Iniciando com Zend Framework

Por Rob Allen, www.akrabat.com
Tradução: José Luciano Brandão Calazans Jr., luciano.calazans@gmail.com
Revisão do documento 1.5.2
Copyright © 2006, 2008

Este tutorial pretende dar uma introdução básica ao uso do Zend Framework através de uma aplicação baseada em bancos de dados.

NOTA: Este tutorial foi testado com a versão 1.5 do Zend Framework. Existe uma grande chance de ser compatível com versões posteriores da série 1.5.x, mas certamente não funcionará com versões anteriores a 1.5.

Arquitetura Model-View-Controller (Modelo-Visão-Controle)

A maneira tradicional de desenvolver uma aplicação PHP é fazer algo parecido com o seguinte:

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

Através do tempo de vida de uma aplicação, uma aplicação escrita desta forma se torna passível de difícil manutenção conforme o cliente continue requisitando mudanças que são incluídas em diversos locais de seu código.

Um método que melhora a manutenção de uma aplicação é separar o código de um arquivo em três partes distintas (e normalmente arquivos separados):

Model	A parte de modelo de uma aplicação é a parte que se preocupa com os dados específicos a serem mostrados. No código de exemplo acima, é o conceito de “news”. Dessa forma, model é geralmente relacionado com a lógica de negócios de uma aplicação e administra o carregamento e o salvamento de dados em um banco de dados.
View	A view consiste em uma pequena parte da aplicação que é responsável em mostrar a informação ao usuário. Normalmente, é o HTML.
Controller	O controller amarra o view e o model para garantir que as informações corretas sejam mostradas na página.

O Zend Framework usa a arquitetura Model-View Controller (MVC). Isto é usado para separar as diferentes partes de sua aplicação para tornar o desenvolvimento e manutenção mais fácil.

Requisitos

O Zend Framework possui os seguintes requisitos:

- PHP 5.1.4 (ou superior)
- Um servidor web que suporte a funcionalidade de mod_rewrite (reescrita de URLs).

Pressupostos do Tutorial

Pressupomos neste tutorial que você está executando o PHP 5.1.4 ou superior com o servidor web Apache. Sua instalação Apache deve ter a extensão mod_rewrite instalada e configurada.

Você também deve assegurar-se que o Apache está configurado para suportar arquivos *.htaccess*. Isto é geralmente feito pela mudança na configuração:

AllowOverride None

para

AllowOverride All

no seu arquivo httpd.conf. Consulte a sua documentação da distribuição para detalhes precisos. Você não será capaz de navegar para qualquer página que não seja a página inicial neste tutorial se não tiver configurado o mod_rewrite e o .htacess corretamente.

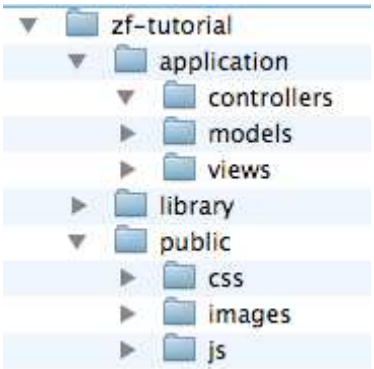
Obtendo o Framework

O Zend Framework pode ser baixado de http://framework.zend.com/download nos formatos .zip ou .tar.gz.

Estrutura do Diretório

Apesar de que o Zend Framework não designa uma estrutura de diretório, o manual recomenda uma estrutura comum, é assim que vamos proceder neste tutorial. Esta estrutura assume que você tenha controle completo sobre a sua configuração do Apache, de modo que você possa manter a maior parte dos arquivos fora do diretório raiz da web.

Comece criando um diretório no diretório raiz do servidor web chamado `zf-tutorial` e crie os seguintes subdiretórios para receber os arquivos da aplicação:



Como você pode ver, temos diretórios separados para os arquivos do `model`, `view` e `controller` da nossa aplicação. O diretório public/ é o diretório raiz do site, o que significa que a URL para a aplicação será `http://localhost/zf-tutorial/public/`. Esse é, portanto, o porque que a maior parte dos arquivos da aplicação não são acessíveis diretamente pelo Apache e por isso são mais seguros.

> **Note:**
>
> Em um live site, você iria criar um host virtual para o site e definir o documento raiz diretamente à pasta pública. Por exemplo, você poderia criar um host virtual chamado `zf-tutorial.localhost` que parece algo assim:
>
> ```
> <VirtualHost *:80>
>       ServerName zf-tutorial.localhost
>       DocumentRoot /var/www/html/zf-tutorial/public
>       <Directory "/www/cs">
>             AllowOverride All
>       </Directory>
> </VirtualHost>
> ```
>
> O site passaria então ser acessado pelo endereço `http://zf-tutorial.localhost/` (certifique-se de atualizar seu /etc/hosts ou c:\windows\system32\drivers\etc\hosts, arquivo para que zfia.localhost seja mapeado para 127.0.0.1)

Imagens, scripts e arquivos CSS são guardados em diretórios separados sob o diretório public. Os arquivos do Zend Framework serão colocados no diretório library. Se nós precisarmos utilizar outras bibliotecas, elas poderão ser colocadas lá.

Extraia o arquivo descarregado, no meu caso, ZendFramework-1.5.0.zip, para um diretório temporário. Todos os arquivos foram colocados em um subdiretório chamado `ZendFramework-1.5.0`. Copie o subdiretório `/library/Zend` para `zf-tutorial/library/`. Seu diretório `zf-tutorial/library` agora deve conter um sub-diretório chamado `Zend`.

Inicializando

O controller do Zend Framework, Zend_Controller é projetado para dar suporte a websites com urls limpas. Para alcançar isso, todas as requisições precisam passar por um único arquivo chamado index.php, conhecido como bootstraper. Isto nos provê um ponto central para todas as páginas da aplicação e garante que o ambiente está configurado corretamente para rodar a aplicação. Nós alcançamos isso usando um arquivo .htaccess no diretório `zf-tutorial/public/`:

```
# Rewrite rules for Zend Framework
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule .* index.php

# Security: Don't allow browsing of directories
Options –Indexes

# PHP settings
php_flag magic_quotes_gpc off
php_flag register_globals off
php_flag short_open_tag on
```

A RewriteRule é muito simples e pode ser interpretada como “para qualquer url, redirecione para index.php”.

Nós também setamos um par de diretivas PHP para segurança e sanidade e setamos a configuração do short_open_tag para on para ser usado nos scripts do view. Estes valores já deveriam estar setados corretamente no php.ini, mas nós queremos ter certeza disso! Note que a flag php_flag no .htaccess só funcionará se você estiver usando mod_php. Se você usa CGI/FastCGI, então você deverá se certificar que o seu php.ini está correto.

Note que para que os arquivos .htaccess sejam usados pelo Apache, a diretiva de configuração AllowOverride precisa estar setada como All no seu arquivo httpd.conf.

O arquivo de inicialização: index.php

O arquivo `zf-tutorial/public/index.php` é o nosso arquivo de bootstrap e nós o iniciaremos com o código a seguir:

zf-tutorial/public/index.php

```
<?php

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

// directory setup and class loading
set_include_path('.' . PATH_SEPARATOR . '../library/'
       . PATH_SEPARATOR . '../application/models'
       . PATH_SEPARATOR . get_include_path());
include "Zend/Loader.php";
Zend_Loader::registerAutoload();

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

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

Repare que nós não colocamos a tag ?> no final do arquivo porque isso não é necessário e deixar isso for a irá prevenir alguns erros difíceis de debugar quando utilizar o redirecionamento via função `header()` caso exista algum espaço em branco após a tag ?>.

Vamos percorrer o arquivo.

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

Estas linhas irão nos garantir que nós veremos qualquer erro que gerarmos. Nós também selecionamos nosso fuso-horário corrente conforme é requerido pelo PHP 5.1+. Obviamente você deve escolher o seu fuso horário.

```
// directory setup and class loading
set_include_path('.' . PATH_SEPARATOR . '../library/'
       . PATH_SEPARATOR . '../application/models'
       . PATH_SEPARATOR . get_include_path());
include "Zend/Loader.php";
Zend_Loader::registerAutoload();
```

O Zend Framework é projetado para que seus arquivos estejam no include_path. Nós também colocamos nosso diretório de modelos (models) no include_path para que nós possamos carregá-los facilmente depois. Para iniciar, nós precisamos incluir o arquivo `Zend/Loader.php` que nos dará acesso à classe `Zend_Loader` e depois chamar as funções

do `registerAutoload()`na ordem para carregar automaticamente todos os arquivos do Zend Framework que nós instanciamos.

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

Nós precisaremos configurar o front controller para que ele saiba em qual diretório se encontra os nossos controllers.

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

Como isto é um tutorial e nós estamos executando um sistema de testes, eu decidi instruir o front controller para disparar todas as exceções que ocorrerem. Por padrão, o front controller irá capturá-los para nós e redirecioná-los para o controller ErrorController para você. Isto pode ser um pouco confuso para as pessoas que são novatas com o Zend Framework, então é mais fácil apenas re-jogar as exceções. É claro que em um servidor de produção você não deverá mostrar os erros ao usuário de forma nenhuma.

O front controller utiliza uma classe de roteamento para mapear o URL solicitado para a função PHP correta para ser usado para exibir a página. Na ordem do roteador para operar, ele precisa para verificar qual parte do URL é o caminho para a nossa index.php para que ele possa enxergar os elementos URI após este ponto. Isso é feito pelo objeto Request (solicitante). Ele faz um trabalho muito bom de auto-detectar a URL base correta, mas se ele não funcionar para sua configuração, então você pode sobrepor-lo usando a função `$frontController->setBaseUrl()`.

Finalmente, nós chegamos no ponto principal e vamos rodar nossa aplicação:

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

Se você digitar http://localhost/zf_tutorial/public/ para testar, você certamente encontrará um erro similar a:

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

Isto está nos dizendo que nós não configuramos nossa aplicação ainda. Antes de fazermos isso, nós iremos discutir o que nós iremos construir, então vamos fazer isso a seguir.

O Website

Nós iremos construir um sistema muitos simples estoque para mostrar a nossa coleção de CD. A página principal irá listar a nossa coleção e nos permitirá adicionar, editar e excluir CDs. Nós iremos gravar nossa lista em um banco de dados em um esquema como este:

Campo	*Tipo*	*Null?*	*Descrição*
Id	Integer	No	Primary key, Autoincrement
Artist	Varchar(100)	No	
Titl	Varchar(100)	No	

Páginas necessárias

As páginas a seguir serão necessárias.

Home page	Irá mostrar a lista dos álbuns e providenciar links para editá-los e deleta-los. Um link que permitirá adicionar novos Álbuns também estará disponível.
Add New Album	Mostrará um formulário para adicionar um novo álbum
Edit Album	Mostrará um formulário para edição de um álbum
Delete Album	Esta página irá confirmar que nós queremos deletar um álbum e então o deletará.

Organizando as páginas

Antes de nós configurarmos nossos arquivos, é importante entender como o framework espera que as páginas sejam organizadas. Cada página da aplicação é conhecida como uma "ação" e ações são agrupadas em "controllers". Ex: para a URL com o formato `http://localhost/zf-tutorial/public/news/view`, o controller é `news` e a ação é `view`. Isto permite um agrupamento de ações relacionadas. Por exemplo, o controller `news` pode ter ações de `list`, `archived` e `view`. O sistema de MVC do Zend Framework também suporta módulos para agrupar controllers, mas esta aplicação não é grande o suficiente para nos preocuparmos com isso.

Por padrão, o controller do Zend Framework reserva uma ação especial chamada `index` como uma ação padrão. Isto é, uma url como `http://localhost/zf-tutorial/public/news/` executará uma ação `index` que está no controller news. O Controller do Zend Framework também reserve um controller padrão para caso nenhuma seja solicitado. Não é nenhuma surpresa se ele se chamar index também. Dessa forma, a url http://localhost/zf-tutorial/ irá fazer com que a ação index no controller index seja executada.

Como este é um simples tutorial, nós não iremos nos incomodar com coisas "complicadas" como login! Isso pode esperar por um tutorial separado...

Como nós temos quarto páginas que se aplicam a álbuns, nós iremos agrupá-las em um único controller como quatro ações. Nós devemos usar o controller default e as quatro ações serão:

Página	*Controller*	*Ação*
Página	Index	index
Adicionar novo álbum	Index	Add
Editar álbum	Index	Edit
Excluir Album	Index	Delete

Bom e simples!

Configurando o Controle

Agora nós estamos prontos para configurar o nosso controller. No Zend Framework, o controller é uma classe que precisa ser denominada `{Nome do controller}Controller`. Veja que `{Nome do controller}` precisa começar com uma letra maiúscula. Esta classe precisa estar em um arquivo chamado `{Nome do controller}Controller.php` dentro do diretório de `controllers` especificado. Novamente, `{Nome do controller}` precisa iniciar com uma letra maiúscula e todas as outras precisam ser minúsculas. Cada ação é um método público dentro da classe de controle e precisa se chamar `{nome da ação}Action`. Neste caso, `{nome da ação}` deve iniciar como uma letra minúscula. Nomenclatura mista

(maiúsculas e menúsculas) dos controllers e das ações são permitidas, mas têm regras especiais que você deve compreender antes de usá-las. Verifique a documentação primeiro!

Assim, nossa classe de controle é denominada `IndexController` que está definida em `zf-tutorial/application/controllers/IndexController.php`. Crie este arquivo para configurar o esqueleto:

zf-tutorial/application/controllers/IndexController.php

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

Agora configuramos as quatro ações que vamos utilizar. Elas não irão funcionar até que seja criado as views. As URLs para cada ação são as seguintes:

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

Nós temos agora um roteador funcionando e as ações estão sendo executadas corretamente para cada página de nossa aplicação. Caso isso não funcione com você, confira na seção Defeitos que se encontra no final deste tutorial e veja se te ajuda.

É hora de construirmos a view.

Configurando a Visão

O Componente de visão do Zend Framework é chamado, de certa forma sem surpreender, `Zend_View`. O componente de visão nos permitirá separar o código que mostra a página do código dos métodos de ação.

O uso básico do `Zend_View` é:

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

Podemos observar facilmente que se nós colocarmos este esqueleto diretamente em cada um de nossos métodos de ação, nós iremos repetir o código de configuração, o que não é de interesse da ação. Nós devemos de preferência, fazer a inicialização da visão em algum local e então acessar o nosso objeto que já está inicializado, em cada um dos métodos de ação.

Os projetistas do Zend Framework previram este tipo de problema e a solução foi criada no “action helper” para nós. *Zend_Controller_Action_Helper_ViewRenderer* teve o cuidado de

inicializar uma propriedade do view (`$this->view`) para nós e irá renderizar um script do view também. Para a renderização, ele também configura o objeto `Zend_View` para procurar em `views/scripts/{nome do controller}` pelos scripts de visão(view) que devem ser renderizados e irá (por padrão, ao menos) renderizar o script que está nomeado depois da ação com extensão .phtml ({ação}.phtml). Isto é, o script do view renderizado é `views/scripts/{nome do controller}/{nome da ação}.phtml` e anexará isso ao corpo do objeto Response. O Objeto Response é usado para combinar todos os cabeçalhos HTTP, conteúdo de corpo e exceções geradas como resultado do uso do sistema de MVC. O front controller então automaticamente envia os cabeçalhos e em seguida o corpo do conteúdo no final.

Para integrar a visão na nossa aplicação nós precisamos criar alguns arquivos de views e então assegurar que isto funciona, nós adicionaremos alguns conteúdo específico do action (o título da página) dentro das ações do controller.

As mudanças no `IndexController` serão mostradas a seguir (mudanças em negrito):

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";
	}
}
```

Em cada método, nós determinamos uma variável title à propriedade view e então é isso! Repare que a exposição não ocorre neste ponto - Isto é feito pelo front controller no final do processo.

Agora nós precisamos adicionar quatro arquivos de visão para a nossa aplicação. Estes arquivos são conhecidos como scripts ou templates como referido anteriormente, cada arquivo de template é nomeado após esta ação e terá a extensão .phtml para demonstrar que este é um arquivo de template. O arquivo precisa estar em um subdiretório que foi criado após o controller, então os quatro arquivos são:

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>
```

O teste para cada controle/ação deverá mostrar os quatro títulos em negrito.

Códigos HTML comum

Rapidamente notamos que existem muito código HTML comum em nossas views. Isto é um problema muito comum e o componente *Zend_Layout* é desenvolvido para resolver este problema. Zend_Layout nos permite mover todos os códigos comuns dos cabeçalhos(header) e rodapés(footer) para um layout do view script que inclui o código do view, que é específico para a ação ser executada.

As seguintes alterações são necessárias. Primeiramente temos de decidir onde colocar os nossos scripts de layout do view. O local recomendado é no diretório `application`, com isso crie um diretório chamado `layouts` no diretório `zf-tutorial/application`.

Temos que inicializar o sistema Zend_Layout no arquivo de inicialização, assim adicionamos em `public/index.php`, como isto:

zf-tutorial/public/index.php:

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

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

A função `startMvc()`faz alguns trabalhos nos bastidores para configurar um plugin front controller que irá garantir que o componente Zend_Layout renderize o script de layout com os scripts de ação do view dentro dele no final do processo dispatch.

Agora precisamos de um script de layout do view. Por padrão, ele é chamado de layout.phtml e fica no diretório de layouts. Ele é assim:

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>
```

Note que fizemos o nosso código XHTML compatível e é o código HTML padrão para a exibição de uma página. Como o título da página na tag <h1> é exibido em todas as páginas, passamos ele para o arquivo de layout e utilizaremos o ajudante (helper) do view `escape()` para assegurar que ele está codificado adequadamente.

Para obter o script do view para a ação atual para exibição, nós exibiremos o conteúdo do placeholder usando o ajudante (helper) do view do layout():`echo $this->layout()->content;` o que faz o trabalho por nós. Isto significa que os scripts do view para a ação estão executando do script de layout do view.

Podemos agora esvaziar os 4 scripts de ação pois ainda não temos nada específico para colocar nele, assim prosseguimos e esvaziamos os arquivos index.phtml, add.phtml, edit.phtml e delete. phtml.

Agora você pode testar todas as 4 URLs novamente e poderá ver que não tem nenhuma diferença da última vez que você testou! A diferença chave é que desta vez, todo trabalho é feito no layout.

Estilos

Mesmo que isso seja “somente” um tutorial, nós precisaremos de um arquivo CSS para tornar a nossa aplicação um pouco apresentável! Isso causa um pequeno problema porque atualmente nós não sabemos como referenciar o arquivo CSS porque a URL não aponta para o diretório raiz correto. Para resolver isso, nós criamos nosso próprio ajudante (helper) do view, chamado `baseUrl()` que coleta a informação que solicitamos do objeto solicitado. Isto nos proporciona o bit da URL que nós não conhecemos.

Os ajudantes (helpers) do view ficam no subdiretório `application/views/helpers` e é nomeado `{Nome do ajudante}.php` (a primeira letra deve ser maiúscula) e a classe dela deve ser chamada `Zend_Controller_Helper_{Nome do ajudante}` (novamente, a primeira letra é maiúscula). Deve haver uma função na classe chamada `{nome do ajudante}()` (primeira letra minúscula - não esqueça!). Em nosso caso, o arquivo é chamado `BaseUrl.php` e é assim:

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

```
<?php
class Zend_View_Helper_BaseUrl
{
       function baseUrl()
       {
```

```
        $fc = Zend_Controller_Front::getInstance();
        return $fc->getBaseUrl();
        }
}
```

Não é uma função complicada. Nós simplesmente recriamos uma instancia para o front controller e retornamos a função membro `getBaseUrl()`.

Nós precisamos adicionar o arquivo CSS para a seção <head> do arquivo application/layouts/layout.phtml:

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>
...
```

Finalmente, nós precisamos de alguns estilos CSS:

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;
}
```

Isto deve tornar ligeiramente mais bonito!

O Banco de dados

Agora que temos o controle da aplicação separado da visão, é hora de olhar a seção de modelo (model) da nossa aplicação. Lembre-se que o modelo é a parte que lida com o núcleo da aplicação (também chamado de “regras de negócio”) , no nosso caso, lida com o banco de dados. Nós faremos uso da classe `Zend_Db_Table` do Zend Framework que é usada para pesquisar, inserir, alterar e excluir linhas de uma tabela do banco de dados.

Configuração

Para usar a `Zend_Db_Table`, nós precisamos dizer a ela qual é o banco de dados que usaremos e também o usuário e a senha. Como nós preferimos não colocar estes dados dentro da nossa aplicação, nós usaremos um arquivo de configuração para guardar esta informação.

O Zend Framework oferece a classe `Zend_Config` para oferecer um acesso orientado a objetos a arquivos de configuração. O arquivo de configuração pode ser um arquivo INI ou um arquivo XML. Nós usaremos o arquivo INI chamado `config.ini` e armazenaremos ele no diretório `application/`:

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
```

Obviamente você deve usar o seu nome de usuário, senha e banco de dados, não o meu! Para aplicações maiores com muitos arquivos de configuração você pode decidir criar um diretório separado, como `application/config` e armazenar todos seus arquivos de configuração juntos, fora do jeito.

É muito fácil usar o `Zend_Config`:

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

Repare que neste caso, `Zend_Config_Ini` carrega uma seção do arquivo INI e não todas as seções do arquivo (mas todas as seções podem ser carregadas se você desejar). Ele suporta uma notação no nome da seção para permitir que sejam carregadas seções adicionais. `Zend_Config_Ini` também trata o “ponto” no parâmetro como separadores hierárquicos que permite o agrupamento de parâmetros de configuração relacionados. No nosso `config.ini`, os parâmetros servidor (host), usuário (username) e o nome do banco de dados (dbname) serão agrupados em `$config->params->config`.

Nós iremos carregar o nosso arquivo de configuração no nosso arquivo de inicialização (public/index.php):

Parte relevante de 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();
...
```

As mudanças estão em negrito. Nós carregamos as classes que vamos utilizar (`Zend_Config_Ini` e `Zend_Registry`) e então carregar a sessão '`general`' de `application/config.ini` em nosso objeto `$config`. Finalmente nós atribuimos o objeto $config ao registro para que possamos recuperá-lo em qualquer lugar da aplicação.

Nota: Neste tutorial, nós não precisamos guardar o `$config` no registro, mas é uma boa prática em uma aplicação 'real' que você terá mais que uma configuração de banco de dados no arquivo INI. Porém, esteja avisado que o registro é um pouco global e causa dependências entre objetos que não deveriam depender um do outro, se você não for cuidadoso.

Configurando Zend_Db_Table

Para usar a `Zend_Db_Table`, nós precisaremos informar a configuração de banco de dedos que nós carregamos. Para fazer isso nós precisamos criar uma instência da `Zend_Db` e então registra-la usando a função estática `Zend_Db_Table::setDefaultAdapter()`. Novamente, nós fazemos isso no arquivo de inicialização (adições em negrito):

Parte relevante de 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();
...
```

Como você pode ver, a `Zend_Db_Table` tem uma função estática `factory()`que interpreta os dados a partir do objeto `$config->db` e instancia o adaptador correto do banco de dados para nós.

Criando a tabela

Eu estarei usando MySQL e então o comando SQL para criar a tabela é:

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

Execute o comando em um client MySQL como o phpMyAdmin ou o cliente de linha de comando MySQL padrão.

Inserindo albums de teste

Nós precisaremos inserir algumas linhas na tabela para que nós possamos testar a funcionalidade de recuperação da página principal. Eu estarei pegando os primeiros dois CDs do "Top Sellers" do site Amazon.co.uk:

```
INSERT INTO albums (artist, title)
VALUES
       ('Duffy', 'Rockferry'),
       ('Van Morrison', 'Keep It Simple');
```

O Modelo

`Zend_Db_Table` é uma classe abstrata, então nós temos que derivar nossa classe que é específica para gerenciar os albums. Não importa como denominaremos a nossa classe, mas faz sentido que ela tenha o mesmo nome que a tabela do banco de dados. Assim, nossa classe será denominada `Albums` como a nossa tabela que se chama `albums`. Para informar a `Zend_Db_Table` o nome da tabela que iremos gerenciar, nós temos que setar a propriedade protegida `$_name` para o mesmo nome da tabela. Também, `Zend_Db_Table` assume que a chave primária da sua tabela seja denominada `id` que é auto-incrementada pelo banco de dados. O nome deste campo pode ser mudado também se necessário.

Nós iremos guardar nossa classe Album num arquivo chamado Album.php no diretório applications/models:

zf-tutorial/application/models/Albums.php

zf-tutorial/application/models/Albums.php

```
<?php

class Albums extends Zend_Db_Table
{
       protected $_name = 'albums';
}
```

Não é muito complicado, né?! Felizmente para nós, nossas necessidades são muito simples e a `Zend_Db_Table` fornece toda a funcionalidade que precisamos. De qualquer forma, se você necessita de alguma funcionalidade específica para gerenciar seu modelo, então é nesta classe que você deve colocá-la. Geralmente, as funcionalidades adicionais que serão métodos como “find” que retornarão coleções de dados que você precisa. Você também pode informar a `Zend_Db_Table` sobre relacionamentos de tabelas e ela busca os dados relacionados também.

Listando os albums

Agora que nós já fizemos a configuração e as informações do banco de dados, nós podemos ir para a parte a aplicação que mostra alguns álbuns. Isto sera feito na classe `IndexController` e nós iniciamos listando os albums numa tabela na função `indexAction()`:

zf-tutorial/application/controllers/IndexController.php

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

A função `fetchAll()` retorna a `Zend_Db_Table_Rowset` que nos permitirá fazer a iteração das linhas retornadas no arquivo de script das ações do view. Agora nós podemos preencher o arquivo índex.phtml

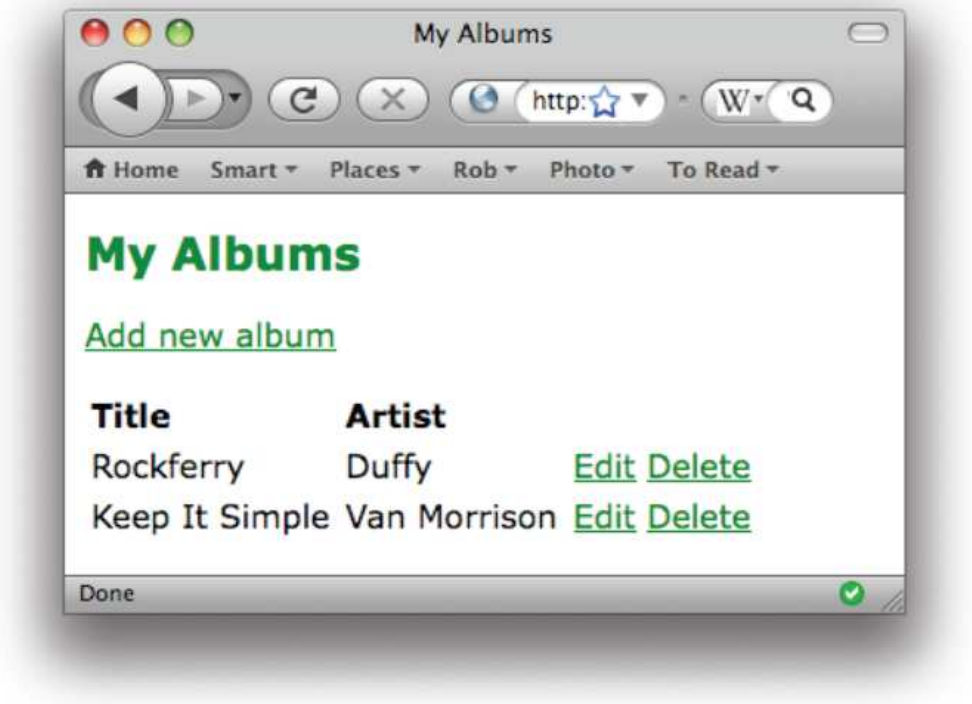
zf-tutorial/application/views/scripts/index/index.phtml

```
<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>
```

A primeira coisa que faremos é criar um link para adicionar um novo álbum. A `url()` é fornecido pelo framework e ajuda a criar links incluindo a URL base correta. Nós simplesmente passamos em um array dos parâmetros que.

Nós então criaremos uma tabela html para mostrar cada título do álbum, artista e fornecer links para permitir a edição e apagar o registro. Um loop padrão `foreach` é usado para fazer a iteração da lista de álbuns, e utilizaremos a forma alternativa usando um colon `endforeach`. Novamente, a `url()`do helper do view é usado para criar os links de editar e apagar.

http://localhost/zf-tutorial/ (ou de onde você estiver chamando!) deverá mostrar agora uma bela lista de (dois) álbuns, como esta:



Adicionando novos álbuns

Nós poderemos agora codificar a funcionalidade para adicionar novos albums. Abaixo duas partes:

- Mostrar um formulário para o usuário fornecer os detalhes
- Processar a submissão do formulário e gravar em um banco de dados

Usamos *Zend_Form* para fazer isto. O componente Zend_Form nos permite criar um formulário e validar a entrada (input). Nós criamos um novo classe do model `AlbumForm` que se estende do `Zend_Form` para definir nosso formulário:

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));
	}
}
```

Dentro do construtor do `AlbumForm`, criamos quatro elementos de formulário para a id, artista, título, e botão submit. Para cada item nós definiremos vários atributos, incluindo o rótulo a ser exibido. Para os elementos de texto, acrescentamos dois filtros, `StripTags` e `StringTrim` para eliminar indesejadas HTML e espaços em branco desnecessários. Nós também iremos configurá-los para ser necessário e adicionar um validador `NotEmpty` a fim de garantir que o usuário insira a informação necessária.

Precisamos agora obter o formulário para exibir e depois processá-lo na submissão. Isso é feito no `addAction()`:

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);
			}
		}
	}
...
```

Vamos examinar isto com um pouco mais de detalhes:

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

Nós instanciamos nossa `AlbumForm`, setamos o label do botão submit para “Add” e, em seguida, atribuímos a view para renderizar.

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

Se os objetos solicitados do método `isPost()`for true, então o formulário foi submetido e, por isso, nós recuperamos os formulários de dados do pedido usando `getPost()` e verificar para ver se ele é válido usando a função membro `isValid()`.

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

Se o formulário for válido, então instanciamos a classe `Albums` do model e utilizamos `createRow()` para obter uma linha vazia e, em seguida, preencher o artista (artist) e título (title) antes de salvar. Depois de salvar a nova linha do album, redirecionamos usando o método `_redirect()` do controller para voltar à página inicial.

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

Se os dados do formulário não for válido, então nós preencheremos o formulário com os dados que o usuário preencheu reexibiremos.

Precisamos agora renderizar o formulário no script `add.phtml` do view.

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

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

Como você pode ver, renderizar um formulário é muito simples, como o formulário sabe como ser mostrado na tela.

Editando um álbum

Editar um álbum é quase idêntico a adicionar um, então o código é muito similar:

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());
		}
	}
}
...
```

Vamos analisar as diferenças em adicionar um album. Primeiramente, ao exibir o formulário para o usuário nós precisamos de recuperar o artista do álbum e o título do banco de dados e preencher os elementos do formulário:

```
		// 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());
		}
```

Isto é feito se o pedido (request) não é um POST e utiliza o modelo (model) para recuperar a linha do banco de dados. A classe `Zend_Db_Table_Row` tem uma função membro `toArray()` que podemos utilizar para preencher o formulário diretamente.

Finalmente, precisamos salvar o dado na linha direita do banco de dados. Isto estará concluído pela recuperação da linha de dados e depois salvá-la novamente.

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

O template do view é o mesmo de add.phtml.

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

Agora você já poderá adicionar e editar álbuns.

Excluindo um álbum

Para deixar nossa aplicação redonda, nós precisamos adicionar a exclusão. Nós temo um link Delete perto de cada álbum em nossa página da listagem e ele tem que executar uma exclusão quando ele é clicado passando os parâmetros por GET. Isto pode estar errado. Relembrando nossa especificação HTTP, nós não devemos fazer uma ação irreversível usando GET e devemos usar POST.

Nós mostraremos um formulário de confirmação quando o usuário clicar em excluir e se ele clicar em “sim”, nós faremos a exclusão. Como este formulário é trivial, nós iremos apenas criar o formulário HTML no script do view.

Vamos iniciar com o código da ação:

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);
		}
	}
}
...
```

Utilizaremos o método de requisição isPost() para o funcionamento se nós queremos exibir o formulário de confirmação ou se nós quisermos fazer uma exclusão através da classe Albm(). A real exclusão é feita através de uma chamada para o método delete() da `Zend_Db_Table`. Se a requisição não é um post, então nós olharemos para um parâmetro id e recuperaremos o registro correto do banco de dados e atribuiremos para o view.

O Template (script) é um simples formulário:

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;?>
```

Neste script, apresentaremos uma mensagem de confirmação para o usuário e, em seguida, um formulário com os botões “sim” e “não”. Na ação, verificamos especialmente para o valor “Sim” ao fazer a exclusão.

É isso aí – você já tem uma aplicação completa funcionando.

Defeitos

Se você está tendo problemas ao executar outra ação diferente de index/index, então o problema é que o roteador está incapaz de determinar em qual subdiretório seu website está. Das minhas investigações, isso normalmente acontece quando a url para seu website difere do caminho para o diretório raíz do servidor web.

Se o código padrão não funcionou para você, então você deve setar a variável $baseURL para o valor correto do seu servidor:

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'));
...
```

Você deverá substituir `'/mysubdir/zf-tutorial/public'` pela URL correta para o index.php. Por exemplo, se a URL para seu index.php é http://localhost/~ralle/zf_tutorial/public/index.php então o valor correto para `$baseUrl` é `'/~ralle/zf-tutorial/public'`.

Conclusão

Isto conclui nossa visão geral construindo uma aplicação MCV simples, mas completamente funcional usando o Zend Framework. Eu espero que você tenha achado interessante e informativo. Se você achou que algo está errado, por favor, me envie um email para: rob@akrabat.com!

Este tutorial mostrou apenas o básico do Zend Framework; ainda existem muitas classes a serem exploradas. Você realmente deveria ir e ler o manual (http://framework.zend.com/manual) e olhar o wiki (http://framework.zend.com/wiki) para mais introspecções! Se você está interessado no desenvolvimento do framework, então o wiki de desenvolvimento (http://framework.zend.com/developer) é o seu lugar.