RBAC0模型的学习心得

RBAC0模型简介：

RBAC0是RBAC系列模型中的基础模型，随后的RBAC系列模型都将按照RBAC0基础模型进行改进，所以学习RBAC模型就必须要牢牢掌握RBAC0模型的思想以及使用方法。

RBAC0引入了角色的思想解决了用户与权限之间的分离问题。用户其实质上并没有真正的权限，只有当用户成为了某一个特定的角色的时候才会被赋予特定的权限。可以这么理解，一个普通人，在正常情况下是无法合法持有枪支的，但当这个普通人参加公务员考试，并成为一名刑警之后，他就可以合法持有枪支了。这里普通人为用户，持有枪支为权限，二者并没有直接的关系，而是通过警察这一角色来进行关联，将持有枪支合法化。换句话说，这个普通人永远不能持有枪支，而警察永远拥有持有枪支的权限。每一个用户都有可能成为警察，从而拥有持有枪支的权限。这就是RBAC0中主要解决的权限分离问题。

我们通过为某一用户赋予角色来让用户拥有特定的权限。这里用户、角色、权限三者都可以是多个的。

数据库模型：

数据库表中需要有USER（用户表）、ROLE（角色表）、PERMISSION（权限表）、PERMISSION\_ASSIGNMENT（简称PA）(权限与角色关联表)、USER\_ASSIGNMENT（简称UA）(用户与角色关联表)，一共五张基础表构成RBAC0模型。

当我们为某一用户指定角色时我们就需要用到UA表了，管理员为用户赋予角色时将uid和rid同时提交到PA表。系统在进行用户角色判断是，则从UA表中抽取其特定uid下的rid。不同的用户可以成为相同的角色，而不同的角色同样也可以被赋予给同一个用户。举例，网站中有用户：A、B、C，网站中有角色：普通用户、管理员、超级管理员。A、B、C都被赋予了普通用户的权限，此时，A又被赋予了管理员权限来处理网站的大小事务，则A现在同时拥有普通用户和管理员两个角色。而经过一段时间后，超级管理员发现A工作忙不过来了，于是将B也提升为管理员。此时，管理员角色同时被赋给了A和B二人。

在为用户赋予角色的同时，我们也要为角色赋予权限。每一个角色可以指定不同的权限，且权限赋予是从0到无穷大。也就是说可以为同一个角色赋予多个权限。而同一个权限也可以赋予给多个角色，所以PA也是多对多的关系。比如角色普通用户与管理员，他们同时拥有访问网站并浏览信息的权限，所以同样的一个浏览权限被赋予给了不同的角色。而管理员不仅能看信息，还能够对信息进行编辑、删除。所以管理员同时拥有浏览、编辑、删除三种不同的权限。

系统对用户进行权限判断的主要流程是，一个用户登陆到网站，网站利用UA对用户进行角色判断并将用户拥有的角色结果保存为SESSION，当用户进行操作时，网站将SESSION中的角色与PA进行比对然后对用户所拥有的角色进行权限判断。

用户进行权限操作的流程：

1. 对访问客户端进行登陆判断，判断是否登陆成功，如果不成功则提示用户没有登陆。
2. 用户成功登陆后，系统将匹配成功的用户ID从USER表中取出，并与UA表对应的UID进行比对，罗列出符合条件的所有rid并将其作为一个SESSION保存。
3. 当用户进行特定权限操作时，系统用户提交的pid与数据库中的与数据库中的rid进行匹配，然后再判断用户SESSION中是否有匹配的值，如果有则赋予用户相应的权限，并提示操作成功，否则提示用户没有权限。

授权者如何为用户进行权限赋予：

1. 授权者自身首先需要拥有赋予其他用户权限的能力，且能赋予用户的权限能且只能赋予与自身权限相当的权利，无法赋予用户高于授权者自身权限的能力。例如对网站数据库进行操作的权限只有顶级授权者有，普通授权者没有这样的权限，那么同样的，普通授权者也无法赋予用户这样的权限。
2. 首先，授权者要对角色进行权限赋予，系统判断授权者是否拥有为用户赋予权限的权利。
3. 当授权者有为用户赋予权限的能力时，显示一个表单，罗列出所有比授权者权限小的角色，并罗列出授权者自身所有权限，如果赋予用户的权限，不在用户现在拥有的角色权限列表中，则添加一个符合条件的新角色给用户。

关于在RBAC0中“角色管理员”角色的控制问题

首先，如果系统中需要引入角色管理员这一角色协助顶级管理员一起管理角色的话，我们需要考虑并遵循至少以下几个要点：

1. 角色管理员角色的父角色必须是顶级管理员，直接受顶级管理员控制
2. 使用角色管理员这一角色的用户必须为可信赖的用户
3. 角色管理员无法创建子角色（可选）
4. 角色管理员必须严格遵循角色继承思想，角色管理员无法对顶级管理员进行操作
5. 角色管理员只能对UA/PA/ROLE三表的完全或者不完全的控制权，对USER/PERMISSION表没有任何操作权限

关于角色继承的问题：

随着系统中用户、角色、权限不断增加，我们也许会考虑让角色所有者自有派生其子角色的权限，这里我们就需要注意到一下几个问题：

1. 我们可以对角色继承进行强行控制，不是每一个角色都有派生其子角色的权限，解决方案，在角色中创建一个“角色继承”或者其他什么名字的角色都可以，为这个角色添加唯一一个权限，角色继承权限，我们应该在PA以及PERMISSION表中添加者一权限的存在，当需要的时候我们为某一角色添加这一特别的角色就可以实现对角色继承的严格控制。当然“角色继承”这一角色我们也不是非要创建，可以在每次添加角色的时候为该角色赋予继承权限也可以了。但是由于权限较多，所以有时候我们往往在删除某一角色这一权限的时候会疏忽，造成安全问题，所以我个人认为添加一个特定的专用于继承的权限是有必要的。
2. 子角色必须要遵循几个要点，1.无法对父角色的权限，行为，包括状态进行浏览或者干涉，2.无法或者超出父角色权限的角色，3.有且只有一个父角色，且当被父角色创建并继承后，无法删除父子关系，除非删除该子角色。
3. 谁添加了这个角色就是这个角色的父角色，反应到数据库表中就是超级管理员的ID为1，PID为0，添加了“管理员”这样一个角色，那个这个角色的ID为2，PID为1，所以在角色表中的PID为0的有且只有一个。PID为每一个角色在创建是就必须要有的属性，并且有且只有一个。
4. 同一个用户不可能既有父角色也有子角色，这样的道理就像一个人不可能自己给自己当爹或者当儿子是一样的。

RBAC思想关键点：

1. PERMISSION表中的fname(function name,类方法名)才是最后真正实现对权限控制的一个要素，但他不一定要有。
2. RBAC中的权限控制，其实可以被理解为是对系统中各个类以及类中方法，参数的一个控制思想，比如：

文章管理（class article{}）

|-添加文章（public function addArticle () {}）

|-删除文章（public function delArticle () {}）

|-是否显示作者（public $author）

对整个类进行精确控制，可以让我们的系统安全性提高到一个新的级别，也就是任何一个用户必须要有特定的角色，才可以访问到特定的权限，而这个权限可能必须要满足在比如必须要满足是在article类中的addArticle方法才可以，如果是换了其他的地方则没有，这样我们也可以对类中方法的命名进行一个简化，比如：

Class article {

Public function add () {}

Public function del () {}

Public function show () {}

}

Class user {

Public function add () {}

Public function del () {}

Public function show () {}

}

两个类中都有add/del/show的方法，但是他们的意义其实是不同的，所以我们通过对访问类的控制达到对方法的控制，比如我们允许“编辑”这个角色去访问article类中的三个方法，但是我们不允许他进行对用户的任何操作，所以，我们就可以首先限制他访问的类，然后再对其访问的类方法进行一个准确控制，甚至，我们可以对他的类成员进行一个准确控制，比如我们有一个类成员，是public $modifyTime专门存放修改时间的参数，那么这个参数“编辑”不能访问到，我们就可以在权限管理下面添加上modifyTime在这个文章类下面，然后不赋予给“编辑”就可以让编辑没有权限去操作这个成员了。而这个成员在“编辑”进行文章修改的时候会使用time()自动创建，所以就有效的防止了权限的滥用，也将权限最小化升级到了成员访问控制最小化。当然这个例子也许并不怎么恰当，但是有这样的思想至少是对的，也许在不一定的某一个时候就需要用到这样的极小权限控制。当然访问权限的控制不是一定要使用面向对象进行编程才可以实现，我们也可以使用面向过程，只要对文件进行访问控制就可以实现面向对象同样的效果，但是考虑到当一个系统了RBAC模型的时候，大概已经不是一个小系统了，所以运用面向对象的编程方法更加恰当。

1. 同一个拥有两个或两个以上角色时，如果这两个角色的权限重复是允许的，这并不能造成对某一个角色的权限限制。比如当“文章编辑”角色拥有查看添加删除修改文章的权限的时候，“用户角色”拥有查看文章的权限，且test1用户同时拥有者两个角色，那么他在UA表中应该是 UA{test1:文章编辑;test1:用户}在PA表中的呈现时PA{文章编辑：show;文章编辑：add；文章编辑：edit;文章编辑：del;用户:show}。而并不能减掉{文章编辑:show;}或者{用户:show}中的其中任何一个，因为这涉及到今后角色的增删。当test用户的用户权限没有的时候，也应该保证他能够正常查看文章。而如果当我们在为其赋予权限的时候看到用户角色已经有了查看文章的权限，而不将文章编辑中的show权限赋予给用户的时，那么将来就会产生一个严重的问题，当test1的“用户”角色被删除，随之而来的是即使test1是文章编辑的角色也会狗血的发现自己能对文章进行任何操作，但就是无法正常得查看文章。
2. 角色间权限不允许交叉继承，只能垂直继承，所有平级角色必须保持权限平行。
3. 超级管理员具有唯一性，ID必须锁死，且每一个系统中必须存在超级管理员。