

Введение в Zend Framework

Роб Алэн (<http://akrobat.com>)

Ревизия 1.4.2

Copyright © 2006, 2007

Перевод: Александр Мусеев (<http://paradigm.ru>)

Краткая аннотация: Данный материал рассчитан на то, чтобы дать общее представление об использовании Zend Framework для создания простейших приложений с использованием баз данных. Приведенные примеры были протестированы на Zend Framework версии 1.0.0. Скорее всего, они будут работать и с более поздними версиями, но не с более ранними.

Архитектура Модель-Вид-Контроллер

PHP-приложение, написанное традиционным способом, может представлять собой некое подобие следующего примера:

```
<?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"; ?>
```

В ходе жизненного цикла приложений подобного типа, трудоемкость их поддержки становится крайне высока, т.к. меняющиеся требования заказчика приводят к необходимости вносить большое количество изменений («заплаток») в исходный код. Это делает его плохо структурированным и трудно читаемым. Один из методов для повышения гибкости приложения состоит в разделении кода на три категории:

- **Модель.** Моделью называют ту часть приложения, которая относится к работе с данными. В приведенном выше фрагменте кода, это реализация новостной ленты. Подобные модели широко применимы в управляющей логике приложений, имеющих в своей основе базы данных.
- **Вид.** Под термином «вид» подразумевается пользовательский интерфейс приложения.

- **Контроллер.** Контроллеры реализуют задачи, связанные со взаимодействием между моделью и видом.

Zend Framework основан на программной архитектуре Модель-Вид-Контроллер (Model-View-Controller). Ее суть состоит в разделении приложения на перечисленные выше три отдельные компоненты таким образом, что модификация каждого из них оказывает минимальное воздействие на остальные. Это приводит к существенному облегчению процесса разработки и поддержки.

Системные требования

У Zend Framework следующие требования:

- PHP версии 5.1.4 или выше;
- Веб-сервер, поддерживающий `mod_rewrite`.

В данной статье используется сервер Apache.

Где скачать Zend Framework

Zend Framework доступен для свободного скачивания в виде ZIP или TAR.GZ архивов по адресу <http://framework.zend.com/download/stable>.

Структура директорий

На данный момент в Zend Framework жестко не стандартизирована структура директорий приложения, но в официальной документации рекомендуется использовать общепринятую схему. Эта схема основана на том, что пользователь, как предполагается, имеет полный доступ к конфигурированию сервера Apache. Мы же будем использовать немного видоизмененный подход, чтобы смягчить данные требования и сделать Zend Framework более применимым в условиях широко распространенного разделяемого хостинга.

Для начала, создайте директорию `zf-tutorial` в корневом каталоге вашего тестового веб-сайта. Это будет означать, что URL этой директории примет вид <http://localhost/zf-tutorial> (адрес может варьироваться, в зависимости от настроек вашего сервера).

После этого дополнительно создайте следующую структуру каталогов для хранения файлов веб-приложения:

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

Как можно понять из названий, мы выделили специальные директории для файлов моделей, видов и контроллеров приложения. Графика, скрипты и CSS-файлы будут храниться в отдельных подкаталогах, расположенных внутри открытой для публичного доступа директории `public`.

Для начала, разархивируйте скачанный файл `ZendFramework-1.0.0.zip` (или `.tar.gz`) во временную директорию. Все файлы внутри архива находятся в директории `ZendFramework-1.0.0`. Скопируйте содержимое `library/Zend` в `zf-tutorial/library`. Теперь ваша директория `zf-tutorial/library` должна содержать подкаталог `Zend`.

Начальная загрузка

Контроллер `Zend_Controller` из библиотеки `Zend Framework` спроектирован для поддержки сайтов с хорошо читаемыми («чистыми») URL. Для достижения этой цели, все запросы к серверу перенаправляются для обработки на специальный файл `index.php`, именуемый так же файлом начальной загрузки (“bootstrapper”). Такой подход обеспечивает централизованную организацию инфраструктуры приложения и гарантирует корректную для его функционирования настройку программного окружения. Это достигается благодаря использованию конфигурационного файла `.htaccess`, который находится в директории `zf-tutorial`.

zf-tutorial/.htaccess:

```
RewriteEngine on
RewriteRule .* index.php
php_flag magic_quotes_gpc off
php_flag register_globals off
```

Директивой `RewriteRule` задано очень простое правило перенаправления URL, которое может быть переведено на человеческий язык, как «использовать `index.php` вместо любого URL». Для обеспечения безопасности так же откорректированы некоторые настройки интерпретатора PHP. В принципе, они уже должны быть заданы соответствующим образом в конфигурационном файле `php.ini` любого грамотно настроенного сервера, но в нашем примере они дублируются локально для большей надежности. Обратите внимание на то, что директива `php_flag` внутри файла `.htaccess` может применяться только при использовании `mod_php`. Если вы используете PHP через CGI или FastCGI, необходимо удостовериться в правильной настройке `php.ini`.

Вопреки сказанному выше, запросы, относящиеся к графическим файлам, JavaScript и CSS, не должны быть перенаправлены на `index.php`. Учитывая, что все они хранятся в отдельной директории `public`, мы с легкостью можем настроить Apache, чтобы сервер отдавал их напрямую. Для этого необходимо создать еще один файл `.htaccess` в директории `public` и задать в нем соответствующую директиву.

zf-tutorial/public/.htaccess:

```
RewriteEngine off
```

Для повышения безопасности, можно создать дополнительные файлы `.htaccess` в директориях `zf-tutorial/application` и `zf-tutorial/library`, с одинаковым содержимым:

zf-tutorial/application/.htaccess, zf-tutorial/library/.htaccess:

```
Deny from all
```

Эта директива закрывает доступ из web к содержимому указанных директорий. (Прим. переводчика: последние два файла можно не создавать вообще, если все, что находится вне директории `public`, заведомо закрыто глобальными настройками сервера).

Обратите внимание на то, что для использования файлов `.htaccess`, в настройках сервера Apache (`httpd.conf`) должна быть задана директива `AllowOverride` со значением `All`. Приведенная в данной статье идея использования файлов `.htaccess` принадлежит Джейсону Майнард (Jayson Minard) и опубликована в статье «Схема построения PHP-приложений: начальная загрузка (Часть 2)» (<http://devzone.zend.com/node/view/id/119>). Начинаям разработчика стоит ознакомиться с обеими ее частями.

Файл начальной загрузки `index.php`

Файл начальной загрузки `zf-tutorial/index.php` содержит следующий код:

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();
```

Заметьте, что в конце файла отсутствует закрытие блока PHP-кода `?>`, т. к. в нем нет необходимости. Кроме того, наличие закрывающего тега может привести к появлению сложных для выявления ошибок при редиректе функцией `header()`, если в конце файла по недосмотру останутся лишние пробелы.

Рассмотрим код подробнее.

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

Данные строки обеспечивают вывод интерпретатором всех сообщений о происходящих ошибках (подразумевается, что в конфигурационном файле `php.ini`

параметру `display_errors` задано значение `on`). Вторая строка кода задает часовой пояс, согласно стандартному требованию PHP 5.1+. Приведенное в примере значение параметра следует заменить на соответствующее вашему географическому положению.

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

Zend Framework рассчитан на то, чтобы его файлы находились в `include_path`, поэтому мы добавляем к значению этого параметра путь к нужной директории. Помимо него, в `include_path` добавляется директория моделей, чтобы облегчить подключение файлов и из нее.

Файл `Zend/Loader.php` содержит класс `Zend_Loader` со статическими функциями, позволяющими подключать любой класс из Zend Framework.

```
Zend_Loader::loadClass('');
```

Метод `Zend_Loader::loadClass()` подключает класс с заданным именем. Для этого выполняется преобразование указанного имени в путь к соответствующему файлу: символы «`_`» заменяются на «`/`», а в конце получившейся строки добавляется расширение `.php`. Таким образом, `Zend_Controller_Front` преобразуется в `Zend/Controller/Front.php`. Если следовать аналогичной схеме построения имен для собственных библиотечных классов, то метод `Zend_Loader::loadClass()` можно будет использовать и с ними.

Класс, который понадобится нам прежде всего, — первичный контроллер. При его работе, в свою очередь, используется специальный класс-роутер, чья задача состоит в нахождении по заданному URL функции отображения соответствующей веб-страницы. Для этого роутеру должен быть определен базовый URL файла `index.php`, относительно которого он сможет выделять URI отдельных страниц.

Базовый URL может автоматически определяться с помощью объекта `Request`, а для того, чтобы применить его в нашем примере следует воспользоваться методом `$frontController->setBaseUrl()`.

Далее нам понадобится сообщить первичному контроллеру о местоположении директории с другими контроллерами:

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

Учитывая, что наша программа предназначена для образовательных целей и функционирует на тестовой системе, можно разрешить все исключения, которые могут происходить при ее работе. По-умолчанию, первичный контроллер будет перехватывать их все до одного и хранить в свойстве `_exceptions` своего объекта (данному объекту соответствует англоязычный термин “Response object”). Этот объект используется для хранения всей информации, возвращаемой при обращении к заданному URL, что включает HTTP-заголовок, содержимое веб-страницы и массив произошедших исключений.

Первичный контроллер автоматически выдает заголовок и контент страницы непосредственно перед окончанием своей работы. Тех, кто ранее не встречался с архитектурой веб-приложений, подобной Zend Framework, может привести в некоторое недоумение тот факт, что для отображения сообщений об исключительных ситуациях, их (исключения) необходимо повторно возбуждать. Эта особенность становится актуальной для серверов, работающих в своем штатном режиме, когда вывод сообщений об ошибках на страницах неуместен.

В итоге, после всех выполненных приготовлений, наше приложение может быть запущено:

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

Если вы сейчас попытаетесь открыть страницу <http://localhost/zf-tutorial/>, то увидите сообщение о фатальной ошибке, гласящее примерно следующее:

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

Это говорит о том, что мы еще не до конца сформировали наше веб-приложение. И прежде чем приступить к делу, стоит уяснить, в чем конкретно состоит задача.

Веб-сайт

Предположим, что мы хотим создать простую базу данных для хранения информации о компакт-дисках. На главной странице будет отображен список дисков в нашей коллекции, а так же будет предоставлена возможность добавлять новые CD, редактировать и удалять ненужные записи. Для хранения данных, используем базу со схемой, приведенной ниже:

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

Необходимые страницы

Для реализации перечисленных выше функций, нам понадобятся следующие страницы:

Начальная («домашняя») страница	Список альбомов со ссылками для их редактирования и удаления. На странице так же будет присутствовать ссылка для добавления новых дисков.
Добавить альбом	Страница с формой добавления нового диска в базу.
Редактировать альбом	Страница с формой для редактирования записи.
Удалить альбом	Страница для запроса пользовательского подтверждения перед удалением диска.

Организация страниц

Прежде чем приступить к созданию файлов, следует понять, как принято организовывать их в Zend Framework. Согласно принятой терминологии, каждая страница веб-приложения определяется термином «действие¹». Действия, в свою очередь, объединяются в специальные группы — «контроллеры». Например, для URL `http://localhost/zf-tutorial/news/view` контроллером будет `news`, а действием — `view`. Контроллеры предназначены для объединения родственных действий. Для упомянутого `news`, могут быть определены действия `current`, `archived` и `view` (соответственно для отображения последней новости, архива записей и новостной ленты).

В Zend Framework так же используются модули для группировки контроллеров, но наше приложение не настолько велико, чтобы имело смысл их применять.

В контроллерах Zend Framework существует понятие действия по-умолчанию, для которого зарезервировано имя `index`. Такие действия исполняются при обращении к URL вида `http://localhost/zf-tutorial/news/` (в данном примере выполняется действие `index` контроллера `news`). Помимо исполняемых по-умолчанию действий, так же предусмотрены выбираемые по-умолчанию контроллеры с аналогичным именем — `index`. Это означает, что URL <http://localhost/zf-tutorial/> будет соответствовать действию `index` контроллера `index`.

Учитывая, что данная статья ориентирована на первое знакомство с Zend Framework, мы не будем пока касаться таких нюансов работы веб-приложения, как авторизация пользователей. Ограничимся реализацией основной функциональности, а именно — набором перечисленных выше страниц. Учитывая, что все они относятся к работе с альбомами, можно отнести их к одному контроллеру. Мы используем стандартный контроллер, а его действия будут выглядеть следующим образом:

Страница	Контроллер	Действие
Начальная страница	Index	Index
Добавить альбом	Index	Add
Редактировать альбом	Index	Edit
Удалить альбом	Index	Delete

Настройка контроллера

Приступим к созданию нашего контроллера. В Zend Framework контроллер представляет собой класс с именем типа `{Имя_контроллера}Controller`. Класс должен быть описан внутри файла с именем `{Имя_контролле-`

¹ *Примечание переводчика.* Небольшое пояснение терминологии. Англоязычным оригиналом используемого в данной статье термина «действие» является слово “action”. Приведенный перевод на данный момент еще не является устоявшимся понятием, и в некоторых текстах о модели MVC вместо него попадают иные определения. Например, так же известна прямая (и режущая слух) транслитерация — «экшен».

pa}Controller.php, в общей директории контроллеров. Стоит обратить внимание на важный нюанс: первая буква в имени класса-контроллера и имени его файла должна быть заглавной, а все остальные — строчными. Каждое действие должно быть открытой функцией класса-контроллера с именем типа {имя_действия}Action. В данном случае, имена действий должны начинаться со строчной буквы.

Таким образом, наш класс-контроллер будет носить имя IndexController и находиться в файле zf-tutorial/application/controllers/IndexController.php.

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>";
    }
}
```

В данном макете контроллера каждое действие будет выводить свое имя. Это можно протестировать, задавая приведенные ниже URL:

URL	Displayed text
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()

Теперь у нас имеется работающий класс-роутер с набором действий, соответствующих каждой странице веб-приложения. При возникновении каких-либо проблем в процессе его создания, обратитесь к разделу «Устранение неполадок» в конце статьи.

А теперь пришло время добавить в наше приложение новый вид.

Вид

Класс Zend Framework, на базе которого строится вид, называется просто и бесхитро — `Zend_View`. Вид позволяет отделить код, отвечающий за отображение страницы, от кода функций-действий.

Простейший вариант использования `Zend_View` выглядит как показано ниже:

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

Легко можно понять, что если бы мы захотели поместить приведенный код в каждую функцию-действие, это привело бы к его многократному дублированию, что не явилось бы красивым решением. Вместо этого, мы выполним инициализацию вида отдельно, а из функций-действий затем будем только обращаться к нему.

Разработчики Zend Framework предусмотрели решение этой задачи и для ее решения создали так называемые «помощники действий» (“action helpers”). Класс `Zend_Controller_Action_Helper_ViewRenderer` инициализирует свойство `view` для нашего последующего использования (`$this->view`), а так же занимается отображением скриптов вида. Для рендеринга веб-страниц он использует объект `Zend_View`, который в свою очередь находит скрипты видов в файлах `views/scripts/{имя_контроллера}/{имя_действия}.phtml` (по умолчанию они хранятся именно там, но при необходимости дислокацию скриптов можно менять).

После рендеринга, сгенерированный контент веб-страницы сохраняется в объекте `Response`, который, как было упомянуто ранее, служит для группировки HTTP-заголовков, содержимого страницы и сгенерированных в ходе работы скрипта исключительных ситуаций. В итоге, первичный контроллер автоматически отправляет веб-клиенту заголовок страницы и ее содержательную часть.

Итак, для того, чтобы добавить вид в наше приложение, все, что нам потребуется, — создать несколько файлов с тестовым кодом. Никаких принципиальных изменений внутри контроллера не понадобится, но для большей наглядности мы выполним небольшую корректировку выводимого им текста.

Изменения внутри класса `IndexController` выделены жирным шрифтом.

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";
}
}

```

Внутри каждой функции мы присваиваем значение переменной `title` свойства `view`. Заметьте, что при этом в действительности не происходит никакого отображения текста — это задача первичного контроллера, которая будет выполнена в самом конце работы системы.

Теперь нам понадобится добавить четыре файла для видов нашего веб-приложения.

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>

```

При тестировании каждого из действий, должны быть отображены выделенные заголовки каждой из страниц.

Общий HTML-код

Уже сейчас становится заметно использование одинаковых фрагментов HTML-кода в наших видах. Выделим идентичные для всех файлов части в два отдельных файла: `header.phtml` и `footer.phtml`, созданных внутри все той же директории со скриптами.

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">
```

(Обратите внимание на изменения, добавленный в код.)

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

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

Файлы видов так же понадобится откорректировать.

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'); ?>
```

Стилевое оформление

Для того, чтобы сделать наше веб-приложение визуально более привлекательным, используем CSS. На этом этапе возникает небольшая проблема, связанная с тем, что заранее неизвестен URL корневой директории, необходимый для ссылки на CSS-файл. Для разрешения этой задачи используется функция

`getBaseUrl()` передаваемого в вид объекта `Request`. Она и предоставит нам неизвестный заранее фрагмент URL.

Это свойство становится доступным из любого действия, после выполнения `IndexController::init()`.

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

```
...
class IndexController extends Zend_Controller_Action
{
    function init()
    {
        $this->view->baseUrl = $this->_request->getBaseUrl();
    }

    function indexAction()
    {
        ...
    }
}
```

Ссылку на CSS-файл понадобится добавить в секцию `<head>` файла `header.phtml`:

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>
...
```

В завершении, понадобится создать, собственно, сам стилевой файл.

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;
}
```

Это заставит страницы выглядеть несколько приятнее.

База данных

Теперь, когда управляющая часть нашего приложения и код визуализации разделены, пришло время заняться моделью. Помните, что модель — базовая часть приложения, которая реализует его основные функции. Следовательно, в нашем случае модель выполняет работу с базой данных. Мы используем класс Zend Framework `Zend_Db_Table`, предназначенный для поиска, вставки, обновления и удаления записей в таблице базы данных.

Настройка

Для использования `Zend_Db_Table`, понадобится сообщить ему, к какой базе данных, под каким именем пользователя и с каким паролем он будет обращаться. Принимая во внимание, что эту информацию предпочтительно не забивать в код, воспользуемся конфигурационным файлом для ее хранения.

Zend Framework предоставляет для этой цели класс `Zend_Config`, обеспечивающий гибкий объектно-ориентированный доступ к конфигурационным файлам в форматах INI и XML. Остановим свой выбор на INI-файле:

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
```

(Безусловно, вам понадобится использовать свои собственные параметры доступа к БД, а не приведенные в примере.)

Использовать `Zend_Config` будет очень просто:

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

В данном случае, `Zend_Config_Ini` загружает одну секцию из INI-файла (таким же образом при необходимости можно загрузить любую другую секцию). Возможность именования секций реализована для того, чтобы лишние данные без нужды не загружались. `Zend_Config_Ini` использует точку в именах параметров в качестве иерархического разделителя, благодаря чему можно группировать родственные параметры. В нашем файле `config.ini`, параметры `host`, `username`, `password` и `dbname` будут сгруппированы в объекте `$config->db->config`.

Мы будем загружать конфигурационный файл из файла начальной загрузки (index.php):

zf-tutorial/index.php:

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

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

// setup controller
...
```

После загрузки необходимых для работы классов (Zend_Config_Ini и Zend_Registry), в объект \$config загружается секция конфигурационного файла application/config.ini под именем "general". Далее объект \$config включается в реестр, что обеспечивая доступ к нему из всего приложения.

Замечание: В данном примере нет реальной потребности хранить \$config в реестре. Это сделано в качестве примера работы «настоящего» приложения, в конфигурационном файле которого вам, вероятно, придется хранить нечто большее, чем параметры доступа к БД. При использовании реестра, обратите так же внимание, что данные в нем доступны на глобальном уровне, и что при неосторожном использовании это может стать причиной потенциальных конфликтных ситуаций внутри приложения.

Использование Zend_Db_Table

Для того, чтобы использовать класс Zend_Db_Table, нам понадобится передать в него параметра доступа к базе данных, которые были только что загружены. Для этого необходимо создать объект Zend_Db и зарегистрировать его функцией Zend_Db_Table::setDefaultAdapter(). Ниже приведен соответствующий фрагмент кода в файле первичной загрузки:

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');

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

// setup database
$db = Zend_Db::factory($config->db->adapter,
$config->db->config->toArray());
```

```
Zend_Db_Table::setDefaultAdapter($db);

// setup controller
...
```

Создание таблицы

Мы будем использовать базу данных MySQL, поэтому SQL запрос на создание таблицы будет выглядеть так:

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

Этот запрос можно выполнить через любой MySQL-клиент, например, phpMyAdmin или стандартную консольную утилиту.

Добавление тестовой записи

Добавим несколько тестовых записей в таблицу для проверки функциональности главной страницы, на которой они в последствии должны отображаться.

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

Модель

Zend_Db_Table — абстрактный класс, поэтому на его основе необходимо создать специализированный на нашей задаче класс-наследник. Имя нового класса не имеет принципиального значения, но такие классы стоит называть аналогично соответствующим им таблицам БД (это повысит уровень самодокументируемости кода). Таким образом, для нашей таблицы `album` имя класса будет `Album`.

Для того, чтобы передать в `Zend_Db_Table` имя таблицы, которой он будет управлять, необходимо присвоить это значение защищенному свойству класса `$_name`. Стоит отметить, что в классе `Zend_Db_Table` по-умолчанию всегда используется ключевое автоинкрементное поле таблицы с именем `id`. При необходимости значение имени этого поля так же можно менять.

Класс `Album` будет храниться в директории моделей.

zf-tutorial/application/models/Album.php:

```
<?php

class Album extends Zend_Db_Table
{
    protected $_name = 'album';
}
```

Ничего сложного, не так ли? Учитывая, что наши потребности в данном случае совсем невелики, базовой функциональности `Zend_Db_Table` хватит, чтобы полностью их удовлетворить. Если же в ваших задачах понадобится реализовать более сложные операции взаимодействия с базой данных, то класс модели — это именно то место, куда стоит поместить соответствующий код.

Список альбомов

Теперь, когда мы сконфигурировали базу данных, можно приступить к выполнению основной задачи приложения и отобразить несколько записей. Это должно быть реализовано в классе `IndexController`. Каждая функция-действие внутри `IndexController` взаимодействует с таблицей `album` через класс `Album`. Поэтому имеет смысл загрузить его при инициализации контролера внутри функции `init()`.

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

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

Замечание: В данном фрагменте кода приведен пример использования метода `Zend_Loader::loadClass()` для загрузки нестандартных классов. Это срабатывает, т. к. директория моделей (в которой хранится подгружаемый класс `Albums`) была добавлена нами в `include_path`.

Создадим список альбомов из базы с помощью `indexAction()`:

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

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

Функция `fetchAll()` возвращает объект `Zend_Db_Table_Rowset`, который позволит нам сформировать список из всех записей в шаблонном файле вида:

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'); ?>

```

По адресу <http://localhost/zf-tutorial/> теперь должен отображаться список из двух наших альбомов.

Добавление нового альбома

Теперь перейдем к функции добавления нового диска в базу. Эта задача состоит из двух частей:

- отображение формы, через которую пользователь будет вводить данные;
- добавление принятых из формы данных в базу.

Реализуем перечисленные операции в функции-действии `addAction()`:

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

```

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

    if ($this->_request->isPost()) {
        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';
}
...

```

Обратите внимание, как в начале функции было определено, имела ли место пересылка данных их формы. Если данные из формы были переданы, мы извлекаем значения `artist` и `title`, и обрабатываем их фильтром HTML тегов `Zend_Filter_StripTags`. Далее, если строки имеют непустое значение, мы используем класс `Album()` для добавления новой записи в таблицу БД. После добавления альбома, пользователь переадресуется обратно на главную страницу методом контроллера `_redirect()`.

Последнее, что понадобится сделать, — подготовить HTML-форму для шаблона вида. Забегая вперед, можно отметить, что форма редактирования записей будет выглядеть идентично форме для их добавления. Поэтому мы используем общий шаблонный файл (`_form.html`), который будет использован в `add.phtml` и `edit.phtml`:

Шаблон страницы добавления нового альбома:

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>
```

Получился достаточно несложный код. Учитывая, что мы предполагаем применять `_form.phtml` еще и при редактировании записей, используем переменную `$this->action`, вместо того, чтобы жестко задавать имя необходимого действия. Таким же образом, через переменную задается надпись на кнопке передачи данных из формы.

Редактирование альбома

Редактирование альбома во многом идентично добавлению новой записи, поэтому и код получается похожим:

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

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

    if ($this->_request->isPost()) {
        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';
}
...
```

Стоит отметить, что в тех случаях, когда не была выполнена передача данных в скрипт из формы, мы можем получить параметр `id` из свойства `params` объекта `Request`, с помощью метода `getParam()`.

Код шаблона приведен ниже:

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'); ?>
```

Рефакторинг

Безусловно, от вашего внимания не могло ускользнуть, что `addAction()` и `editAction()` очень похожи, а шаблоны добавления и редактирования записей вообще идентичны. Логично предположить необходимость рефакторинга кода. Решите эту задачу самостоятельно, в качестве дополнительного практического упражнения.

Удаление альбома

Для того, чтобы завершить разработку приложения, понадобится добавить функцию удаления записей. У нас предусмотрены ссылки для удаления альбомов напротив каждого элемента в списке. Первое приходящее на ум решение — удалять записи сразу при клике по одной из этих ссылок. Но это неправильный подход. Помните, что согласно спецификации HTTP, не следует выполнять необратимых действий с помощью метода GET. В таких случаях рекомендуется применять POST. Примером того может быть работа Google Accelerator.

Мы должны запрашивать подтверждение перед удалением записей и удалять их только после того, как оно будет получено. Код, реализующий эти действия, в некоторой степени похож на уже существующие в нашем приложении функции-действия:

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

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

    if ($this->_request->isPost()) {
        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) {
                // render template automatically
                return;
            }
        }
    }

    // redirect back to the album list unless we have rendered the view
    $this->_redirect('/');
}
...
```

Использован тот же способ определения метода обращения к скрипту и выбора требуемой операции (удаления записи или выдачи формы подтверждения). Точно так же, как в случае с добавлением и редактированием, удаление выполняется через `Zend_Db_Table` методом `delete()`. В конце функции выполняется перенаправление пользователя на страницу со списком альбомов. Таким образом, если одна из проверок корректности параметров не пройдена, происходит возврат на исходную страницу без помощи многократного обращения к `_redirect()` внутри функции.

Шаблон представляет собой простую форму:

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

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

Устранение неполадок

Если возникают сложности при обращении любым действием помимо `index/index`, скорее всего причина состоит в том, что класс-роутер не может корректно определить, в какой директории находится ваш веб-сайт. Такая ситуация может возникнуть, если URL сайта не соответствует пути к его директории относительно корневого каталога, открытого для доступа из сети.

Если приведенный выше код не соответствует вашему случаю, необходимо задать переменной `$baseUrl` корректное для вашего сервера значение:

zf-tutorial/index.php:

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

Значение `"/mysubdir/zf-tutorial/"` понадобится заменить на действительный путь к файлу `index.php`. Например, если ваш URL к `index.php` выглядит как `http://localhost/~ralle/zf-tutorial/index.php`, корректным значением для переменной `$baseUrl` будет `"/~ralle/zf-tutorial/"`.

Заключение

На этом построение простого но полнофункционального MVC-приложения можно считать завершенным. Надеюсь, этот обзор был полезен и информативен для вас. Любые замечания к оригинальному тексту можно отправлять автору статьи по адресу rob@akrabat.com. Комментарии, связанные с русским переводом, отправляйте на musayev@yandex.ru.

В данной статье выполнен лишь поверхностный обзор Zend Framework, в котором существует великое множество других классов, помимо перечисленных здесь. Для подробного ознакомления с ними следует обратиться к соответствующим ресурсам:

- Официальная документация:
<http://framework.zend.com/manual>
- Wiki с дополнительными материалами:
<http://framework.zend.com/wiki>
- Wiki для разработчиков:
<http://framework.zend.com/developer>

При переводе использовались материалы свободной энциклопедии Wikipedia.org.

Данный перевод может быть найден по постоянному адресу:

<http://archive.paradigm.ru/zend-fw-intro.pdf>

Оригинал статьи на английском доступен на сайте автора:

<http://akrabat.com/zend-framework-tutorial/>