

Zend Framework 1

Alisson G. Chiquitto¹

¹Tecnologia em Análise e Desenvolvimento de Sistemas
Universidade Paranaense

Cianorte, 2016

Outline

- 1 Introdução
- 2 Zend Framework 1

Outline - Seção

- 1 Introdução
- 2 Zend Framework 1

"Orientação a objetos (OO), em uma definição formal, é um paradigma de **análise, projeto e programação** de sistemas de software baseado na **composição** e **interação** entre diversas unidades de software chamadas de **objetos**."

Este modelo visa, principalmente, **organização**, desempenho, sustentabilidade e **reutilização do software**.

"A idéia do reuso é **evitar retrabalho** no desenvolvimento de um novo projeto, sempre levando em consideração trabalhos anteriores, fazendo com que soluções previamente desenvolvidas sejam aproveitadas e implementadas em novos contextos."

"A idéia do reuso é **evitar retrabalho** no desenvolvimento de um novo projeto, sempre levando em consideração trabalhos anteriores, fazendo com que soluções previamente desenvolvidas sejam aproveitadas e implementadas em novos contextos."

Aviso

Copiar/Colar não é reutilização de software.



”Na ciência da computação, biblioteca é uma **coleção de subprogramas** utilizados no desenvolvimento de software. Bibliotecas contêm código e dados auxiliares, que **provém serviços a programas independentes**, o que permite o compartilhamento e a alteração de códigos e dados de forma modular”.

”Na ciência da computação, biblioteca é uma **coleção de subprogramas** utilizados no desenvolvimento de software. Bibliotecas contêm código e dados auxiliares, que **provém serviços a programas independentes**, o que permite o compartilhamento e a alteração de códigos e dados de forma modular”.

- O cliente chama as funções;
- Não tem fluxo de controle pré-definido;
- Não tem interação pré-definida;
- Não tem comportamento padrão.

Frameworks?

Frameworks?

Frameworks

Na Reversal Russa as Bibliotecas chamam seu código.

Frameworks?

Frameworks

~~Na Reversal Russa as Bibliotecas chamam seu código.~~
Frameworks chamam seu código.

O principal propósito de um framework é a **reutilização**. Desenvolver software pensando em reutilização leva a duas diferentes disciplinas, sendo elas desenvolvidas para reutilização e desenvolvimento com reutilização.

Frameworks reutilizam não somente as linhas de código, como também o **projeto abstrato envolvendo o domínio de aplicação**.

Framework é um **esqueleto** de **classes**, **objetos** e **relacionamentos** agrupados para construir aplicações específicas (Coad, 1992), e como um conjunto de classes abstratas e concretas que **fornece uma infra-estrutura genérica** de soluções para um conjunto de problemas (Johnson e Foote, 1988).

Com a utilização de framework **reutiliza-se análise, projeto e código**.

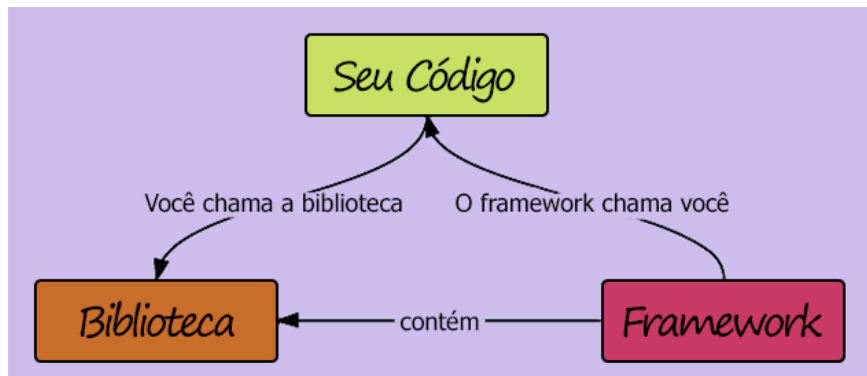
Apesar de todos os tipos de reuso serem importantes, o reuso de análise e de projeto são os que mais compensam em longo prazo.

Uma característica importante dos frameworks é que os **métodos definidos pelo usuário** para especializá-lo **são chamados de dentro do próprio framework**.

O **framework** geralmente faz o **papel de programa principal**, coordenando e sequenciando as atividades da aplicação.

Essa inversão de controle dá força ao framework para servir de esqueletos extensíveis. Os métodos fornecidos pelo usuário especializam os algoritmos genéricos definidos no framework para uma aplicação específica.

Frameworks x Biblioteca



Cristopher Alexander introduziu o conceito de padrões, catalogando um número de 253 padrões usados para resolver problemas da área de arquitetura, partindo de problemas de grande escala, como a organização do mundo em regiões, cidades, definição de espaços para agricultura, até padrões de menor escala na construção de casas, como colocação de janelas numa casa, e assim por diante.

Os conceitos de padrões e linguagens de padrões se tornaram popular na comunidade de software com a publicação dos Design Patterns pelos pesquisadores Erich Gamma, Ralph Johnson, Richard Helm e John Vlissides.

Eles criaram um catálogo de padrões para o projeto de software orientado a objeto, documentando as suas experiências na resolução de problemas independentes de domínio de aplicação.

Padrões de Projetos



Programação Orientada a Objetos para Projetos Dinâmicos

Aprendendo

Padrões de Projeto em PHP



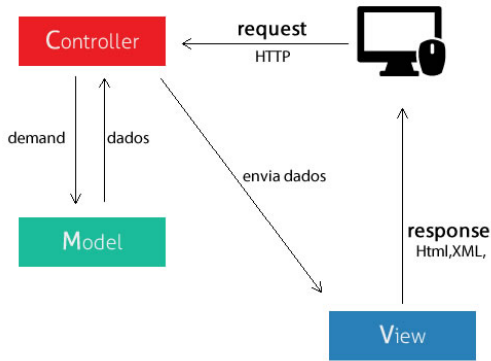
O'REILLY®

William Sanders



MVC

MVC é padrão que separa a aplicação em 3 camadas. A camada de interação do usuário (view), a camada de manipulação dos dados (model) e a camada de controle (controller).



***Model* é onde fica a lógica da aplicação. Só isso.**

Vai disparar um e-mail? Validar um formulário? Enviar ou receber dados do banco? Não importa.

A *model* deve saber como executar as tarefas mais diversas, mas não precisa saber quando deve ser feito, nem como mostrar estes dados.

***View* exibe os dados. Só isso.**

View não é só o HTML, mas qualquer tipo de retorno de dados, como PDF, Json, XML, o retorno dos dados do servidor RESTFull, os tokens de autenticação OAuth2, etc.

A *view* deve saber renderizar os dados corretamente, mas não precisa saber como obtê-los ou quando renderizá-los.

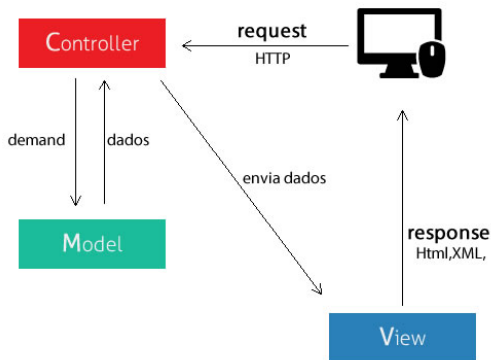
O *controller* diz quando as coisas devem acontecer. Só isso.

É usado para intermediar a *model* e a *view*. Por exemplo, para pegar dados da *model* (guardados em um banco) e exibir na *view* (em uma página HTML), ou pegar os dados de um formulário (*view*) e enviar para alguém (*model*).

Também é responsabilidade do *controller* cuidar das requisições (*request* e *response*).

O *controller* não precisa saber como obter os dados nem como exibi-los, só quando fazer isso.

- *Model* é onde fica a lógica da aplicação;
- *View* exibe os dados;
- O *controller* diz quando as coisas devem acontecer.



1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

Zend Framework

Zend Framework (ZF) é um framework para aplicações Web de código aberto, orientado a objetos, implementado em PHP 5 e licenciado sob a New BSD License.

A arquitetura do ZF permite a reutilização de componentes aplicações diversas sem requerer outros componentes ZF além das dependências mínimas.

ZF fornece componentes para muitos requisitos comuns no desenvolvimento de aplicações web, incluindo autenticação e autorização via listas de controle de acesso (ACL), configuração de aplicações, data caching, filtragem/validação de dados, internacionalização, envio de email, e acesso a Google Data APIs e outros web-services populares.

Zend Framework I

Componentes

- Acl
- Auth
- Cache
- Captcha
- Config
- Console
- Controller
- Crypt
- Currency
- Date
- Db

Zend Framework II

Componentes

- Filter
- Form
- Gdata
- Http
- Json
- Layout
- Locale
- Log
- Mail
- Navigation
- OAuth
- OpenId

Zend Framework III

Componentes

- Paginator
- Pdf
- Reflection
- Rest
- Serializer
- Session
- Soap
- Translate
- Validate
- View
- Xml

Zend Framework

Zend Framework Skeleton

Serve como um ponto de partida para o desenvolvimento de aplicações baseadas em Zend Framework.

<http://framework.zend.com/demos/ZendFrameworkQuickstart.zip>

<https://github.com/chiquitto/zf1-skeleton>

Zend Framework I

Estrutura de diretórios recomendado && Zend Framework Skeleton

```
project_name
├── application
│   ├── configs
│   │   └── application.ini
│   ├── controllers
│   │   └── helpers
│   ├── forms
│   ├── layouts
│   │   ├── filters
│   │   ├── helpers
│   │   └── scripts
│   ├── models
│   └── modules
```

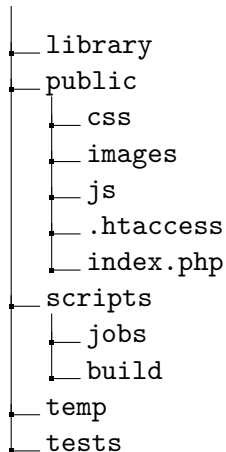

Zend Framework II

Estrutura de diretórios recomendado && Zend Framework Skeleton

```
├── services
├── views
│   ├── filters
│   ├── helpers
│   └── scripts
├── Bootstrap.php
├── data
│   ├── cache
│   ├── indexes
│   ├── locales
│   ├── logs
│   ├── sessions
│   └── uploads
└── docs
```

Zend Framework III

Estrutura de diretórios recomendado && Zend Framework Skeleton



Zend Framework

Executando o Zend Framework Skeleton

Welcome to the **Zend Framework!**

This is your project's main page



Powered By
ZENDECK

Helpful Links:

[Zend Framework Website](#) | [Zend Framework Manual](#)

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

Os controladores são sub-classes de `Zend_Controller_Action` e tem o nome terminando em `Controller` (ex.: `IndexController`).

Cada controlador possui uma ou mais *actions* que são métodos públicos cujo nome termina com `Action` (ex.: `indexAction`). As *actions* são as ações que os usuários podem acessar através de requisições HTTP.

Controller/Actions

Roteamento padrão

O roteamento padrão das URLs segue o formato:

`http://host/controller/action/parametro1/valor-parametro1`

Se o usuário acessar a URL:

`http://dominio.com.br/post/update/id/1`

Então, o ZF irá acessar o `PostController` e tentar invocar o método `updateAction`. Caso necessite, estará disponível o parâmetro `id=1`.

- Controller: `PostController` (`PostController.php`)
- Action: `updateAction`
- Parâmetro(s): `id=1`

Controller/Actions

Roteamento padrão

Acessando a URL `http://host/noticia/ver` o ZF irá invocar o *action* `NoticiaController::verAction`.

Acessando a URL `http://host/eventos/listar` o ZF irá invocar o *action* `EventosController::listarAction`.

O *action* padrão é o `index`. Acessando a URL `http://host/eventos` o ZF irá invocar o *action* `EventosController::indexAction`.

O *controller* padrão é o `index`. Acessando a URL `http://host` o ZF irá invocar o *action* `IndexController::indexAction`.

Criação de um Controller

1. Criar um arquivo `.php` no diretório `/application/controllers`.
O nome do arquivo deverá ser `NOME_CONTROLLER + "Controller.php"`
(Ex.: `PostController.php`)
2. No arquivo criado, adicionar uma classe com o mesmo nome do arquivo
(sem o `.php`) estendendo `Zend_Controller_Action`
(Ex.: `PostController`);
3. No diretório `/application/views/scripts/` criar um diretório com o nome do *controller* (em caixa baixa)
(Ex.: `/application/views/scripts/post/`)

Criação de um Controller - Exemplo

Tomaremos como base a criação de um *controller* chamado Aula.

1. Criar o arquivo `/application/controllers/AulaController.php`
2. Adicionar ao arquivo criado o *script*:

```
<?php  
class AulaController extends Zend_Controller_Action  
{}
```

3. Criar o diretório `/application/views/scripts/aula/`;

Criação de um Action

1. No *controller* que você deseja criar o *action*, crie um método público chamado `NOME_ACTION + "Action"`
(Ex.: `lerAction`)
2. No diretório `/application/views/scripts/NOME_CONTROLLER` crie um arquivo chamado `NOME_ACTION + ".phtml"`
(Ex.: `ler.phtml`)

Criação de um Action - Exemplo

Tomaremos como base a criação do *action* helloworld no *controller* Aula.

1. Criar o método helloworldAction na classe AulaController

```
<?php  
class AulaController extends Zend_Controller_Action  
{  
    public function helloworldAction() {}  
}
```

2. Criar o arquivo
/application/views/scripts/aula/helloworld.phtml
e adicionar o texto Hello World;

Acessando o Controller/Action criado

Para invocar o *action* necessário acesse a URL com os nomes do *controller* e *action* criados.

Para invocar o *action* criado nos exemplos anteriores acesse

<http://host/aula/helloworld>

Capturando parâmetros

O ZF faz a captura de dados enviados pela URL (GET) ou por POST. Em ambos os casos podemos utilizar os métodos:

`Zend_Controller_Action::getAllParams()` Retorna um *array* com os valores de todos os parâmetros;

`Zend_Controller_Action::getParam($paramName, $default = null)`
Retorna o valor do parâmetro `$paramName`. Se o parâmetro não existir retorna o valor de `$default`.

Capturando um parâmetro - Exemplo

Criando o *action* param em AulaController

```
<?php
public function paramAction() {
    $a = $this->getParam('a');
    $b = $this->getParam('b', 0);
    exit;
}
```

Acessando o *action*, teremos os respectivos valores:

- <http://host/aula/param>: a=null e b=0;
- <http://host/aula/param/a/1>: a=1 e b=0;
- <http://host/aula/param/b/2>: a=null e b=2;
- <http://host/aula/param/a/1/b/2>: a=1 e b=2;

Capturando todos os parâmetros - Exemplo

Criando o *action* allparams em AulaController

```
<?php
public function allparamsAction() {
    $all = $this->getAllParams();
    print_r($all);
    exit;
}
```

Acessando a URL:

<http://host/aula/allparams/a/1/b/10/c/100>

Teremos o *array* [a => 1, b => 10, c => 100]

Views no Zend Framework

No contexto de uma aplicação desenvolvida com o ZF, uma *View* é uma porção de código que será visualizada pelo usuário.

É papel do controlador acessar e processar dados (como os vindos do *Model*) e prepará-los para serem visualizados pelo usuário, através da *View*. Um arquivo de *View* nada mais é do que um arquivo PHP com a extensão `.phtml` e cuja função básica é exibir os dados enviados pelo controlador.

Enviar dados para a View

A partir do *controller* podemos utilizar o atributo `Zend_Controller_Action::$view` para acessar a instância de `Zend_View`.

Dentro dos arquivos `.phtml` podemos acessar a instância de `Zend_View` através da variável `$this`.

Enviar dados para a View

/application/controllers/AulaController.php

```
<?php
public function viewAction() {
    $this->view->valor_a = $this->getParam('a');
    $this->view->valor_b = $this->getParam('b', 0);
}
```

/application/views/scripts/aula/view.phtml

```
<p>A = <?php echo $this->valor_a; ?></p>
<p>B = <?php echo $this->valor_b; ?></p>
```

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

Um validador examina a entrada, e de acordo com seus requisitos, ele produz um resultado booleano.

Se a entrada não cumprir os requisitos, o validador ainda pode fornecer informações sobre quais requisitos são necessários para que a entrada se torne válida.

Métodos fornecidos

Todos os validadores implementam a *interface* `Zend_Validate_Interface` que prove os métodos:

`boolean isValid(mixed $value)` Retorna *true* se `$value` cumpre com os requisitos do validador;

`array getMessages()` Retorna um *array* de mensagens indicando os requisitos não cumpridos pelo último teste.

Exemplo: Validador de dígitos

```
<?php
$val = new Zend_Validate_Digits();
if (!$val->isValid('abc')) {
    print_r($val->getMessages());
}

/* Array (
    [notDigits] => 'abc' must contain only digits
) */
```

Exemplo: Validador de endereço de email

```
<?php
$val = new Zend_Validate_EmailAddress();
if (!$val->isValid('teste')) {
    print_r($val->getMessages());
}

/* Array (
    [emailAddressInvalidFormat] => 'teste' is not a valid email address in
        the basic format local-part@hostname
) */
```

Validadores mais comuns I

Alnum Verificar se o valor contém apenas caracteres alfabéticos/números;

Alpha Verificar se o valor contém apenas caracteres alfabéticos;

Between Verificar se um valor esta entre dois valores pré-definidos;

Date Verificar se o valor é uma data (Também permite verificar valores localizados);

Db_RecordExists Verificar se um valor esta presente no banco de dados;

Db_NoRecordExists Verificar se um valor não esta presente no banco de dados;

EmailAddress Verificar se o valor é um endereço de email válido;

GreaterThan Verificar se o valor testado é maior que um valor pré-definido;

LessThan Verificar se o valor testado é menor que um valor pré-definido;

Validadores mais comuns II

NotEmpty Verificar se valor é não vazio;

Regex Verificação baseada em Expressão Regular;

StringLength Verificar se o comprimento da *string* cumpre com critérios pré-definidos.

Exemplo: Validadores mais comuns

```
<?php
$val = new Zend_Validate_Alnum();
$val = new Zend_Validate_Alpha();
$val = new Zend_Validate_Between(array('inclusive' => true,
                                         'min' => 0, 'max' => 10));

$val = new Zend_Validate_Date();
$val = new Zend_Validate_EmailAddress();
$val = new Zend_Validate_GreaterThan(array('min' => 10));
$val = new Zend_Validate_LessThan(array('max' => 10));
$val = new Zend_Validate_Regex(array('pattern' => '/^Test/'));
$val = new Zend_Validate_StringLength(array('min' => 6, 'max' => 12));
```

Exemplo: Db_RecordExists

```
<?php
```

```
// Select * From usuario Where (email = STRING_TESTE) And (id = $id)
$validator = new Zend_Validate_Db_RecordExists(array(
    // 'adapter' => $adapter,
    'table' => 'usuario',
    'field' => 'email',
    // operacao id = valor
    'exclude' => array(
        'field' => 'id',
        'value' => $id
    ),
    // operacao id != valor
    //'exclude' => "id != $id"
));
```

- [Zend Framework Manual: Zend_Validate](#)
- [Zend Framework Manual: Standard Validation Classes](#)
- [Zend Framework API Documentation: Zend_Validate](#)

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

O que é um filtro?

No mundo real, um filtro geralmente é utilizado para remover partes desnecessárias de algo.

Quando estamos preparando café, utilizamos um filtro para remover o pó, para então utilizarmos o que sai do filtro.

Em outras palavras, um filtro é um operador que produz um subconjunto da entrada.



O componente `Zend_Filter` provê um conjunto de filtros de dados comuns.

Todos os filtros implementam a *interface* `Zend_Filter_Interface` que define o método: `mixed filter(mixed $value)`.

O método `filter()` recebe o valor a ser filtrado, e de acordo com as regras do filtro, retorna um novo valor.

Exemplo: Filtro de Dígitos

```
<?php  
$filter = new Zend_Filter_Digits();  
echo $filter->filter('Maio/2016'); // 2016
```


Exemplo: Filtro de Caixa Baixa

```
<?php  
$filter = new Zend_Filter_StringToLower();  
echo $filter->filter('EXEMPLO');
```

Filtros mais comuns I

Alnum Retorna apenas caracteres alfabéticos/números;

Alnum Remove todos os caracteres não alfabéticos;

Digits Remove todos os caracteres não dígitos;

HtmlEntities Converte caracteres "especiais" para HTML;

Null Converte valores vazios para `null`;

PregReplace Filtro baseado em Expressão Regular;

StringToLower Converte a entrada para caixa baixa;

StringToUpper Converte a entrada para caixa alta;

StringTrim Aplica a função *trim* na entrada;

StripNewLines Remove caracteres que representam *newlines*;

StripTags Remove marcações XML/HTML;

Exemplo: Filtros mais comuns

```
<?php
$filter = new Zend_Filter_Alnum();
$filter = new Zend_Filter_Alpha();
$filter = new Zend_Filter_Digits();
$filter = new Zend_Filter_HtmlEntities();
$filter = new Zend_Filter_Null();
$filter = new Zend_Filter_PregReplace(array('match' => '/bob/',
                                             'replace' => 'john'));
$filter = new Zend_Filter_StringToLower();
$filter = new Zend_Filter_StringToUpper();
$filter = new Zend_Filter_StringTrim();
$filter = new Zend_Filter_StripNewLines();
$filter = new Zend_Filter_StripTags();
```

- [Zend Framework Manual: Zend_Filter](#)
- [Zend Framework Manual: Standard Filter Classes](#)
- [Zend Framework API Documentation: Zend_Filter](#)
- [Zend Framework: LocalizedToNormalized and NormalizedToLocalized Filters](#)

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

Zend_Form simplifica a criação e manipulação de formulários na sua aplicação web.

Permite fazer:

- filtragem e validação dos dados enviados pelo usuário;
- renderização de formulários;

Métodos de Zend_Form

- `Zend_Form addElement(Zend_Form_Element $element)` Adiciona um `Zend_Form_Element` ao formulário;
- `mixed getValue(string $name)` Retorna o valor de um elemento;
- `array getValues()` Retorna um *array* com os valores de todos os elementos;
- `boolean isValid(array $data)` Testa os valores de `$data` de acordo com os validadores do formulário;
- `Zend_Form populate(array $values)` Define valores para os elementos;

Elementos de formulário

Cada tipo de elemento de formulário é representado por uma classe que estende `Zend_Form_Element`.

`Zend_Form_Element_Checkbox` Input do tipo *checkbox*;

`Zend_Form_Element_Hidden` Input do tipo *hidden*;

`Zend_Form_Element_Password` Input do tipo senha;

`Zend_Form_Element_Radio` Input do tipo *radio*;

`Zend_Form_Element_Select` Combobox;

`Zend_Form_Element_Submit` Botão *submit*;

`Zend_Form_Element_Text` Input do tipo texto simples;

`Zend_Form_Element_Textarea` Caixa de texto de múltiplas linhas;

Métodos de Zend_Form_Element

`__construct(string $name, array $options = null)` Construtor;

`Zend_Form_Element addFilter(Zend_Filter_Interface $filter)` Adiciona um filtro;

`Zend_Form_Element addValidator(Zend_Validate_Interface $validator)` Adiciona um validador;

Instanciar elementos de formulário: Checkbox

```
<?php
$e = new Zend_Form_Element_Checkbox('checkbox', array(
    'label' => 'Afirmo que gosto de gelatina',
));
```

Instanciar elementos de formulário: Hidden

```
<?php  
$e = new Zend_Form_Element_Hidden('id', array());
```

Instanciar elementos de formulário: Password

```
<?php
$e = new Zend_Form_Element_Password('senha', array(
    'label' => 'Senha',
));
```

Instanciar elementos de formulário: Radio

```
<?php
$e = new Zend_Form_Element_Radio('vidaloca', array(
    'label' => 'Meu nível de vida loka',
    'multioptions' => array(
        1 => 'Mijo nasarvore',
        2 => 'Vida loka',
        3 => 'Lokão'
    )
));
```

Instanciar elementos de formulário: Select

```
<?php
$e = new Zend_Form_Element_Select('sabor', array(
    'label' => 'Sabor preferido',
    'multioptions' => array(
        1 => 'Morango',
        2 => 'Limão',
        3 => 'Manga'
    )
));
```

Instanciar elementos de formulário: Submit

```
<?php
$e = new Zend_Form_Element_Submit('submit', array(
    'label' => 'Salvar'
));
```

Instanciar elementos de formulário: Text

```
<?php
$e = new Zend_Form_Element_Text('nome', array(
    'label' => 'Nome completo',
));
```


Instanciar elementos de formulário: Textarea

```
<?php
$e = new Zend_Form_Element_Textarea('doidera', array(
    'label' => 'Sua maior doidera',
));
```

Criação de um formulário

Iremos criar um formulário de cadastro de categorias de um blog. O formulário deverá conter um campo de texto para o usuário informar o nome da Categoria.

1. Criar o arquivo `/application/forms/Categoria.php`;
2. No arquivo, criar a classe `Application_Form_Categoria`;
3. Criar um método público `init()`;

No método `init`

1. Definir o envio do formulário para POST;
2. Instanciar um elemento para o nome da categoria e adicioná-lo ao formulário;
3. Instanciar um botão *submit* e adicioná-lo ao formulário;

Classe do Formulário de Categoria

```
<?php
class Application_Form_Categoria extends Zend_Form {
    public function init() {
        $this->setMethod('post');

        $categoria = new Zend_Form_Element_Text('categoria', array(
            'label' => 'Nome da categoria',
        ));
        $this->addElement($categoria);

        $submit = new Zend_Form_Element_Submit('submit', array(
            'label' => 'Salvar'
        ));
        $this->addElement($submit);
    }
}
```

Instanciar Formulário e gerar HTML

No *controller* instanciamos o formulário e enviamos para a *view*.

```
<?php  
$form = new Application_Form_Categoria();  
$this->view->form = $form;
```

E na *view* escrevemos o objeto do formulário.

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

Preenchimento obrigatório

Para tornar um elemento para preenchimento obrigatório pelo usuário, adicionamos a opção `required => true`.

```
<?php
$e = new Zend_Form_Element_Text('nome', array(
    'label' => 'Nome completo',
    'required' => true
));
```

- [Zend Framework Manual: Zend_Form](#)
- [Standard Form Elements Shipped With Zend Framework](#)
- [Zend Framework API Documentation: Zend_Form](#)

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

O componente Zend_Db fornece:

- interface orientada a objetos para tabelas de banco de dados;
- métodos para operações comuns em tabelas.

`Zend_Db_Adapter_Abstract` Gateway de conexão;

`Zend_Db_Table_Abstract` Tabela de banco de dados;

`Zend_Db_Expr` Expressão literal de SQL;

`Zend_Db_Select` Geração de instruções SQL Select;

`Zend_Db_Table_Rowset_Abstract` Conjunto de registros;

`Zend_Db_Table_Row_Abstract` Um registro;

`Zend_Db_Table_Select` Geração de instruções SQL Select de uma Tabela;

Definição de uma Tabela

Vamos tomar como base a criação de uma classe para a tabela chamada categoria.

1. Criar o arquivo `/application/models/DbTable/Categoria.php`;
2. Criar a classe `Application_Model_DbTable_Categoria` estendendo `Zend_Db_Table_Abstract`;
3. Atribuir ao atributo `$_name` o nome da tabela (categoria);
4. Atribuir ao atributo `$_primary` o da coluna PK (idcategoria). Se for PK composta utilize um *array*;

Definição da Tabela Categoria

```
<?php
class Application_Model_DbTable_Categoria
extends Zend_Db_Table_Abstract {
    protected $_name = 'categoria';
    protected $_primary = 'idcategoria';
}
```

Consultar Registros

```
<?php
$tb = new Application_Model_DbTable_Categoria();
// retorna todos os registros
$array = $tb->fetchAll();
```

Inserir Registros

```
<?php
$tb = new Application_Model_DbTable_Categoria();
$tb->insert(array(
    'idcategoria' => null,
    'categoria' => 'Esportes',
));
```

Atualizar Registros

```
<?php
$tb = new Application_Model_DbTable_Categoria();
$tb->update(array(
    'categoria' => 'Política',
), 'idcategoria = 1');
```

Apagar Registros

```
<?php  
$tb = new Application_Model_DbTable_Categoria();  
$tb->delete('idcategoria = 1');
```

- [Zend Framework Manual: Zend_Db](#)
- [Zend Framework Manual: Zend_Db_Table](#)
- [Zend Framework API Documentation: Zend_Db](#)

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

O `Zend_Auth` fornece uma API para autenticação e inclui adaptadores para os cenários de uso mais comuns:

- Banco de dados;
- HTTP;
- LDAP;
- Open ID;

E o desenvolvedor pode criar seus próprios adaptadores estendendo os já existentes.

Classes importantes

`Zend_Auth` Classe para autenticação;

`Zend_Auth_Adapter_DbTable` Adaptador de autenticação com Banco de Dados;

`Zend_Auth_Result` Retorno da autenticação;

`Zend_Auth_Storage_Session` Persistência da autenticação por SESSION;

Zend_Auth

`Zend_Auth_Result authenticate(Zend_Auth_Adapter_Interface $a)`

Faz a autenticação do usuário;

`void clearIdentity()` Limpa a identidade do usuário (*logout*);

`mixed|null getIdentity()` Retorna a identidade do usuário;

`Zend_Auth getInstance()` Pega a instância de `Zend_Auth`;

`Zend_Auth_Storage_Interface getStorage()` Retorna o *storage* da autenticação;

`boolean hasIdentity()` Retorna `true` se há usuário autenticado;

Zend_Auth_Adapter_DbTable I

`stdClass getResultRowObject(string|array $returnColumns = null, string|array $omitColumns = null)` Retorna o registro do usuário autenticado;

`Zend_Auth_Adapter_DbTable setCredential(string $credential)`
Define a credencial de autenticação;

`Zend_Auth_Adapter_DbTable setCredentialColumn(string $credentialColumn)` Define o atributo da credencial no banco de dados;

`Zend_Auth_Adapter_DbTable setIdentity(string $value)` Define a identidade para autenticação;

Zend_Auth_Adapter_DbTable II

`Zend_Auth_Adapter_DbTable setIdentityColumn(string $identityColumn)` Define o atributo da identidade no banco de dados;

`Zend_Auth_Adapter_DbTable setTableName(string $tableName)` Define o nome da tabela no banco de dados;

Zend_Auth_Result I

`boolean isValid()` Retorna true se o resultado representa uma autenticação com sucesso;

`Zend_Auth_Adapter_DbTable setCredential(string $credential)`
Define a credencial de autenticação;

`Zend_Auth_Adapter_DbTable setCredentialColumn(string $credentialColumn)` Define o atributo da credencial no banco de dados;

`Zend_Auth_Adapter_DbTable setIdentity(string $value)` Define a identidade para autenticação;

Zend_Auth_Result II

`Zend_Auth_Adapter_DbTable setIdentityColumn(string $identityColumn)` Define o atributo da identidade no banco de dados;

`Zend_Auth_Adapter_DbTable setTableName(string $tableName)` Define o nome da tabela no banco de dados;

Zend_Auth_Storage_Interface

```
void write(mixed $contents) Registra $contents no storage;
```

- [Zend Framework Manual: Zend_Auth](#)
- [Zend Framework Manual: Database Table Authentication](#)
- [Zend Framework API Documentation: Zend_Auth](#)

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

`Zend_Auth` é responsável apenas pela autenticação do usuário, sendo que ele não permite o controle de permissões aos recursos do aplicativo.

O papel de controle de permissões é do `Zend_Acl`.

ACL (*Access Control List* - lista de controle de acesso) é uma solução para realizar o controle do acesso a determinados recursos.

- papel (*role*): um grupo de usuários;
- recurso (*resource*): algo a ser protegido;
- privilégio (*privilege*): o tipo de acesso exigido

```
boolean isAllowed(string $role = null, string $resource = null,  
string $privilege = null) Retorna true se $role tem acesso à  
$resource.$privilege;
```

- [Zend Framework Manual: Zend_Acl](#)
- [Zend Framework API Documentation: Zend_Acl](#)

1 Introdução

2 Zend Framework 1

- Controller/Actions
- Zend_Validate
- Zend_Filter
- Zend_Form
- Zend_Db
- Zend_Auth
- Zend_Acl
- Zend_Navigation

O `Zend_Navigation` pode ser utilizado para criar itens de navegação:

- menus;
- *breadcrumbs*;
- *sitemaps*.

Além disso, pode servir como um modelo para outros propósitos relacionados a navegação, como por exemplo, integração com `Zend_Acl`.

Page Uma página (`Zend_Navigation_Page`);

Container Uma página pode conter um *container* (`Zend_Navigation_Container`). Um *container* pode conter várias páginas.

Uma página (`Zend_Navigation_Page`) no `Zend_Navigation` é um objeto que contém um *link* para uma página web.

Além do link, ela também contém outros dados relevantes como o título da página.

Especializada por `Zend_Navigation_Page_Mvc` e `Zend_Navigation_Page_Uri`.

Um contêiner de navegação (`Zend_Navigation_Container`) é uma caixa para as páginas (`Zend_Navigation_Page`).

Contém métodos para adicionar, pesquisar/recuperar, deletar e iterar as páginas.

Iniciando o Zend_Navigation

`Zend_Navigation_Container` é uma classe abstrata, portanto não podemos instanciar-la.

Use `Zend_Navigation` para criar um *container* de páginas.
Podemos criar um *container* a partir de um *array* ou uma instância de `Zend_Config`.

Criando XML de Páginas do Zend_Navigation

Criar o arquivo `application/configs/navigation.xml` listando as páginas.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <nav>
    <home>
      <label>Home</label>
      <controller>index</controller>
      <action>index</action>
      <uri></uri>
    </home>
    <categorias>
      <label>Categorias</label>
      <controller>index</controller>
      <action>categorias</action>
    </categorias>
  </nav>
</config>
```

Criando XML de Páginas do Zend_Navigation

Adicionando containers

Para criar um *container* use o nó `pages`.

```
<pagina>
  <label>Página</label>
  <controller>pagina</controller>
  <action>index</action>
  <pages>
    <subpagina>
      <label>Subpagina</label>
      <controller>pagina</controller>
      <action>sub</action>
    </subpagina>
  </pages>
</pagina>
```

Criando instancia de Zend_Navigation

```
<?php
```

```
$xml = APPLICATION_PATH . '/configs/navigation.xml';  
$config = new Zend_Config_Xml($xml, 'nav');  
$container = new Zend_Navigation($config);
```


Gerando um menu

Na *view*, use:

```
<?php echo $this->navigation(); ?>
```

Gerando um breadcrumb

Na *view*, use:

```
<?php
echo $this->navigation()->breadcrumbs()
    ->setMinDepth(null)
    ->setMaxDepth(1)
    ->setSeparator(' &raquo; ' . PHP_EOL)
;
?>
```

Criando XML de Páginas do Zend_Navigation

Adicionando containers

Para criar um *container* use o nó `pages`.

```
<pagina>
  <label>Página</label>
  <controller>pagina</controller>
  <action>index</action>
  <pages>
    <subpagina>
      <label>Subpagina</label>
      <controller>pagina</controller>
      <action>sub</action>
    </subpagina>
  </pages>
</pagina>
```

- [Zend Framework Manual: Zend_Navigation](#)
- [Zend Framework Manual: Zend_Navigation Pages](#)
- [Zend Framework Manual: Zend_Navigation Containers](#)
- [Zend Framework API Documentation: Zend_Navigation](#)
- [Zend Framework Manual: Navigation Helpers](#)

Referências I



Marchi, Gécen Dacome

Um framework para sistemas baseados em conhecimento no contexto da metodologia CommonKADS

Dissertação de Mestrado.



Minetto, Elton Luís

Zend Framework na prática



Sanders, William

Aprendendo Padrões de Projeto em PHP




Novatec, 2013.



Freeman, Eric. Freeman, Elisabeth, Sierra, Kathy

Head First Design Patterns

O'Reilly Media, 2004.

-  Entendendo o padrão MVC na prática
<http://tableless.com.br/entendendo-o-padrao-mvc-na-pratica/>
-  Zend Framework Reference
<http://framework.zend.com/manual/1.12/en/reference.html>
-  Zend Framework API Documentation
<http://framework.zend.com/apidoc/1.12/index.html>