

和 Zend Framework 一起成长

By Rob Allen, www.akrobat.com

Document Revision 1.3.0

Copyright © 2006, 2007

翻译: Jason Qi, zft.backupdiy.com

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

注: 本教程已经在 Zend Framework Version 0.9 和 0.9.1 上测试过。它非常有希望和以后的版本一起工作, 但很确定, 它不和 0.9 以前的版本工作。(谁还希望用 0.9 以前的版本? 译者注)

对于 0.9 版的警示: 如果你已经下载了 Zend Framework 的 0.9 版, 那么你必须编辑 `library/Zend/Db/Table/Row/Abstract.php` 文件在第一行的开始加上 “<”

模型-视图-控制器 (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，它就是“news”的概念。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。

The Zend Framework uses the Model-View-Controller (MVC) architecture. This is used to separate out the different parts of your application to make development and maintenance easier.

Zend Framework 使用模型-视图-控制器（Model-View-Controller（MVC））架构。这个用来把你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. 本教程采用 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 0.9.1。对于本教程，你必须使用 0.9 和以上的版本。

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。这个目录结构假设你完全控制 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
      /filters
      /helpers
      /scripts
  /library
  /public
    /images
    /scripts
    /styles
```

正如你所看到的，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。支持的图像，脚本和 CSS 文件被存放在 `public` 目录下的不同子目录中。下载的 Zend Framework 文件放在 `library` 目录下。如果我们还需要其他库文件，都可以放在这里。

把 ZendFramework-0.9.1-Beta.zip 解压到一个临时目录。所有的文件都包含在一个叫做 ZendFramework-0.9.1-Beta 的目录。把 ZendFramework-0.9.1-Beta/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。

启动文件（Bootstrapping）

Zend Framework 的控制器，Zend_Controller，被设计支持使用 clean urls 的网站。为实现这个目的，所有的请求需要通过单一的 index.php 文件，这就是所知的启动文件（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/.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。如果你使用 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：

zf-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 文件的主意来自于 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/Loader.php";
```

```

Zend_Loader::loadClass('Zend_Controller_Front');

// setup controller
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory('./application/controllers');

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

```

注意我们没有在文件的结尾放?>, 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');

```

第几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, 你应该选择你自己的时区。

```

set_include_path('.' . PATH_SEPARATOR . './library'
    . PATH_SEPARATOR . './application/models/'
    . PATH_SEPARATOR . get_include_path());
include "Zend/Loader.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 `include Zend/Loader.php`, 这样我们就能访问 `Zend_Loader` 类, 在 `Zend_Loader` 类中有静态方法使我们能够加载其他 **Zend Framework** 类, 例如:

```

Zend_Loader::loadClass('Zend_Controller_Front');

```

`Zend_Loader::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。如果你在类库里使用相同的命名规则, 就可以用 `Zend_Loader::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。这个由 `Request` 对象完成, 它在自动检测正确的 `base URL` 方面做的很出色, 但如果它对你的设置不工作, 你可以用函数 `$frontController->setBaseUrl()` 来 `override`。

我们需要配置前端路由器, 这样它就知道从哪个目录里找出我们的控制器。

```

$frontController = Zend_Controller_Front::getInstance();
$frontController->setControllerDirectory('./application/controllers');
$frontController->throwExceptions(true);

```

因为这是一个教程并且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 `HTTP` 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, 你不应当给用户显示错误信息!

终于, 我们到了问题的核心并且可以运行我们的程序了:

```

// 运行!
$frontController->dispatch();

```

如果你用 http://localhost/zf_tutorial/ 去测试，致命的错误类似于：

Fatal error: Uncaught exception 'Zend_Controller_Dispatcher_Exception' with message 'Invalid controller specified (index)' in...

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 的表单
Edit Album	这个页面提供一个编辑 Album 的表单。
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。每个应用程序的页面叫做“**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** 的 **actions**。**Zend Framework** 的 **MVC** 系统也支持把控制器组成模块 (**module**)，但这个教程没有足够大到必须用它。

Zend Framework 控制器把 **index** 作为一个缺省的 **action** 而保留为特别的 **action**。这样，对于 <http://localhost/zf-tutorial/news/> 这样的 url，在 **news** 控制器里的 **index** **action** 将被执行。**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**。

作为一个简单的教程，我们不打算涉及“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用于 **albums**，我们将把他们当作四个 **actions** 组织到一个单个的控制器里。我们将使用缺省的控制器和四个 **actions**，如下表：

Page	Controller	Action
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，这个类必需在叫做 `{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并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：

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

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方案，它放在 **Zend_Controller_Action** 里。有两个助手函数：**initView()** 和 **render()**。**initView()** 函数创建 **Zend_View** 对象并赋值给给 **\$view** 属性，准备好让我们赋值给它。它也设置 **Zend_View** 对象去查找 **views/scripts/{controller name}**，以便视图脚本被调用。视图脚本通过 **render()** 来调用，它将调用脚本 **{action_name}.phtml** 并追加到响应对象的 **body** 里。响应对象用来把所有的头，**body content** 和 **MVC** 系统产生的异常放在一起。前端控制器接着自动地发送头以及紧接着的在 **dispatch** 尾端的 **body content**。

To integrate the view into our application we need to initialise the view in the **init()** function and then ensure we call **render()** in each action. We also need to create some view files with test display code.

为了集成视图到程序中，我们需要在 **init()** 函数里初始化视图并确保在每个 **action** 里调用 **render()**。我们也需要创建一些带有测试显示代码的视图文件。

如下修改 **IndexController**。（被修改的部分是黑体字）。正如你看到的，我们添加一个新的函数叫做 **init()**，它被 **Zend_Controller_Action** 的构造器自动地调用。这就确保视图能够在开始被初始化并且我们确信它已经准备好能被用在 **action** 函数中。

zf-tutorial/application/controllers/IndexController.php

```
<?php

class IndexController extends Zend_Controller_Action
{
    function init()
    {
        $this->initView();
    }

    function indexAction()
    {
        $this->view->title = "My Albums";
        $this->render();
    }

    function addAction()
    {
        $this->view->title = "Add New Album";
        $this->render();
    }

    function editAction()
    {
        $this->view->title = "Edit Album";
        $this->render();
    }

    function deleteAction()
    {
        $this->view->title = "Delete Album";
        $this->render();
    }
}
```

在每个函数中，我们分配一个 **title** 变量给视图属性，接着，调用 **render()**来显示视图模板。注意在这点上实际的显示并没有发生—它在 **dispatch** 进程结束后由前端控制器来完成。

我们现在需要做四个视图文件。这些文件就是所知的模板并且 **render()**方法期望每个模板文件的命名在它的 **action** 之后并且带有 **.phtml** 的扩展名来表明它是模板文件。模板文件必须在名字在控制器之后的子目录中，所以这四个文件是：

zf-tutorial/application/views/scripts/index/index.phtml

```
<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/scripts/index/add.phtml

```
<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/scripts/index/edit.phtml

```
<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/scripts/index/delete.phtml

```
<html>
<head>
    <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
    <h1><?php echo $this->escape($this->title); ?></h1>
</body>
</html>
```

测试每个控制器/**action**，它们应该显示出四个黑体字标题。

共同的 HTML 代码

很快，明显地在我们的视图里有许多相同的 HTML 代码。我们把他们提取到两个文件 **header.phtml** 和 **footer.phtml** 中，这两个文件放在脚本目录中。我们用它们存放从视图模版里提取的“共同的”HTML：

新文件是：

zf-tutorial/application/views/scripts/header.phtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```

        <title><?php echo $this->escape($this->title); ?></title>
</head>
<body>
<div id="content">

```

(注意我们已经纠正了 HTML 并且我们现在也已经适应了!)

zf-tutorial/application/views/scripts/footer.phtml

```

</div>
</body>
</html>

```

再一次，我们的视图需要修改：

zf-tutorial/application/views/scripts/index/index.phtml

```

<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>

```

zf-tutorial/application/views/scripts/index/add.phtml

```

<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>

```

zf-tutorial/application/views/scripts/index/edit.phtml

```

<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>

```

zf-tutorial/application/views/scripts/index/delete.phtml

```

<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('footer.phtml'); ?>

```

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。因为 URL 不指向正确的根目录，我们不知道如何定位 CSS 文件，这会导致一个微小的问题。为了解决它，我们使用 `getBaseUrl()` 函数，它是请求的一部分并把它传递给视图。这提供给我们所不知道的一点 URL。

修改 `IndexController::init()` 看上去象这样：

zf-tutorial/application/controllers/IndexController.php

```

...
function init()
{
    $this->initView();
    $this->view->baseUrl = $this->_request->getBaseUrl();
}
...

```

我们需要把 CSS 文件添加到 `header.phtml` 的 `<head>` 一节中：

zf-tutorial/application/views/scripts/header.phtml

```

...
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title><?php echo $this->escape($this->title); ?></title>
    <link rel="stylesheet" type="text/css" media="screen"
        href="<?php echo $this->baseUrl;?>/public/styles/site.css" />
</head>

```

...

最后，我们需要一些 CSS 风格：

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: #800000;
    background-color: transparent;
}

#content {
    width: 770px;
    margin: 0 auto;
}

label {
    width: 100px;
    display: block;
    float: left;
}

#formbutton {
    margin-left: 100px;
}

a {
    color: #800000;
}
```

This should make it look slightly prettier!

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，你应该使用你自己的用户名，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（尽管如果你想，所有的 section 都可以被加载）。它支持 section 名字中的符号并允许加载另外的 section。Zend_Config_Ini 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，主机，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_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Config_Ini');
Zend_Loader::loadClass('Zend_Registry');

// 加载配置
$config = new Zend_Config_Ini('./application/config.ini', 'general');
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);

// 设置控制器
...

```

修改的部分是黑体字。我们加载我们将要用的类（Zend_Config_Ini 和 Zend_Registry）并加载 application/config.ini 中的 'general' section 到 \$config 对象。最后，我们分配 \$config 对象给注册表，这样它可以在程序的任何地方被取出来使用。

注：在这个教程里，我们实际上不需要把 \$config 存到注册表里，但是在“真正的”项目中，你可能在 INI 文件里有很多配置信息，而不仅仅是数据库。同样，要知道如果你不小心，注册表有一点象全局变量并导致对象之间的依赖，而他们并不互相依靠。

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。强调一下，我们在启动文件里完成它（黑体字）：

Relevant part of zf-tutorial/index.php

```
...
Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Config_Ini');
Zend_Loader::loadClass('Zend_Registry');
Zend_Loader::loadClass('Zend_Db');
Zend_Loader::loadClass('Zend_Db_Table');

// 加载配置
$config = new Zend_Config_Ini('./application/config.ini', 'general');
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);

// 设置数据库
$db = Zend_Db::factory($config->db->adapter,
    $config->db->config->asArray());
Zend_Db_Table::setDefaultAdapter($db);

// 设置控制器
...
```

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检索功能。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。如何命名类无关紧要，但类名和表名用同样的名字比较有意义。这样，类名就叫做 **Album** 因为表名是 **album**。为了告诉 Zend_Db_Table 它将管理的表名，我们必须设置保护属性 **\$_name** 为表名。并且，Zend_Db_Table 假定表有一个主键叫做 **id**，它能够自动增长。如果需要的话，这个字段可以更改。

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
{
    protected $_name = 'album';
}
```

```
}
```

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` 提供了所有我们需要的函数。然而，如果你需要特殊的函数管理你的模型，你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，它使你要寻找的精确数据的集合有效。你也可以告诉 `Zend_Db_Table` 关于相关的表并且它也可以获取相关的数据。

列表 Albums

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。

很清楚，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

```
...
function init()
{
    $this->initView();
    $this->view->baseUrl = $this->_request->getBaseUrl();
    Zend_Loader::loadClass('Album');
}
...
```

注：这是一个用 `Zend_Loader::loadClass()` 来加载我们自己的类并且能够工作的例子，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()` 里的表里列出 album：

zf-tutorial/application/controllers/IndexController.php

```
...
function indexAction()
{
    $this->view->title = "My Albums";
    $album = new Album();
    $this->view->albums = $album->fetchAll();
    $this->render();
}
...
```

函数 `Zend_Db_Table::fetchAll()` 返回一个 `Zend_Db_Table_Rowset`，它允许我们通过返回的记录在视图模板文件里重复使用：

zf-tutorial/application/views/scripts/index/index.phtml

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<p><a href="<?php echo $this->baseUrl; ?>/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="<?php echo $this->baseUrl; ?>/index/edit/id/<?php
```

```

        echo $album->id;?>">Edit</a>
        <a href="<?php echo $this->baseUrl; ?>/index/delete/id/<?php
        echo $album->id;?>">Delete</a>
    </td>
</tr>
<?php endforeach; ?>
</table>
<?php echo $this->render('footer.phtml'); ?>

```

<http://localhost/zf-tutorial/> (or wherever you are following along from!) should now show a nice list of (two) albums.

<http://localhost/zf-tutorial/> (或任何你使用的地址) 应该可以列出一个很好的 (两行) album 列表。

添加新的 Albums

我们现在就可以做添加新的 album 的函数。有两点：

- 显示一个表单让用户提供信息
- 处理这个表单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()
{
    $this->view->title = "Add New Album";

    if (strtolower($_SERVER['REQUEST_METHOD']) == 'post') {
        Zend_Loader::loadClass('Zend_Filter_StripTags');
        $filter = new Zend_Filter_StripTags();

        $artist = $filter->filter($this->_request->getPost('artist'));
        $artist = trim($artist);
        $title = trim($filter->filter($this->_request->getPost('title')));

        if ($artist != '' && $title != '') {
            $data = array(
                'artist' => $artist,
                'title' => $title,
            );
            $album = new Album();
            $album->insert($data);

            $this->_redirect('/');
            return;
        }
    }

    // set up an "empty" album
    $this->view->album = new stdClass();
    $this->view->album->id = null;
    $this->view->album->artist = '';
    $this->view->album->title = '';

    // additional view fields required by form
    $this->view->action = 'add';
    $this->view->buttonText = 'Add';

    $this->render();
}
...

```

注意我们如何检查变量 `$_SERVER['REQUEST_METHOD']` 来看表单是否被提交。如果表单被提交，我们就从用 `Zend_Filter_StripTags` 类产生的 `post` 数组里提取 `artist` 和 `title`，以确保里面没有 `html` 代码。接着，假定它们已经被填入，我们用模型类，`album()`，把信息插入到数据库的表里。

在添加了一个 `album` 之后，我们用控制器的 `_redirect()` 方法重定向来返回到程序的根（`root`）。

终于，我们建立了我们将在模板里使用的视图表单。进一步看，我们能预计到编辑 `action` 的表单和这个非常类似，所以我们将使用共同的模板文件（`_form.phtml`），模板文件将被 `add.phtml` 和 `edit.phtml` 调用。

用于添加 `album` 的模板如下：

zf-tutorial/application/views/scripts/index/add.phtml

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('index/_form.phtml'); ?>
<?php echo $this->render('footer.phtml'); ?>
```

zf-tutorial/application/views/scripts/index/_form.phtml

```
<form action="<?php echo $this->baseUrl ?>/index/<?php
echo $this->action; ?>" method="post">
<div>
  <label for="artist">Artist</label>
  <input type="text" name="artist"
    value="<?php echo $this->escape(trim($this->album->artist));?>" />
</div>
<div>
  <label for="title">Title</label>
  <input type="text" 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 `_form.phtml`，用 `$this->action` 比 `hard coding` 要好些。相似地，我们在提交按钮里显示的文字也使用变量。

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()
{
  $this->view->title = "Edit Album";
  $album = new Album();

  if (strtolower($_SERVER['REQUEST_METHOD']) == 'post') {
    Zend_Loader::loadClass('Zend_Filter_StripTags');
    $filter = new Zend_Filter_StripTags();

    $id = (int)$this->_request->getPost('id');
    $artist = $filter->filter($this->_request->getPost('artist'));
    $artist = trim($artist);
    $title = trim($filter->filter($this->_request->getPost('title')));
```

```

        if ($id !== false) {
            if ($artist != '' && $title != '') {
                $data = array(
                    'artist' => $artist,
                    'title' => $title,
                );
                $where = 'id = ' . $id;
                $album->update($data, $where);

                $this->_redirect('/');
                return;
            } else {
                $this->view->album = $album->fetchRow('id='.$id);
            }
        }
    } else {
        // album id should be $params['id']
        $id = (int)$this->_request->getParam('id', 0);
        if ($id > 0) {
            $this->view->album = $album->fetchRow('id='.$id);
        }
    }

    // additional view fields required by form
    $this->view->action = 'edit';
    $this->view->buttonText = 'Update';

    $this->render();
}
...

```

注意程序不在“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/scripts/index/edit.phtml

```

<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->render('index/_form.phtml'); ?>
<?php echo $this->render('footer.phtml'); ?>

```

改进!

你可能已经注意到 `AddAction()` 和 `EditAction()` 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。

这个就留给你, 亲爱的读者... ..

删除一个 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你去点击删除。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单, 当用户点“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()
{
    $this->view->title = "Delete Album";

    $album = new Album();
    if (strtolower($_SERVER['REQUEST_METHOD']) == 'post') {
        Zend_Loader::loadClass('Zend_Filter_Alpha');
        $filter = new Zend_Filter_Alpha();

        $id = (int)$this->_request->getPost('id');
        $del = $filter->filter($this->_request->getPost('del'));

        if ($del == 'Yes' && $id > 0) {
            $where = 'id = ' . $id;
            $rows_affected = $album->delete($where);
        }
    } else {
        $id = (int)$this->_request->getParam('id');
        if ($id > 0) {
            // only render if we have an id and can find the album.
            $this->view->album = $album->fetchRow('id='.$id);

            if ($this->view->album->id > 0) {
                $this->render();
                return;
            }
        }
    }

    // redirect back to the album list unless we have rendered the view
    $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单还是应该删除。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，任何不同的健全的检查失败，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单：

zf-tutorial/application/views/indexDelete.tpl.php

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php if ($this->album) :?>
<form action="<?php echo $this->baseUri ?>/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;?>
<?php echo $this->render('footer.phtml'); ?>

```

故障排除

如果除了 `index/index` 能工作外，你发现其他控制模块不能如你所愿，最有可能的问题是路由不能确定你的网站在那个子目录下。就目前来看，通常发生的情况是你网站的 `url` 不同于 `web-root` 的目录。

如果这个缺省代码不适用，你应该自行设置适合你的服务器的 `$baseUrl` 的值：

zf-tutorial/index.php

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

你应当用正确的 `URL` 指向 `index.php` 的路径来替换 `'/mysubdir/zf-tutorial/'`。例如：如果你的指向 `index.php` 的 `URL` 是 `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，我希望能感兴趣和觉得有用。如果你发现任何错误，请给我发邮件到 rob@akrabat.com！

本教程只涉及到 `Zend Framework` 的最节本的法，你应该去看看手册（<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！如果你对开发 `Zend Framework` 感兴趣，到开发 `wiki`（<http://framework.zend.com/developer>）中去看看... ..

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，如果你觉得某些地方的翻译不是很恰当，请参考原文 <http://www.akrabat.com>。如果你有任何建议，请发邮件到 jason@backupdiy.com，谢谢！