Iniciando com Zend Framework

Por Rob Allen, www.akrabat.com
Tradução: Adler Brediks Medrado, http://www.neshertech.net/adler
Revisão do documento 1.3.0

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 as versões 0.9 e 0.9.1 do Zend Framework. Existe uma grande chance de ser compatível com versões posteriores, mas certamente não funcionará com versões anteriores a 0.9.

AVISO para a versão 0.9: Se você fez o download da versão 0.9 do Zend Framework, então você precisará editar o arquivo library/Zend/Db/Table/Row/Abstract.php e adicionar "<" no início da primeira linha.

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 (or maior)
- Um servidor web que suporte a funcionalidade de mod_rewrite (reescrita de URLs). Este tutorial assume que esteja sendo utilizado o Apache.

Obtendo o Framework

O Zend Framework pode ser baixado de http://framework.zend.com/download/stable nos formatos .zip ou .tar.gz. No momento em que o tutorial era escrito, a versão 0.9 era a versão corrente. Você precisa baixar a versão 0.9 para este tutorial funcionar.

Estrutura de diretórios

Apesar de que o Zend Framework não designa uma estrutura de diretório, o manual recomenda uma estrutura comum. Esta estrutura assume que você tenha controle completo sobre a sua configuração do Apache, no entanto, nós queremos tornar a vida um pouco mais fácil, então usaremos uma modificação.

Comece criando um diretório no diretório raiz do servidor web chamado zf-tutorial. Isto significa que a URL que apontará para a aplicação será: `http://localhost/zf-tutorial`.

Crie os seguintes subdiretórios para receber os arquivos da aplicação:

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

Como você pode ver, nós separamos diretórios para os arquivos de model, view e controller da sua aplicação.
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-0.9.1-Beta.zip, para um diretório temporário. Todos os arquivos foram colocados em um subdiretório chamado `ZendFramework-0.9.1-Beta`. Copie o conteúdo de `ZendFramework-0.9.1-Beta/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 raiz `zf-tutorial`:

zf-tutorial/.htaccess

```
RewriteEngine on
RewriteRule .* index.php

php_flag magic_quotes_gpc off
php_flag register_globals off
```

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. 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.
Porém, requisições para imagens, arquivos JavaScript e CSS não deverão ser redirecionados para o nosso arquivo de inicialização. Mantendo estes arquivos dentro do subdiretório público, nós podemos facilmente configurar o Apache para servir estes arquivos diretamente com outro arquivo .htaccess em `zf-tutorial/public`:

zf-tutorial/public/.htaccess

```
RewriteEngine off
```

Apesar de não ser estritamente necessário, nós podemos adicionar mais um par de arquivos .htaccess para garantir que nossos diretórios application and library estejam protegidos:

zf-tutorial/application/.htaccess

```
deny from all
```

zf-tutorial/library/.htaccess

```
deny from all
```

Note que para que os arquivos .htaccess sejam usados pelo Apache Note that for .htaccess , a diretiva de configuração AllowOverride precisa estar setada como All no seu arquivo httpd.conf.
A idéia aqui apresentada de usar múltiplos arquivos .htaccess é do artigo Blueprint for PHP Applications: Bootstrapping (Parte 2) de Jayson Minard”. Eu recomendo a leitura das duas partes.

O arquivo de inicialização: index.php

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

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();
```

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);
date_default_timezone_set('Europe/London');
```

Estas linhas irão nos garantir que nós veremos qualquer erro que gerarmos (assumindo que a diretiva `display_errors` esteja como on). Nós também selecionamos nosso fuso-horário corrente conforme é requerido pelo PHP 5.1+. Obviamente você deve escolher o seu fuso-horário.

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

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 que possui as funções estáticas que nos permitirá carregar qualquer outra classe do Zend Framework.

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

`Zend_Loader::loadClass` carrega a classe desejada. Isto é feito pela conversão dos underscores do nome da classe em separadores de diretório e adicionando .php no final. Dessa forma, a classe `Zend_Controller_Front` será carregada do arquivo Zend/Controller/Front.php. Se você seguir a mesma convenção para as suas próprias bibliotecas de classes, então você poderá utilizar `Zend_Loader::loadClass()` para carregá-las também.
A primeira classe que nós precisamos é a front controller.

A front controller utilizar uma classe de roteamento que mapeia a URL requisitada para a função correta que será utilizada para mostrar a página.
Para que o roteador seja operado, ele precisa tirar qual parted a URL é o caminho para o nosso index.php e então ele poderá procurar pelos elementos da URI.
Isto é feito pelo objeto Request. Ele faz um bom trabalho de auto-detecção da URL base correta, mas caso não funcione para a sua configuração, então você pode sobrescrevê-lo usando a função `$frontController->setBaseUrl()`.

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 decide instruir o front controller para disparar todas as exceções que ocorrerem. Por padrão, o front controller irá capturá-los para nós e armazená-los na propriedade `_exceptions` do objeto “Response” que ele cria. O objeto response guarda toda a informação sobre a resposta para a URL requisitada. Isto inclui os cabeçalhos http, o conteúdo da página e as exceções. O front controller irá enviar os cabeçalhos e mostrar o conteúdo da página antes de ele completar o trabalho.
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.

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

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

Se você digitar http://localhost/zf_tutorial/ 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 and 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	
Title	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.
Adicionar 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/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 `atual`, `arquivadas` e `ler`.
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.

O Controller do Zend Framework reserve uma ação especial chamada `index` como uma ação padrão. Isto é, uma url como `http://localhost/zf-tutorial/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 principal	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}Controller`. Veja que `{Nome}` `precisa começar com uma letra maiúscula`. Esta classe precisa estar em um arquivo chamado `{Nome}Controller.php` dentro do diretório de controllers especificado. Novamente, `{Nome}` 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 `{ação}Action`. Neste caso, `{ação}` deve iniciar como uma letra minúscula.

Assim, nossa classe de controle é denominada `IndexController` que está definida em `zf-tutorial/application/controllers/IndexController.php`:

zf-tutorial/application/controllers/IndexController.php

```
<?php

class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
        echo "<p>em IndexController::indexAction()</p>";
    }

    function addAction()
    {
        echo "<p>em IndexController::addAction()</p>";
    }

    function editAction()
    {
        echo "<p>em IndexController::editAction()</p>";
    }

    function deleteAction()
    {
        echo "<p>em IndexController::deleteAction()</p>";
    }
}
```

Inicialmente, nós definimos que cada ação imprima seu próprio nome. Teste isso navegando pelas URLs:

URL	*Texto apresentado*
http://localhost/zf_tutorial/	em IndexController::indexAction()
http://localhost/zf_tutorial/index/add	em IndexController::addAction()
http://localhost/zf_tutorial/index/edit	em IndexController::editAction()
http://localhost/zf_tutorial/index/delete	em 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('/caminho/para/arquivos_de_visao');
echo $view->render('view.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 interessa 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 criar o Zend_Controller_Action para nós. Lá tem dois métodos auxiliares: `initView()` e `render()`. O método `initView()` cria um objeto Zend_View para nós e determina-o para a propriedade $view, deixando pronto para que sejam atribuídos dados a ele. Ele também configure o objeto Zend_View para procurar em views/scripts/{controller} pelos scripts de visão que devem ser renderizados. O processo de renderização de um script de visão é feito pelo `render()`, que vai (por padrão, ao menos) renderizar o script `{ação}.phtml` e anexará isso ao corpo do objeto Response. O Objeto Response pe usado para combinar todos os cabeçalhos, 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 inicializar a visão no método init() e então assegurar que chamaremos o método render() em cada ação. Nós precisamos também criar alguns arquivos de visão com código para testes de exibição.

A seguir, as mudanças necessárias no `IndexController` (mudanças em negrito).
Como você pode ver, nós adicionamos um novo método chamado `init()`, que é automaticamente executado pelo construtor do Zend_Controller_Action para nós. Isto nos garante que a visão foi inicializada no início e nós podemos ter confiança que ela está pronta para uso nos métodos de ação.

zf-tutorial/application/controllers/IndexController.php

```
<?php

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

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

    function addAction()
    {
        $this->view->title = "Adicionar novo álbum";
        $this->render();
    }

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

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

Em cada método, nós determinamos uma variável chamada title à propriedade view e então chamamos render() para mostrar a template view.
Repare que a exposição não ocorre neste ponto - Isto é feito pelo front controller no final do processo.

Agora nós precisamos adicionar quarto arquivos de visão para a nossa aplicação. Estes arquivos são conhecidos como templates e o método render() espera que cada arquivo de template tenha 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. Nós iremos colocar o código html que é obvio em dois arquivos: header.phtml e footer.phtml dentro do diretório scripts. Nós poderemos então utilizá-los para armazenar o código HTML “comum” e somente fazer uma referência a eles da template de visão.

Os novos arquivos são:

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">
```

(Repare que nós corrigimos o HTML, então nós estamos compatíveis com o padrão também!)

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

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

Novamente, nossas visões precisam de mudanças:

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'); ?>
```

Estilos

Mesmo que isso é 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 usamos o método getBaseURL() que é parte da requisição (request) e passamos isso para a visão (view). Isto provém para nós a pequena parte da URL que nós desconhecemos.

Altere o IndexController::init() para que se pareça com isso:

zf-tutorial/application/controllers/IndexController.php

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

Nós precisamos adicionar o arquivo CSS para a seção <head> do arquivo 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>
...
```

Finalmente, nós precisamos de alguns estilos CSS:

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;
}
```

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 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:

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
```

Obviamente você deve usar o seu nome de usuário, senha e banco de dados, não o meu!

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->db->config`.

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

Parte relevante de 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
...
```

As mudanças estão em negrito. Nós carregamos as classes que vamos utilizar (`Zend_Config_Ini` and `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 o método estático `Zend_Db_Table:: setDefaultAdapter()`. Novamente, nós fazemos isso no arquivo de inicialização (adições em negrito):

Parte relevante de 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,
                $config->db->config->asArray());
Zend_Db_Table::setDefaultAdapter($db);

// setup controller
...
```

Criando a tabela

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

```
CREATE TABLE album (
  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 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 'Hot 100' do site Amazon.co.uk:

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

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 `Álbum` como a nossa tabela que se chama `álbum`. 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 Álbum no diretório models:

zf-tutorial/application/models/Album.php

```
<?php

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

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, nós podemos ir para a parte a aplicação que mostra alguns álbuns. Isto sera feito na classe `IndexController`.

Claramente, toda ação em `IndexController` será manipulando o banco de dados `álbum` usando a classe `Álbum`, então faz sentido que a classe álbum seja carregada quando o controller seja instanciado. Isso pode ser feito pelo método `init()`:

zf-tutorial/application/controllers /IndexController.php

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

Nota: Este é um exemplo de como usar `Zend_Loader::loadClass()` para carregar nossas próprias classes e funciona porque nós colocamos o diretório `models` no include path em `index.php`.

Nós iremos listar os álbuns em uma tabela com `indexAction()`:

zf-tutorial/application/controllers/IndexController.php

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

O método `Zend_Db_Table::fetchAll()` retorna uma `Zend_Db_Table_Rowset` que nos permitirá iterar pelas linhas retornadas no arquivo de template da view:

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

http://localhost/zf-tutorial/ (or de onde você estiver chamando!) deverá mostrar agora uma bela lista de (dois) álbuns.

Adicionando novos álbuns

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

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

Isso é feito com `addAction()`:

zf-tutorial/application/controllers/IndexController.php

```
...
function addAction()
{
    $this->view->title = "Adicionar novo álbum";

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

```
            $filter = new Zend_Filter_StripTags();

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

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

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

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

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

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

Observe como nós checamos a variável `$_SERVER['REQUEST_METHOD']` para ver se o formulário foi submetido.. Se ele foi, nós recuperamos o artista e o titulo do array post e usamos a classe `Zend_Filter_StripTags` para garantir que nenhum html seja permitido. Então, assumindo que eles foram informados, nós utilizamos nossa classe de modelo, `Álbum()`, para inserir a informação em uma nova linha na tabela do banco de dados.

Depois que nós adicionamos o álbum, nós redirecionamos usando o método `_redirect()` do controle para voltarmos à raiz da nossa aplicação.

Finalmente, nós deixamos a view pronta para o formulário que vamos usar na template. Nós podemos ver que a ação edit do formulário será bastante similar com esta, então nós usaremos um arquivo de template comum (_form.phtml) que é chamado de indexAdd.tpl.php e indexEdit.tpl.php:

As templates para adicionar álbum são:

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>
```

Estes são códigos razoavelmente simples. Como nós intencionamos usar `_form.phtml` para a ação edit, nós usamos uma variável `$this->action`. Similarmente, nós usamos uma variável para o texto que será mostrado no botão de envio.

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 = "Editar Álbum";
    $album = new Album();

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

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

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

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

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

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

Note que quando não estamos em modo “post”, nós recuperamos o parâmetro id da propriedade `params` usando `getParam()`.

O Template é:

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'); ?>
```

Mudança!

Você não deve ter deixado de perceber que os métodos AddAction() e EditAction() são muito similares e que as templates add e edit são idênticas. Alguma mudança é necessária!

Eu deixarei isso como um exercício para você, prezado leitor...

Excluindo um álbum

Para arredondar a nossa aplicação, nós precisamos adicionar a exclusão. Nós temo um link Delete perto de cada álbum em nossa página de 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. O recente accelerator beta do Google trouxe este ponto a tona para muitas pessoas.

Nós mostraremos um formulário de confirmação quando o usuário clicar em excluir e se ele clicar em “sim”, nós faremos a exclusão.

O código parece um pouco familiar com as ações de adicionar e editar:

zf-tutorial/application/controllers/IndexController.php

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

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

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

        if ($del == 'Yes' && $id > 0) {
            $where = 'id = ' . $id;
            $rows_affected = $album->delete($where);
        }
    } else {
        $id = (int)$this->_request->getParam('id');
        if ($id > 0) {
            $this->view->album = $album->fetchRow('id='.$id);

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

    // redireciona à listagem novamente
    $this->_redirect('/');
}
...
```

Novamente, nós usamos o mesmo truque de checar o método de requisição para verificar se iremos mostrar o formulário de confirmação ou se nós devemos realizar a exclusão, via classe Álbum(). Como insert e update, a atual exclusão é feita via uma chamada a `Zend_Db_Table::delete()`.

Observe que nós retornamos imediatamente após setar o response. Isto é porque nós podemos redirecionar de voltar para a listagem de álbuns no final do método. Assim, se alguma das checagens falharem, nós voltamos à listagem de álbuns sem precisar chamar `_redirect()` diversas vezes dentro do método.

O Template é um simples formulário:

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

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

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/index.php

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

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

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.