# 角色指派规则生成研究

叶春晓[1] 符云清[2] 吴中福[1]

(重庆大学计算机学院 重庆400044)[1] (重庆大学网络教育学院 重庆400044)[2]

**摘 要** RBAC中,用户-角色指派常常由系统管理员完成,适用于用户数量不大,指派关系简单的环境。在分布环境中,用户数量巨大,指派关系复杂且易变,传统的用户-角色指派方法效率较低。角色指派规则的出现为实现URA过程的自动化提供了可能,也是角色对用户的一种限制条件。用户只有满足指派规则后才能获得角色。已有研究工作未说明如何生成指派规则,本文提出了角色指派规则的生成方法。根据角色获得权限的方式,研究了PRA过程中角色指派规则的生成,以及角色继承关系中子角色的角色指派规则生成。同时,本文也对上面两种情况所引起的角色权限变化而带来的角色指派规则的变化进行了研究。

**关键词** RBAC,URA,角色指派规则

## Formulation of User-Role Assignment Rule

YE Chun-Xiao[1] FU Yun-Qing[2] WU Zhong-Fu[1]

(Computer School of Chongqing University,Chongqing400044)[1]

(College of Network Education of Chongqing University,Chongqing400044)[2]

**Abstract** In some environments,the number of users is not huge and the user-role assignment relations are not complex. User-role assignment in such environment is managed by system administrator. But in a distributed environment which has huge number of users and complex and inconstant relations of user-role assignment, traditional URA of RBAC is inefficient. User-Role assignment rule emerged as a solution of automatically URA and a constraint condition of role on user. User who satisfies such rules can be assigned to proper role. There has little work on the formulation of user-role assignment rules. According to the ways that roles gain permissions, this paper introduces two methods of formulation of user-role assignment rules. One of two methods concerns on the formulation of user-role assignment rules in the course of PRA, the other concerns on how to formulate user-role assignment rules in role hierarchy. Also, this paper has introduced how to modify user-role assignment rules in the course of PRA and role hierarchy.

**Keywords** RBAC,URA,User-role assignment rule

## 1 概述

访问控制作为信息系统的安全技术,获得了极大的发展。而基于角色的访问控制技术-RBAC,作为信息安全领域的一种新技术,正不断受到重视。其中具有代表性的模型为Sandhu提出的RBAC96模型[1],该模型一经提出,即受到瞩目,在此基础上人们又提出了许多补充和改进模型。相关的标准也在研究发布中。

作为RBAC96的重要组成部分-ARBAC97[2]由三部分构成,即URA97[3],PRA97,RRA97。通过该三个模型,完成了对用户-角色指派,角色-权限指派和角色-角色指派的管理,使得整个管理过程分散为多个系统管理角色来完成,减轻了系统管理人员的负担。

RBAC中,通过将用户指派相应的角色来获得权限。在实际系统中,角色数量相对固定和较少,而用户数量可以是数以百计、千计。现有模型中的用户-角色指派更多地依靠系统管理员来保证其正确性。约束只是在某种程度和范围保证指派之间不发生冲突,满足系统安全策略。实际应用环境中,为实现用户-角色指派过程的自动化和减少指派过程正确性对系统管理员的依赖,提出了URA指派规则的概念[4]。该规则实际上规定了用户需要具有的相关属性和属性值。只要用户属性和属性值满足该规则的要求,用户可以自动被指派给该角色。这一过程可大大减轻系统管理员的工作负担,并极大地保证了用户-角色指派结果的正确性,适合用户数量巨大且URA过程频繁的环境。

URA指派规则从另一个角度来说,可看作是角色对用户的限制条件。用户只有当自己所具有的属性和属性值达到指派规则后才可被指派角色。该规则可看作是URA过程的指派先决条件,进一步加强指派过程对用户的限制。现有的URA过程中,由于指派先决条件只有角色范围,实际上对用户的限制太弱,也使得URA过程的正确性更多地依靠系统管理员,降低了指派结果的安全。

URA过程中的指派规则显然与角色密切相关,一个角色一般情况下只有一个指派规则,可在形成该角色的时候产生指派规则。由于角色之间存在继承关系,因而子角色的指派规则与父角色指派规则有关。同时,角色是权限的集合,因而指

叶春晓 博士生,讲师,研究方向为网络安全,数据库技术,软件工程。符云清 博士,副教授,研究方向为现代远程教育技术,分布协同作业。吴中福 教授,博士生导师,研究方向为网络安全,现代远程教育技术。

派规则的生成与PRA过程相关。同时考虑到继承关系的更改和角色权限的变化，指派规则的产生可说明如下：

·角色继承。这种情况下，子角色的规则可继承其父角色的规则。若子角色只有一个父角色，则子角色的规则首先可通过继承该父角色的规则获的。若该子角色存在多个父角色，子角色的规则需从多个父角色继承而来，但需考虑各个父角色规则之间的关系。角色继承关系改变后，子角色指派规则应相应调整。

·PRA。这种情况下需考虑该角色现有权限和新指派权限之间的关系。同时，角色权限撤销后该角色的指派规则也应相应地调整。

由于URA的最终结果是使用户获得相应的权限，因而角色的指派规则可分解为各个权限的指派规则。因而由PRA过程产生的角色指派规则为其中各个权限的指派规则的集合。该指派规则集合的生成将在后面章节说明，也是本文主要解决的问题。

## 2 相关工作

ARBABC97及其以后的模型[5,6]均包含URA过程，分别为URA97，URA99[5]，URA02[6]。用户-角色指派过程需满足其中的指派先决条件[7]。URA97和URA99的指派先决条件只包含用户的先决角色。URA02对先决条件进行了扩展，使其可以包含用户的组织结构。但现有的URA过程的指派先决条件还很弱，缺少进行自动指派和加强对用户限制的措施。

ARBAC管理模型中的PRA也同样可分为多个版本，如PRA97[2]，PRA99[5]等。PRA过程完成将权限指派给角色的工作。由于现有RBAC模型中不存在指派规则，因而PRA过程也不可能形成指派规则。

RBAC中引入了角色概念，也同时引入了角色继承概念。通过角色的继承，使得子角色可通过角色继承获得父角色的相关权限，简化了角色-权限指派工作。如果在RBAC中引入指派规则，则存在由于指派规则不同而引起的角色继承关系改变的问题[8]。但由于角色继承引起的父子角色的指派规则关系的研究则相对较少，特别是角色继承下的指派规则生成。

文[9]认为角色应当具有更为丰富的属性，该属性用来表明指派给该角色的用户应当具有的资格和能力。文[4]中的用户属性和属性值若满足指派规则就可以自动指派给相应的角色。文[4]对指派规则的形成没有说明，系统假设指派规则已经存在。

## 3 角色指派规则

**定义1** 指派规则 *rule* 定义如下：

*rule*::=*SE* | True | False

*SE*::=*se*|*se* ∧ *se*

*se*::=*ua roprt uav*

*roprt*::=<|≤|>|≥|=

*uav*::=〈由系统决定〉

*ua*::=〈由系统决定〉

其中，*SE* 为 *rule* 中的 *se* 集合，*se* 为构成属性表达式的一个属性项，也可称为一个状态表达式。∧为逻辑与运算，*roprt* 为关系运算符，*uav* 为属性值，*ua* 为属性。

指派规则表明权限或角色对用户目前状态的限制，也表明具有该权限或角色的用户的资格和能力。如“age＝23∧degree＝‘bachelor’”。权限或角色的指派规则为True表示该权限或角色可被指派给所有用户，或对用的状态没有限制。指派规则为False表示该权限或角色不能被指派给任何用户。一般情况下，可将 *uav* 的数据类型限定在字符、数值和日期三类。

由于用户的属性值不同，属性值之间产生了相应的级别。与文[6]类似，可将属性值之间的级别规定如下：

**定义2** 若 $se_1$、$se_2$中的属性相同，称 $se_1$、$se_2$可比。若 $se_1$、$se_2$中的属性不相同，则称 $se_1$、$se_2$不可比，表示为 $se_1 \approx se_2$。在 $se_1$、$se_2$可比的情况下，若 $se_1$、$se_2$的 *roprt* 和 *uav* 均相同，则称 $se_1$、$se_2$相同，记为 $se_1 = se_2$。

如，“age＝23”与“age＝40”之间可比，而“age＝23”与“degree＝‘master’”之间由于属性不同而不可比。

**定义3** $U(se)$为满足 *se* 限制的用户集合，在 $se_1$、$se_2$可比的情况下，若 $U(se_1) \subset U(se_2)$，则 $se_1 > se_2$。若 $U(se_2) \subset U(se_1)$，$se_1 < se_2$。若 $U(se_2) \cap U(se_1) = \Phi$，则 $se_1 <> se_2$。若 $U(se_2) \cap U(se_1) \neq \Phi$ 且 $U(se_2) \cap U(se_1) \neq U(se_2)$或 $U(se_2) \cap U(se_1) \neq U(se_1)$，则 $se_1 >< se_2$。

实际应用时，*se* 所限制的用户集合之间的子集关系可通过系统设定来简化判断。如 $se_1$:*age*＞23，$se_2$:*age*＞30，则 $se_1 < se2$。$se_1$:*bdate*＜1990/01/01，$se_2$:*bdate*＞1980/01/01，则 $se_1 >< se_2$。需要说明的是，当 *uav* 为字符型时，*se* 之间的优先级判断需人为指定。

## 4 角色指派规则生成

指派规则生成需考虑角色权限获得方式。RBAC中，角色有两种获得权限的方式：通过PRA过程获得和通过角色之间的继承关系获得。前一种方式中，权限的指派规则需要同角色现有的指派规则进行合并以生成角色新的指派规则。为详细说明，本文将其分为单个权限指派和多个权限指派两种方式。同时，角色的权限撤销也会引起角色指派规则的改变，本文对权限撤销后指派规则的修改也做了深入研究。

子角色继承父角色权限的同时，也因为权限的获得而获得指派规则。此时通过角色之间的指派规则合并来生成子角色新的指派规则。同时，由于子角色与父角色之间继承关系的改变，以及子角色增加和删除父角色也会引起的指派规则的改变。本文对此也做了相应研究。

### 4.1 规则生成算法

两个指派规则合并以生成新的指派规则的过程，实际上就是对两个规则中的各个 *se* 之间的关系进行判断后，生成 *se* 之间关系的过程。设 $se_1$，$se_2$为两个状态表达式，则生成新的指派规则判断如下：

1)$se_1 <> se_2$:*rule*＝false

此时 $se_1 \wedge se_2$＝false。即两个状态表达式表示的用户集合不相交，没有任何用户满足。

2)$se_1 > se_2$:*rule*＝ $se_1$

此时 $se_1 \wedge se_2 = se_1$。即 $se_1$对用户的限制性更强，保留对用户限制性更强的状态表达式。

3)$se_1 \approx se_2$:*rule*＝ $se_1 \wedge se_2$

此时 $se_1$和 $se_2$不可比，但其用户集合可能相交，因而需保留两个状态表达式。如 $se_1$:*age*＞30，$se_2$:$se_2$＝‘*F*’

4)$se_1 >< se_2$:*rule*＝ $se_1 \wedge se_2$

此时 $se_1$和 $se_2$之间相交，显然应保留两个状态表达式。

两个指派规则之间合并以生成新的指派规则算法如下：

**算法1** rule($rule_1$,$rule_2$)-两个指派规则合并生成新的指派规则

```
输入:指派规则 rule1,rule2
输出:合并后新的指派规则
方法:
    if rule1=T then
        rule= rule2
    else if rule2=T then
        rule= rule1
    else
        SE←rule1中的 se 集合
        SE'←rule2中的 se 集合
        for all sei in SE' do
            for all sej in SE do
                if sei<>sej then
                    rule=False
                    return rule
                if sei>sej then
                    SE=SE-{sej}
                    SE=SE∪{sei}
                if sei≈sej OR sei><sej then
                    SE=SE∪{sei}
        rule←SE 中各个 se 之间以∧连接生成的表达式
    return rule
```

图1　指派规则生成算法

设|*rule*|为 *rule* 中 *se* 的个数，max(|$rule_1$|,|$rule_2$|)=n，则算法1的时间复杂度为 $O(n^2)$。

### 4.2　权限指派

角色在获得指派的权限时，该权限的指派规则将加入到角色已经有的指派规则中以生成新的指派规则。生成角色新的指派规则需考虑规则中状态表达式之间的关系。本文将权限指派过程分为单个权限指派和多个权限指派两种方式。

4.2.1　*单个权限指派*　其过程中的指派规则生成实际上就是角色现有的指派规则和权限的指派规则之间的合并过程，因而其本质就是两个规则的合并问题。因而只需要将角色和指派权限的规则作为参数带入算法1即可获得合并后的指派规则。设角色 *role* 需指派权限 *p*，角色 *role* 的指派规则为 *role*.rule，权限 *p* 的指派规则为 *p*.rule，则生成 *role* 新的指派规则过程为：*role*.rule=rule (*role*.rule, *p*.rule)。

4.2.2　*多个权限指派*　角色指派多个权限生成指派规则的过程可看作是各个权限指派规则依次同角色指派规则进行合并的过程，

```
算法2  assign(role,p1,…,pi)-角色 role 加入多个权限后指派规则生成
输入:角色 role,权限 p1,…,pi
输出:加入 p1,…,pi 后 role 的指派规则
方法:
    P←{p1,…,pi}
    for pk in P do
        role.rule=rule (role.rule, pk.rule)
```

图2　多个权限指派规则生成算法

设|*P*|为赋予角色的权限数，|*P*|=*m*。|*p*|为权限 *p* 中状态表达式 *se* 的个数，max(|$p_1$|,…,|$p_i$|)=*n*。考虑最极端的情况，即所有 *se* 均进入 *role* 中形成指派规则。则每次进入的 *se* 最多为 *n* 个，且实际情况中|*p*|与 *m* 可比，则算法2的时间复杂度为 $O(1+n^2+2n^2+\cdots+(m-1)n^2)=O((m\cdot n)^2)$。

### 4.3　权限撤销

角色的权限撤销后，该权限对应的指派规则就不再存在于角色中，因而需对角色的指派规则按现有权限进行调整。在本文采用的算法中，需根据角色剩余权限重新生成指派规则。由于重新生成指派规则的过程可看作是各个权限指派规则之间依次合并，因而可利用算法1。

```
算法3  revoke(role, p)-撤销权限 p 后角色 role 指派规则生成
输入:角色 role,权限 p
输出:撤销权限 p 后角色 role 指派规则
方法:
    P←r 中的权限集合-{p}
    for pk in P do
        role.rule=rule (role.rule, pk.rule)
```

图3　权限撤销后指派规则生成算法

算法3的时间复杂度分析同算法2类似。

### 4.4　角色继承

子角色继承父角色时，由于继承了父角色的权限，因而相应的应当继承父角色的指派规则。子角色可以继承一个或多个父角色。先考虑子角色只继承一个父角色的情况，设子角色 *role* 继承父角色 *r*。若 *role* 先前具有权限，也就具有权限规定的指派规则，*role* 在继承父角色 *r* 时，*r* 的指派规则需同 *role* 已有的指派规则合并以生成新的指派规则。若 *role* 先前没有权限，则在继承 *r* 权限时就相应地继承了父角色的指派规则。对 *role* 继承多个父角色 $r_1,\cdots,r_i$ 的情况，实际上就是由单个父角色继承构成，其指派规则生成过程可通过 *role* 的指派规则逐个同父角色指派规则进行合并得到。算法4的时间复杂度分析同前。

```
算法4  assign(role,r1,…,ri)-继承多个父角色后指派规则生成
输入:角色 role,权限 r1,…,ri
输出:继承 r1,…,ri 后 role 的指派规则
方法:
    R←{r1,…,ri}
    if |R|=1 and role.rule=True then /*若 role 只有一个父角色
      且 role 无指派规则*/
      role.rule=r.rule
    else
      for rk in R do
          role.rule=rule (role.rule, rk.rule)
```

图4　多个父角色指派规则生成算法

### 4.5　继承关系删除

继承关系删除是指删除子角色的一个或多个父角色。父角色的删除会引起子角色权限的改变，最终会引起子角色指派规则的改变。先考虑单个父角色的撤销。设 *r* 为角色 *role* 的一个父角色。删除 *r* 后，需重新对 *role* 的其他父角色 $r_1,\cdots,r_i$ 的指派规则进行合并以生成新的指派规则。若同时撤销 *role* 的多个父角色 $r_m,\cdots,r_n$，同样需对 *role* 剩余父角色指派规则进行合并以生成 *role* 新的指派规则。生成新指派规则这一过程可通过对剩余父角色的指派规则分别调用算法1而获得。算法5的时间复杂度分析同前。

```
算法5  revoke(role, r1,…,ri)-撤销父角色 r1,…,ri 后角色 role 指派规则生成
输入:角色 role,需撤销的父角色 r1,…,ri
输出:撤销父角色 r1,…,ri 后角色 role 的指派规则
方法:
    R←role 中的角色集合-{r1,…,ri}
    for rk in R do
        role.rule=rule (role.rule, rk.rule)
```

图5　撤销父角色后指派规则生成算法

**总结**　本文着眼于角色指派规则的生成。对角色指派规则的作用进行了研究，认为其不但可实现 URA 过程的自动化，也是角色对指派该角色的用户资格和能力条件的限制。角色指派规则本身提出较晚，因而其生成过程和方法的研究较少。本文对角色指派规则形式进行了定义，并定义了构成指派规则的状态表达式之间的关系。本文将指派规则生成分为 PRA 过程中的规则生成和角色继承中的规则生成两种类型。并讨论了上述两种类型中由于角色权限的改变而引起的新的角色指派规则的生成。本文给出了相关算法，并进行了性能分析。

本文中指派规则包含的属性相对较少，属性值的数据类

型也较少，属性之间的关系也较简单。进一步的工作包括：支持较多但必要的属性；支持多种属性值数据类型，如集合类型；支持多种运算如集合运算；目前指派规则中属性之间只是AND关系，将来可考虑扩展到OR关系。

## 参考文献


1 Sandhu R S，et al. Role-based access control models. IEEE Computer，1996，29(2)：38～47

2 Sandhu R S，Bhamidipati V，et al. The ARBAC97 model for role-based administration of roles. TISSEC，1999，2(1)：105～135

3 Sandhu R S，Bhamidipati V. The URA97 for Role-based User-Role Assignment. In：Proc. of IFIP WG 11.3 workshop on database security，lake tahoe，California，Aug. 1997. 11～13

4 Al-Kahtani M A，Sandhu R. A Model for Attribute-Based User-Role Assignment. 18th Annual Computer Security Applications Conf. Dec. Las Vegas，Nevada，2002. 9～13

5 Sandhu R，Munawer Q. The ARBAC99 model for administration of roles. In：Proc. of the Annual Computer Security Applications Conf.，1999

6 Oh S，Sandhu R. A Model for Role Administration Using Organization Structure. SACMAT'02，Monterey，California，USA. June 2002. 3～4

7 赵青松，孙玉芳，孙波. 基于系统先决条件的授权模型研究. 计算机研究与发展，2003，40(3)：406～412

8 Al-Kahtani M A，Sandhu R. Induced Role Hierarchies with Attribute-Based RBAC. SACMAT'03，COMO，ITALY，June，2003. 1～4

9 Goh C，Baldwin A. Towards a More Complete Model of Role. Internet Business Management Department HP Laboratories Bristol HPL-98-92 May 1998


（上接第220页）

7.55×95%＋(7.55＋5.65)×5%＝7.8325秒

接下来计算录入分数的时间，首先分析最常见的2位数的分数，操作步骤是：

·输入2个字符：KK

·敲击回车：KKK

根据规则0，所有K前需要增加M：MKMKMK

根据规则2，删除非第一个K前的M：MKKMK

根据规则4，最后一个回车前的M要保留：MKKMK

总时间为：M＋K＋K＋M＋K＝1.35＋0.2＋0.2＋1.35＋0.2＝3.3秒

类似地，录入3位数的100分需要MKKKMK＝3.5秒的时间，录入1位数的分数需要MKMK＝3.1秒的时间。

根据我们对四川省自考考试登分系统的采样统计，出现3位数100分的概率是1%，出现1位数分数(主要是0分)的概率是5%。录入30份试卷的总时间平均为：

(3.3×94%＋3.5×1%＋3.1×5%)×30＝98.76秒

最后是点击"保存"按钮的时间，类似前面的分析，应为HMPK＝3.05秒。

根据前面的分析，完成一个试卷袋30份试卷的成绩录入平均总时间为：

7.8325＋98.76＋3.05＝109.6425≈1.8分钟

## 3 考试登分系统的界面改进

对于四川自考的成绩录入员而言，5个登分小组要在短短几天时间内完成10多万份试卷的成绩录入，这样的操作界面其可用性是不能满足用户需要的。因此，我们对操作界面进行了修正。

对于录入保密号，我们修改成扫描试卷上的保密号条形码。对于分数录入，我们制订了如下规则，录入字母"aa"代表100分，录入符号"\"代表0分；录入两位数的分数、字母"aa"或者符号"\"后，输入焦点自动转移到下一个分数的文本录入框上。这样就减少了大量的键入回车或者Tab的操作。而在录入完成第30份试卷分数后，键入回车相当于点击"保存"按钮。

下面我们分析改进后的平均操作时间：

·把手移动到条形码扫描仪：H

·指向卷袋上的保密号条形码：HP

·点击条形码扫描仪上的"扫描"按钮：HPK

·将手重新移回到键盘：HPKH

根据规则0和规则1，操作应为：HMPKH

这一步的操作时间为：H＋M＋P＋K＋H＝3.45秒。因为省略了6个字符的录入，并且条形码扫描的方式杜绝了录入错误的情况，操作时间得到减少。

对于分数录入步骤而言，使用改进的录入方式，2位和3位的分数的操作时间均为MKK＝1.75秒，1位的分数操作时间为MK＝1.55秒，30份试卷的平均录入时间为：

(1.75×95%＋1.55×5%)×30＝52.2秒

点击"保存"按钮的时间也减少为：MK＝1.55秒

那么完成一个试卷袋录入的总时间为：

3.45＋52.2＋1.55＝57.2秒≈0.9分钟

可以看出，这种方式的操作时间比改进前减少了约一半，改进的效果非常明显。

**总结** 我们在使用一个界面时，常常直观地凭感觉认为某界面操作比较方便，某界面操作非常麻烦，而总是很难说清楚到底方便在哪里、麻烦在哪里。GOMS击键模型给我们提供了一个定量的分析界面操作时间的方法，可以指导我们对系统可用性进行量化评估，分析和计算界面改进前后操作时间的变化，这对于软件界面的设计和选择具有很大意义。

在一个实际使用的登分界面中，我们通过GOMS模型进行分析和改进，将单位登分平均操作时间由近2分钟减少到了1分钟，大大提高了系统可用性。

## 参考文献


1 Jef Raskin. 人机界面. 北京：机械工业出版社，2004

2 刘颖. 人机交互界面的可用性评估及方法. 人类工效学，2002

3 Freed M，Remington R. GOMS，GOMS＋，and PDL. 1999

4 Henrybai. 软件产品的可用性的测试. www.csdn.net. 2004