Atitit 安全领域 身份验证技术总结 权限设计 访问控制ac技术 acl rbac abac模式

目录

[1. 1](#_Toc10284)

[2. 1.1什么是权限管理（用户认证 +用户授权）？ 1](#_Toc1685)

[3. 2](#_Toc2695)

[4. 访问控制ac模式总结 2](#_Toc22810)

[4.1. ACL：Access Control List，访问控制列表，用户直接和权限挂钩 DAC 简单 2](#_Toc30298)

[4.1.1. 自主访问控制（DAC: Discretionary Access Control） 2](#_Toc31524)

[4.2.  RBAC基于角色访问控制的核心是用户只和角色关联 3](#_Toc12503)

[4.3. ABAC(Attribute Base Access Control):基于属性的权限控制 4](#_Toc30332)

[4.3.1. 强制访问控制（MAC: Mandatory Access Control） 6](#_Toc6269)

[4.4. IBAC模型：基于身份的访问控制id-Based Access Control 6](#_Toc9242)

[5. 具体实现 6](#_Toc3346)

[5.1. 拦截器模式aop验证 7](#_Toc19629)

[5.2. Cookie token session 7](#_Toc1154)

[5.3. 安全框架Shiro和Spring Security比较 7](#_Toc1671)

[5.4. 用户登录系统也可以使用nginx这类网关简单实现即可 7](#_Toc7072)

[6. Shiro和Spring Security比较 7](#_Toc968)

[7. 客户端验证 vs 服务器验证 8](#_Toc10790)

[7.1.1. 基于服务器验证方式暴露的一些问题 8](#_Toc23291)

[8. Ref 9](#_Toc23390)

# 

# 1.1什么是权限管理（用户认证 +用户授权）？

只要有用户参与的系统一般都要有权限管理，权限管理实现对用户访问系统的控制，按照安全规则或者安全策略控制用户可以访问而且只能访问自己被授权的资源。

对权限的管理又分为两大类别：

用户认证

用户授权

1.1.1用户认证

用户认证，用户去访问系统，系统要验证用户身份的合法性

最常用的用户身份验证的方法：1、用户名密码方式、2、指纹打卡机、3、基于证书验证方法。。系统验证用户身份合法，用户方可访问系统的资源。

# 

# 访问控制ac模式总结

## ACL：Access Control List，访问控制列表，用户直接和权限挂钩 DAC 简单

是前几年盛行的一种权限设计，它的核心在于用户直接和权限挂钩。 RBAC的核心是用户只和角色关联，而角色代表对了权限，这样设计的优势在于使得对用户而言，只需角色即可以，而某角色可以拥有各种各样的权限并可继承

ACL用来描述权限规则或用户和权限之间关系的数据表.系统会识别用户,谈话根据ACL(用户权限关系表)来判断用户是否对操作的对象(Subject)有权限.这种设计最常见的应用就是文件系统的权限设计，如微软的NTFS。

### 自主访问控制（DAC: Discretionary Access Control）

系统会识别用户，然后根据被操作对象（Subject）的权限控制列表（ACL: Access Control List）或者权限控制矩阵（ACL: Access Control Matrix）的信息来决定用户的是否能对其进行哪些操作，例如读取或修改。

而拥有对象权限的用户，又可以将该对象的权限分配给其他用户，所以称之为“自主（Discretionary）”控制。

这种设计最常见的应用就是文件系统的权限设计，如微软的NTFS。

DAC最大缺陷就是对权限控制比较分散，不便于管理，比如无法简单地将一组文件设置统一的权限开放给指定的一群用户。

## RBAC基于角色访问控制的核心是用户只和角色关联

RBAC: Role-Based Access Control) 最为普及

因为DAC和MAC的诸多限制，于是诞生了RBAC，并且成为了迄今为止最为普及的权限设计模型。

由于ACL对权限控制比较分散,不便于管理,比如无法将同一类的对象操作权限分配给同一类用户.基于这样的需求就有了RBAC,RBAC不用给用户单个分配权限,只用指向对应的角色就会有对应的权限,而且分配权限和收回权限都很方便.

如菜单权限的设计：用户与角色关联，角色与菜单关

如图所示，每个用户关联一个或多个角色，每个角色关联一个或多个权限，从而可以实现了非常灵活的权限管理。角色可以根据实际业务需求灵活创建，这样就省去了每新增一个用户就要关联一遍所有权限的麻烦。简单来说RBAC就是：用户关联角色，角色关联权限。另外，RBAC是可以模拟出DAC和MAC的效果的。

例如数据库软件MongoDB便是采用RBAC模型，对数据库的操作都划分成了权限（[MongoDB权限文档](https://link.jianshu.com?t=https://docs.mongodb.com/manual/reference/privilege-actions/" \t "_blank)）：

#### 职责分离(Separation of Duty)

为了避免用户拥有过多权限而产生利益冲突，例如一个篮球运动员同时拥有裁判的权限（看一眼就给你判犯规狠不狠？），另一种职责分离扩展版的RBAC被提出。

职责分离有两种模式：

* 静态职责分离(Static Separation of Duty)：用户无法同时被赋予有冲突的角色。
* 动态职责分离(Dynamic Separation of Duty)：用户在一次会话（Session）中不能同时激活自身所拥有的、互相有冲突的角色，只能选择其一。

## ABAC(Attribute Base Access Control):基于属性的权限控制

其实RBAC已经可以解决大部分的场景了.

但是当出现这样一个场景,就拿上面的例子,经理和销售的访问权限不一样,现在业务上需要一种权限,比如华东地区的销售只能看华东地区的客户信息,华北地区的销售只能看华北地区的客户信息,这个时候一般都是在用户数据表里添加一个地区的信息,在客户表里添加一个地区的信息,如果一个销售查询一个客户的时候,这个销售和这个客户的地区信息一样都是华北的,那么就可以访问,如果销售是华南的而客户是华北的就不可以访问. 但是这种情况往往需求会变,比如某一天华东和华南地区销售合并,就是说华南的销售可以访问华南和华东的客户信息,程序员就炸了,代码要重新写一套业务逻辑.

这个时候就有了ABAC(基于属性的权限控制),ABAC则是通过动态计算一个或一组属性来是否满足某种条件来进行授权判断（可以编写简单的逻辑）.属性通常来说分为四类：用户属性（如用户年龄）,环境属性（如当前时间）,操作属性（如读取）和对象属性（如一篇文章,又称资源属性）,所以理论上能够实现非常灵活的权限控制,几乎能满足所有类型的需求.

所以上面的例子就可以写一个规则:"华北地区销售查看华北地区客户信息",其中华北地区销售是用户属性,查看是操作属性,华北地区客户信息是对象属性.这样一来需求变动的时候只需要修改规则就可以了.

但是ABAC是比较复杂的一种权限控制,如果规则过多计算会比较复杂且耗时,太过复杂对于问题追查也不方便.所以现在大部分还是使用的RBAC.

为了实现便捷的规则设置和规则判断执行，ABAC通常有配置文件（XML、YAML等）或DSL配合规则解析引擎使用。

BAC有如下特点：

1. 集中化管理
2. 可以按需实现不同颗粒度的权限控制
3. 不需要预定义判断逻辑，减轻了权限系统的维护成本，特别是在需求经常变化的系统中
4. 定义权限时，不能直观看出用户和对象间的关系
5. 规则如果稍微复杂一点，或者设计混乱，会给管理者维护和追查带来麻烦
6. 权限判断需要实时执行，规则过多会导致性能问题

既然ABAC这么好，那最流行的为什么还是RBAC呢？

我认为主要还是因为大部分系统对权限控制并没有过多的需求，而且ABAC的管理相对来说太复杂了

ABAC有时也被称为PBAC（Policy-Based Access Control）或CBAC（Claims-Based Access Control）。

### 强制访问控制（MAC: Mandatory Access Control）

MAC是为了弥补DAC权限控制过于分散的问题而诞生的。在MAC的设计中，每一个对象都都有一些权限标识，每个用户同样也会有一些权限标识，而用户能否对该对象进行操作取决于双方的权限标识的关系，这个限制判断通常是由系统硬性限制的。比如在影视作品中我们经常能看到特工在查询机密文件时，屏幕提示需要“无法访问，需要一级安全许可”，这个例子中，文件上就有“一级安全许可”的权限标识，而用户并不具有。

MAC非常适合机密机构或者其他等级观念强烈的行业，但对于类似商业服务系统，则因为不够灵活而不能适用

## IBAC模型：基于身份的访问控制id-Based Access Control

* + 基于身份的访问控制模型
  + 举例：登录验证
  + 比如Java中使用cookie、session存储回话标识；

传统的访问控制以用户为中心, 使用标识符来表示主体。如果主体是人, 其标识符就是身份证号。在RBAC中, 主体则是角色, 而客体、环境和操作等也是角色

# 具体实现

## 拦截器模式aop验证

## Cookie token session

## 安全框架Shiro和Spring Security比较

Spring Security拥有完整的生态且文档全面，Shiro简单易用无需依赖能独立运

Apache Shiro是Java的一个安全框架。目前，使用Apache Shiro的人越来越多，因为它相当简单，对比Spring Security，可能没有Spring Security做的功能强大，但是在实际工作时可能并不需要那么复杂的东西，所以使用小而简单的Shiro就足够了。对于它俩到底哪个好，这个不必纠结，能更简单的解决项目问题就好了。

## 用户登录系统也可以使用nginx这类网关简单实现即可

当使用nginx搭建web服务，需要限制访问，又没有构建用户系统的必要，可使用nginx进行简单配置

# **Shiro和Spring Security比较**

Shiro比Spring更容易使用，实现和最重要的理解

Spring Security更加知名的唯一原因是因为品牌名称

“Spring”以简单而闻名，但讽刺的是很多人发现安装Spring Security很难

然而，Spring Security却有更好的社区支持

# 客户端验证 vs 服务器验证

### ****基于服务器验证方式暴露的一些问题****

****Seesion：****每次认证用户发起请求时，服务器需要去创建一个记录来存储信息。当越来越多的用户发请求时，内存的开销也会不断增加。

****可扩展性：****在服务端的内存中使用Seesion存储登录信息，伴随而来的是可扩展性问题。

****CORS(跨域资源共享)：****当我们需要让数据跨多台移动设备上使用时，跨域资源的共享会是一个让人头疼的问题。在使用Ajax抓取另一个域的资源，就可以会出现禁止请求的情况。

****CSRF(跨站请求伪造)：****用户在访问银行网站时，他们很容易受到跨站请求伪造的攻击，并且能够被利用其访问其他的网站。

# Ref

Atitit 安全技术访问控制 ABAC 与IBAC RBAC

目录

1. 访问控制的三个基本要素：主体（请求实体）、客体（资源实体）、控制策略（属性集合）； 1

2. 发展历程 1

2.1. 示意图： 2

3. 访问控制理论模型： 2

3.1. DAC&MAC模型 2

3.2. IBAC模型：基于身份的访问控制id-Based Access Control 3

3.3. RBAC模型：基于角色的访问控制（Role-Based Access Control） 3

3.4. ABAC模型： 3

3.5. 基于身份的访问控制（IBAC）不 4

3.6. 基于角色的访问控制（RBAC 4

4. 他们之间的关系 4

5. Ref 4