Shiro学习笔--RBAC模型

|  |  |
| --- | --- |
|  | 🢂 内容概览 |
|  | Why：此文档用来做什么？它存在的意义是什么？为解决什么问题？   |  | | --- | |  |   What：当前包含了那些内容？   |  | | --- | |  |   How：此文档应如何参考？   |  | | --- | |  |   Who：此文档适用于那些人员阅读参考？   |  | | --- | |  | |

目录

[1 RBAC模型 3](#_Toc442364104)

[1.1.1 用户（Subject） 3](#_Toc442364105)

[1.1.2 权限（Permission） 3](#_Toc442364106)

[1.1.3 角色（Role） 3](#_Toc442364107)

[1.1.4 RBAC模型 4](#_Toc442364108)

[1.1.4.1 RBAC0：基本模型 4](#_Toc442364109)

[1.1.4.2 RBAC1：角色分级模型 6](#_Toc442364110)

[1.1.4.3 RBAC2：限制模型 7](#_Toc442364111)

[1.1.4.4 RBAC3： 统一模型 8](#_Toc442364112)

# RBAC模型

在软件系统中，出于安全和隐私的考虑，很多时候权限控制是一项不可或缺的功能，按照个人的理解，权限控制主要解决如下问题：

##### 哪些用户（subject）可以访问资源（系统，模块，接口，服务，文件等）？

##### 不同种类（role）的用户分别具有什么样的权限（permission）？

**凝练：谁（Who）可以到哪里（Where）做什么(What);**

### 用户（Subject）

用户（Subject）之所以使用Subject（主体）这一单词，是因为它是一个泛指，可以是任何访问当前系统的对象，包括：人，第三方子系统等；就如同OOA中用例分析的Actor一样；

### 权限（Permission）

权限（permission），定义了用户在系统中具体可以做的事情，通常是根据不同的业务需求而定的，灵活性相对较高，常见的权限比如：

* 菜单权限：用户可以访问系统的哪些菜单；
* 页面权限：用户可以看到系统的哪些页面；
* 操作权限：用户可以操作系统的哪些功能，比如：修改密码功能就可以视作一项权限；
* 数据权限：用户可以看到和操作哪些数据，比如：erp系统中员工A只能看到自己的工资；

### 角色（Role）

角色(role)的引入是为了降低数据冗余，减少关系维护的复杂度，增加系统模型的抽象性，从而使系统更容易理解（按照具体的业务抽象和归类，便于理解）和使用（维护关系变得更加简单，便于使用）。

对于简单（用户数少，功能少）的系统而言，我们可以直接实现用户到权限的映射，即定义用户可以具有哪些权限；此时，用户和权限的关系是多对多关系，当我们新增/修改用户或权限时，需要同时维护相应的关系；用户&权限直接关联的实体-关系模型如下：

|  |
| --- |
| TODO |

然而，用户和权限通常都是灵活多变的，用户可以是张三、李四、…，权限也可以有很多种类，当系统中用户/权限的数量很多时，产生的关系（用户与具体权限的关系）数量可能会变得难以维护；同时，很多用户的权限存在共性，由此产生的重复关系也会很多，如：用户A可以增删改查自己的博客，需要用4条关系描述增、删、改、查四个权限，假如系统有100万个用户，就需要维护400万份权限关系，数量巨大。大量的关系不仅维护困难，也有可能降低系统的性能。

虽然用户和权限是多变的，但是按照面向对象的思路，我们总是可以将其划分归类，抽取共性，抽象出不同的类别（role），而类别发生改变的次数相对较少，所以如果通过将用户与类别关联，再将类别(role)与权限映射将会大大减少关系维护的数量，比如：拥有对博客增删改查的用户，可以归类为”普通用户”这一角色，所有具有这些权限的用户，都是”普通用户”角色，新增的用户只需要定义所归属的类别，即可获得相应权限。假如有100万个用户，需要维护的关系数量：100万 + 4条，比原先少了很多，并且增加了角色这一层后，权限关系变得更加清晰简单，定义用户时，只需要关注用户具有哪些的角色即可，而新增权限，也只需要考虑哪些角色应该具有该权限，不用逐个用户、逐条权限进行维护。添加角色后，系统的实体-关系模型如下：

|  |
| --- |
| TODO |

### RBAC模型

RBAC模型其实就是我们常用的用户（subject）-角色（role）-权限（permission）模型，只是为了规范化，将其定义为一些列的标准，取名为RBAC模型；

RBAC(role-based-access-control，基于角色的访问控制)，核心思路便是通过角色关联用户和权限，将两者解耦，降低关系维护的复杂度，减少冗余数据。其中用户-角色，角色-权限，都是多对多关系。标准RBAC模型由4个部件模型组成：基本模型RBAC0（Core RBAC）、角色分级模型RBAC1（Hierarchal RBAC）、限制模型RBAC2（Constraint RBAC）和统一模型RBAC3（Combines RBAC）。RBAC0模型是RBAC的基础模型。

#### RBAC0：基本模型

|  |
| --- |
| 用户角色分配  UA  角色许可分配  PA  RBAC基本思想 |

基本模型维护两套关系：用户-角色，角色-权限；

|  |
| --- |
| 用户角色分配  UA  与用户相关的会话集合  Session激活的角色  角色许可分配  PA  RBAC0模型 |

RBAC0定义了能构成一个RBAC控制系统的最小的元素集合。在RBAC之中,包含用户users(USERS)、角色roles(ROLES)、目标 objects(OBS)、操作operations(OPS)、许可权permissions(PRMS)五个基本数据元素， 此模型指明用户、角色、访问权限和会话之间的关系。每个角色至少具备一个权限，每个用户至少扮演一个角色；可以对两个完全不同的角色分配完全相同的访问权限；会话由用户控制，一个用户可以创建会话并激活多个用户角色，从而获取相应的访问权限，用户可以在会话中更改激活角色，并且用户可以主动结束一个会话。

#### RBAC1：角色分级模型

|  |
| --- |
|  |

对角色进行进一步分类和抽象，具有继承关系，子角色具有父角色的所有权限，同时，能够关联自己所特有权限。比如：经理角色可以划分为公司总经理，地区部经理，部门经理等，不同的经理可能具有管理公司的一些通用权限，也有各自特有的权限，比如：总经理可以任免地区部经理，反之则不行；对于角色的进一步抽象，减少了同类角色的冗余关系，能够使角色的职责更加明确；该模型适合于角色层次分明且角色较多的系统。角色间的继承关系可分为一般继承关系和受限继承关系。一般继承关系仅要求角色继承关系是一个绝对偏序关系，允许角色间的多继承。而受限继承关系则进一步要求角色继承关系是一个树结构，实现角色间的单继承。

注：模型的使用，视问题的规模而定；如果系统本身角色不是很多，且角色变动频繁，则使用分级模型可能反而会使系统变得复杂，降低易用性和使用效率；

#### RBAC2：限制模型

|  |
| --- |
|  |

RBAC2模型中添加了责任分离关系。RBAC2的约束规定了权限被赋予角色时,或角色被赋予用户时,以及当用户在某一时刻激活一个角色时所应遵循的强制性规则。责任分离包括静态责任分离和动态责任分离。约束与用户-角色-权限关系一起决定了RBAC2模型中用户的访问许可，此约束有多种，主要包括：

• 互斥角色：同一用户只能分配到一组互斥角色集合中至多一个角色，支持责任分离的原则。

• 基数约束：一个角色被分配的用户数量受限；一个用户可拥有的角色数目受限；同样一个角色对应的访问权限数目也应受限，以控制高级权限在系统中的分配。

• 先决条件角色：可以分配角色给用户仅当该用户已经是另一角色的成员；对应的可以分配访问权限给角色，仅当该角色已经拥有另一种访问权限。

• 运行时互斥：例如，允许一个用户具有两个角色的成员资格，但在运行中不可同时激活这两个角色。

#### RBAC3： 统一模型

|  |
| --- |
|  |