Iniciando com o Zend Framework

Autor Rob Allen, www.akrabat.com
Tradução Gilberto Albino
Revisão do document 1.7.6

Este tutorial tem como objetivo oferecer uma introdução em como utilizar o Zend Framework através da criação de uma aplicação baseada em banco de dados utilizando o paradigma Model-View-Controller.

Nota: Este tutorial foi testado na versão 1.10.1 ate 1.11.4 do Zend Framework. Há uma chance muito boa de funcionar com versões posteriores da série 1.x, mas não irá funcionar com versões anteriores a 1.10.1.

Requerimentos

O Zend Framework necessita dos seguintes requerimentos:

- PHP 5.2.4 (ou superior)
- Um servidor web com suporte a mod_rewrite ou funcionalidade similar.

Pressupostos do Tutorial

Assumindo que você tem rodando o PHP 5.2.4 ou superior com o servidor web Apache, sua instalação do Apache ***obrigatóriamente deve ter a extensão mod_rewrite instalada e configurada***.

Você obrigatóriamente precisa se certificar que o Apache está configurado com suporte a arquivos .htaccess. Isto é feito normalmente alterando a configuração:

```
AllowOverride None
```

para

```
AllowOverride All
```

no seu arquivo `httpd.conf`. Para detalhes exatos verifique na documentação da distribuição de seu Apache. Você não conseguirá acessar qualquer outra página além da pagina inicial neste tutorial se não configurar corretamente o mod_rewrite e .htaccess.

Obtendo o framework

O Zend Framework pode ser baixado gratuitamente em http://framework.zend.com/download/latest nos formatos .zip ou .tar.gz. Olhe abaixo na pagina para os links diretos. Você precisa da versão “Minimal”.

Configurando a Zend_Tool

O Zend Framework é fornecido com uma nova ferramenta de linha de commando. Vamos começar configurando-a.

Zend_Tool para Windows

- Crie um novo diretório em `Arquivos de Programas` chamado `ZendFrameworkCli`
- Dê um duplo clique no arquivo baixado, `ZendFramework-1.10.6-minimal.zip`.
- Copie as pastas `bin` e `library` da janela da pasta `ZendFramework-1.10.6-minimal.zip` para a pasta `C:\Arquivos de Programas\ZendFrameworkCli`. Esta pasta deve agora conter duas sub-pastas: `bin` e `library`.
- Adicione o diretório `bin` para o seu Path do sistema:
  - Vá para a seção “Sistema” do Painel de Controle.
  - Escolha Configurações Avançadas e então pressione o botão “Variáveis de Ambiente”.
  - Em na lista “Variáveis do Sistema”, encontre a variável `Path` e dê um duplo clique.

- Adicione `;C:\Arquivos de Programas\ZendFrameworkCli\bin` no final da campo de entrada e pressione o botão OK. (O ponto-e-vírgula inicial é obrigatório!)
- Reinicie.

Zend_Tool para OS X (Linux é similiar)

- Extraia o arquivo `ZendFramework-1.10.6-minimal.zip` em seu diretório `Downloads` fazendo duplo clique nele.
- Copie para `/usr/local/ZendFrameworkCli` abrindo o Terminal e digitando:
- `sudo cp -r ~/Downloads/ZendFramework-1.10.6-minimal /usr/local/ZendFrameworkCli`
- Edite seu bash_profile para disponibiliar um alias:
  - A partir do Terminal, digite: `open ~/.bash_profile`
  - Adicione `alias zf=/usr/local/ZendFrameworkCli/bin/zf.sh` para o final do arquivo
  - Salve e saia do TextEdit.
  - Saia do Terminal.

Testando o Zend_Tool

Você pode testar sua instalação da interface de linha de comando do Zend_Tool abrindo o Terminal ou Prompt de Comando e digitando:

```
zf show version
```

Se tudo deu certo, você deverá ver:

```
Zend Framework Version: 1.10.0
```

Se não, verifique o caminho de sua instalação está correto e se o diretório bin existe no diretório ZendFrameworkCli. Uma vez que a ferramenta `zf` esteja funcionando, `zf --help` lhe mostrará todos os commandos disponíveis.

Nota: Se sua distribuição do PHP vem com Zend Framework, por favor, verifique se ela não está utilizando o ZF 1.9, pois este tutorial não vai funcionar. No momento da criação deste tutorial, a distribuição do xampp trazia.

A aplicação do tutorial

Agora que todas as peças estão no lugar que precisamos para construir uma aplicação com Zend Framework, vamos analisar a aplicação que iremos construir. Vamos construir um sistema simples de controle de álbuns para exibir nossa coleção de CD's. A página principal irá lista nossa coleção e permitir que possamos Adicionar, Editar e Excluir CD's. Assim como em qualquer engenharia de software, é bom se for feito um pequeno planejamento antecipado. Vamos precisar de quatro páginas para nosso site:

Página Inicial	Esta página irá exibir a lista de álbuns e oferecer links para editá-los ou excluí-los. Também um link para adicionar um novo álbum será oferecido.
Adicionar Novo Álbum	Esta página fornecerá um formulário para adicionar um novo álbum
Editar Álbum	Esta página fornecerá um formulário para editar um
Deletar Álbum	Esta página confirmará se queremos excluir um álbum e então excluí-lo.

Nós também precisaremos armazenar nossos dados dentro de um banco de dados. Somente precisaremos de uma tabela com estes campos dentro:

Nome do Campo	Tipo	Null?	Notas
id	integer	No	Primary key, auto increment
artist	varchar(100)	No	
title	varchar(100)	No	

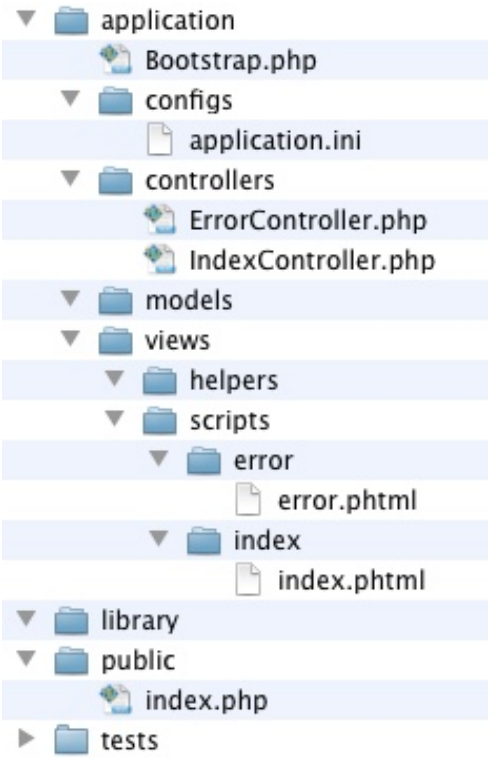
Tirando nossa aplicação do papel

Vamos começar a construir nossa aplicação. Onde for possível, vamos utilizar a linha de comando da ferramenta `zf` uma vez que ela economiza tempo e esforço. O primeiro trabalho é criar o esqueleto de arquivos e diretórios.

Abra o Terminal ou Prompt de Comando e modifique o diretório atual para o diretório raiz de seu servidor web utilizando o comando `cd`. Certifique se você possui permissões para criar arquivos neste diretório e que seu servidor tem permissões de leitura. Digite:

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

A ferramenta ZF criará um diretório chamado `zf-tutorial` e preencherá com a estrutura de diretório recomendada. Esta estrutura assume que você tenha total controle sobre sua configuração do Apache, com a finalidade de manter a maior parte dos arquivos fora do diretório raiz do site. Você deverá ver os seguintes arquivos e diretórios:



(Há também um arquivo oculto `.htaccess` dentro de `public/`).

O diretório `application/` é onde o código-fonte do website reside. Como você pode ver, separamos os diretórios para os arquivos model, view e controller de nossa aplicação. O diretório `public/` é o diretório de alcance publico, o que significa que o URL para chegar até a aplicação será `http://localhost/zf-tutorial/public/`. Isto é feito para que a maior parte dos arquivos da aplicação não sejam acessados diretamente pelo Apache e, portanto, estão mais seguros.

Nota:

Em um site online, você precisa criar um virtual host para o website e configurar o diretório raiz diretamente para o diretório `public`. Por exemplo, você pode criar um virtual host chamado zf-tutorial.localhost, que deve se parecer com isto:

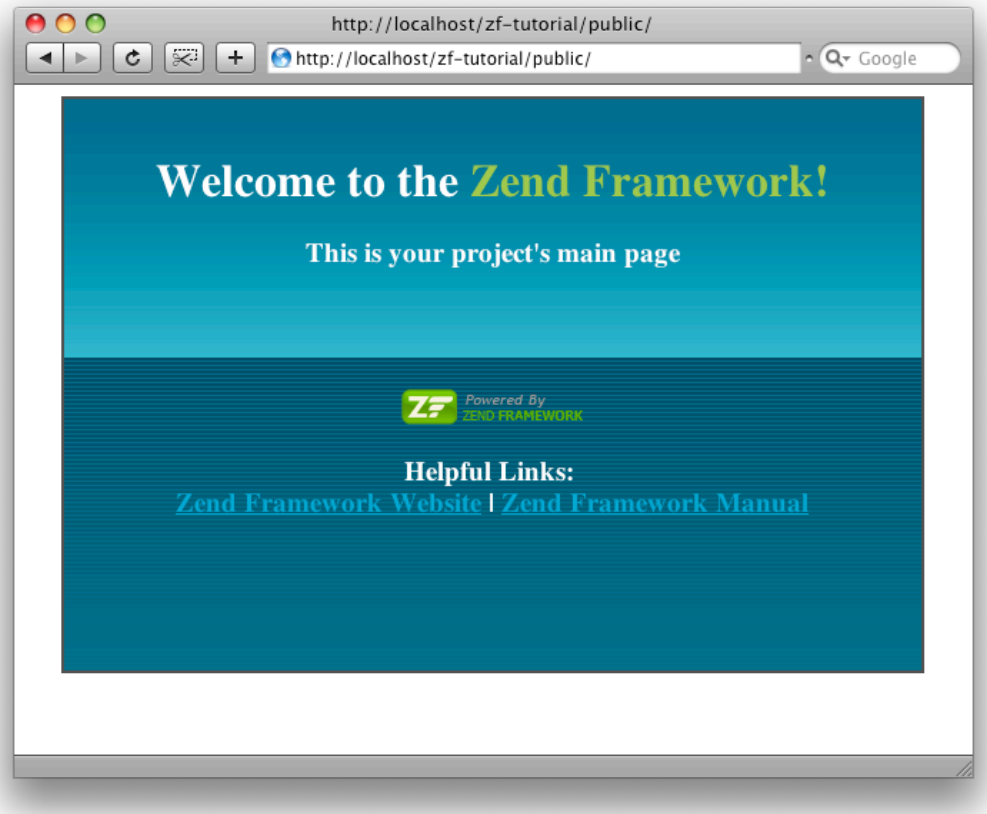
```
<VirtualHost *:80>
    ServerName zf-tutorial.localhost
    DocumentRoot /var/www/html/zf-tutorial/public
    <Directory "/var/www/html/zf-tutorial/public">
        AllowOverride All
    </Directory>
</VirtualHost>
```

O site então seria acessado utilizando `http://zf-tutorial.localhost/` (certifique-se de que você atualizou seu arquivo /etc/hosts ou o arquivo c:\windows\system32\drivers\etc\hosts file para que zf-tutorial.localhost seja mapeado para 127.0.0.1). Não faremos isto neste tutorial, apesar de que isto é tão simples quanto utilizar um subdiretório para testes.

Os arquivos de imagens, JavaScript e CSS auxiliares são armazenados em um diretório separado dentro do diretório `public/`. Os arquivos do Zend Framework baixados devem ser colocados dentro do diretório `library/`. Se precisarmos utilizar quaisquer outras bibliotecas, elas também podem ser colocas ai.

Copie o diretório `library/Zend/` do arquivo baixado (`ZendFramework-1.10.6-minimal.zip`) para dentro de seu diretório `zf-tutorial/library/`, para que seu diretório `zf-tutorial/library/` contenha um subdiretório chamado `Zend/`.

Você pode testar se tudo está certo acessando http://localhost/zf-tutorial/public. Você deve ver algo parecido com isto:



Por trás do Processo de Iniciação (Bootstrap)

O controlador do Zend Framework's utiliza o Padrão de Software "Front Controller" e roteia todas as solicitações através de um único arquivo `index.php`. Isto garante que o ambiente está configurado corretamente para rodar a aplicação (conhecido como bootstrapping). Nós obtemos isto utilizando um arquivo `.htaccess` dentro do diretório `zf-tutorial/public` que é gerado por pelo `Zend_Tool` no qual redireciona todas as solicitações para `public/index.php` que também é pelo `Zend_Tool`.

O arquivo `index.php` é o ponto de entrada para nossa aplicação e é utilizado para criar uma instância de `Zend_Application` para inicializar nossa aplicação e então executá-la. Este arquivo define duas constantes: `APPLICATION_PATH` e `APPLICATION_ENV` que definem o caminho para o diretório `application/` e o modo do ambiente da aplicação. O padrão é definido como `production` no `index.php`, mas você deve configurá-lo para `development` no arquivo `.htaccess` adicionando esta linha:

```
SetEnv APPLICATION_ENV development
```

O component `Zend_Application` é utilizado para iniciar a aplicação e está configurado para utilizar as diretivas de configuração do arquivo `application/configs/application.ini`. Este arquivo também é gerado automaticamente pelo zf.

Uma classe Bootstrap que extende `Zend_Application_Bootstrap_Bootstrap` é fornecida no arquivo `application/Bootstrap.php` que pode ser utilizada para executar qualquer código de iniciação específico.

O `application.ini`, que está armazenado no diretório `application/configs` é carregado utilizando o component `Zend_Config_In`. O `Zend_Config_Ini` entende o conceito de herança das seções que são definidas utilizando um ponto-duplo no nome da seção.
Por exemplo:

```
[staging : production]
```

Isto significa que a seção `staging` herda todas as configurações da seção `production`. A constante `APPLICATION_ENV` define qual seção é carregada. Obviamente, no meio do desenvolvimento, a seção `development` é melhor e quando estiver no servidor remoto, a seção `production` deve ser utilizada. Colocaremos todas as modificações que forem feitas no arquivo `application.ini` dentro da seção `production` para que todas as configurações carreguem as alterações que fizermos.

Editando o arquivo application.ini

A primeira modificação que precisamos fazer é adicionar nossa informação de fuso-horário para as funcionalidades de data e hora do PHP. Edite o arquivo `application/configs/application.ini` e adicione a sua região:

```
phpSettings.date.timezone = "Europe/London"
```

logo após todos os outros valores `phpSettings` na seção `[production]`. Obviamente, você deve utilizar seu próprio fuso-horário. Estamos em uma posição agora para adicionar o código específico para a aplicação.

Código Específico da Aplicação

Antes de configurarmos nossos arquivos, é importante entender como o Zend Framework espera que as páginas sejam organizadas. Cada página da aplicação é conhecida como um **action**(ação) e ações são agrupados dentro de **controllers**(controlador). Para o formato de URL `http://localhost/zf-tutorial/public/news/view`, o controlador é `News` e a ação é `view`. Assim é permitido agrupar ações relacionadas. Por exemplo, um controlador `News` poderia ter ações como `listar`, `arquivos` e `ver`. O sistema MVC do Zend Framework também suporta módulos para agrupamento de controladores, mas esta aplicação não é grande o suficiente para se preocupar com eles!

Por padrão, o controlador do Zend Framework reserva uma ação especial chamada `index` como padrão. Isto é para casos como `http://localhost/zf-tutorial/public/news/` onde a ação `index` dentro do controlador `News` será executada. Também há um nome para o controlador padrão, na qual também é chamado de `index` e, portanto, o URL `http://localhost/zf-tutorial/public/` faz com que a ação `index` no controlador `Index` seja executada.

Como este é um tutorial simples, não iremos nos preocupar com coisas “complicadas” como autenticação de usuário. Isto pode esperar por um tutorial separado (ou você pode ser sobre isto no livro *Zend Framework in Action*!)

Como quatro páginas se aplicam aos álbuns, vamos agrupá-las em um único controlador com quatro ações. Vamos usar o controlador padrão e as quatro ações serão:

Page	Controller	Action
Home page	Index	index
Adicionar novo álbum	Index	add
Editar album	Index	edit
Deletar album	Index	delete

Na medida em que o site fica mais complicado, controladores adicionais são necessários e você poderá até mesmo agrupar controladores dentro de módulos, se necessário.

Configurando o Controlador

Agora estamos prontos para configurar nosso controlador. No Zend Framework, o controlador é uma classe que deve ser chamada assim`{Controller nome}Controller`. Note que `{Controller nome}` deve começar com uma letra maiúscula. Esta classe deve estar dentro de um arquivo chamado `{Controller nome}Controller.php` dentro do diretório `application/controllers`. Cada ação é um método public dentro da classe controladora que deve ser nominada `{action nome}Action`. Neste caso, `{action nome}` começa com uma letra minúscula e novamente deve estar completamente em minúscula. Nomes de controladores e ações mistos são permitidos, mas possuem regras especiais que você deve entender antes de começar a utilizá-los. Confira a documentação primeiro!

Nossa classe controladora é chamada `IndexController` que está definida em `application/controllers/IndexController.php` e foi automaticamente criada através do Zend_Tool. Ela também contém nosso primeiro método, `indexAction()`. Nós apenas precisamos adicionar nossas ações adicionais.

A adição de ações adicionais aos controladores é feita utilizando a ferramenta de linha de comando `zf` com o comando `create action`. Abra o Terminal ou Prompt de Comando e mude para o diretório para o diretório `zf-tutorial/`. Então digite estes três comandos:

```
zf create action add Index
zf create action edit Index
zf create action delete Index
```

Estes commandos criam três novos métodos: `addAction`, `editAction` e `deleteAction` em `IndexController` e também cria os scripts apropriados para o View que iremos precisar mais adiante. Agora temos todas as quarto ações que precisamos utilizar.

O URL para cada ação é:

URL	Action method
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()`

Você pode testar as três novas ações e deve ver uma mensagem como esta:

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

Nota: Se você receber um erro 404, então você não configurou o Apache com mod_rewrite ou não configurou corretamente o AllowOverride dentro de seu arquivo httpd.conf para que o arquivo `.htaccess` dentro da pasta `public/` esteja sendo usado.

O Banco de Dados

Agora que nós temos o esqueleto de nossa aplicação com métodos para as ações do controlador e os arquivos para o View prontos, é hora de olhar para a seção de **Models**(modelos) de nossa aplicação. Lembre-se que o Model é a parte que lida com o objetivo central da aplicação. (as tão famosas "regras de negócio") e, em nosso caso, lida com o banco de dados. Nós utilizaremos a class `Zend_Db_Table` do Zend Framework que serve para encontrar, inserir, atualizar e deletar linhas de uma tabela no banco de dados.

Configuração do Banco de Dados

Para utilizar `Zend_Db_Table`, precisamos informar qual banco de dados utilizar com um usuário e senha. Como nós preferimos não informar esta informação todas as vezes que precisarmos, vamos utilizar um arquivo de configuração para armazenar esta informação. O Componente `Zend_Application` é passado dentro de um recurso de configuração do banco de dados, por isto somente precisamos configurar a informação apropriada no arquivo `configs/application.ini` e ele fará o resto.

Abra `application/configs/application.ini` e adicione o seguinte no final da seção `[production]` (ou seja, acima da seção `[staging]`):

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

Obviamente, você deve utilizar seu usuário, senha e banco, não os meus! A conexão com o banco de dados será agora feita automaticamente e o adaptador padrão `Zend_Db_Table`'s será configurado. Você pode ler mais sobre outros plug-ins de recursos disponíveis em: http://framework.zend.com/manual/en/zend.application.available-resources.html.

Criar a tabela no banco de dados

Como citado no planejamento inicial, iremos utilizar um banco de dados para armazenar os dados de nossos álbuns. Vou utilizar MySQL. O código SQL para criar a tabela é o seguinte:

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

Execute este código em algum cliente MySQL, tal como phpMyAdmin ou o cliente de linha de comando padrão do MySQL.

Inserir os dados teste

Também iremos inserir algumas linhas dentro da tabela para que possamos utilizar a funcionalidade de extração de informação da página inicial. Eu vou pegar os primeiros CD “Bestsellers” do site do Amazon UK. Execute o código a seguir em seu cliente MySQL:

```
INSERT INTO albums (artist, title)
VALUES
('Paolo Nutine', 'Sunny Side Up'),
('Florence + The Machine', 'Lungs'),
('Massive Attack', 'Heligoland'),
('Andre Rieu', 'Forever Vienna'),
('Sade', 'Soldier of Love');
```

Agora temos alguns dados dentro do banco de dados e podemos escrever um modelo bem simples para eles.

O Modelo (Model)

O Zend Framework não oferece uma classe `Zend_Model` como Modelo uma vez que o modelo é sua lógica de negócio da aplicação e só depende de você como você deseja que ele funcione. Existem vários componentes que você pode utilizar para isto, dependendo de suas necessidades. Uma abordagem é ter classes modelo que representem cada entidade em sua aplicação e então utilizar objetos mapeados (mappers) que carregam e salvam as entidades no banco de dados. Esta abordagem está documentada no site do Zend Framework no QuickStart em: http://framework.zend.com/manual/en/learning.quickstart.create-model.html.

Para este tutorial, vamos criar um modelo que extende `Zend_Db_Table` e utiliza `Zend_Db_Table_Row`. O Zend Framework oferece o `Zend_Db_Table` que implementa o Padrão de Software “Table Data Gateway” que permite criar uma interface com os dados de uma tabela no banco de dados. No entanto, esteja consciente que o Padrão “Table Data Gateway” pode se tornar limitado em grandes sistemas. Há também a tentação de colocar o código de acesso ao banco de dados dentro do método de uma ação, uma vez que isto é possível com Zend_Db_Table.

`Zend_Db_Table_Abstract` é uma classe abstrata, do qual derivaremos nossa classe que é especifica para gerenciamento dos álbuns. Não importa o nome que daremos à nossa classe, mas faz sentido que leve o nome da tabela do banco de dados. Nosso projeto possui uma carregador automático (autoloader) padrão instanciado por `Zend_Application` que mapeia as classes recurso dentro de um módulo para o diretório onde ela está definida. Para as pastas `application/` utilizamos o prefixo `Application_`.

O autoloader mapeia os recursos nos diretórios utilizando este mapeamento:

Prefix	Directory
Form	forms
Model	models
Model_DbTable	models/DbTable
Model_Mapper	models/mappers
Plugin	plugins
Service	services
View_Filter	views/filters
View_Helper	views/helpers

Como estamos chamando pelo nome da tabela do bando de dados “album” e nossa classe utilizará `Zend_Db_Table` então nossa classe sera chamada `Application_Model_DbTable_Albums` que será armazenada em `applications/models/DbTable/Albums.php`.

Para informar ao `Zend_Db_Table` o nome da tabela que ele estará gerenciando, devemos definir a propriedade `$_name` com o nome da tabela. Inclusive, `Zend_Db_Table` assume que sua tabela possui uma chave primária chamada `id` que é auto incrementada pelo banco de dados. O nome deste campo pode ser mudado também se for preciso.

Podemos utilizar a ferramenta de linha de comando zf para fazer parte do trabalho, para isto, execute o comando a seguir na linha de comando:

```
zf create db-table Albums albums
```

A ferramenta agora criou o arquivo `Albums.php` dentro da pasta `application/models/DbTable`. Dentro deste arquivo está uma classe chamada `Application_Model_DbTable_Albums` e dentro da mesma está definido o nome da tabela do banco de dados do qual esta classe se comunicará.

Agora nós precisamos adicionar alguma funcionalidade, por isto edite application/models/DbTable/Albums.php e acione os métodos `getAlbum()`, `addAlbum()`, `updateAlbum()` e `deleteAlbum()` e agora ele deve parecer com o código abaixo:

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

```
<?php

class Application_Model_DbTable_Albums extends Zend_Db_Table_Abstract
{
    protected $_name = 'albums';

    public function getAlbum($id)
    {
        $id = (int)$id;
        $row = $this->fetchRow('id = ' . $id);
        if (!$row) {
            throw new Exception("Could not find row $id");
        }
        return $row->toArray();
    }

    public function addAlbum($artist, $title)
    {
        $data = array(
            'artist' => $artist,
            'title' => $title,
        );
        $this->insert($data);
    }

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

    public function deleteAlbum($id)
    {
        $this->delete('id =' . (int)$id);
    }
```

```
}
```

Nós criamos quarto métodos de auxílio que nossa aplicação utilizará para se conectar com a tabela do banco de dados. `getAlbum()` retorna um array com uma linha única, `addAlbum()` cria um novo registro no banco de dados, `updateAlbum()` atualiza um álbum e `deleteAlbum()` remove um registro complemente. O código para cada um destes métodos é auto explicativo. Apesar de não ser necessário neste tutorial, você pode dizer ao `Zend_Db_Table` sobre tabelas relacionadas e ele pode buscar os dados relacionados também.
Precisamos preencher os controladores com os dados a partir do model e obter o scripts para os views a fim de exibí-los, entretanto, antes que façamos isto, precisamos entender como o sistema de views do Zend Framework funciona.

Layouts e views

O componente view do Zend Framework é chamado de, algo não surpreendente, `Zend_View`. O componente view permitirá separação do código que exibe as páginas do código dentro dos métodos das ações.

O uso básico de `Zend_View` é:

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

Pode-se facilmente ver que se colocássemos este código diretamente dentro de nossos métodos, nós estaríamos repetindo um código 'estrutural', muito chato, que não é de interesse para a ação. Ao invés disto, devemos realizar a inicialização do view em qualquer outro lugar e então acessar nosso objeto view, já inicializado dentro de cada método para as ações. O Zend Framework oferece uma auxiliador (helper) para Actions chamado de `ViewRenderer`. Ele toma conta de inicializar a propriedade view no controlador (`$this->view`) para que possamos utilizar e também exibirá um script view depois que a ação for disparada.

Para a exibição, o `ViewRenderer` define o objeto `Zend_View` para procurar em views/scripts/{controller name} pelos scripts a serem exibidos e exibirá (por padrão, pelo menos) o script que é chamado pelo nome da ação e terá a extensão `phtml`. Ou seja, o script view exibido será `views/scripts/{controller nome}/{action_nome}.phtml` e os conteúdos apresentados são anexados ao corpo do objeto Response. O Objeto Response é utilizado para combinar todos os cabeçalhos HTTP, corpo do conteúdo e exceções gerados como resultado da utilização do sistema MVC. O front controller então automaticamente envia os cabeçalhos seguidos pelo corpo do conteúdo no final do envio.

Isto tudo é definido para nós através do `Zend_Tool` quando criamos o projeto e adicionamos controladores e ações utilizando os comandos `zf create controller` e `zf create action`.

Código HTML em comum: Layouts

Rapidamente fica bem óbvio que existe um monte de códigos HTML em comum em nossos views, pelo menos para o cabeçalho e rodapé e talvez para uma ou duas barras laterais também. Este é um problema muito comum e o componente `Zend_Layout` foi projetado para resolver este problema. `Zend_Layout` permite-nos mover todo código em comum para o cabeçalho, rodapé e outros, para um script view na qual inclui o código especifico para a ação em execução.

O local padrão para manter nossos layouts é em `application/layouts/` e há um recurso disponível para `Zend_Application` que configurará `Zend_Layout` para nós. Utilizamos o `Zend_Tool` para criar o script do view para o layout e atualizar apropriadamente o arquivo application.ini. Novamente, abra o Terminal ou Prompt de Comando e dentro do seu diretório `zf-tutorial` digite:

```
zf enable layout
```

O Zend_Tool agora criou a pasta application/layouts/scripts e colocou um script view layout.phtml dentro dela. Ele também atualizou o arquivo application.ini e adicionou a linha resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/" na seção [production].

No final do ciclo de despacho, depois que os métodos de ações terminaram, Zend_Layout exibirá nosso layout. O Zend_Tool oferece um arquivo de layout básico que apenas exibe o conteúdo do script do view da ação. Vamos estender este arquivo com o HTML necessário para nosso website. Abra layouts.phtml e substitua o código dentro dele por:

zf-tutorial/application/layouts/scripts/layout.phtml

```
<?php
$this->headMeta()->appendHttpEquiv('Content-Type', 'text/html;charset=utf-8');
$this->headTitle()->setSeparator(' - ');
$this->headTitle('Zend Framework Tutorial');

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

O arquivo layout contém o código HTML “externo” na qual é bem padrão. Como este é um arquivo PHP normal, podemos utilizar PHP dentro dele. Há uma variável disponível, $this, que é uma instância do objeto view que foi criado durante o processo de inicialização. Nós podemos utilizar a mesma para recuperar dados que foram atribuídos para o view e também para chamar métodos. Os métodos (conhecidos como view helpers) retornam um string que podemos então imprimir.

Primeiramente, nós configuramos alguns auxiliadores para a seção do cabeçalho da página e então imprimimos o tipo correto de doctypte. Dentro de <body>, criamos um DIV com um <h1> contendo o título. Para obter o script do view para a ação atual exibir, imprimimos o conteúdo reservado utilizando o auxiliador do view layout(): echo $this->layout()->content; que faz o trabalho para nós. Isto quer dizer que os scripts do view para ação são executadas antes do script do view do layout.

Precisamos definir o doctype para a página antes de exibirmos quaisquer scripts dos views. Como os scripts do view das ações são exibidos antes, eles podem precisar conhecer qual doctype está sendo usado. Isto é especialmente importante para o Zend_Form.

Para definir o doctype, adicionamos uma outra linha em nosso arquivo application.ini, na seção [production]:

```
resources.view.doctype = "XHTML1_STRICT"
```

O auxiliador do view doctype() exibirá o doctype correto para componentes como Zend_Form e gerará o HTML compatível.

Estilizando

Apesar de este ser “apenas” um tutorial, precisamos de um arquivo CSS para fazer com que nossa aplicação pareça um pouco mais apresentável! Isto causa um probleminha uma vez que nós não sabemos como referenciar o arquivo CSS porque o URL não aponta para o diretório raiz correto. Felizmente, um auxiliador chamado baseUrl() está disponível para o View. Este auxiliador colecta as informações

requeridas a partir do objeto de solicitação e nos oferece uma parte do URL que não conhecemos. Podemos então adicionar o arquivo CSS a seção `<head>` do arquivo `application/layouts/scripts/layout.phtml` e mais uma vez utilizamos um auxiliador para o view, `headLink()`:

zf-tutorial/application/layouts/scripts/layout.phtml

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

Utilizando o método `prependStylesheet()` de `headLink()`, nos é permitido que arquivos adicionais mais específicos sejam adicionados dentro do view do controlador que será exibido dentro da seção depois de `site.css`.

Por fim, precisamos alguns estilos CSS, portanto, crie um diretório `css` dentro de `public/` e adicione ao `site.css` o código a seguir:

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 deverá fazer com que pareça um pouco mais bonito, mas como você pode ver, eu não sou um designer!

Podemos agora limpar os quatro scripts para ações que foram geradas automaticamente. Então, vá em frente e esvazie os arquivos `index.phtml`, `add.phtml`, `edit.phtml` e `delete.phtml` que, você sem dúvidas se lembrará, estão no diretório `application/views/scripts/index`.

Listando álbuns

Agora que definimos as configurações, informações de banco de dados e o esqueleto de nossos views, podemos ir a fundo em nossa aplicação e exibir alguns álbuns. Isto é feito na classe `IndexController` e começaremos listando alguns álbuns em uma tabela dentro do método `indexAction()`:

zf-tutorial/application/controllers/IndexController.php

```
    ...
    function indexAction()
    {
        $albums = new Application_Model_DbTable_Albums();
        $this->view->albums = $albums->fetchAll();
    }
    ...
```

Criamos uma instância de nosso modelo baseado no padrão Table Data Gateway. O método `fetchAll()` retorna um `Zend_Db_Table_Rowset` que nos permite iterar sobre as linhas retornadas dentro do arquivo do view da ação.

Podemos agora preencher o script do view associado, `index.phtml`:

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

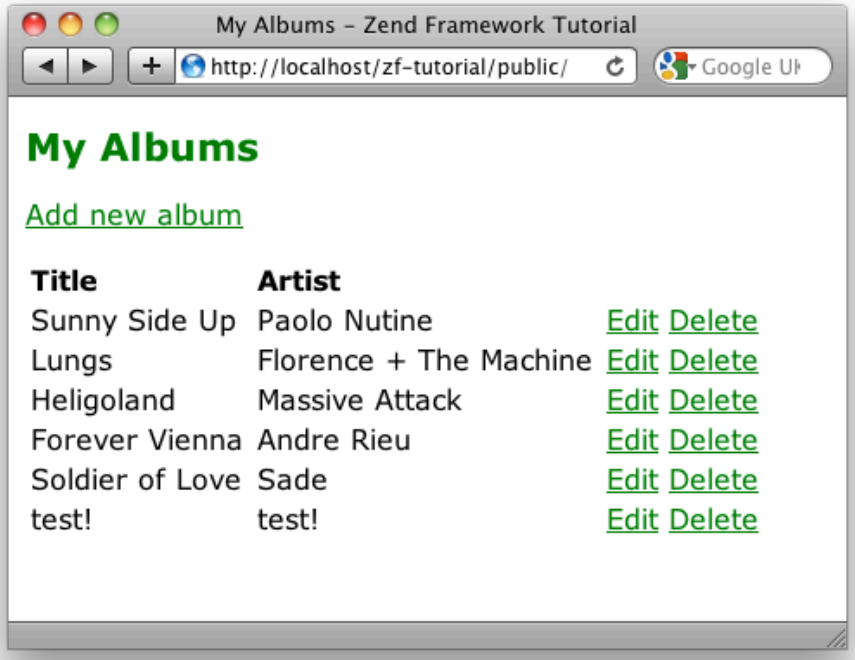
```
<?php
$this->title = "My Albums";
$this->headTitle($this->title);
?>
<p><a href="<?php echo $this->url(array('controller'=>'index',
        'action'=>'add'));?>">Add new album</a></p>
<table>
<tr>
    <th>Title</th>
    <th>Artist</th>
    <th> </th>
</tr>
<?php foreach($this->albums as $album) : ?>
<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 fazemos é definir o título da página (utilizado no layout) e também definir o título para a seção `<head>` utilizando o auxiliador para view `headTitle()`, que exibirá a barra de títulos no navegador. Então, criamos um hyperlink para adicionar um novo álbum. O auxiliador para `url()` para view, vem junto com o framework e, de grande ajuda, cria os links incluindo o URL base correto. Nós apenas atribuímos os parâmetros que precisamos dentro de um array e ele executará o resto.

Criamos então uma tabela html para exibir cada título e artista do álbum e disponibilizamos um link para que possamos editar e deletar um registro. Um laço padrão `foreach` é utilizado para iterar na lista de álbuns, e

utilizamos a forma alternativa com os dois pontos e `endforeach;` já que assim é mais fácil de escanear do que tentar igualar as chaves. Novamente, o auxiliar `url()` é utilizado para criar os links edit e delete.

Se você navegar em http://localhost/zf-tutorial/public/ (ou qualquer lugar que você esteja realizando o tutorial!) então você deve ver uma bela lista de álbuns, algo parecido com:



Adicionando novos álbuns

Agora nós podemos codificar a funcionalidade para adicionar novos álbuns. Existem duas coisas nesta parte:

- Exibir um formulário para o usuário informar os detalhes
- Processar o envio do formulário e armazenar no banco de dados

Utilizaremos `Zend_Form` para realizar esta tarefa. O componente `Zend_Form` nos permite criar umformulário e validar a entrada e dados. Criamos uma nova classe `Form_Album` que é extendida a partir de `Zend_Form` para definir o nosso formulário. Como esta é um recurso da aplicação, a classe é armazenada no arquivo `Album.php` dentro do diretório `forms`. Vamos utilizar o script da linha de comando do `zf` para criar o arquivo certo:

```
zf create form Album
```

Isto cria o arquivo `Album.php` dentro de `application/forms` e inclui um método `init()` onde poderemos definir o formulário e adicionar os elementos que precisamos. Edite o arquivo `Aplication/forms/Album.php` e remova o comentário dentro do método `init()` e adicione o código a seguir:

zf-tutorial/application/forms/Album.php

```
<?php

class Application_Form_Album extends Zend_Form
{

    public function init()
    {
```

```
        $this->setName('album');

        $id = new Zend_Form_Element_Hidden('id');
        $id->addFilter('Int');

        $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 método `init()` de `Application_Form_Album`, criamos quatro elementos para o formulário, respectivamente, para o id, artist, title, e botão submit. Para cada item definimos vários atributos, incluindo o texto a ser exibido para o usuário. Para o id, queremos nos certificar que este é somente do tipo inteiro, a fim de evitar problemas com injeção SQL. O filtro `Int` fará isto para nós.

Para os proxies elementos de texto, adicionamos dois filtros `StripTags` and `StringTrim` para remover HTML indesejado e espaço em branco desnecessário. Também definimos como obrigatórios e adicionamos uma validação `NotEmpty` para garantir que o usuário realmente inseriu a informação que desejamos. (o validador `NotEmpty` não é tecnicamente obrigatório, uma vez que sera automaticamente adicionado pelo sistema, pois `setRequired()` foi marcado como `true`; ele está aqui como uma demonstração de como adicionar um validador.)

Agora precisamos obter o formulário para exibir e processá-lo no envio. Isto é feito dentro do método `addAction()` do controlador `IndexController`:

zf-tutorial/application/controllers/IndexController.php

```
    ...
    function addAction()
    {
        $form = new Application_Form_Album();
        $form->submit->setLabel('Add');
        $this->view->form = $form;

        if ($this->getRequest()->isPost()) {
            $formData = $this->getRequest()->getPost();
            if ($form->isValid($formData)) {
                $artist = $form->getValue('artist');
                $title = $form->getValue('title');
                $albums = new Application_Model_DbTable_Albums();
                $albums->addAlbum($artist, $title);

                $this->_helper->redirector('index');
            } else {
                $form->populate($formData);
            }
        }
```

```
    }
    ...
```

Vamos examinar com um pouco mais de detalhes:

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

Instanciamos nosso `Form_Album`, definimos o texto para o botão submit como “Add” e então atribuímos o view para exibição.

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

Se o método `isPost()` do objeto solicitado for `true`, então o formulário foi enviado e podemos então obter os dados do formulário a partir da solicação utilizando `getPost()` e verificando se o mesmo é valido utilizando o método `isValid()`.

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

Se o formulário for válido, então instanciamos a classe modelo `Application_Model_DbTable_Albums` e utilizamos o método `addAlbum()` que criamos antes para criar um novo registro no banco de dados.

```
            $this->_helper->redirector('index');
```

Depois de salvo o registro de um novo album, redirecionamos o auxiliador para ação `Redirector` para retornar à ação `index` (ou seja, voltamos para a página inicial).

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

Se os dados do formulário não forem válidos, então preenchemos o formulário com os dados que usuário inseriu e exibimos novamente.

Agora podemos exibir o formulário no script do view dentro de `add.phtml`:

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

```
<?php
$this->title = "Add new album";
$this->headTitle($this->title);
echo $this->form ;
?>
```

Como você pod ever, exibir o formulário é muito simples, - apenas imprimimos ele, sendo que o formulário sabe como exibir a si mesmo. Você agora já pode utilizar o link “Add new album” na página inicial da aplicação e adicionar um novo álbum no registro.

Editando um álbum

Editar um álbum é quase idêntico a adicionar um, por isto o código é muito parecido:

zf-tutorial/application/controllers/IndexController.php

```
...
function editAction()
{
    $form = new Application_Form_Album();
    $form->submit->setLabel('Save');
    $this->view->form = $form;

    if ($this->getRequest()->isPost()) {
        $formData = $this->getRequest()->getPost();
        if ($form->isValid($formData)) {
            $id = (int)$form->getValue('id');
            $artist = $form->getValue('artist');
            $title = $form->getValue('title');
            $albums = new Application_Model_DbTable_Albums();
            $albums->updateAlbum($id, $artist, $title);

            $this->_helper->redirector('index');
        } else {
            $form->populate($formData);
        }
    } else {
        $id = $this->_getParam('id', 0);
        if ($id > 0) {
            $albums = new Application_Model_DbTable_Albums();
            $form->populate($albums->getAlbum($id));
        }
    }
}
...
```

Vamos analisar as diferenças comparando com a adição de um álbum. Primeiro, quando exibimos o formulário para o usuário precisamos obter o artista e título a partir do banco de dados e preencher os elementos do formulário com eles. Isto está no final do método:

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

Observe que isto é feito se a solicitação não for um POST, já que um POST implicaria em já termos preenchido o formulário e o processado. Para a exibição inicial do formulário, obtemos o id a partir da solicitação utilizando o método `_getParam()`. Então, utilizamos o modelo para obter a linha do registro no banco de dados e preencher o formulário diretamente com os dados do mesmo. (Agora você sabe porque o método `getAlbum()` retornou um array!)

Após validar o formulário precisamos salvar os dados novamente no banco de dados na linha correta. Isto é feito utilizando o método `updateAlbum()` em nosso modelo:

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

O template para o view é o mesmo utilizado para o add.phtml:

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

```
<?php
$this->title = "Edit album";
$this->headTitle($this->title);
```

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

Você agora pode editar álbuns.

Excluindo um álbum

Para deixar nossa aplicação redondinha, precisamos incluir a exclusão. Temos um link Delete próximo a cada álbum em nossa página de listagem e a abordagem simples seria deletar quando o mesmo é clicado. Isto poderia ser errado. Lembrando de nossa especificação HTTP, nos lembramos que você não pode fazer uma operação irreversível através de GET e deve usar POST para isto.

Devemos apresentar um formulário de confirmação quando o usuário clicar em Delete e se eles então clicarem em “Yes”, fazermos a exclusão. Como o formulário é trivial, vamos codificá-lo diretamente em nosso view (`Zend_Form` é, no final das contas, opcional!).

Começamos com o código da ação em `IndexController::deleteAction()`:

zf-tutorial/application/controllers/IndexController.php

```
    ...
     public function deleteAction()
    {
        if ($this->getRequest()->isPost()) {
            $del = $this->getRequest()->getPost('del');
            if ($del == 'Yes') {
                $id = $this->getRequest()->getPost('id');
                $albums = new Application_Model_DbTable_Albums();
                $albums->deleteAlbum($id);
            }
            $this->_helper->redirector('index');
        } else {
            $id = $this->_getParam('id', 0);
            $albums = new Application_Model_DbTable_Albums();
            $this->view->album = $albums->getAlbum($id);
        }
    }
    ...
```

Assim como add e edit, utilizamos o método `isPost()` do Request para determiner se devemos exibir o formulário de confirmação ou se devemos realizar a exclusão. Utilizamos o modelo `Application_Model_DbTable_Albums` para realmetne excluir uma linha utilizando o método `deleteAlbum()`. Se a solicitação não é um POST, então procuramos por um parametro ID e obtemos o registro correto no banco de dados e atribuímos ao view.

O script do view é um simples formulário:

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

```
<?php
$this->title = "Delete album";
$this->headTitle($this->title);
?>
<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>
```

Neste script, exibimos uma mensagem de confirmação para o usuário e então um formulário com botões yes e no. Na ação verificamos especificamente o valor “Yes” para fazer a exclusão.

É isto aí – você agora tem uma aplicação funcionando complemente.

Conclusão

Isto conclui nossa breve abordagem sobre como construir uma aplicação MVC simples, mas funcional, utilizando Zend Framework. Eu espero que você tenha considerado interessante e informativo. Se você encontrar algo errado, por favor, envie um email para mim em rob@akrabat.com!

Este tutorial abordou o básico da utilização do framework; existem muitos outros componentes a explorar! Eu também passei por cima de um monte de explicação. Meu website http://akrabat.com tem vários artigos sobre Zend Framework e você deve ler o manual também em http://framework.zend.com/manual!

E por fim, se você prefere livros, eu escrevi um livro *Zend Framework in Action* que está disponível para compra. Você encontra mais detalhes disponíveis em http://www.zendframeworkinaction.com. Confira! ☺