**RBAC权限模型设计**

1. RBAC概念

RBAC即基于RBAC角色的用户权限控制系统。

* 基本模型RBAC0模型

定义：RBAC0模型由以下描述确定：

U、R、P、S分别表示用户集合、角色集合、许可权集合和会话集合。

PA P×R表示许可权与角色之间多对多的指派关系。

UA U×R表示用户与角色之间多对多的指派关系。

用户：S→U每个会话si到单个用户user(si)的映射函数（常量代表会话的声明周期）。

角色：S→2每个会话si到角色子集roles(si) {r|user(si, r')∈UA}（能随时间改变）的映射函数，会话si有许可权Ur∈roles(si){p|(p,r')∈PA}。

在使用RBAC0模型时，应该要求每个许可权和每个用户至少应该被分配给一个角色。两个角色被分配的许可权完全一样是可能的，但仍是两个完全独立的角色，用户也有类似情况。角色可以适当的被看做是一种语义结构，是访问控制策略形式化的基础。

RBAC0把许可权处理未非解释符号，因为其精确含义只能由实现确定且与系统有关。RBAC0中的许可权只能应用于数据和资源类客体，但不能应用于模型本身的组件。修改集合U、R、P和关系PA和UA的权限称为管理权限，后面将介绍RBAC的管理模型。因此，在RBAC0中假定只有安全管理员才能修改这些组件。

会话是由单个用户控制的，在模型中，用户可以创建会话，并有选择的激活用户角色的某些子集。在一个会话中的角色的激活是由用户来决断的，会话的终止也是由用户初始化的。RBAC0不允许由一个会话去创建另一个会话，会话只能由用户创建。

* 角色分级模型RBAC1

定义：RBAC1由以下内容确定

U、R、P、S分别表示用户集合、角色集合、许可权集合和会话集合。

PA P×R表示许可权与角色之间多对多的指派关系。

UA U×R表示用户与角色之间多对多的指派关系。

RH R×R是对R的偏序关系，称为角色等级或角色支配关系，也可用≥符号表示。

用户：S→U每个会话si到单个用户user(si)的映射函数（常量代表会话的声明周期）。

角色：S→2每个会话si到角色子集roles(si) {r|(r'≥r)[user(si,r')∈UA]}（能随时间改变）的映射函数，会话si有许可权Ur∈roles(si){p|(r''≤r)[(p,r'')∈PA]}。

* 限制模型RBAC2

RBAC2模型是在RBAC0模型增加限制后形成的，它与RBAC1并不兼容。RBAC2定义如下：

定义：除了在RBAC0中增加了一些限制因素外，RBAC2未加改变的来自于RBAC0，这些限制是用于确定RBAC0中各个组件的值是否是可接受的，只有那些可接受的值才是允许的。

RBAC2中引入的限制可以施加到RBAC0模型中的所有关系和组件上。RBAC2中的一个基本限制时互斥角色的限制，互斥角色是指各自权限尅一互相制约的两个角色。对于这类角色一个用户在某一次活动中只能被分配其中的一个角色，不能同时获得两个角色的使用权。

例如，在审计活动中，一个角色不能同时被指派给会计角色和审计员角色。又如，在公司中，经理和副经理的角色也是互斥的，合同或支票只能由经理签字，不能由副经理签字。在为公司建立的RBAC2模型中，一个用户不能同时兼得经理和副经理两个角色。模型汇总的互斥限制可以支持权责分离原则的实现。

更一般化而言，互斥限制可以控制在不同的角色组合中用户的成员关系是否是可接受的。例如，一个用户可以既是项目A的程序员，也可以是项目B的测试员和项目C的验收员，但他不能同时成为同一个项目中的这3个角色。RBAC2模型可以对这种情况进行限制。

另一个用户指派限制的例子是一个角色限制其最大成员数，这被称为角色的基数限制。例如，一个单位的最高领导只能为1人，中层干部的数量也是有限的，一旦分配给这些角色的用户数超过了角色基数的限制，就不再接受新配给的用户了。

限制角色的最小基数实现起来有些困难。例如，如果规定占用某个角色的最小用户数，问题是系统如何在任何时刻都能知道这些占用者中的某个人没有消失，如果消失的话，系统又应该如何去做。

在为用户指派某个角色A时，在有的情况下要求该用户必须是角色B的一个成员，B角色成为角色A的先决角色。先决角色（PrerequisiteRoles）的概念来自于能力和适应性。对先决绝对的限制成为先决限制。一个通俗的例子是，一个数学副教授应该从数学讲师中提拔，讲师是任副教授的先决角色。但在实际系统中，不兼容角色之间的先决限制的情况也会发生。

在图ap08-03中，可以限制只有本项目的成员才有资格担任程序员的角色，通常在一个系统中，先决角色比新指派的角色的级别要低一些。但有的情况下，却要求只有当用户不是某个特殊角色时，才能担任另一个角色A。如，需要执行回避策略时需要这样做，例如，本课题组成员不应当是本项目成果鉴定委员会的成员。这类限制也可以推广到许可权方面。

由于用户与角色的作用会与会话联系在一起，因此对会话也可以施加限制。例如，可以允许一个用户被指派给两个角色，但不允许在同一时间内把该用户在两个角色中都激活。另外，还可以限制一个用户在同一时间内可以激活的会话的数量，相应的，对该用户所激活的会话中所分配许可权的数量也可以施加限制。

前面提到的继承概念也可以视为一种限制。被分配给低级别角色的权限，也必须分配给该角色的所有上级角色。或等价的，一个指派给较高级别的角色的用户必须指派给该角色的所有下级角色。因此从某种角度上讲，RBAC1模型是冗余的，它被包含在RBAC2中。但RBAC1模型比较简洁，用继承代替限制可使概念更清晰。

实现时可以用函数来实现限制，当为用户指定角色或为角色分配权限时就调用这些函数进行检查，根据函数返回的结果决定分配是否满足限制的要求，通常只对那些可被有效检查和那些惯例性的一些简单限制给与实现，因为这些限制可以保持较长的时间。

模型中的限制机制的有效性建立在每个用户只有唯一标识符的基础上，如果一个实际系统支持用户拥有多标识符，限制将会失效。同样，如果同一个操作可以有两个以上的许可权来比准，那么，RBAC系统也无法实施加强的基本限制和责任分离和限制。因此要求用户与其标识符，许可与对应的操作之间一一对应。

* 统一模型RBAC3

RBAC3把RBAC1和RBAC2组合在一起，提供角色的分级和继承的能力。但把这两种概念组合在一起也引起一些新问题。

限制也可以应用于角色等级本身，由于角色间的等级关系满足偏序关系，这种限制对模型而言是本质性的，可能会影响这种偏序关系。例如，附加的限制可能会限制一个给定角色的应有的下级角色的数量。

两个或多个角色由可能被限制成没有公共的上级角色或下级角色。这些类型的限制在概念角色等级的权力已经被分散化的情况下是有用哦，但是安全主管却希望对所有允许这些改变的方法加以限制。

在限制和角色的等级之间也会产生敏感的相互影响。在图ap08-03的环境中，一个项目成员不允许同时担任程序员与测试员的角色，但项目管理员所处的位置显然是违反了该限制。在某种情况i下由高等级的角色违反这种限制是可接受的，但在其他情况下又不允许这种违反现象发生。

从严格性的角度来讲，模型的规则不应该是一些情况下不允许而在另一情况下是允许的。类似的情况也会发生在对基数的限制上。假定限制一个用户至多能分配给一个角色，那么对图中的测试员的一个指派能够未被这种限制吗？换句话说，基数限制是不是只能用于直接成员，是否也能应用于继承成员上？

私有角色的概念可以说明这些限制是有用的。同样在图ap08-03的环境中，可以把测试员'，程序员'和项目管理员3个角色说明为互斥的，它们处于同一等级，没有共同的上级角色，所以管理员角色没有违反互斥限制。通常私有角色和其他角色之间没有公共上级角色，因为它们是这个等级的最大元素，所以私有角色之间互斥关系可以无冲突的定义。

诸私有角色之间的相同部分可以被说明为具有0成员的最大技术限制。根据这种方法，测试员必须被指派给测试员'这个角色，而测试员角色就作为与管理员角色共享许可权的一种工具。

1. 相关应用
   1. HoloStudio



使用HoloLens可以实现真正的3D设计，因为你的视野就是真实世界，在这里面进行3D设计是最现实不过的了。HoloStudio正好利用这项优势来展现出3D设计的魅力，让用户切身体会到自己设计的成果。从宣传片来看，应用界面和设计结果都是偏卡通风格，似乎并非专业方向。

这款应用可轻松上传成果至OneDrive，并且也可以方便地扩展进行3D打印和Sketchfab

<http://v.youku.com/v_show/id_XMTQ4NzUzNDcwNA==.html>

* 1. 微软HoloLens展示AutoDesk Maya三维建模

<http://v.youku.com/v_show/id_XMTI4NDM0OTQ4NA==.html>

* 1. 与Autodesk合作

联合Autodesk的Fusion360，推出基于HoloLens的3D产品设计解决方案。

1. 最近动态
   1. 开发版对外发售

所有“开发者和商业客户”在美国和加拿大从现在开始每人最多可以订购5台HoloLens开发套件。

1. 总体概述

首先，项目开发我们选用了DDD开发思想，将领域模型切实的应用到此次开发中，摒弃了一贯的贫血模型。使模块之间的耦合度更低，更易扩展。

所谓贫血模型就是模型对象之间存在完整的关联(可能存在多余的关联)，但是对象除了get和set方外外几乎就没有其它的方法，整个对象充当的就是一个数 据容器，用C语言的话来说就是一个结构体，所有的业务方法都在一个无状态的Service类中实现，Service类仅仅包含一些行为。

与贫血模型相反，领域模型要承担关键业务逻辑，业务逻辑在多个领域对象之间分配，而Service只是完成一些不适合放在模型中的业务逻辑，它是非常薄的一层，它指挥多个模型对象来完成业务功能。

1. 框架简介

* spring
* springMVC
* mybatis

1. 架构设计

面向接口编程：

1. 模块设计
   1. Domain领域
      1. Model 模型

领域模型。只在domain层可以引用，与其他层完全解耦和，通过viewModel与外界交互。

* + 1. Repository 仓储

首先定义了一个通用的仓储接口IRepository，里边对应mapper和常规实现定义了一些通用的方法。然后根据Model和ViewModel生成对应的模型接口并继承IRepository，最后在模型对应的接口中根据需要添加自己特有的方法。

* 仓储接口方法命名

|  |  |
| --- | --- |
| Insert | 保存一个实体的方法 |
| update | 编辑更新实体的方法 |
| findById | 通过id查询实体 |
| findAll | 查询全部 |
| delete | 删除方法 |

* + 1. Service 服务

Service服务分为commandService和queryService两部分。

CommandService包括接口和实现类两部分，主要是针对model模型编写对应的接口类，因model对于外界其它模块是处于一种保密状态，所以在commandService实现类我们需要调用model，viewModel和model对应的repository接口。外界接受到和返回的都是ViewModel模型，利用dozer在方法内部进行viewModel和model的转化，再调用repository进行相应的处理，利用mapper自动更新数据库，必要的话，再次利用dozer在方法内部进行viewModel和model的转化，将符合要求的信息返回给外界。

queryService相对简单些，其内部也包括接口和实现类两部分，内部直接对viewModel进行相应的操作，无需模型的转化，从外界接收和返回给外界的都是ViewModel，直接调用queryRepository接口对ViewModel进行相应的处理，将符合要求的信息返回给外界。

* + 1. ViewModel 视图模型

视图模型是字段与数据库对应的entity，为了方便外界对视图模型的引用，利用dozer中的mapper注释将model和viewModel进行字段对应，从而达到类之间可相互转换的效果。

* + 1. Type 枚举类型

一些特定的枚举类，类中定义了相应的枚举类型。如RoleType类中定义了Annonymous（匿名角色类型）、administrator（管理员角色类型）、general（普通角色类型）等。这样做的优点是方便修改角色类型，实现了代码容易扩展功能。

* 1. Application应用

Application层主要包含两个部分：commandApplication和queryApplication。为web层的controller提供服务接口。

CommandApplication调用commandService，内部实现增、删和改的方法。如insert、update、delete等。

QueryApplication调用queryService，内部实现查询方法。如findById、findAll、findByName等。

* 1. Web 表示层
     1. Controller

Web API 。调用Application层的接口，给页面访问提供接口。

* + 1. View

包括html，js，css等

* + 1. Mapper

实现数据持久化，