ThinkPHP 知识系列

ThinkPHP Framework 1.0
RBAC

ThinkPHP 1.0
基于 RBAC 的权限访问控制设计

编写：ThinkPHP 文档组

最后更新：2008-06-24

目录

1 概述

RBAC 是指基于角色的访问控制，本篇主要描述了 ThinkPHP 的 RBAC 设计思想和使用实例，给出了一个统一和方便的权限控制解决方案。

2 安全体系

要了解权限访问控制 ,我们必须了解下什么是安全体系。企业级的应用是离不开安全保护的 ,ThinkPHP 以基于 Spring 的 Acegi 安全系统作为参考原型，并做了简化，以适合目前的 ThinkPHP 结构，提供了一个多层的、可定制的安全体系来为应用开发提供安全控制。安全体系中主要有：

- 安全拦截器
- 认证管理器
- 决策访问管理器
- 运行身份管理器

安全拦截器

安全拦截器就好比一道道门，在系统的安全防护系统中可能存在很多不同的安全控制环节，一旦某个环节你未通过安全体系认证，那么安全拦截器就会实施拦截。

认证管理器

防护体系的第一道门就是认证管理器，认证管理器负责决定你是谁，一般它通过验证你的主体（通常是一个用户名）和你的凭证（通常是一个密码），或者更多的资料来做到。更简单的说，认证管理器验证你的身份是否在安全防护体系授权范围之内。

访问决策管理

虽然通过了认证管理器的身份验证，但是并不代表你可以在系统里面肆意妄为，因为你还需要通过访问决策管理这道门。访问决策管理器对用户进行授权，通过考虑你的身份认证信息和与受保护资源关联的安全属性决定是是否可以进入系统的某个模块，和进行某项操作。

例如，安全规则规定只有主管才允许访问某个模块，而你并没有被授予主管权限，那么安全拦截器会拦截你的访问操作。

决策访问管理器不能单独运行，必须首先依赖认证管理器进行身份确认，因此，在加载访问决策过滤器的时候已经包含了认证管理器和决策访问管理器。

为了满足应用的不同需要，ThinkPHP 在进行访问决策管理的时候采用两种模式：登录模式和即时模式。

登录模式，系统在用户登录的时候读取改用户所具备的授权信息到 Session，下次不再重新获取授权信息。也就是说即使管理员对该用户进行了权限修改，用户也必须在下次登录后才能生效。

即时模式就是为了解决上面的问题，在每次访问系统的模块或者操作时候，进行即使验证该用户是否具有该模块和操作的授权，从更高程度上保障了系统的安全。

运行身份管理器

运行身份管理器的用处在大多数应用系统中是有限的，例如某个操作和模块需要多个身份的安全需求，运行身份管理器可以用另一个身份替换你目前的身份，从而允许你访问应用系统内部更深处的受保护对象。这一层安全体系目前的 RBAC 中尚未实现。

3 认证过程

对应上面的安全体系，ThinkPHP 的 RBAC 认证的过程大致如下：

1. 判断当前模块的当前操作是否需要认证
2. 如果需要认证并且尚未登录，跳到认证网关，如果已经登录 执行 5

3. 通过委托认证进行用户身份认证

4. 获取用户的决策访问列表

5. 判断当前用户是否具有访问权限

4 认证设置

我们要进行安全拦截，就必须要启用用户认证，要启用 RBAC 请在项目配置文件中设置

USER_AUTH_ON 为 True

并设置认证类型 USER_AUTH_TYPE

1 普通认证（认证一次 通常是在登录后认证）

2 高级认证（实时认证）

不设置默认为 1

例如，下面的设置可供参考

```
return array(
'USER_AUTH_ON'=>true,
'USER_AUTH_TYPE'    => 1,
'USER_AUTH_KEY'    => 'authId',
'USER_AUTH_PROVIDER' => 'DaoAuthentictionProvider',
'USER_AUTH_GATEWAY' => '/Public/login',
'NOT_AUTH_MODULE'   => 'Public',
'REQUIRE_AUTH_MODULE'=> '',
);
```

认证识别号 USER_AUTH_KEY 是用于检查用户是否经过身份认证的标识，一旦用户经过系统认证，系统会把该用户编号保存在$_SESSION[C('USER_AUTH_KEY')]中

为了满足应用系统的需要，RBAC 插件中可以设置

REQUIRE_AUTH_MODULE 需要认证的模块

REQUIRE_AUTH_ACTION 需要认证的操作

NOT_AUTH_MODULE 无需认证的模块

NOT_AUTH_ACTION 无需认证的操作

通常需要认证和无需认证只用根据应用的需求配置一个即可，例如：

```
'USER_AUTH_GATEWAY' => '/Public/login',
'NOT_AUTH_MODULE'  => 'Public',
'REQUIRE_AUTH_MODULE'=> '',
```

多个模块之间用逗号分割 ，如果某个模块需要认证，但是用户还没有经过身份认证，就会跳转到 USER_AUTH_GATEWAY 认证网关， 经过这样的配置后 ,我们访问 http://localhost/MyApp/index.php 地址后 ,因为默认执行的是 Index 模块的 index 操作 ,而系统设置了除了 Public 模块以外都需要认证，所以会跳转到我们设置的认证网关 http://localhost/MyApp/index.php/Public/login/

当然，我们可以反过来配置，例如我们配置 Member 模块和 Order 模块必须认证，其他可以不认证

```
'REQUIRE_AUTH_MODULE'=> 'Member,Order'
'NOT_AUTH_MODULE'  => '',
```

这样的话，只有当执行 Member 模块或者 Order 模块的操作的时候才会需要认证。

那有没有可能对某些操作进行单独认证呢，答案是肯定的。我们可以设置

```
'REQUIRE_AUTH_ACTION'=> 'insert,update,delete'
'NOT_AUTH_ACTION'  => '',
```

这样，只有当要执行 insert、update 或者 delete 操作的时候才会进行认证，操作认证可以和模块认证配合起来使用，达到最精密的认证控制。

```
'USER_AUTH_PROVIDER' => 'DaoAuthentictionProvider',
```

表示设置了用户认证的委托方式，目前是用 DAO 委托认证，这个下面会有描述。

5 委托认证

进行用户身份认证的方法有很多，ThinkPHP 采用委托认证的方法，以方便扩展其他方式，目前提供的是基于 DAO 的认证方式，通过查询数据库的相关信息来进行用户身份认证，这也是最常用的一种认证。其他的认证方法还包括基于 LADP（轻量级目录访问协议）的认证，更加高级的包括基于 CA（数字证书认证中心）的认证。下面我们讲述下 ThinkPHP 默认的 DAO 委托认证。

实际上，ThinkPHP 的 DAO 委托认证是通过 DaoAuthentictionProvider 委托认证类来实现的，但是我们并不需要直接操作该类，而是直接使用 RBAC 类的委托认证方法静态方法 authenticate 就可以了。

```
authenticate($map,$model='User',$provider=USER_AUTH_PROVIDER)
```

方法是静态方法，支持三个参数，其中第一个认证条件$map 是必须的，可以灵活地控制需要认证的字段。

第二个参数是进行认证的模型类，默认是 UserModel 类

第三个参数是委托方式 由 USER_AUTH_PROVIDER 设置委托认证管理器的委托方式，目前支持的是 DaoAuthentictionProvider 通过数据库进行认证。

6 访问决策

在上面的委托认证成功后，我们使用了下面的代码：

```
//获取并保存用户访问权限列表
RBAC::saveAccessList();
```

这个时候其实是获取当前登录用户对应的权限列表信息，系统会把用户的权限列表保存在 SESSION 里面，格式类似于

```
$_SESSION['AppName']['ModuleName']['ActionName']
```

如果存在这样一个数据，就表示用户具有 AppName 的 ModuleName 模块的 ActionName 操作的访问

权限，否则就表示没有权限。如果有些模块并没有定义操作，那么就会读取 Public 模块的操作权限。

在应用系统的开发过程中，只需要设置相关的配置项和添加上面的认证方法，其他的认证和决策访问就由 RBAC 组件的 AccessDecision 方法自动完成了。你其实并不需要关心系统是如何判断用户是否具有权限的，这个在传统的权限控制模式里面是需要手动来一一判断的。

7 权限分配

所谓的权限分配操作其实就是往 DB_PREFIX_groupuser 表和 DB_PREFIX_access 里面写入数据 ,来给用户增加权限信息。

RBAC 相关的类库都位于基类库的 ORG/RBAC 目录下面。授权和认证功能涉及到 5 个数据表，DB_PREFIX 为配置文件中设置的数据库前缀 。

DB_PREFIX_group 权限组表 ，参考结构：

```
CREATE TABLE `think_group` (
  `id` smallint(6) unsigned NOT NULL auto_increment,
  `name` varchar(20) NOT NULL,
  `pid` smallint(6) default NULL,
  `status` tinyint(1) unsigned default NULL,
  `remark` varchar(255) default NULL,
  PRIMARY KEY  (`id`)
)
```

也可以称之为角色表，主要管理不同的角色或者权限组，其中 pid 字段用以表示组的从属关系，也就是可以支持权限组的继承，status 字段用于标记组的状态，在其他的表中都会看到这个字段。

DB_PREFIX_user 用户表 ，参考结构：

```
CREATE TABLE `think_user` (
  `id` mediumint(8) unsigned NOT NULL auto_increment,
  `account` varchar(64) NOT NULL,
  `nickname` varchar(50) NOT NULL,
```

```
  `password` char(32) NOT NULL,
  `remark` varchar(255) NOT NULL,
  `status` tinyint(1) default '0',
  PRIMARY KEY  (`id`)
)
```

也是我们在用户认证过程中要进行验证的表 ,这个表设计的自由度比较大 ,主要是用于登录检查而已。

DB_PREFIX_groupuser 组-用户关联表 ，参考结构：

```
CREATE TABLE `think_groupuser` (
  `groupId` mediumint(9) unsigned default NULL,
  `userId` mediumint(9) unsigned default NULL,
  KEY `groupId` (`groupId`),
  KEY `userId` (`userId`)
)
```

记录了用户和组的对应关系，一个用户可以属于多个权限组，在 ThinkPHP 的权限设计思想中，单一的用户是没有任何权限的，只有加入某个权限组后，才具有该权限组对应的权限，如果属于多个权限组，就可以叠加所有权限组的权限访问信息。

DB_PREFIX_access 访问权限表 ，参考结构：

```
CREATE TABLE `think_access` (
  `groupId` smallint(6) unsigned NOT NULL,
  `nodeId` smallint(6) unsigned NOT NULL,
  `level` tinyint(1) NOT NULL,
  `parentNodeId` smallint(6) NOT NULL,
  `status` tinyint(1) default NULL,
  `module` varchar(25) default NULL,
  KEY `groupId` (`groupId`),
  KEY `nodeId` (`nodeId`),
  KEY `level` (`level`),
  KEY `parentNodeId` (`parentNodeId`)
)
```

原则上，权限表可以支持无限级别的控制，但是事实上，我们目前只是针对 ThinkPHP 框架的要求，

实现了基于项目、模块和操作的三层权限控制。其中 groupId 就是记录组 id ,nodeId 用于记录节点 id，level 用于记录节点的级别，parentNodeId 用于记录节点的从属，module 是用于记录模块信息，可以配合完成记录级别的权限访问。

DB_PREFIX_node 权限节点表，参考结构：

```
CREATE TABLE `think_node` (
  `id` smallint(6) unsigned NOT NULL auto_increment,
  `name` varchar(20) NOT NULL,
  `title` varchar(50) default NULL,
  `status` tinyint(1) unsigned default NULL,
  `remark` varchar(255) default NULL,
  `seqNo` smallint(6) unsigned default NULL,
  `pid` smallint(6) unsigned NOT NULL,
  `level` tinyint(1) unsigned NOT NULL,
  PRIMARY KEY  (`id`),
  KEY `name` (`name`),
  KEY `parentId` (`pid`),
  KEY `level` (`level`)
)
```

权限节点用于管理需要授权的节点，其中 pid 表示从属关系，level 表示节点的级别，没有定义的节点默认是没有权限访问的。

因为目前的 RBAC 类的实现和数据表是相关的，如果你的实际数据表字段有所区别，可能需要修改 AccessDecisionManager.class 类来完成。

8 实例解析

常规的方式是通过设置

```
'USER_AUTH_ON'=>true,
```

来自动开启 RBAC 权限控制，这样的做法是会在 App 的初始化过程中进行权限认证和决策，但是有些

情况下这样的设计并无法满足有些应用的特殊设计和定制需要。而我们可以用另外一种方式来手动进行认证判断，做法就是在一个公共的 Action 初始化方法里面进行权限认证判断。这种情况我们无需开启用户认证的开关，相当于和我们传统的权限控制模式有点类似。

```
public function _initialize() {
import('ORG.RBAC.RBAC');
    // 检查认证
    if(RBAC::checkAccess()) {
        //检查认证识别号
        if(!$_SESSION[C('USER_AUTH_KEY')]) {
            //跳转到认证网关
            redirect(PHP_FILE.C('USER_AUTH_GATEWAY'));
        }
        // 检查权限
        if(!RBAC::AccessDecision()) {
            $this->error('没有权限！');
        }
    }
parent::_initialize();
 }
```

在没有登录的情况下输入任何地址都会跳转到我们设置的认证网关，也就是 Public 模块的 login 操作，下图显示了 login 操作的页面：

登录后提交验证操作地址是/Public/CheckLogin，那么我们可以在 Public 模块的 checkLogin 操作方法

中采用如下方式进行认证：

```
<?php
// 生成认证 Map 条件
// 这里使用用户名、密码和状态的方式进行认证
$map = array();
$map["account"] = $_POST['name'];
$map["password"] = $_POST['password'];
$map["status"] = 1;
// 进行委托认证
$authInfo = RBAC::authenticate($map);
if(false === $authInfo) {
$this->error('登录失败，请检查用户名和密码是否有误！');
}else {
// 设置认证识别号
$_SESSION[C('USER_AUTH_KEY')] = $authInfo['id'];
//获取并保存用户访问权限列表
RBAC::saveAccessList();
// 登录成功，页面跳转
$this->success('登录成功！');
```

```
}
?>
```

这段代码或许是我们唯一需要在 Action 里面手动添加的代码，剩下的权限认证工作都由框架来自动完成。其实在这个例子里面，因为我们使用的是数据库方式的认证，所以看起来我们不使用

```
$authInfo = RBAC::authenticate($map);
```

一样也可以通过数据库的查询来进行登录检查，那为什么要使用 `RBAC::authenticate` 方法进行统一认证呢？原因很简单，当我们要使用其他委托认证方式的时候，可以无需修改代码，而直接修改配置文件就可以了。

这里的 `authenticate` 默认是使用 User 模型进行认证，如果你的模型不是 UserModel 而是 MemberModel，那么需要修改为：

```
// 进行委托认证
$authInfo = RBAC::authenticate($map,'Member');
```

这里的 Member 并不是指数据表名称，而是你的模型名称，不要混淆了。

在这里系统对超级管理员的身份做了判断：

```
if($authInfo['account']=='admin') {
  // 管理员不受权限控制影响
  $_SESSION['administrator']      =    true;
}
```

所以，当我们使用 admin 帐号登录的时候，就绕过了权限认证，因此可以进行所有的操作。这样做的目的，也是为了防止我们在授权错误后，仍然可以进入系统。我们首先要在节点管理里面添加需要进行权限管理的项目、模块和操作节点。只有定义的节点才能进行授权，并且默认情况下，如果访问了一个没有定义的模块或者操作，会认为是没有权限。我们按照顺序在节点管理里面分别添加项目、模块和操作，并且我们把一些公共的操作定义到了 Public 模块里面，这个和我们后面的授权也有关系。

首先定义项目节点，这里我们要用项目的名称，这个例子的项目名称是 RbacAdmin。

编号	名称	显示名	说明	状态	操作
1	RbacAdmin	后台项目	后台管理项目	启用	禁用

选择项目节点点击进入后，添加项目所属的模块节点，这里我们添加了数据管理、用户管理、权限管理、节点管理、默认模块和公共模块。他们其实分别对应了 FormAction、UserAction、GroupAction、NodeAction、IndexAction 和 PublicAction 这些 Action 类，注意名称要要类名一致，否则权限控制就不会起作用。

编号	名称	显示名	说明	状态	操作
7	Form	数据管理	数据管理模块	启用	禁用
6	User	用户管理	用户模块	启用	禁用
5	Group	权限管理	权限管理模块	启用	禁用
4	Node	节点管理	授权节点管理	启用	禁用
3	Index	默认模块	项目默认模块	启用	禁用
2	Public	公共模块	项目公共模块	启用	禁用

为了授权的简单起见，我们就把所有的操作放到 Public 模块里面定义，如果你不需要对各个模块的相同操作分别授权的话，这是一种方便的方法，否则你可能需要在各个模块下面都重复定义类似新增和插入的操作。下面是我们在后台管理要用到的操作方法，我们全部定义在 Public 模块下面。

编号	名称	显示名	说明	状态	操作
15	resume	恢复		启用	禁用
14	forbid	禁用		启用	禁用
13	delete	删除		启用	禁用
12	update	更新		启用	禁用
11	insert	写入		启用	禁用
10	edit	编辑		启用	禁用
9	add	增加		启用	禁用
8	index	列表		启用	禁用

添加完所有的节点后，我们就需要对用户组进行权限的授权了，我们选择权限管理，在里面添加了两个权限组：管理员组和普通用户组。

编号	组名	上级组	状态	描述	操作
2	普通用户组	无上级组	启用	一般用户权限	授权 用户列表
1	管理员组	无上级组	启用	具有一般管理员权限	授权 用户列表

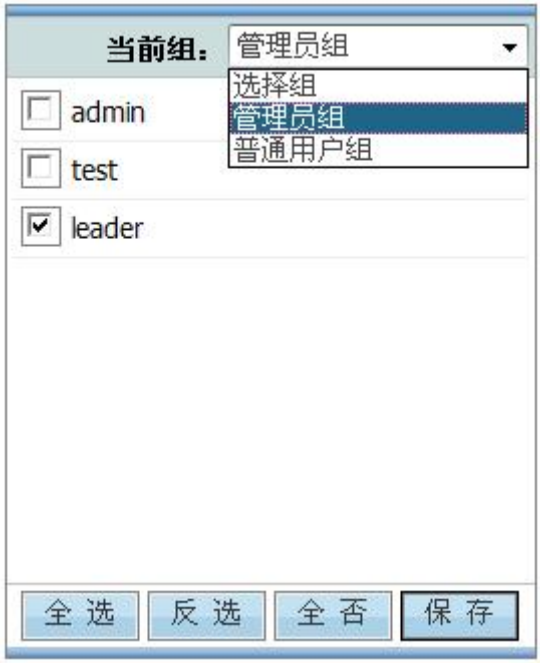
然后选择右边的操作区域的授权，进行组的授权，和添加节点的顺序一样，分别完成对项目、模块和操作的授权。

对项目授权

对项目下面的模块授权

对模块的操作授权

用户组的权限授权完成后，我们就需要把相关的用户放到某个组里面，让这些用户真正具有一定的权限。在权限组的列表中，我们选择操作区域的用户列表进入。



如果把某个用户加入了多个不同的组，那么他就会叠加所有组的权限。

我们对三个用户设置了不同的权限，分别是：

编号	用户名	昵称	状态	操作
1	admin	管理员	启用	禁用
2	test	测试用户	启用	禁用
3	leader	领导	启用	禁用

admin 超级管理员权限：可以访问所有操作

leader 管理员组：可以访问默认模块、公共模块、用户管理和数据管理，并且具有这些模块的所有的操作权限。

test 普通用户组：只能访问默认模块、公共模块和数据管理，并且只有列表、增加和编辑三个操作的权限。

下面我们就来实际看下 ThinkPHP 的 RBAC 权限控制的效果，我们分别用三种不同身份登录后可以看到的不同的菜单导航：

超级管理员的管理菜单

管理员组的管理菜单

普通用户组的管理菜单

关于动态菜单的控制，可以参考 PublicAction 里面的相关代码，其实是把我们在节点管理中授权的模块根据用户权限的不同动态显示出来而已。

如果某个用户同时具有两个组的权限，那么就可以看到两个组的管理菜单。

假设我们用 leader 帐号登录，但是却想输入节点管理的操作 URL 地址来试图绕过权限直接访问 http://localhost/rbacadmin/index.php/Node/

由于节点管理和权限管理这两个模块并没有授权给管理员组，所以 leader 帐号当然也就没有这个模块的操作权限，因此会提示下面的错误页面：

同样，当我们用 test 帐号登录，并且对数据管理中的数据进行禁用操作的时候，也会提示没有权限，因为我们并没有给普通用户组授予禁用操作的权限。

示例中涉及到 RBAC 认证的代码并不多，大部分工作都由系统的 RBAC 类库来完成了。通常在最简单的模式下，我们只需要开启用户认证，并在用户认证的提交方法里面添加委托认证方法和获取权限列表信息就行了。这样，一个完善的用户权限控制体系就完成了。示例中的大部分代码都是用于权限的授权，这部分代码就不详细解析了，主要都是一些数据的存取操作和模板显示，更加详细的请参考示例中心的代码实例。