Zend Framework 入门教程(简体中文版)

Getting Started with Zend Framework

By Rob Allen, www.akrabat.com

Document Revision 1.6.3

翻译：虫少侠，http://www.roln.cn

【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。小虫也不是ZF高手，在开始学习ZF时前辈推荐了这个教程，小虫就边学习边练习边翻译。其中疏漏和用词不当之处，请见谅。在看这个教程之前，我也找过一些其他的教程，但对新手来说都不是那么容易。这个教程算是我见过最适合初学者的教程。

翻译过程参考了Altair翻译的 1.5.2 入门教程，特此感谢。

本教程详细地介绍了使用Zend Framework来开发数据库驱动的MVC架构应用程序的过程。

注意：本教程在Zend Framework 1.8 和 1.9 两个版本中测试通过。在以后的 1.x系列版本中，它有很大可能会顺利运行，但在 1.8 以前的版本中不能运行。

英文原版地址：http://akrabat.com/zend-framework-tutorial/

如果发现翻译有问题，请发邮件至chongzi@roln.cn或到我的博客任意文章后面留言

需求

使用Zend Framework需要满足下面的条件：

- PHP 5.2.4 或以上版本
- 支持mod_rewrite或类似功能的WEB服务器

本教程的一些假设

本教程假设你在运行PHP5.2.4 或更高版本的Apache Web服务器。而且Apache已安装并正确配置了mod_rewrite扩展。

必须保证Apache支持 .htaccess文件， 这通常中通过修改httpd.conf中的

```
AllowOverride None
```

为

```
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Zend Framework

可以到这个链接获取Zend Framework http://framework.zend.com/download ，有.zip和.tar.gz两种格式，链接在该网页的最底部。

安装 Zend_Tool

Zend Framework提供了新的命令行工具。我们首先安装这个工具。

注：ZF1.9.0 中的Zend_Tool不能用Windows中运行。因为写这个教程时，ZF最新版本是 1.9.0，所以我用 1.8.4 来代替。

在Windows中安装Zend_Tool

- 在C:\Program Files\中新建文件夹ZendFrameworkCli
- 双击打开下载的ZendFramework-1.8.4PL1-minimal.zip文件
- 复制其中的bin和library文件夹到C:\Program Files\ZendFrameworkCli中
- 将bin目录添加到系统pash环境变量中
  - 打开“控制面板”的“系统”项
  - 选中“高级”选项卡，打开“环境变量”
  - 在“系统变量”中找到“Path”，双击
  - 到输入框最后面添加;C:\Program Files\ZendFrameworkCli\bin，按“确定”完成修改(不要漏写分号)
  - 重启

在OS X和Linux中安装Zend_Tool

- 双击Downloads文件夹下已下载的ZendFramework-1.9.0-minimal.zip解压
- 将解压的文件拷贝到/usr/local/ZendFrameworkCli，可以在终端输入下面的命令来执行：

  ```
  sudo cp -r ~/Downloads/ZendFramework-1.9.0-minimal /usr/local/
  ZendFrameworkCli
  ```

- 编辑bash profi le以给Zend_Tool设置别名
- 在终端输入：open ~/.bash_profile
- 在文件的末尾加入一行：alias zf=/usr/local/ZendFrameworkCli/bin/zf.sh
- 保存退出
- 退出终端

测试Zend_Tool

你可以打开Linux终端或Windows的命令提示符来测试Zend_Tool命令行接口。

提示符下输入：

```
zf show version
```

如果一切顺利，你会看到：

```
Zend Framework Version: 1.9.0
```

如果没有，请检查是否正确设置path、在ZendFrameworkCli文件夹下是否有bin文件夹

开始编写我们的程序

所有的工具都安装完毕，现在我们可以开始构建一个Zend Framework程序。我们将构建一个非常简单的库管系统，用来管理我们收藏的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信息以下面这种非常简单的表格形式保存在数据库中：

字段名	类型	允许空？	备注
id	integer	No	Primary key,auto increment
artist	varchar(100)	No	
title	varchar(100)	No	

需要下面的一些页面：

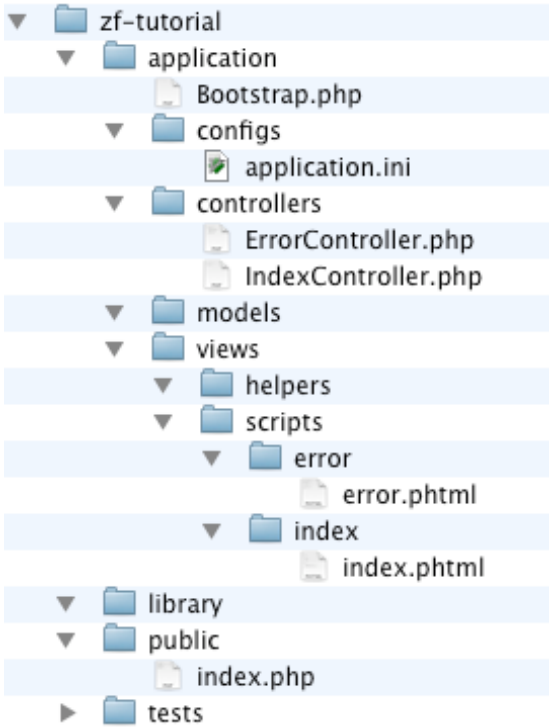
主页	显示所有唱片列表，提供修改、删除的链接。另外还提供一个新增唱片的链接。
添加新唱片页面	提供了一个表单，用于添加新唱片
修改唱片页面	提供了一个表单来修改唱片
删除唱片页面	提示是否删除的提示和删除功能

创建项目

打开终端或命令提示符，使用cd命令切换到WEB服务器的根目录下，确保你有权限在该目录下创建文件，并且WEB服务器有该目录的读取权限。然后输入：

```
zf create project zf-tutorial
```

ZF将创建一个名为zf-tutorial的文件夹，文件夹中已经建立好推荐的目录结构。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。



(在public文件夹下还有个隐藏的.htaccess文件).

application/目录是存放源代码的地方。正如上面看到的，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目录是网站对外公开的根目录，这就表示我们可以通过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。

注释：

在一个有多个网站的服务器上，你最好为你的网站创建一个虚拟主机，将其根目录直接指向public/目录。比如，你可以这样创建一个名为zf-tutorial.localhost的虚拟主机：

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

使用这个网址访问这个网站：http://zf-tutorial.localhost/（请确保你修改了 /etc/hosts 或 c\windows\system32\drivers\etc\hosts 使zf-tutorial.localhost指向 127.0.0.1）。因为这很简单，所以我们就不在这个教程中讨论这些了。

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。

从下载的压缩文件ZendFramework-1.9.0.zip中把library/Zend/目录拷贝到 zf-tutorial/library/下，这样zf-tutorial/library/ 目录下就包含一个名为Zend的子目录。

打开这个链接来测试是否所有的都配置好了： http://localhost/zftutorial/public，应该能看到下面的页面：

引导文件介绍

Zend Framework的控制器使用前端控制器（Front Controller）设计模式，所有的请求都通过index.php文件进入。这就是我们所说的引导文件，它能确保程序的运行环境都正确地配置。我们可以使用放在zf-tutorial/public文件夹下的.htaccess文件将所有的请求转向public/index.php文件，public/文件夹和.htaccess文件都已由Zend_Tool自动创建。

index.php作为文件程序的入口，用来创建Zend_Application的实例，以初始化应用程序并运行程序。index.php同时声明两个常量：APPLICATION_PATH（application/文件夹的的路径） 和APPLICATION_ENV（程序的运行环境）。The generated .htaccess file sets it to development。

Zend_Application组件用来启动应用程序，它使用配置文件(application/configs/application.ini)中的指令进行配置。 这个文件也已经自动生成。application/Boostrap.php 中有一个扩展自Zend_Application_Bootstrap_Bootstrap的类：Bootstrap，它可以用来处理任何需要的引导代码。

编辑application.ini文件

虽然Zend_Tool已经提供了一个不错的默认配置文件，但仍然需要添加应用程序的特定配置。打开application/configs/application.ini，在[production]部分的最后添加下面的代码：

```
phpSettings.date.timezone = "UTC"
```

显然，你需要改成自己的时区。

扩展自动加载类

我们需要创建一个自动加载器来自动从程序目录下加载资源，比如从models目录和forms目录。Zend_Application_Module_Autoloader类用来做这些工作，我们可以在Bootstrap类中使用它。要添加的部分已经加粗：

application/Bootstrap.php

```
<?php
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
    protected function _initAutoload()
    {
        $moduleLoader = new Zend_Application_Module_Autoloader(array(
        'namespace' => '',
        'basePath' => APPLICATION_PATH));
        return $moduleLoader;
    }
}
```

以_init开头的方法会被Zend_Application自动调用。方法名称的其余部分你可以自定义。组件自动加载器将按照下面表格中内容自动加载application/下特定文件夹中特定前缀名称的类，

文件夹	前缀	举例
api	Api_	Api_Rest
forms	Form_	Form_Login
models	Model_	Model_News
models/DbTable	Model_DbTable_	Model_DbTable_News
plugins	Plugin_	Plugin_

现在开始我们就可以向应用程序中加入具体代码。

程序具体代码

在创建文件之前，先理解Zend Framework要求页面如何组织是非常重要的。程序的每个页面就是一个动作（Action），而动作又组合到控制器（controller）中。

对于http://localhost/public/zf-tutorial/news/view这种形式的URL，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，但我们要做的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。比如http://localhost/public/zf-tutorial/news将执行news控制器的index动作。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。因此，http://localhost/public/zf-tutorial/将执行index控制器的index动作。

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，它的四个动作如下：

页面	控制器	动作
主页	Index	index
添加新唱片	Index	add
编辑唱片	Index	edit
删除唱片	Index	delete

当站点更复杂时就需要添加更多的控制器；如果需要，甚至可以将控制器组合成模块（module）。

编写控制器

现在可以编写控制器类了。在Zend Framework中，控制器类名必须命名为{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。每个动作是控制器中一个的public函数，且必须以这种形式命名：{action name}Action。{action name}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。

我们的控制器名为IndexController,它放IndexController.php文件中，现在已经由Zend_Tool自动创建好了。我们只需要增加需要的动作即可。

我们用Zend_Tool的zf命令来增加控制器动作。打开终端或命令行，将当前目录切换到zf-tutorial/。然后输入下面三个命令：

```
zf create action add index
zf create action edit index
zf create action delete index
```

这三个命令在IndexController类中创建名为xxxAction的函数，同时创建相应的视图文件（后面会用到）。我们现在已经建立了想要的四个动作。

每个动作的URL访问地址是：

地址	动作名
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()
http://localhost/zf-tutorial/public/index/delete	IndexController::deleteAction()

可以测试一下上面的地址，会看到类似下面的信息：

View script for controller **index** and script/action name **add**

下面我们开始弄数据库部分。

数据库

我们已经构建好了由控制器动作函数和视图构成的程序骨架，现在应该考虑程序的模型（model）部分了。模型是用来处理程序的核心议题（即所谓的“业务规则”）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。

数据库配置

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_Application组件带有一个数据库配置资源（resource），所以我们需要做的只是在配置文件config/application.ini中正确配置信息，其它的都由这个组件完成。

打开configs/application.ini，将下面的内容添加到[production]的末尾（[staging]上面）：

```
resources.db.adapter = PDO_MYSQL
resources.db.params.host = localhost
resources.db.params.username = rob
resources.db.params.password = 123456
resources.db.params.dbname = zf-tutorial
```

很明显，你必须使用自己的数据库名称，用户名和密码，而不是我的！现在数据库连接和Zend_Db_Table类的默认配置都已设置完毕。

创建数据库表格

我准备使用MYSQL数据库，创建表格的SQL语句是：

```
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（phpMyAdmin或MySQL命令行）运行上面的语句。

添加测试唱片数据

我们先插入一些记录到albums表中，用来测试主页从数据库中读取数据的功能。我们要在表格中插入一些记录，我们将Amazon UK上最畅销的几张CD插入到表中。在Mysql客户端运行下面的语句：

```
INSERT INTO albums (artist, title)
VALUES
('Bob Dylan', 'Together Through Life'),
('Various Artists', 'Now That\'s what I Call Music! 72'), ('Lady Gaga', 'The Fame'),
('Lily Allen', 'It\'s Not Me, It\'s You'), ('Kings of Leon', 'Only By The Night');
```

现在数据库中已经有了一些数据，我们可以为它写一个非常简单的模型。

模型

Zend Framework提供的Zend_Db_Table类实现了表数据入口（Table Data Gateway）设计模式，以方便与数据库表格建立数据接口。对于较复杂的项目，经常需要建一个模型类，这个类以保护（protected）成员变量的方式使用一个或多个Zend_Db_Table实例。但在这个教程中，我们将创建一个扩展自Zend_Db_Table的模型。

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都可以，但使用数据表名来命名更容易让人理解。因为我们表名为albums，加上前缀后类名将是Model_DbTable_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。

我们要把类Model_DbTable_Albums放到applications/models/DbTable/Albums.php中。创建文件Albums.php并键入下面的内容：

zf-tutorial/application/models/DbTable/Albums.php

```
<?php
class Model_DbTable_Albums extends Zend_Db_Table_Abstract
{
    protected $_name = 'albums';
    public function getAlbum($id)
    {
        $id = (int)$id;
        $row = $this->fetchRow('id = ' . $id);
        if (!$row) {
            throw new Exception("Count not find row $id");
        }
        return $row->toArray();
    }
    public function addAlbum($artist, $title)
    {
        $data = array(
           'artist' => $artist,
           'title' => $title,
        );
        $this->insert($data);
    }
    public function updateAlbum($id, $artist, $title)
```

```
    {
        $data = array(
            'artist' => $artist,
            'title' => $title,
        );
        $this->update($data, 'id = '. (int)$id);
    }
    public function deleteAlbum($id)
    {
        $this->delete('id =' . (int)$id);
    }
}
```

我们创建了四个辅助方法，用来与数据表接口。getAlbum()查询一条记录存放到数组中，addAlbum()向数据库中增加一条记录，updateAlbum()修改一条唱片记录，deleteAlbum()完全删除一条记录。每个方法的代码都很容易理解。你也可以为Zend_Db_Table设置关联表获取关联数据，虽然这个教程中不需要这么做。

我们需要使用从模型中获取的数据填充到控制器中，并获取视图脚本来展示这些数据，但在此之前，我们需要理解Zend Framework的视图系统是如何工作的。

布局和视图（Layouts and Views）

毫不奇怪，Zend Framework的视图组件叫做Zend_View。Zend_View让我们可以将页面和动作函数分离。Zend_View的基本使用方法如下：

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

显然，如果我们直接将这些代码写到每一个动作函数中，就不得不到处无聊地重复这些跟动作关系不大的“结构化”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为我们提供一个叫做ViewRenderer的视图助手(Action Helper)。它负责为我们初始化控制器中的view属性（$this->view），并在动作调度后显示视图脚本。

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。

在用zf命令创建项目和添加控制器与动作的时候，这些都已经由Zend_Tool为我们创建好。

公共HTML代码：Layouts

很快你就会发现在我们的视图中有大量相同的HTML代码，至少有用于网页头部或底部的相同代码，还有可能有一两个相同的侧边栏。由于这个问题非常普遍，因此Zend Framework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。

这些布局文件默认保存在application/layouts/，有一个资源可以被Zend_Application用来配置Zend_Layout。

首先创建application/layouts/文件夹，然后在配置文件configs/applications.ini中的[production]部分最后添加一行：

```
resources.layout.layoutpath = APPLICATION_PATH "/layouts"
```

我们还需要在Bootstrap类中为视图设置全局变量。我们再一次使用_init方法，命名为_initViewHelpers()。编辑application/Bootstrap.php文件，将下面的代码添加到_initAutoload()方法下面：

application/Bootstrap.php

```
...
protected function _initViewHelpers()
{
    $this->bootstrap('layout');
    $layout = $this->getResource('layout');
    $view = $layout->getView();
    $view->doctype('XHTML1_STRICT');
    $view->headMeta()->appendHttpEquiv('Content-Type', 'text/html;charset=utf-8');
    $view->headTitle()->setSeparator(' - ');
    $view->headTitle('Zend Framework Tutorial');
}
...
```

我们使用bootstrap()成员变量确保初始化布局资源（layout resource），然后使用getResource()方法得到Zend_Layout对象，最后利用getView()方法得到视图。

一旦我们有了$view实例，我们就可以用一些助手函数来准备后面的渲染。doctype()视图助手用来设置我们想要的DOCTYPE。它对保证其他的视图助手能生成准备的HTML代码来说是非常有用的。我们使用headMeta()来设置content-type META标签，headTitle()视图助手用来设置title中的分隔符和title最后一部分。

调试的时候，Zend_Layout将从application/layouts目录下寻找名称为layout.phtml的布局视图脚本，所以我们最好写好这个文件，下面是这个文件的内容：

zf-tutorial/application/layouts/layout.phtml

```
<?php echo $this->doctype(); ?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<?php echo $this->headMeta(); ?>
<?php echo $this->headTitle(); ?>
</head>
<body>
<div id="content">
<h1><?php echo $this->escape($this->title); ?></h1>
<?php echo $this->layout()->content; ?>
</div>
</body>
</html>
```

布局文件包含非常标准的外层HTML代码。因为这个文件是个普通的PHP文件，我们可以在里面使用PHP。里面有个变量$this，它是视图对象的一个实例，在程序引导时被创建。我们可以用它来获取已经赋给视图的数据，也可以调用方法。这些方法（即视图助手view helpers）返回数据可以直接用echo输出。

首先输出Bootstrap::_initViewHelpers()中创建的脚本助手，它将为<head>部分创建准备的代码。在<body>中，我们创建一个div层，div中有一个包含标题的<h1>。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。

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。

在zend Framework 1.9 中，引入视图助手baseUrl()。它可以从请求对象中收集我们需要的信息，提供我们不知道的URL内容。

在zend Framework 1.8 中，还没有提供baseUrl()视图助手，所以我们要自己创建。视图助手放在application/views/helpers子目录下，并被命名为{HelpName}.php（首字母必须大写），其中的类有一个预料中的命名规则：必须被命名为Zend_View_Helper_{HelperName}（首字母必须大写）。类中必须有一个名为{helperName}()（首字母小写，勿记！）的函数。在我们的例子中，这个文件命名为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();
    }
}
```

这个函数并不复杂。我们简单地得到一个前端控制器的实例，然后返回getBaseUrl()成员函数的值。

注意在ZF1.8 和 1.9 中，如果你想创建自己的视图助手，创建过程都跟上面我们建立baseUrl()一样。

下面，我们需要添加CSS文件到application/layouts/layout.phtml的<head>部分，我们再使用一个视图助手-headLink()：

zf-tutorial/application/layouts/layout.phtml

```
...
<head>
<?php echo $this->HeadMeta(); ?>
<?php echo $this->headTitle(); ?>
<?php echo $this->headLink()->prependStylesheet($this->baseUrl().'/css/site.css'); ?>
</head>
...
```

利用headLink()，我们允许有其它特殊作用的CSS文件添加在控制器视图脚本中，它们将显示在<head>部分的site.css之后。

最后，我们需要添加CSS样式，所以在public中创建一个css目录：

zf-tutorial/public/css/site.css

```
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;
}
```

这会使它看起来稍微好看一些，但像你可能会说的那样，我不是个设计师（所以别指望太好看）！

现在我们可以清理前面为了填充内容而自动创建的四个动作脚本，清空index.phtml、add.phtml、add.phtml、edit.html和delete.phtml（提醒一下，它们在application/view/scripts/index目录下）。

唱片列表功能

既然我们已经配置好配置文件、数据库和视图骨架，我们可以开始进行程序的中心内容，首先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";
    $this->view->headTitle($this->view->title, 'PREPEND');
    $albums = new Model_DbTable_Albums();
    $this->view->albums = $albums->fetchAll();
}
...
```

首先我们为页面设置标题，然后把这引标题添加到head title的前面，显示在浏览器标题栏。

fetchAll函数返回一个Zend_Db_Table_Rowset对象，它允许我们在动作的视图脚本文件中遍历返回的记录。现在我们可以填充相关的视图脚本index.phtml：

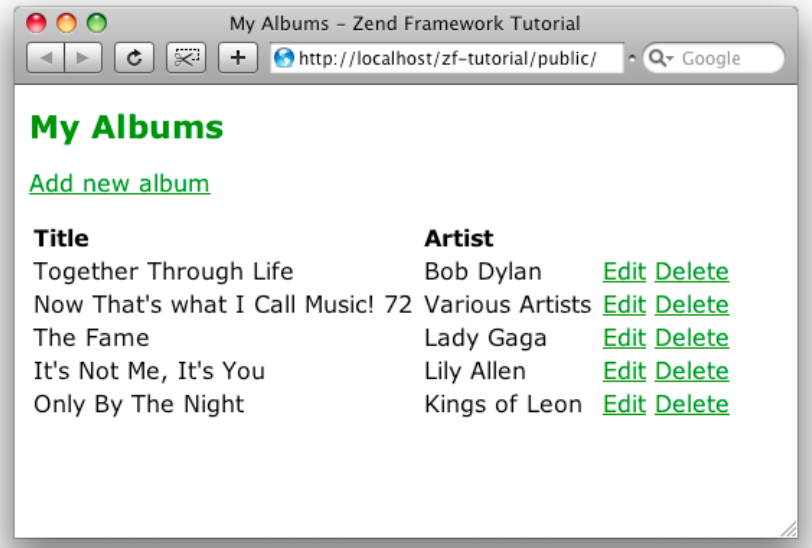
zf-tutorial/application/views/scripts/index/index.phtml

```
<p><a href="<?php echo $this->url(array('controller'=>'index','action'=>'add'));?>">Add new album</a></p>
<table>
    <tr>
        <th>Title</th>
        <th>Artist</th>
        <th> </th>
    </tr>
    <?php foreach($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 endforeach; ?>
</table>
```

首先我们创建一下链向“添加新唱片”的超链接。ZF提供的url()视图助手非常有助于创建包含正确baseurl的超链接。我们只需要将需要的参数以数组的形式传递给它，剩下的工作它都自动完成。

下面我们创建一个html表格来显示每个唱片的标题和作者，并提供修改和删除唱片的链接。我们用一个标准的foreach:循环来显示唱片列表。这里我们使用替代语法：用一个冒号和endforeach;，因为这样比使用配对大括号更容易查看。同样的，我们可以使用url()视图助手来创建编辑和删除的链接。

打开http://localhost/zf-tutorial，应该显示一个不错的唱片列表：



添加新唱片

我们可以开始编写添加新唱片的功能了。这部分包括两点：

- 提供一个表单让用户填写资料
- 处理提交的表单并添加到数据库中

我们使用Zend_From来做这个工作。Zend_Form组件可以用来创建表单和验证表单。我们创建一个扩展自Zend_Form的新类Form_Album来定义我们的表单。因为我们使用模块自动加载，这个类应该存储在forms目录下的Album.php文件中：

zf-tutorial/application/forms/Album.php

```
<?php
class Form_Album 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form_Album的构造函数中，我们创建四个表单元素，分别用作id、artist、title和submit按钮。我们给每个元素设置不同的属性，包括要显示的label。对text元素，我们添加两个过滤器-StripTags和StringTrim，来移除不想要的HTML代码和空格。我们也可以设置他们是必填项，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，然后在提交后进行处理。这结都在IndexController的addAction()中做到：

zf-tutorial/application/controllers/IndexController.php

```
...
function addAction()
{
    $this->view->title = "Add new album";
    $this->view->headTitle($this->view->title, 'PREPEND');
    $form = new Form_Album();
    $form->submit->setLabel('Add');
    $this->view->form = $form;
    if ($this->getRequest()->isPost()) {
        $formData = $this->getRequest()->getPost();
        if ($form->isValid($formData)) {
            $artist = $form->getValue('artist');
            $title = $form->getValue('title');
            $albums = new Model_DbTable_Albums();
            $albums->addAlbum($artist, $title);
            $this->_redirect('/');
        } else {
            $form->populate($formData);
        }
    }
}
...
```

下面详解部分细节：

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

我们实例化Form_Album，将提交按钮的lable设置为“Add”，然后定义到视图中以备显示。

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

如果请求对象的isPost()方法返回真(true)则表示表单已经提交，然后使用getPost()获取表单数据，使用isValid成员函数来判断是否有效。

```
$artist = $form->getValue('artist');
$title = $form->getValue('title');
$albums = new Model_DbTable_Albums();
$albums->addAlbum($artist, $title);
```

如果表单有效，则实例化Model_DbTable_Albums模型类，然后用我们之前创建addAlbum()方法向数据库中添加一条新记录。

```
$this->_redirect('/');
```

在我们保存这条新唱片记录后，利用控制器的_redirect()方法使页面重定向到首页。

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

如果表单数据无效，则将用户刚才填写的数据填充表单，并再次显示表单。现在我们需要在add.phtml视图脚本中显示表单：

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

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

如你所见，显示一个表单非常简单，就好像表单自己知道如何去显示一样。

编辑唱片

编辑唱片跟添加唱片差不多是一样的，所以代码也差不多：

zf-tutorial/application/controllers/IndexController.php

```
...
function editAction()
{
     $this->view->title = "Edit album";
     $this->view->headTitle($this->view->title, 'PREPEND');
     $form = new Form_Album();
     $form->submit->setLabel('Save');
     $this->view->form = $form;
     if ($this->getRequest()->isPost()) {
          $formData = $this->getRequest()->getPost();
          if ($form->isValid($formData)) {
               $id = (int)$form->getValue('id');
               $artist = $form->getValue('artist');
```

```
                $title = $form->getValue('title');
                $albums = new Model_DbTable_Albums();
                $albums->updateAlbum($id, $artist, $title);
                $this->_redirect('/');
            } else {
                $form->populate($formData);
            }
        } else {
            $id = $this->_getParam('id', 0);
            if ($id > 0) {
                $albums = new Model_DbTable_Albums();
                $form->populate($albums->getAlbum($id));
            }
        }
}
...
```

让我们看一下与添加唱片的不同。首先，在为用户显示表单的时候，我们需要读取数据库的唱片的作者和标题，填充在表单中。这是这个方法的最后一部分：

```
$id = $this->_getParam('id', 0);
if ($id > 0) {
    $albums = new Model_DbTable_Albums();
    $form->populate($albums->getAlbum($id));
}
```

当页面请求不是POST的时候，这部分会被执行。我们使用_getParam()方法从请求中获得id。然后用模型查询数据库，并将结果直接显示在表单中。

在验证完表单后，我们需要把数据更新到当前操作的数据库记录中。这个工作由模型中的updateAlbum()方法完成：

```
$id = (int)$form->getValue('id');
$artist = $form->getValue('artist');
$title = $form->getValue('title');
$albums = new Model_DbTable_Albums();
$albums->updateAlbum($id, $artist, $title);
```

视图模板跟add.phtml一样：

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

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

现在你就可以添加和编辑唱片信息了。

删除唱片

为了完善程序，我们需要增加删除功能。需要在唱片列表页的每个唱片后面加一个删除链接，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，如果在用户选择了“是”，我们就进行删除操作。因为这个表单非常简单，我们直接将表单的HTML代码写到视图中。

先写IndexController::deleteAction()中的动作代码：

zf-tutorial/application/controllers/IndexController.php

```
...
public function deleteAction()
{
    $this->view->title = "Delete album";
    $this->view->headTitle($this->view->title, 'PREPEND');
    if ($this->getRequest()->isPost()) {
        $del = $this->getRequest()->getPost('del');
        if ($del == 'Yes') {
            $id = $this->getRequest()->getPost('id');
            $albums = new Model_DbTable_Albums();
            $albums->deleteAlbum($id);
        }
        $this->_redirect('/');
    } else {
        $id = $this->_getParam('id', 0);
        $albums = new Model_DbTable_Albums();
        $this->view->album = $albums->getAlbum($id);
    }
}
...
```

和添加和编辑时一样，我们通过使用请求对象的isPost()方法来确定是显示确认表单还是进行删除操作。实际的删除操作是通过调用Model_DbTable_Albums()的deleteAlbum()方法来删除记录的。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。

视图脚本是个简单的表单：

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

```
<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>
```

在这个脚本中，我们先给用户显示一条警告信息，然后是一个包含“Yes”和“No”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。

现在你有了一个完整可用的程序了。

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有用。如果你发现错误，请发送邮件到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！你也应该去读读帮助手册(http://framework.zend.com/manual)！如果你有意于开发这个框架，就把development wiki(http://framework.zend.com/developer)也读完...

最后，如果你喜欢看印刷版，我已经写了一本书《*Zend Framework in Action*》，目前正在销售中。

更多信息请访问 http://www.zendframeworkinaction.com。去看下吧~~