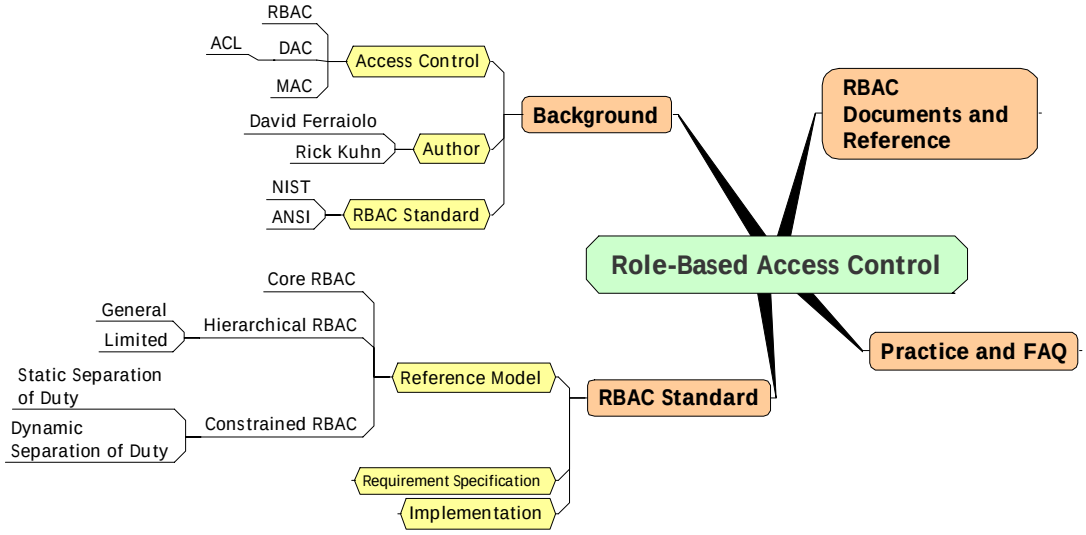
An Introduction to Role-Based Access Control

2004-08-12

冰云 icecloud@sina.com
http://icecloud.51.net
nemo nemo_fc@sina.com
http://icecloud.51.net/nemo



Chapter 1 Background

Section 1.1 Access Control History

访问控制(Access Control)或者说授权(Authorization)，简单的来说就是关注“who can do what”。

访问控制是众多计算机安全解决方案中的一种，是最直观最自然的一种方案。信息安全的风险(Information security risks)可以被宽泛的归结为 CIA：信息机密性（Confidentiality)、信息完整性（Integrity）和信息可用性（Availability)。访问控制主要为信息机密性和信息完整性提供保障。

60 年代末，Lampson 开始着手正式定义并对访问控制进行完善的数学描述方面的工作，他提出了主体（Principal）和客体（Subject）这两个重要的基本概念。并提出，需要有一个访问矩阵来描述主体与客体之间的访问关系。

70 年代，Bell 和 LaPadula 将军事领域的访问控制形式化为一套适合定义和评估计算机系统安全的数学模型，并在这个模型实现了类似于政府文件分级策略的多极安全访问模型。

1983 年，对于访问控制模型标准化是有纪念意义的一年。在这一年美国国防部提出了《计算机系统可信赖性评估标准》(TCSEC)，该标准中详细的定义了两种军事系统的访问控制模式：DAC（discretionary access control）和 MAC（mandatory access

control)。

DAC 即自主访问控制：对象（指文件等，与 oo 语言中的对象无关）的属主全权管理该对象的访问控制策略，有权泄漏、修改对象信息，并且这种权限可以在主体间转移。通常 DAC 使用 ACL（访问控制列表）来限定哪些主体可以对哪些客体执行什么样的操作。

DAC 在一定程度上实现了多用户的权限隔离和资源保护并且实现简便。但是 DAC 的缺陷也十分明显：资源管理过于分散，给控制这个系统的安全造成很大不便；无法防范木马型攻击。

MAC 即强制访问控制，它诞生的初衷是为了弥补 DAC 在防范木马型攻击方面不足。在 MAC 系统独立于用户行为强制执行访问控制策略，每一个主体拥有固定的安全标记，每个客体同样具有安全标记，主体能否对客体进行指定操作取决于主体和客体所拥有安全标记的关系（如：安全标记同为"机密"则可以执行操作）。MAC 的缺陷在于只能应用于等级观念明显的行业（如军队），应用范围过于狭小。

随着时间的推移，越来越多的商业用户意识到 DAC 和 MAC 在商业领域中并不十分有效。而同时期 Clark-Wilson 通过大量商业安全的实际系统形式化出一套与 DAC 和 MAC 完全不同的一套访问控制模型，并第一次提出了职责分离 (SoD, Separation of Duty). 的概念。

在 80 年到 90 年代初这段时期，访问控制领域的研究者逐渐认识到将 Role 作为一个管理 Privilege 的实体单独抽象出来的好处，这时 Role 被认为是组织中的一个职位或位置，是与分离于 User 之外的，并通过将 Role 指派给 User 使之获得某些权限。

1992 年 David Ferraiolo 和 Rick Kuhn 合作提出了 RBAC（Role-Based Access Control）模型，参见：http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf.

在 RBAC 中，在用户(user)和访问权限(permission)之间引入角色(role)的概念，用户与特定的一个或多个角色相联系，角色与一个或多个访问许可权相联系，角色可以根据实际的工作需要生成或取消。由于 RBAC 在管理大型网络应用安全时所表现出的灵活性和经济性迅速成为最具影响的高级访问控制模型。

Section 1.2 About the Author

David Ferraiolo 和 Rick Kuhn 这两位 RBAC 标准的制定者目前均供职于 NIST (National Institute of Standards and Technology)。

http://csrc.nist.gov/rbac/

David Ferraiolo 是一位有 19 年计算机和通信领域系统安全经验的专家，主要研究领域为以 Role 为核心的访问控制、先进访问控制方法论，并为计算机安全领域贡献了 20 多篇学术论文：The NIST model for role-based access control、which access control technique will provide the greatest overall benefit 等。

Rick Kuhn 是一位在计算机安全和软件质量方面有 17 年经验的专家，并在这两个领域著有 40 多篇论文。目前 Rick Kuhn 在 NIST 参与 Quantum Information Networks、Role Based Access Control、Telecommunications Security 三个项目，同时他还作为 IEEE 的高级成员活跃于信息安全领域。

Section 1.3 RBAC Standards

在 RBAC 社区的不断努力下，RBAC 在 2004 年 2 月被美国国家标准委员会（ANSI）和 IT 国际标准委员会(INCITS)接纳为 ANSI INCITS 359-2004 标准。

RBAC 标准包括两个主要部分：RBAC 参考模型和 RBAC 功能描述。

RBAC 参考模型定义了 RBAC 的基本语义和基本元素集合，并通过集合论给出了一套

RBAC 的数学模型。RBAC 功能描述定义了 RBAC 系统必须的特性。包括 administrative operations, administrative reviews, system level functionality.

近期出现了一种称为 XRBAC 的标准，该标准使用 XML，定义了一套 Schema，目的是为了在 Web Service 中进行权限的验证。有兴趣的朋友不妨关注一下。

下面我们主要就 RBAC 的 Reference Model 进行讲述。

Chapter 2 Reference Model

从前面的说明我们了解到，为了将人和权限解耦，RBAC 模型引入了 Role 的概念，整个 RBAC 参考模型都就是围绕 Role 来建立的。

根据对不同复杂度权限的需求，RBAC 参考模型定义了三部分组件（见下图），分别是：

1 Core RBAC

2 Hierarchical RBAC

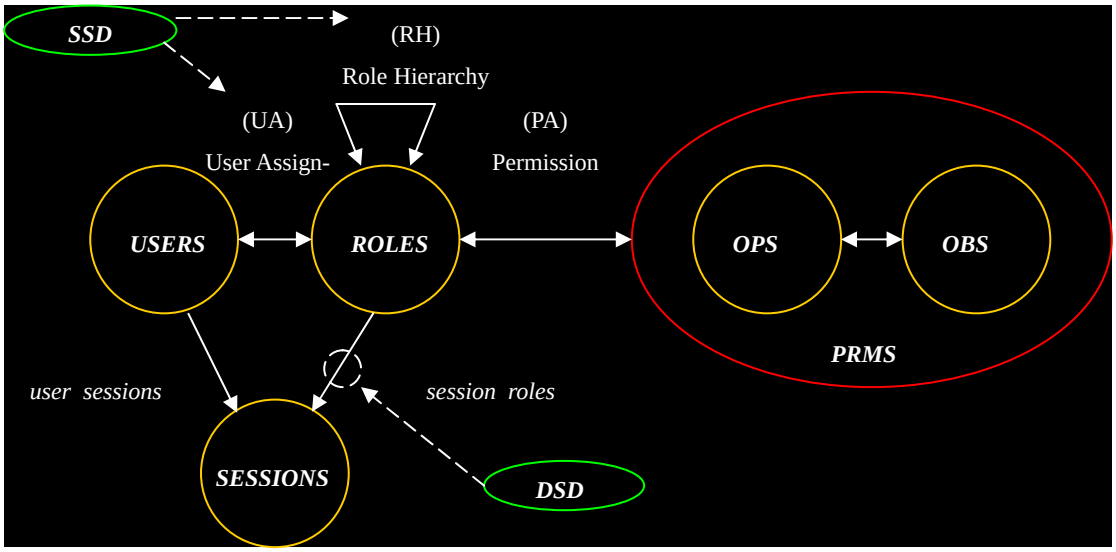
General

Limited

3 Constraining RBAC

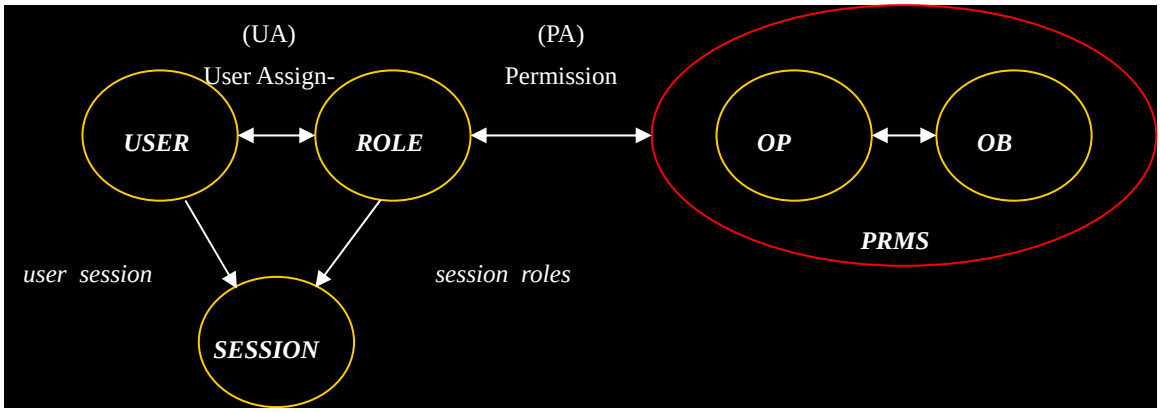
Static Separation of Duty (SSD)

Dynamic Separation of Duty (DSD)



Section 2.1 Core RBAC

Core RBAC 定义了 RBAc 模型最基本的五个元素：User, Role, Operations, Objects Privileges(Permissions)。注：这里提到的 Privilege 和 Permission 等同，Object 和 Resource 等同。



User 代表人，但是也可以是一台机器，agent，或其他任何智能型物品。

Role 表示一个工作职责，在一个组织机构环境中的工作职责。该职责可以关联一些关于权力和责任的语义。

Permission 是一个许可，对在一个或多个 Objects 上执行 Operation 的许可。我们说，一条新闻，这是权限吗？删除，这是权限么？都不是。只有 对新闻的删除，才是权限。另外，RBAC 标准中定义的权限指正权限，但是并没有禁止负权限，因此也可以自己实现。

Operation 是程序的可执行的反映(image)，被 User 调用和执行。Operation 的类型取决于实现系统的类型。例如文件系统可能的操作时读写，执行等，数据库系统则是 CRUD。

Object：表示资源或对象，任何访问控制机制都是为了保护系统的资源。Objects 可能包括文件，目录，数据库表，行，字段等等，甚至于磁盘空间，打印机，CPU 周期等都是资源。

RBAC 的关注点在于 Role 和 User, Permission 的关系。称为 User assignment (UA)和 Permission assignment(PA).关系的左右两边都是 Many-to-Many 关系。就是 user 可以有多个 role，role 可以包括多个 user。

凡是用过 RDBMS 都知道，n:m 的关系需要一个中间表来保存两个表的关系。这 UA 和 PA 就相当于中间表。事实上，整个 RBAC 都是基于关系模型，因此 SSD,RH,DSD 等也都是关系。

Session 在 Core RBAC 中是比较隐晦的一个元素。标准上说：每个 Session 是一个映射，一个用户到多个 role 的映射。当一个用户激活他所有角色的一个子集的时候，建立一个 session。每个 Session 和单个的 user 关联，并且每个 User 可以关联到一或多个 Session.

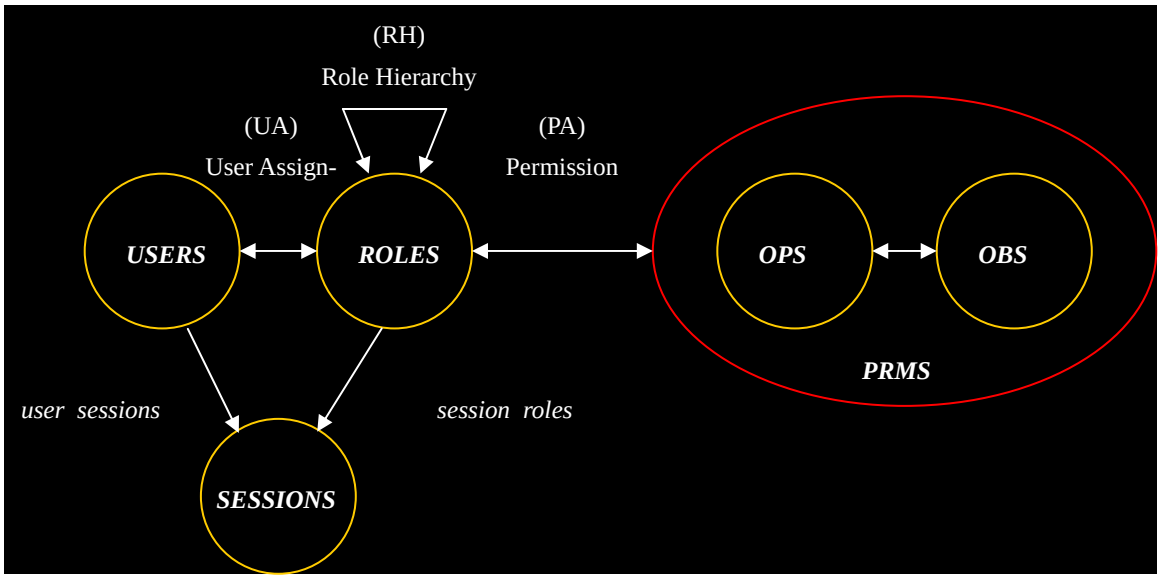
这段话在很多文档中的理解都不同，有的人说，当用户登录到系统后，就建立了一个 Session，生命周期等同于 Web Server 中的 Session。我认为这是错误的。个人理解这段话为：当一个用户执行一段 Process 或 Action 的时候，会建立一个 Session。

举例来说，用户交一个删除新闻的请求，验证权限，系统删除该新闻，返回结果。这个过程中，势必要用到相关的 Role，这用到的 Role 是所有 Role 的子集。这段过程就可以称为一个 Session。

Section 2.2 Hierarchal RBAC

Hierarchal RBAC 组件引入了角色继承的概念。如图。

约束的 RBAC 增加了职责关系分离的部分。这部分是 RBAC 最复杂的部分。

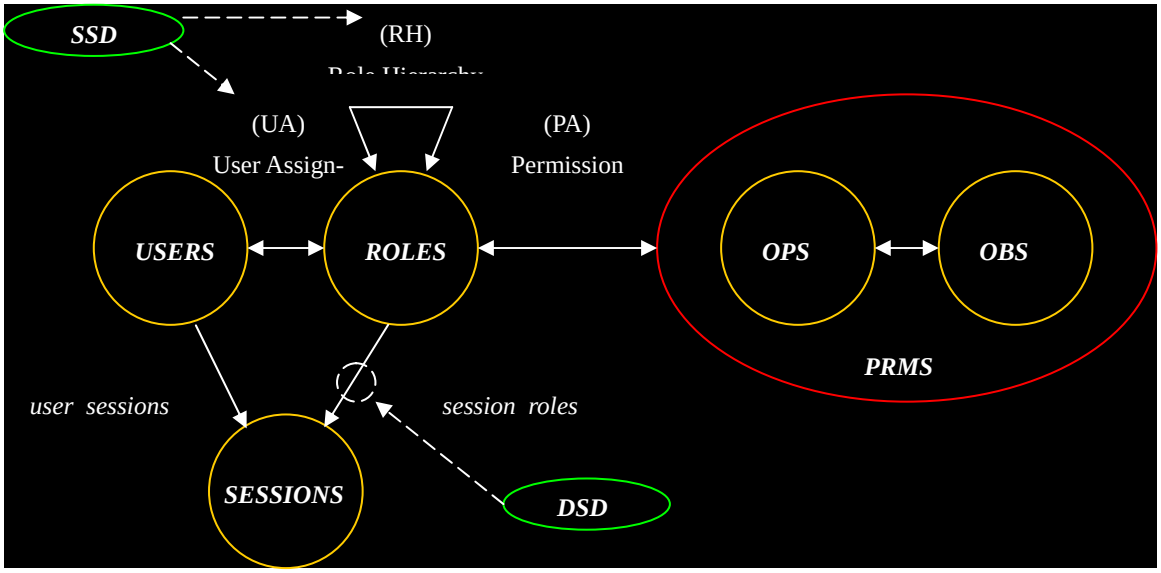


角色继承用于解决复杂组织机构之间的权限关系。它是一种很自然的对组织机构层次中权限和责任的映射。例如，总经理->部门经理->程序员。角色继承方向和用户的关系方向是相反的，即权限最小的在顶端：程序员<-部门经理<-总经理。即如果所有 role2(程序员)的权限都是 role1(经理)的权限，那么 role1 继承自 role2。

层次的 RBAC 包括 General 和 Limited 两种。其区别是：Limited 可以有多个父节点但是只能有一个子节点。是一个反向树结构。而 General 是图。

Section 2.3 Constrained RBAC

约束的 RBAC 增加了职责关系分离的部分。这部分是 RBAC 最复杂的部分。



标准中定义如下： 职责关系隔离（Separation of duty, SOD）用于增强利益冲突策略，这种策略某些组织机构可能会需要，即避免用户超出其当前职位合理的权限等级。

简单的说，就是避免两个角色间的冲突。例如会计和出纳，在一般的公司都不允许同一个人兼任。因此在分配角色的时候，应该禁止这两种角色赋给同一人。

SSD，静态职责隔离，比较容易理解。即在系统初始化的时候，当角色授给用户时来判断是否将冲突的角色给了同一个用户。在 RBAC 标准中，冲突的角色被定义为一个二元关系，就是说，任何一个用户只能拥有其中的一个。

DSD，动态职责隔离，指相冲突的角色可以同时给一个人，但是在一次 Session 中不能同时扮演两个冲突的角色。例如：标准中的例子，某个人可以是收银员或收银员主管。收银员必须经过主管才能打开收银机的抽屉修改某次的结账错误。如果收银员角色的一个单独的行为中需要从收银员切换到主管，那么 DSD 要求，这个 user 必须先放弃收银员角色。也就是说，当该收银员正在收银时候发现错误，必须要先关闭抽屉，然后再次以主管身份打开抽屉才行。

收银机就是最早的的权限系统。

Chapter 3 Requirement Specification

这一部分实际上是把模型中每个组件的行为描述了出来。例如 AddUser, DeleteUser, AddRole, DeleteRole 等等。相当于标准的方法。RBAC 基于集合模型，所有的方法也都是按照集合关系得运算来定义的。

前面提到，功能描述包括三部分：

Administrative Functions：创建，管理元素集合及关系，以建立 RBAC 的多种组件

Supporting System Functions：RBAC 实现需要支持的 RBAC 模型功能，用于当用户与 IT 系统进行交互的时候。

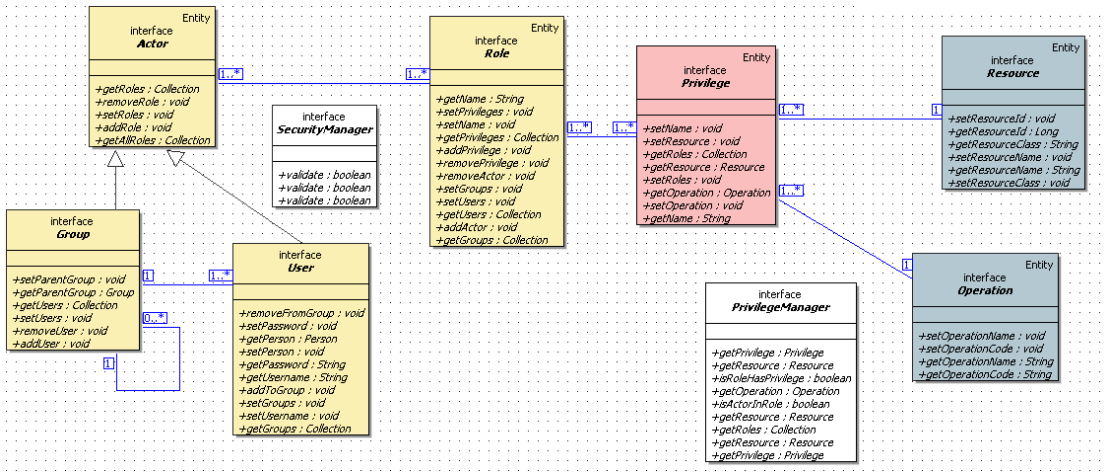
Reviews Functions：检查管理功能的行为结果。

实现一个 RBAC 模型，可以考虑使用这里给出标准方法。由于是语言无关的，这里的方法同时适合面向对象和面向过程的编程泛型，硬搬到 Java 实现有些古怪。

这部分我觉得要求并不是那么严格的。而且难度也不大。

Chapter 4 Implementation

我前段实现了一个比较完整的 Core RBAC。参见图。



http://icecloud.51.net/data/pic/rbac.gif

RBAC 标准说，他们定义的是最小的元素集，可以按照实际情况增加一些其他元素等。

在 RBAC 系统中，User 实际上是在扮演角色(Role)，我用扮演者，也就是 Actor 来取代 User，这个想法来自于 Business Modeling With UML 一书 Actor-Role 模式。考虑到多人可以有相同权限，引入了 Group 的概念。Group 同样也是 Actor。而 User 的概念就具象到一个人。

这里的 Group 和 Group-Based Access Control 中的 Group 不同。GBAC 多用于

操作系统中。其中的 Group 直接和权限相关联，实际上 RBAC 也借鉴了一些 GBAC 的概念。

Group 和 User 都和组织机构有关，但不是组织机构。二者在概念上是不同的。组织机构是物理存在的公司结构的抽象模型，包括部门，人，职位等等，而权限模型是对抽象概念描述。组织结构一般用 Martin fowler 的 Party 或责任模式来建模。

Party 模式中的 Person 和 User 的关系，是每个 Person 可以对应到一个 User，但可能不是所有的 User 都有对应的 Person。Party 中的部门 Department 或组织 Organization，都可以对应到 Group。反之 Group 未必对应一个实际的机构。例如，可以有副经理这个 Group，这是多人有相同职责。

引入 Group 这个概念，除了用来解决多人相同角色问题外，还用以解决组织机构的另一种授权问题：例如，A 部门的新闻我希望所有的 A 部门的人都能看。有了这样一个 A 部门对应的 Group，就可直接授权给这个 Group。

Chapter 5 Practice

RBAC 中的一个比较重要的问题是粒度。通过对 Resource 的粒度控制，可以做到大到整个系统，小到数据库表字段的控制。Resource 可以考虑通过某种规则引擎机制来实现对业务规则的控制。这一点是和大家讨论的结果，还没有实践过。

RBAC 给出的是参考模型，是 RBAC 的最小元素集合。因此可以通过扩展一些部分来对 RBAC 进行扩展。例如上面我将 User 扩展了 Group。或者对权限扩展出负权限等等。不要被局限于标准。

基于 AOP 的权限验证可能是一种解决方案。但是从普通基于动态代理的 AOP 无法得到资源的信息，因此需要让组件暴露出权限给 Aspect Component。这种方法仍然需要进一步的思考与实践。

另外，由于国内政府机关的一些客观原因，权限模型可能会变得异常复杂，在实施阶段可能会有很大工作要做。但是，个人认为，考虑到管理成本和维护成本，RBAC 还是值得考虑的。

以上是我对 RBAC 的一些理解和介绍。本人才研究 RBAC 不久，难免有疏漏及谬误，请批判的对待此文。

Reference Document

RBAC 官方网站

http://csrc.nist.gov/rbac/

RBAC-ACM-Standard

http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf

RBAC-NCITS-Standard

http://csrc.nist.gov/rbac/rbac-std-ncits.pdf

RBAC Book

Role-Based Access Control

ISBN 1-58053-370-1

David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli

代文龙《权限系统概要》

http://www.jdon.com/jive/thread.jsp?forum=46&thread=7309&start

=0&msRange=15
http://dev.csdn.net/develop/article/19/19751.shtm
http://dev.csdn.net/develop/article/19/19753.shtm