2015年3月29日

终于搞明白scenario要怎么用了。

这里是yii1.0里的代码，和yii2.0大同小异。

public function rules ()

{

// NOTE: you should only define rules for those attributes that will receive user inputs.

return array (

array ( 'create\_user\_id, update\_user\_id', 'numerical', 'integerOnly'=> true ) ,

array ( 'name', 'length', 'max'=>128 ) ,

array ( 'create\_time, update\_time', 'safe' ) ,

// The following rule is used by search().

// Please remove those attributes that should not be searched.

array ( 'id, name, description, create\_time, create\_user\_id,

update\_time, update\_user\_id', 'safe', 'on'=>'search' ) ,

) ;

scenario(情景)允许你限制验证规则应用在特定的上下文中。一种典型的例子是 insert(插入)或update(更新)。例如：如果被指定为 'on'=>'insert'，这将表明验证规则只适用于模型的插入情景。这同样适用于'update'或其它的任何你希望定义的情景。你可以设置一个模型的 scenario(情景)属性或通过构造函数传给一个模型的实例。

使用fixtures为测试单独配置一个数据库。

一个测试fixture是指在测试运行中的一个系统状态或上下文。我们想多次运行我们的测试，每次运行我们希望能够让他们可以返回重复的结果。夹具的目的是提供一个众所周知在固定的环境中运行验证。通常情况下，一个夹具的主要工作是确保所有参与测试的对象在测试运行中初始化一个指定的状态。一个典型的夹具例子是使用固定和已知的数据加载一个数据库的表。

这里是yii2.0的官方讲解，很清晰

<http://www.yiiframework.com/doc-2.0/guide-test-fixtures.html>

yii2.0中将filter换成了behaviors

现在需要用到用户了，但是用户验证成了一个大问题，所以需要处理用户验证的事。

二进制比较

strcasecmp($user['username'], $username)

我自己做了一个user表，但是直接替换yii2.0 basic里面的那个表就出错。

于是就根据其内容改来改去的。

有几个地方要注意：

1.user表必须要用这个系统接口

implements \yii\web\IdentityInterface

2.表里必须要有authKey和accessToken两个字段，这两个字段主要是yii2.0系统用来加密用户信息的。

3.yii2.0系统里面是通过findIdentityByAccessToken()、getAuthKey()、validateAuthKey()来读取authKey和accessToken两个字段的，所以这几个方法必须要写进user的model中。

设定表格的controller(搞这个搞了一晚上)

[

'class' => 'yii\grid\ActionColumn',

'controller' => 'issue',

],

2015年4月5日

跳过了测试章节，又跳过了权限分配，接下来要实现评论功能了。

2015年4月5日

还是老老实实地实现权限功能吧。

实现权限功能，就得开启调用Yii2.0的RBAC功能。

1.首先配置RBAC

打开“项目根目录/config/web.php”，找到“components”，添加如下内容：

'authManager' => [

'class' => 'yii\rbac\DbManager',

],

打开“项目根目录/config/console.php”，找到“components”，添加如下内容：

'authManager' => [

'class' => 'yii\rbac\DbManager',

],

2.RBAC需要在数据库里添加几个特定的表，所以需要初始化。

yii migrate --migrationPath=@yii/rbac/migrations

运行以上代码，就能在数据库里添加四个表格，而RBAC的数据文件都存在里面。

这里注意，必须要在web.php和console.php两个文件里都添加了RBAC的配置代码，不然会报错。错误代码：Exception 'yii\base\InvalidConfigException' with message 'You should configure "authManager" component to use database before executing this migration.'

3.建立权限数据文件

各个关系网：“用户(user)”有“角色(role)”，“角色(role)”有“权限(permission)”，“权限(permission)”有“规则(rule)”。

我们要实现权限功能，

首先要新建好user、role、permission、rule，

再将rule分配给permission，将permission分配给role，将role分配给user；

整体就是“新建”和“分配”的过程。

在“项目根目录/commands/”下新建一个文件RbacController.php，添加如下代码：

<?php  
namespace app\commands;  
  
use Yii;  
use yii\console\Controller;  
  
class RbacController extends Controller  
{  
    public function actionInit()  
    {  
 $auth = Yii::$app->authManager;

// 清空原权限配置

$auth->removeAll();

/\* －－－－－－－－－－－－－创建权限开始－－－－－－－－－－－ \*/

// user增删改查

$createUser = $auth->createPermission('createUser');

$createUser->description = 'Create a user';

$auth->add($createUser);

$viewUser = $auth->createPermission('viewUser');

$viewUser->description = 'View a user';

$auth->add($viewUser);

$updateUser = $auth->createPermission('updateUser');

$updateUser->description = 'Update a user';

$auth->add($updateUser);

$deleteUser = $auth->createPermission('deleteUser');

$deleteUser->description = 'Delete a user';

$auth->add($deleteUser);

// issue增删改查

$createIssue = $auth->createPermission('createIssue');

$createIssue->description = 'Create a issue';

$auth->add($createIssue);

$viewIssue = $auth->createPermission('viewIssue');

$viewIssue->description = 'View a issue';

$auth->add($viewIssue);

$updateIssue = $auth->createPermission('updateIssue');

$updateIssue->description = 'Update a issue';

$auth->add($updateIssue);

$deleteIssue = $auth->createPermission('deleteIssue');

$deleteIssue->description = 'Delete a issue';

$auth->add($deleteIssue);

// project增删改查

$createProject = $auth->createPermission('createProject');

$createProject->description = 'Create a project';

$auth->add($createProject);

$viewProject = $auth->createPermission('viewProject');

$viewProject->description = 'View a project';

$auth->add($viewProject);

$updateProject = $auth->createPermission('updateProject');

$updateProject->description = 'Update a project';

$auth->add($updateProject);

$deleteProject = $auth->createPermission('deleteProject');

$deleteProject->description = 'Delete a project';

$auth->add($deleteProject);

/\* －－－－－－－创建权限结束－－－－－ \*/

/\* －－－－创建角色开始－－－－－－－ \*/

// reader

$reader = $auth->createRole('reader');

$auth->add($reader);

// member

$member = $auth->createRole('member');

$auth->add($member);

// owner

$owner = $auth->createRole('owner');

$auth->add($owner);

/\* －－－－－创建角色结束－－－－－\*/

/\* －－－－－分配权限开始－－－－－－\*/

// reader

$auth->addChild($reader, $viewProject);

$auth->addChild($reader, $viewIssue);

$auth->addChild($reader, $viewUser);

// member

$auth->addChild($member, $reader); // member享有reader的所有权限

$auth->addChild($reader, $createIssue);

$auth->addChild($reader, $updateIssue);

$auth->addChild($reader, $deleteIssue);

// owner

// $auth->addChild($owner, $reader); // owner享有member的所有权限

$auth->addChild($owner, $member); // owner享有member的所有权限

$auth->addChild($reader, $createUser);

$auth->addChild($reader, $updateUser);

$auth->addChild($reader, $deleteUser);

$auth->addChild($reader, $createProject);

$auth->addChild($reader, $updateProject);

$auth->addChild($reader, $deleteProject);

/\* －－－－－－－－－－－－－分配权限结束－－－－－－－－－－－ \*/

    }  
}

随后在根目录下运行：yii rbac/init

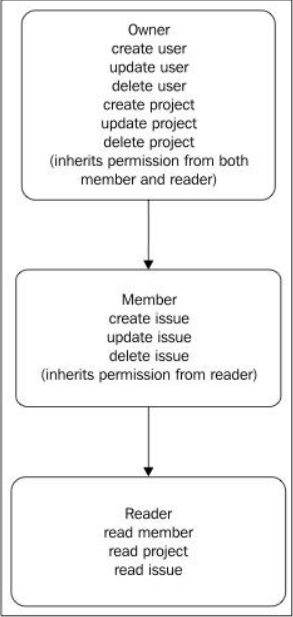
运行过后，数据库里的表的内容都有所变化。在该文件夹里创建出来的只是permission和role，user是直接调用之前创建的user模块来添加，而rule也需要创建额外的代码文件才能添加。

不过所有的关系分配都是在这里完成的。

Rule的代码文件理论上放哪里都可以，因为真正的调用都是在Rbac.php里完成。如果需要用到哪个rule，只要把这个rule的类引进来就行了。

一般来说，所谓的权限，其实就是在访问某个页面前查一查当前用户的权限表里有没有这个页面的权限字段，所以一般来说，permission就已经够用了。那什么情况下要用到rule呢？如果权限不是控制页面的访问，就需要用到rule了。

这里的角色权限分配如下：

角色为三种：owner，member，reader。

owner，member，reader。

现在权限数据是有了，但是不同的用户在不同的项目下权限是不一样的，所以就需要一个表来存储这些信息。

创建好了表之后，就可以在project的类里添加授权的方法了。

接下来就创建一个rule文件吧。

为了让项目的结构更整洁，我在根目录下创建一个名为rbac的文件夹，把所有的rule文件都放在这里面。

现在在该目录下创建AuthorRule.php文件，添加以下代码：

namespace app\rbac;  
  
use yii\rbac\Rule;  
  
/\*\*  
 \* Checks if authorID matches user passed via params  
 \*/  
class AuthorRule extends Rule  
{  
    public $name = 'isAuthor';  
  
    /\*\*  
     \* @param string|integer $user the user ID.  
     \* @param Item $item the role or permission that this rule is associated with  
     \* @param array $params parameters passed to ManagerInterface::checkAccess().  
     \* @return boolean a value indicating whether the rule permits the role or permission it is associated with.  
     \*/  
    public function execute($user, $item, $params)  
    {  
        return isset($params['post']) ? $params['post']->createdBy == $user : false;  
    }  
}