

Zend Framework 入门教程(简体中文版)

Getting Started With the Zend Framework

Author: Rob Allen, www.akrobat.com

Document Revision: 1.5.2

Copyright© 2006, 2008

翻 译: Altair (eniact2008@163.com)

中文版本号: v0.12

【翻译说明】这是 Zend Framework 的非常经典的入门教程，它的原作者 Rob Allen 是《Zend Framework In Action》一书的作者。在翻译这个版本之前，只知道 Jason Qi 翻译的 0.9 版(实际内容是关于 Zend Frame 0.6 版本的)。因为自 0.9 版以来 Zend Framework 已经有了很大的变化，教程也有了比较大的改变。因此才决定重新翻译一次。翻译完成后才发现 Jason Qi 翻译的最新版本已经是 1.4.3 了。但毕竟已经翻译完成，而且 Jason 翻译的还不是最新版本，因此还是决定将这个版本的翻译发布出来。因此，实际上这个翻译版本是根据英文版 v1.5.2 完全重新翻译的。

这个翻译版本是比较初步的一个版本，里面可能还有不少翻译错误，主要是中文表述的问题，我会一直对其进行修订。如果你发现文中有一些错误，那很有可能是我在翻译的时候造成的，欢迎大家将发现的错误及时通知我，以便我能及时更新，以保证此中文版本跟原文一样的高品质。Thanks!

本教程的英文版原文链接: <http://akrobat.com/zend-framework-tutorial/>

Jason Qi 翻译的本教程的早期版本可以在这里找到: <http://zft.backupdiy.com/download/>

本教程首发: <http://bbs.phpchina.com>, <http://www.phpatoz.com> (Under construction)

History:

2008/05/30 v0.01 First Release.

2008/05/30 v0.02 Bug Fix

2008/06/01 v0.03 Bug Fix

2008/06/02 v0.04 Bug Fix

2008/06/05 v0.10/0.11 对文字做了比较大的修订。

2008/06/06 v0.12 Bug Fix

What's New?

v0.12 P12 “样式”小节中有关辅助函数类保存的地址 helper 应为 helpers 感谢 phpeye.com/lxq73061 指出此错误。

v0.10 对文字做了比较大的修订。

v0.04 (1) 因原文中有关目录配置的地方有一处写得有点矛盾，因此我在 v0.03 版的译注是针对使用虚拟主机的配置的。对于不使用虚拟主机的情况，我的译注与文中后来访问的 URL 有冲突。新版中对此作了一些修改。原则上，v0.03 版的译注是没有问题的。

本教程对使用 Zend Framework 来开发数据库驱动的应用程序作了非常基本的介绍。

注意：本教程在 Zend Framework 1.5 版下测试通过。在以后的 1.5.x 版本上，它也有很大可能正常运行，但在 1.5 以前的版本上本教程不能运行。

Model-View-Controller架构

下面是传统的 PHP 应用程序编写方式：

```
<?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>
    }
?>
</table>
<?php include "footer.php"; ?>
```

对于采用这种方式编写的应用程序来说，为了适应其生命周期中客户不断变化的需求，将不得不在代码的多个地方打上补丁，最后导致它变得无法维护。

提高程序的可维护性的一种方法是将这个程序的代码分成如下三个不同的部分（通常也是独立的文件）：

模型	应用程序的模型部分关心的是欲显示的数据的细节。在上面的示例代码中模型是“news”。因此，模型通常关注的是应用程序的业务逻辑部分，关注的是如何使用数据库来读取和保存数据。
视图	视图关心的是用户显示的部分，它通常是 HTML。
控制器	控制器将特定的模型和视图结合起来，保证将正确的数据显示到页面上。

Zend Framework 使用 Model-View-Controller(MVC)架构。它将程序中不同部分独立开来，使得应用程序的

开发和维护更加容易。

需求

使用 Zend Framework 需要下列环境：

- PHP 5.1.4 或以上版本
- 支持 mod_rewrite 功能的 Web 服务器

一些假设

本教程假定你运行 PHP 5.1.4 或以上版本，外加 Apache Web 服务器。Apache 必须已安装并正确配置了 mod_rewrite 扩展。

必须保证 Apache 已配置成支持.htaccess 文件的模式。通常这可以通过在 httpd.conf 中将

```
AllowOverride None
```

改成

```
AllowOverride All
```

来实现。更详细的设置方法可以在 Apache 发行文档中找到。如果没有正确配置 mod_rewrite 及.htaccess，那么除了本教程的首页外你将不能看到任何其它的页面。

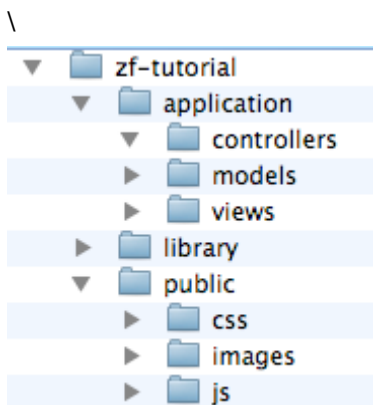
获取框架

Zend Framework 可以从 <http://framework.zend.com/download> 下载（有 .zip 或 .tar.gz 两种格式）。

目录结构

虽然 Zend Framework 对目录结构没有特别要求，但其手册上还是推荐了一种常用的目录结构，本教程也使用这种目录结构。这种结构要求你能完全控制 Apache 的配置文件，以便可以将大多数的文件存放在 web 的根目录之外。

首先在 web 服务器的根目录下创建一个 zf-tutorial 目录¹，然后分别创建下面的子目录来存放网站²的文件：



我们使用单独的目录来保存应用程序的模型、视图和控制器文件。public 目录是网站的根目录³，这样就可

¹ 译注：按道理此处的 Web 服务器的根目录不应指网站的根目录，即不是 Apache 配置文件中的 DocumentRoot 目录。但实际上本教程中这个目录就是指的是 DocumentRoot 目录。原文这个地方写得与上文中“将大多数文件存放在 web 的根目录之外”的原则有点矛盾。

² 译注：此处的网站是将教程中开发的应用程序看作一个独立的网站的应用来对待的。因此，在下文中有时会将网站或(应用)程序的概念混用，只要记住它都是指我们在这个教程中正在开发的应用程序就可以了。

以通过URL <http://localhost/zf-tutorial/public/> 来访问我们的程序。并且应用程序的绝大多数文件都不能直接通过Apache来访问，从而提高了系统的安全性。

注意：

在一个包括其它网站的服务器中，最好还是为我们的网站创建一个虚拟主机，将其根目录设置为public子目录。例如你可以创建一个 zf-tutorial.localhost的虚拟主机⁴：

```
<VirtualHost *:80>
    ServerName zf-tutorial.localhost
    DocumentRoot /var/www/html/zf-tutorial/public
    <Directory "/www/cs">
        AllowOverride All
    </Directory>
</VirtualHost>
```

这样就可以通过 <http://zf-tutorial.localhost/> 来访问该网站。(使用这种方式必须修改 /etc/hosts 或 c:\windows\system32\drivers\etc\hosts文件，将 zf-tutorial.localhost映射到 127.0.0.1)。

辅助的图像文件，JavaScript 文件和 CSS 文件分别保存在 public 目录下的不同文件夹中。下载后的 Zend Framework 文件将保存在 library 文件夹中。如果需要使用其它的库文件，也可以放在该文件夹下。

将下载的 Zend Framework 软件包，我使用的是 ZendFramework-1.5.0.zip，解压到一个临时目录中，解压后的所有文件都放在 ZendFramework-1.5.0 子文件夹下，将它的子目录 library/Zend 拷贝到 zf-tutorial/library/ 文件夹，现在 zf-tutorial/library/ 目录中包含了 Zend 子目录。

引导文件

Zend Framework 控制器类 Zend_Controller 支持网站使用“干净的URL”⁵。为此所有的请求都需要通过 index.php 进入。这就是通常所说的前端控制器(Front Controller)设计模式。它为我们的应用程序的所有页面提供了一个中心控制点并确保程序的运行环境已经正确设置。要完成这一切，都必须在zf-tutorial/public目录下创建一个.htaccess文件：

zf-tutorial/public/.htaccess

```
# Zend Framework rewrite 规则
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule .* index.php
# 安全性考虑：不允许列表目录内容
Options -Indexes
# PHP 设置
php_flag magic_quotes_gpc off
php_flag register_globals off
php_flag short_open_tag on
```

³ 译注：此处意味着我们必须将.htaccess 及 index.php 两个文件保存在 public 目录下。

⁴ 译注：如果配置虚拟主机，必须将此处的 DocumentRoot 以及 Directory 中的路径换成自己的实际目录。

⁵ 译注：所谓“干净的URL”是指URL中不含那些杂乱的参数，如：<http://localhost/index/add/>，而不是 <http://localhost/index.php?action=add>。

RewriteRule 非常简单，可理解为“对所有不能映射到磁盘上已存在的文件的 url，都用 index.php 来代替”。

为了安全起见，我们设定了一些 PHP 的 ini 设置；我们还将 short_open_tag 选项设置为 on，因为将来视图文件可能会用到它。当然这些设置可能已经正确设置过了，但我们必须确保这一点。注意只有在使用 PHP 模块(mod_php)的方式下才可以在.htaccess 文件中使用 php_flag 标记。如果使用 CGI/FastCGI 模式，必须保证在 php.ini 中正确设置了这些参数。注意，为了让.htaccess 起作用，必须在 httpd.conf 中将配置指令 AllowOverride 设置为 All。

引导文件：index.php

zf-tutorial/public/index.php 是应用程序的引导文件，我们用下面的代码开始我们的教程：

zf-tutorial/public/index.php

```
<?php
error_reporting(E_ALL|E_STRICT);
ini_set('display_errors', 1);
date_default_timezone_set('Europe/London');
// 目录设置和类装载
set_include_path('! . PATH_SEPARATOR !../library/'
    . PATH_SEPARATOR . !../application/models'
    . PATH_SEPARATOR . get_include_path());
include "Zend/Loader.php";
Zend_Loader::registerAutoload();
// 设置控制器
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory('../application/controllers');
// run!
$frontController->dispatch();
```

注意在文件的结尾我们没有加上 ?>，因为在文件的结尾它不是必需的。这样可以避免产生一些难于调试的错误问题。例如，在使用 header() 函数来重定向(redirect)时，如果在其前面某个包含文件中 ?> 后面不小心加上了空格就会出现错误。

下面开始详细解释这个文件：

```
error_reporting(E_ALL|E_STRICT);
ini_set('display_errors', 1);
date_default_timezone_set('Europe/London');
```

前 2 行保证在程序出错时能看到相应的错误信息。第 3 行设置了时区（PHP 5.1+以后要求如此）。当然，你应该选择自己所在地的时区。

```
// 目录设置和类装载
set_include_path('!' . PATH_SEPARATOR . '../library/'
    . PATH_SEPARATOR . '../application/models'
    . PATH_SEPARATOR . get_include_path());
include "Zend/Loader.php";
Zend_Loader::registerAutoload();
```

Zend Framework 在设计时就要求它必须在包含路径(include path)中。为了将来很容易的装入模型类，我们也将模型类目录加到了包含路径中。在正式开始之前，我们首先包含进 Zend/Loader.php 文件，这样就可以使用 Zend_Loader 类了。接着调用 Zend_Loader 类的 registerAutoload()成员函数，这样以后在实例化 Zend Framework 对象时就可自动装入它们。

```
// 设置控制器
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory('../application/controllers');
```

必须设置前端控制器使得它知道从哪里去找我们的控制器类。

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

因为这只是一个教程并且通常只是在测试系统上运行，因此我决定让前端控制器(front controller)抛出它遇到的所有异常。默认情况下，前端控制器将捕获这些异常并将其传递到 ErrorController 控制器，但这容易造成 Zend Framework 新手的困惑。直接抛出异常相对简单，这样错误也很容易看到。当然，在实际运行中的服务器，不管怎样都不能将错误直接显示给用户。

前端控制器使用了一个路由类将请求的 URL 映射到相应的显示页面内容的 PHP 函数。路由类要正常工作，必须知道该 URL 中哪些部分用于映射到 index.php 的路径，这样它才可以从其后余下的部分中读取 URI 元素。这个工作由 Request 对象来完成。Request 对象可以有强大的自动检测正确的基地址(base URL)的能力，但如果它在你的设置环境中不能正常工作，可以使用\$frontController->setBaseUrl()来覆盖它。

现在到了程序的核心部分了，程序正式开始运行：

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

此时如果你用浏览器来测试<http://localhost/zf-tutorial/public/>，你会看到一条致命错误信息：

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

这个错误信息告诉我们还没有创建 `application` 的内容。在此之前，我们最好先讨论一下要做什么，而这就是我们下一节的内容。

网站内容

我们打算建立一个非常简单的库存管理系统，用于显示我们收藏的 CD。主页上显示我们收藏的 CD 列表，并允许我们增加、修改、删除 CD。CD 信息保存在如下结构的数据库中：

字段名	类型	是否允许为空?	备注
id	Integer	No	主键，自动增加
artist	Varchar(100)	No	
title	Varchar(100)	No	

需要的页面

在我们的程序中需要下列四个页面：

主页	显示所有唱片列表，提供修改、删除的链接。另外还提供一个新增唱片的链接。
添加新唱片页面	提供了一个表单，用于添加新唱片
修改唱片页面	提供了一个表单来修改唱片
删除唱片页面	本页确认并删除唱片

页面的组织

在我们继续下一步工作之前，理解 Zend Framework 要求页面如何组织至关重要。程序的每个页面就是一个“动作” (action)，而动作又组合到控制器中。例如，对于 <http://localhost/public/zf-tutorial/news/view> 形式的 URL 来说，控制器是 `news`，动作是 `view`。这样可以将相关的动作组合到一起。例如，一个 `news` 控制器可以有 `list`、`archive` 和 `view` 动作。Zend Framework 的 MVC 系统还支持将不同的控制器组合在一起，但这个程序没有那么复杂，所以我们不用考虑这些。

默认情况下，Zend Framework 保留了一个特殊的动作，称之为 `index`，并将它作为默认动作。因此 URL <http://localhost/public/zf-tutorial/news> 将执行控制器 `news` 的动作 `index`。同样，如果不提供控制器名称，那么就使用 `index` 作为默认的控制器的。因此 <http://localhost/public/zf-tutorial/> 将执行控制器 `index` 的动作 `index`。

作为一个简单的教程，我们不考虑那些复杂的情况如“登录”等，它们也许需要另外一篇教程才能讲清楚..... 现在我们有四个页面，并且都跟唱片有关，因此我们可以将它们组合到包含四个动作的控制器中。在这里我们使用默认的控制器的，它的四个动作如下：

页面	控制器	动作
主页	Index	index
添加新唱片页面	Index	add
修改唱片页面	Index	edit

删除唱片页面	Index	delete
--------	-------	--------

编写控制器

现在可以编写控制器类了。在 Zend Framework 中，控制器类名必须为 {Controller name}Controller。注意控制器类名称 (Controller name) 的首字母必须大写。控制器类必须保存在 application/controllers 目录中，文件名为 {Controller name}Controller.php。同样，这里 {Controller name} 必须以大写字母开头，所有其它字母均为小写。每个动作 (action) 必须是一个名为 {action}Action 的 public 函数，这里 {action} 必须以小写字母开头并且全是小写字母。混合大小写的控制器名称或动作名称是允许的，但在使用它们之前你必须理解这种用法有其特殊的规则。建议你在使用它们之前先阅读一下相关的文档。

因此我们的控制器名为 IndexController，它在 zf-tutorial/application/controllers/IndexController.php 文件中定义。我们先创建该文件并编写它的框架代码：

zf-tutorial/application/controllers/IndexController.php

```
<?php
class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
    }

    function addAction()
    {
    }

    function editAction()
    {
    }

    function deleteAction()
    {
    }
}
```

现在已经设置了将要用到的四个动作。在视图代码完成后，它们就可以工作了。每个动作对应的URL如下⁶：

URL	Action
http://localhost/zf-tutorial/public	IndexController::indexAction()
http://localhost/zf-tutorial/public/index/add	IndexController::addAction()
http://localhost/zf-tutorial/public/index/edit	IndexController::editAction()

⁶ 译注：对于使用虚拟主机的配置，其对应的URL类似于 <http://zf-tutorial.localhost/index> ...

<http://localhost/zf-tutorial/public/index/delete>

IndexController::deleteAction();

现在已经有有了一个可以工作的“路由器”，每个页面都定义了一个动作函数。接下来我们将开始创建视图。

编写视图

毫不奇怪，Zend Framework 的视图组件就叫做 Zend_View。视图组件允许我们将显示页面的代码与动作函数的代码分离。

Zend_View 的基本用法如下：

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

显而易见，如果直接将这些骨架代码写到每一个动作函数中，我们就不得不到处无聊的重复这些跟动作关系不大的代码。我们希望能将视图的初始化代码放在其它的地方，然后在每个动作函数中直接访问已初始化过的视图对象。

Zend Framework 的设计人员已经预见到了这种情况，建立了一个“动作辅助类(action helper)”来解决这个问题。这个辅助类是 Zend_Controller_Action_Helper_ViewRenderer，它初始化视图属性（`$this->view`），我们可以使用这个属性，并用来显示视图文件。显示过程首先通知 Zend_View 对象在 `views/scripts/{controller name}` 目录中查找显示脚本，然后显示与动作名称相同，扩展名为 `.phtml` 的显示脚本。即显示的视图文件名为 `views/scripts/{controller name}/{action name}.phtml`，显示的内容将附加到应答对象（Response Object）的应答内容(body)中。应答对象将 MVC 系统生成的 HTTP 头，应答内容以及所有异常信息整合在一起。前端控制器在调度过程的结尾处自动将 HTTP 头以及应答内容返回给用户。

我们需要创建一些视图文件来将视图集成到我们的程序中。为了验证这些视图文件是否能正常工作，我们在控制器的动作中增加了一些与动作相关的内容（页面标题）。

首先需要修改 IndexController，修改的内容见下面的黑体部分：

zf-tutorial/application/controllers/IndexController.php

```
<?php
class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
        $this->view->title = "My Albums";
    }

    function addAction()
```

```
{
    $this->view->title = "Add New Album";
}

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

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

在每个动作函数中，我们做的所有工作就是为视图属性(view)属性增加了一个 title 变量。注意在这个时候实际的显示操作还没有开始——它是在前端控制器在调度过程的最后进行的。

现在需要在我们的程序中增加四个视图文件。前面已经提到过，这些文件称为显示脚本(scripts)或显示模板(templates)，每一个显示模板根据相应的动作命名，并且扩展名为.phtml。模板文件必须放在与控制类同名的子目录中，因此这四个模板文件分别是：

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>
```

现在浏览上面的四个 url 地址测试每个控制器/动作，在浏览器中应该可以看到四个页面的标题了。

相同的 HTML 代码

很快你就会发现在我们的视图中有大量相同的 HTML 代码，由于这个问题非常普遍，因此 Zend Framework 中专门设计了 Zend_Layout 组件来解决这个问题。Zend_Layout 组件允许我们将相同的头部和尾部代码移到独立的布局显示脚本(layout view script)中，并在布局显示脚本中包含与正在执行的动作相关的显示代码。

为此我们需要对程序做一些修改。首先要确定的是将布局显示脚本保存到哪里。建议的路径是在 application 目录下，因此我们在 zf-tutorial/application 目录中创建一个 layouts 子目录。

其次我们需要通知启动文件启用 Zend_Layout，这只需要在 public/index.php 中增加一行代码即可(黑体部分):

zf-tutorial/public/index.php:

```
...
$frontController->throwExceptions(true);
$frontController->setControllerDirectory('../application/controllers');
Zend_Layout::startMvc(array('layoutPath'=>'../application/layouts'));
// run!
$frontController->dispatch();
```

startMvc()函数内部为前端控制器安装了一个插件(plugin)，该插件保证 Zend_Layout 组件在调度过程的最后显示布局脚本，而该布局脚本中包含了动作的显示脚本。

现在我们就需要这个布局显示脚本了。布局显示脚本默认的文件名是 layout.phtml，它保存在在 layouts 目

录中，内容如下：

zf-tutorial/application/layouts/layout.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">
    <h1><?php echo $this->escape($this->title); ?></h1>
    <?php echo $this->layout()->content; ?>
</div>
</body>
</html>
```

注意我们创建了一个符合 XHTML 规范的标准 HTML 文件来显示页面。因为页面中<h1>标记中的标题在所有页面上都显示，我们将它移到了布局文件中，并且使用 escape()函数来保证它被恰当的编码。

为了显示当前动作的显示脚本(view scripts)，我们使用了 layout()辅助函数: echo \$this->layout()->content; 它将内容显示到 content 占位符中。这也意味着动作的显示脚本在布局显示脚本之前执行。

现在我们可以来清理一下 4 个动作的显示脚本了，因为还没有什么特殊的東西需要放到它里面,所以我们可以将 index.phtml, add.phtml, edit.phtml, delete.phtml 文件内容清空。

现在再测试一下上述四个 URL，你会发现看到的内容与上次一模一样！然而它们有一个关键的不同，就是这次所有的工作都是利用布局显示脚本(layout)来完成的。

样式

虽然只是一个教程，我们还是需要一个 CSS 文件来使得我们的程序看起来漂亮一些。因为 URL 并不是指向正确的根目录，这使得我们在如何引用 CSS 文件时碰到一点小麻烦。这时我们可以通过创建自己的视图辅助类 baseUrl()来解决这个问题。该辅助类通过收集请求对象(request object)的相关信息使得我们所不知道的实际 URL。

视图辅助类保存在 application/views/helpers 文件夹中名为{Helper name}.php（文件名第一个字母大写）文件中，视图辅助类的名字必须是 Zend_Controller_Helper_{Helper name}形式（同样，Helper name 的第一个字母必须大写）。在这个类中，必须有一个名为(helper name)()的函数（函数名的第一个字母必须小写—可别忘了!）。在我们的例子中，这个文件名为 BaseUrl.php，内容如下：

zf-tutorial/application/views/helpers/BaseUrl.php

```
<?php
class Zend_View_Helper_BaseUrl
{
    function baseUrl()
    {
        $fc = Zend_Controller_Front::getInstance();
        return $fc->getBaseUrl();
    }
}
```

zf-tutorial/application/layouts/layout.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();?>/css/site.css" />
</head>
...
```

最后, 我们需要一些 CSS 样式:

```
body,html {
    margin: 0 5px;
    font-family: Verdana,sans-serif;
}
h1 {
    font-size:1.4em;
    color: #008000;
}
a {
    color: #008000;
}
/* Table */
th {
    text-align: left;
}
td, th {
    padding-right: 5px;
}
/* style form */
form dt {
    width: 100px;
    display: block;
```

```
float: left;
clear: left;
}
form dd {
margin-left: 0;
float: left;
}
form #submitbutton {
margin-left: 100px;
}
```

现在看起来要漂亮一些了。但可以告诉你，我不是一个设计师，所以别指望它有多漂亮！

数据库

既然我们已将程序的控制与视图分离开来了，现在该是考虑模型的时候了。记住，模型是用来处理程序的核心议题（即所谓的“商业规则” business rules）的，在我们的例子中就是存取数据库。我们将利用 Zend Framework 提供的 Zend_Db_Table 类来进行查找、插入、修改和删除数据库表中的记录。

配置

为了使用 Zend_Db_Table 类，我们必须先告诉它使用的数据库名称以及该数据库的用户名和密码。因为不希望将这些信息直接硬编码(hard-code)到程序中，所以我们使用配置文件来保存这些信息。

Zend Framework 提供了一个 Zend_Config 类，它可以用灵活的面向对象的方式来访问配置文件。配置文件可以是一个 INI 文件也可以是一个 XML 文件。此处我们使用 INI 方式将配置信息保存在 application 目录下的 config.ini 文件中。

```
zf-tutorial/application/config.ini
```

```
[general]
db.adapter = PDO_MYSQL
db.params.host = localhost
db.params.username = rob
db.params.password = 123456
db.params.dbname = zftest
```

当然你必须使用自己的数据库名称，用户名和密码而不是我的！对于那些比较大而且包含多个配置文件的应用程序，你可以将配置文件存放在一个单独的目录如 application/config 中。

使用 Zend_Config 非常简单：

```
$config = new Zend_Config_ini ('config.ini', 'setion');
```

注意在这种情况下，Zend_Config_Ini 从 INI 中装入一节(section)的数据而不是所有节的数据（当然只要你愿意，可以装入每一节的数据）。它支持一个节名的参数，这样就可以装入附加节的数据。Zend_Config_Ini 将参数名称中的点当做层次分隔符，这样可以将一组相关的配置参数组合在一起。在我们的 config.ini 中，主机、用户名、密码和数据库名称参数均组合在 \$config->params->config 中。现在可以在启动文件 public/index.php 中装入配置文件信息了：

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

```
...
include "Zend/Loader.php";
Zend_Loader::registerAutoload();
// load configuration
$config = new Zend_Config_Ini('./application/config.ini', 'general');
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);
// setup controller
$frontController = Zend_Controller_Front::getInstance();
...

```

修改的地方见上面粗体字显示的部分。首先装载两个将要使用的类 Zend_Config_Ini 以及 Zend_Registry，然后将 application/config.ini 文件中的 general 节中的数据读取到 \$config 对象中。最后我们将 \$config 保存到 \$registry 对象中，这样在程序的其它地方也可以使用它。

注意：在本教程中，我们实际上并不需要把 \$config 保存到 \$registry 中，但这样做是一个好习惯，因为在大的应用程序中 config.ini 中保存的可能并不仅仅是数据库的连接信息。另外你必须知道的是，注册表有点类似于全局变量，如果不小心的话，会产生不该发生的对象的互相依赖的问题。

设置 Zend_Db_Table

为了使用 Zend_Db_Table 类，必须告诉它我们刚装入的数据库配置信息。为此我们必须先创建 Zend_Db 类的一个实例，然后使用静态函数 Zend_Db_Table::setDefaultAdapter() 将它注册到 Zend_Db_Table 中。同样，我们需要修改启动文件（黑体部分）：

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

```
...
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);
// setup database
$db = Zend_Db::factory($config->db);
Zend_Db_Table::setDefaultAdapter($db);
// setup controller
$frontController = Zend_Controller_Front::getInstance();
...

```

Zend_Db_Table⁷有一个静态成员函数factory(), 它利用\$config->db对象中的信息为我们实例化了一个合适的数据库适配器。s

创建表

在本教程我们使用 MySQL 作为后台数据库,下面是用来创建数据库表的语句:

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

在 MySQL 客户端如 phpMyAdmin 或标准的 MySQL 命令行中运行该语句即可生成数据库。

插入测试唱片数据

我们先插入一些记录到 albums 表中,这样可用来测试主页从数据库中读取数据的功能。我们将 Amazon.co.uk 上最畅销的两张 CD 插入到表中:

```
INSERT INTO albums (artist, title)  
VALUES  
    ('Duffy', 'Rockferry'),  
    ('Van Morrison', 'Keep It Simple');  
(很高兴看到 Van Morrison 专辑仍在热卖...)
```

模型 (Model)

Zend_Db_Table 是抽象类,因此必须用它派生一个类才能管理我们的唱片。虽然派生类叫什么名字无关紧要,但使用与数据库表相同的名字会更容易让人理解。因为我们的表名为 albums,因此我们称该派生类为 Albums。为了让 Zend_Db_Table 知道它要操作的表的名称,我们需要将其保护成员属性\$_name 设置为数据表名称。另外, Zend_Db_Table 假定表有一个字段名为 id 自动增加(Auto Increment)的主键,当然根据需要,这个字段名称可以改变。

将 Albums 类保存至 applications/models 目录下的 Albums.php 文件中:

zf-tutorial/application/models/Albums.php

```
<?php  
class Albums extends Zend_Db_Table  
{  
    protected $_name = 'albums';
```

⁷ 译注:原文如此,应该是 Zend_Db。

```
}

```

并不是很复杂，不是吗?! 那是因为我们比较幸运，需求非常简单而 `Zend_Db_Table` 正好提供了我们所需要的所有功能。如果需要添加为模型的增加一些特殊功能，可以将它放在这个类中。通常你需要提供的附加功能是这样一组“查询(find)”方法，它可以返回你欲查找的确切的数据集。你也可以为 `Zend_Db_Table` 设置关联表，让它从关联表中获取数据。

唱片列表功能

我们已经设置好了配置信息和数据库信息，现在我们开始讨论程序的核心部分，首先是显示一个唱片的列表。这在 `IndexController` 类中的 `indexAction()` 函数中实现，开始时我们将唱片的列表在一个表格中显示出来：

zf-tutorial/application/controllers/IndexController.php

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

```

`fetchAll()` 函数返回一个 `Zend_Db_Table_Rowset` 对象，在动作的显示脚本中可以使用它对返回的记录进行遍历。现在我们可以重写 `index.phtml` 文件：

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

```
<p><a href="<?php echo $this->url(array('controller'=>'index',
    'action'=>'add'));">Add new album</a></p>
<table>
<tr>
    <th>Title</th>
    <th>Artist</th>
    <th>&nbsp;</th>
</tr>
<?php foreach($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 $this->url(array('controller'=>'index',
            'action'=>'edit', 'id'=>$album->id));?>">Edit</a>
        <a href="<?php echo $this->url(array('controller'=>'index',
            'action'=>'delete', 'id'=>$album->id));?>">Delete</a>
    </td>
</tr>
</?php>

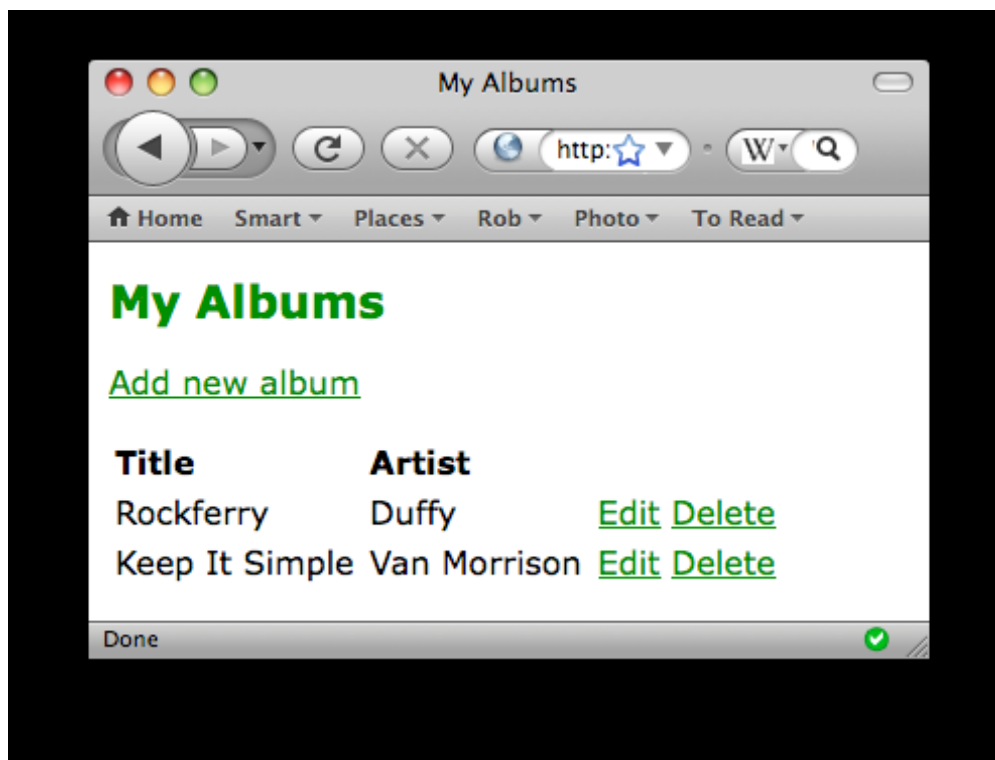
```

```
</td>
</tr>
<?php endforeach; ?>
</table>
```

我们做的第一件事是创建增加新唱片的链接。Zend Framework 提供的辅助函数 `url()` 来可为我们生成包含正确的基地址的链接。我们只须将需要的参数通过数组的形式传递给它，余下生成链接的工作 `url()` 会自动完成。

然后我们创建一个 HTML 表格来显示唱片名称，艺术家以及修改或删除记录的链接。我们使用标准的 `foreach` 在唱片列表上循环，这里使用了 `foreach` 的另一种写法即冒号(:)加 `endforeach;` 的方式，这样做比找配对的括号更容易。同样的使用 `url()` 辅助函数来生成修改和删除的链接。

现在浏览 `http://localhost/zf-tutorial/` 可以看到如下图所示的效果：



添加新唱片

现在我们可以编写添加新唱片功能的代码了。它包括了两个方面的工作：

- 显示一个表单供用户输入详细的唱片信息
- 处理提交的表单，将唱片信息保存到数据库中

我们使用 `Zend_Form` 来完成这个工作。`Zend_Form` 组件可以用来创建输入表单并对用户的输入信息进行验证。我们引入一个新的派生自 `Zend_Form` 的模型类 `AlbumForm` 来定义该表单：

```
zf-tutorial/application/models/AlbumForm.php
```

```
<?php
class AlbumForm extends Zend_Form
{
    public function __construct($options = null)
    {
        parent::__construct($options);
        $this->setName('album');

        $id = new Zend_Form_Element_Hidden('id');
        $artist = new Zend_Form_Element_Text('artist');
        $artist->setLabel('Artist')
            ->setRequired(true)
            ->addFilter('StripTags')
            ->addFilter('StringTrim')
            ->addValidator('NotEmpty');

        $title = new Zend_Form_Element_Text('title');
        $title->setLabel('Title')
            ->setRequired(true)
            ->addFilter('StripTags')
            ->addFilter('StringTrim')
            ->addValidator('NotEmpty');

        $submit = new Zend_Form_Element_Submit('submit');
        $submit->setAttrib('id', 'submitbutton');

        $this->addElements(array($id, $artist, $title, $submit));
    }
}
```

在 AlbumForm 的构造函数中，我们创建了包括四个元素的表单: id, artist, title 和 submit 按钮。每个元素都设置了不同的属性，包括要显示的文本标签。对于文本元素，我们增加了两个过滤器，StripTags 和 StringTrim，它们分别用来删除不必要的 HTML 标记和不必要的空格。我们还将它们设置为必需的字段，通过添加一个 NotEmpty 验证器来保证用户确实输入了我们需要的信息。

现在我们需要显示这个表单，并能在表单提交后进行处理。这可以通过 addAction()来实现：

zf-tutorial/application/controllers/IndexController.php

```
...
function addAction()
{
    $this->view->title = "Add New Album";
    $form = new AlbumForm();
```

```
$form->submit->setLabel('Add');
$this->view->form = $form;
if ($this->_request->isPost()) {
    $formData = $this->_request->getPost();
    if ($form->isValid($formData) {
        $albums = new Albums();
        $row = $albums->createRow();
        $row->artist = $form->getValue('artist');
        $row->title = $form->getValue('title');
        $row->save();
        $this->_redirect('/');
    } else {
        $form->populate($formData);
    }
}
}
```

我们再详细分析一下这段代码。

```
$form = new AlbumForm();
$form->submit->setLabel('Add');
$this->view->form = $form;
```

创建了一个 AlbumForm 的新实例 \$form，将表单提交按钮的标签改为“Add”，然后将 \$form 保存到视图的 form 属性，这样将来可以显示它。

```
if ($this->_request->isPost()) {
    $formData = $this->_request->getPost();
    if ($form->isValid($formData) {
```

如果请求对象(request object)的 isPost()方法返回 true,说明这个表单处于提交状态,我们就用 getPost()方法从请求对象中获取表单提交的数据,然后用 isValid()成员函数来验证表单是否有效:

```
$albums = new Albums();
$row = $albums->createRow();
$row->artist = $form->getValue('artist');
$row->title = $form->getValue('title');
$row->save();
$this->_redirect('/');
```

如果表单有效,我们创建一个新的 Albums 模型类的实例,使用 createRow()方法来生成一条新记录,再将 artist 和 title 信息填充到这条新记录中,最后再将其保存到数据库中。在保存工作完成后,用控制器的 _redirect()方法重定向到主页上。

```
} else {  
    $form->populate($formData);  
}
```

如果表单无效，则将用户输入的数据重新填回表单显示给用户。

现在需要在显示脚本 `add.phtml` 中显示表单：

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

```
<?php echo $this->form ;?>
```

显示表单非常简单，因为表单知道如何显示自己。

修改唱片信息

修改唱片信息与增加新唱片的过程非常相似，因此它们的代码也很像：

zf-tutorial/application/controllers/IndexController.php

```
...  
function editAction()  
{  
    $this->view->title = "Edit Album";  
    $form = new AlbumForm();  
    $form->submit->setLabel('Save');  
    $this->view->form = $form;  
    if ($this->_request->isPost()) {  
        $formData = $this->_request->getPost();  
        if ($form->isValid($formData)) {  
            $albums = new Albums();  
            $id = (int)$form->getValue('id');  
            $row = $albums->fetchRow('id='.$id);  
            $row->artist = $form->getValue('artist');  
            $row->title = $form->getValue('title');  
            $row->save();  
            $this->_redirect('/');  
        } else {  
            $form->populate($formData);  
        }  
    } else {  
        // album id is expected in $params['id']  
        $id = (int)$this->_request->getParam('id', 0);  
    }  
}
```

```

        if ($id > 0) {
            $albums = new Albums();
            $album = $albums->fetchRow('id='.$id);
            $form->populate($album->toArray());
        }
    }
}
...

```

先看一下修改唱片信息与增加新唱片操作的不同。修改唱片时我们首先需要从数据库中将原唱片信息获取出来，然后填充到表单元素中再显示：

```

// album id is expected in $params['id']
$id = (int)$this->_request->getParam('id', 0);
if ($id > 0) {
    $albums = new Albums();
    $album = $albums->fetchRow('id='.$id);
    $form->populate($album->toArray());
}

```

如果请求不是 POST，那么将执行这个填充过程，它通过模型从数据库中获取记录的值。我们可以使用 Zend_Db_Table_Row 类的成员函数 toArray() 来直接填充表单。

最后我们需要将数据重新写回数据库的相应记录。这可以通过先获取相应记录，再保存该记录的方式来实现。

```

$albums = new Albums();
$id = (int)$form->getValue('id');
$row = $albums->fetchRow('id='.$id);

```

视图模板与 add.phtml 相同：

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

```
<?php echo $this->form ;?>
```

现在我们的程序可以增加、修改唱片记录了。

删除唱片

为了使程序完整，还需要一个删除操作。在主页的唱片列表中每张唱片都有一个删除该唱片的链接，当点击该链接时相应的唱片记录就会被删除，但这样做是错误的。记住我们的 HTTP 规范，对于不可逆的操作，不应该使用 GET，而应使用 POST。

当用户点击删除链接时我们应该显示一个确认表单，只有在用户选择了“是”的时候我们才做这个删除操作。因为这个表单非常简单，我们直接将表单的 HTML 代码写到显示脚本中。先看代码：

zf-tutorial/application/controllers/IndexController.php

```
...
function deleteAction()
{
    $this->view->title = "Delete Album";
    if ($this->_request->isPost()) {
        $id = (int)$this->_request->getPost('id');
        $del = $this->_request->getPost('del');
        if ($del == 'Yes' && $id > 0) {
            $albums = new Albums();
            $where = 'id = ' . $id;
            $albums->delete($where);
        }
        $this->_redirect('/');
    } else {
        $id = (int)$this->_request->getParam('id');
        if ($id > 0) {
            $albums = new Albums();
            $this->view->album = $albums->fetchRow('id='.$id);
        }
    }
}
...
```

这段程序通过使用请求对象的 isPost()方法来确定是显示确认表单还是用 Album()类来做删除操作。实际的删除操作是通过调用 Zend_Db_Table 的 delete()方法来完成的。如果不是 POST 请求，就通过 id 参数将相应的记录从数据库中读取出来并保存至视图中。

显示脚本包含一个简单的表单：

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

```
<?php if ($this->album) :?>
<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>
<form action="<?php echo $this->url(array('action'=>'delete')); ?>"
method="post">
<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: ?>
```

在这段代码中，我们先显示一段确认信息给用户，其后跟随一个包括 Yes 和 No 的按钮的表单。在动作代码中做删除操作时先检查是否含有“Yes”值。

现在你已经有了一个完整的程序了，就这么简单 :-)

故障检查

如果index/index之外的动作出现问题，通常是因为路由类不能确定网站在哪个子目录下。根据我目前的观察，这通常是由于网站的url路径与web根目录的路径不同。

如果缺省的代码不能工作，就应该将\$baseUrl设置成服务器的正确路径：

zf-tutorial/public/index.php

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

此处应该将'/mysubdir/zf-tutorial/public'替换成访问index.php的正确路径。例如，访问index.php的路径是<http://localhost/~ralle/zf-tutorial/public/index.php>，那么\$baseUrl就是 '/~ralle/zf-tutorial/public'。

结束语

使用Zend Framework来创建的简单但功能全面的MVC应用程序到此就结束了。希望你觉得它有意思并能给你带来一些有用的信息。如果你发现有什么地方错了，请发邮件至 rob@akrabat.com!

本教程只讨论了Zend Framework最基本的用法，还有很多的类等待你去探索！如果想对它有更深刻的理解，一定要记得阅读它的手册(<http://framework.zend.com/manual>)或wiki (<http://framework.zend.com/wiki>)！

如果对开发Zend Framework本身有兴趣，那么<http://framework.zend.com/developer>一定值得你一看。

另外，如果你喜欢捧着书看，作者正在写《Zend Framework In Action》，现在已经可以预订。更详细的

信息请访问网站 <http://www.zendframeworkinaction.com>. Check it out ☺

译注：《Zend Framework In Action》一书已出版，目前中文版国内还没有引进。