可能是史上最全的权限系统设计

权限系统设计

前言

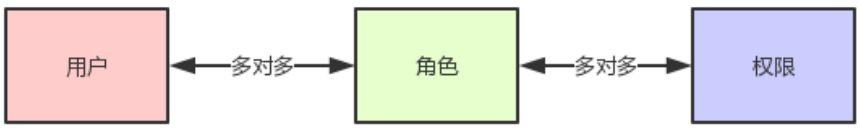
权限管理是所有后台系统的都会涉及的一个重要组成部分，主要目的是对不同的人访问资源进行权限的控制，避免因权限控制缺失或操作不当引发的风险问题，如操作错误，隐私数据泄露等问题。目前在公司负责权限这块,所以对权限这块的设计比较熟悉,公司采用微服务架构,权限系统自然就独立出来了,其他业务系统包括商品中心,订单中心,用户中心,仓库系统,小程序,多个APP等十几个系统和终端

1.权限模型

迄今为止最为普及的权限设计模型是RBAC模型,基于角色的访问控制（Role-Based Access Control）

1.1 RBAC0模型

RBAC0模型如下:



这是权限最基础也是最核心的模型,它包括用户/角色/权限,其中用户和角色是多对多的关系,角色和权限也是多对多的关系。
用户是发起操作的主体,按类型分可分为2B和2C用户,可以是后台管理系统的用户,可以是OA系统的内部员工,也可以是面向C端的用户,比如阿里云的用户。
角色起到了桥梁的作用,连接了用户和权限的关系,每个角色可以关联多个权限,同时一个用户关联多个角色,那么这个用户就有了多个角色的多个权限。有人会问了为什么用户不直接关联权限呢?在用户基数小的系统,比如20个人的小系统,管理员可以直接把用户和权限关联,工作量并不大,选择一个用户勾选下需要的权限就完事了。但是在实际企业系统中,用户基数比较大,其中很多人的权限都是一样的,就是个普通访问权限,如果管理员给100人甚至更多授权,工作量巨大。这就引入了"角色(Role)"概念,一个角色可以与多个用户关联,管理员只需要把该角色赋予用户,那么用户就有了该角色下的所有权限,这样设计既提升了效率,也有很大的拓展性。
权限是用户可以访问的资源,包括页面权限,操作权限,数据权限:

- 页面权限: 即用户登录系统可以看到的页面,由菜单来控制,菜单包括一级菜单和二级菜单,只要用户有一级和二级菜单的权限,那么用户就可以访问页面
- 操作权限: 即页面的功能按钮,包括查看,新增,修改,删除,审核等,用户点击删除按钮时,后台会校验用户角色下的所有权限是否包含该删除权限,如果是,就可以进行下一步操作,反之提示无权限。有的系统要求"可见即可操作",意思是如果页面上能够看到操作按钮,那么用户就可以操作,要实现此需求,这里就需要前端来配合,前端开发把用户的权限信息缓存,在页面判断用户是否包含此权限,如果有,就显示该按钮,如果没有,就隐藏该按钮。某种程度上提升了用户体验,但是在实际场景可自行选择是否需要这样做

大佬，请问数据权限组织id这个是干嘛的?

--继续向前走

2. Re:可能是史上最全的权限系统设计

数据权限id是什么，请问?

--Master.Zhao

3. Re:可能是史上最全的权限系统设计

makr

--ChainFnas

4. Re:可能是史上最全的权限系统设计

不错，看了后感觉现在做的项目权限设计真的没有考虑到这么多，可能是现在做的项目比较小吧。

--Monstereat

5. Re:可能是史上最全的权限系统设计

每一个关联关系表都有个数据权限组织ID，是干什么的？另外数据表设计缺了职位的授权体系和用户与角色的授权关系。最麻烦的事多种组合后的权限计算

--pandgewo

阅读排行榜

1. 可能是史上最全的权限系统设计(30870)
2. [Apache Pulsar] 企业级分布式消息系统-Pulsar入门基础(2016)
3. [Apache Pulsar] 企业级分布式消息系统-Pulsar快速上手(1269)
4. 为什么使用Redis(762)
5. 单点登录(618)

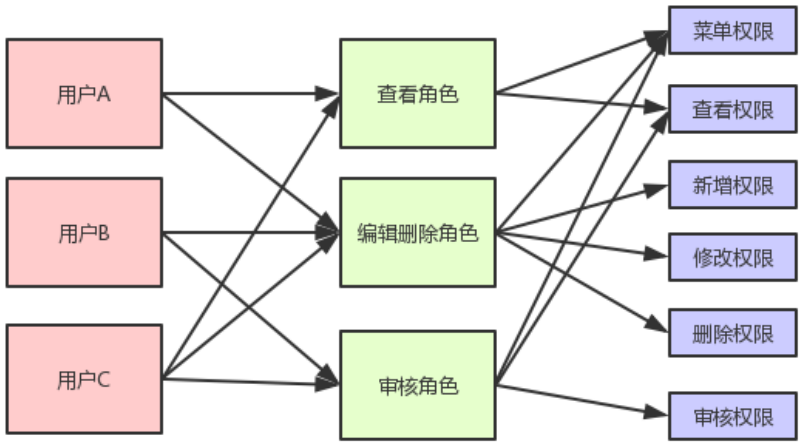
评论排行榜

1. 可能是史上最全的权限系统设计(26)
2. MySQL索引原理及SQL优化(1)
3. 多线程入门(1)

推荐排行榜

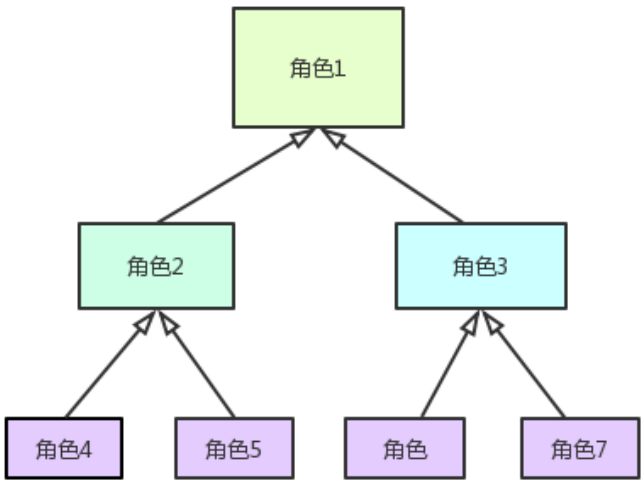
1. 可能是史上最全的权限系统设计(41)
2. 多线程入门(3)
3. JDK源码分析系列---String,StringBuilder,StringBuffer(3)
4. MySQL索引原理及SQL优化(2)

- 数据权限: 数据权限就是用户在同一页面看到的数据是不同的,比如财务部只能看到其部门下的用户数据,采购部只看采购部的数据,在一些大型的公司,全国有很多城市和分公司,比如杭州用户登录系统只能看到杭州的数据,上海用户只能看到上海的数据,解决方案一般是把数据和具体的组织架构关联起来,举个例子,再给用户授权的时候,用户选择某个角色同时绑定组织如财务部或者合肥分公司,那么该用户就有了该角色下财务部或合肥分公司下的的数据权限。



以上是RBAC的核心设计及模型分析,此模型也叫做RBAC0,而基于核心概念之上,RBAC还提供了扩展模式。包括RBAC1,RBAC2,RBAC3模型。下面介绍这三种类型

1.2 RBAC1模型



此模型引入了角色继承(Hierarchical Role)概念,即角色具有上下级的关系,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。这种设计可以给角色分组和分层，一定程度简化了权限管理工作。

1.3 RBAC2模型

基于核心模型的基础上，进行了角色的约束控制,RBAC2模型中添加了责任分离关系,其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。主要包括以下约束:

- 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。互斥角色是指各自权限互相制约的两个角色。比如财务部有会计和审核员两个角色,他们是互斥角色,那么用户不能同时拥有这两个角色,体现了职责分离原则
- 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: 即用户想获得某上级角色,必须先获得其下一级的角色

1.4 RBAC3模型

即最全面的权限管理,它是基于RBAC0,将RBAC1和RBAC2进行了整合

1.5 用户组

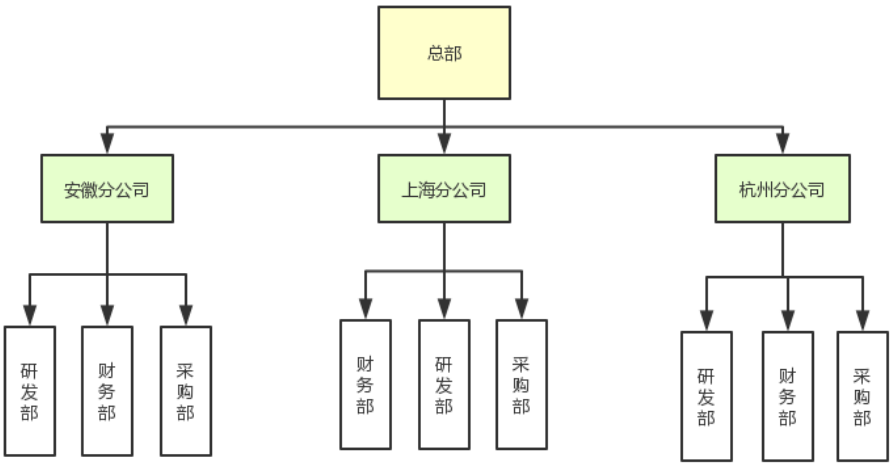
当平台用户基数增大，角色类型增多时，而且有一部分人具有相同的属性,比如财务部的所有员工,如果直接给用户分配角色，管理员的工作量就会很大,如果把相同属性的用户归类到某用户组,那么管理员直接给用户组分配角色,用户组里的每个用户即可拥有该角色,以后其他用户加入用户组后,即可自动获取用户组的所有角色,退出用户组,同时也撤销了用户组下的角色,无须管理员手动管理角色。
根据用户组是否有上下级关系,可以分为有上下级的用户组和普通用户组:

- 具有上下级关系的用户组: 最典型的例子就是部门和职位,可能多数人没有把部门职位和用户组关联起来吧。当然用户组是可以拓展的,部门和职位常用于内部的管理系统,如果是面向C端的系统,比如淘宝网的商家,商家自身也有一套组织架构,比如采购部,销售部,客服部,后勤部等,有些人拥有客服权限,有些人拥有上架权限等等,所以用户组是可以拓展的
- 普通用户组: 即没有上下级关系,和组织架构,职位都没有关系,也就是说可以跨部门,跨职位,举个例子,某电商后台管理系统,有拼团活动管理角色,我们可以设置一个拼团用户组,该组可以包括研发部的后台开发人员,运营部的运营人员,采购部的人员等等。

每个公司都会涉及到到组织和职位,下面就重点介绍这两个。

1.5.1 组织

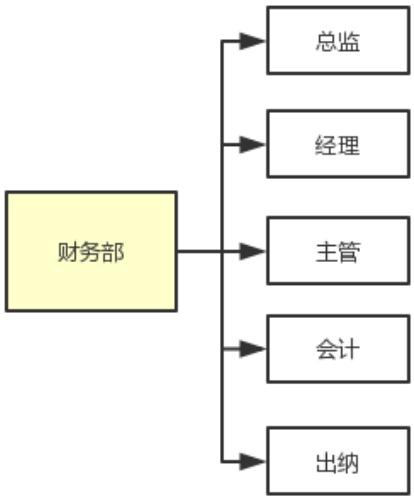
常见的组织架构如下图:



我们可以把组织与角色进行关联,用户加入组织后,就会自动获得该组织的全部角色,无须管理员手动授予,大大减少工作量,同时用户在调岗时,只需调整组织,角色即可批量调整。组织的另外一个作用是控制数据权限,把角色关联到组织,那么该角色只能看到该组织下的数据权限。

1.5.2 职位

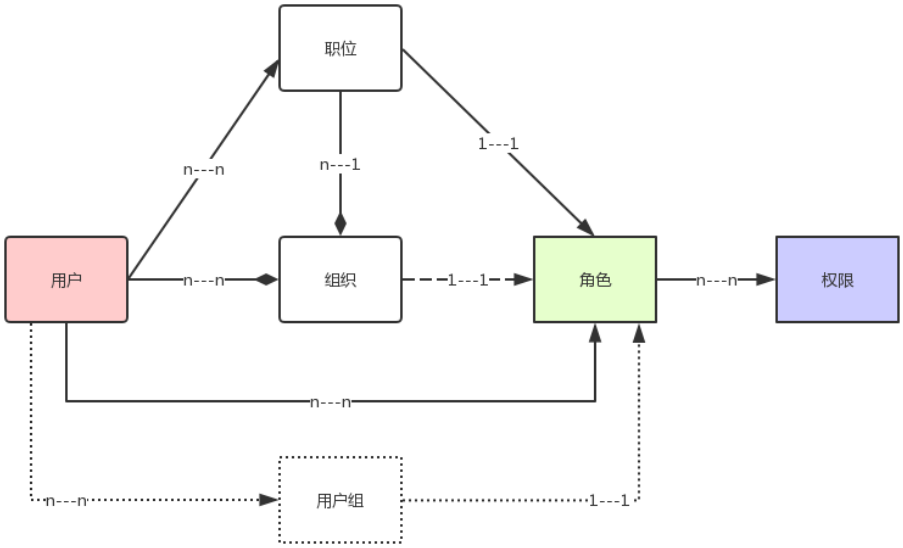
假设财务部的职位如下图:



每个组织部门下都会有多个职位,比如财务部有总监,会计,出纳等职位,虽然都在同一部门,但是每个职位的权限是不同的,职位高的拥有更多的权限。总监拥有所有权限,会计和出纳拥有部分权限。特殊情况下,一个人可能身兼多职。

1.6 含有组织/职位/用户组的模型

根据以上场景,新的权限模型就可以设计出来了,如下图:



根据系统的复杂度不同,其中的多对多关系和一对一关系可能会有变化,

- 在单系统且用户类型单一的情况下,用户和组织是一对一关系,组织和职位是一对多关系,用户和职位是一对一关系,组织和角色是一对一关系,职位和角色是一对一关系,用户和用户组是多对对关系,用户组和角色是一对一关系,当然这些关系也可以根据具体业务进行调整。模型设计并不是死的,如果小系统不需要用户组,这块是可以去掉的。
- 分布式系统且用户类型单一的情况下,到这里权限系统就会变得很复杂,这里就要引入了一个"系统"概念,此时系统架构是个分布式系统,权限系统独立出来,负责所有的系统的权限控制,其他业务系统比如商品中心,订单中心,用户中心,每个系统都有自己的角色和权限,那么权限系统就可以配置其他系统的角色和权限。
- 分布式系统且用户类型多个的情况下,比如淘宝网,它的用户类型包括内部用户,商家,普通用户,内部用户登录多个后台管理系统,商家登录商家中心,这些做权限控制,如果你作为架构师,该如何来设计呢?大神可以在评论区留言交流哦!

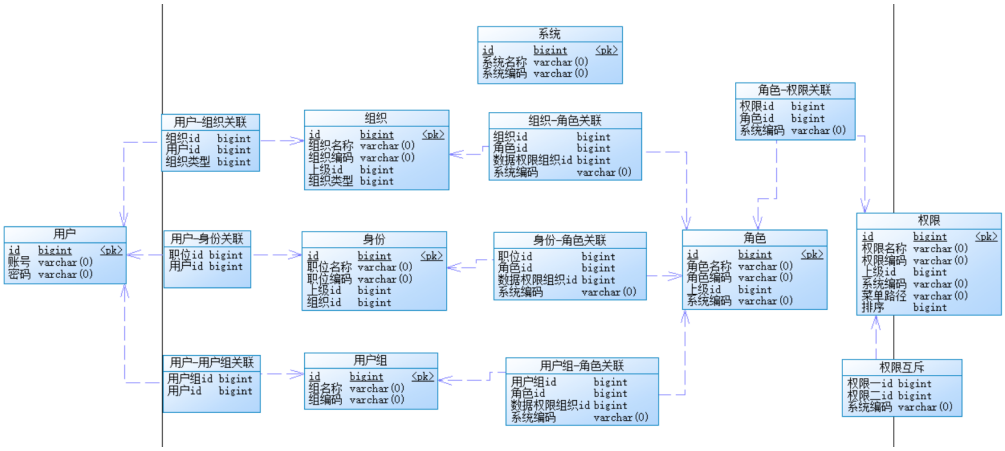
2.授权流程

授权即给用户授予角色,按流程可分为手动授权和审批授权。权限中心可同时配置这两种,可提高授权的灵活性。

- 手动授权: 管理员登录权限中心为用户授权,根据在哪个页面授权分为两种方式:给用户添加角色,给角色添加用户。给用户添加角色就是在用户管理页面,点击某个用户去授予角色,可以一次为用户添加多个角色;给角色添加用户就是在角色管理页面,点击某个角色,选择多个用户,实现了给批量用户授予角色的目的。
- 审批授权: 即用户申请某个职位角色,那么用户通过OA流程申请该角色,然后由上级审批,该用户即可拥有该角色,不需要系统管理员手动授予。

3.表结构

有了上述的权限模型,设计表结构就不难了,下面是多系统下的表结构,简单设计下,主要提供思路:



4.权限框架

- Apache Shrio
- Spring Security

在项目中可以采用其中一种框架,它们的优缺点以及如何使用会在后面的文章中详细介绍.

5.结语

权限系统可以说是整个系统中最基础,同时也可以很复杂的,在实际项目中,会遇到多个系统,多个用户类型,多个使用场景,这就需要具体问题具体分析,但最核心的RBAC模型是不变的,我们可以在其基础上进行扩展来满足需求。
最后,如果您觉得这篇文章对您有帮助,可以点个赞,谢谢支持!

分类: 系统设计

iceblow
关注 - 4
粉丝 - 27
+加关注

41

« 上一篇: JDK源码分析系列---ArrayList和LinkList
» 下一篇: [Apache Pulsar] 企业级分布式消息系统-Pulsar入门基础

posted on 2019-07-12 18:03 iceblow 阅读(30880) 评论(26) 编辑 收藏

发表评论

#1楼 2019-07-12 20:39 | 彪悍的代码不需要注释

666

支持(1) 反对(0)

#2楼 2019-07-13 09:14 | 逗苗

持续关注，自己建了8个数据表来维护权限，本来以为挺多了，跟文中的图一比还是比较简单的了，感谢分享

支持(2) 反对(0)

#3楼 2019-07-13 09:36 | 我屮艸䒑蹦

mark

支持(1) 反对(0)