浅谈 RBAC 权限模型

写作背景

工作两年半了，笔者一直在做 `To B` 的产品，像是后端管理系统、 `Saas` 系统都有接触过，它们都有一个共同点：权限管理。我每天都在接触但只是从前端开发这个角色去理解，我对整个业务流程其实是比较模糊的，所以写下这篇文章方便去帮助自己理解整个业务流程，并且尽量用易懂的文字去表达，希望大家看了也能有所收获。本篇文章主要讲述的是 RBAC 模型，后面再写相关的设计，敬请期待。

为什么需要权限系统

任何业务的产生总是伴随着一定的背景，权限系统也不例外，况且它还是一个“刚需”。现如今的产品其实就是 **数据制作** 与 **数据消费** 的时代，但并不是所有数据都能被全量消费的，为什么呢？

- 产品需要盈利，某些数据的“制造”与“消费”通过订阅的方式按需收费。比如 `Sass` 平台服务会推出各种定价版本：个人版本、团队版本、公司版本，比如 `ProcessOn` 、 `石墨文档` 、 `蓝湖` 等平台。
- 公司内部人员使用管理平台需要各司其职，不能越权操作。
- 产品安全性，防止某些非法分子越权窃取数据。

可见权限系统是很有必要的，而基于角色的权限访问控制模型更为流行（ `RBAC` ），翻译过来其实就是： **具有某个角色的用户对某个资源操作的权限。** 下面我们看看这个模型的发展过程。

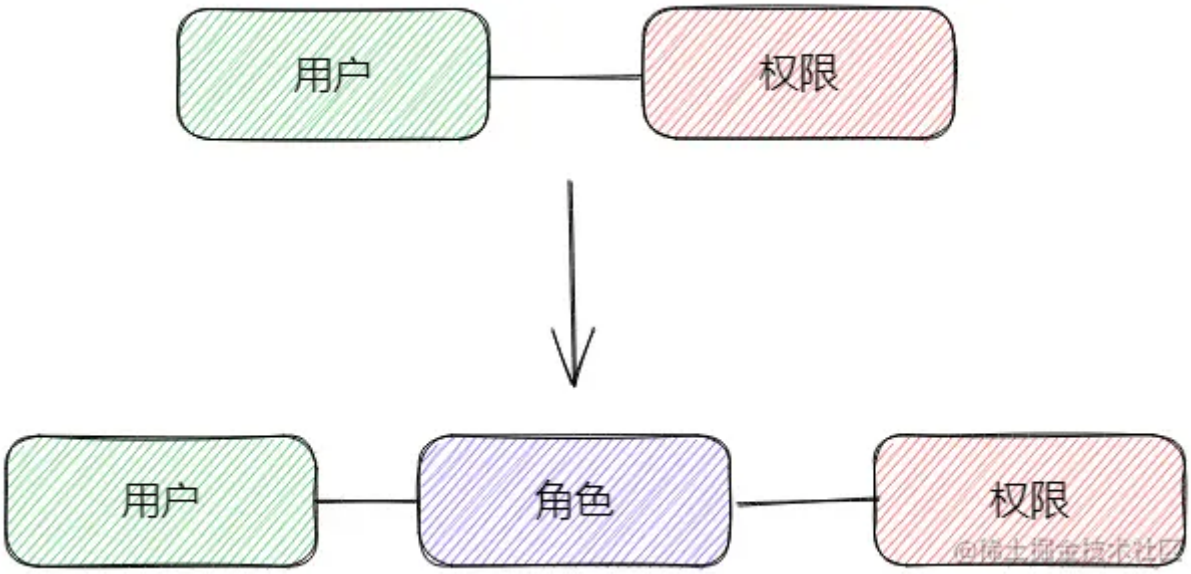
RBAC 模型发展历程

`RBAC` 全称 `Role-Based-Access-Control` ， 即基于角色的访问控制。它的发展主要有四个模型： `RBAC0` 、 `RBAC1` 、 `RBAC2` 、 `RBAC3` 下面给大家详细介绍。

基础模型 - RBAC0

假设我们去设计一个权限系统，一定会有最基础的需求：给用户分配权限，一个用户可以被分配多个权限。其实这就是最简单的权限模型了，但有什么问题呢？

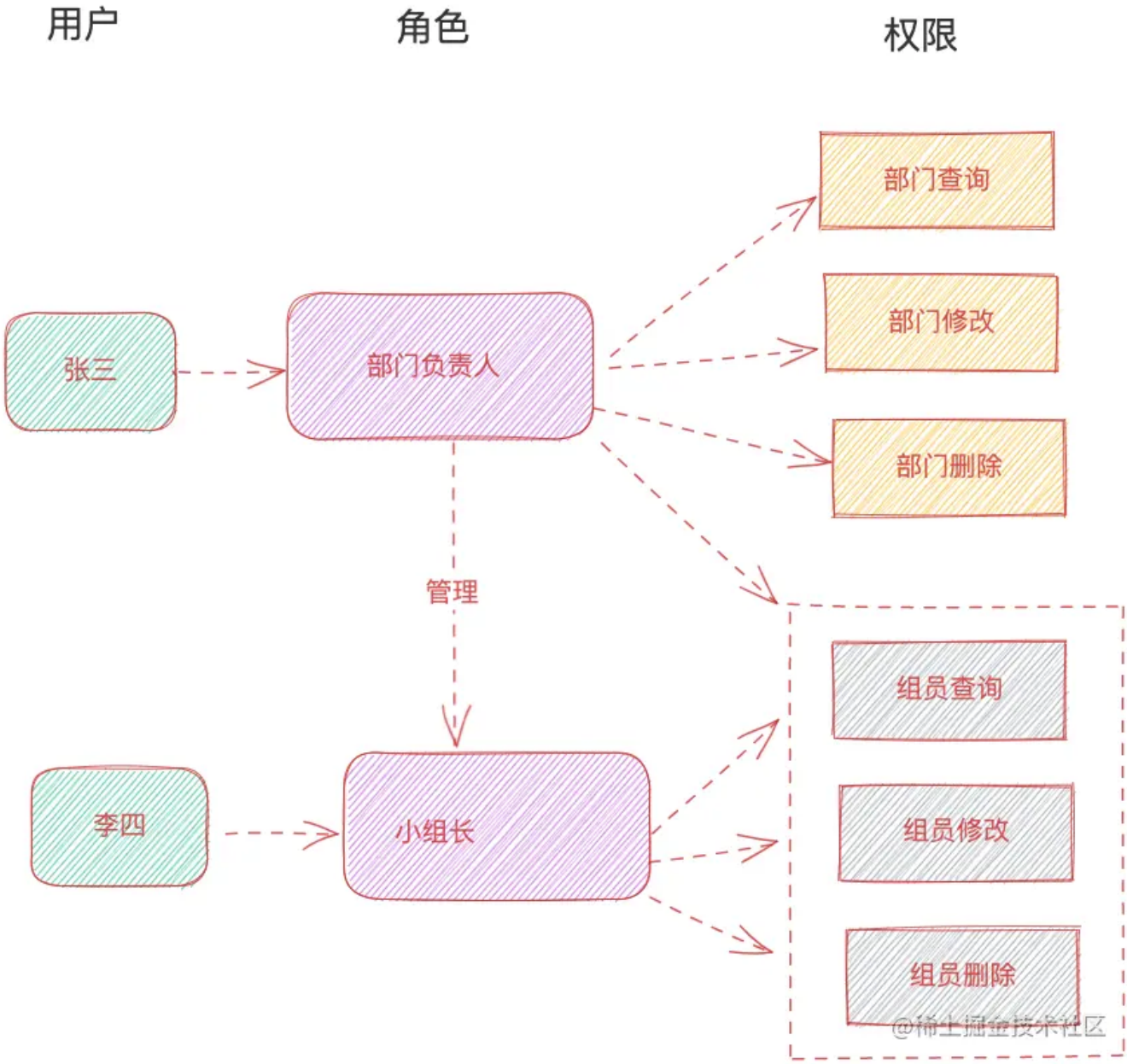
这种模型在只有几十个人的系统会很好维护，但对于具有成千上万用户的系统来说这就是一个灾难了，维护很困难。于是便在用户和权限之间引入了 **角色** 这个概念，这样只需要把权限授予给角色，不同的角色有不同的权限，而用户只需要关联某个角色就能完成整个流程了。



上述 **用户-角色-权限** 之间的关系其实就是 `RBAC` 最基础的模型了，它和 `Linux` 中的 **用户-用户组-权限** 很像，这样的好处就是可以批量调整权限了，比如某个角色某天需要添加别的权限时，只需要给角色绑定这个权限即可，而那些与这个角色关联的用户也便有了这个权限了。

角色继承 - RBAC1

`RBAC0` 是最为经典且简单的模型，也是目前最通用的模型。但是对于复杂一点的业务这个模型就需要调整一下了，比如公司组织架构划分，这里面涉及到 **上下级** 的关系，而且高级别角色会向下兼容拥有低级别角色的权限，于是便有了基于 **角色继承** 的 `RBAC1` 模型。



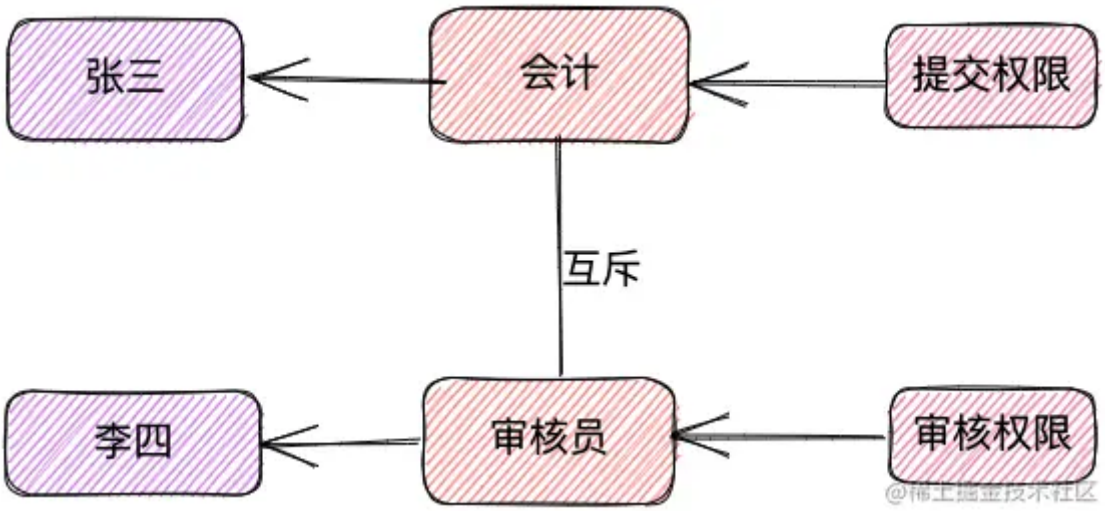
这里的角色用图表示的话其实得用树状图，这里大家心里有个部门架构图就可以啦。

这个模型非常贴近现实业务的，比如公司的部门数据是不应该给所有员工看的，部门负责人是可以看本部门所有数据的，而小组长只能操作自己小组的数据，不能操作其他小组的数据。

角色权限限制 - RBAC2

在 `RBAC1` 基础上其实已经难满足大部分业务了，但是实际生活中，难免会遇到这样的情况：

- 一个人不能 **同时** 扮演会计和审核员两个角色。
- 部门老大只有一个。
- 员工成为正式岗位前可能会经过实习期、试用期阶段，可用的权限不一样。



以上三个场景分为对应了：**角色互斥性、角色唯一性、角色先决条件** 等特征，这正是 `RBAC2` 模型的内容。这个模型的好处就是比如在使用某个系统的时候，不同角色操作同一份数据时可能存在冲突，此时需要限定同一时间只能使用一个身份进行数据操作。

最全的模型 - RBAC3

`RBAC3` 它整合了前面三种模型，形成了一个十分完整的权限管理模型，既有等级分层也有角色约束，比如最常见的公司组织架构。

新的 RBAC - 基于资源的访问控制

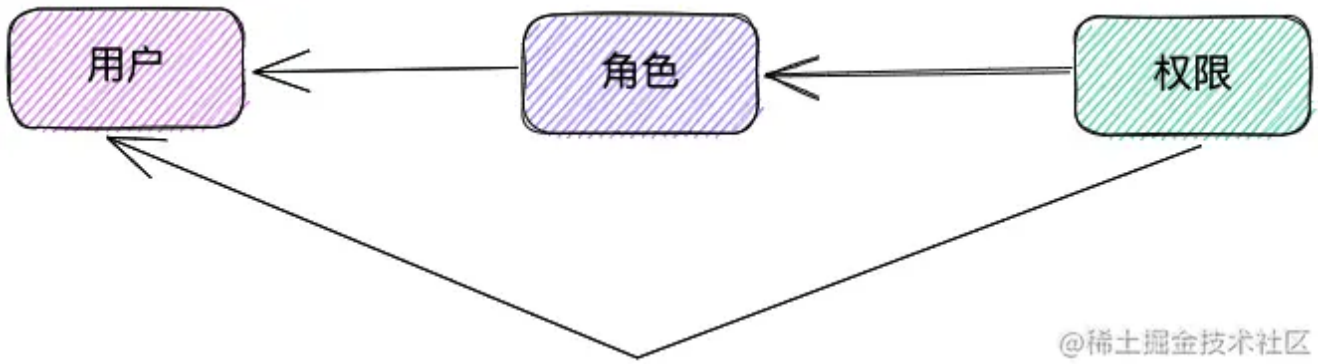
传统的 `RBAC` 会有一个问题，用户的想拥有某些权限必须先赋予对应的角色，当用户权限发生变化时必须通过添加一个角色来应对变化，但是这样在编码上就不好维护了。比如部门管理中，部门经理可以修改某些数据，用伪代码可以这么判断：

```js
if (user.hasRole('manager')) {
    修改数据
}
```

当业务变更，组长也可以修改数据时：

```js
if (user.hasRoles(['manager', 'team_leader'])) {
    修改数据
}
```

这样代码维护就成了问题，所以基于资源的权限访问控制模型（ Resource-Based-Access-Control ）就流行了起来，在传统模型基础上，它让用户也可以直接关联权限，这样就更加灵活了。



还是拿上面的场景来说，现在直接关联权限后：

```js
if(user.hasPermission("修改权限标识")) {
    修改数据
}
```

这种基于权限点判断的方式更加灵活，这样用户从部门经理变更为组长的身份后，不需要修改代码，代码的拓展性强。像很多后端的方案其实基于这种方式去做的，比如很经典的 shiro 权限系统。

总结

前面主要给大家介绍了 RBAC 的几个传统模型，现在做个总结：

- RBAC0 ：最基础的模型，后面所有模型都会基于这个去演进。它的特点是非常简单且流程简单，但是对于复杂的业务比较难以进行细粒度的控制。
- RBAC1 ：基于 RBAC0 ，引入角色继承概念，即角色存在了上下级关系，高等级向下兼容低等级角色拥有的权限，最常见的比如 **部门负责人 - 部门经理 - 组长 - 组员** 之间的关系。
- RBAC2 ：在 RBAC1 基础上给角色增加了一定的限制，比如 **角色互斥性、角色唯一性、角色先决条件** 等特征，更加细粒度地去控制角色权限。
- RBAC3 ：最全的权限模型，企业级模型。

这些模型很优秀，但是实际业务千变万化，有时候业务简单可能根本用不上这么庞大的权限模型，比如小公司就十几号人完全可以采用前文最开始提到的 **用户 -> 权限** 模型。而对于特别复杂的业务，我们有必要根据实际情况去调整策略，比如可以从开发成本、效率、公司组织架构去思考。

当然，目前比较流行的方式是：用户可以拥有角色的权限，也可以直接配置权限点，可以适应不同的业务需求。

小结最后提个疑问：前文说到 **用户 -> 角色 -> 权限** 解决了单独管理单个用户权限问题，可是用户与角色之间其实也是多对多关系，那么来一个用户就要授予一个角色，有成千上万个用户该怎么办？

> 提示：用户组。`RBAC` 和 `Linux` 中的 **用户-用户组-权限** 模型很像，假如大家对这个有所了解，对理解权限模型很有帮助。大家有兴趣可以参考我之前写的这篇文章 # Linux用户管理与权限那些事 。

参考

- # 权限系统设计
- # RBAC 权限设计实战 - 用户权限设计从入门到精通