


Zend_Acl / Zend_Auth scénario d'exemple

Traduction de l'article de  **Simon Mundy**

par [Simon Mundy \(Son site\)](#) [Alain Sahli \(Tutoriels PHP de Alain Sahli\)](#) ([Blog](#))

Date de publication : 13.09.2007

Dernière mise à jour :

Cet article explique brièvement comment fonctionne le couple Zend_Auth / Zend_Acl.

- I - Introduction
- II - Règles d'accès
 - II-A - Structure de l'application
 - II-B - Bootstrap
 - II-C - Auth.php
- III - Liens

I - Introduction

Nous allons créer un environnement web dans lequel les utilisateurs de type 'public' et de type 'member' auront des restrictions d'accès. En fonction du contexte dans lequel ils se situent ils n'auront accès qu'à certaines ressources. Dans la plupart des cas, ce large concept se rapporte très bien au besoin des petits-moyens sites des développeurs Zend Framework (à mon avis). Pour mieux comprendre, nous supposons que c'est un groupe de SIG pour les utilisateurs MAC afin qu'ils puissent discuter de tout ce qui se rapporte à Mac OS X. Le site a trois zones (*home*, *news*, *tutorials*) qui sont accessible à tout le monde. Les membres peuvent également accéder au forum de discussion, à la newsletter de la communauté et à la demande de support afin de partager des problèmes communs.

Ci-dessous la structure du site:

Structure du site

Exprimé sous forme de :controller/:action notation:-

```
/home

/news/index
  /view
  /email

/tutorials/index
  /view

/forum/index
  /category
  /view
  /add
  /update
  /reply
  /search
  /report - report des abus etc...

/support/index
  /view
  /search
  /submit
  /confirmation -
  /comment - ajouter un commentaire

/login/index - gestion de l'authentification

/logout/index - détruis l'instance courante de Auth

/error/noroute - gère toutes les erreurs 404
  /failure - gère les erreurs du site
  /privileges - gère les erreurs de privilèges

/admin - un cms pour gérer le site
```

Ceci illustre rapidement les fonctionnalités et le contenu du site - pour des raisons de brièveté nous supposons que le concept général ainsi que les fonctions du site sont comprises et vous sont familières. Ce qui nous intéresse, c'est de gérer l'authentification des utilisateurs puis leur accès, mais au moins cela nous donne un exemple réel de ce qui est exigé.

II - Règles d'accès

Trois types d'accès ont été identifiés pour le site:

Types d'accès

- Invité (non-authentifié) : Les invités peuvent seulement accéder à la partie *home*, *news* et *tutorial*. Les invités qui essayeront d'accéder à une zone réservée aux membres seront invités à s'authentifier.
- Membre (authentifié) : Les membres ont accès à tous les contrôleurs supérieurs. Ils peuvent mettre à jour les messages sur le forum, mais seulement ceux dont ils sont les auteurs. Ils ne peuvent pas accéder à la section admin. S'il essaye d'y accéder un message leur indiquera qu'ils n'ont pas les privilèges nécessaires pour y accéder.
- Admin (authentifié) : Aucun accès restreint.

II-A - Structure de l'application

Nous utiliserons la structure conventionnelle donnée par [ici](#)Gavin.

Le bootstrap se situe dans `htdocs/index.php`.

II-B - Bootstrap

Le bootstrap utilise les modules habituels - Db, View, Config, Log, Router - et les stocke dans le `Zend_Front_Controller`, ainsi ils sont accessibles à partir de tous les contrôleurs à l'aide de la méthode `getInvokeArgs()`. Ceci nous évite d'utiliser un objet d'enregistrement supplémentaire et (heureusement) nous facilite le dépistage des dépendances.

Afin de satisfaire aux besoins des règles d'accès, nous créons une sous-classe qui hérite de `Zend_Acl`:

Classe MyAcl

```
class MyAcl extends Zend_Acl
{
    public function __construct(Zend_Auth $auth)
    {
        parent::__construct();

        $roleGuest = new Zend_Acl_Role('guest');

        $this->add(new Zend_Acl_Resource('home'));
        $this->add(new Zend_Acl_Resource('news'));
        $this->add(new Zend_Acl_Resource('tutorials'));
        $this->add(new Zend_Acl_Resource('forum'));
        $this->add(new Zend_Acl_Resource('support'));
        $this->add(new Zend_Acl_Resource('admin'));

        $this->addRole(new Zend_Acl_Role('guest'));
        $this->addRole(new Zend_Acl_Role('member'), 'guest');
        $this->addRole(new Zend_Acl_Role('admin'), 'member');

        // Les invités peuvent uniquement voir le contenu
        $this->allow('guest', 'home');
        $this->allow('guest', 'news');
        $this->allow('guest', 'tutorials');
        $this->allow('member', 'forum');
```

Classe MyAcl

```
$this->deny('member', 'forum', 'update'); // Suppression d'un privilège spécifique
$this->allow('member', 'support');
$this->allow('admin'); // Accès sans aucune restriction

// Ajout d'un nouvel objet ACL
$this->allow('member', 'forum', 'update', new MyAcl_Forum_Assertion($auth));
// NOTE: Zend_Acl dépend de Zend_Auth, il faut toujours lui passer en paramètre pour
obtenir l'identité
}
```

Ensuite il faut l'ajouter au bootstrap. La version finale du fichier index.php ressemble à ceci:

index.php

```
<?php

// Initialisation de la configuration / environnement
$config = new Zend_Config(new Zend_Config_Ini('../application/config/config.ini', 'live'));

// Création du sitemap à partir du .ini en utilisant la structure de l'exemple
$sitemap = new Zend_Config(new Zend_Config_Ini('../application/config/sitemap.ini', 'live'));

// Création de l'objet de base de données et activation / désactivation du débogage
$db = Zend_Db::factory($config->db->connection, $config->db->asArray());
...etc...

// Création de l'objet Auth
$auth = Zend_Auth::getInstance();

// Création de l'objet Acl
$acl = new MyAcl($auth); // see

// Création du routeur et configuration (Ordre LIFO pour les routes)
$routeur = new Zend_Controller_RewriteRouter;
...add rules...

// Création des vues et enregistrement des objets
$view = new My_View;
...init view...

$front = Zend_Controller_Front::getInstance();
$front->throwExceptions(true);
$front->setRouter($routeur)
->setDispatcher(new Zend_Controller_ModuleDispatcher())
->registerPlugin(new My_Plugin_Auth($auth, $acl))
->registerPlugin(new My_Plugin_Agreement($auth))
->registerPlugin(new My_Plugin_View($view))
->setControllerDirectory(array('default' => realpath('../application/controllers/default'),
                              'admin' => realpath('../application/controllers/admin')))
->setParam('auth', $auth)
->setParam('view', $view)
->setParam('config', $config)
->setParam('sitemap', $sitemap)
->dispatch();
```

C'est un joli (IMO) bootstrap standard - les lignes à noter sont les deux premier plugins que l'on passe au contrôleur.

II-C - Auth.php

Le but de ce plugin est avant tout de déterminer le 'rôle' de l'identité actuelle (Auth). Si `Zend_Auth::getIdentity()` retourne `false`, nous n'avons pas de rôle défini pour cette identité c'est pourquoi nous attribuons le rôle 'guest' (invité). Si l'utilisateur est authentifié l'identité va nous être retournée sous forme d'objet de type `Zend_Auth`, ce qui va nous

permettre d'extraire le rôle. Pour des raisons de simplicité dites-vous que le rôle est stocké dans une base de données et qu'il est retourné par une propriété publique de l'objet d'identité (par exemple 'member' ou 'admin').

Le rôle va alors à chaque fois interroger les règles ACL. Si nous interrogeons l'ACL et que nous sommes autorisés à voir le contrôleur courant (mappé dans les 'ressources' de chaque objet ACL) alors le dispatcher continue son joyeux chemin.

Si l'ACL nous bloque l'accès, nous déterminons si l'utilisateur a une identité valide. Si ce n'est pas le cas nous demandons à l'objet 'request' de nous rediriger vers un nouveau contrôleur (login) afin de nous authentifier. Dans ce cas aucune donnée n'est requise, ça va être géré *via* un formulaire dans le LoginController.

Cependant si l'identité est valide et que nous savons qu'elle n'a pas accès, nous la redirigeons vers le contrôleur 'error' qui va lui afficher l'erreur 'no privileges'.

J'ai choisi cette stratégie en pensant qu'aucun des contrôleurs n'a besoin de savoir quelque chose sur le processus ACL. Les contrôleurs peuvent donc se dire que l'accès aux actions a été approuvé et ils ne doivent que contrôler les actions spécifiques (par exemple view, valid articles, forum threads etc.).

Cependant un développeur peut décider d'ajouter d'autres règles ACL si c'est nécessaire et réduire ainsi la quantité de liens entre ACL et les contrôleurs.

```
index.php
```

```
<?php

class My_Plugin_Auth extends Zend_Controller_Plugin_Abstract
{
    private $_auth;
    private $_acl;

    private $_noauth = array('module' => 'default',
                             'controller' => 'login',
                             'action' => 'index');

    private $_noacl = array('module' => 'default',
                            'controller' => 'error',
                            'action' => 'privileges');

    public function __construct($auth, $acl)
    {
        $this->_auth = $auth;
        $this->_acl = $acl;
    }

    public function preDispatch($request)
    {
        if ($this->_auth->hasIdentity() {
            $role = $this->_auth->getIdentity()->getUser()->role;
        } else {
            $role = 'guest';
        }

        $controller = $request->controller;
        $action = $request->action;
        $module = $request->module;
    }
}
```

index.php

```
$resource = $controller;

if (!$this->_acl->has($resource)) {
    $resource = null;
}

if (!$this->_acl->isAllowed($role, $resource, $action)) {
    if (!$this->_auth->hasIdentity()) {
        $module = $this->_noauth['module'];
        $controller = $this->_noauth['controller'];
        $action = $this->_noauth['action'];
    }
}
```

III - Liens

 [La documentation de Zend_Acl](#)

 [La documentation de Zend_Auth](#)

