得权限列表，并返回权限列表；

4) 系统管理员设置模块权限；

5) 系统管理员将模块权限写入权限集，系统返回写入成功与否标识。

**4、建立组织结构**

建立组织结构的过程如图 3-8 所示。

**基本流程：**

1) 系统管理员按系统提示进行部门的新建操作;

2) 系统返回新建部门。

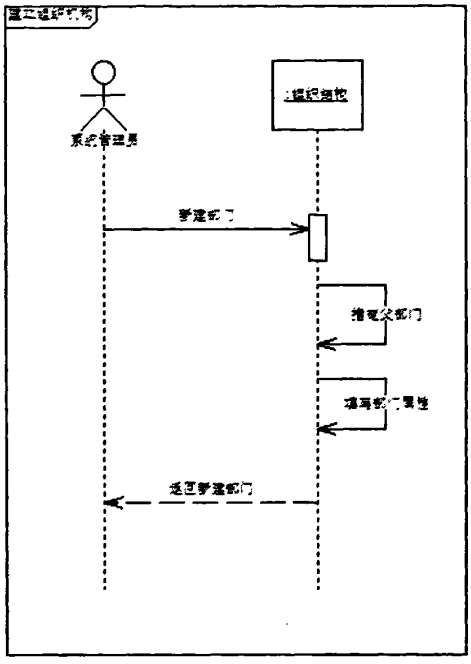


**图 3-8 建立组织结构**

**5、 建立用户**

**基本流程：**

1) 系统管理员取用户所在部门，并返回相应部门;

2) 系统管理员在相应部门新建用户并为用户指定帐户、口令;

3) 系统管理员为新建用户指定用户所属角色，该用户取得用户权限;

4) 系统管理员为新建用户设置用户权限范围，并返回权限范围集合。

5) 系统返回新建用户。

约束1：同步更新角色的已分配数量;

约束2：用户名不能重复。

遵循完整性规则 1，3，13，16，17

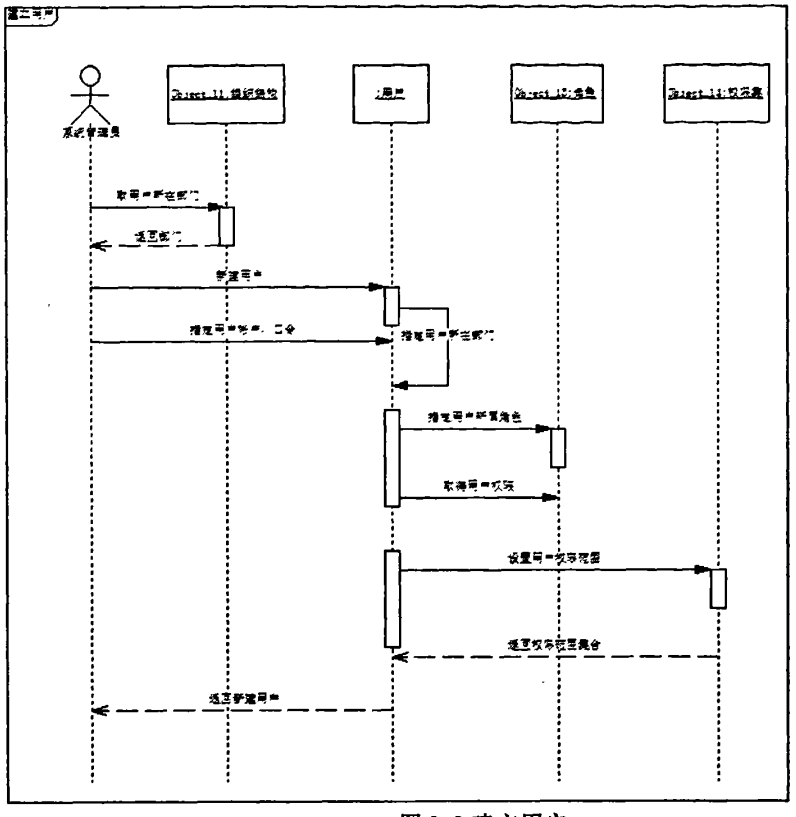


图 3-9 建立用户

6、设置用户权限范围

基本流程：

1） 系统管理员取待授权用户

2） 系统返回用户；

3） 系统管理员取所属角色集合；

4） 系统返回角色集合；

5） 系统管理员取角色集合权限集；

6） 系统返回权限集合；

7） 系统管理员设置功能权限集范围；

8） 系统返回设置成功标识；

9） 功能模块在组织结构集中取相应的部门并返回部门；

10）功能模块在用户集中取用户列表并返回相应的用户。

图 3-10 所示为设置用户权限范围。

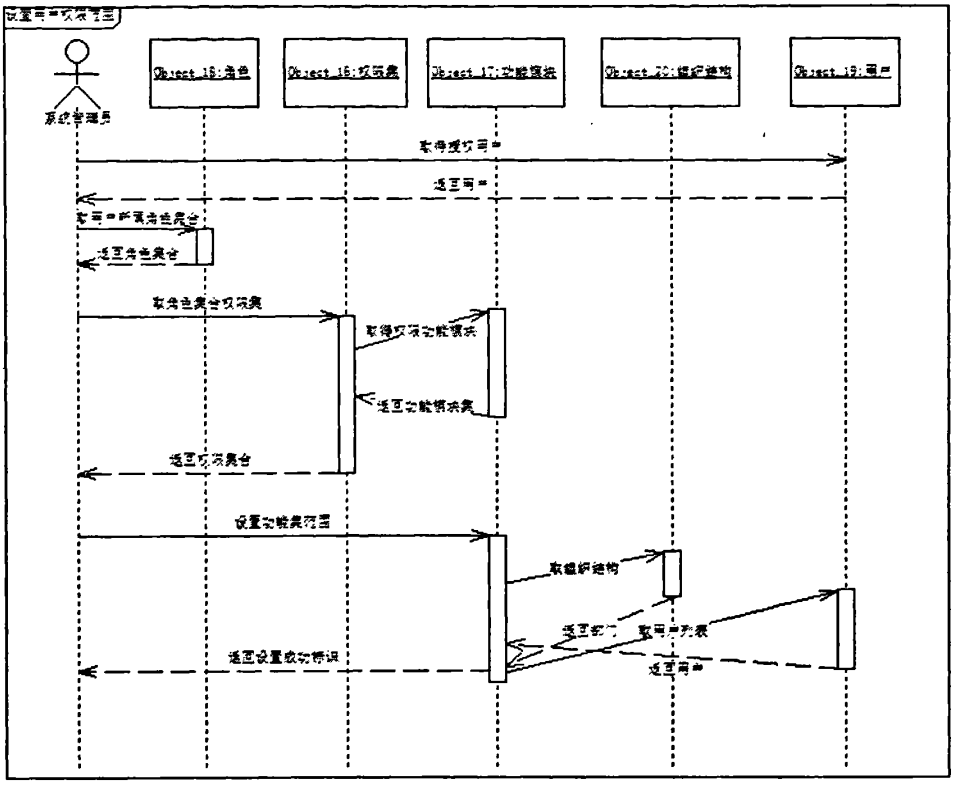


图 3-10 设置用户权限范围

**7、建立用户角色关系**

**基本流程：**

1) 系统管理员取用户所在部门，并返回相应部门；

2) 系统管理员在相应部门新建用户并为用户指定帐户、口令；

3) 系统管理员为新建用户指定用户所属角色，该用户取得用户权限；

4) 系统管理员为新建用户设置用户权限范围，并返回权限范围集合。

5) 系统返回新建用户。

建立用户角色关系如图 3-11 所示。

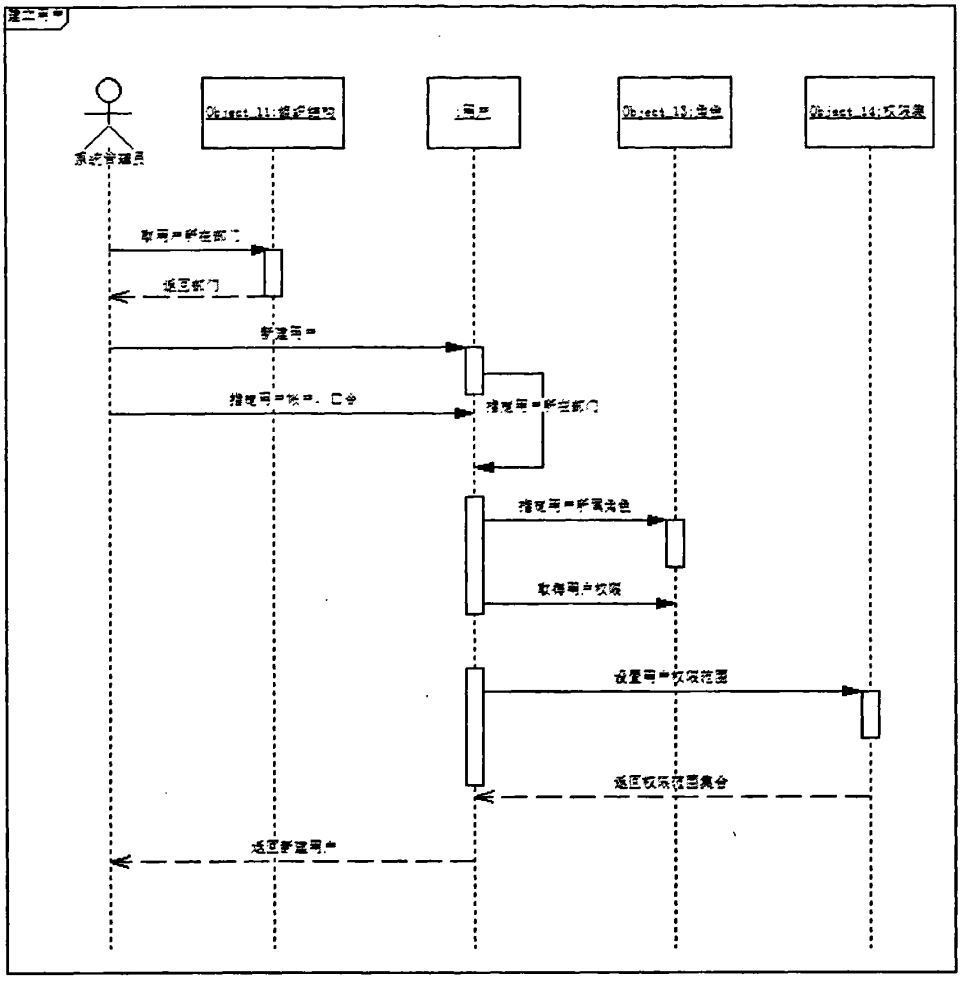


图 3-11 建立用户角色关系

## 3.2. 改良的 E-RBAC 模型

如图 3-12 所示，RBAC 模型图。其中：

1）角色不需要继承关系，全部为平面的

2）所有用户均对应于部门，只有创建了部门才能创建用户，一个用户仅隶属于某一部门

3）一个用户对应 0 或多个角色

4）角色、用户之间，部门、用户之间的关系通过关系表反映

5）角色、用户均有权限集，一个用户的权限集是由所属角色权限集的合集加上用户范围所指定

6）一个用户拥有某一权限指的是用户所属的角色首先有这个权限集，其次是

用户在这个权限集上有操作范围才能说该用户拥有这个权限。

7）权限相对简化为“读”、“写”两种操作，在特殊情况下可扩展。

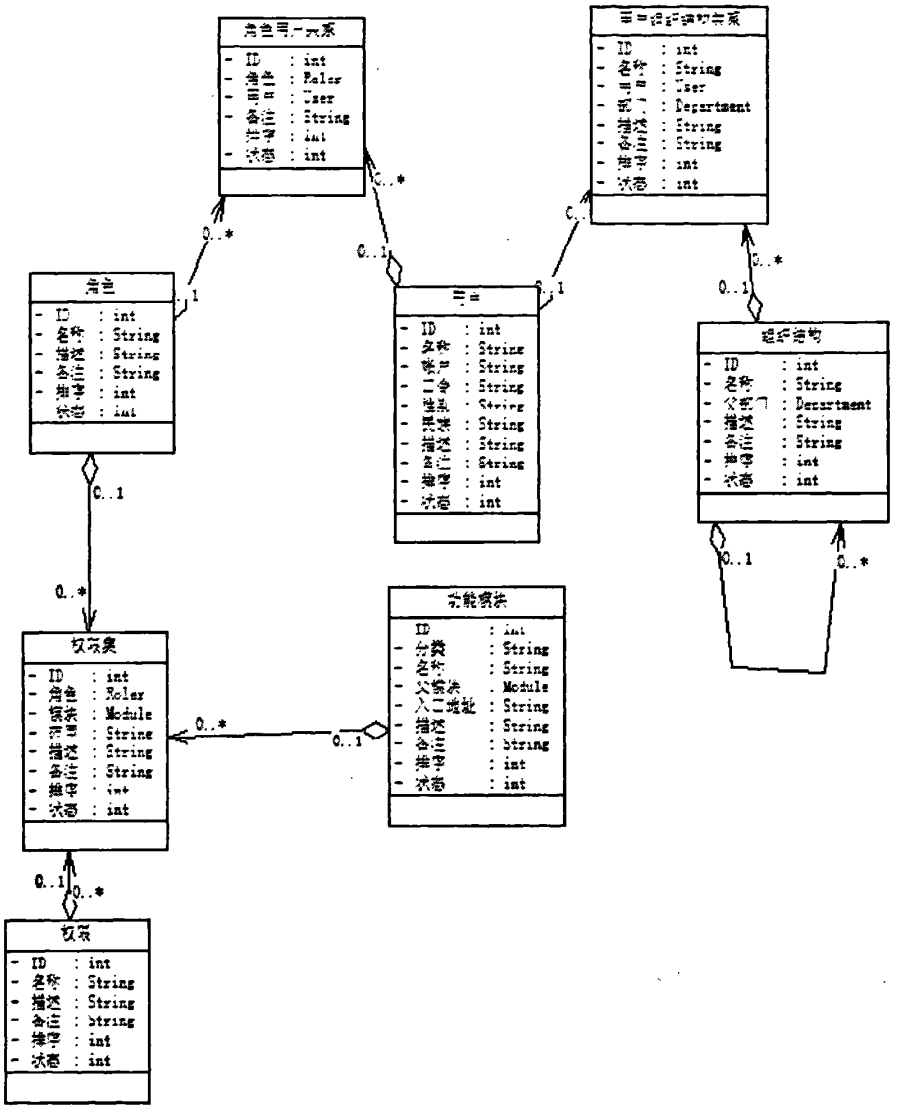



**图 3-12 改良的 E-RBAC 模型图**

## 3.3. 关于资源的管理

描述资源是基于以下两项需求：RBAC可以对多种资源进行授权和访问控制；权限控制信息与目标对象的存储分离。

### 3.3.1. 权限控制信息与目标对象的存储分离

1)对象范围是什么，如何标识一个对象

在RBAC应用中，对象为受保护的资源，对对象的标识，实际是是对受保护的资源的标识，该项需求实际上是为了确定一个合适的资源标识框架。

2)权限控制信息与对象分离的含义及其所带来的问题

所谓权限控制信息与对象分离，是指权限控制信息的存储在“物理”位置上与资源的存储独立，所引发的问题包括：在资源的收集过程中如何获得受控资源的清单，是用户提供，还是程序自动完成？收集到的资源清单如何展现？是根据资源存储的“物理”结构，还是根据资源本身的语义结构？收集到的资源清单如何存储？是根据资源存储的“物理”结构，还是根据资源本身的语义结构？存储结构与资源查找/定位(ACDF 的需求)的效率有密切的关系。在资源位置、内容发生变更的情况下，又如何保证其与资源标识的一致性？

针对以上分析，将需求分解为三个子模块：资源的采集、授权以及存储模块，并形成如下的资源管理工作流程如图3-13所示。

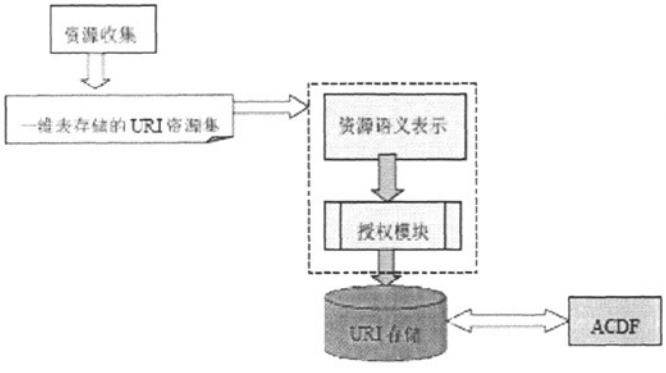



**图 3-13 URI 的工作过程**

根据图3-13中的流程，用户操作与系统交互分为如下步骤。

**第一步：资源收集**

我们常常使用且希望加以控制的资源可分为两类：层次的和关系的。两者典型的例子分别是目录树和数据库记录。由于这两种资源存在着如此明显的差异，因此，根据这两类资源的特点分别设计，可以大大加速资源收集的速度。我们为这两种资源分别提出了自动和手工两种收集方式，以提高资源收集的效率。自动

方式利用资源的层次关系遍历，大大减轻了管理员的工作量。

在完成收集后，还需对该资源集进行管理维护，如增删等，这些需求也必须在系统中体现出来。

对于种类繁多的资源类型，我们还希望对资源的操作不仅仅限于系统规定好的几种权限，如读、写、执行等，虽然可以通过包括尽量多的可选项，但更为灵活的方式是让系统提供定制权限的功能，如用户可根据需要增设上传，下载等等权限。为此，我们让每个资源集都绑定一个定制的操作权限集，可控资源集必须与一套操作权限集相对应，没有相应的权限集的资源是个永远无法达到的“虚”资源。这样，需求中应包括对权限集的定义、维护和资源集绑定的维护。我们把资源集和其对应的权限集绑定在一起后的结合体称之为抽象对象。

简单的说，就是：1 个抽象对象＝1 个资源集＋1 个操作权限集 。

**第二步：资源授权**

在完成抽象对象的构造之后，就可以进行授权了，系统获取可用角色信息后，根据业务需求增删对角色的授权；当然，其间还需指定资源集及显示其中的资源。

**第三步：资源存储**

用户需要保存在系统中完成的角色－资源－授权信息，以便ACDF 的调用。

### 3.3.2. 精化需求

通过对资源标识方法的分析，就可以较容易地实现对常用资源(如Web 资源)的表示和收集工作，同时以界面友好的方式提供对资源的访问授权，形成查找效率高的存储权限表，存储在特定的介质中供ACDF 调用。具体需求描述为：创建URI 资源集：包括自动和手工方式；导入URI 资源；修改URI 资源集；创建URI操作权限集；修改URI 操作权限集；绑定资源集和操作权限集形成一个抽象对象；修改角色对URI 资源的授权；增加角色对URI 资源的授权；删除角色对URI 资源的授权；保存授权信息；载入授权信息；记录相关日志。

综合以上的功能叙述，我们可以举一个针对URL 资源存储的用例(图3-14)：硬盘上的目标目录①映射为IIS 服务的宿主目录②，通过我们的授权模块对其按照事先约定的规则转换为相应的访问控制文件③；如果浏览器提出访问页面①(文件形式)的请求，ACDF 功能模块根据事先已形成的访问控制文件③进行判

决，而后给出允许还是拒绝的判决结果。

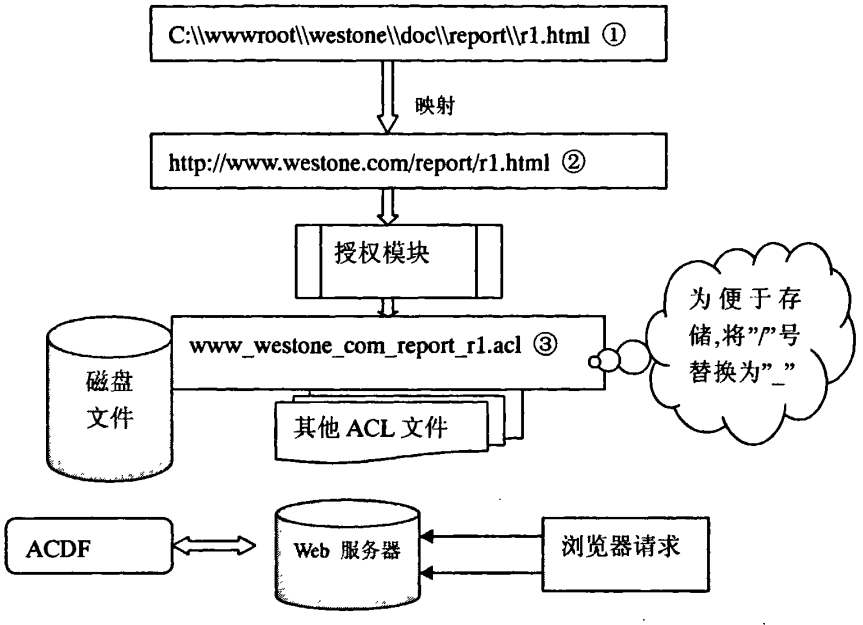



**图3-14 一个URI 的例子**

## 3.4. 关于 ACDF 和 ACEF

ACDF 这部分的需求随着应用平台和服务器的不同，存在相应的变化。作为RBAC 的核心部分，ACDF 根据传递的参数(用户、操作、资源)进行访问控制判断，并把判定结果告诉ACEF，ACEF 根据判定结果作出相应的处理。

系统的设计目标主要是针对WEB 应用，在B/S 应用中，ACEF 通常是由不同平台下的某种Server 担任，如J2EE-based Application Server、Apache、IIS 等，这些Server 虽然没有明确包括ACEF 功能，但都以某种方式提供了编程接口，允许对资源访问请求进行预处理，方便开发者对其进行扩展。因此，我们的需求可以归结为：对应这三种主要的平台，实现ACDF 和ACEF 的联结，替换原平台的安全机制中的访问控制服务，为管理员提供更加灵活自主的安全控制。当然，我们这部分的需求就一分为三，分别是Apache、IIS 和J2EE 下的ACDF 实现，由于涉及具体的平台，为便于浏览，我把他们的需求和设计都放在详细设计部分。

图3-15 是ACDF 的通用流程说明。

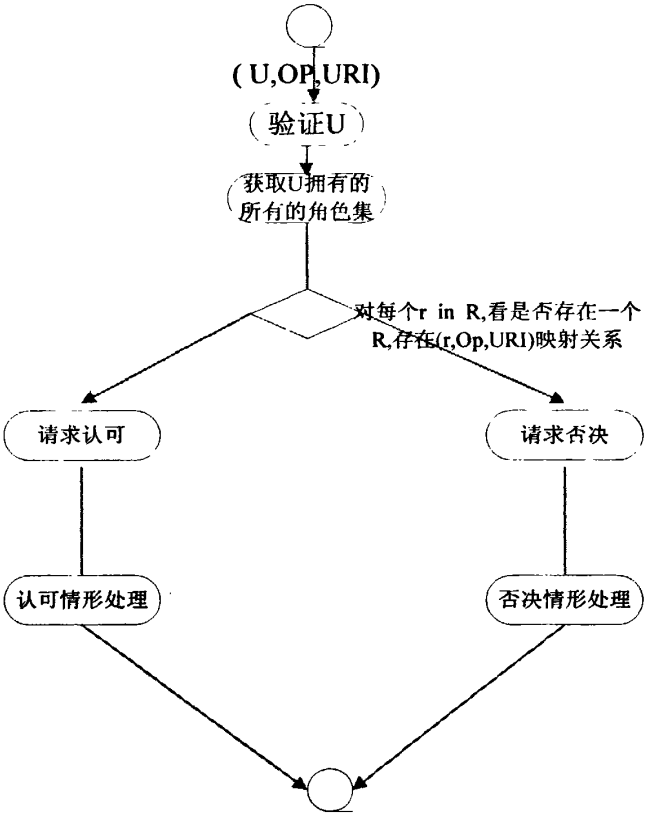



**图3-15通用的ACDF流程图**

ACDF 有着比较高的性能要求和具备多线程能力，这里也是整个系统的瓶颈所在。对ACDF 的性能优化能显著提高访问控制并发能力。

对ACDF 的测试分为两个阶段，初期是功能测试，后期主要是对其并发能力作压力测试，这样的测试工具有很多，如Load Runner，Grinder，至于测试用例也不复杂，这里就不再多费笔墨了。

## 3.5. 关于抽象数据的存储

这一需求是为了屏蔽所有跟数据存取有关的操作，使得其他模块不再需要了解具体的存储模式和位置，只需要提交必要的存储信息给本模块，然后在模块内部将请求分派到具体的存取功能。这样的设计保证了足够的灵活性，大大提高了系统的重用能力。下面是该模块需要提供的高级存取接口：取角色信息；存取图元信息；存取用户信息；存取用户角色分配；存取资源；存取权限；存取抽象对象；存取授权；读取角色列表；更改角色已分配数量信息。

## 3.6. 模型架构 MVC

MVC 模式是"Model-View-Controller"的缩写，中文翻译为"模式-视图-控制器"。MVC 应用程序总是由这三个部分组成。即把一个应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离，这样一个应用被分成三个层：模型层、视图层、控制层。Event(事件)导致 Controller 改变 Model 或 View，或者同时改变两者。只要 Controller 改变了 Models 的数据或者属性，所有依赖的 View 都会自动更新。类似的，只要 Controller 改变了 View，View 会从潜在的 Model 中获取数据来刷新自己。

对应于权限管理系统，View 代表用户交互界面，处于显示部分，它既可以是 B/S 结构程序的页面，也可以是 C/S 程序的前端用户界面。随着应用的复杂性和规模性，界面的处理也变得具有挑战性。一个应用可能有很多不同的视图，MVC 设计模式对于视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。业务流程的处理交予模型(Model)处理。比如一个订单的视图只接受来自模型的数据并显示给用户，以及将用户界面的输入数据和请求传递给控制和模型。

Model 是系统的存储部分，它用于保存系统的各个参与实体在计算机中的存储，存储既包括了内存中的存储部分，也包括了实体在关系数据库中的物理保存。模型就是业务流程/状态的处理以及业务规则的制定。业务流程的处理过程对其它层来说是黑箱操作，模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。目前流行的 EJB 模型就是一个典型的应用例子，它从应用技术实现的角度对模型做了进一步的划分，以便充分利用现有的组件，但它不能作为应用设计模型的框架。它仅仅告诉你按这种模型设计就可以利用某些技术组件，从而减少了技术上的困难。对一个开发者来说，就可以专注于业务模型的设计。MVC 设计模式告诉我们，把应用的模型按一定的规则抽取出来，抽取的层次很重要，这也是判断开发人员是否优秀的设计依据。抽象与具体不能隔得太远，也不能太近。MVC 并没有提供模型的设计方法，而只告诉你应该组织管理这些模型，以便于模型的重构和提高重用性。我们可以用对象编程来做比喻，MVC 定义了一个顶级类，告诉它的子类你只能做这些，但没法限制你能做这些。业务模型还有一个很重要的模型那就是数据模型。数据模型主要指实体

对象的数据 保存(持续化)。比如将一张订单保存到数据库，从数据库获取订单。我们可以将这个模型单独列出，所有有关数据库的操作只限制在该模型中。

Controller 是系统的逻辑部分，这部分主要在后台执行，用于完成系统各个部分之间调用关系的协调。控制(Controller)可以理解为从用户接收请求，将模型与视图匹配在一起，共同完成用户的请求。划分控制层的作用也很明显，它清楚地告诉你，它就是一个分发器，选择什么样的模型，选择什么样的视图，可以完成什么样的用户请求。控制层并不做任何的数据处理。例如，用户点击一个连接，控制层接受请求后，并不处理业务信息，它只把用户的信息传递给模型，告诉模型做什么，选择符合要求的视图返回给用户。因此，一个模型可能对应多个视图，一个视图可能对应多个模型。

模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其它依赖于这些数据的视图都应反映到这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，导致显示的更新。这实际上是一种模型的变化-传播机制。模型、视图、控制器三者之间的关系和各自的主要功能，如下图 3-16 所示。

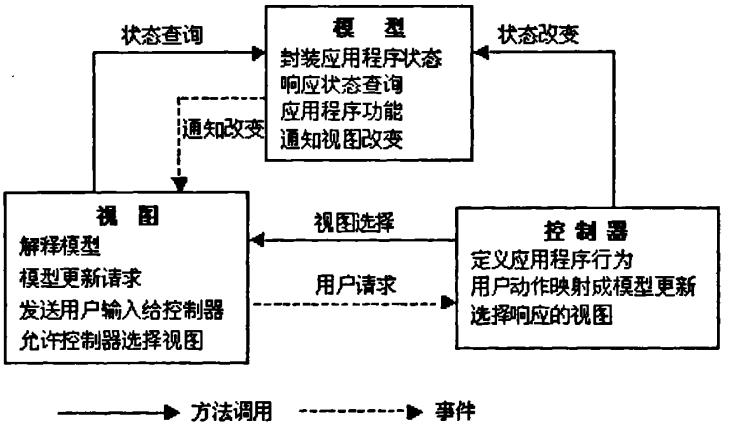



图 3-16 MVC 组件类型的关系和功能

大部分用过程语言比如 ASP、PHP 开发出来的 Web 应用，初始的开发模板就是混合层的数据编程。例如，直接向数据库发送请求并用 HTML 显示,开发速度往往比较快,但由于数据页面的分离不是很直接,因而很难体现出业务模型的样子或者模型的重用性。产品设计弹性力度很小，很难满足用户的变化性需求。MVC

要求对应用分层，虽然要花费额外的工作，但产品的结构清晰，产品的应用通过模型可以得到更好地体现。

首先，最重要的是应该有多个视图对应一个模型的能力。在目前用户需求的快速变化下，可能有多种方式访问应用的要求。例如，订单模型可能有本系统的订单，也有网上订单，或者其他系统的订单，但对于订单的处理都是一样，也就是说订单的处理是一致的。按 MVC 设计模式，一个订单模型以及多个视图即可解决问题。这样减少了代码的复制，即减少了代码的维护量，一旦模型发生改变，也易于维护。 其次，由于模型返回的数据不带任何显示格式，因而这些模型也可直接应用于接口的使用。

再次，由于一个应用被分离为三层，因此有时改变其中的一层就能满足应用的改变。一个应用的业务流程或者业务规则的改变只需改动 MVC 的模型层。

控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起完成不同的请求，因此，控制层可以说是包含了用户请求权限的概念。

最后，它还有利于软件工程化管理。由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化产生管理程序代码。

## 3.7. 实施方法

对于一个这样的权限管理系统，采用什么样的方式实施，决定了这个系统的成败，在规则和需求瞬息万变的商业领域，传统的软件工程方法很难适应快速的变化，给软件企业带来了巨大的困扰。

敏捷方法给我们一种新的思路：与其试图控制变化，不如努力拥抱变化。建立一种适应变化、拥抱变化的开发模式，软件企业将在充满变化的市场竞争中占得先机。为拥抱变化，敏捷方法提倡采用先进的技术和高水平的小型团队，并辅以人性化的管理，充分发挥软件开发者的才华，建立畅通的交流机制，使软件开发团队与客户共同进步，在项目中实现双赢。

应该特别强调的是：尽管敏捷方法强调人性化管理，但在敏捷团队内部对于软件开发者有着极其严格的纪律约束，甚至比绝大多数传统软件工程方法更加严格。对于软件开发者的个人修养和工作方式，XP（eXtreme Programming，XP）等敏捷方法提出了一整套“军规”，其严格、精密的程度甚至能够以分钟计。拥

有良好习惯的优秀开发者，才有可能充分发挥自己的才华。

要理解“敏捷开发”还得从软件工程说起，为了理解软件工程，我们首先需要了解在早期的软件工程文献中提到的那些项目。稍做研究，你就会发现一个令人惊讶的事实：这些文献中几乎没有对商用软件的报告。在所有的案例中，绝大多数都是大型国防项目。这些大型项目是真正的“系统工程”项目。这些项目通常同时包含硬件和软件的开发，其中的硬件是专为与软件协作而开发的。这一类项目有一个共通的特点：在项目的前期，软件开发者需要等待硬件的开发；而在项目的后期，则是硬件开发者在等待软件的开发。软件工程正是在这样一对矛盾中发展起来的。

在典型的软件工程项目的前期，软件开发者会有很多的时间。这时，硬件的发明或者设计尚未完成，所以软件开发者有大把的时间来做需求调研，并编写出详尽的软件设计规格书。设计团队则可以根据需求文档编写出极其详尽的设计规格书。他们甚至可以进行细致入微的设计复审——反正也没有别的事可以做。

一旦硬件能够投入使用，软件开发者就应该立刻动手将详细设计规格转换成代码。如何加快交付的速度？最简单的答案就是：“投入大量的人手。”于是，软件工程的理论就这样顺理成章地建立起来了：项目前期，用大量的时间进行需求分析和详尽的设计；进入编码阶段之后，投入大量“软件蓝领”进行简单而繁复的代码编写。不同职能的开发者泾渭分明，“分析师”、“设计师”、“程序员”这样的称谓标明了他们的细化分工，彼此之间用浩如烟海的文档交流。

软件工程另一个引人注目的方面就是那种确定的、可重复的开发方式。对于强调安全性的软件系统，这种有组织、有纪律、可计量的开发方式已经被证明是非常有效的。美国国防部的某些项目甚至达到了低于同等规模商用项目 1%的错误率。但是，在这样的过程中，其他的约束条件则不得不被放松，就在那些错误率极低的项目中，平均每行代码的资金投入达到了 1 美元。

对于普通的商用软件，你或许要问：我们是否愿意投入那么大的成本，换取那么高的质量？然而，更加重要的是：我们是否有那么多的时间来进行详尽的分析和设计？面对用户本身对需求都不甚明了、业务规则又时常变化的软件项目，这种“有组织、有纪律、可计量的开发方式”真的合适吗？

在这种形势下“敏捷方法”就诞生了。2001 年，一群强调实践、以自己的

程序员身份为荣的软件大师们成立了一个松散的组织：敏捷联盟（Agile Alliance）。他们提出了自己对于软件开发的价值观：

- 个人和交流　　胜过　过程和工具
- 可以工作的软件　胜过　面面俱到的文档
- 客户合作　　胜过　合同谈判
- 响应变化　　胜过　遵循计划

这就是所谓的“敏捷宣言”。不难看出，列在右边的恰好是传统软件工程一直关注的主题。敏捷方法的实践者们并非认为这些东西（例如过程、文档……）不重要，而是借出这个宣言提醒世人：在过去相当长的时间里，我们一直把注意力集中在这些“工程化”的东西上面，它们能带给我们的边际效益已经降到了非常低的水平试问，哪位项目经理不会写文档、哪家软件公司不知道如何与客户谈判呢？当此时，如果将更多的精力投入到左边的东西，我们有机会获得更大的收益。透过这份敏捷宣言，你不难读出一种人本主义的精神。早在 1986 年，Fred Brooks 就在《没有银弹》 一文中指出，软件开发的任务分为两方面：根本任务——打造由抽象软件实体构成的复杂概念结构；次要任务——使用编程语言表达这些抽象实体，在空间和时间限制内将它们映射成机器语言。所有工程化的尝试（过程控制、文档管理、新技术和新方法……）无不是局限于解决“次要任务”，而软件开发的“根本任务”——理解真实世界的需求，并将其投射为软件模型——不是任何技术或过程可以解决的。借用 Brooks 的比喻，如果软件的复杂度是可怕的人狼，那么射杀这只怪物的银质子弹中必须有这样一个配方：软件开发者的技艺。归根结底，软件是由人来开发、给人使用的东西，我们无法、也不应当回避人的问题。

拥抱变化，如果你从事着商业软件（例如 ERP 系统、电子政务系统）的开发，我几乎可以肯定地说：你的项目面临着需求变化的挑战。商业规则总在发生着变化，甚至连用户本身在项目之初都不知道自己的需求是什么。软件业的人们常常自嘲地说：“在这个行业里，唯一不变的就是变化。”问题就摆在你面前：既然变化是生活的常态，你是要为自己准备一副盾牌来抵抗它呢，还是要积极地拥抱它？就像汤普金斯先生在《最后期限》 里所说的，我们或许可以换一个角度来思考：作为软件开发者的我们，和用户本应该站在一条战壕里，我们有着共同的

目标——为用户提供最大的价值。惟有拥抱变化，我们的目标才能达成。

如何确保随时为用户提供最大化的价值？我们可以把这个问题想得再极端一点：假如明天就要终止这个项目，如何确保今天能够提交给用户的是最有价值的东西？显然，如果提交给用户一堆需求文档和设计文档，它们对用户毫无价值。答案是快速迭代。一个拥抱变化的团队应该尽快开始生产真正可用的软件系统，尽快为用户交付产品，尽快接收用户的反馈并做出反应。一次迭代的周期应该短至一周甚至几天，并且每天都必须保证可以集成一个可用的版本——这并非天方夜谭，在微软公司的开发实践中你就可以找到“每日构建”这一条目。

除了快速提供可用的产品之外，一个拥抱变化的团队还必须快速获得用户的反馈——不是通过电话或者通过商务部门的中转，也不是通过每周一次的项目例会，敏捷团队需要每天、每时每刻与用户交流，才能随时根据用户的反馈调整自己的作品。一种可行的策略就是现场客户：一位客户代表和开发团队在同一间办公室工作，随时解答开发团队的疑问。如果客户代表的本职工作可以通过网络进行，而且解答一个疑问只需要短短几句话，这种策略或许并不像它听起来那么难以实现。如果确实无法获得现场客户，在团队内部确定一名领域专家扮演现场客户的角色也是一种替代方案。

需求变更总是软件开发中的一个大问题，甚至由此衍生出了名为“变更控制”的学问。但是，如果把决定权交还给客户，由客户来决定每个迭代版本应该包含哪些特性、应该做出哪些修改，开发团队就无须面临艰难的抉择。毕竟，时间和预算都是有限的，只有客户能判断开发哪些东西能够提供最大化的价值。但这就要求开发团队能够提供精确的估算——只有当每次迭代周期都非常短时，你才有可能准确地估算出下一次迭代可以完成多少工作，估算未来三个月的工作量是毫无意义的。

最重要的（也是最根本的）是：一个拥抱变化的团队必须力求简单。在各种可行的方案中，他们应该始终选择最简单的一种；开发过程生成的产品也应该尽量精简——既然可以用单元测试来描述程序模块的功能，就不要再强求详细设计文档。俗话说“船小好调头”，越简洁的设计越容易接纳变化。

面向架构的软件开发软件是一门抽象的艺术，软件的发展史就是人们不断提升抽象层次的历史。进入 21 世纪，面向对象编程语言等技术已经非常成熟，软

件开发的最大问题已经不再是编写程序代码，而是如何建立概念模型使之体现业务需求、如何设计系统结构使之具有灵活性和可维护性。换句话说，软件系统中最重要的部分将是架构（architecture），而不是具体的实现技术和实现代码；描述软件系统最重要的语言将是模式语言（pattern language），而不是具体的编程语言。掌握架构能力和模式语言，将是一个称职的软件开发者必备的技能。然而，架构和模式又不能脱离于具体的语言和实现技术存在——多年开发企业级应用的经验告诉我，如果一个系统号称“完全不依赖任何语言和实现技术”，那么这个系统也不会有任何用处。举一个最简单的例子：实现同样功能的一个软件产品，如果它主要作为类库（library）提供进程内服务，那么它的架构应该完全遵循面向对象原则，通过多个对象的多次协作完成业务操作；而如果需要提供远程调用服务，客户端与服务器端的每次交互就会造成巨大的网络开销，那么就需要按照面向服务的体系结构（Service-Oriented Architecture，SOA）来设计，尽量降低交互的频率。所以，一个称职的软件开发者不仅必须具备架构设计的能力，还必须清楚地知道用什么技术来实现自己的架构，否则他所设计的架构就会成为空中楼阁。

面向架构的软件开发，再加上各种各样极大提升编程效率的工具，不难想象，未来的大多数软件开发将由少数精英来完成——我更愿意把这些人叫做“软件工匠”，因为他们的工作将更多地体现出工艺性甚至是艺术性，更少地体现工程性。实际上，这已经是迫在眉睫的事实了。在过去的十年中，微软、Borland 等世界领先的软件企业一直实行着小型团队、人性化管理、重视个人才华胜于制度的软件开发模式；最近一两年，真正在贯彻着 CMM 之类工程化方法的，在世界范围内也只剩下印度、爱尔兰等国以外包项目为主的软件企业了。

面向架构的软件开发要求高水平的小型开发团队，要求客户与开发团队之间直接而频繁的交流，要求在短迭代中不断验证和修改整个软件系统。因此，面向架构的软件开发必然是敏捷的。惟有敏捷的团队才能获得真正符合用户需要的软件架构和产品——而不是在汗牛充栋的文档中不断地浪费时间。

## 3.8. 实施技术与平台

**1、实施的主要技术**

对于权限管理，做为一个嵌入其它系统中的系统而言，一个很关键的因素是如何和实际的系统接合，通常的做法是以函数库的方式提供，开发者使用本系统时所要做的就是通过调用函数库的方式进行调用。在以面向过程的编程时代这是一个通常的做法。现阶段主流的程序开发技术均是面向对象的开发技术，如（JAVA，C#，C++等），再去采用函数库的方式去做，首先概念上不易理解，容易造成理解偏差，其次技术上实现要进行很多转换，代来不必要的麻烦。

**2、常用的平台**

一种好的办法是采用组件的方式来实现，基于 windows 的程序可以采用 COM 模型来实现，但也有一个问题的跨平台的特性，其它的操作系统不能直接调用，由此造成的问题是系统只能用于 Windows 操作系统。

JAVA EE 是一种面向对象的主流企业级应用架构，它所提供的 JavaBean 和 Enterprise JavaBean（EJB）提供了另一种组件对象模型，同时由于 JAVA 的跨平台特性，在已知的主流操作系统中均可运行，由此为我们提供了一种很好的实现方式。现阶段，大型的企业级应用多采用 JAVA EE 实现，采用 JAVA 做为开发语言，能满足绝大多数用户的要求。

因此我们选择开发平台时选择 JAVA EE 做为开发平台，同时 JAVA 有很多开源的架构来支持软件的开发，如 Spring，Hibernate，JSF，Struct 等架构，做为开发者只需专注于程序逻辑的实现，而无需关注系统的架构，从而大大减少开发的难度。

JAVA 的开发工具众多，既有 SUN 的 NetBean,也有 ECLIPSE 开源组织的 Eclipse，这两个工具均以开源的方式提供，用户视自己的熟悉程度可自由选择，权限系统选用 Eclipse 做为开发工具。

数据库系统主流均是关系型数据库，本系统选用 Oracle 9i。

# 第4章 模型设计

## 4.1. 设计任务

设计任务须指明模型设计所有完成的工作，根据 RBAC 模型，本设计任务需要完成以下工作：

设计约定，设计过程中所要遵守的共同规则

对象设计，为完成 RBAC 模块所要使用的对象

数据库设计，用于持久化对象的存储结构

测试设计，本模块采用 TDD 的开发模式，因此在设计之初要进行测试方案的设计

## 4.2. 设计约定、对象设计

1、设计约定

1）对象命名规则

2）对象表示方法

3）数据库命名规则

4）变量定义规则

5）文档书写规则

6）测试用例编写规则

本模型中涉及到用户、角色、组织结构、权限、权限集、角色用户关系、用户组织结构结构关系、功能模块八种对象。对象之间的关系如下图 3-1 所示：

2、对象设计

现将每种对象及其主要属性描述如下：

1）用户（User）

用户所属属性如下：

- ID ：int 型
- 名称：String

- 帐户：String
- 口令：String
- 性别：String
- 民族：String
- 描述：String
- 备注：String
- 排序：int
- 状态：int

2）角色

角色所属属性如下

- ID ：int 型
- 名称：String
- 描述：String
- 备注：String
- 排序：int
- 状态：int

3）组织结构

组织结构所属属性如下

- ID ：int 型
- 名称：String
- 父部门：Deparment
- 描述：String
- 备注：String
- 排序：int
- 状态：int

4）权限

权限所属属性如下

- ID ：int 型
- 名称：String

- 描述：String
- 备注：String
- 排序：int
- 状态：int

5）权限集

权限集所属属性如下

- ID ：int 型
- 角色：Roler
- 模块：module
- 范围：string
- 描述：string
- 备注：String
- 排序：int
- 状态：int

6）角色用户关系

角色用户关系所属属性如下:

- ID ：int 型
- 角色：Roler
- 用户：User
- 备注：String
- 排序：int
- 状态：int

7）功能模块

功能模块所属属性如下:

- ID ：int 型
- 分类：String
- 名称：String
- 父模块：Module
- 入口地址：String

- 备注：String
- 排序：int
- 状态：int

8）用户组织结构结构关系

用户组织结构结构关系所属属性如下：

- ID ：int 型
- 名称：String
- 用户：User
- 部门：Deparment
- 描述：string
- 备注：String

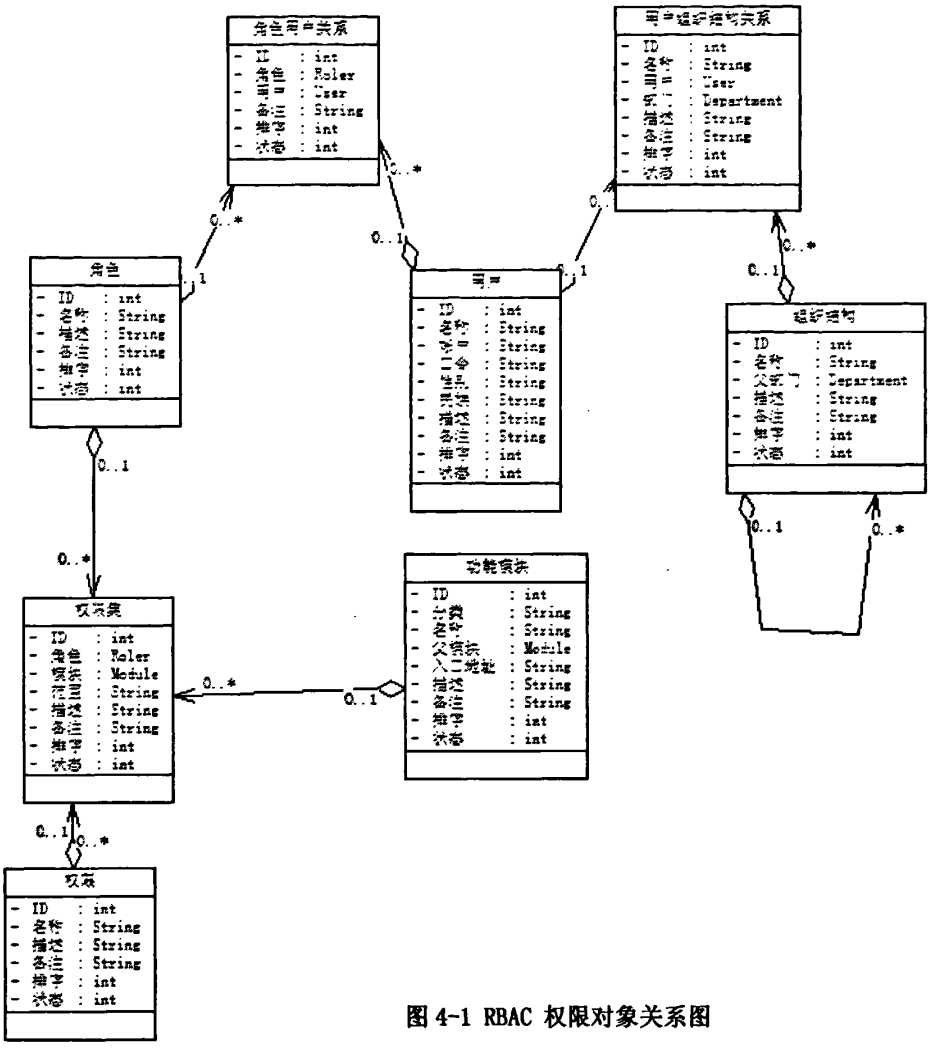



图 4-1 RBAC 权限对象关系图

## 4.3. 数据库设计

虽然 RBAC 模型采用的是面向对象的设计方法，但现在主流的数据库均为关系型数据库，为适应关系型数据库的要求，规范数据设计，将对象在数据库中的关系如图 4-2 所示。

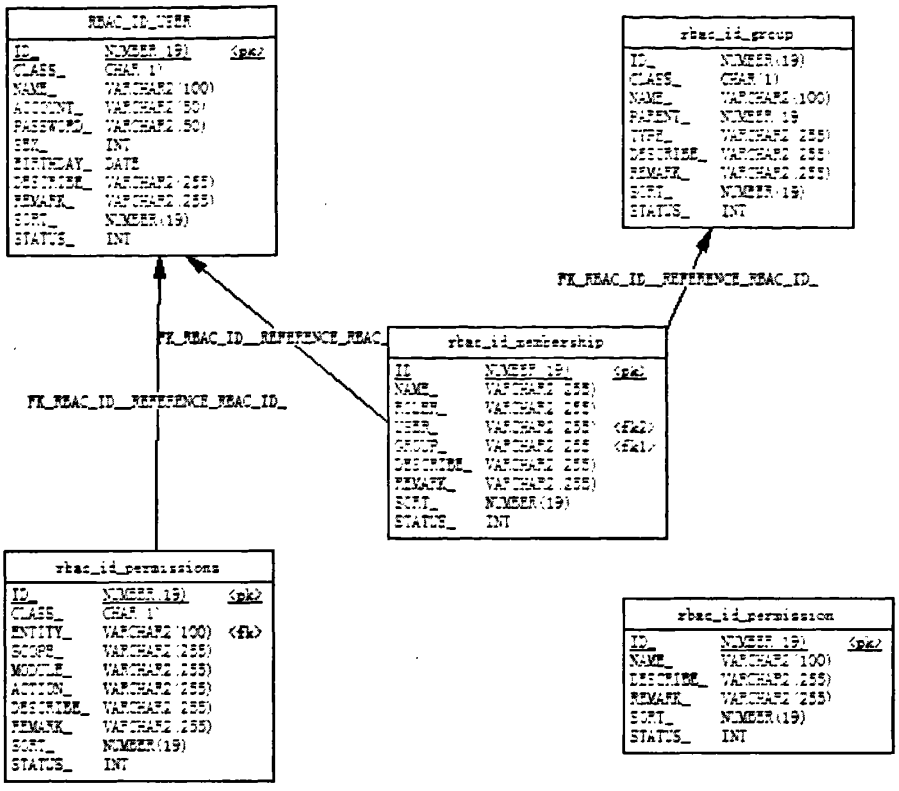



图 4-2 对象在数据库中的关系

### 4.3.1. 用户、角色表(RBAC_ID_USER)

1、用户和角色具有部分相同结构，赋于权限时也要使用同一个 ID，因此将这两个对象置于同一个物理表中，用表中的字段 CLASS_来区分，其中 CLASS_为“U”表示用户，“R”表示角色；

2、在所有属性名之后均加_是为了和系统名称不冲突，又顾名思义；

3、表中的属性说明：

1) ID_代表本条数据的唯一标识，表中的主键，它和对象表中的 ID 一一对应，其长度为 19 位是目前最长的设置，主要为了确保若干年以后的用户及角色扩展；

2）CLASS_用于标识本条数据是角色还是用户，CLASS_为“U”表示用户，“R”表示角色；

3）NAME_用于表示用户及角色的名称；

4）ACCOUNT_本属性用于存储用户的登陆帐户；

5）PASSWORD_本属性用于存储用户登录系统时的口令；

6）SEX_本属性用于记录用户的性别；

7）BIRTHDAY_用于存储用户的的出生日期；

8）DESCRIBE_用于存储对角色或用户的常规说明；

9）REMARK_用于存储对角色或用户的例外说明；

10）SORT_用于对用户或角色进行排序；

11）STATUS_用于标识用户或角色的权限状态；

### 4.3.2.组织结构(RBAC_ID_GROUP)

1．部门和功能模块具有部分相同结构，赋于权限时也要使用同一个 ID，因此将这两个对象置于同一个物理表中，用表中的字段 CLASS_来区分，其中 CLASS_为“D”表示部门，“M”表示模块；

2．在所有属性名之后均加_是为了和系统名称不冲突，又顾名思义；

3．表中的属性说明：

1) ID_代表本条数据的唯一标识，表中的主键，它和对象表中的 ID 一一对应，其长度为 19 位是目前最长的设置，主要为了确保若干年以后的数据扩展；

2) CLASS_用于标识本条数据是模块还是部门，CLASS_为“D”表示部门，“M”表示模块；

3) NAME_用于表示部门及模块的名称；

4) PARENT_用于标识模块的父模块和部门的父部门，以便完成权限的继承；

5) TYPE_用于标识模块的功能及部门的类型；

6) DESCRIBE_用于存储对模块及部门的常规说明；

7) REMARK_用于存储对对模块及部门的例外说明；

8) SORT_用于对对模块及部门进行排序；

9) STATUS_用于标识对模块及部门权限状态；

### 4.3.3.成员关系（RBAC_ID_MEMBERSHIP）

1．本张表主要存储用户、角色、部门之间的关系；

2. 表中的主要属性说明：

1）ID_代表本条数据的唯一标识，表中的主键，它和对象表中的 ID 一一对应，其长度为 19 位是目前最长的设置，主要为了确保若干年以后的数据扩展；

2）NAME_本名称可以为空也可以为角色或用户或部门名称；

3）ROLER_为角色的名称；

4）USER_为用户的名称；

5）GROUP_为部门的名称；

6）DESCRIBE_用于存储对角色、用户关系或部门、用户关系的常规说明；

7）REMARK_用于存储对角色、用户关系或部门、用户关系的例外说明；

8）SORT_用于对记录进行排序；

9）STATUS_用于标识关系的可用状态；

### 4.3.4.权限（RBAC_ID_PERMISSION）

1. 本张表用于存储用户、角色权限；

2. 表中的主要属性说明：

1）ID_代表本条数据的唯一标识，表中的主键，它和对象表中的 ID 一一对应，其长度为 19 位是目前最长的设置，主要为了确保若干年以后的数据扩展；

2）NAME_表示权限的名称；

① DESCRIBE_用于存储对对权限的常规说明；

② REMARK_用于存储对权限的例外说明；

③ SORT_用于对权限进行排序；

④ STATUS_用于标识权限状态；

### 4.3.5.权限集（RBAC_ID_PERMISSIONS）

1. 本张表用于存储用户及角色的权限；

2. 表中的属性说明：

1）ID_代表本条数据的唯一标识，表中的主键，它和对象表中的 ID 一

一对应，其长度为 19 位是目前最长的设置，主要为了确保若干年以后的数据扩展；

2）CLASS_用于标识本权限用于角色还是用户，CLASS_为“R”表示角色，“U”表示用户；

3）ENTITY_用于存储角色或用户的 ID_；

4）SCOPE_用于存储本权限所适用的用户范围；

5）MODULE_用于存储模块的 ID_；

6）ACTION_用于表示角色或用户拥有那些权限；

7）DESCRIBE_用于存储对对权限的常规说明；

8）REMARK_用于存储对权限的例外说明；

9）SORT_用于对权限进行排序；

10） STATUS_用于标识权限状态；

## 4.4. 功能设计

功能设计指明，系统主要完成什么样的功能，按需要约定，需完成以下功能：如图 4-3 所示

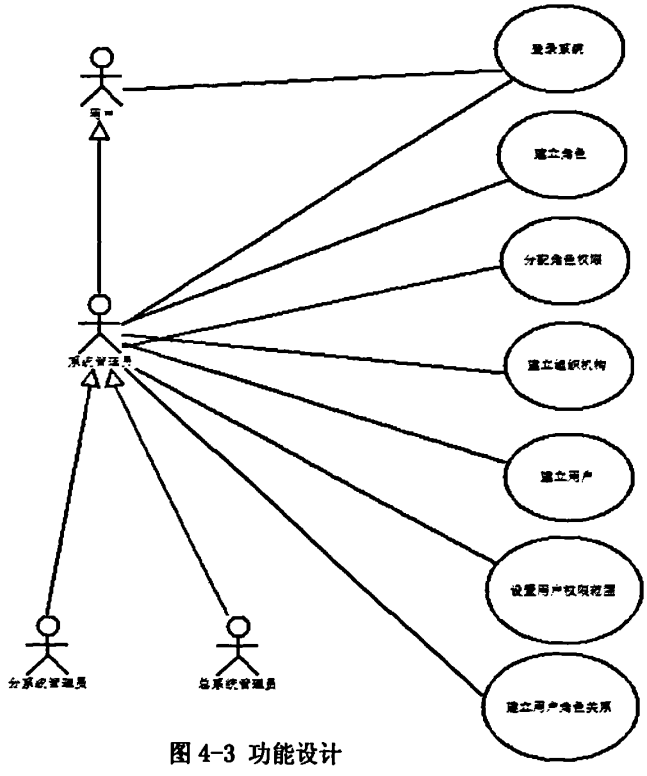



图 4-3 功能设计

### 4.4.1. 登录系统

本模块主要完成系统的登陆工作，主要核对帐号和口令，完成安全、高效的单点登录（Single Sign-On）。

SSO（Single Sign On），单点登录。SSO是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。它包括可以将这次主要的登录映射到其他应用中用于同一个用户的登录的机制。它是目前比较流行的企业业务整合的解决方案之一。

### 4.4.2. 角色管理、部门管理及用户管理

1. 角色管理

本模块主要完成以下功能：

1）角色的新建

2）角色的编辑

3）角色和用户关系的设置

4）角色权限的分配。

2. 部门管理

本模块主要完成以下功能：

1）新建组织结构

2）编辑组织结构

3）设置部门和用户关系

3. 用户管理

本模块主要完成以下功能：

1、用户的新建

2、用户的编辑

3、用户和角色关系的设置

4、用户权限范围的设置

5、用户和部门关系的设置

## 4.5. 测试设计（junit）

基于 TDD 的开发原则要求先进行测试案例的设计，在测试案例的基础上再进行编码，本系统首先采用 JAVA EE 平台实现，使用基于 JAVA EE 的测试框架 junit;

**1. 登录系统测试**

分别输入已存在的帐号、口令用户，和不存在的帐号及口令的用户对登陆系统进行测试。

**2. 角色管理测试**

分别对角色管理的角色新建、编辑、用户和角色关系的设置、角色权限的设置进行测试。

**3. 部门管理测试**

分别对部门管理的部门的建立、编辑及部门和用户关系设置等模块进行测试。

**4. 用户管理测试**

分别对用户管理的用户新建、编辑、用户和角色关系的设置、用户权限范围设置、用户和部门的关系设置等模块进行测试。

# 第5章 模型实施

## 5.1. 实施任务及表现形式

1. 实施任务

实施是根据设计要求将系统在某一平台下，用代码的形式实现，能以界面的方式展示供用户使用。实施应完成以下任务：

界面设计、数据库生成，对象生成，对象－关系数据库映射，编码技术的考虑，模式的采用，编码实现，详细设计说明，测试案例编写等工作。详述如下：

2. 表现形式

表现形式说明用户与系统进行交互时以什么样的方式进行展示，尽管本系统未设定仅适用于以 WEB 方式展示的系统，对于基于富客户端的系统，甚至既有 WEB 方式，又有富客户端的系统同样适用，为说明问题期间，仅以 WEB 方式的展示。要求用户在 IE，或 FIREFOX 或其它的浏览器中输入网址，即可展示系统。

## 5.2. 对象－数据库映射

现阶段，主流的编程技术均是基于面向对象的编程方法的，而现阶段主流的数据库主要是关系型数据库，面向对象中的对象和关系型数据库中的实体，无论从哪个角度均存在着巨大的差异，为使用两种主流技术需要将对象和数据库进行转换，从而在编程的环境中不去思考二者语义的差别，专心于系统主要的逻辑关系的处理。

处理对象－关系映射的方案很多，就性能和使用的平台而言，采用 Hibernate 是一个不错的选择。

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，使得 Java 程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序实用，也可以在 Servlet/JSP 的 Web 应用中使用，最具革命意义的是，Hibernate 可以在应用 EJB 的 J2EE 架构中取代 CMP，完成数据持久化的重任。

## 5.3. 对象复用考虑

对象复用主要是为了减少编程工作量，提高代码效率，增加代码灵活性而使用前人使用的总结，并证明行之有效的经验进行编程，现阶段设计模式已总结出了很多，编程过程中就尽量采用。

设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。

毫无疑问，设计模式于己于他人于系统都是多赢的，设计模式使代码编制真正工程化，设计模式是软件工程的基石，如同大厦的一块块砖石一样。

GOF 的“设计模式”是第一次将设计模式提升到理论高度，并将之规范化为编码约定及对象编码约定。

**1．编码约定**

编码规范对于程序员而言尤为重要，有以下几个原因：

1) 一个软件的生命周期中，80%的花费在于维护；

2) 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护；

3) 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码；

4) 如果你将源码作为产品发布，就需要确任它是否被很好的打包并且清晰无误；一如你已构建的其它任何产品；

5) 为了执行规范，每个软件开发人员必须一致遵守编码规范。每个人。

**2．对象编码约定**

① 文件名　　②文件后缀　　③常用文件名

**3．文件组织**

① Java 源文件　　②开头注释　　③包和引入语句　　④ 类和接口声明

**4．缩进排版**

① 行长度　　②换行

**5．注释**

①实现注释的格式　　②块注释　　③单行注释

④尾端注释　　⑤行末注释　　⑥文挡注释

6. 声明

①每行声明变量的数量　　②初始化

③布局　　④类和接口的声明

7. 语句

①简单语句　②复合语句 ③返回语句 ④if，if-else，if else-if else 语句

⑤for 语句　⑥while 语句　⑦do-while 语句　⑧switch 语句

⑨try-catch 语句　⑩ 空白、 空行、 空格

8. 命名规范

9. 编程惯例

①提供对实例以及类变量的访问控制　　②引用类变量和类方法

③常量　④变量赋值　⑤其它惯例　⑥圆括号　⑦返回值

⑧条件运算符"?"前的表达式"?"前的表达式　⑨特殊注释

### 5.3.1. 数据库编码约定

1. 为了件开发的设计过程中关于数据库设计时的命名规范和具体工作时的编程规范，便于交流和维护，特制定此规范

2. 有条件可以使用 Sybase 公司的设计工具 Power Designer 做为数据库的设计工具，要求为主要字段做详尽说明:

a)对于 SQL Server 尽量使用企业管理器对数据库进行设计，并且要求对表，字段编写详细的说明（这些将作为扩展属性存入 SQL Server 中）

b) 有条件可以通过 Power Designer 定制 word 格式报表，并导出 word 文档，作为数据字典保存，格式可参看附件 1。（PowerDesigner v10 才具有定制导出 word 格式报表的功能）。对于 SQL Server 一旦在企业管理器进行数据库设计时加入扩展属性，就可以通过编写简单的工具将数据字典导出。

c) 编写数据库建数据库、建数据库对象、初始化数据脚本文件

### 5.3.2. 设计原则

1. 采用多数据文件

2. 禁止使用过大的数据文件；unix 系统不大于 2GB,window 系统不超过 500MB

3. oracle 数据库中必须将索引建立在索引表空间里。

4. 基本信息表在建立时就分配足够的存储空间，禁止其自动扩展功能

5. 大文本字列、blob 列要独立出一张表，此表只有 id 和 blob(或大文本)列

6. 为每一个数据库创建独立的管理员用户，使用该用户进行设计，尽量不要使用 sa 或者系统管理员身份进行数据库设计。

### 5.3.3.设计的更新

1. 在设计阶段，由数据库管理员或指定的项目组其一成员进行维护。

2. 运行阶段，由数据库管理员进行维护。

3. 如对表结构进行修改，应先在数据字典文档进行修改，最后在数据库中进行修改。如果修改的是数据库字典表，必须由数据库管理员进行。

4. 编写更新的 SQL 代码，如果使用 PowerDesigner，禁止由 PowerDesigner 直接连数据库进行数据库操作(如果是更改表或者字段的说明性文字可以通过数据库管理器图形界面进行修改)

5. 修改数据库要通过 SQL，禁止其它方式对数据进行修改

6. 修改数据库的 SQL 要添加说明后保存备查

### 5.3.4.命名总体原则及规范

1. 命名总体原则

1）设定的前缀一律用小写字母；

2）标识名称命名全部小写；

3）整个命名的全长不得超过 30 个字母；

4） 全部使用字母和下划线'_'，不能使用中文和其他字符，有特别情况允许使用末尾数字编号。例如：t_Finace1, t_Finace2... ；

5）命名名称来自于业务，全部采用英文单词；

6）英文单词过长可以采用通用的缩写，尽量表达出业务的含义 ；

7）如需要两个以上的英文单词做标识名称，单词之间要用下划线'_'连接 ；

8）名称全是由名词组成的，名词由大范围到小范围排序取名 ；

9）完成某功能的名称，如函数和过程，以动宾形式取名；

2．命名规范（逻辑对象）

1）数据库结构命名

① 数据库命名 数据库的命名要求使用与数据库意义相关联的英文字母，即<业务系统名称>。例如：china care 数据库的命名为 ccnet； 客户资料数据库的命名为 Customer_Info。

② 数据库日志设计命名

数据库日志的的命名以<数据库名>_<日志名>.log 格式命名。

例如： ccnet_logredo.log

③ 数据库配置设计命名

数据库配置设计方案是以文件形式保存的，其内容是关于特定数据库的配置项目的具体值。

数据库配置文件的命名以:<数据库类型简写〉_<应用系统标志>_cfg.ini 格式命名。 其中，数据库类型简写见附件 2《数据库类型简写》，cfq 表示该文件是数据库配置文件。

例如：ORA_ ccnet_cfg.ini

④ 数据库复制与存储设计命名

数据库复制与存储设计方案是以文件形式保存的，其内容是关于特定数据库之间的复制策略的具体细节。

数据库复制与存储设计文件的命名以：<数据库类型简写>_<应用系统标志>_rep.txt 格式命名。rep 表示该文件是数据库复制与存储文件。

⑤ 数据库连接设计命名

数据库连接设计方案是以文件形式保存的，其内容是关于特定分布式数据库之间的连接设计的具体细节。

数据库连接设计文件的命名以：<数据库类型简写>_<应用系统标志>_dbl.sql 格式命名。dbl 表示该文件是数据库连接设计文件。

⑥ 表空间、数据文件命名(主要针对 Oracle)

表空间命名格式：ts<系统标识>_i

数据库文件命名格式：ts<系统标识>_i[n].dbf

临时表空间命名格式：ts<系统标识>_t

数据库文件命名格式：ts<系统标识>_t[n].dbf

回滚表空间命名格式：ts<系统标识>_r

数据库文件命名格式：ts<系统标识>_r[n].dbf

数据表空间命名格式：ts<系统标识>_d

数据库文件命名格式：ts<系统标识>_d[n].dbf

注：表空间名不超过 8 位，n 可取 00-99 或 0-9，根据系统数据量确定。

2） 数据库对象命名

① 表： 表的命名必须以“t_”（Table 缩写）开头，

格式为：t_[系统标识]_<数据表类型标识>_<表标识>。

其中，[ ]表示可选项，依据实际情况而增加；<表标识> 要求与表意义相关联的英文字母，例如：t_Customers。

数据表大致分为：业务数据表、基本编码表、辅助编码表、系统信息表、累计数据表、结算数据表、决策数据表；

基本编码表用 base 标志

累计数据表用 count 标志

系统信息表用 info 标志 ...

例如：t_trade_base_trade_code,t_trade_info_help ...

② 字段/域：根据业务要求进行命名，不需设定固定的前缀。

③ 索引： 针对数据库表中一个或多个字段建立的索引的命名格式应以“idx_”开头，索引列名间用_隔开，即为 idx_ColumnName1_ColumnName2_...

其中，ColumnName1 是数据库表中（第一个）索引字段的名称或名称简写；ColumnName2 是数据库表中（第二个）索引字段的名称或名称简写；索引名的总长必需符合数据库的规定。

例：idx_cert_number (表示在字段 cert_number 上创建索引)

为了避免重名索引出现，可选命名方式为 idx_<表名>_<递增号> 作为索引的命名，但是要求在数据字典中进行详细说明

④ 视图：视图的命名必须以“v_”(View 缩写)开头，格式为：

v_<视图类型>_[系统标识] _<视图标识>。

其中，视图类型参见“表的分类说明”；[系统标识_]是可选项，依据情况而增加；<视图标识> 应与视图意义相关联的英文字母。

例：v_user_detail_info

⑤存储过程： 存储过程的命名必须符合 sp_[系统标识]_<存储过程标识>格式。

其中，sp 表示是存储过程；[系统标识]为可选项，依据情况而增加；<存储过程标识>是与存储过程意义相关联的英文字母。

例如：USP_Query_Write_to_Disk。

⑥ 触发器：触发器的的命名必须符合 tr_<表名>_<i,u,d 的任意组合> 格式。其中,tr 表示是触发器；<i,u,d 的任意组合>是与触发器意义相关联的英文字母。

例：tr_user_info_iu （表示对 user_info 表进行插入、更新的触发器）

⑦ 函数：函数的命名必须符合 fn_[系统标识]_<函数标识>格式。

其中，fn 表示是函数，[系统标识]为可选项，依情况而定；<函数标识>是与函数意义相关联的英文字母。

例：fn_create_id（以动宾方式取名）

⑧ 自定义数据类型：自定义数据类型的命名格式为：ud_<自定义数据类型标识>_<数据类型>

⑨ Default（缺省）：Default（缺省）的命名格式一般为：df_<Default 标识>；对于非绑定的默认可取系统默认的名字。

例：df_begin_date 缺省开始日期 ‘20030101’

```
if exists (select * from sysobjects where type = 'D' and name = 'df_begin_date')
drop default dbo.df_begin_date go
create default df_begin_date as '20030101'
go
```

⑩Check、Constraint（约束）：约束的命名格式一般为：ck_<表名>_ <Check 标识>；一些约束可直接放在生成表的语句中。

⑪ Rule（规则）：规则的命名格式一般为：rl_<Rule 标识>；对于非绑定规则(约束) 可取系统默认的名字。

例：rl_not_zero （定义一个不等于 0 的规则）

```
if exists (select * from sysobjects where type = 'R' and name = 'rl_not_zero')
drop rule dbo.rl_not_zero
go
create rule rl_not_zero as @i <> 0
go
```

⑫ 主键：主键的命名格式为 pk_<表名>_<主键标识>。

例：pk_user_info_userid （表 user_info 以字段 userid 创建主键）

⑬外键：外键的命名格式为 fk_<表名>_<主表名>_<外键标识>。

可选命名方式为 fk_<表名>_<递增号> 作为索引的命名，但是要求在数据字典中进行详细说明

例：fk_user_info_department_deptid （在表 user_info 的字段 department_id 上创建外键，参照主表 department）

⑭同义词（ORACLE）：同义词的命名格式为：sy_<同义词标识>

例：sy_user_info （为所有权属于 ben 的表 user_info 的公共同义词）

### 5.3.5.脚本注释

1．存储过程或触发器

1）每一个存储过程或触发器都要在最前面写注释，注释如下:

```
/*  writer:
    create date:
    ver:
    Depiction:
 remark:  */
```

另外，过程中声明的重要变量要有注释，例如：

```
@iActionFlag int = 0 /* 0 => Checkout, 1 => GetLatest, 2 => UndoCheckOut */
```

2） 如果只对存储过程或触发器进行部分修改时须添加以下注释：

修改描述：

*/ /* 原代码内容*/（修改时）

/*rewriter: date：<格式：YYYY-MM-DD> end1： */

/*rewriter: add（rewriter）: date：<格式：YYYY-MM-DD> start2： */

新代码内容

/*rewriter: date：<格式：YYYY-MM-DD> end2： */

3） 如果对存储过程或触发器有较大的修改，可增加修改内容的注释。

/*Log Id：<Log 编号,从 1 开始一次增加>

rewriter：

rewrite date：<格式：YYYY-MM-DD>

Depiction： */

2. 自定义函数

1） 每一个自定义函数都要在其前面写注释，注释如下

/* function name:xxxx

Depiction：<对此函数的描述>

param（a,b）

功能或描述....

output: x x=0 表示..... x=1 表示.....

writer：

create date：<创建日期，格式：YYYY-MM-DD>

ver：

remark： * /

另外，函数中声明的重要变量要有注释，例如：

@iActionFlag int = 0 /* 0 => Checkout, 1 => GetLatest, 2 => UndoCheckOut

*/

2） 如果只对函数进行部分修改时须添加以下注释：

/*rewriter: add（rewriter）: date：<格式：YYYY-MM-DD> start1：

修改描述：

*/ /* 原代码内容*/（修改时）

/*rewriter: date：<格式：YYYY-MM-DD> end1： */

/*rewriter： add（rewriter）： date：<格式：YYYY-MM-DD> start2： */

新代码内容

/*rewriter： date：<格式：YYYY-MM-DD> end2： */

3） 如果对函数有较大的修改，可增加修改内容的注释。

/*Log id：<Log 编号,从 1 开始一次增加>

rewrite date：<修改日期，格式：YYYY-MM-DD> Depiction： */

3、数据库操作规则

1）建立、删除、修改库表操作

在开发环境中，对于自己的库表可任意进行修改、删除操作；但需保留相应的建表语句和说明，与建表人建表时间。

2）添加、删除、修改表数据

在开发环境中，开发人员所开发模块独自使用的库表，可自由操作表中数据；

对其他模块关联的库表，应取得其他模块的开发人员同意后再执行操作；系统的信息表、字典表的修改应向数据库的管理员提出操作需求，由数据库的管理员执行操作。

## 5.4. 测试案例编写

测试用例，是一份关于具体测试步骤的文档，它描述了测试的输入参数、条件及配置、预期的输出结果等，以判断被测软件的工作是否正常。设计、书写和执行测试案例是测试活动中重要的组成部分，测试案例通常由测试案例管理系统或工具进行管理。

测试用例，是一份关于具体测试步骤的文档，它描述了测试的输入参数、条件及配置、预期的输出结果等，以判断被测软件的工作是否正常。设计、书写和执行测试案例是测试活动中重要的组成部分，测试案例通常由测试案例管理系统或工具进行管理。

### 5.4.1. 编写测试用例的原则

测试用例的重要性是毋庸置疑的，它是软件测试全部过程的核心，是测试执

行环节的基本依据。测试用例编写应该遵循的原则：

1. 测试用例要达到最大覆盖软件系统的功能点。测试工程师应该测试计划编写完成之后，在开发阶段编写测试用例，参考需求规格说明书和软件功能点对每个功能点进行操作上的细化，尽可能趋向最大需求覆盖率。

2. 测试用例对测试功能点、测试条件、测试步骤、输入值和预期结果应该有准确的定义。

3. 测试用例的设计应包括各种类型的测试用例。在设计测试用例的时候，除了满足系统基本功能需求外，还应该考虑各种异常情况、边界情况和承受压力的能力等。

4. 测试用例的管理。使用测试用例管理系统对测试用例进行管理。

一个好的测试用例应该具有较高的发现某个尚未发现的错误的可能性，而一个成功的测试案例能够发现某个尚未发现的错误，通常一个好的测试案例有以下特性：

1) 具有高的发现错误的概率

2) 没有冗余测试和冗余的步骤

3) 测试是“最佳类别”

4) 既不太简单也不太复杂

5) 案例是可重用和易于跟踪的.

6) 确保系统能够满足功能需求

测试用例不可能设计得天衣无缝，也不可能完全满足软件需求的覆盖率，测试执行过程里肯定会发现有些测试路径或数据在用例里没有体现，那么事后该将其补充到用例库里，以方便他人和后续版本的测试。

### 5.4.2. 测试用例设计过程

对一个全新的产品来说，首先需要了解的是产品需求文档和产品模块之间的关系。然后需要从需求文档中书写与所有需求相对应的主路径测试案例和烟雾测试案例，这个时候也同时会包括一定的基本路径测试案例甚至是详细测试案例。在这个时候，因为对产品没有直接的使用感受，书写测试案例要考虑面广而不要太过精细。继续阅读产品功能定义文档，将所有的功能定义直接对应写相关的测

试案例，这个时候，最好能够对程序的本身有一定的接触，加深对程序的了解，以便写出更好，更全面的测试案例。最后，在实际测试中，还需要不断扩充，修改以前的测试案例，得到完整的基本功能测试案例和详细测试案例。

如果对于一个已有一定或大部分案例的产品来说，不管测试者是否本身熟悉这个产品，其主要的任务就是阅读，检查需求及相关的变更，然后对原有的案例进行理解，扩充和修改。这就是案例的重用/复用。

## 5.5. 详细设计

详细设计说明每一个系统功能如何实现功能，以下便是本系统的详细设计：

### 5.5.1. 程序结构

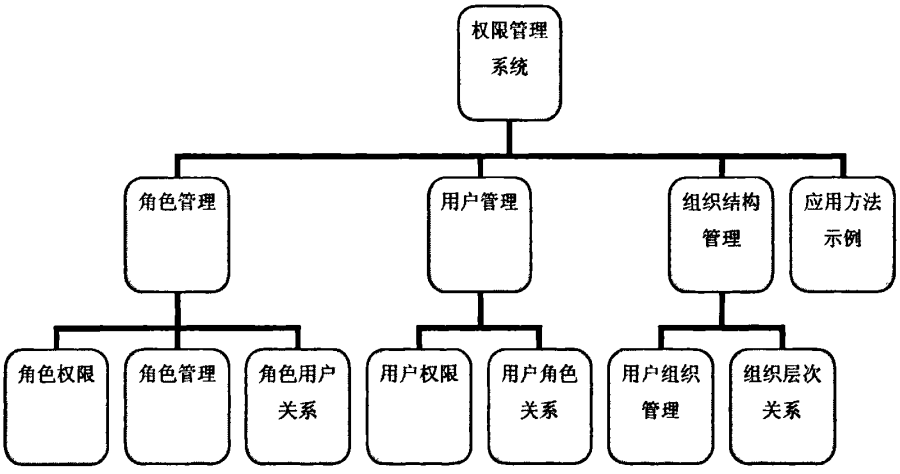



**图 5-1 程序结构**

权限管理系统由四大部分组织，分别是角色管理，用户管理，组织结构管理，为方便用户使用，我们又建立了一个模块，一方面用于调试系统，另一方面用于示例权限系统在系统中如何应用。

角色管理，角色管理主要管理角色，完成新增，删除，修改，查询；角色用户关系，管理角色下的用户的角色；角色权限，管理每一个角色的权限，其内容包括变更，取消，增加等。

用户管理，用户管理主要包括用户的新建，删除，修改，查询，以及用户应归属于某一组织结构下；用户角色关系确定用户归属于某一角色，以及当用户属

于多个角色时，用户所具有的权限集合；用户权限，管理用户特有权限。

组织结构管理，包括组织结构的建立，删除，修改，查询，因组织结构有一定的层次结构，还应管理组织结构之间的层次管理。

应用方法示例，包括程序如何组织，界面如何显示，模块如何调用，权限如何分配与读取，如何进行数据存取等。如下所示的界面为例予以说明。

系统界面分为四大部分，第一部分为 LOGO 区，位于系统顶部，用于显示系统的 LOGO，第二部分为功能区，位于系统中左部，用于列出根据权限列出系统的功能列表，该部分可根据需要，缩进至左部，以便有更多屏幕用于功能操作；第三部分为系统的内容操作区，位于系统的中右部，系统的主要操作均在此完成，根据需要，此区还可分为上下结构，上部用于显示信息列表，下部用于显示详细信息；第四部分为页脚，列出系统有关内容，如研制单位等。

### 5.5.2. 主要类设计

**1. 用户（User.java）**

```
package net.jlobo.db.entity
// default package
import java.util.Date;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;
/**  User generated by MyEclipse Persistence Tools  */
public class User  implements java.io.Serializable {
 // Fields
      private Long id;
      private String name;
      private String localName;
      private String account;
      private String password;
      private Date birthDay;
      private String nation;
      private String place;
      private String telecall;
      private String mobile;
      private String address;
      private String sex;
      private String email;
      private String describe;
```

```
    private String remark;
    private Long sort;
    private String status;
    private Set memberships = new HashSet(0);
    private Set raiseQuestions=new HashSet(0);
    private Set raiseExaminations=new HashSet(0);
    private Set papers=new HashSet(0);
    private String groupName;
    private boolean select;
   // Constructors
   private Long groupId;
   private String lastModule;
   public Long getGroupId()
        {return groupId; }
   public void setGroupId(Long groupId)
        {this.groupId = groupId; }
   public String getGroupName()
        {return groupName; }
   public void setGroupName(String groupName)
        {this.groupName = groupName; }
   public Set getRaiseExaminations()
        {return raiseExaminations; }
   public void setRaiseExaminations(Set raiseExaminations)
        {this.raiseExaminations = raiseExaminations; }
   public Set getRaiseQuestions()
        {return raiseQuestions; }
   public void setRaiseQuestions(Set raiseQuestions)
        {this.raiseQuestions = raiseQuestions; }
 /** default constructor */
    public User() { }
 /** full constructor */
    public User(String name, String localName, String account, String password, Date birthDay,
String nation, String place, String telecall, String mobile, String address, String sex, String
email,String describe, String remark, Long sort, String status/*, Set memberships*/) {
        this.name = name;
        this.localName = localName;
        this.account = account;
        this.password = password;
        this.birthDay = birthDay;
        this.nation = nation;
        this.place = place;
        this.telecall = telecall;
        this.mobile = mobile;
        this.address = address;
```

```
        this.sex = sex;
        this.email = email;
        this.describe = describe;
        this.remark = remark;
        this.sort = sort;
        this.status = status;
//      this.memberships = memberships; }
// Property accessors
    public Long getId()
        {return this.id; }
    public void setId(Long id)
       { this.id = id; }
    public String getName()
        {return this.name; }
    public void setName(String name)
        {this.name = name; }
    public String getLocalName()
       { return this.localName; }
    public void setLocalName(String localName)
        {this.localName = localName; }
    public String getAccount()
         {return this.account; }
    public void setAccount(String account)
        {this.account = account; }
    public String getPassword()
        {return this.password; }
    public void setPassword(String password)
        {this.password = password; }
    public Date getBirthDay()
        {return this.birthDay; }
    public void setBirthDay(Date birthDay)
        {this.birthDay = birthDay; }
    public String getNation()
       { return this.nation; }
    public void setNation(String nation)
      {   this.nation = nation; }
    public String getPlace()
       { return this.place; }
    public void setPlace(String place)
       { this.place = place; }
    public String getTelecall()
        { return this.telecall; }
    public void setTelecall(String telecall)
         {this.telecall = telecall; }
```

```
    public String getMobile()
        {return this.mobile; }
    public void setMobile(String mobile)
       { this.mobile = mobile; }
    public String getAddress()
        {return this.address; }
    public void setAddress(String address)
       { this.address = address; }
    public String getSex()
        {return this.sex; }
    public void setSex(String sex)
      {   this.sex = sex; }
    public String getDescribe()
       { return this.describe; }
    public void setDescribe(String describe)
        {this.describe = describe; }
    public String getRemark()
       { return this.remark; }
    public void setRemark(String remark)
        {this.remark = remark; }
    public Long getSort()
        {return this.sort; }
    public void setSort(Long sort)
         {this.sort = sort; }
    public String getStatus()
        {return this.status; }
    public void setStatus(String status)
        {this.status = status; }
//  public Set getMemberships()
//        { return this.memberships; }
//  public void setMemberships(Set memberships)
//        {this.memberships = memberships; }
    public String getEmail()
        {return email; }
    public void setEmail(String email)
        {this.email = email; }
    public Set getMemberships()
        {return memberships; }

    public void setMemberships(Set memberships)
        {Iterator it=memberships.iterator();
        while(it.hasNext())
            {Membership ms=(Membership)it.next();
            this.groupName=ms.getGroup().getLocalName();
```

```
            this.groupId=ms.getGroup().getId();
            break; }
            this.memberships = memberships; }
    public Set getPapers()
        {return papers; }
    public void setPapers(Set papers)
        {this.papers = papers; }
    public boolean isSelect()
        {return select; }
    public void setSelect(boolean select)
        {this.select = select; }
    public String getLastModule()
        {return lastModule; }
    public void setLastModule(String lastModule)
        {this.lastModule = lastModule; }}
```

2. 角色

```
package net.jlobo.db.entity;
// default package
import java.util.HashSet;
import java.util.Set;
/**   Group generated by MyEclipse Persistence Tools   */
public class Group   implements java.io.Serializable {
    // Fields
     private Long id;
     private Group parent;
     private String _class;
     private String name;
     private String localName;
     private String type;
     private String describe;
     private String remark;
     private Long sort;
     private String status;
     private Set groups = new HashSet(0);
     private Set memberships = new HashSet(0);
  // Constructors
  /** default constructor */
    public Group() { }
    public Group(String id)
    { this.id=new Long(id); }
 /** full constructor */
    public Group(Group parent, String _class, String name, String localName, String type, String
describe, String remark, Long sort, String status, Set groups/*, Set memberships*/) {
```

```
        this.parent = parent;
        this._class = _class;
        this.name = name;
        this.localName = localName;
        this.type = type;
        this.describe = describe;
        this.remark = remark;
        this.sort = sort;
        this.status = status;
        this.groups = groups;
//      this.memberships = memberships; }
// Property accessors
    public Group(Long long1)
        {this.id=long1; }
    public Long getId()
        {return this.id; }
    public void setId(Long id)
        {this.id = id; }
    public Group getParent()
       { return this.parent; }
    public void setParent(Group parent)
         {this.parent = parent; }
    public String get_class()
        {return this._class; }
    public void set_class(String _class)
        {this._class = _class; }
    public String getName()
         {return this.name; }
    public void setName(String name)
        {this.name = name; }
    public String getLocalName()
       { return this.localName; }
    public void setLocalName(String localName)
        {this.localName = localName; }
    public String getType()
       { return this.type; }
    public void setType(String type)
        {this.type = type; }
    public String getDescribe()
        {return this.describe; }
    public void setDescribe(String describe)
        {this.describe = describe; }
    public String getRemark()
        {return this.remark; }
```

```
    public void setRemark(String remark)
        {this.remark = remark; }
    public Long getSort()
       { return this.sort; }
    public void setSort(Long sort)
        {this.sort = sort; }
    public String getStatus()
        {return this.status; }
    public void setStatus(String status)
        {this.status = status; }
    public Set getGroups()
        {return this.groups; }
    public void setGroups(Set jotIdGroups)
        {this.groups = jotIdGroups; }
    public Set getMemberships()
        {return memberships; }
    public void setMemberships(Set memberships)
        {this.memberships = memberships; }
//  public Set getMemberships() {
//          return this.memberships; }
//    public void setMemberships(Set memberships) {
//          this.memberships = memberships; }}
```

3. 组织结构

```
package net.jlobo.db.entity;
// default package
import java.util.HashSet;
import java.util.Set;
/**  * Group generated by MyEclipse Persistence Tools      */
public class Group   implements java.io.Serializable {
    // Fields
     private Long id;
     private Group parent;
     private String _class;
     private String name;
     private String localName;
     private String type;
     private String describe;
     private String remark;
     private Long sort;
     private String status;
     private Set groups = new HashSet(0);
     private Set memberships = new HashSet(0);
    // Constructors
```

```
    /** default constructor */
    public Group() {}
    public Group(String id)
    { this.id=new Long(id); }
    /** full constructor */
    public Group(Group parent, String _class, String name, String localName, String type, String
        describe, String remark, Long sort, String status, Set groups/*, Set memberships*/)
       {this.parent = parent;
        this._class = _class;
        this.name = name;
        this.localName = localName;
        this.type = type;
        this.describe = describe;
        this.remark = remark;
        this.sort = sort;
        this.status = status;
        this.groups = groups;
//      this.memberships = memberships; }
 // Property accessors
    public Group(Long long1)
        {this.id=long1; }
    public Long getId()
        {return this.id; }
    public void setId(Long id)
        {this.id = id; }
    public Group getParent()
       { return this.parent; }
    public void setParent(Group parent)
         {this.parent = parent; }
    public String get_class()
       { return this._class; }
    public void set_class(String _class)
        {this._class = _class; }
    public String getName()
       { return this.name; }
    public void setName(String name)
        {this.name = name; }
    public String getLocalName()
        {return this.localName; }
    public void setLocalName(String localName)
        { this.localName = localName; }
    public String getType()
        {return this.type; }
    public void setType(String type)
```

```
        { this.type = type; }
    public String getDescribe()
        {return this.describe; }
    public void setDescribe(String describe)
        {this.describe = describe; }
        public String getRemark()
       { return this.remark; }
    public void setRemark(String remark)
        {this.remark = remark; }
    public Long getSort()
        { return this.sort; }
         public void setSort(Long sort)
        {this.sort = sort; }
      public String getStatus()
          {return this.status; }
       public void setStatus(String status)
       { this.status = status; }
       public Set getGroups()
       { return this.groups; }
    public void setGroups(Set jotIdGroups)
       { this.groups = jotIdGroups; }
        public Set getMemberships()
        {return memberships; }
    public void setMemberships(Set memberships)
        {this.memberships = memberships; }
//    public Set getMemberships()
//        {return this.memberships; }
//    public void setMemberships(Set memberships)
//        { this.memberships = memberships; }}
```

# 第6章 系统应用举例

随着我国电子政务建设的深入发展，网络基础设施平台已经初具规模，以电子政务应用为建设目标已经逐步成为各党政机关信息化部门的工作重心。支撑安全电子政务应用的信息安全基础平台的建设已经迫在眉睫。这就需要建立以身份认证、授权管理、责任认定等为主要内容的网络信任体系。

为体现RBAC 系统的典型应用和实用价值，本章将国税部门使用的在线考试学习系统的实际应用环境中，我们设计实现的具有RBAC访问控制功能的应用实例。

WEB作为目前最流行的网络信息表现方式，无论是个人还是企业都在广泛使用。对于一个公司企业或政府组织来说，信息在网络中的安全可靠使用，已成为重中之重。RBAC 系统作为构建信息安全访问控制的基础性平台，当然就能够在WEB 信息浏览中一显身手。

由于本部分在前面相关章节中有所详述，这里就只是简要介绍一下RBAC在线考试系统中的应用设计和相关部署。

在线考试系统是在一个全网的、统一身份认证和授权管理的安全基础设施平台，以实现对资源分级授权安全访问控制。让这个平台能提供一个标准的、扩充性良好的，能够灵活、方便地支撑试题中心、出卷中心、答卷中心、阅卷中心、成绩中心，统计中心等应用业务系统的安全应用。

针对在线考试系统的安全要求，我们采用了RBAC 的应用模式，利用RBAC进行访问控制。按照国税系统的目标，要求用户在不同时间段，不同的部门之间，随时随地可进行考试、阅卷、出题、出卷、组织考试和成绩查询的要求，这套在线考试系统采用B/S模式，并建立在认证和授权这个安全基础设施上，由于是基础设施，所以必须先搭建起来，然后在此基础上构建应用系统。

## 6.1 相关设计

1.需求分析

国税系统目前的这套在线考试系统系统采用JAVA开发平台，后台连接Oracle 9i数据库，利用JDBC 进行数据访问，采用JSF+SPRING+HIBERNATE的开发架构。完成的工作基本可以抽象成固定化的工作流系统：从试卷的录入、出卷、组织考试、参加答卷、阅卷、成绩查询到最后成绩统计，在应用界面上能够进行相应的操作。

为了尽快设计出符合在线考试要求的设计方案，我们将重点放在了对数据库数据的访问控制上。因为第一、数据库内容才是访问控制的对象，第二、很多对访问控制的需求实际上可以转化成对数据库数据的访问控制。将用户所能访问的界面，操作均以资源的形式进行管理。如此对资源的访问就变成了数据库记录，而对数据库的操作通过对象关系转换则变成了对对象的访问控制，易于理解，便于实施。

2.设计原则

•统一规划、分步实施

•标准化：遵循开放、标准的原则

•管理方便灵活

•扩充性良好：有效保护用户投资，扩充升级性好，实现最优性价比

3.功能划分

鉴于以上分析我们特把系统分为两大部分：身份认证和授权管理。

•利用标准SSO技术，构建全网统一身份认证管理。

•基于RBAC技术，实现有效的权限集中管理和资源自主授权访问控制。

•权限管理与访问控制系统，将各个应用系统中的多套权限管理与访问控制机制(不同用户对不同资源具有不同访问权限)剥离出来，形成统一的用户库、权限库、资源库，根据总体策略规范，结合具体应用进行授权并实施访问控制，实现灵活方便、集中统一的授权管理与访问控制系统。解决我能做什么的访问控制问题。

## 6.2 系统实现

1)用户管理

•建立一套CA系统，提供合法的在线考试系统用户身份信息注册

•导入原税务管理系统的用户，由在线考试系统统一进行用户身份信息的注

册、更新、吊销等维护管理。

•系统设计时要求1千人能同时进行在线考试。

2)角色定义

一个系统管理员；两个各个分局的系统管理员；普通参加考试人员；出题人员、出卷人员、组织考试者、阅卷者、成绩统计员

3)资源定义

针对不同的应用系统，授权管理系统中的资源对象可以是多种类型，包括数据、文件、菜单、按钮、权限等对象。针对在线系统特点以及业务流程，本次系统实施中我们定义了以下权限资源，实现有效的统一授权与管理。他们依次是：

•系统功能模块清单；
•考试出题科目；
•考试；
•试卷；
•组织考试部门；
•统计范围；
•成绩查询时间；
•考试时间；
•考试日志
•答题人
•出题人


**图 6-1 本系统的应用示例**

## 6.3 本系统的使用说明

本系统的扩展性较好,其他系统调用它时需要做以下工作:

1、建立数据库及所需的用户、角色表、组织结构、成员关系、权限、权限集等表；并将库文件复制至制定路径。

2、建立对象映射关系

在 applicationcontext.xml 文件中给出库文件所在的计算机的 IP 地址及数据库名。

4、在程序中导入所需的包。

命令格式：import net.jlobo.roler.*(角色)

import net.jlobo.user.*（用户）

总之我们的权限管理访问控制系统只要用以上操作过程就可以避免重复的代码编写，扩展性较好。

# 第7章 结论与前景

本论文的工作是以实际使用的“在线考试系统”的指导下，参阅了大量国内外参考文献而完成的。经过对系统规范兼容性的测试，基于角色的访问控制系统平台的设计已经基本达到NIST RBAC模型和相应控制性规则的要求，并在性能上接近或超过国外一些著名的免费访问控制实现产品。同时，在课题研究和设计过程中还解决了角色基数容量的限制、公共抽象数据接口等关键技术。该系统具有以下一些特点：

- 兼容NIST RBAC 模型，遵从规范中大部分完整性一致性规则
- 支持角色的静态互斥，实现角色基数容量的限制
- 支持访问控制的二次开发，节约了系统扩展升级的开销
- 基于通用开发技术，在J2EE平台架构下，开发出有针对性的实际应用

目前，我们开发的RBAC系统平台已成功于在线考试系统，临夏国税局内外网站等系统，取得良好的效果。

RBAC 作为一种访问控制思想越来越受到人们的关注，是因为目前大多数的访问控制方式存在种种缺点，人们希望能找到一种访问控制方案来代替目前的方式。随着对RBAC 的理论研究越来越成熟和深入，RBAC 将得到越来越多的应用，本人有机会接触RBAC 并参与其中的研究与开发，深感获益甚多。

本项目主要是针对Web 应用，通过合理的简化模型，折衷了实现难度和模型的适用环境；提出并实现了基于资源描述方案，进一步拓宽了系统的适用范围；为将来的扩展提供了必要的机制。

RBAC 在理论上虽然逐渐成熟，但在我们的应用中，感觉还有诸多值得商榷的地方和扩展的余地，有待未来更好的解决。比如，目前我们的系统尚不支持继承、动态SOD这两个RBAC 理论模型的特性，有否必要实现应视未来的应用需求而定。我们在实现这个系统时也注意到了。

## 7.1. 结论

本文所提出的通用权限管理系统已经在多个系统中得到了应用,并表现出了以下优于其他权限管理系统的功能特点:

(1)通用性、扩展性强

该系统适用于任何B/S 系统和C/S 系统,它可以加入到任何具有权限管理功能的系统中,而且,可以不断地被重复利用,避免了程序重新开发,节省了大量的人力物力。

(2)集成简单

程序完成后,业务系统只需要添加少量代码,调用一个方法,即可把本模块无缝集成到其中,非常适合各类软件开发人员使用。

(3)权限管理灵活、用户体验好

本系统提供了管理员对业务系统的部门、用户、角色、操作、对象和权限信息的自定义功能,充分体现了权限管理的灵活性和多样性。

## 7.2.前景

我们设计的权限系统主要想从以下三方面对权限管理系统的应用进行变更:

1) 首先从观念上让大家明白采用统一的权限管理系统是可行的;

目前,权限管理系统也是重复开发率最高的模块之一。在企业中,不同的应用系统都拥有一套独立的权限管理系统。每套权限管理系统只满足自身系统的权限管理需要,无论在数据存储、权限访问和权限控制机制等方面都可能不一样,这种不一致性存在很多弊端。

采用统一的安全管理设计思想,规范化设计和先进的技术架构体系,构建一个通用的、完善的、安全的、易于管理的、有良好的可移植性和扩展性的权限管理系统,使得权限管理系统真正成为权限控制的核心,在维护系统安全方面发挥重要的作用,是十分必要的。

2) 本权限管理系统适合于任意规模、意义系统的使用。

对于这一点我们已在实际中进行了检验。

3) 采用统一的权限管理系统,对国家、单位、个人在都有明显的经济效益。

# 参考文献

[1] 江水，基于角色的存取控制——RBAC. 计算机工程，1998. 24（10）

[2] 吴远成等，基于角色的访问控制RBAC 系统的概要设计. 电子科技大学－卫士通信息安全联合实验室，2001.

[3] David F. Ferraiolo, Janet A. Cugini, D. Richard Kuhn, Role-Based Access Control (RBAC): Features and Motivations, National Institute of Standards and Technology, U. S. Department of Commerce

[4] Ravi S.Sandhu, Edward J Coyne, Hal L.Feinstein and Charles E.Youman, Role-Based Access Control Models, IEEE Computer 29（2）

[5] 何斌，顾健，基于角色访问控制的权限管理系统[J].计算机工程，2004.30（增刊）：326 - 328.

[6]梁彬, 孙玉芳, 石文昌等. 一种改进的基于角色的访问控制实施BLP 模型及其变种的方法[J]. 计算机学报，2004. 27(5):636-644.

[7] Michael E. Shin, Gail-Joon Ahn. 基于角色访问控制的UML表示[J]. 非程序员，2001.（25）：20 —22.

[8] 蔡兰, 郭顺生, 李益兵，基于角色访问控制的动态权限配置研究与实现[J]. 管理技术, 2005. 20 (3) :86 - 87.

[9] 徐震, 李斓, 冯登国. 基于角色的受限委托模型[J]. 软件学报, 2005. 16 (5)

[10] 龙勤，刘鹏，潘爱民，基于角色的扩展可管理访问控制模型研究与实现[J ]. 计算机研究与发展，2005. 42(5)

[11]周沈刚, 赵嵩正. 一种基于RBAC 的Web 环境下信息系统权限控制方法[J]. 计算机应用研究，2005 , (6)

[12]周福才, 李金双, 曹光辉等. 基于MIS 系统访问控制模型的研究[J ]. 小型微型计算机系统, 2004, 25(9)

[13]邹晓. 基于角色的访问控制模型分析与实现[J]. 微计算机信息, 2006, 22(6 3):108-110, 224.

[14]张世明, 杨寅春, 基于角色的访问控制技术在大型系统中的应用[J]. 计算机工程与设计, 2006, 27(19):3723-3725.

[15]王振江,刘强,基于RBAC 的扩展访问控制模型[J]. 计算机工程与应用,2005,(35):23-25.

[16]乔颖,须德,戴国忠. 一种基于角色访问控制(RBAC)的新模型及其实现机制[J]. 计算机研究与发展,2000,37(1):37-44.

[17] National Computer Security Center, A Guide to Understanding Discretionary Access Control in Trusted Systems, NCSC-TG-003, September 1987.

[18]苗雪兰, 一种基于角色的授权管理安全模型的研究与实现[J]. 计算机工程,2002 (9)

[19]王广慧,基于角色的访问控制[J]. 网络安全技术与应用,2002. (9)

[20] Sandhu Ravi, Coyne Edward J , Feinstein Hal L , etal . Role2 Based Access Cont rol Models[J]. IEEE Computer, 1996,29(2): 382-407.

[21]黄建,卿斯汉,温红子,带时间特性的角色访问控制[j]. 软件学报,2003. (14):11

# 致　谢

本论文是在我的导师马义忠教授的精心指导下完成的。在求学和科研期间，马老师对我的专业课程学习、硕士论文撰写等方面都作了周密的安排和悉心的指导。

在此，我特别向马老师致以最衷心的感谢和最诚挚的敬意。同时，我还要感谢我的校外导师——西北民航高级工程师兼甘肃宏科信息技术公司经理孙勋老师以及凌波公司的李凌波经理和全体员工。他们在我的毕业设计和整个项目的实施过程中，给了我耐心细致的帮助和指导，使我在计算机专业领域方面的研究取得了很大的进步。

此外，我还要感谢我的家人以及和我一起学习工作的同事、同学和朋友们，没有他们的帮助和配合，我不可能顺利地完成论文的撰写。

衷心地感谢在百忙之中评阅论文和参加答辩的各位专家、教授！

# 基于角色的权限管理访问控制系统平台研究与实践



作者： 李兰崇
学位授予单位： 兰州大学

## 相似文献(10条)

1.学位论文 杨刚 基于PKI+RBAC具有快表特性的权限管理器的设计与实现 2005

随着计算机网络的普及和发展，网络安全越来越成为人们关注的焦点，成为计算机网络领域的亟待解决和发展的重点。为了加强网络安全，世界各国提出了众多解决方案，PKI等安全体系结构成为其中较为完善和成熟的方案。而为了适应网络应用，特别是适应电子商务和电子政务的发展，权限管理和访问控制已经和网络安全紧密结合，为网络应用提供了坚实的基础和保障。而其中基于角色的访问控制(RBAC)是一种方便、安全、高效的访问控制机制，受到普遍关注。本文在分析了RBAC的基本思想和模型之后，介绍了用户角色分配和角色许可分配的基本方法，并给出了如何在本系统的权限管理器中运用RBAC思想实现权限管理的方法，使该系统具有基于角色的权限管理的功能。

PKI是通过使用公共密钥技术和数字证书来确保系统信息安全，负责验证数字证书持有者身份的一种体系。但仅仅做到了认证用户身份不足以满足网络应用，为此，人们在网络权限管理方面又提出了使用属性证书的PMI体系，使权限管理和身份认证紧密结合，拓展了网络应用。本文分析了PKI和PMI的系统组织结构，认识到从PKI向PMI的扩展存在一定的难度，为了将具有PKI功能的系统平稳过渡到PMI系统，本文就提出了一个具有PMI功能的过渡方案，将PKI与RBAC技术相互结合，构建具有角色特性和PKI身份认证的权限管理器。该系统的结构简单，模块分明，实现了大部分信息资源的统一授权，访问控制和权限管理，并利用数字证书和HTTPS协议保证了网络数据的安全传输。该权限管理器可以在原有的PKI系统上实现简单快捷的扩展，使原PKI网站系统能够通过该权限管理器系统过渡到具有PMI属性证书的网站系统。

在PKI系统的应用中需要查询证书状态，而OCSP(OnlineCertificateStatusProtocol，即在线证书状态查询协议)使用户可以实时查询用户的证书状态，实现了证书状态在线验证。但由于证书状态验证存在跨CA的多级验证问题，验证的速度和复杂性导致了OCSP的使用有一定的问题，为此本文在原OCSP在线验证的基础上，提出了快表的概念，利用在应用系统端构建快表系统，将OCSP验证本地化，实现身份的快速验证。PKI+RBAC网络系统与快表系统的结合，使身份认证和权限管理功能基本实现了本地化，为网络系统，特别是网站资源权限管理提供了良好的解决方案。

2.期刊论文 向号.李明.XIANG Hao.LI Ming 基于协同设计的网络安全机制研究 -煤矿机械2008,29(6)

网络安全是协同设计得以实现和网络发挥效能的重要保证.分析了传统网络协同设计系统框架中的安全隐患,指出了一个完善的网络协同设计系统必须建立自己的安全机制,从信息传输、人员行为等多方面进行防范.提出了集VPN加密传输、基于项目的身份认证和按角色进行权限管理这3项措施于一体的安全防范机制.

3.学位论文 赵亚文 JTang分布式权限管理系统研究 2006

身份认证和权限管理是网络安全的两个核心内容。公钥基础设施(PKI)通过密钥和证书管理方式建立起来的身份认证技术已经非常成熟，但是单独的身份认证技术已经不能完全满足大型企业的安全需求。作为PKI的重要完善和补充，权限管理基础设施(PMI)可提供安全可靠的权限管理服务。

论文针对现有权限管理系统在实际应用中出现的模型柔性差、互操作性不高以及性能不能适应企业级应用等问题，结合浙江省科技厅资助的重大科技攻关项目JTang应用服务器，提出一个基于RBAC的权限管理模型JTangPMI，借助RBAC中成熟的角色理论简化PMI的权限管理，为企业应用系统提供了全面统一的访问控制和授权服务。提出了支持权限约束的PMI集群计算技术，实现PMI系统的高可用性和高可靠性，提高海量访问控制并发操作的性能和服务质量。论文分为以下几个部分：

第一章，概述权限管理基础设施，介绍权限管理基础设施的背景和应用前景；并阐述当前的发展状况；探讨当前发展中面临的挑战；最后，给出论文的课题背景和主要研究内容。

第二章，概述PMI理论体系，给出PMI相关的基本概念，分析了PMI和公钥基础设施(PKI)的关系；并且介绍了PMI的核心部件；然后重点分析PMI中的属性证书框架，最后研究比较了几个现有的同类PMI产品。

第三章，结合RBAC成熟的访问控制理论，提出一个跨应用、跨系统、跨企业的分布式PMI模型。先对访问控制技术进行分析，重点研究基于RBAC的JTangPMI模型和模型的体系结构，针对权限分配、授权等核心功能，给出具体的执行流程和设计过程。

第四章，为了提高企业级权限管理的性能，提出支持权限约束的PMI集群计算技术。首先，介绍了集群环境下的集中常用负载平衡策略，研究集群计算中的服务高可用性、策略可配置性和出错恢复等几个关键技术的设计；然后给出了权限约束的传递规则来简化约束的操作。

第五章，JTangPMI的实现，结合JTang应用服务器的设计，给出JTangPMI的框架设计，详细介绍了重要数据结构的定义和部分核心功能模块的设计。文章最后(第六章)总结了论文的主要工作，并指出了有待进一步研究解决的问题。

4.学位论文 娄巍 Web系统中基于角色的权限管理的研究、设计与应用 2008

随着计算机网络的普及和企业信息化的不断发展，网络成为企业高效处理信息的重要方式，同时针对企业网络的攻击也越来越频繁，网络安全成为一个倍受关注的问题。基于角色的访问控制(RBAC)由于其特有的优点引起了学术和工业界的广泛关注。基于角色的访问控制是将用户划分成与其职能和职位相符合的角色，根据角色赋予相应操作权限，以减少授权管理的复杂性、降低管理开销和为管理员提供一个比较好的实现复杂安全策略的环境。
<br>
本文首先对RBAC产生背景、国内外研究动态及其优点进行简单介绍，分析比较访问控制三种主流技术一自主访问控制、强制访问控制、基于角色的访问控制，着重分析比较了基于角色访问控制的几种主要模型和扩展模型。在此基础上，依据RBAC权限管理模型的关键内容，设计并实现了一个较为通用的RBAC权限管理模块，给出了关键接口说明和数据库描述。最后将该RBAC权限管理模块应用于药品销售查询系统，应用结果表明该模块可以成功解决一般的WEB应用系统中的权限管理问题。

5.会议论文 李超.朱平.秦海权 基于PKI和PMI的安全生物认证系统 2007

随着各种生物识别算法的不断改进,生物认证技术作为一种准确高效的身份认证方法越来越广泛的应用于身份认证的领域.但是目前还没有一种面向开放式网络的通用的生物认证系统。基于X.509(公钥基础设施)的认证机制是开放的互联网络应用最为广泛的身份认证机制之一；利用权限管理基础设施(PMI)进行网络上的权限管理是一种十分有效的方法。为了提高网络中的身份认证和权限管理的,提出了一种通用的基于生物证书的生物认证系统,同时提出了结合PKI、PMI共同实现身份认证和权限管理的系统解决方案。

6.学位论文 成晨 基于PKI的企业网络安全平台 2008

随着信息和网络技术的迅速发展，网络安全问题越来越重要，如何确保开放环境下企业网络的安全接入和安全管理，成为了网络安全领域研究的难点。

本文针对现代企业网络安全实施过程中存在的身份认证，权限管理，安全资源管理三个方面问题进行研究，设计了基于PKI的企业网络安全平台，该平台可以提供安全接入和安全管理的功能。在平台的研究和设计过程中，主要解决了三个方面的关键问题：

第一，对于用户身份认证简单、认证方式多样的问题，可以采用X.509强认证协议进行统一认证，但X.509强认证协议在实际应用中存在私钥不安全和用户冒用的问题。为此，本文在X.509强认证协议基础上增加了USBKey和信息绑定机制，使用硬件保证私钥的安全性，信息绑定机制可以确保用户使用的唯一性，提高了X.509强认证协议的实际应用价值和安全性。

第二，对于安全系统中权限控制管理重复、权限分配和修改工作量较大的问题，论文提出了一种改进的复合数字证书模型。该模型在标准证书的Extension项中添加权限信息，并融合了基于角色的访问控制技术，利用PKI技术的安全性和基于角色访问控制技术的灵活性，达到身份和权限的统一认证。

第三，对于不同的业务系统不能统一管理、管理安全性较低的问题，设计并实现了基于USBkey的安全资源管理模型。基于上述两个方案，在集中式管理模型上加以改进，增加了身份认证、权限管理和加密通信，USBKey作为改进型证书的载体。用户经过认证和授权以后，可以对企业资源的进行管理

。用户客户端与管理服务器端的信息是加密传输的，保证信息不会外泄。管理服务器和内网资源服务器之间采用自定义的协议进行通信，客户端的命令经过管理服务器解析后，再以相应的报文格式传递给内网资源服务器，内网资源服务器执行后台程序并返回结果，达到了监控和管理的目的。

在系统具体实现的过程中，本文采用改进的X.509强认证协议和改进的复合证书模型实现了安全接入子系统，采用安全资源管理模型实现了安全管理子系统。

7.学位论文 陈丹丹 基于RBAC的权限管理组件的设计与实现 2008

访问控制是网络安全防范和保护的主要策略，它的主要任务是保证网络资源不被非法使用和访问。传统的访问控制已经不能满足日益增长的安全性需求。基于角色的访问控制(RBAC)通过引入角色的概念，将用户映射为在一个组织中的某种角色，将访问权限授权给相应的角色，根据用户在组织内所处的角色进行访问授权与控制，有效整合了传统访问控制技术的优势，又克服了他们的不足，使执行企业保护策略的过程更加灵活，并为管理员提供了一个更好的实现安全政策的环境。

本文以RBAC模型为基础，采用Spring框架，结合ibatis技术，设计并实现了一个能提供完整的用户身份认证和集中的应用授权体系的权限管理组件。论文主要工作包括：

1.具体分析RBAC模型，结合Spring和ibatis技术在组件开发中的优势，设计了一个通用的、安全的组件应用框架。基于此框架和RBAC模型，对组件功能模块、访问控制和数据库进行详细设计。

2.采用Spring框架，结合ibatis技术，开发实现了组件持久层、业务层和控制层。由于Spring与多种框架(例如Struts，JSF)相互整合，业务层提供的接口可以供不同的外部应用程序调用，从而实现组件的通用性。

3.基于Spring框架的拦截机制，实现用户身份验证和权限验证。运用信息摘要散列算法(MD5)实现用户登录口令加密传输以防止窃听，并进行数据库口令数据加密保存，实现组件安全机制，有效地完成了访问控制、传输加密、数据库加密的整合。

4.结合具体的项目，将权限管理组件应用到某电视台的后台管理系统中。

论文设计实现的权限管理组件已成功运用在某电视台后台管理系统中。实践表明，该组件具有通用性好、授权灵活、安全性强的特点。

8.学位论文 井营 权限管理基础设施机制研究 2006

近年来，权限管理作为安全的一个领域得到快速发展，目前应用和研究的热点集中在基于公钥基础设施PKI(Public Key Infrastructure) 的权限管理基础设施PMI(Privilege Management Infrastructure)研究。应用PKI的目的是管理密钥并通过公钥算法实现用户身份验证。但通过PKI系统的身份验证仅可以确定用户身份，却不能区分出每个用户的权限到底有多大，这就是PMI产生的一个原因。PMI实际提出了一个新的信息资源保护基础设施，能够系统地建立起对认可用户的授权。但是当用户数量巨大时，PMI对授权用户的验证往往容易出现瓶颈，而且PMI授权策略很容易受具体应用的影响，没有一个统一的标准。

通过改进属性证书的格式，采用短有效期和长有效期相结合的属性证书，短有效期属性证书格式简单，采用“推”的方式与验证服务器进行证书交换，长有效期属性证书采用“拉”方式与验证服务器进行证书交换，这就使系统在处理大用户量访问的过程中，大大降低了验证用户属性证书时出现的瓶颈，同时保证安全性。通过给出基于角色访问控制的授权策略模型，使权限管理基础设施进行授权管理时尽可能地独立于具体的应用，让授权策略模型更广泛地应用在不同应用中。

在改进的属性证书和基于角色访问控制的授权策略模型基础上，结合电子商务相关知识，给出了一个电子支付系统。该系统既解决了交易瓶颈问题以及电子现金容易被重用、伪造或篡改的问题，又解决了对授权用户验证的瓶颈问题以及安全性问题。

9.期刊论文 毕江 电视台大型新闻制作网络的安全策略 -广播与电视技术2003,30(3)

本文对网络安全的意义和新闻网络的安全性要求进行了说明,并从网络整体结构、服务器配置、数据存储以及用户权限管理等四个方面较为详细地讨论了电视台大型新闻制作网络的安全策略.

10.期刊论文 孙超.SUN Chao 基于SOA的网络安全探讨 -电脑编程技巧与维护2010(20)

介绍了一种面向服务架构SOA的网络安全系统的设计与实现.概述了SOA架构的概念、特点、网络安全管理,并详细阐述了基于SOA的网络安全系统的实现.应用实践表明,相对于同类系统,基于SOA的网络安全系统的柔性、重用性和可扩展性更好.