

# 和Zend Framework一起成长 (简体中文版)

By Rob Allen, [www.akrabat.com](http://www.akrabat.com)

Document Revision 1.2.0

Copyright © 2006

翻译: Jason Qi, [zft.backupdiy.com](http://zft.backupdiy.com)

本教程打算介绍用Zend Framework写一个最基本的数据库驱动的应用程序。

注: 本教程只工作于Zend Framework Version 0.6, 没有打算和Subversion里的最新开发代码及其他发行版本一起工作。(例如Version 0.2,译者注)

## Model-View-Controller 架构

下面是传统的构建PHP应用程序的方法:

```
<?php
include "common-libs.php";
include "config.php";
mysql_connect($hostname, $username, $password);
mysql_select_db($database);
?>
<?php include "header.php"; ?>
<h1>Home Page</h1>
<?php
$sql = "SELECT * FROM news";
$result = mysql_query($sql);
?>
<table>
<?php
while ($row = mysql_fetch_assoc($result)) {
?>
<tr>
<td><?php echo $row['date_created']; ?></td>
<td><?php echo $row['title']; ?></td>
</tr>
<?php
}
?>
</table>
<?php include "footer.php"; ?>
```

随着时间的推移, 当客户不断地有新的需求产生, 这种分布于不同地方的基于代码的应用就变得不可维护。一种改善应用程序可维护性的方法就是把代码分离成截然不同的三个部分 (并且通常分离成不同的文件):

模型	模型是处理被显示的数据的程序。在上面的例子中, 模型就是“news”。这样, 模型一般来说关心“business”逻辑并从数据库中读取和存储数据。
视图	视图包括一些关于显示给用户的程序, 通常是HTML
控制器	控制器同模型和视图配合在一起, 把正确的数据显示在合适的页面上。

Zend Framework 使用 Model-View-Controller (MVC) 架构. 这个 (MVC) 用来把您的程序分离成不同部分使得开发和维护变得容易。

## 需求

运行Zend Framework 需要:

- PHP 5.1.4 (或更高)
- Web 服务器支持mod\_rewrite 功能. 本教程使用Apache。

## 获取Zend Framework

从这里<http://framework.zend.com/download>下载Zend Framework, 有两种格式. zip或者. tar.gz。当前的版本是0.6, 对于本教程, 您必须使用0.6版本。

## 目录结构

虽然Zend Framework 没有强求使用一个标准的目录结构, 但它的手册还是推荐了一个通用的目录结构。这个目录结构假设您完全控制Apache的配置。然而我们想使它更容易一点, 所以做了一点修正。

建立一个根目录叫做 zf-tutorial. 这个意味着指向程序的URL 将会是 <http://localhost/zf-tutorial>。

建立如下的子目录, 这些目录将存放程序所需要的文件:

```
zf-tutorial/  
    /application  
        /controllers  
        /models  
        /views  
    /library  
    /public  
        /images  
        /scripts  
        /styles
```

正如您所看到的, 我们已经把我们的程序中的模型、视图和控制器的文件分离到不同的子目录中。图像, Scripts和CSS文件被存放在public目录下的不同子目录中。Zend Framework文件放在library目录下。如果我们还需要其他库文件, 都放在这里。以我来举例说明, 我把ZendFramework-0.6.zip解压到一个临时目录。所有的文件都包含在一个叫做ZendFramework-0.6的目录。把 ZendFramework-0.6/library中的文件拷贝到zf-tutorial/library。在这个目录下, 有一个子目录叫做Zend和一个文件叫做Zend.php。

## 启动文件 (bootstrapper)

Zend Framework的控制器，`Zend_Controller`，支持clean urls。为实现这个目的，所有的客户端请求必需通过`index.php`，这就是所谓的启动文件（bootstrapper）。这给我们提供了程序中所有页面的中心点并确保运行环境配置正确。我们用`.htaccess`文件来实现这个目的。`.htaccess`在`zf-tutorial`的根目录中，内容如下：`zf-tutorial/library`

```
zf-tutorial/.htaccess
RewriteEngine on
RewriteRule .* index.php
php_flag magic_quotes_gpc off
php_flag register_globals off
```

`RewriteRule`非常简单并可以翻译为“对任何url, 重定向到`index.php`”。为安全和稳定起见，我们也在PHP ini中做一些设置。虽然这些都已经设置正确，我们还是要确认一下！注意在`.htaccess`中的`php_flag`只工作于`mod_php`下。如果您使用CGI/FastCGI,就需要确保`php.ini`配置正确。

然而，对于图像，JavaScript和CSS文件的请求不应该重定向到启动文件，把这些文件放到`public`目录下，我们很容易地通过另一个`.htaccess`文件来配置Apache。这个`.htaccess`文件在`zf-tutorial/public`下

```
zf-tutorial/public/.htaccess
RewriteEngine off
```

虽然我们当前的rewrite rules不需要太严格，但我们还是在`application`和`library`目录下添加一些`.htaccess`文件用来保护我们的程序。

```
f-tutorial/application/.htaccess
deny from all
zf-tutorial/library/.htaccess
deny from all
```

注意在Apache下设置`.htaccess`文件，`httpd.conf`中的`AllowOverride`必须设置为`All`。这些设置多个`.htaccess`文件的灵感来自于 Jayson Minard 的文章 [“Blueprint for PHP Applications: Bootstrapping \(Part 2\)”](#)。我建议大家去读读整个系列文章。

## 启动文件: `index.php`

`zf-tutorial/index.php` 是我们的启动文件，我们从下面的代码开始：

```
zf-tutorial/index.php
<?php
error_reporting(E_ALL|E_STRICT);
date_default_timezone_set('Europe/London');

set_include_path('.' . PATH_SEPARATOR . './library'
                . PATH_SEPARATOR . './application/models/'
                . PATH_SEPARATOR . get_include_path());
include "Zend.php";

Zend::loadClass('Zend_Controller_Front');

// setup controller
$baseUrl = substr($_SERVER['PHP_SELF'], 0,
strpos($_SERVER['PHP_SELF'], '/index.php'));
$frontController = Zend_Controller_Front::getInstance();
$frontController->setBaseUrl($baseUrl);
$frontController->setControllerDirectory('./application/controllers');
```

```

    $frontController->throwExceptions(true);

    // run!
    $frontController->dispatch();

```

注意我们没有在文件的结尾方?>, 因为它不是必须的并且有个好处是: 我们可以防止当多余的 **whitespace** 发生在?>后面出现难以调试的错误。

让我们过一遍这个文件

```

error_reporting(E_ALL|E_STRICT);
date_default_timezone_set('Europe/London');

```

第一行确保我们能看见我们犯的错误 (假定在 `php.ini` 中的 `display_errors` 设置为 `on`), **PHP5.1+** 要求设置当前时区。显然, 您应该选择您自己的时区。

```

set_include_path('.' . PATH_SEPARATOR . './library'
. PATH_SEPARATOR . './application/models/'
. PATH_SEPARATOR . get_include_path());
include "Zend.php";

```

**Zend Framework** 是这样设计的, 所有的文件必须包含在 `include path` 中。我们也把我们的模型目录包含在 `include path` 中, 这样我们以后就能很容易加载我们的模型类。一开始, 我们必须用 `include "Zend.php"`, 这样我们就能访问 `Zend` 类。在 `Zend` 类, 我们用静态方法加载其他 `Zend Framework` 类, 例如:

```

Zend::loadClass('Zend_Controller_Front');

```

`Zend::loadClass` 加载已经命名的类。它是把下划线转换成路径隔离符来实现的, 并在最后加上 `.php` 后缀。这样, 类 `Zend_Controller_Front` 将从 `Zend/Controller/front.php` 加载。如果您在您的类库里使用相同的命名规则, 就可以用 `Zend::loadClass()` 来加载它们。我们需要加载控制器类和路由类。

前端控制器用路由类来映射请求的 `URL` 到正确的 `PHP` 函数, 然后显示页面。为了能使路由工作, 需要解决 `URL` 的哪一部分是指向 `index.php` 的路径, 这样它就可以在那个点后面寻找 `url` 元素。原则上, 请求对象 (一个 `Zend_Controller_Request_Http` 的实例) 应该能自动检测到正确的 `base URL`。实践上, 我发现它至少不工作于我的特定的安装。幸运地, 我们能通过 `setBaseUrl()` 来设置 `base URL`。这个函数在 `http request` 类和前端控制器类中都有效, 这样我们不用担心实例化我们自己的请求对象。

我们现在为路由设置 `base URL`, 它是一个指向我们的 `index.php` 的正确的 `URL` 的字符串。我使用 `$_SERVER['PHP_SELF']`, 它工作得很好, 它应该能适用于大多数的服务器配置。如果它在您的系统上不工作, 就把指向 `index.php` 的正确 `URL` 的字符串赋值给变量 `$baseUrl`。

```

// setup controller
$baseUrl = substr($_SERVER['PHP_SELF'], 0,
strpos($_SERVER['PHP_SELF'], '/index.php'));
$frontController = Zend_Controller_Front::getInstance();
$frontController->setBaseUrl($baseUrl);

```

我们也需要配置前端控制器, 这样它就知道我们的控制器在那个目录。

```

$frontController->setControllerDirectory('./application/controllers');

```

因为这是一个教程，我们打算运行一个测试系统。我决定让前端控制器抛出所有发生的异常。缺省地，前端控制器将捕获它们并把它们存储在“响应”对象所建立的 `_exceptions` 属性里。响应对象掌握所有关于响应给URL的信息。这包括头，页面内容和异常。前端控制器在完成它的工作前将自动发送头和显示页面内容。

这对新学Zend Framework的人有点迷惑，仅仅重新抛出很容易，这样异常容易可见。当然，在一个生产服务器上，您不应当给用户显示错误信息。

```
$frontController->throwExceptions(true);
```

终于，我们可以运行我们的程序了：

```
// run!
$frontController->dispatch();
```

如果您用 [http://localhost/zf\\_tutorial/](http://localhost/zf_tutorial/) 去测试，致命的错误类似于：

```
Fatal error: Uncaught exception 'Zend_Exception' with message 'File
"./application/controllers/IndexController.php" was not found.
(etc.)
```

这个告诉我们还没有设置好我们程序。在我们动手之前，最好讨论一下我们要打算做什么，那就开始吧。

## 网站

我们打算建立一个非常简单的库存系统网站，它用来管理我们的CD收藏。在主页上将列出我们的收藏并允许我们添加，编辑和删除这些收藏的CD。我们打算把这些存储到一个数据库里。数据库的设计是这样的：

```
Fieldname Type Null? Notes
id Integer No Primary key, Autoincrement
artist Varchar(100) No
title Varchar(100) No
```

## 所需要的页面

下面这些页面是必需的。

**Home page**，它将显示albums的列表并提供编辑和删除的链接。当然还有一个添加新album的链接。

**Add New Album**，这个页面提供一个添加新Album的表格。

**Edit Album**，这个页面提供一个编辑Album的表格。

**Delete Album**，这个页面将确认我们想删除一个Album，然后删除它。

## 组织页面

在设置文件之前，理解Zend Framework如何组织页面很重要。每个应用的页面叫做“action”，许多“action”组成控制器。例如，对于这样一个格式的URL <http://localhost/zftutorial/>

news/view 控制器是news, action 是view。它允许把相关的action组织成组, 例如, 一个news控制器可以有current, archived 和view。

Zend Framework 控制器把index作为一个特别的action而保留为缺省的action。这样, 对于 <http://localhost/zf-tutorial/news/> index action将被news 控制器执行。Zend Framework 也保留了一个缺省的控制器, 也毫不惊讶地叫做index。这样, <http://localhost/zf-tutorial/> 将执行index控制器下的action index。

作为一个教程, 我们打算涉及“复杂的”事情如登陆, 那将作为另外一个教程... ..

因为我们有四个页面, 我们把他们组织到一个控制器里并且这个控制器是缺省的index控制器, 如下表:

<i>Page</i>	<i>Controller</i>	<i>Action</i>
Home page	Index	Index
Add new album	Index	Add
Edit album	Index	Edit
Delete album	Index	Delete

简单而漂亮!

## 设置控制器

现在可以设置控制器了。在Zend Framework里, 控制器是一个必需被叫做{Controller name}Controller的类。注意{Controller name}必需以大写字母开头。并且, 这个类必须在{Controller name}Controller.php这样的文件中, 这个文件还必需在特定的控制器目录中。强调一下, {Controller name}必需以大写字母开头而其他字母一定是小写。每个action是在控制器类里的public函数, 名字必需是{action name}Action。在这里, {action name}应该以小写字母开头。

这样在文件zf-tutorial/application/controllers/IndexController.php里我们的控制器类叫做IndexController

### zf-tutorial/application/controllers/IndexController.php

```
<?php

class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
        echo "<p>in IndexController::indexAction()</p>";
    }
    function addAction()
    {
        echo "<p>in IndexController::addAction()</p>";
    }
    function editAction()
    {
        echo "<p>in IndexController::editAction()</p>";
    }
    function deleteAction()
    {
        echo "<p>in IndexController::deleteAction()</p>";
    }
}
```

```
}
```

开始，我们让每个action输出它们的名字，导航到下表的URL做测试：

URL	<i>Displayed text</i>
http://localhost/zf_tutorial/	in IndexController::indexAction()
http://localhost/zf_tutorial/index/add	in IndexController::addAction()
http://localhost/zf_tutorial/index/edit	in IndexController::editAction()
http://localhost/zf_tutorial/index/delete	in IndexController::deleteAction()

现在，我们在程序里有个能工作的路由器和能够在每个页面被执行的正确的action。如果它不能工作，请到本教程后面查阅“故障排除”一节，看看能否得到帮助。

好啦，现在来做视图。

## 设置视图

Zend Framework的视图叫做Zend\_View，有点顾名思义。视图将允许我们把显示页面的代码从action函数里分离出来。

基本的Zend\_View的用法是：

```
$view = new Zend_View();
$view->setScriptPath('/path/to/view_files');
echo $view->render('view.php');
```

如果我们打算直接把这个骨架放到action函数并重复这个对action毫无意义的代码，这将非常容易。但我们宁愿在其他地方初始化它，然后在action里使用这个已经初始化过的对象。

Zend Framework的设计者已经预料到这类问题并提供了一个注册表来存储和获取对象。注册一个对象的代码是：

```
Zend::register('obj', $object);
```

获取对象的代码是：

```
$object = Zend::registry('obj')
```

为了集成视图到程序中，我们将在启动文件(zf-tutorial/index.php)里建立视图对象并把它加到注册表中

### Relevant part of zf-tutorial/index.php

```
...
Zend::loadClass('Zend_Controller_Front');
Zend::loadClass('Zend_Controller_RewriteRouter');
Zend::loadClass('Zend_View');
// register the view we are going to use
$view = new Zend_View();
$view->setScriptPath('./application/views');
Zend::register('view', $view);
// setup controller
$route = new Zend_Controller_RewriteRouter();
$controller = Zend_Controller_Front::getInstance();
...

```

文件中，黑体字部分是被修改过的并且它的意义很明确。注意，在建立实例和设置到视图目录的脚本路径之前，我们首先必需用**Zend::LoadClass()**加载**Zend\_View**类。

我们已经把视图注册到注册表里，现在，我们需要建立一些带有测试代码的视图文件并在控制器里的**action**中测试。

如下修改 **IndexController**。再强调一下，被修改的部分是黑体字：

```
zf-tutorial/application/controllers/IndexController.php
<?php
class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
        $view = Zend::registry('view');
        $view->title = "My Albums";
        $this->_response->setBody($view->render('indexIndex.tpl.php'));
    }
    function addAction()
    {
        $view = Zend::registry('view');
        $view->title = "Add New Album";
        $this->_response->setBody($view->render('indexAdd.tpl.php'));
    }
    function editAction()
    {
        $view = Zend::registry('view');
        $view->title = "Edit Album";
        $this->_response->setBody($view->render('indexEdit.tpl.php'));
    }
    function deleteAction()
    {
        $view = Zend::registry('view');
        $view->title = "Delete Album";
        $this->_response->setBody($view->render('indexDelete.tpl.php'));
    }
}
```

在每个函数中，我们从注册表里获取视图对象并给它分配一个**title**变量，接着，显示（**render**）响应的模板。好过仅仅从**render()**回应数据，我们使用给响应对象分配**body**属性，这个是**Zend Framework MVC**系统的一部分。作为使用**MVC**系统的产生的结果，响应对象用来收集头，**body**内容和异常。前端控制器接着在派遣的最后自动地发送头和**body**内容。

很显然，我们需要做四个视图文件。传统地，这些视图文件叫做模板。我使用它们的**action**名字加上**.tpl.php**扩展名来作为视图文件名。

```
zf-tutorial/application/views/indexIndex.tpl.php
<html>
<head>
    <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
    <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
zf-tutorial/application/views/indexAdd.tpl.php
<html>
<head>
    <title><?php echo $this->escape($this->title); ?></title>
```

```

</head>
<body>
    <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
zf-tutorial/application/views/indexEdit.tpl.php
<html>
<head>
    <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
    <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
zf-tutorial/application/views/indexDelete.tpl.php
<html>
<head>
    <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
    <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>

```

测试每个action，它们应该显示出黑体字标题。

## 共同的 HTML 代码

很快，在我们的视图里有许多相同的HTML代码。我们把他们提取到一个文件site.tpl.php中。这样，我们对页面提供“外部”片段并在site.tpl.php中调用。

强调一下，控制器需要修改：

```

zf-tutorial/application/controllers/IndexController.php
<?php
class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
        $view = Zend::registry('view');
        $view->title = "My Albums";
        $view->actionTemplate = 'indexIndex.tpl.php';
        $this->_response->setBody($view->render('site.tpl.php'));
    }
    function addAction()
    {
        $view = Zend::registry('view');
        $view->title = "Add New Album";
        $view->actionTemplate = 'indexAdd.tpl.php';
        $this->_response->setBody($view->render('site.tpl.php'));
    }
    function editAction()
    {
        $view = Zend::registry('view');
        $view->title = "Edit Album";
        $view->actionTemplate = 'indexEdit.tpl.php';
        $this->_response->setBody($view->render('site.tpl.php'));
    }
    function deleteAction()
    {

```

```

        $view = Zend::registry('view');
        $view->title = "Delete Album";
        $view->actionTemplate = 'indexDelete.tpl.php';
        $this->_response->setBody($view->render('site.tpl.php'));
    }
}

```

我们已经介绍过一个新的变量叫做actionTemplate并在所有的action里调用同一个site.tpl.php视图文件如下:

```

zf-tutorial/application/views/site.tpl.php
<html>
<head>
    <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
    <div id="content">
        <?php echo $this->render($this->actionTemplate); ?>
    </div>
</body>
</html>
zf-tutorial/application/views/indexIndex.tpl.php
<h1><?php echo $this->escape($this->title); ?></h1>
zf-tutorial/application/views/indexAdd.tpl.php
<h1><?php echo $this->escape($this->title); ?></h1>
zf-tutorial/application/views/indexEdit.tpl.php
<h1><?php echo $this->escape($this->title); ?></h1>
zf-tutorial/application/views/indexDelete.tpl.php
<h1><?php echo $this->escape($this->title); ?></h1>

```

## 风格

尽管这仅仅是个教程, 我们需要CSS文件来使我们的程序看起来更可展示的。

```

zf-tutorial/application/views/site.tpl.php
...
<head>
    <title><?php echo $this->escape($this->title); ?></title>
    <link rel="stylesheet" type="text/css" media="screen"
        href="/zf-tutorial/public/styles/site.css" />
</head>
...
zf-tutorial/public/styles/site.css
body,html {
    font-size:100%;
    margin: 0;
    font-family: Verdana,Arial,Helvetica,sans-serif;
    color: #000;
    background-color: #fff;
}
h1 {
    font-size:1.4em;
    color: #000080;
    background-color: transparent;
}
#content {
    width: 770px;
    margin: 0 auto;
}
label {

```

```

        width: 100px;
        display: block;
        float: left;
    }
    #formbutton {
        margin-left: 100px;
    }
    a {
        color: #000080;
    }
}

```

## 数据库

既然我们从显示视图里分离了程序的控制，是时候看一下程序中的模型部分。记住，模型是处理程序的核心意图（所谓的“business rules”）因此，对我们来说就是数据库。我们将利用Zend Framework中的 `Zend_Db_Table` 类来操作数据库中表的插入，更新和删除记录。

## 配置

为了使用 `Zend_Db_Table`，我们需要告诉它哪个数据库（连同用户和密码）将被使用。因为我们不愿意在程序中 `hard-code`，所以我们用一个配置文件来保存这些信息。

Zend Framework提供了一个 `Zend_Config` 来提供灵活的面向对象访问配置文件。此刻，配置文件可以是一个PHP数组，一个INI文件或者XML文件。我们将使用INI文件：

```

zf-tutorial/application/config.ini
[general]
db.adapter = PDO_MYSQL
db.config.host = localhost
db.config.username = rob
db.config.password = 123456
db.config.dbname = zftest

```

显然，您应该使用您自己的用户名，密码和数据库名，而不是我的！

使用 `Zend_Config` 非常容易：

```

$config = new Zend_Config_Ini('config.ini', 'section');

```

注意在这个例子中，`Zend_Config_Ini`从INI文件中加载一个section,而不是每一个section。它支持section名字中的符号并允许加载另外的section。`Zend_Config`还认为参数中的“点”等级分离符，它允许把一组相关的参数组成一个组。在我们的config.ini里，用户名，密码，数据库名参数都是在组 `$config->db->config` 之下。

我们将在启动文件里加载配置文件（index.php）

```

Relevant part of zf-tutorial/index.php
...
Zend::loadClass('Zend_Controller_Front');
Zend::loadClass('Zend_View');
Zend::loadClass('Zend_Config_Ini');
// load configuration
$config = new Zend_Config_Ini('./application/config.ini', 'general');
Zend::register('config', $config);
// register the view we are going to use

```

```
$view = new Zend_View();
$view->setScriptPath('./application/views');
...
```

修改的部分是黑体字。我们加载我们将要用的类（Zend\_Config\_Ini）并加载application/config.ini中的'general' section到\$config对象。

注：在这个教程里，我们不需要把\$config存到注册表里，但是在“真正的”项目中，您可能在INI文件里有很多配置信息，而不仅仅是数据库。

## 建立 Zend\_Db\_Table

为了使用Zend\_Db\_Table,我们需要告诉它我们刚刚加载的数据库配置信息。我们需要建立一个Zend\_Db的实例并用静态函数Zend\_Db\_Table::setDefaultAdapter()来注册它。我们在启动文件里完成它（黑体字）：

### Relevant part of zf-tutorial/index.php

```
...
Zend::loadClass('Zend_Config_Ini');
Zend::loadClass('Zend_Db');
Zend::loadClass('Zend_Db_Table');
// load configuration
$config = new Zend_Config_Ini('./application/config.ini', 'general');
Zend::register('config', $config);
// setup database
$db = Zend_Db::factory($config->db->adapter,
$config->db->config->asArray());
Zend_Db_Table::setDefaultAdapter($db);
// register the view we are going to use
$view = new Zend_View();
$view->setScriptPath('./application/views');
Zend::register('view', $view);
...
```

## 建表

我打算用MySQL，所以建表SQL语句如下：

```
CREATE TABLE album (
    id int(11) NOT NULL auto_increment,
    artist varchar(100) NOT NULL,
    title varchar(100) NOT NULL,
    PRIMARY KEY (id)
)
```

在MySQL的客户端运行这个语句，例如phpMyAdmin或者标准的MySQL命令行客户端。

## 插入测试Albums

我们还要给表中插入一些记录来测试主页中的列表功能。我打算用Amazon.co.uk中”Hot 100”CD中的头两条：

```
INSERT INTO album (artist, title)
VALUES
('James Morrison', 'Undiscovered'),
('Snow Patrol', 'Eyes Open');
```

## 模型

`Zend_Db_Table` 是一个 **abstract** 类，所以我们派生一个专门管理 `album` 的类。缺省地，`Zend_Db_Table` 希望类名和表名一样。这样，类名就叫做 `Album` 因为表名是 `album`。并且，`Zend_Db_Table` 假定表有一个主键叫做 `id`，它能够自动增长。不过，如果需要的话，这些都可以重新定义 (`overridden`)。

我们将保存我们的 `Album` 表到模型目录：

```
zf-tutorial/application/models/Album.php
<?php
class Album extends Zend_Db_Table
{
}
```

不是很复杂吧！我们很幸运，我们的需求非常简单并且 `Zend_Db_Table` 直接提供了足够的函数。然而，如果您需要特殊的函数管理您的模型，您可以把它们放到这个类里。一般来说，附加的函数将是附加的“`find`”类型方法，它能够是您要寻找的精确数据的集合。

## 列表 Albums

既然我们设置了配置和数据库信息，程序应该可以列出一些 `album`。这些在 `IndexController` 类里完成。

在 `IndexController` 里，每个 `action` 将用 `Album` 类来处理 `album` 数据库。当控制器实例化的时候来加载 `album` 类是很有意义的。这个在 `init()` 函数里完成。

```
zf-tutorial/application/controllers/IndexController.php
<?php
class IndexController extends Zend_Controller_Action
{
    function init()
    {
        Zend::loadClass('Album');
    }
    function IndexAction()
    {
        ...
    }
}
```

这是一个用 `Zend::loadClass()` 来加载我们自己的类的例子并且能够工作，因为我们已经在 `index.php` 中把模型路径放到 `include path` 里。我们打算在 `indexAction()` 里列出 `album`。

```
zf-tutorial/application/controllers/IndexController.php
...
function indexAction()
{
    $view = Zend::registry('view');
    $view->title = "My Albums";
Page 14 of 19
    $album = new Album();
    $view->albums = $album->fetchAll();
    $view->actionTemplate = 'indexIndex.tpl.php';
    $this->_response->setBody($view->render('site.tpl.php'));
}
```

```
}  
...
```

函数 `Zend_Db_Table::fetchAll()` 返回一个 `Zend_Db_Table_Rowset`，它允许我们在视图模板里重复使用：

```
zf-tutorial/application/views/indexIndex.tpl.php  
<h1><?php echo $this->escape($this->title); ?></h1>  
<p><a href="/zf-tutorial/index/add">Add new album</a></p>  
<table>  
<tr>  
    <th>Title</th>  
    <th>Artist</th>  
    <th>&nbsp;</th>  
</tr>  
<?php foreach($this->albums as $album) : ?>  
<tr>  
    <td><?php echo $this->escape($album->title);?></td>  
    <td><?php echo $this->escape($album->artist);?></td>  
    <td>  
        <a href="/zf-tutorial/index/edit/id/<?php echo $album->id;?>">  
            >Edit</a>  
        <a href="/zf-tutorial/index/delete/id/<?php echo $album->id;?>">  
            >Delete</a>  
    </td>  
</tr>  
<?php endforeach; ?>  
</table>
```

<http://localhost/zf-tutorial/> 应该可以列出一个很好的album。显然，如果您的程序不在/zf-tutorial里，修改一下您的路径。

## 处理 Post 和 Get 变量

在传统的PHP程序里，“magic”全局变量 `$_POST` 和 `$_GET` 用来获取用户提交的变量，但是很容易忘记验证这些数据。如果数据不能很好验证，那将导致一些安全问题，和中断程序。Zend Framework提供 `Zend_Filter_Input` 类是验证用户数据很容易。

To use `Zend_Filter_Input`:

```
$postArray = new Zend_Filter_Input($_POST, false);  
$username = $postArray->testName('username');  
if ($username !== false) {  
    // $username is a valid name  
}
```

一个记住 `Zend_Filter_Input` 的关键点是它通常 **wipe** 输入数组。这就是，在建立一个 `Zend_Filter_Input` 之后， `$_POST`, `$_POST` 被设置为 `NULL`。这个受到MVC系统的干涉，所以我们用传递第二个参数到 `Zend_Filter_Input` 的构造器来不让它自动清空输入数组。这样，我们在 `index.php` 设置我们自己的过滤并把它们存储在 `Zend::registry` 里，然后我们可以在任何需要的地方访问它。

```
Relevant part of zf-tutorial/index.php  
...  
Zend::loadClass('Zend_Db');  
Zend::loadClass('Zend_Db_Table');  
Zend::loadClass('Zend_Filter_Input');  
// register the input filters
```

```

Zend::register('post', new Zend_Filter_Input($_POST, false));
Zend::register('get', new Zend_Filter_Input($_GET, false));
// load configuration
...

```

现在我们通过下面的代码获得一个post变量

```

$post = Zend::registry('post');
$myVar = $post->testAlpha('myVar');

```

## 添加 New Albums

既然我们已经设置了Post 输入过滤，我们就可以做添加新的album。有两点：

- 显示一个表格让用户提供信息
- 处理这个表格的提交并把数据保存到数据库里

这个在addAction() 里完成：

```

zf-tutorial/application/controllers/IndexController.php
...
function addAction()
{
    $view = Zend::registry('view');
    $view->title = "Add New Album";
    if (strtolower($_SERVER['REQUEST_METHOD']) == 'post') {
        $post = Zend::registry('post');
        $artist = trim($post->noTags('artist'));
        $title = trim($post->noTags('title'));
        if ($artist != '' && $title != '') {
            $data = array(
                'artist' => $artist,
                'title' => $title
            );
            $album = new Album();
            $album->insert($data);
            $this->_redirect('/');
            return;
        }
        // set up an "empty" album
        $view->album = new stdClass();
        $view->album->artist = '';
        $view->album->title = '';
        // additional view fields required by form
        $view->action = 'add';
        $view->buttonText = 'Add';
        $view->actionTemplate = 'indexAdd.tpl.php';
        $this->_response->setBody($view->render('site.tpl.php'));
    }
    ...
}

```

注意我们如何检查变量\$\_SERVER['REQUEST\_METHOD'] 来看表格是否提交。如果表格被提交，就用noTags()函数来获取post数组并确保里面没有html代码，我们用模型类，album()，把信息插入到数据的表里。

在添加了一个album之后，我们用控制器的\_redirect()方法重定向。注意\_request()知道所有的base URL，所以不传递'/zf-tutorial/' 作为URL的重定向。

终于，我们建立了我们将在模板里使用的视图表格。进一步看，我们能预计到编辑action的表格和这个非常类似，所以我们将使用共同的模板文件，模板文件将被indexAdd.tpl.php和indexEdit.tpl.php调用。

用于添加album的模板如下：

```
zf-tutorial/application/views/indexAdd.tpl.php
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('_indexForm.tpl.php'); ?>
zf-tutorial/application/views/_indexForm.tpl.php
<form action="/zf-tutorial/index/<?php echo $this->action; ?>"
method="post">
<div>
    <label for="artist">Artist</label>
    <input type="text" class="input-large" name="artist"
value="<?php echo $this->escape(trim($this->album-
>artist));?>" />
</div>
<div>
    <label for="title">Title</label>
    <input type="text" class="input-large" name="title"
value="<?php echo $this->escape($this->album->title);?>" />
</div>
<div id="formbutton">
    <input type="hidden" name="id"
value="<?php echo $this->album->id; ?>" />
    <input type="submit" name="add"
value="<?php echo $this->escape($this->buttonText); ?>" />
</div>
</form>
```

这是相当简单的代码。我们打算在编辑action里也用\_indexForm.tpl.php，用\$this->action比hard coding要好些。相似地，我们在提交按钮里的文字也使用变量。

## 编辑一个 Album

编辑一个album 几乎和添加一个album相同，所以代码非常类似：

```
zf-tutorial/application/controllers/IndexController.php
...
function editAction()
{
    $view = Zend::registry('view');
    $view->title = "Edit Album";
    $album = new Album();
    if (strtolower($_SERVER['REQUEST_METHOD']) == 'post') {
        $post = Zend::registry('post');
        $id = $post->testInt('id');
        $artist = trim($post->noTags('artist'));
        $title = trim($post->noTags('title'));
        if ($id !== false) {
            if ($artist != "" && $title != "") {
                $data = array(
                    'artist' => $artist,
                    'title' => $title,
                );
            }
        }
    }
}
```

```

        $where = 'id = ' . $id;
        $album->update($data, $where);
        $this->_redirect('/');
        return;
    } else {
        $view->album = $album->find($id);
    }
} else {
    // album id should be $params['id']
    $id = (int)$this->_request->getParam('id', 0);
    if ($id > 0) {
        $view->album = $album->find($id);
    }
}
// additional view fields required by form
$view->action = 'edit';
$view->buttonText = 'Update';
$view->actionTemplate = 'indexEdit.tpl.php';
$this->_response->setBody($view->render('site.tpl.php'));
}
...

```

注意当程序不在“post”模式，我们使用getParam()从请求的params属性中读取id参数，模板如下：

```

zf-tutorial/application/views/indexEdit.tpl.php
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('_indexForm.tpl.php'); ?>

```

## 改进!

您可能已经注意到 非常类似并且添加和编辑使用同一个模板，看来需要改进一下，这个就留给您，亲爱的读者... ..

## 删除 an Album

为了使我们的程序完整，需要一个删除的功能。在列表页里，每个album旁边有个删除链接，这是个非常朴素的办法让您去点击删除。这可能错了。记得我们的HTTP spec，不能使用不可以撤回动作的GET，作为代替，要使用POST。Google的最近的加速器beta把这点带给许多人。

我们要用一个确认表格，如果用户点“yes”，我们将做删除动作。

这段代码看起来还是和添加、编辑的action有点象：

```

zf-tutorial/application/controllers/IndexController.php
...
function deleteAction()
{
    $view = Zend::registry('view');
    $view->title = "Delete Album";
    $album = new Album();
    if (strtolower($_SERVER['REQUEST_METHOD']) == 'post') {
        $post = Zend::registry('post');
        $id = $post->getInt('id');
    }
}

```

```

        if (strtolower($post->testAlpha('del')) == 'yes' && $id > 0) {
            $where = 'id = ' . $id;
            $album->delete($where);
        }
    } else {
        // album id should be $params['id']
        $id = (int)$this->_request->getParam('id', 0);
        if ($id > 0) {
            $view->album = $album->find($id);
            $view->actionTemplate = 'indexDelete.tpl.php';
            // only render if we have an id.
            $this->_response->setBody($view->render('site.tpl.php'));
            return;
        }
    }
    // redirect back to the album list in all cases unless we are
    // rendering the template
    $this->_redirect('/');
}
...

```

再一次，我们使用同样的计策去检查请求方法来决定我们要显示确认表格还是应该删除。通过 `Album()` 类，就像插入和更新，实际的删除动作通过调用 `Zend_Db_Table::delete()` 来完成。

注意，我们在设置响应 `body` 后立即返回。这就是为什么我们在函数的最后能重定向到 `album` 列表。这样，任何健全的检查失败，我们就返回到 `album` 列表而不需要在这个函数里调用 `_redirect()` 多次。

模板是一个简单的表格：

```

zf-tutorial/application/views/indexDelete.tpl.php
<h1><?php echo $this->escape($this->title); ?></h1>
<?php if ($this->album) :?>
<form action="/zf-tutorial/index/delete" method="post">
    <p>Are you sure that you want to delete
        '<?php echo $this->escape($this->album->title); ?>' by
        '<?php echo $this->escape($this->album->artist); ?>'</p>
    <div>
        <input type="hidden" name="id"
            value="<?php echo $this->album->id; ?>" />
        <input type="submit" name="del" value="Yes" />
        <input type="submit" name="del" value="No" />
    </div>
</form>
<?php else: ?>
<p>Cannot find album.</p>
<?php endif;?>

```

## 故障排除

如果除了 `index/index` 能工作外，您发现其他控制模块不能如您所愿，最有可能的问题是路由不能确定您的网站在那个子目录下。就目前来看，通常发生的情况是您网站的 `url` 不同于 `web-root` 的目录。当前，在 `index.php` 中，我们试图用 `$_SERVER['PHP_SELF']` 来解决问题。如果这个不适用于，您应该自行设置适合您的服务器的 `$baseUrl` 的值

```

zf-tutorial/index.php
...

```

```
// setup controller
$baseUrl = '/mysubdir/zf-tutorial';
$frontController = Zend_Controller_Front::getInstance();
$frontController->setBaseUrl($baseUrl);
$frontController->setControllerDirectory('./application/controllers');
...
```

您应当用正确的URL指向index.php的路径来替换 '/mysubdir/zf-tutorial/'。例如：如果您的指向index.php的URL是[http://localhost/~ralle/zf\\_tutorial/index.php](http://localhost/~ralle/zf_tutorial/index.php)，正确的\$baseUrl的值就是 '/~ralle/zf\_tutorial/'。

## 结论

本教程使用Zend Framework开发了一个简单但功能齐全的MVC应用例程，我希望您能感兴趣和觉得有用。如果您发现任何错误，请给我发邮件到 [rob@akrabat.com](mailto:rob@akrabat.com)! 本教程只涉及到Zend Framework的最基本的用法，您应该去看看手册（<http://framework.zend.com/manual>）那里有更多的类可以使用，还有wiki(<http://framework.zend.com/wiki>) 中更多的内幕！如果您对开发Zend Framework感兴趣，到开发wiki(<http://framework.zend.com/developer>) 中去看看。

## 最后的想法

在写这个教程的同时，对我来说最明显的缺失的事情是做更好的模型。这就是为什么ActiveRecord模型当前非常流行！我注意到在下面几个月中，1.0版发行之前，Zend\_Db\_Table将有重大改善。在Framework wiki中有一个关于Zend\_Db\_Model的提议，大意是：“An object that wraps a row in a database table or view, encapsulates the database access, and allows to adds domain logic on that data” 这个对于用Zend Framework开发应用程序将非常有帮助。

总之，发展相当顺利和0.6版中的新的MVC系统非常好。

## 译者注

由于看好Zend Framework的发展前景，所以想系统地学习它的每个部分。如果能通过一个好的教程来学习，那将事半功倍。这个教程是我目前看到的最好的一个，限于我的水平，如果您觉得某些地方的翻译不是很恰当，请参考原文 <http://www.akrabat.com>。如果您有任何建议，请发邮件到 [jason@backupdiy.com](mailto:jason@backupdiy.com)