2km INTERATIVA!

Curso de

Belo Horizonte - Minas Gerais - Brasil

www.2km.com.br
+55 31 3282.6280

Nota de copyright

As informações contidas neste documento estão sujeitas a alterações sem prévio aviso. As empresas, os nomes de pessoas e os dados aqui mencionados são fictícios, salvo indicação em contrário.
Nenhuma parte deste documento pode ser reproduzida ou transmitida de qualquer forma ou meio, eletrônico ou mecânico, para qualquer propósito, sem a permissão expressa, por escrito, da 2km interativa!.

©2011 2km interativa!. Todos os direitos reservados.

CakePHP™ é marca registrada da Cake Software Foundation, Inc.

Conteúdo

Introdução

Desenvolver um sistema com qualidade, dentro do prazo e custo estipulados é um desafio que as empresas de desenvolvimento de *software* sempre buscaram atender. Uma solução conhecida para este problema é a reusabilidade.

Esta consiste em reaproveitar partes do sistema, que já existem e desta forma obter maior produtividade e qualidade no desenvolvimento. O ganho de qualidade acontece em virtude da reutilização de partes que já foram testadas previamente em outros trechos do *software*, ou até mesmo em outras aplicações. O aumento da produtividade decorre do reaproveitamento de partes já existentes.

A reusabilidade pode ser feita através de cópia de código-fonte, o que não é recomendado, pois replicar o código-fonte pode implicar em problemas de manutenção. Por meio de classes, componentes e herança, podendo chegar a um nível mais alto que são os *frameworks*.

Um *framework* é uma infra-estrutura ou esqueleto de uma família de aplicações projetado para ser reutilizado. Basicamente, aplicações específicas são construídas especializando as classes do *framework* para fornecer a implementação de alguns métodos e, assim a aplicação implementa somente as funcionalidades específicas (DEUTSCH, 1989).

As vantagens do uso de *frameworks* nos projetos, de acordo com Assis (2003) e Sauvé (2004) são:

- Redução do tempo de codificação: muitas funcionalidades necessárias já estão disponíveis no *framework*;
- Soluções bem testadas por outras pessoas: Os *frameworks* são utilizados por muitas pessoas, isto garante um alto nível de maturidade ao se descobrir erros e adicionar novas funcionalidades;
- Programadores implementam somente o que é necessário: não é preciso que se codifique todo o *software*, pois se utiliza os componentes que já estão prontos;
- Redução de erros: os *frameworks* diminuem o número de linhas de codificação, desta forma, reduzem também a possibilidade de erros comuns.

É importante destacar a diferença de um *framework* para uma biblioteca de classes, dentre as diferenças a de maior destaque é a forma como o fluxo de execução é controlado. Quando se utiliza uma biblioteca de classes, é o programador que define como vai ser a execução dos comandos. Diferentemente das bibliotecas, os *frameworks* invertem este controle de execução, sendo eles os responsáveis por controlar o fluxo da execução.

Sobre o CakePHP

CakePHP é um *framework* para desenvolvimento ágil, escrito na linguagem PHP. Ele usa padrões de projetos conhecidos, como ORM e MVC; utiliza também o paradigma das convenções no lugar do uso de configurações. Este *framework* busca a redução do custo de desenvolvimento através da diminuição do número de linhas de código-fonte.

Ele é um *software* livre, licenciado sob a licença MIT - http://www.opensource.org/licenses/mit-license.php.

Vantagens do CakePHP

Dentre os diversos recursos que agregam valor ao utilizar o CakePHP, podemos citar:

- *Software* livre com uma comunidade crescente e ativa.
- Tem uma licença flexível (MIT).
- Compatível com PHP 4 e 5.
- Facilidade de gerar a codificação do CRUD.
- *Scaffolding* - geração de telas dinâmicas (*on the fly*).
- Geração de código-fonte
- Arquitetura MVC
- Uso de URLs amigáveis
- Validação embutida
- Templates rápidos e flexiveis, utilizando sintaxe PHP
- Desenvolvimento de *views* com *helpers*
- Diversos componentes disponíveis

- Lista de controle de acessos (ACL)
- Proteção contra dados maliciosos (*Data Sanitization*)
- Ferramentas para *caching*
- Internacionalização e Localização
- Funciona em qualquer servidor *web* rodando PHP.
- E o principal: devolve ao programador a diversão de programar.

1 - Instalação

1.1 - Requisitos para Utilização do CakePHP

O *framework* CakePHP funciona em qualquer servidor web que seja capaz de executar *scripts* em linguagem PHP 4 ou superior.

- Servidor web: Apache 2.0, LightHTTP, IIS. Sendo recomendado a utilização do apache com o módulo mod_rewrite.
- PHP 5.2.6 ou superior. O CakePHP também tem suporte para versão 4 PHP.
- Banco de Dados suportados pelo CakePHP:
  - MySQL (4 ou superior)
  - PostgreSQL
  - Firebird
  - Microsoft SQL Server
  - Oracle
  - SQLite
  - ODBC
  - ADOdb

1.2 - Configurando o Apache

O servidor web Apache é um *software* livre mantido pela organização Apache Foundation. O servidor Apache é o servidor mais utilizado para hospedar páginas na internet.

O CakePHP pode ser configurado de diversas maneiras no servidor Apache, este capítulo irá apresentar as diferentes formas de configuração, destacando seus prós e contras.

O arquivo de configuração do Apache é o **httpd.conf**. Este arquivo é encontrado nos sistemas *nix dentro do diretório /etc/apache2/. No windows o arquivo será encontrado dentro do diretório "conf", geralmente localizado no diretório de instalação do Apache.

1.2.1 - Configurando o mod_rewrite

Este módulo é utilizado para criar endereços (URLs) mais amigáveis, evitando endereços longos com diversos &. Por exemplo:

Nota

O endereço: http://www.dominio.com.br/index.php?nome=teste&data=20081110. Utilizando a reescrita de URL poderia ser escrito assim: http://www.dominio.com.br/teste/20081110

Para habilitar o módulo *rewrite* no servidor apache basta descomentar* a seguinte linha:

```
#LoadModule rewrite_module modules/mod_rewrite.so
```

Observação: O caminho do módulo pode alterar dependendo da instalação do Apache ou do sistema operacional.

Nota

*comentários no arquivo de configuração são as linhas que começam com #

1.2.2 - Configurando um endereço por meio de *alias*

As configurações de *alias* no apache são utilizadas para gerar endereços como:

```
http://dominio/nome_do_alias
```

Para criar um *alias* no apache é necessário editar o arquivo httpd.conf inserindo as seguintes linhas:

```
Alias /projetos /projetos/cakephp/
<Directory /projetos/cakephp/>
        Options Indexes FollowSymLinks
        AllowOverride All
        Order allow,deny
        Allow from All
</Directory>
```

A primeira linha indica o nome do *alias*, sendo este nome utilizado no endereço após o domínio. A seguir é inserido o endereço físico do site.

As próximas linhas passam algumas configurações do *alias* para o apache. Sendo importante observar que para o módulo *rewrite* funcionar neste *alias* é necessário deixar a opção **AllowOverride** com o valor **All**.

Neste exemplo foi configurado um *alias* chamado projetos; isto significa, que ao fazer uma requisição no endereço: http://localhost/**projetos**, o servidor Apache envia a página principal (index) de projetos encontrada no diretório /projetos/cakephp/. Além disto, este *alias* está com a opção de reescrever a url ativada (**módulo rewrite** + **allowOverride All**), sendo assim o servidor procura a existência do arquivo .htaccess no diretório solicitado e executa as regras escritas nele.

1.2.3 - Configurando um endereço usando virtual host

O recurso de *host* virtual é utilizado para configurar diversos endereços em uma mesma máquina, com um ou mais endereços IPs.

O uso do virtual *host* na máquina de desenvolvimento é aconselhável, pois simula o ambiente de produção e pode reduzir problemas com alguma configuração adicional do mod_rewrite.

Para habilitar e configurar o *host* virtual é necessário seguir os seguintes passos:

1. Abrir o arquivo httpd.conf e descomentar a linha:

   ```
   #LoadModule vhost_alias_module libexec/apache2/mod_vhost_alias.so
   ```

2. No arquivo httpd.conf, descomentar também a linha:

   ```
   #Include /private/etc/apache2/extra/httpd-vhosts.conf
   ```

3. Editar o arquivo extra/httpd-vhosts.conf

   ```
   NameVirtualHost *

   <VirtualHost *>
      ServerName projetos
      DocumentRoot /projetos/cakephp/
      ErrorLog /var/log/apache2/projetos-error_log
      CustomLog /var/log/apache2/projetos-access_log combined
      <Directory "/projetos/cakephp/">
          Options Indexes FollowSymLinks
          AllowOverride All
          Order allow,deny
          Allow from all
      </Directory>
   </VirtualHost>
   ```

```
<VirtualHost *>
   ServerName localhost
   DocumentRoot /projetos/sistemaslegados/
   ErrorLog /var/log/apache2/sistemaslegados-error_log
   CustomLog /var/log/apache2/sistemaslegados-access_log combined
   <Directory "/projetos/sistemaslegados/">
       Options Indexes FollowSymLinks
       AllowOverride All
       Order allow,deny
       Allow from all
   </Directory>
</VirtualHost>
```

Listagem 1 - Configuração do arquivo `httpd-vhosts.conf`

4. Editar o arquivo hosts:

   Caminho no *nix: /etc/hosts

   Caminho no windows: C:\Windows\system32\drivers\etc\hosts

```
127.0.0.1       localhost
127.0.0.1       projetos
```

Listagem 2 - Configuração do arquivo `hosts`

5. Reiniciar o servidor Apache

A configuração mostrada acima cria dois *hosts* virtuais - localhost e projetos. Ao acessar `http://localhost/` no *browser*, o servidor apache irá direcionar para o conteúdo da pasta: */projetos/sistemaslegados/*. Ao acessar `http://projetos/`, o Apache direciona para o conteúdo da pasta: */projetos/cakephp/*.

O virtual *host* pode ter configuração de *log* independente para cada *host*, sendo que os arquivos responsáveis em armazenar a informação do *log* podem ser indicados pelos parâmetros: **ErrorLog** e **CustomLog**

1.2.4 - Configurando o mod_userdir

O módulo de usuário habilita cada usuário do sistema operacional a ter seu próprio site. Ao utilizar este módulo o endereço do site fica assim: `http://dominio/~usuarioDoSistema/`

Para habilitar este módulo basta descomentar a linha:

```
#LoadModule userdir_module libexec/apache2/mod_userdir.so
```

Listagem 3 - Configuração do módulo `userdir_module` no arquivo `httpd.conf`

1.2.5 - Comparando as configurações do Apache

Os desenvolvedores do CakePHP recomendam o uso do módulo rewrite. Este módulo procura sempre um arquivo chamado `.htaccess` e verifica suas regras.

No CakePHP existem três arquivos `.htaccess` que ficam localizados nos diretórios:

- /CAMINHO_DO_CAKE/.htaccess
- /CAMINHO_DO_CAKE/app/.htaccess
- /CAMINHO_DO_CAKE/app/webroot/.htaccess

Dependendo da configuração do Apache utilizada é necessário editar estes três arquivos `.htaccess`.

No caso de utilizar a configuração de sua aplicação com uso de *alias*, deve-se adicionar a seguinte linha em seu `.htaccess`:

```
<IfModule mod_rewrite.c>
   RewriteEngine on
   RewriteBase    /NOME_DO_ALIAS/
   #Por exemplo: RewriteBase  /projetos/
   RewriteRule    ^$ app/webroot/    [L]
   RewriteRule    (.*) app/webroot/$1 [L]
</IfModule>
```

Listagem 4 - Configuração do `.htaccess` utilizando *alias*

No módulo *userdir* torna-se necessário editar o arquivo `.htaccess` na seguinte linha:

```
RewriteBase       /~NOME_DO_USUARIO/
```

Listagem 5 - Configuração do `.htaccess` utilizando *mod_userdir*

A configuração por meio de *host* virtual dispensa a configuração dos arquivos .htaccess, porém seus *hosts* virtuais não são acessíveis para outros computadores da rede.

1.3 - Instalando o CakePHP

O primeiro passo para a instalação do *framework* é fazer o seu *download* no seguinte endereço:

- Na página principal do projeto existe um link para a última versão estável:

  ```
  http://www.cakephp.com.br
  ```

Após o *download* dos fontes, é necessário descompactar o arquivo na pasta desejada.

O servidor web deve ser configurado para o diretório onde o CakePHP foi extraído. Por exemplo:

O CakePHP está na pasta: *C:\projetos\cakephp*

Sendo o conteúdo de *C:\projetos\cakephp* as pastas *app*, *cake* e *vendors*.

O servidor web deve ser configurado para o diretório: *C:\projetos\cakephp*

Nota

A versão do framework CakePHP utilizada neste material é a versão estável 1.3.10.

1.4 - Estrutura de diretórios

Após descompactar o arquivo baixado pelo site do CakePHP, a seguinte estrutura é encontrada:

Diretório	Descrição
/app/	Aqui ficam os arquivos da aplicação que será desenvolvida.
/app/config/	Onde ficam os arquivos de configuração. O arquivo de configuração de banco de dados (database.php) se encontra aqui.
/app/controllers	As classes da camada de controle devem ser escritas neste diretório.

Diretório	Descrição
/app/libs	Contém as bibliotecas primárias que não vieram de terceiros ou empresas externas. Isto permite que você separe suas bibliotecas internas das biliotecas de fornecedores.
/app/locale	Contém os arquivos com as strings para a internacionalização.
/app/models	As classes da camada de modelo devem ser escritas neste diretório.
/app/views	Os arquivos da camada de visão devem ser escritas neste diretório.
/cake/	Os arquivos do *framework* ficam neste diretório. O desenvolvedor não deve alterar o conteúdo desta pasta, somente se souber o que está fazendo.
/app/webroot/	Todas as requisições são direcionadas para este diretório, que contém o arquivo index.php (o *Dispatcher*) que trata quais arquivos são necessários para atender cada requisição. As pastas deste diretório servem como abrigo para arquivos css, imagens, javascripts e qualquer outro arquivo que precisa estar disponível para a requisição.

Tabela 1 - Estrutura de diretórios do CakePHP

1.5 - Arquivos de configuração do CakePHP

O CakePHP possui poucos arquivos de configuração, devido sua arquitetura baseada em convenções.

1.5.1 - O arquivo de configuração `database.php`

Para iniciar o desenvolvimento de uma aplicação, é necessário apenas configurar o banco de dados por meio do arquivo: *diretório_do_cake/conf/database.php*.

O banco de dados é configurado pelo array: `$default`, que é ilustrado abaixo.

```
var $default = array(
        'driver'      => 'mysql',
        'persistent'  => false,
        'host'        => 'localhost',
        'login'       => 'cakephpuser',
        'password'    => 'c4k3roxx!',
        'database'    => 'my_cakephp_project',
        'prefix'      => ''
    );
```

Listagem 6 - Configuração do arquivo `database.php`

Segue abaixo a listagem de configurações do array de conexão com o banco de dados.

Chave	Descrição

Chave	Descrição
driver	Define qual driver de banco de dados será usado na aplicação, sendo possível escolher as opções:mysql, postgres, sqlite, pear-nomeDoDriver, adodb-nomeDoDriver, mssql, oracle, or odbc.
persistent	Define se a conexão ao banco de dados deve ser feita de forma persistente ou não.
host	O endereço do computador onde o SGBD está instalado.
login	Usuário que possui acesso ao banco de dados.
password	Senha do usuário para acesso ao banco de dados.
database	Nome do banco de dados que será utilizado.
prefix	Chave opcional que, deve ser configurada caso as tabelas do banco de dados utilizem algum prefixo.
port	Chave opcional, porta TCP para conexão com o banco de dados.
encoding	Tipo de caracteres que serão utilizados na conexão, por default a maioria dos bancos de dados utilizam UTF-8.
schema	Usado apenas pelo PostgreSQL para definir qual esquema usar.

Tabela 2 - Chaves disponíveis para a configuração com o banco de dados no arquivo `database.php`

1.5.2 - O arquivo de configuração `core.php`

O arquivo `core.php` possui as configurações básicas do *framework*. Confira abaixo as suas opções de configuração:

Variável	Descrição
debug	Configura o nível de mensagens de debug que será mostrado na tela. **0** = Modo sem nenhuma mensagem, para uso do sistema em produção. **1** = Mostra mensagens de erros e advertências. **2** = Mostra mensagens de erros, advertências e comandos SQL. **3** = Mostra mensagens de erros, advertências, comandos SQL e variáveis do controle.
App.baseUrl	Descomente esta linha caso deseje utilizar o CakePHP sem o mod_rewrite. É importante apagar todos os arquivos .htaccess No IIS pode ser necessário colocar uma ? concatenada, como demonstrado abaixo: Configure::write('App.baseUrl', env('SCRIPT_NAME')."?");
Routing.prefixes	Descomente a linha para utilizar as rotas administrativas. Por default o nome da rota administrativa é admin, podendo ser alterada para qualquer outra palavra.

Cache.disable	Quando definido como *true*, o *cache* será desabilitado para toda a aplicação.
Cache.check	Quando o valor é verdadeiro, o recurso de *cache* na visão é habilitado. Para usar *cache* ainda é necessário configurar as classes de controle.
Session.save	Configura de qual forma as sessões serão armazenadas. **php** = usa o padrão do PHP. **cake** = os dados de sessão são armazenados no diretório /app/tmp **database** = os dados serão armazenado em uma tabela do banco de dados, para utilizar este recurso é necessário criar a tabela. O script SQL para criação é encontrado em: /app/config/sql/sessions.sql.
Session.table	Nome da tabela que o CakePHP utilizará para gravar informações da sessão.
Session.database	Nome do banco de dados usado para gravar informações da sessão.
Session.cookie	Nome do *cookie* usado para as sessões.
Session.timeout	Tempo de duração de uma sessão (em segundos). Este valor depende da configuração do Security.level.
Session.start	O *framework* inicia uma sessão automaticamente quando este parâmetro é verdadeiro.
Session.checkAgent	Quando esta configuração é falsa o CakePHP não verifica a variável $_SERVER["HTTP_USER_AGENT"], ou seja não existe uma verificação se a requisição esta sendo feita pelo mesmo agente (mesmo navegador).
Security.level	Nível de segurança do CakePHP. O valor definido em 'Session.timeout' é multiplicado pelo peso do nível de segurança definido abaixo: Valores válidos: 'high' = x 10 'medium' = x 100 'low' = x 300 Os valores 'high' e 'medium' também habilitam a configuração de sessão do php: session.referer_check O ID da sessão do CakePHP é recriado entre as requisições, quando 'Security.level' é 'high'.
Security.salt	Uma string aleatória, usada para a tabela *hash* de segurança.
Security.cipherSeed	Uma string numérica (somente dígitos) utilizada para criptografar/descriptografar strings.
Acl.classname, Acl.database	Constantes utilizadas pelo recurso de *Access Control List* (ACL, ou lista de controle de acessos).

Tabela 3 - Configurações do *framework* no arquivo `core.php`

1.5.3 - O arquivo de configuração `bootstrap.php`

Este arquivo de configuração pode ser utilizado para:

- Registrar constantes globais
- Criar funções utilitárias
- Definir caminhos adicionais para as classes de modelo, classes de controle e páginas da visão

O arquivo `bootstrap.php` é carregado na inicialização da aplicação.

Um exemplo prático de uma função que pode ser codificada no `bootstrap.php` seria a função abaixo, que imprime o conteúdo do array passado como parâmetro e logo depois executa o die(), que interrompe a execução do script. Este é um método que pode auxiliar no *debug* da aplicação.

```
/*
```

```
* Função para imprimir o conteúdo passado como parâmetro e terminar a execução do
script
*/
function prd($message){
   die(pr($message));
 }
```

Listagem 7 - Função que pode ser codificada no arquivo bootstrap.php

Nota

A função pr() no cake é um *alias* para a função print_r() nativa do PHP.

1.6 - Instalação Avançada do CakePHP

A instalação avançada do CakePHP consiste em separar o *framework* da aplicação, desta forma é possível ter uma única instalação do CakePHP para diversas aplicações.

O primeiro passo é mover a pasta do framework: */CAMINHO_DO_CAKE/cake* para o diretório desejado. Por exemplo: *C:\cake*, desta forma a pasta ficaria *C:\cake\cake* (arquivos e pastas do framework).

Nos sistemas *nix é recomendado usar a pasta */usr/lib/*, logo, deve-se criar o diretório */usr/lib/cake* e mover a pasta cake para o novo diretório, desta forma ficaria: */usr/lib/cake/cake/* (arquivos e pastas do framework)

Como a pasta do *framework* foi movida para outro local, o CakePHP precisa ser configurado com o novo endereço. Para isto precisamos editar a seguinte linha do arquivo /CAMINHO_DO_CAKE/app/webroot/index.php:

```
if (!defined('CAKE_CORE_INCLUDE_PATH')) {
        define('CAKE_CORE_INCLUDE_PATH', ROOT);
}
```

Listagem 8 - Configuração original do caminho para o *framework* no arquivo /CAMINHO_DO_CAKE/app/webroot/index.php

Deixando desta forma:

```
//WINDOWS
if (!defined('CAKE_CORE_INCLUDE_PATH')) {
        define('CAKE_CORE_INCLUDE_PATH', 'C:'.DS.'cake');
}
/*-------------------------------------------------------------------------*/
//*nix
if (!defined('CAKE_CORE_INCLUDE_PATH')) {
        define('CAKE_CORE_INCLUDE_PATH', DS.'usr'.DS.'lib'.DS.'cake');
}
```

Listagem 9 - Configuração alterada do caminho para o *framework* no arquivo /CAMINHO_DO_CAKE/app/webroot/index.php

Pronto!!! A aplicação está funcionando separada do *framework*.

Outra opção seria separar a pasta *webroot* da sua aplicação. As aplicações, no ambiente de produção, devem ser configuradas desta forma.

Neste caso é necessário configurar as seguintes linhas do arquivo /CAMINHO_DO_CAKE/app/webroot/index.php:

```
//Caminho para sua aplicação
if (!defined('ROOT')) {
        define('ROOT', dirname(dirname(dirname(__FILE__))));
}
//nome da pasta da sua aplicação
if (!defined('APP_DIR')) {
        define('APP_DIR', basename(dirname(dirname(__FILE__))));
}
```

Listagem 10 - Configuração original do arquivo /CAMINHO_DO_CAKE/app/webroot/index.php

Como exemplo real vamos sugerir a seguinte configuração, para um sistema de bookmarks:

- Biblioteca do *framework*: */usr/lib/cake/*
- A aplicação: */home/usuario/bookmarks/*
- O webroot: */var/www/htdocs/* (Este diretório deve ser o *Document Root* da configuração do Apache)

A configuração necessária seria:

```
//Caminho para sua aplicação
if (!defined('ROOT')) {
        define('ROOT', DS.'home'.DS.'usuario');
}
//nome da pasta da sua aplicação
if (!defined('APP_DIR')) {
        define('APP_DIR', 'bookmarks');
}
//caminho da biblioteca do framework
if (!defined('CAKE_CORE_INCLUDE_PATH')) {
        define('CAKE_CORE_INCLUDE_PATH', DS."usr".DS."lib".DS."cake");
}
```

Listagem 11 - Configuração alterada do arquivo /CAMINHO_DO_CAKE/app/webroot/index.php

2 - Desenvolvendo com o CakePHP

O CakePHP utiliza o paradigma de desenho de *software* da Convenção ao invés da Configuração (em inglês *Convention over Configuration* ou simplesmente *Coding by convention*). Este paradigma procura diminuir o número de decisões que os desenvolvedores precisam tomar, ganhando em simplicidade, mas não necessariamente perdendo flexibilidade.

2.1 - Convenções no CakePHP

Apesar de tomar um pouco de tempo para aprender as convenções do CakePHP, você ganha tempo em um longo processo: seguindo as convenções, você ganha funcionalidades gratuitamente e livra-se de madrugadas de manutenção de arquivos de configuração. Convenções também fazem com que o sistema fique uniformemente desenvolvido, permitindo que outros desenvolvedores o ajudem mais facilmente.

As convenções no CakePHP têm sido produzidas por anos de experiência em desenvolvimento web e boas práticas. Apesar de sugerimos que você use essas convenções enquanto desenvolve em CakePHP, devemos mencionar que muitos desses princípios são facilmente sobrescritos - algo que especialmente acontece quando trabalha-se com sistemas legados.

2.1.1 - Convenções para nome de arquivos e nome de classes

Na maioria das vezes, o nome dos arquivos utiliza *underscores*, enquanto o nome das classes utiliza o padrão CamelCase, ou seja, a primeira letra de cada palavra em maiúsculo. A classe LinksUteisController pode ser encontrada no arquivo links_uteis_controller.php, por exemplo.

Em alguns casos o nome da classe e o seu tipo não fazem parte do nome do arquivo. A classe CookieComponent pode ser encontrada no arquivo cookie.php e a classe SecurityComponent pode ser encontrada no arquivo security.php

2.1.2 - Convenções para a camada de modelo e banco de dados

Os nomes dos arquivos devem ser singular e separados por um *underscore* caso tenha mais de uma palavra. Ex.: usuario.php, saida_produto.php.

Os nomes das classes de modelo devem ser singular e CamelCased. Ex.: Usuario, SaidaProduto.

As tabelas de banco de dados devem estar no plural e com *underscore* caso tenha mais de uma palavra. Ex.: usuarios, saida_produtos.

A chave primária deve ter o nome id. Chaves estrangeiras devem ter o nome da tabela relacionada no singular e acrescentar no final _id. Logo se grupos têm usuários, na tabela usuarios deve existir o campo grupo_id.

Tabelas de ligação, utilizadas em relacionamento de N-M, devem possuir o nome das duas tabelas ligadas no plural separadas por *underscore* e em ordem alfabética. Ex.: usuarios HABTM produtos -> produtos_usuarios

2.1.3 - Convenções para a camada de controle

Os arquivos da camada de controle devem seguir o padrão:

Os nomes das classes de controle são no plural, CamelCased e sempre terminam em 'Controller'. Ex: UsuariosController, FuncionariosController e TerceirosController são alguns exemplos de *controllers*.

Nota

O primeiro método que você deve escrever em uma clase de controle é o método `index()`. Quando alguém requisita um *controller* sem ação, o comportamento padrão é renderizar o método `index()` do *controller*. Por exemplo, a requisição para `http://www.exemplo.com.br/usuarios` mapeia para a chamada do método `index()` do UsuariosController, assim como `http://www.exemplo.com.br/usuarios/view` mapeia para a chamada do método `view()` no UsuariosController.

Você também pode alterar a visibilidade dos métodos da classe de controle, para isto, basta colocar um *underscore* na frente do nome dos métodos. Se o método da classe de controle estiver com *underscore* na frente, o método não será disponibilizado para acesso *web* através do *Dispatcher*, mas estará disponível para uso interno. Por exemplo:

```
<?php

class NoticiasController extends AppController {

    function ultimas() {
        $this->_ultimasNoticias();
    }

    function _ultimasNoticias() {
        // Aqui entra a lógica para recuperar as últimas notícias
    }
}

?>
```

Listagem 12 - Exemplo de um método privado

Apesar da página `http://www.dominio.com.br/noticias/ultimas/` possa ser acessada pelo usuário normalmente, alguém que tente acessar a página `http://www.dominio.com.br/noticias/_ultimasNoticias/` receberá um erro porque o método é precedido de um *underscore*.

Como você acabou de ver, *controllers* de uma única palavra são mapeados facilmente para uma URL simples em letras minúsculas. Por exemplo, a classe UsuariosController (que seria definida em um arquivo de nome `usuarios_controller.php`) é acessada a partir da URL `http://www.dominio.com.br/usuarios`.

Controllers de com muitas palavras são mapeados para URLs em *camelBacked* mantendo a forma plural. Por exemplo, UsuariosLegadosController (`usuarios_legados_controller.php`) seria mapeado para `http://www.dominio.com.br/usuariosLegados` e SistemasExternosController (`sistemas_externos_controller.php`) seria mapeado para `http://www.dominio.com.br/sistemasExternos`.

2.1.4 - Convenções para a camada de visão

Os arquivos da camada de visão devem ser nomeados depois dos métodos dos *controllers*. Os arquivos da camada de visão devem ser escritos com *underscores*. O método getReady() da classe PeopleController irá procurar pela visão /app/views/people/get_ready.ctp.

O modelo básico é /app/views/nome_do_controller/nome_da_funcao_com_underscore.ctp.

Nomeando os pedaços da aplicação usando as convenções do CakePHP, você ganha funcionalidades sem luta e sem amarras de configuração. Aqui o exemplo final que vincula as associações:

- Tabela no banco de dados: 'people'
- Classe do Modelo: 'Person', encontrada em /app/models/person.php

- Classe do Controlador: 'PeopleController', encontrado em /app/controllers/people_controller.php
- Template da Visão: encontrado em /app/views/people/index.ctp

Usando estas convenções, CakePHP sabe que a requisição para http://www.exemplo.com.br/people/ mapeia para a chamada do método index() do PeopleController, onde o modelo Person é automaticamente disponibilizado (e automaticamente associado à tabela 'people' no banco de dados), e renderiza isso para o arquivo. Nenhuma destas relações foram configuradas por qualquer meio que não seja através da criação de classes e arquivos que você precise criar em algum lugar.

2.1.5 - Todas as Convenções

Local	Regra	Exemplo
Banco de dados	Nome das tabelas: plural, minúsculo.	usuarios, posts, artigos
Banco de dados	Chave primária: nome id, tipo int, auto_increment	id int(11)
Banco de dados	Nome dos campos: minúsculo, usando *underscore*.	nome, rua, data_de_nascimento
Banco de dados	Chave estrangeira: nome da tabela relacionada no singular, acrescentando *underscore* e id.	cliente_id,grupo_id
Banco de dados	Relacionamento N-M (HABTM), regra para a tabela de ligação: nome das tabelas em ordem alfabética, separadas por *underscore*.	produto_usuario, grupo_usuario
Modelo	Nome do arquivo: singular e CamelCase	usuario.php, grupo.php, cliente.php
Modelo	Nome da classe: singular e CamelCase	Usuario, Grupo
Controle	Nome do arquivo: plural, *underscores* e terminado com Controller	usuarios_controller.php, grupos_controller.php
Controle	Nome da classe: plural, CamelCase e terminado com Controller	UsuariosController, GruposController
Visão	Local do arquivo: dentro da pasta views/nome_do_controle/	views/usuarios/, views/grupos/
Visão	Nome do arquivo: deve ter o nome da ação do controle, com *underscores*.	views/usuarios/index.ctp, views/grupos/add.ctp

Tabela 4 - Lista de todas as convenções do CakePHP

2.2 - Constantes

As constantes listadas abaixo são utilizadas no CakePHP:

Constante	Descrição
APP	Diretório raiz.
APP_PATH	Diretório app.
CACHE	Diretório dos arquivos de cache.
CAKE	Diretório do cake.

Constante	Descrição
COMPONENTS	Diretório components.
CONFIGS	Diretório dos arquivos de configuração.
CONTROLLER_TESTS	Diretório de testes do controlador.
CONTROLLERS	Diretório dos controllers.
CSS	Diretório dos arquivos CSS.
ELEMENTS	Diretório dos elements.
HELPER_TESTS	Diretório de testes do helper.
HELPERS	Diretório helpers.
IMAGES	Diretório das imagens.
INFLECTIONS	Diretório inflections (geralmente dentro do diretório de configuração).
JS	Diretório dos arquivos JavaScript (no webroot).
LAYOUTS	Diretório layouts.
LIB_TESTS	Diretório de testes da biblioteca do CakePHP.
LIBS	Diretório das bibliotecas do CakePHP.
LOG_ERROR	Constante de Erro. Utilizada para diferenciar erro de log e debug. Atualmente o PHP suporta LOG_DEBUG.
LOGS	Diretório logs (no app).
MODEL_TESTS	Diretório de testes do modelo.
MODELS	Diretório models.
SCRIPTS	Diretório de scripts do Cake.

Constante	Descrição
TESTS	Diretório de testes (pai dos diretórios de teste modelos, controles, etc.)
TMP	Diretório tmp.
VENDORS	Diretório vendors.
VIEWS	Diretório views.
WWW_ROOT	Caminho completo para o webroot.

Tabela 5 - Lista das constantes no CakePHP

2.3 - A Arquitetura MVC

A medida que os *softwares* foram ficando maiores e mais complexos, tornou-se necessário organizar o código-fonte de forma a facilitar o seu entendimento.

Uma solução criada para este problema foi a divisão da aplicação em 3 camadas com funcionalidades distintas, está solução é denominada arquitetura MVC (Model - View - Controller).

Basicamente a arquitetura MVC separa o desenvolvimento da aplicação em:

- Camada de Modelo: parte da programação destinada ao tratamento dos dados.
- Camada de Controle: responsável pelas regras de negócio
- Camada de Visão: responsável pelo o que é apresentado para o cliente

A seguir, é apresentado o fluxo de execução de uma requisição feita ao CakePHP.

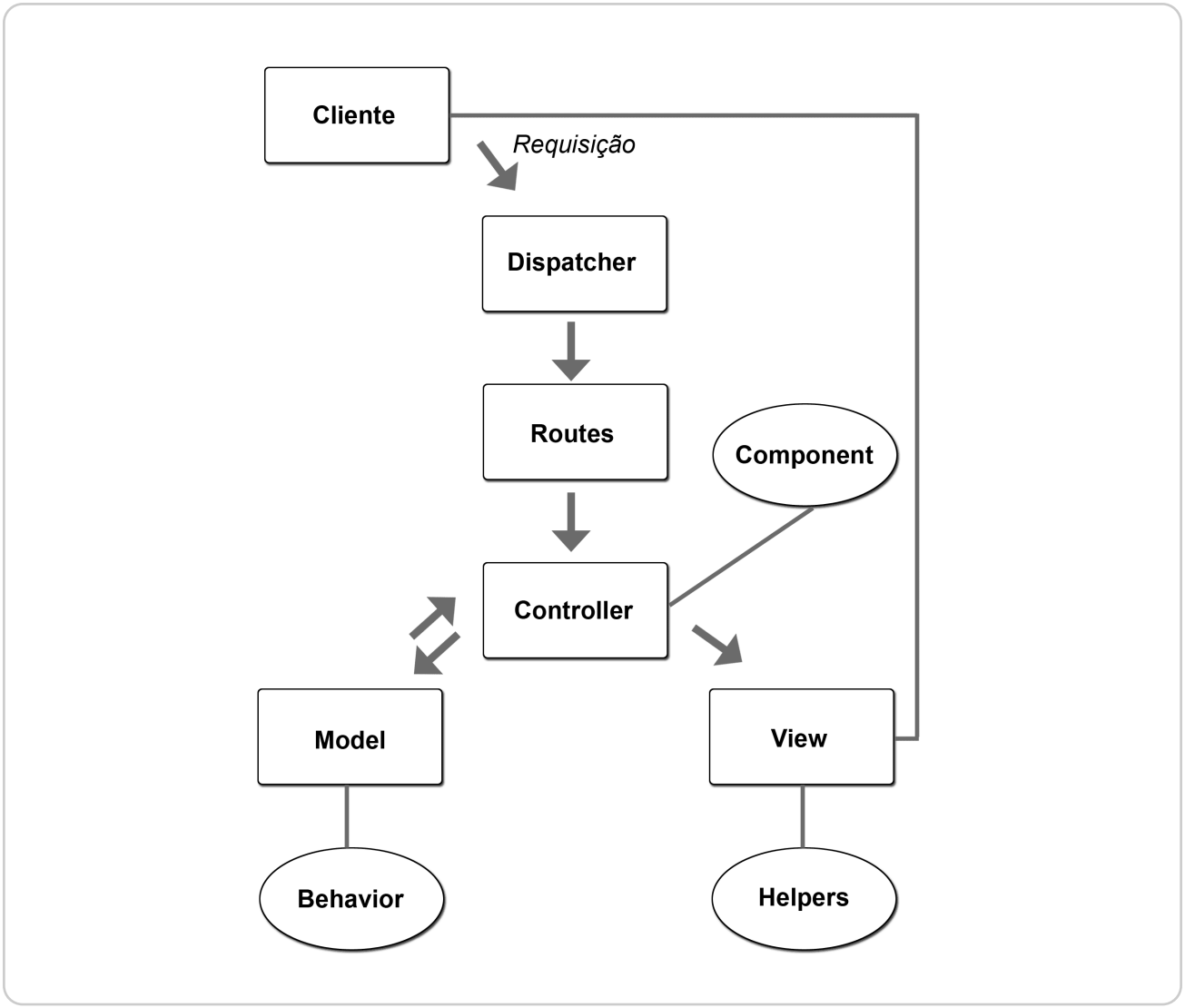


Imagem 1 - Fluxo da execução de uma requisição no CakePHP

Os passos da requisição podem ser descritos da seguinte forma:

- O cliente* faz uma requisição ao endereço http://www.exemplo.com.br/clientes/index
- O servidor de página encaminha a requisição para o Dispatcher.
- O Dispatcher verifica as regras de rotas.
- O *framework* executa o *controller* e a ação informados na requisição.
- Durante a execução da ação do *controller* são feitas requisições a camada de modelo.
- Ao final da execução uma view é renderizada.

Nota

O cliente é quem faz a requisição, ele pode ser um browser, um computador, etc.

2.3.1 - A camada de modelo

Essa camada é responsável por representar dados. Ela é utilizada para fornecer acesso aos dados, que podem estar armazenados em um banco de dados, um arquivo texto, um arquivo CSV, dentre outras possibilidades. De forma mais usual, geralmente um modelo específico representa uma tabela de um banco de dados.

Assim como podemos fazer relacionamentos entre tabelas em um banco de dados, um modelo também pode estabelecer esses relacionamentos entre modelos. Existem quatro tipos de relacionamento entre modelos: hasOne, hasMany, belongsTo ou hasAndBelongsToMany. Esses relacionamentos serão vistos com mais detalhe

no tópico 2.3.1.2 - Associando os modelos. Como exemplo prático de relacionamento entre modelos podemos citar a tabela Marca de Carro que possui relacionamento do tipo HasMany com a tabela Carros.

Em programação orientada a objeto podemos dizer que um modelo é um objeto que representa algo da realidade. Em outras palavras, um modelo pode ser a representação de uma televisão ou uma cama ou qualquer outro objeto. Para se definir um modelo no CakePHP deve se fazer o seguinte:

```
<?php
class Cliente extends AppModel {
        var $name = 'Cliente';
}
?>
```

Como se pode ver, neste exemplo, a classe Cliente herda da classe AppModel que por sua vez herda da classe Model. A princípio a classe AppModel está vazia. Ela existe para que seja possível gerar determinados comportamentos comuns a todos os modelos.

2.3.1.1 - Atributos dos modelos

2.3.1.1.1 - useDbConfig

A propriedade *useDbConfig* define, através de uma *string*, o nome da conexão com o banco de dados. Por padrão a conexão que o CakePHP usa é definida no arquivo app/config/database.php e tem o valor *default*. É necessário criar o array no arquivo database.php com os dados da conexão. Veja o exemplo a seguir:

```
<?php
  class DATABASE_CONFIG {
     var $default = array(
             'driver'               => 'mysql',
             'host'                 => 'localhost',
             'login'                => 'root',
             'password'             => 'root',
             'database'             => 'my_db'
     );
     var $teste = array(
             'driver'               => 'mysql',
             'host'                 => 'http://www.teste.com',
             'login'                => 'teste',
             'password'             => 'testando',
             'database'             => 'my_db'
     );
  }
?>
```

Listagem 13 - Código no arquivo app/config/database.php

```
<?php
class Cliente extends AppModel {

        var $name = 'Cliente';

        function beforeSave() {

                $this->useDbConfig = 'teste';
                return true;

        }
```

```
}
?>
```

Listagem 14 - Código no arquivo do modelo

Importante

Se você estiver utilizando dois ou mais bancos de dados na sua aplicação, certifique-se de utilizar contas de usuários diferentes para cada banco, pois no PHP se você utiliza a mesma conta para acessar bancos diferentes no mesmo servidor o PHP irá reutilizar a conexão.

2.3.1.1.2 - useTable

Este atributo define o nome da tabela que o modelo utilizará, somente existirá a necessidade de atribuir um valor, caso não esteja sendo utilizada as convenções. Se o modelo não necessitar usar tabela alguma, pode-se utilizar essa propriedade com o valor false.

```
<?php
class Teste extends AppModel {
        var $useTable = false;
}
?>
```

Listagem 15 - Exemplo de um model que não utiliza tabela

```
<?php
class Teste extends AppModel {
        var $useTable = 'testes';
}
?>
```

Listagem 16 - Exemplo de um model que utiliza tabela

2.3.1.1.3 - tablePrefix

Este atributo é utilizado para definir o prefixo de alguma tabela. Por padrão nenhum prefixo é definido. Pode-se também definir o prefixo no arquivo app/config/database.php, entretanto se for usado tablePrefix no model, essas definições sobrescrevem as definições definidas no app/config/database.php.

```
<?php
class Teste extends AppModel {
        var $tablePrefix = 'user_';
}
?>
```

Listagem 17 - Exemplo de uso da tablePrefix:

2.3.1.1.4 - primaryKey

Usualmente toda tabela possui o campo id definido como chave primária. Entretanto, existem casos onde a tabela pode não possuir um campo id, ou pretende-se usar outro campo como chave primária. Neste caso usamos o atributo primaryKey para definir qual campo funcionará dessa forma.

```
<?php
class Teste extends AppModel {
        var $primaryKey = 'email';
```

```
}
?>
```

Listagem 18 - Exemplo de uso da primaryKey:

2.3.1.1.5 - displayField

O displayField define qual campo da tabela será usado como rórulo (label). Esse rótulo é usado quando se utiliza scaffolding e find('list'). Por padrão a camada de modelo usa os campos name ou title. Não se pode usar mais que um valor para displayField. Portanto, não funcionará utilizar var $displayField = array('valor1', 'valor2').

```
<?php
class Teste extends AppModel {
        var $displayField = 'nome';
}
?>
```

Listagem 19 - Exemplo de uso da displayField:

2.3.1.1.6 - recursive

Recursive é utilizado quando o modelo possui relacionamento com outros modelos. Através dessa propriedade pode-se definir até onde o modelo deve trazer os relacionamentos quando existem chamadas para as funções find e read.

Como exemplo prático, imagine uma aplicação onde uma universidade possui diversos cursos, estes pussuem disciplinas, que têm professores. Logo, Universidade HasMany Curso; Curso HasMany Disciplina; Disciplina HasMany Artigo. Ao fazer uma busca na entidade Curso ($this->Curso->Find()), a busca pode se comportar das seguintes formas:

Profundidade	Descrição
-1	Cake procura somente os cursos.
0	Cake procura por cursos e a universidade ao qual os cursos pertencem.
1	Cake procura por cursos, universidades e disciplinas associados.
2	Cake procura por cursos, universidade, disciplinas associados e os professores associados aos produtos.

Tabela 6 - Valores possíveis para a variável `recursive`

2.3.1.1.7 - order

Define qual a ordem que os resultados das pesquisas feitas com find vão retornar. Pode ter o valor ASC, DESC, pode-se também ordenar por um campo, ou por um Modelo.campo. É possível ainda utilizar um array com mais de um critério de ordenação.

```
var $order = "nome";
var $order = "Cliente.nome";
var $order = "Cliente.nome asc";
var $order = "Cliente.nome ASC";
var $order = "Cliente.nome DESC";
```

```
var $order = array("Cliente.nome" => "asc", "Cliente.email" => "DESC")
```

Listagem 20 - Exemplos de uso de order

2.3.1.1.8 - data

Neste atributo é armazenado todos os dados retornados em uma pesquisa. Esses dados são armazenados em forma de arrays multidimensionais, que possuem os nomes dos campos como índices dos arrays.

2.3.1.1.9 - _schema

Este atributo armazena informações dos campos das tabelas do banco de dados. Cada campo armazena o seu nome, o tipo de dado que ele armazena (boolean, string, datetime, dentre outros), se permite valor null, o valor padrão que o campo recebe e o tamanho.

```
pr($this->Model->schema());
//retorna
var $_schema = array(
        'Nome' => array('type' => 'string', 'length' => 30),
        'Email' => array('type' => 'string', 'length' => 30),
        'Tel' => array('type' => 'int', 'length' => 10),
        'Comentário' => array('type' => 'text')
);
```

Listagem 21 - Exemplos de uso de _schema:

2.3.1.1.10 - validate

Através deste atributo é possível ao modelo fazer validações antes de armazenar os dados no banco de dados ou onde estes dados forem armazenados.

```
<?php
class Teste extends AppModel {
        var $validate = array(
                'nome' => array(
                        'rule'=>array('between',2,20),
                        'message'=>'O campo deve possuir de 2 a 20 caracteres.
                ),
                'email' => array(
                        'rule'=>array('email'),
                        'message'=>'O campo deve conter e-mail válido.'
                ),
                'tel' => array(
                        'rule'=>array('numeric'),
                        'message'=>'O campo deve ser numérico.'
                ),
                'comentario' => array(
                        'rule'=>array('between',1,500),
                        'message'=>'O campo deve possuir de 5 a 500 caracteres.'
                ),
        );
}
?>
```

Listagem 22 - Exemplo para uma validação com uma regra:

Os parâmetros do array validate

Chave	Descrição
Rule	Regra de validação do campo
Required	O campo precisa ter dados
allowEmpty	O campo aceita valor vazio
on	Define em qual lugar a validação deve ser feita. 'on'=>'create' ou 'on'=>'update'
message	Define a mensagem que será apresentada, caso o campo não passe na validação

Tabela 7 - Parâmetros do *validate*

Regras padrões de validação do CakePHP

Regra	Validação
alphaNumeric	Para o campo ser válido ele deve conter apenas caracteres alfanuméricos. **array('login' => array('rule' => 'alphaNumeric'));**
between	O campo pode ter qualquer caracter, mas o tamanho do conteúdo do campo estar entre a faixa determinada. **array('senha' => array('rule' => array('between',6,8)));**
blank	O campo deve estar em branco. **array('id' => array('rule' => 'blank', 'on'=>'create'));**
boolean	O campo deve ser um boleano válido. **array('flag' => array('rule' => 'boolean'));**
cc	O campo deve ser um cartão de crédito. **array('cartao' => array('rule' => array('cc',array('visa','maestro'))));**
comparison	Compara um valor inteiro, a comparação pode ser maior que (>), menor que (<), maior ou igual (>=), menor ou igual(<=), igual(==), ou diferente (!=). **array('anos' => array('rule' => array('comparison','>',20)));**
date	O campo precisa ser uma data válida. **array('data_inicio' => array('rule' => 'date'));**
decimal	O campo precisa ser um número decimal. **array('preco' => array('rule' => array('decimal',2)));**
email	O campo precisa ser um email. **array('email' => array('rule' => 'email'));**

Regra	Validação
equalTo	O campo precisa ser igual ao valor informado. **array('nome' => array('rule' => array('equalTo','josé')));**
extension	O arquivo precisa ser da extensão especificada. **array('arquivo' => array('rule' => array('extension',array('jpg','gif'))));**
file	O campo precisa ser um arquivo (upload). **array('documento' => array('rule' => 'file'));**
ip	O campo precisa ser um ip (versão 4). **array('client_ip' => array('rule' => 'ip'));**
isUnique	O campo deve ser único no banco de dados. **array('flag' => array('rule' => 'isUnique'));**
minLength	O campo precisa ter o tamanho mínimo informado. **array('senha' => array('rule' => array('minLength',8)));**
maxLength	O campo precisa ter o tamanho máximo informado. **array('nome' => array('rule' => array('maxLength',32)));**
money	O campo precisa ser um valor financeiro. **array('valor_da_compra' => array('rule' => 'money'));**
Multiple	Verifica campos select multiple. **array('tags' => array('rule' => array('multiple', array('in'=>array('teste', outro teste), 'min'=>1, 'max'=>3))));**
inList	O campo precisa estar na lista informada. **array('grupos' => array('rule' => array('inList', array('Admin','Usuario'))));**
numeric	O campo precisa ser um número válido. **array('valor' => array('rule' => 'numeric'));**
notEmpty	O campo não pode ser vazio. **array('texto' => array('rule' => 'notEmpty'));**

Regra	Validação
phone	O campo precisa ser um número de telefone válido. Para telefones fora do padrão americano é possível informar uma expressão regular no segundo parâmetro. **array('telefone' => array('rule' => array('phone',null,'us')));**
postal	O campo precisa ser um cep válida. Para ceps fora do padrão americano é possível informar uma expressão regular no segundo parâmetro. **array('cep' => array('rule' => array('postal',null,'us')));**
range	O campo precisa ser um valor entre os dois valores informados. **array('numero' => array('rule' => array('range',0,10)));**
ssn	O campo precisa ser um número do seguro social. Para números fora do padrão americano é possível informar uma expressão regular no segundo parâmetro. **array('flag' => array('rule' => array('ssn',null,'us')));**
url	O campo precisa ser uma url válida. **array('flag' => array('rule' => 'url'));**

Tabela 8 - Regras padrões de validação do CakePHP

2.3.1.1.10.1 - Validação Avançada

Usando uma regra customizada.

```
var $validate = array(
        'login' => array(
                'rule' => array('custom', '/^[a-z0-9]{3,}$/i'),
                'message' => 'Somente letras e números são aceitos, mínimo de 3
caracteres'
        )
);
```

Listagem 23 - Exemplo de uma regra customizada utilizando expressão regular

```
var $validate = array(
        'codigoPromocional' => array(
                'rule' => array('limiteDeUso', 250),
                'message' => 'Código informado foi usado mais vezes do que o
permitido.'
        )
);
function limiteDeUso($data, $limite){
        $contador = $this->find('count', array('conditions' => $data, 'recursive' =>
-1));
        return $contador < $limite;
}
```

Listagem 24 - Exemplo de uma regra customizada utilizando uma função própria

```
var $validate = array(
        'login' => array(
                'alphanumeric' => array(
                        'rule' => 'alphaNumeric',
                        'message' => 'Somente alfanuméricos são aceitos'
                ),
                'between' => array(
                        'rule' => array('between', 4,8),
                        'message' => 'O campo deve possuir entre 4 e 8 caracteres'
                ),
                'isUnique' => array(
                        'rule' => array('isUnique'),
                        'message' => 'O login já existe em nossa base de dados'
                ),
        )
);
```

Listagem 25 - Usando mais de uma validação no mesmo campo

2.3.1.1.11 - name

Este atributo permite que o código do CakePHP na camada de modelo seja compatível com o php4. Portanto, é recomendável sempre utilizar esta declaração já que pode ocorrer da aplicação ter que mudar para um servidor que utiliza o php nessa versão.

```
<?php
class Teste extends AppModel {
        var $name = 'Teste';
}
?>
```

Listagem 26 - Exemplo de uso de name:

2.3.1.1.12 - cacheQueries

Se é atribuído true para essa propriedade, os dados retornados pelo model são armazenados em cache. Esses dados ficam em cache durante a requisição.

2.3.1.2 - Associando os modelos

2.3.1.2.1 - hasOne

Esse tipo de relacionamento é usado quando uma tabela pode possuir apenas um resultado relacionado em outra tabela. Por exemplo, em uma aplicação de comunidade virtual onde se tem uma tabela de Usuário e outra de Perfil do Usuário. Cada usuário pode ter apenas um perfil. Neste caso o relacionamento entre a tabela Usuário e Perfil de Usuário é do tipo hasOne.

Para que esse relacionamento funcione corretamente, deve-se definir uma chave estrangeira na tabela Perfil de Usuário. Por convenção do CakePHP a chave estrangeira, neste exemplo, deveria se chamar usuario_id.

Depois de estruturado corretamente o banco de dados, deve-se definir o tipo de relacionamento no modelo Usuário, que deverá ser salvo no arquivo app/models/usuario.php.

```
<?php
class Usuario extends AppModel {
        var $name = 'Usuario';
        var $hasOne = 'Perfil';
}
```

```
?>
```

Listagem 27 - Exemplo de uso do hasOne

O exemplo acima é a forma mais simples de se estabelecer o relacionamento do tipo hasOne. Existem casos onde é necessário ter mais controle sobre o relacionamento estabelecido. Veja o exemplo seguinte:

```
<?php
class Usuario extends AppModel {
        var $name = 'Usuario';

        var $hasOne = array(
                'Perfil' => array(
                        'className'    => 'Perfil',
                        'conditions'   => array('Perfil.publicado' => '1'),
                        'dependent'    => true
                )
        );

}
?>
```

Listagem 28 - Exemplo de uso avançado do hasOne

O array das associações do tipo hasOne podem conter as seguintes chaves:

Chave	Descrição
className	O nome da classe utilizada pelo modelo relacionado. Se o relacionamento é com o modelo Perfil, o nome de className deverá ser Perfil.
foreignKey	O nome da chave estrangeira contida no modelo relacionado. Se a chave estrangeira na tabela Perfil é usuario_id, então este deverá ser o nome contido em foreignKey.
conditions	Em conditions pode-se introduzir fragmentos da query para filtrar os dados no modelo relacionado. É bom utilizar o nome do modelo antes do nome do campo. Exemplo: 'Perfil.publicado = 1'
fields	Pode-se definir quais campos da tabela serão recuperadas. Por padrão todos os campos são recuperados.
dependent	Se *dependent* contiver o valor *true* e for chamado o método *delete()* com o parâmetro *cascade*, então todos os dados relacionados serão apagados. Exemplo: Se for apagar um usuário dessa forma, o seu respectivo perfil será apagado também.

Tabela 9 - Chaves da associação *hasOne*

2.3.1.2.2 - hasMany

Seguindo nosso exemplo, o Usuário de nossa comunidade virtual pode ter diversos artigos. Portanto a tabela Usuário estabelece um relacionamento do tipo hasMany com a tabela Artigos.

Neste caso a chave estrangeira deverá se encontrar na tabela Artigos. Seguindo o padrão do CakePHP o nome da chave estrangeira deverá ser Artigo.usuario_id.

```
<?php
class Usuario extends AppModel {
        var $name = 'Usuario';
        var $hasMany = 'Artigo';
}
?>
```

Listagem 29 - Exemplo de uso de hasMany

```
<?php
class Usuario extends AppModel {
        var $name = 'Usuario';
        var $hasMany = array(
                'Artigo' => array(
                        'className'    => 'Artigo',
                        'foreignKey'    => 'usuario_id',
                        'conditions'    => array('Artigo.publicado' => '1'),
                        'order'    => 'Artigo.created DESC',
                        'limit'        => '20',
                        'dependent'=> true
                )
        );
}
?>
```

Listagem 30 - Exemplo avançado de uso de hasMany

O array das associações do tipo hasMany podem conter as seguintes chaves:

Chave	Descrição
className	O nome da classe que está associada ao modelo atual. Se Usuário possui muitos Artigos, o nome de className deverá ser Artigo.
foreignKey	O nome da chave estrangeira contida no modelo relacionado. Se a chave estrangeira na tabela Artigo é usuario_id, então este deverá ser o nome contido em foreignKey.
conditions	Em conditions pode-se introduzir fragmentos da query para filtrar os dados no modelo. É bom utilizar o nome do modelo antes do nome do campo. Exemplo: 'Artigo.publicado = 1'
fields	Pode-se definir quais campos da tabela serão recuperadas. Por padrão todos os campos são recuperados.
order	Fragemento de código que permite definir o tipo de ordenação dos resultados.
limit	O número máximo de resultados retornados.
offset	O número de linhas associadas que devem ser excluidas do resultado.

Chave	Descrição
dependent	Se for associado o valor true possibilita apagar os resultados em cascata. Ou seja, as linhas gravadas em outras tabelas que possuem relacionamento com a tabela deste modelo também serão apagadas.
exclusive	Se for associado o valor true ao apagar o registro irá ser invocado o método *deleteAll()*. Existe ganho de performance, mas em certas ocasiões não é a forma ideal a ser utilizada.
finderQuery	Pode-se fazer uma query inteira usando esta chave. Utilizado quando é necessário obter resultados muito específicos.

Tabela 10 - Chaves da associação *hasMany*

2.3.1.2.3 - belongsTo

Depois de estabelecido o relacionamento do tipo HasOne ou HasMany podemos estabelecer o relacionamento belongsTo na tabela relacionada. Esse passo pode ser importante se pretendemos recuperar dados a partir dos registros da tabela que é relacionada à tabela que estabeleceu o hasOne ou HasMany. Seguindo o exemplo exposto em hasOne, a tabela Perfil do Usuário pertence a tabela Usuário. Ou seja, um perfil inscrito na tabela Perfil do Usuário pertence a um usuário pertencente à tabela Usuário.

No banco de dados deve-se seguir a mesma convenção usada no hasOne e hasMany. Ou seja, a chave estrangeira deverá estar na tabela que irá estabelecer o relacionamento belongsTo.

```
<?php
class Perfil extends AppModel {
        var $name = 'Perfil';
        var $belongsTo = 'Usuario';
}
?>
```

Listagem 31 - Exemplo de uso de belongsTo

```
<?php
class Perfil extends AppModel {
        var $name = 'Perfil';
        var $belongsTo = array(
                'Usuario' => array(
                        'className'    => 'Usuario',
                        'foreignKey'    => 'usuario_id'
                )
        );
}
?>
```

Listagem 32 - Exemplo avançado de uso de belongsTo

O array das associações do tipo belongsTo podem conter as seguintes chaves:

Chave	Descrição
className	O nome da classe que está associada ao modelo atual. Se Perfil pertence à classe Usuário, o nome de className deverá ser Usuário.

Chave	Descrição
foreignKey	O nome da chave estrangeira contida no modelo atual. Se a chave estrangeira na tabela Perfil é usuario_id, então este deverá ser o nome contido em foreignKey.
conditions	Em conditions pode-se introduzir fragmentos da query para filtrar os dados no modelo. É bom utilizar o nome do modelo antes do nome do campo. Exemplo: 'Perfil.publicado = 1'
fields	Pode-se definir quais campos da tabela serão recuperadas. Por padrão todos os campos são recuperados.
counterCache	Se estiver associado o valor true irá aumentar ou diminuir o valor em *[singular_model_name]_count* quando se for salvar ou apagar um registro. O valor desse campo representa o número de registros relacionados. Se um Usuário possui 10 amigos, então Usuario.amigo_count = 10.

Tabela 11 - Chaves da associação *belongsTo*

2.3.1.2.4 - hasAndBelongsToMany (HABTM)

Quando um modelo pode ter diversos resultados associados de outro modelo e vice-versa é necessário utilizar o relacionamento do tipo hasAndBelongsToMany. Neste caso, será necessário criar uma tabela intermediária na estrutura do banco de dados. O nome dessa nova tabela deverá conter o nome das duas tabelas relacionadas em ordem alfabética e separadas por *underscore* (_). Deverá existir dois campos nessa tabela do tipo inteiro. Esses campos deverão ser as chaves estrangeiras das duas tabelas. É recomendável adicionar também um campo id para esta tabela, assim mantém-se o padrão nesta tabela igual o padrão de todas.

O nome das chaves estrangeiras deve seguir o padrão do CakePHP. Em nosso exemplo, um usuário pode pertencer em diversas comunidades. E, ao mesmo tempo, uma comunidade pode possuir diversos usuários. Então o nome da nossa tabela seria comunidades_usuarios e o nome dos campos na tabela intermediária seria respectivamente id, comunidade_id e usuario_id.

Depois de criado toda a estrutura de banco de dados, passamos para o código no modelo.

```
<?php
class Usuario extends AppModel {
        var $name = 'Usuario';
        var $hasAndBelongsToMany = array(
                'Comunidade' => array(
                        'className' => 'Comunidade',
                        'joinTable' => 'usuarios_comunidades',
                        'with' => '',
                        'foreignKey' => 'usuario_id',
                        'associationForeignKey' => 'comunidade_id',
                        'unique' => true,
                        'conditions' => '',
                        'fields' => '',
                        'order' => '',
                        'limit' => '',
                        'offset' => '',
                        'finderQuery' => '',
                        'deleteQuery' => '',
                        'insertQuery' => ''
                )
```

```
        );
}
?>
```

Listagem 33 - Exemplo de hasAndBelongsToMany

O array das associações do tipo hasAndBelongsToMany podem conter as seguintes chaves:

Chave	Descrição
className	O nome da classe que está associada ao modelo atual. Se Comunidade está relacionada à classe Usuário, o nome de className deverá ser Usuário.
joinTable	O nome da tabela intermediária que possui as chaves estrangeiras.
with	O nome do modelo utilizado para fazer a junção entre os modelos. Por padrão o CakePHP já cria este modelo automaticamente. Neste caso, o nome seria ComunidadeUsuario. Caso se deseja utilizar outro modelo para desempenhar este papel, pode-se colocar nesta chave o nome deste modelo.
foreignKey	O nome da chave estrangeira contida no modelo atual. Torna-se útil quando se pretende estabelecer mais de um relacionamento HABTM no mesmo modelo.
associationforeignKey	O nome da chave estrangeira contida no modelo relacionado. Torna-se útil quando se pretende estabelecer mais de um relacionamento HABTM no mesmo modelo.
unique	Se é associado o valor *true* a essa chave o CakePHP irá apagar os relacionamentos entre os dados antes de inserir novamente em uma chamada de *update*.
conditions	Em conditions pode-se introduzir fragmentos da query para filtrar os dados no modelo. É bom utilizar o nome do modelo antes do nome do campo. Exemplo: 'Comunidade.ativa = 1'
fields	Pode-se definir quais campos da tabela serão recuperadas. Por padrão todos os campos são recuperados.
order	Fragemento de código que permite definir o tipo de ordenação dos resultados.
limit	O número máximo de resultados retornados.
offset	O número de linhas associadas que devem ser excluidas do resultado.
finderQuery, deleteQuery, insertQuery	Pode-se fazer uma query inteira usando esta chave. Essa query pode ser de busca, deleção ou inserção. Utilizado quando é necessário obter resultados muito específicos.

Tabela 12 - Chaves da associação *hasAndBelongsToMany*

2.3.1.3 - Recuperando dados

2.3.1.3.1 - find

find($type, $params)

$type - pode receber os valores 'all', 'first', 'list', 'neighbors', 'count' ou 'threaded'. O valor padrão utilizado é o 'first'.

$params - trata-se de um array que pode possuir os seguintes parâmetros:

```
<?php
array(
        //array de condições
        'conditions' => array('Modelo.campo' => $thisValue),

        //valor inteiro
        'recursive' => 1,

         //array contendo o nome dos campos da tabela a ser recuperadas
        'fields' => array('Modelo.campo1', 'Modelo.campo2'),

        //string ou array que define a ordem dos resultados
        'order' => array('Modelo.created', 'Modelo.campo3 DESC'),

        //campos para serem usados no GROUP BY
        'group' => array('Modelo.campo'),

        //valor inteiro
        'limit' => n,

        //valor inteiro
        'page' => n,

        //outros valores possíveis são false, before e after
        'callbacks' => true
)
?>
```

Listagem 34 - Parâmetros do array `$params`

Se estiver sendo usado $type do tipo list, o campo $key no array $params irá definir o índice e o valor.

```
<?php

// gera uma lista contendo Usuario.id como índice e Usuario.nome como valor
$this->Post->find('list', array('fields'=>'Usuario.nome'));

// gera uma lista contendo Usuario.email como índice e Usuario.nome como valor
$this->Post->find('list', array('fields'=>array('Usuario.email', 'Usuario.nome')));

// gera uma lista agrupada por Usuario.comunidade_id e cada grupo com Usuario.id como
índice e Usuario.nome como valor
$this->Post->find('list', array('fields'=>array('Usuario.id', 'Usuario.nome',
'Usuario.comunidade_id')));

?>
```

Listagem 35 - Exemplos do `$type` *list*

2.3.1.3.2 - findBy

findBy<fieldName>(string $value)

Através deste método é possível procurar por certo campo da tabela. Para isso basta adicionar o nome do campo no formato *CamelCase* logo após o findBy. O parâmetro $value deve ser uma *string* com o critério da busca.

```
<?php
$this->Usuario->findByNomeCompleto('José da Silva');
?>
```

2.3.1.3.3 - findAllBy

findAllBy<fieldName>(string $value)

Através deste método é possível procurar por certo campo da tabela. Para isso basta adicionar o nome do campo no formato *CamelCase* logo após o findAllBy. O parâmetro $value deve ser uma *string* com o critério da busca.

```
<?php
$this->Usuario->findAllByComunidade('Hackers');
?>
```

2.3.1.3.4 - query

Através desse método do Modelo é possível fazer requisições ao banco de dados de forma customizada. Neste caso é recomendável utilizar a biblioteca Sanitize para prevenir ataques do tipo injection e cross-site scripting.

```
<?php
$this->Produto->query("SELECT preco, cor FROM produto LIMIT 100;");
?>
```

2.3.1.3.5 - field

field(string $name, array $conditions = null, string $order = null)

Esse método retorna um único resultado do campo especificada pelo parâmetro *$name*. Se for passado alguma condição em *conditions* será retornado o primeiro resultado filtrado pela condição. Pode-se utilizar $order para definir o tipo de ordenação do resultado. Se for setado o id no modelo, é retornado o resultado do id escolhido. Se nenhum resultado seja trazido, esse método retorna *false*.

```
<?php
$produto->id = 22;
echo $produto->field('name');

echo $usuario->field('name', array('sexo = masc'), 'idade DESC');
?>
```

2.3.1.3.6 - Construindo condições complexas

Existem casos onde será necessário recuperar dados com condições mais complexas. Muitas vezes o uso de *arrays* pode ajudar a simplificar e facilitar a manipulação dos dados.

```
<?php
// Separando a condição do find
$condicoes = array("Carro.cor" => "Vinho");
$this->Carro->find('all', array('conditions'=> $condicoes));

// Para encontrar um resultado diferente do especificado
$condicoes = array("Carro.cor <>" => "Vinho");
$this->Carro->find('all', array('conditions'=> $condicoes));

// Para encontrar mais de um resultado
$condicoes = array(
        "Comentario.titulo" => array("Sem comentários.", "Muito bem. Parabéns.")
)
$this->Comentario->find('all', array('conditions'=> $condicoes));

// Para encontrar resultados que não encaixam nas condições estipuladas
$condicoes = array(
        "NOT" => array("Comentario.titulo" => array(
                "Sem comentários.",
                "Muito bem. Parabéns.",
                "Muito mal."))
)
$this->Comentario->find('all', array('conditions'=> $condicoes));

//Para encontrar resultados com mais de uma condição
$condicoes = array(
        "Comentario.titulo" => array("Sem comentários.", "Muito bem. Parabéns."),
        "Comentario.autor" => array("Maria", "José")
)
$this->Comentario->find('all', array('conditions'=> $condicoes));

//Comparar os valores de dois campos
$condicoes = array("Comentario.criado = Comentario.modificado");
$this->Comentario->find('all', array('conditions'=> $condicoes));

//Condições com OR
$condicoes = array( "or" =>
        array (
                "Comentario.titulo" => array("Olá pessoal.", "Nunca mais volto
aqui."),
                "Comentario.criado >" => date('Y-m-d', strtotime("-2 weeks"))
        )
);
$this->Comentario->find('all', array('conditions'=> $condicoes));

//Condições AND e OR na mesa cláusula
$condicoes = array(
        "Author.nome" => "Daniel",
        "or" => array (
                "Comentario.titulo LIKE" => "gostei",
                "Comentario.criado >" => date('Y-m-d', strtotime("-2 weeks"))
        )
);
$this->Comentario->find('all', array('conditions'=> $condicoes));
```

```
//Condição BETWEEN
$condicoes = array('Comentario.id BETWEEN ? AND ?' => array(23,54));
$this->Comentario->find('all', array('conditions'=> $condicoes));

//Usando GROUP BY
$condicoes = array('fields' => array('Produto.tipo','MIN(Produto.preco) as preco'),
'group' => 'Produto.tipo');
$this->Produto->find('all', $condicoes);

//Condição complexa
$condicoes = array(
        'OR' => array(
                array('Categoria.nome' => 'Eletrodomésticos'),
                array('Categoria.nome' => 'Informática')
        ),
        'AND' => array(
                array(
                        'OR'=>array(array('Categoria.ativa' => 'true')),
                        'NOT'=>array(array('Categirua.ativa'=> 'inativa'))
                )
        )
);
$this->Categoria->find('all', array('conditions'=> $condicoes));

?>
```

2.3.1.4 - Salvando dados

2.3.1.4.1 - save

save(array $data = null, boolean $validate = true, array $fieldList = array())

$data: array contendo os dados a serem salvos, tendo como índice os nomes dos campos onde estes dados deverão ser salvos.

$boolean: se for atribuido o valor *true*, o método save irá validar os dados antes de salvá-lo. As regras de validação podem ser montadas no array $validate no próprio Modelo.

$fieldList: os campos onde os dados deverão ser salvos.

Este método possui uma sintaxe alternativa:

save(array $data = null, array $params = array())

No parâmetro $params é possível passar as seguintes opções:

- validate: true ou false
- fieldList: array com o nome dos campos onde os dados serão salvos
- callbacks: true, false, after ou before

Depois que os dados foram salvos, pode-se acessar o valor do id pode ser recuperado no atributo $id do objeto do modelo.

2.3.1.4.2 - saveAll

saveAll(array $data = null, array $options = array())

Esse método pode ser utilizado para salvar dados de um modelo ou, além de salvar os dados do modelo, gravar os dados nos seus modelos relacionados.

No parâmetro *$options* pode ser usado os seguintes opções:

Chave	Descrição
validate	Se atribuido *false* não ocorre validação. Se atribuído *true*, cada registro é validado antes de serem salvos. Se atribuído *first* todos os dados são validados antes de ser salvo. Se for atribuído *only* os dados são apenas validados e não salvos.
atomic	Se for atribuído *true* o CakePHP irá tentar salvar todos os registros em uma transação única. Se o banco de dados não der suporte a transação única, deverá ser usado false. O valor padrão é true.
fieldList	array com o nome dos campos onde os dados serão salvos

Tabela 13 - Chaves do parâmetro `$options`

2.3.1.4.3 - create

create(array $data = array())

Esse método é utilizado para gravar um novo registro. Se, ao pedir para criar um novo registro com esse método, o modelo estiver com o parâmetro $data populado com dados, então estes serão utilizados para criar o novo registro.

2.3.1.4.4 - saveField

saveField(string $fieldName, string $fieldValue, $validate = false)

Esse método do modelo é utilizado para salvar uma linha em um único campo do banco de dados. Para utilizar esse método é necessário antes atribuir o ID ao modelo. O parâmetro fieldName deve conter apenas o nome do campo do banco de dados. Veja a seguir um exemplo de uso de *saveField*:

```
<?php
        $this->Usuário->saveField('nome', 'José da Silva');
?>
```

Listagem 36 - Exemplo de uso do método saveField()

2.3.1.4.5 - updateAll

updateAll(array $fields, array $conditions)

Atualiza diversos registros em uma só operação. Os registros a serem atualizados deverão ser identicados no parâmetro *$conditions* e os seus valores no parâmetro *$fields*.

```
<?php
        $this->Produto->updateAll(
                array('Produto.disponivel' => "indisponivel"),
                array('Produto.categoria_id' => 76)
        );
?>
```

Listagem 37 - Exemplo de uso do método updateAll()

2.3.1.4.6 - counterCache

Este método ajuda a ter um contador dos registros de modelos relacionados ao modelo atual. O nome do campo será o nome do modelo seguido de *underscore* (_) seguido de count. Por exemplo, se você possui uma tabela chamada BlogComentarios e outra chamada Blog. Neste caso, deveria-se criar um campo na tabela Blog deveria ser criado um campo do tipo inteiro com o nome blog_comentario_count. Uma vez que o campo tenha sido inserido na tabela então pode-se utilizar o *counterCache* no array $belongsTo e atribuir o valor true. Veja o exemplo a seguir:

```
<?php
class Blog extends AppModel {
        var $hasMany = array('BlogComentario');
}

class BlogComentario extends AppModel {
        var $belongsTo = array('Blog' => array('counterCache' => true));
}
?>
```

Listagem 38 - Exemplo de uso do atributo counterCache

2.3.1.5 - Apagando dados

2.3.1.5.1 - del

delete(int $id = null, boolean $cascade = true);

Apaga o registro identificado pelo parâmetro $id. Por padrão apaga os registros relacionados a ele. Se uma categoria de produto for apagada, por padrão o CakePHP irá apagar todos os produtos relacionados à essa categoria.

2.3.1.5.2 - deleteAll

deleteAll(mixed $conditions, $cascade = true, $callbacks = false)

Funciona igual a del e remove. A única diferença é que pode-se estabelecer condições, no formato de fragmentos de uma query, no parâmetro $conditions.

2.3.1.6 - Callbacks

2.3.1.6.1 - beforeFind

beforeFind(mixed $queryData)

Esse *callback* é chamado antes que alguma operação de find seja chamada. O parâmetro *$queryData* contém informações sobre a *query* atual (condições, campos, etc.). Se este método retornar *false* a operação de busca não irá ocorrer.

2.3.1.6.2 - afterFind

afterFind(array $results, bool $primary)

Esse *callback* é utilizado para introduzir lógica após uma operação de find. É importante para tratar dados como alterar datas ou valores retornados pela busca. Os resultados da busca podem ser acessados através do parâmetro $results. Este métodof deverá retornar o resultado da busca com os valores alterados.

Se o parâmetro *$primary* for false o resultado da busca vai ser armazenado no array results de forma diferente. Veja a diferença a seguir:

```
<?php
// Resultados retornados com o parâmetro $primary = true
$results = array(
```

```
        0 => array(
                'NomeDoModelo' => array(
                        'campo1' => 'valor1',
                        'campo2' => 'valor2'
                )
        )
);

// Resultados retornados com o parâmetro $primary = false
        $results = array(
                'campo_1' => 'valor1',
                'campo_2' => 'valor2'
);
?>
```

Listagem 39 – Exemplo de uso do parâmetro `$primary`

2.3.1.6.3 – beforeValidate

beforeValidate()

Esse callback é utilizado para alterar alguma validação ou inserir lógica antes da validação ocorrer. Se esse *callback* retornar *false* a validação não irá ocorrer.

2.3.1.6.4 – beforeSave

beforeSave()

Esse *callback* é utilizado para inserir lógica antes do dados serem salvos e logo após os dados serem validados. É importante para alterar algum formato de data que precise ser armazenado de forma diferente no banco de dados. Veja o exemplo a seguir:

```
<?php
function beforeSave() {
        if(!empty($this->data['Curso']['dtInicio']) &&
                !empty($this->data['Curso']['dtFim'])) {

                $this->data['Curso']['dtInicio'] = $this->dtAntes(
                        $this->data['Curso']['dtInicio']);
                $this->data['Curso']['dtFim'] = $this->dateFormat(
                        $this->data['Curso']['dtFim']);
        }
return true;
}

function dtAntes($data) {
        return date('Y-m-d', strtotime($data));
}
?>
```

Listagem 40 – Exemplo de uso do método beforeSave()

2.3.1.6.5 – afterSave

afterSave(boolean $created)

Callback utilizado para inserir lógica depois que um registro foi salvo. O parâmetro *$created* irá ser *true* se um novo registro foi criado, será false se tiver ocorrido uma operação de atualização.

2.3.1.6.6 - beforeDelete

beforeDelete()

Se for necessário inserir alguma lógica antes de apagar um registro, esta deve ser posta neste *callback*. Ela deverá retornar o valor *true* se pretende-se que a deleção ocorra e *false* em caso contrário.

2.3.1.6.7 - afterDelete

afterDelete()

Se for necessário inserir qualquer lógica depois de apagar um registro, esta lógica deve ser inserida nesse *callback*.

2.3.1.6.8 - onError

onError()

Se algum erro ocorrer esse *callback* é chamado.

2.3.2 - A camada de controle

A camada de controle é usada para gerenciar a lógica da sua aplicação. Na maioria das vezes um controle irá gerenciar a lógica de um modelo. O nome dos controles deverão ser compostos pelo nome do modelo no plural seguidos de *underscore* (_) e a palavra "*controller*". Os arquivos de controle devem ser salvos dentro do diretório *app/controllers/*.

Todo controle herda da classe *appController* que por sua vez herda da classe *Controller*. A classe *appController* encontra-se no arquivo app/app_controller.php. Os métodos criados nesta classe ficam disponíveis para todos os controles da aplicação. A classe *Controller*, por sua vez, é uma biblioteca padrão do CakePHP.

Cada método de um controller é tido como uma ação. As ações são usadas para gerar as telas na camada de Visão. O usuário final escreve na url o seguinte caminho: nomeDoModelo/Ação/Parâmetro1/Parâmetro2... Assim o *dispatcher* do CakePHP pega essa url e traduz em utilizar o método do controle, passando os determinados parâmetros.

```
<?php
class UsuarioController extends AppController {
        function view($id) {
                //ações desse método vem aqui
        }
        function edit($id = null) {
                //ações desse método vem aqui
    }
        function delete($id) {
                //ações desse método vem aqui
        }
}
?>
```

Listagem 41 - Exemplo de um arquivo de controle

2.3.2.1 - Atributos da camada de Controle

2.3.2.2.1 - name

Este parâmetro deve se usado para garantir que a aplicação seja compatível com php versão 4. O valor de *$name* deve ser igual ao nome do controle.

Listagem 42 - Exemplo de declaração de name

```
<?php
        class NoticiasController extends AppController {
                var $name = 'Noticias';
        }
?>
```

2.3.2.2.2 - components

Esse atributo é utilizado quando se precisa usar um componente no controle. Os componentes são bibliotecas do CakePHP que trazem funcionalidades à camada de controle.

```
<?php
        var $components = array('Email');
?>
```

2.3.2.2.3 - helpers

Esse atributo é utilizado quando se precisa usar um helper no controle. Geralmente helpers são bibliotecas do CakePHP que trazem funcionalidades na camada de visão. Embora sua importância resida na camada de visão, eles devem ser importados na camada de controle.

```
<?php
        var $helpers = array('Ajax');
?>
```

2.3.2.2.4 - uses

As vezes existe a necessidade de se usar outros modelos que não aquele relacionado com o *controller*. Nestes casos, é necessário utilizar esse atributo para selecionar um ou mais modelos diferentes.

```
<?php
        var $uses = array('Usuarios', 'Perfis');
?>
```

2.3.2.2.5 - layout

Esse atributo é utilizado para selecionar qual *view* (tela de apresentação gerada na camada de visão) será utilizada. Neste caso, o CakePHP irá procurar no diretório */app/views/layouts* pelo arquivo atribuído a esse atributo. O nome que deve ser atribuido a *layout* deverá ser o nome do arquivo menos a sua extensão, no caso *.ctp*. Caso não seja utilizado esse atributo, o CakePHP irá procurar por */app/views/layouts/default.ctp*.

```
<?php
class NoticiasController extends AppController {
        function apresenta() {
                $this->layout = 'index';
        }
}
?>
```

2.3.2.2.6 - params

$this->params

Esse atributo fornece acesso aos parametros da requisição atual. É muito utilizado para obter informações sobre informações que foram enviadas através dos métodos *POST* ou *GET*.

Parâmetro	Descrição
form	Qualquer dado postado via método POST em qualquer formulário é armazenado aqui. Inclui dados encontrados também em $_FILES.
admin	Se contiver o valor 1 a ação atual foi feita através da rota admin.
bare	Armazena o valor 1 se o layout estiver vazio, caso contrário guarda o valor 0.
isAjax	Se a requisição atual é do tipo Ajax, é armazenado o valor 1. Caso contrário é armazenado o valor 0. Essa variável só é utilizada quando o componente RequestHandler é utilizado no controller.
controller	Armazena o nome do controller utilizado pela requisição atual.
action	Armazena o nome da ação utilizada pela requisição atual.
pass	Armazena a string com as variáveis passadas via GET.
url	Armazena a url atual que está sendo utilizada. É composto de um array, onde a url fica armazenada na chave 'url' e as variáveis enviadas via GET ficam armazenadas nas chaves 'var1', 'var2', e assim por diante.
data	Utilizado para manipular os dados enviados via POST pelo formulário de FormHelper.
prefix	Armazena o valor do prefixo utilizado na rota. Um exemplo de prefixo é o 'admin'.
named	Armazena qualquer parâmetro enviado na url no formato chave:valor. Esses dados são armazendos em um array, podendo ser enviados diversos parãmetros.

Tabela 14 - Chaves do atributo `$params`

2.3.2.2.7 - data

$this->data

Armazena os dados enviadas da camada de visão para a camada de controle através do *FormHelper*.

```
<?php
class UsuariosController extends AppController {
        function apresenta() {
                if(isset($this->data['Usuario']['nome'])) {
                        echo $this->data['Usuario']['nome'];
```

```
                }
        }
}
?>
```

2.3.2.2 - Métodos da camada de Controle

2.3.2.2.1 - set

set(string $var, mixed $value)

Através desse método é possível enviar dados para a camada de visão. Também é possível passar um array como parâmetro. Por exemplo: $this->set($nomes).

```
<?php
//Pega os dados do usuário retornado pela pesquisa e associa à variável $usuario,
//que ficará acessível na camada de visão.
$this->set('usuario', $this->data['Usuario']);
?>
```

Importante

Para alterar o título da página na visão, você deve definir o valor da variável `title_for_layout`:

```
$this->set('title_for_layout', 'Título da página');
```

2.3.2.2.2 - render

render(string $action, string $layout, string $file)

Esse método sempre é chamado no fim de todo método de um controle. Através dele é definido qual arquivo de visão será embutido dentro do layout. Se *render* não é utilizado explicitamene no métdo do controle, então o controle irá procurar por uma tela com mesmo nome da ação do controle. Por exemplo, no método list do controle usuario o render por padrão iria procurar pelo arquivo */app/views/usuario/list.ctp*.

É possível especificar outra visão através de *$action*.

Se *$this->autoRender* estiver atribuído com o valor *false*, no fim do controle não irá ser executado esse método. Se pretende-se ler um elemento, deve-se escrever o seguinte código: *$this->render('/elements /ajax')*, que irá procurar pelo arquivo *app/views/elements/ajax.ctp*.

2.3.2.2.3 - redirect

redirect(string $url, integer $status, boolean $exit)

Utilizado para redirecionar o usuário para outra *view*. O parâmetro *$url* deve conter um caminho relativo no padrão de url do CakePHP. O parâmetro *$status* permite definir o estado da requisição HTTP. Pode-se, por exemplo, atribuir o valor 404, que corresponde a erro na busca do arquivo. Se for atribuído o valor *false* ao parâmetro *exit*, o método não vai executar a função *exit()* após executada.

```
<?php
if($success) {
        $this->redirect(array('controller' => 'vendas', 'action' => 'sucesso'));
} else {
        $this->redirect(array('controller' => 'vendas', 'action' => 'insucesso'));
```

```
}
?>
```

2.3.2.2.4 - flash

flash(string $message, string $url, integer $pause)

O método *flash* é similar ao *redirect*. A única diferença é a possibilidade de mostrar uma mensagem ao usuário antes dele ser redirecionado à página. No parâmetro *$message* deve-se atribuir a mensagem que deseja-se mostrar ao usuário antes que este seja redirecionado para a *url* definida no parâmetro *$url*. No parâmetro *$pause* define-se o tempo (em segundos) de exposição da mensagem.

2.3.2.2.5 - referer

Retorna a url que foi utilizada para chegar à requisição atual. Veja o exemplo a seguir:

```
<?php
//Irá retornar a string admin/usuario/index se o método foi
//acessado via tela admin_index do controle usuário
class UsuariosController extends AppController {
        function admin_view($id = null) {
                echo $this->referer();
        }
}
?>
```

2.3.2.2.6 - disableCache

Utilizado para determinar ao *browser* do usuário não fazer *cache*.

```
<?php
//Ao acessar a tela admin/usuario/view o browser do usuário não irá fazer cache.
class UsuariosController extends AppController {
        function admin_view($id = null) {
                $this->disableCache();
        }
}
?>
```

2.3.2.2.7 - postConditions

postConditions(array $data, mixed $op, string $bool, boolean $exclusive)

Transforma os dados postados em critérios de busca para o modelo. Pode-se utilizar o *FormHelper* para gerar o formulário que irá fornecer os dados. Veja o exemplo a seguir:

```
<?php
//Irá setar na variável $usuario os dados retornados pela busca por usuário
//seguindo os critérios de busca definidos por postConditions. Neste caso, as
// condições são os dados contidos no array $this->data.

class CanaisController extends AppController {
        function view() {
                if(isset($this->data)) {
```

```
$dados=$this->Canal->find('all',$this->postConditions($this->data));
                        $this->set('canal', $dados);
                }
        }
}
?>
```

2.3.2.2.8 - paginate

Esse método é utilizado para gerar paginação dos resultados. Pode-se utilizar as chaves conditions, fields, order, limit. Da mesma forma como se faz no método find.

```
<?php
class NoticiasController extends AppController {
        var $paginate = array(
                'conditions' => 'Noticia.canal_id = 2'
                'fields' => array('Noticia.titulo', 'Noticia.manchete'),
                'limit' => 4,
                'order' => array(
                        'Noticia.criada' => 'asc'
                )
        );
}
?>
```

2.3.2.2.9 - requestAction

requestAction(string $url, array $options)

Esse método é utilizado para acessar uma ação de qualquer controle e retornar dados dessa ação. No parâmetro *url* deve-se fornecer um caminho no padrão do cake: nomeDoControle/nomeDaAção/parâmetros. O parâmetro *$options* é utilizado para passar dados extras ao controle. Se nesse parâmetro for passado o valor *return* o resultado será a renderização da visão relacionada ao controle procurado pela *$url*.

```
<?php
class CanaisController extends AppController {
        $url = 'noticias/view/2';
        requestAction($url, array('return'));
}
?>
```

2.3.2.3 - Callbacks

2.3.2.3.1 - beforeFilter

Esta função é executada antes de qualquer ação do *controller*. É importante utilizá-la para inspecionar a existência de alguma variável de sessão ou verificar se o usuário possui permissões de acesso.

2.3.2.3.2 - beforeRender

Esta função é executada depois da ação do *controller* e antes da visão ser renderizada.

2.3.2.3.3 - afterFilter

Este *callback* é chamado depois que uma ação no *controller* é executada.

2.3.2.3.4 - afterRender

Chamada depois que uma ação é renderizada.

2.3.2.3.5 - _beforeScaffold

_beforeScaffold($method)

Este *callback* é utilizado quando está sendo feito uso do *scaffold*. O parâmetro *$method* deve conter o nome da ação que será executada.

2.3.2.3.6 - _afterScaffoldSave

_afterScaffoldSave($method)

Callback igual ao *_beforeScaffold*, entretanto é executado depois de uma ação de edição ou update realizada por scaffold. O parâmetro *$method* deve conter o valor *edit* ou *update*.

2.3.2.3.7 - _afterScaffoldSaveError

_afterScaffoldSaveError($method)

Callback igual ao *_afterScaffoldSave*, entretanto é executado quando ocorre algum erro na tentativa de edição ou atualização. O parâmetro *$method* deve conter o valor *edit* ou *update*.

2.3.2.3.8 - _scaffoldError

_scaffoldError($method)

Essa função é chamada quando ocorre algum erro em uma ação feita via scaffold. O parâmetro *$method* deve conter o nome da ação.

2.3.3 - A camada de visão

Essa camada é responsável de gerar a interface com o cliente. Na maioria das vezes o resultado dessa camada será um arquivo de extensão XHTML, mas é possível também gerar aruivos de RSS. É possível gerar também arquivos de extensão CSV.

Os arquivos da visão são salvos com a extensão .ctp e escritos em php. A extensão ctp quer dizer *CakePhp Template*. Esses arquivos possuem toda a lógica para receber os dados enviados pela camada de controle e apresentá-los para o usuário.

Os arquivos de visão são salvos no diretório app/views, dentro de uma pasta com o nome do controle que está relacionado. O nome do arquivo deve ser o nome da ação que a visão vai apresentar com extensão .ctp. Por exemplo, O arquivo de visão da ação de adicionar um produto do controle produto deve ser salvo no seguinte arquivo *app/views/produto/adicionar.ctp*

A camada de visão é composto de três partes distintas: *layout, elements* e *helpers*.

Layouts são arquivos de visão que irão abrigar diversos arquivos da aplicação. A maioria dos arquivos de visão são renderizadas dentro de um arquivo de *layout*.

Elements são pequenos pedaços de código que podem ser reusados em diversos lugares da aplicação. Geralmente são renderizados dentro de uma *view*.

Helpers fornecem lógica para os arquivos de visão. Através de *helpers* o CakePHP fornece formas fáceis de se construir formulários, aplicações Ajax, paginação, RSS, dentre diversas outras funcionalidades.

2.3.3.1 - Layouts

O *layout* contém código de apresentação que abriga os arquivos de visão. Tudo que se deseja ver em uma visão deve ser posto dentro de um arquivo de *layout*.

Os arquivos de *layout* estão disponíveis dentro do diretório */app/views/layouts*. Por padrão o CakePHP irá procurar pelo arquivo */app/views/layouts/default.ctp* para renderizar os arquivos de visão dentro deste arquivo. Entretanto, é possível sobrescrever esse arquivo.

Um arquivo de *layout* deve conter uma região destinada para carregar os arquivos de visão. Isto é feito através da variável `$content_for_layout`. Pode-se também definir o título da tela através da função `$this->set('title_for_layout','Título da página')`. Para incorporar arquivos de javascript externos ao arquivo de *layout*, deve-se utilizar a variável `$scripts_for_layout`.

Ao invés de definir o título da visão em *layout* é possível definir no próprio arquivo de controle através da variável `$this->layout`. O mesmo ocorre com o título, que pode ser definido com `$this->set('title_for_layout','Título da página');` no arquivo de controle para definir o título da tela. Para ver um exemplo de arquivo de layout abra o arquivo `/app/views/layouts/default.ctp` do CakePHP.

```
<?php
class UsuariosController extends AppController {
        function mostrarAlternativo() {
                $this->set('title_for_layout', ':: Home ::');
                $this->layout = 'alternativo';
        }
        function mostrarPadrao() {
                $this->layout = 'padrao';
        }
}
?>
```

Listagem 43 - Exemplo de uso de `$this->set('title_for_layout','Título da página')` e `$this->layout` no arquivo de controle

Além do *layout* padrão que é o *default* o CakePHP também fornece o *ajax* que é adequado para renderizar telas com recursos *ajax* e o *layout* e o *flash* que é útil quando irá se lidar com mensagens enviadas via método *flash*.

2.3.3.2 - Elements

Muitas aplicações possuem partes da apresentação que precisam ser repetidas em diversas partes. Para isso existe o *element*. Geralmente essas partes da apresentação podem ser um menu, formulários de login, caixas de ajuda, dentre diversos outros exemplos. Um elemento pode ser visto como um mini-arquivo de visão que é embutido dentro de um arquivo de visão, *layouts* e mesmo dentro de outro arquivo *element*.

Arquivos desse tipo são salvos dentro do diretório */app/views/elements/* e utilizam a extensão .ctp.

```
<?php
        echo $this->element('helpbox');
?>
```

Listagem 44 - Exemplo de como importar um *Element*

2.3.3.2.1 - Passando variáveis dentro de um elemento

É possível passar dados para um elemento através do segundo parâmetro.

```
<?php
$this->element('caixaDeMensagem',array("mensagem" => "nononoo nononoon"));
?>
```

Neste exemplo, dentro de *element*, a variável $mensagem ficaria disponível com o valor "nononoo nononoon". Além de dados, a função *element* também fornece as opções *cache* e *plugin*.

```
<?php
```

```
        $this->element('caixaDeMensagem',
                array(
                        "mensagem" => "nononoo nononoon"
                        "mensagem2" => "outra variável"
                        "cache" => "+2 days"
                        "plugin" => ""
                )
        );
?>
```

2.3.3.2.2 - Fazendo o *cache* de elementos

Se o parâmetro *cache* estiver atribuído com o valor *true*, será feito cache de um dia. Pode-se também definir valores diferentes deste valor padrão.

```
<?php
echo $this->element('caixa', array('cache' => true)); //cache de 1 dia

echo $this->element('caixa', array('cache' =>
        array('key' => 'first_use','time' => '+1 day'),
        'var' => $var)
);

echo $this->element('caixa', array('cache' =>
        array('key' => 'second_use','time' => '+1 day'),
        'var' => $varDiferente));
?>
```

2.3.3.2.3 - Requisão de elementos através de um plugin

Para utilizar um *element* dentro de um *plugin*, basta utilizar o parâmetro *plugin* no método *element*.

```
<?php
        echo $this->element('caixaDeMensagem', array('plugin' => 'NomeDoPlugin'));
?>
```

2.3.3.3 - Themes

Usando *Themes* é possível alterar o visual da aplicação de forma fácil e prática. Para usar temas é necessário utilizar no *controller* a classe *ThemeView* no lugar da classe de visão padrão. Depois, através da variável *$theme* é possível definir o tema que será utilizado.

```
<?php
class UsuariosController extends AppController {
        var $view = 'Theme';
        var $theme = 'exemplo';
}
?>
```

Também é possível trocar os temas dentro das funções de *callback: beforeFilter* e *beforeRender*.

```
<?php
$this->theme = 'outro_exemplo';
?>
```

Os temas são salvos dentro do diretório */app/views/themed/*, da mesma forma que acontece com os arquivos da *view*, cada tema deve estar dentro de uma pasta que, por sua vez, deve ter o nome do tema. Em nosso exemplo passado, o arquivo de tema seria salvo dentro do diretório */app/views/themed/exemplo/*.

Se fosse um arquivo do tipo *edit*, este arquivo teria que ser salvo no seguinte caminho: */app/views/themed /exemplo/edit.ctp*.

Se o tema não for achado dentro do diretório *app/views/themes*, o CakePHP irá procurar em *app/views* um arquivo substituivo. Assim fica possível criar um tema principal e em casos específicos utilizar temas específicos.

É possível salvar arquivos CSS e javascript, que são necessários nos temas, seguindo a mesma lógica. Neste caso os aquivos deverão ser salvos nos diretórios */app/webroot/themed/exemplo/css/* e */app/webroot /themed/exemplo/js/* respectivamente.

2.3.3.4 - Media Views

Através do uso de *Media Views* é possível enviar arquivos binários para o usuário. É útil, por exemplo, caso se queira ter um diretório fora de webroot, para prevenir que usários acessem o diretório webroot diretamente. É possível usar o *Media View* para outra pasta dentro de /app. Além disso, permite fazer autentificação do usuário antes que ele acesse esse arquivo.

Para usar *Media View* é preciso dizer ao arquivo de controle que esta classe será usada no lugar da classe *View* padrão. Depois disso, basta passar como parâmetro onde o arquivo está localizado.

```
<?php
class ExemplosController extends AppController {
        function download () {
                $this->view = 'Media';
                $params = array(
                        'id' => 'banner.swf',
                        'name' => 'banner',
                        'download' => true,
                        'extension' => 'swf',
                        'path' => 'flash' . DS
                );
                $this->set($params);
        }
}
?>
```

Listagem 45 - Exemplo de uso do *Media View*

Parâmetro	Descrição
id	Deve conter o nome do arquivo como está salvo no servidor, com sua extensão.
name	Pode-se escolher outro nome do arquivo para ser apresentado ao cliente. O nome não deve conter a extensão.
download	Valor lógico que indica se os cabeçalhos devem ser setados para forçar o download.
extension	A extensão do arquivo. Deve ser uma extensão válida para o *mimeType* selecionado.

Parâmetro	Descrição
path	O caminho onde o arquivo está salvo. Deve incluir o separador final. O caminho é relativo ao diretório app/.
mimeType	Array que pode conter adicionais *mime types* para serem aceitos com na Media View

Tabela 15 - Parâmetros do *Media View*

2.3.3.5 - Usando os Helpers básicos

O CakePHP carrega automaticamente os helpers Html e Form. Estes helpers são os mais utilizados no desenvolvimento de uma aplicação.

Para imprimir uma imagem, por exemplo, utilizamos a função `image()` do helper Html:

```
// Código fonte
$this->Html->image('nome_da_imagem.jpg',array('alt'=>'Texto alternativo para a
imagem'));

// Html gerado
<img src="/bookmarks/img/nome_da_imagem.jpg" alt="Texto alternativo para a imagem"/>
```

Para imprimir um link, utilizamos a função `link()` do helper Html:

```
// Código fonte
$this->Html->link('texto do link','/bookmarks/caminhoRelativo',
array('class'=>'nomeDaClasseCss'));

// Html gerado
<a href="/caminhoRelativo" class="nomeDaClasseCss">texto do link</a>
```

Para imprimir o campo de um formulário, utilizamos a função `input()` do helper Form:

```
// Código fonte
$this->Form->input('Usuario.senha',array('type'=>'password'));

// Html gerado
<input type="password" id="BookmarkSenha" value="" maxlength="32" name="data[Bookmark]
[senha]">
```

Nota

A lista completa das funções disponíveis nos helpers do CakePHP podem ser encontradas no capítulo 3.

2.4 - Usando o bake

O cake possui um gerador de códigos que pode ser utilizado pela linha de comando, o bake. O bake pode ser utilizado para gerar uma aplicação completa! Incluindo as classes de modelo, controle e as páginas da visão.

Para criar nossa primeira aplicação utilizando o bake, vamos considerar o seguinte cenário:

- Diretório da instalação do cake: */usr/lib/cake/1.3.x.x/*. Este diretório será referenciado como *DIR_INSTALAÇÃO_CAKE*.
- Diretório do novo projeto: */projetos/bookmarks*. Este diretório será referenciado como *DIR_NOVO_PROJETO*.
- O caminho para o console do cake */usr/lib/cake/1.3.x.x/cake/console/* foi adicionado ao *PATH* e está disponível na linha de comando
- O DocumentRoot do Apache foi configurado para apontar para o diretório */projetos*.

2.4.1 - Criando um novo projeto

Para criação de um novo projeto utilizando o console do CakePHP você possui duas opções: Ir para o diretório onde deseja criar o projeto e executar o bake ou passar como primeiro parâmetro o caminho para o projeto. Vamos optar pela segunda opção e passar como parâmetro o caminho para o novo projeto.

```
$ cake bake project DIR_NOVO_PROJETO
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : projetos
Path: /projetos
---------------------------------------------------------------
Bake Project
Skel Directory: /usr/lib/cake/1.3.x.x/cake/console/libs/templates/skel
Will be copied to: /projetos/bookmarks
---------------------------------------------------------------
Look okay? (y/n/q)
[y] >
```

Pressionando `Enter` ou `y` o esqueleto da aplicação localizada no diretório */usr/lib/cake/1.3.x.x/cake/console/libs/templates/skel* será copiado para a pasta */projeto/bookmarks*.

O bake irá perguntar se deseja que ele imprima as tarefas que estão sendo executadas na tela (*verbose output*). Repare que o valor padrão é *[n]* de não. Basta pressionar a tecla `Enter` ou `n` para confirmar a criação do projeto.

```
Do you want verbose output? (y/n)
[n] > n
---------------------------------------------------------------
Created: bookmarks in /projetos/bookmarks
---------------------------------------------------------------

Creating file /projetos/bookmarks/views/pages/home.ctp
Wrote /projetos/bookmarks/views/pages/home.ctp
Welcome page created
Random hash key created for 'Security.salt'
CAKE_CORE_INCLUDE_PATH set to /usr/lib/cake/1.3.x.x in webroot/index.php
CAKE_CORE_INCLUDE_PATH set to /usr/lib/cake/1.3.x.x in webroot/test.php
Remember to check these value after moving to production server
```

Note que a página de boas-vindas foi criada, uma chave *hash* foi criada e salva no arquivo `core.php`. A variável CAKE_CORE_INCLUDE_PATH foi setada para o caminho correto nos arquivos `index.php` e `test.php`.

2.4.2 - Gerando o arquivo de configuração do banco de dados

Para gerar o arquivo de configuração do banco de dados, você deve ir até o diretório do projeto e executar o

bake novamente.

```
$ cd DIR_NOVO_PROJETO
$ cake bake
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
Your database configuration was not found. Take a moment to create one.
---------------------------------------------------------------
```

O bake irá solicitar informações sobre a configuração do banco. Siga as instruções e entre com as informações uma a uma:

A primeira pergunta é sobre o nome da configuração que será utilizada. Vamos utilizar a *default*.

```
Database Configuration:
---------------------------------------------------------------
Name:
[default] > default
```

Depois ele apresenta a lista dos drivers suportados e pergunta qual deles será utilizado para conexão com o banco de dados. Iremos utilizar o mysql.

```
Driver: (db2/firebird/mssql/mysql/mysqli/odbc/oracle/postgres/sqlite/sybase)
[mysql] > mysql
```

Depois ele irá perguntar se você deseja que a conexão seja persistente ou não. Vamos utilizar uma conexão não persistente.

```
Persistent Connection? (y/n)
[n] > n
```

Depois ele irá perguntar o nome do host para conexão com o banco de dados. Vamos considerar localhost.

```
Database Host:
[localhost] > localhost
```

Depois ele irá perguntar qual a porta para a conexão com o banco. Vamos utilizar a padrão, por isso não é necessário informar uma porta.

```
Port?
[n] > n
```

Depois ele irá perguntar as credenciais (usuário e senha) para a conexão com o banco. Vamos utilizar o usuário root e a senha curso_cake_2km.

```
User:
[root] > root
Password:
> cake_2km
```

Depois ele irá perguntar qual o nome do banco. Vamos utilizar o curso_cake.

```
Database Name:
[cake] > curso_cake_2km
```

Depois ele irá perguntar se você deseja definir um prefixo para as tabelas. Não vamos utilizar nenhum prefixo.

```
Table Prefix?
[n] > n
```

Depois ele irá perguntar qual o *encoding* data tabela. Vamos utilizar o padrão

```
Table encoding?
[n] > n
```

No final ele mostrará todas as informações que foram fornecidas. Confirme as informações e digite `y` ou `Enter`.

```
---------------------------------------------------------------
The following database configuration will be created:
---------------------------------------------------------------
Name:         default
Driver:       mysql
Persistent:   false
Host:         localhost
User:         root
Pass:         *****
Database:     curso_cake_2km
---------------------------------------------------------------
Look okay? (y/n)
[y] > y
```

Depois ele pergunta se você deseja criar outra configuração de banco. Não vamos criar outra configuração, por isso digite `n` ou `Enter`.

```
Do you wish to add another database configuration?
[n] > n
```

Aí então ele irá criar o arquivo de configuração do banco:

```
Creating file /projetos/bookmarks/config/database.php
Wrote /projetos/bookmarks/config/database.php
```

O arquivo de configuração do banco ficará assim:

```
<?php
class DATABASE_CONFIG {

        var $default = array(
                'driver' => 'mysql',
                'persistent' => false,
                'host' => 'localhost',
                'login' => 'root',
                'password' => 'cake_2km',
                'database' => 'curso_cake_2km',
        );
}
```

```
?>
```

Listagem 46 - Código gerado pelo bake do arquivo de configuração do banco de dados

Nota

A distribuição do cake possui um projeto pré-configurado que encontra-se na pasta *app*. Se você desejar utlizar o bake neste projeto vá para o diretório da instalação do cake e execute o bake:

```
$ cd DIR_INSTALAÇÃO_CAKE
$ cake bake
```

2.4.3 - Gerando as classes de modelo

Antes de gerarmos as classes de modelo precisamos criar as tabelas no banco de dados. A aplicação que iremos desenvolver será uma aplicação para gerenciar bookmarks.

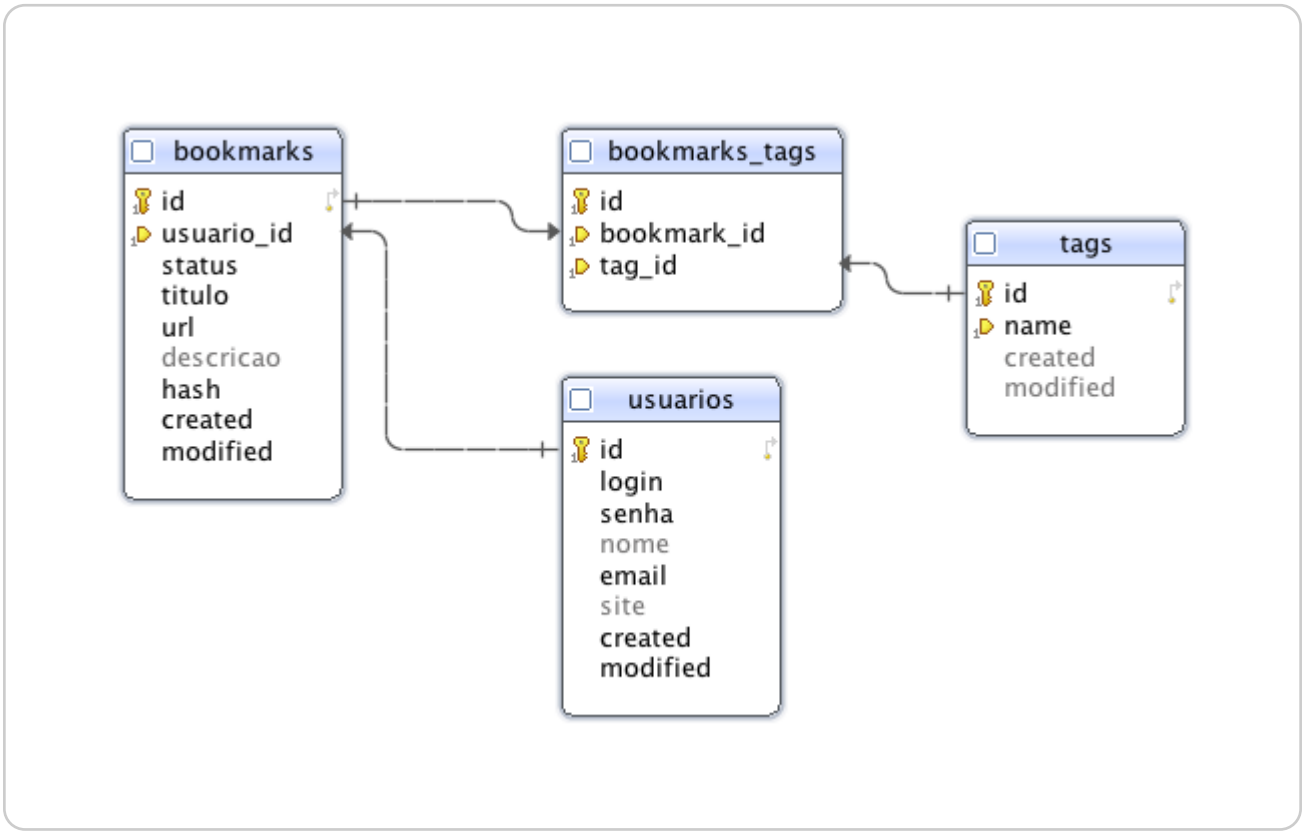


Imagem 2 - Esquema da aplicação que iremos desenvolver utilizando o bake.

```
--
-- Estrutura da tabela `bookmarks`
--

CREATE TABLE IF NOT EXISTS `bookmarks` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `usuario_id` int(11) NOT NULL DEFAULT '0',
  `status` tinyint(1) NOT NULL DEFAULT '0',
  `titulo` varchar(255) NOT NULL DEFAULT '',
  `url` text NOT NULL,
  `descricao` text,
```

```
  `hash` varchar(32) NOT NULL DEFAULT '',
  `created` datetime NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id`),
  KEY `usuario_id` (`usuario_id`)
);

-- --------------------------------------------------------

--
-- Estrutura da tabela `bookmarks_tags`
--

CREATE TABLE IF NOT EXISTS `bookmarks_tags` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `tag_id` int(11) NOT NULL,
  `bookmark_id` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `tag_id` (`tag_id`),
  KEY `bookmark_id` (`bookmark_id`)
);

-- --------------------------------------------------------

--
-- Estrutura da tabela `tags`
--

CREATE TABLE IF NOT EXISTS `tags` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(32) NOT NULL,
  `created` datetime NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `tag_bookmark` (`nome`)
);

-- --------------------------------------------------------

--
-- Estrutura da tabela `usuarios`
--

CREATE TABLE IF NOT EXISTS `usuarios` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `login` varchar(25) NOT NULL DEFAULT '',
  `senha` varchar(40) NOT NULL DEFAULT '',
  `nome` varchar(50) DEFAULT NULL,
  `email` varchar(50) NOT NULL DEFAULT '',
  `site` varchar(255) DEFAULT NULL,
  `created` datetime NOT NULL,
  `modified` datetime NOT NULL,
  PRIMARY KEY (`id`)
);
```

Script 1 – Script das tabelas da aplicação.

2.4.3.1 - Gerando a classe do modelo Bookmark

Após a geração do arquivo de configuração do banco de dados, vamos executar o bake novamente para gerar as classes de modelo:

```
$ cake bake
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

Para gerar as classes do modelo vamos selecionar a opção *Model* pressionando a tecla `m` e depois a `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> m
---------------------------------------------------------------
Bake Model
Path: /projetos/bookmarks/models/
---------------------------------------------------------------
Possible Models based on your current database:
1. Bookmark
2. BookmarksTag
3. Tag
4. Usuario
```

Vamos escolher um dos modelos que foram identificados pelo CakePHP através dos dados da conexão fornecidos anteriormente. Vamos considerar como exemplo a geração do modelo Bookmark, pressionando a tecla `1` e depois `Enter`.

Nota

Caso você queira sair do console do bake você pode pressionar a tecla `q`.

```
Enter a number from the list above, type in the name of another model, or 'q' to
exit
[q] > 1
```

2.4.3.1.1 - Definindo as validações

O bake irá perguntar se você deseja informar algum critério de validação para os campos do modelo selecionado, vamos informar que sim, pressionando a tecla `Enter` ou a tecla `y`.

```
Would you like to supply validation criteria for the fields in your model? (y/n)
```

```
[y] > y
```

O CakePHP possui diversas validações prontas para serem utilizadas, como data, dinheiro, email, tempo, url, etc.

Nota

A tabela com os detalhes de todos os tipos de validação disponíveis no framework estão listados na tabela "Regras padrões de validação do CakePHP" no item `2.3.1.1.10 - validate` deste capítulo.

Para cada campo do modelo selecionado, o bake irá exibir seu tipo definido no banco de dados e irá perguntar qual tipo de validação será utilizada. O campo `id`, por exemplo, é do tipo `INTEGER` no banco de dados. Como ele é a chave primária e foi definido como `NOT NULL` na criação da tabela, não vamos fazer a validação. Repare que a opção padrão (o número 29 entre colchetes) é a opção que não define nenhum tipo de validação para o campo atual. Você pode pressionar as teclas `2` e `9` e depois `Enter` ou simplesmente `Enter`.

```
Field: id
Type: integer
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[29] > Enter
```

O próximo campo é o campo `usuario_id`, que também é do tipo `INTEGER` no banco de dados. Vamos utilizar o tipo de validação *numeric*. Repare que o valor padrão passou a ser `21` por isso podemos pressionar a tecla `Enter` para confirmar a validação.

```
Field: usuario_id
Type: integer
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[21] > Enter
```

O próximo campo é o campo `status`. Como no banco de dados foi definido como um `tinyint(1)` o bake mostra o campo como sendo do tipo `boolean`. Vamos utilizar o tipo de validação *boolean* que aceita valores como `true` ou `false`, os inteiros `0` ou `1` ou as strings `'0'` ou `'1'`. Para isso vamos pressionar a tecla `4` e depois `Enter`.

```
Field: status
Type: boolean
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
```

```
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[21] > 4
```

O próximo campo é o campo `titulo` que é do tipo `string`. Vamos utilizar o tipo de validação *notEmpty* que obriga o usuário a preencher o campo com algum valor. Repare que a opção padrão sugerida é a número `20` por isso vamos pressionar a tecla `Enter`.

```
Field: titulo
Type: string
-----------------------------------------------------------------
Please select one of the following validation options:
-----------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
```

```
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[20] > Enter
```

O próximo campo é o campo `url`. No banco ele foi definido como um `text` para suportar muitos caracteres, mas vamos utilizar o tipo de validação *url*. Para isso vamos pressionar as teclas `2`, `7` e depois `Enter`.

```
Field: url
Type: text
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

```
[29] > 27
```

O próximo campo é o campo `descricao`. No banco ele foi definido como um `text` para suportar muitos caracteres. Vamos limitar o seu conteúdo selecionando a opção de validação *maxLength* que define o número máximo de caracteres que um campo pode conter. Para isso vamos pressionar as teclas `1`, `6` e depois `Enter`.

```
Field: descricao
Type: string
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[29] > 16
```

O próximo campo é o campo `hash`. Este campo irá guardar uma chave única (md5) para cada url e foi definido como um `varchar(32)`. Não vamos utilizar nenhuma validação pois o campo será preenchido automaticamente antes de salvarmos o bookmark. Para isso vamos pressionar as teclas `2`, `9` e depois `Enter`.

```
Field: hash
Type: string
---------------------------------------------------------------
Please select one of the following validation options:
```

```
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[20] > 29
```

O próximo campo é o campo `created`. Quando o registro for criado o dia e a hora serão salvos automaticamente pelo CakePHP. Neste campo não iremos utilizar nenhuma validação. Vamos confirmar a validação padrão pressionando a tecla `Enter`.

```
Field: created
Type: datetime
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
```

```
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[29] > Enter
```

O próximo campo é o campo `modified`. Quando o registro for alterado, o dia e a hora serão salvos Este campo será preenchido automaticamente pelo CakePHP. Neste caso também não iremos utilizar nenhuma validação. Vamos confirmar a validação padrão pressionando a tecla `Enter`.

```
Field: modified
Type: datetime
-----------------------------------------------------------------
Please select one of the following validation options:
-----------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
```

```
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.

[29] > Enter
```

2.4.3.1.2 - Definindo os relacionamentos

Após as validações em todos os campos, o bake irá perguntar se gostaríamos de definir as associações do modelo. Ele irá tentar detectar as associações de acordo com as convenções e devemos confirmar cada uma delas.

Mais informações sobre as associações entre os modelos podem ser encontradas no tópico 2.3.1.2 - Associando os modelos.

Vamos confirmar pressionando a tecla `Enter`.

```
Would you like to define model associations (hasMany, hasOne, belongsTo, etc.)?
(y/n)
[y] > Enter

One moment while the associations are detected.
---------------------------------------------------------------
Please confirm the following associations:
---------------------------------------------------------------
```

Um bookmark pertence a um usuário. Vamos confirmar pressionando a tecla `Enter`.

```
Bookmark belongsTo Usuario? (y/n)
[y] > Enter
```

Um bookmark pode possuir muitas tags e uma tag pode pertencer a muitos bookmarks. Esta associação é conhecida como HABTM (HasAndBelongsToMany). Vamos confirmar pressionando a tecla `Enter`.

```
Bookmark hasAndBelongsToMany Tag? (y/n)
[y] > Enter
```

Não precisamos definir nenhuma associação adicional, por isso vamos pressionar a tecla `Enter`.

```
Would you like to define some additional model associations? (y/n)
[n] > Enter

---------------------------------------------------------------
The following Model will be created:
---------------------------------------------------------------
Name:       Bookmark
Validation: Array
(
    [usuario_id] => numeric
    [status] => boolean
    [titulo] => notempty
    [url] => url
    [descricao] => maxlength
)
```

```
Associations:
                        Bookmark belongsTo Usuario
                        Bookmark hasAndBelongsToMany Tag
---------------------------------------------------------------
```

Podemos conferir os dados exibidos na tela e pressionar a tecla `Enter`.

```
Look okay? (y/n)
[y] > Enter
Baking model class for Bookmark...

Creating file /projetos/bookmarks/models/bookmark.php
Wrote /projetos/bookmarks/models/bookmark.php
```

A classe de modelo bookmark.php ficará assim:

```
<?php
class Bookmark extends AppModel {

  var $name = 'Bookmark';
  var $validate = array(
    'usuario_id' => array('numeric'),
    'status' => array('boolean'),
    'titulo' => array('notempty'),
    'url' => array('url'),
    'descricao' => array('maxlength')
  );

  //The Associations below have been created with all possible keys, those that are
not needed can be removed
  var $belongsTo = array(
      'Usuario' => array('className' => 'Usuario',
                'foreignKey' => 'usuario_id',
                'conditions' => '',
                'fields' => '',
                'order' => ''
      )
  );

  var $hasAndBelongsToMany = array(
      'Tag' => array('className' => 'Tag',
            'joinTable' => 'bookmarks_tags',
            'foreignKey' => 'bookmark_id',
            'associationForeignKey' => 'tag_id',
            'unique' => true,
            'conditions' => '',
            'fields' => '',
            'order' => '',
            'limit' => '',
            'offset' => '',
            'finderQuery' => '',
            'deleteQuery' => '',
            'insertQuery' => ''
      )
```

```
    );

}
?>
```

Listagem 47 - Classe de modelo bookmark.php gerada pelo bake.

A variável $name é definida apenas para manter a compatibilidade com o PHP4. O array $validate é o responsável pela validação dos campos definidos neste modelo. O array $belongsTo e o array hasAndBelongsToMany definem, respectivamente, o relacionamento do modelo Bookmark com o modelo Usuario e o modelo Tag.

Mais informações sobre as associações entre os modelos podem ser encontradas no tópico 2.3.1.2 - Associando os modelos.

Após a geração do modelo o bake pergunta se queremos gerar os arquivos de teste unitário. Este tópico é explicado passo-a-passo no Capítulo 10 - Testes.

Como nosso objetivo é apenas a geração do código, vamos pressionar a tecla `n` e depois `Enter` para que os testes não sejam gerados.

```
Cake test suite not installed.  Do you want to bake unit test files anyway? (y/n)
[y] > n
---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

A classe de modelo bookmark.php foi gerada com as validações escolhidas e salva no diretório *DIR_NOVO_PROJETO/models/*.

2.4.3.2 - Gerando a classe do modelo Tag

Para a geração da próxima classe de modelo (Tag) vamos executar selecionar a opção Model digitando `m` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> m
---------------------------------------------------------------
Bake Model
Path: /projetos/bookmarks/models/
---------------------------------------------------------------
Possible Models based on your current database:
1. Bookmark
2. BookmarksTag
3. Tag
4. Usuario
```

O bake exibe os modelos encontrados e podemos selecionar o modelo Tag pressionando a tecla `3` e depois `Enter`.

```
Enter a number from the list above, type in the name of another model, or 'q' to
```

```
exit
[q] > 3
```

2.4.3.2.1 - Definindo as validações

Vamos informar os critérios de validação para cada um dos campos pressionando `Enter`.

```
Would you like to supply validation criteria for the fields in your model? (y/n)
[y] > Enter

Field: id
Type: integer
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

Para o campo `id` vamos manter a validação padrão. Para isso, vamos pressionar a tecla `Enter`.

```
[29] > Enter

Field: nome
Type: string
---------------------------------------------------------------
Please select one of the following validation options:
```

```
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

Para o campo `nome`, vamos utilizar a validação `notEmpty` para garantir que a tag terá um nome. Para isso, vamos pressionar a tecla `Enter`.

```
[20] > Enter

Field: created
Type: datetime
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
```

```
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

Como vimos anteriormente o campo `created` será preenchido pelo CakePHP, por isso não será necessária uma validação. Vamos pressionar a tecla `Enter`.

```
[29] > Enter

Field: modified
Type: datetime
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
```

```
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

Da mesma forma, o campo `modified` também será preenchido pelo CakePHP, por isso não será necessária uma validação. Vamos pressionar a tecla `Enter`.

```
[29] > Enter
```

2.4.3.2.2 - Definindo as associações

Agora vamos definir as associações com os outros modelos. Vamos pressionar a tecla `Enter`.

```
Would you like to define model associations (hasMany, hasOne, belongsTo, etc.)?
(y/n)
[y] > Enter
One moment while the associations are detected.
---------------------------------------------------------------
Please confirm the following associations:
---------------------------------------------------------------
```

Uma Tag pode pertencer a muitos bookmarks e um bookmark pode possuir muitas tags. Vamos confirmar a associação pressionando a tecla `Enter`.

```
Tag hasAndBelongsToMany Bookmark? (y/n)
[y] > Enter
```

Não queremos definir associações adicionais portanto vamos pressionar a tecla `Enter`.

```
Would you like to define some additional model associations? (y/n)
[n] > Enter

---------------------------------------------------------------
The following Model will be created:
---------------------------------------------------------------
Name:       Tag
Validation: Array
(
    [nome] => notEmpty
)

Associations:
                        Tag hasAndBelongsToMany Bookmark
---------------------------------------------------------------
```

Podemos confirmar as informações e pressionar a tecla `Enter`.

```
Look okay? (y/n)
[y] > Enter

Baking model class for Tag...

Creating file /projetos/bookmarks/models/tag.php
Wrote /projetos/bookmarks/models/tag.php
```

A classe tag.php foi gerada e salva no diretório dos modelos *DIR_NOVO_PROJETO/models*. Não vamos gerar as classes de teste, por isso vamos digitar a tecla `n` e depois `Enter`.

```
Cake test suite not installed.  Do you want to bake unit test files anyway? (y/n)
[y] > n
---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

2.4.3.3 - Gerando a classe do modelo Usuário

Agora vamos gerar a próxima classe de modelo, a classe dos usuários. Para isso, vamos selecionar a opção Model digitando `m` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> m
---------------------------------------------------------------
Bake Model
Path: /projetos/bookmarks/models/
---------------------------------------------------------------
Possible Models based on your current database:
1. Bookmark
2. BookmarksTag
3. Tag
4. Usuario
```

Vamos selecionar o modelo Usuario pressionando a tecla `4` e depois `Enter`.

```
Enter a number from the list above, type in the name of another model, or 'q' to
exit
[q] > 4
```

2.4.3.3.1 - Definindo as validações

Vamos definir as validações do modelo Usuario pressionando a tecla `Enter`.

```
Would you like to supply validation criteria for the fields in your model? (y/n)
[y] > Enter

Field: id
Type: integer
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
```

```
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

O primeiro campo `id` não será validado. Vamos confirmar pressionando a tecla `Enter`.

```
[29] > Enter

Field: login
Type: string
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
```

```
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

O campo `login` será validado com a opção between, que valida o campo de acordo com dois parâmetros: o menor e o maior número de caracteres permitidos. Não é possível informar estes parâmetros antes da criação da classe, por isso vamos digitar a tecla `2` e depois `Enter`.

```
[20] > 2

Field: senha
Type: string
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

Importante

Após a geração da classe de modelo, será necessário informar o valor mínimo e máximo permitidos para a validação do campo *login* definido como *between*. Assim como as mensagens para todos os campos que estão sendo validados.

O campo `senha` será validado apenas como `notEmpty`. Para isso vamos digitar `Enter`.

```
[20] > Enter

Field: nome
Type: string
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

O campo `nome` também será validado como `notEmpty`. Para isso vamos digitar `2`, `0` e depois `Enter`.

```
[29] > 20

Field: email
Type: string
---------------------------------------------------------------
```

```
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

O campo `email` será validado como `email`. Para isso vamos pressionar a tecla `Enter`.

```
[10] > Enter

Field: created
Type: datetime
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
```

```
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

O campo `created` não será validado, por isso vamos pressionar `Enter`.

```
[29] > Enter

Field: modified
Type: datetime
---------------------------------------------------------------
Please select one of the following validation options:
---------------------------------------------------------------
1 - alphaNumeric
2 - between
3 - blank
4 - boolean
5 - cc
6 - comparison
7 - custom
8 - date
9 - decimal
10 - email
11 - equalTo
12 - extension
13 - file
14 - inList
15 - ip
16 - maxLength
17 - minLength
18 - money
19 - multiple
20 - notEmpty
21 - numeric
22 - phone
23 - postal
24 - range
25 - ssn
26 - time
27 - url
28 - userDefined
```

```
29 - Do not do any validation on this field.
... or enter in a valid regex validation string.
```

O campo `modified` também não será validado, por isso vamos pressionar `Enter`.

```
[29] > Enter
```

2.4.3.3.2 - Definindo os relacionamentos

Agora vamos definir as associações com os outros modelos. Vamos pressionar a tecla `Enter`.

```
Would you like to define model associations (hasMany, hasOne, belongsTo, etc.)?
(y/n)
[y] >
One moment while the associations are detected.
---------------------------------------------------------------
Please confirm the following associations:
---------------------------------------------------------------
```

Um usuário possui vários bookmarks. Vamos confirmar pressionando a tecla `Enter`.

```
Usuario hasMany Bookmark? (y/n)
[y] > Enter
```

Como definimos anteriormente, um usuário não possui apenas um bookmark, possui vários. Vamos negar essa associação pressionando a tecla `n` e depois `Enter`.

```
Usuario hasOne Bookmark? (y/n)
[y] > n
```

Não precisamos definir nenhuma associação adicional, por isso vamos pressionar a tecla `Enter`.

```
Would you like to define some additional model associations? (y/n)
[n] > Enter

---------------------------------------------------------------
The following Model will be created:
---------------------------------------------------------------
Name:       Usuario
Validation: Array
(
    [login] => between
    [nome] => notempty
    [senha] => notempty
    [email] => email
)

Associations:
                        Usuario hasMany Bookmark
---------------------------------------------------------------
```

Podemos confirmar as informações e pressionar a tecla `Enter`.

```
Look okay? (y/n)
```

```
[y] >

Baking model class for Usuario...

Creating file /projetos/bookmarks/models/usuario.php
Wrote /projetos/bookmarks/models/usuario.php
```

Não queremos gerar os arquivos de teste, por isso vamos pressionar a tecla `n` e depois a tecla `Enter`.

```
Cake test suite not installed.  Do you want to bake unit test files anyway? (y/n)
[y] > n
---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

2.4.4 - Gerando as classes de controle

Para gerar as classes de controle vamos selecionar a opção *Controller* pressionando a tecla `c` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> c
---------------------------------------------------------------
Bake Controller
Path: /projetos/bookmarks/controllers/
---------------------------------------------------------------
Possible Controllers based on your current database:
1. Bookmarks
2. BookmarksTags
3. Tags
4. Usuarios
```

O CakePHP lista todas as possíveis classes de controle baseadas na configuração do banco de dados. Vamos gerar a classe de controle `Bookmarks` pressionando a tecla `1` e depois `Enter`

```
Enter a number from the list above, type in the name of another controller, or
'q' to exit
[q] > 1
---------------------------------------------------------------
Baking BookmarksController
---------------------------------------------------------------
```

O bake irá perguntar se gostaríamos de construir a classe de controle interativamente, utilizando a variável especial $scaffold. Não vamos utilizá-la, por isso vamos pressionar a tecla `n` e depois `Enter`.

```
Would you like to build your controller interactively? (y/n)
[y] > n
```

Vamos gerar os métodos básicos (listar, adicionar, ver e editar) pressionando a tecla `enter`.

```
Would you like to include some basic class methods (index(), add(), view(),
edit())? (y/n)
[y] > Enter
```

Não vamos criar os métodos para as rotas de administração. Por isso vamos digitar a tecla `n` e depois `Enter`.

```
Would you like to create the methods for admin routing? (y/n)
[y] > n

---------------------------------------------------------------
The following controller will be created:
---------------------------------------------------------------
Controller Name:  Bookmarks
---------------------------------------------------------------
```

Podemos conferir as informações e confirmar a criação da classe pressionando a tecla `Enter`.

```
Look okay? (y/n)
[y] > Enter

Creating file /projetos/bookmarks/controllers/bookmarks_controller.php
Wrote /projetos/bookmarks/controllers/bookmarks_controller.php
```

A classe de controle bookmarks_controller.php ficará assim:

```
<?php
class BookmarksController extends AppController {

        var $name = 'Bookmarks';
        var $helpers = array('Html', 'Form');

        function index() {
                $this->Bookmark->recursive = 0;
                $this->set('bookmarks', $this->paginate());
        }

        function view($id = null) {
                if (!$id) {
                        $this->Session->setFlash(__('Invalid Bookmark.', true));
                        $this->redirect(array('action'=>'index'));
                }
                $this->set('bookmark', $this->Bookmark->read(null, $id));
        }

        function add() {
                if (!empty($this->data)) {
                        $this->Bookmark->create();
                        if ($this->Bookmark->save($this->data)) {
                                $this->Session->setFlash(__('The Bookmark has been
saved', true));
                                $this->redirect(array('action'=>'index'));
                        } else {
                                $this->Session->setFlash(__('The Bookmark could not
```

```
be saved. Please, try again.', true));
                        }
                }
                $tags = $this->Bookmark->Tag->find('list');
                $usuarios = $this->Bookmark->Usuario->find('list');
                $this->set(compact('tags', 'usuarios'));
        }

        function edit($id = null) {
                if (!$id && empty($this->data)) {
                        $this->Session->setFlash(__('Invalid Bookmark', true));
                        $this->redirect(array('action'=>'index'));
                }
                if (!empty($this->data)) {
                        if ($this->Bookmark->save($this->data)) {
                                $this->Session->setFlash(__('The Bookmark has been
saved', true));
                                $this->redirect(array('action'=>'index'));
                        } else {
                                $this->Session->setFlash(__('The Bookmark could not
be saved. Please, try again.', true));
                        }
                }
                if (empty($this->data)) {
                        $this->data = $this->Bookmark->read(null, $id);
                }
                $tags = $this->Bookmark->Tag->find('list');
                $usuarios = $this->Bookmark->Usuario->find('list');
                $this->set(compact('tags','usuarios'));
        }

        function delete($id = null) {
                if (!$id) {
                        $this->Session->setFlash(__('Invalid id for Bookmark', true));
                        $this->redirect(array('action'=>'index'));
                }
                if ($this->Bookmark->del($id)) {
                        $this->Session->setFlash(__('Bookmark deleted', true));
                        $this->redirect(array('action'=>'index'));
                }
        }

}
?>
```

Listagem 48 - Classe bookmarks_controller.php gerada pelo bake.

O código gerado pelo bake, inclui os *helpers* necessários (Html e Form) e cria os métodos index, view, add, edit e delete.

Não vamos gerar as classes de teste. Vamos pressionar a tecla `n` e depois `Enter`.

```
Cake test suite not installed.  Do you want to bake unit test files anyway? (y/n)
[y] > n
---------------------------------------------------------------
Interactive Bake Shell
```

```
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

Vamos agora para o próximo controle. Vamos pressionar a tecla `c` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> c
---------------------------------------------------------------
Bake Controller
Path: /projetos/bookmarks/controllers/
---------------------------------------------------------------
Possible Controllers based on your current database:
1. Bookmarks
2. BookmarksTags
3. Tags
4. Usuarios
```

Vamos selecionar a classe de controle Tags pressionando a tecla `3` e depois `Enter`.

```
Enter a number from the list above, type in the name of another controller, or
'q' to exit
[q] > 3
---------------------------------------------------------------
Baking TagsController
---------------------------------------------------------------
```

Não vamos construir a classe interativamente, por isso vamos pressionar a tecla `n` e depois `Enter`.

```
Would you like to build your controller interactively? (y/n)
[y] > n
```

Vamos gerar os métodos básicos (listar, adicionar, ver e editar) pressionando a tecla `enter`.

```
Would you like to include some basic class methods (index(), add(), view(),
edit())? (y/n)
[y] > Enter
```

Não vamos criar os métodos para as rotas de administração. Por isso vamos digitar a tecla `n` e depois `Enter`.

```
Would you like to create the methods for admin routing? (y/n)
[y] > n

---------------------------------------------------------------
The following controller will be created:
---------------------------------------------------------------
Controller Name:  Tags
---------------------------------------------------------------
```

Vamos confirmar as informações e digitar `Enter`.

```
Look okay? (y/n)
[y] > Enter

Creating file /projetos/bookmarks/controllers/tags_controller.php
Wrote /projetos/bookmarks/controllers/tags_controller.php
```

A classe tags_controller.php foi gerada e salva com sucesso. Não vamos gerar os arquivos de teste, por isso vamos pressionar a tecla `n` e depois a `Enter`.

```
Cake test suite not installed.  Do you want to bake unit test files anyway? (y/n)
[y] > n
---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

Vamos gerar agora a última classe de controle Usuarios. Para isso vamos pressionar a tecla `c` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> c
---------------------------------------------------------------
Bake Controller
Path: /projetos/bookmarks/controllers/
---------------------------------------------------------------
Possible Controllers based on your current database:
1. Bookmarks
2. BookmarksTags
3. Tags
4. Usuarios
```

Vamos selecionar o controller Usuarios digitando `4` e depois `Enter`.

```
Enter a number from the list above, type in the name of another controller, or
'q' to exit
[q] > 4
---------------------------------------------------------------
Baking UsuariosController
---------------------------------------------------------------
```

Não vamos construir a classe interativamente, por isso vamos pressionar a tecla `n` e depois `Enter`.

```
Would you like to build your controller interactively? (y/n)
[y] > n
```

Vamos gerar os métodos básicos (listar, adicionar, ver e editar) pressionando a tecla `enter`.

```
Would you like to include some basic class methods (index(), add(), view(),
edit())? (y/n)
```

```
[y] > Enter
```

Não vamos criar os métodos para as rotas de administração. Por isso vamos digitar a tecla `n` e depois `Enter`.

```
Would you like to create the methods for admin routing? (y/n)
[y] > n

---------------------------------------------------------------
The following controller will be created:
---------------------------------------------------------------
Controller Name:  Usuarios
---------------------------------------------------------------
```

Vamos confirmar a geração do arquivo...

```
Look okay? (y/n)
[y] >

Creating file /projetos/bookmarks/controllers/usuarios_controller.php
Wrote /projetos/bookmarks/controllers/usuarios_controller.php
```

O arquivo foi gerado e salvo com sucesso. Não precisamos gerar os arquivos de teste...

```
Cake test suite not installed.  Do you want to bake unit test files anyway? (y/n)
[y] > n
---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

Nota

Para a geração das classes de controle utilizando a variável *$scaffold* basta selecionar a opção y quando ele perguntar se você deseja utilizar o scaffolding.

2.4.5 - Gerando as páginas da visão

Não vamos gerar mais nenhuma classe de controle. Vamos agora gerar as páginas da visão digitando a tecla `v` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> v
```

O bake irá listar todas as possíveis classes de controle para serem utilizadas na geração das páginas. Após a seleção da classe de controle, o bake irá perguntar se deseja criar visões (índice, adicionar, ver e editar) para ele.

```
---------------------------------------------------------------
Bake View
Path: /projetos/bookmarks/views/
---------------------------------------------------------------
Possible Controllers based on your current database:
1. Bookmarks
2. BookmarksTags
3. Tags
4. Usuarios
```

Vamos gerar as páginas da visão para a classe de controle Bookmarks. Para isso, vamos digitar a tecla `1` e depois `Enter`.

```
Enter a number from the list above, type in the name of another controller, or
'q' to exit
[q] > 1
```

Vamos gerar as páginas para o índice, adicionar, ver e editar. Para isso vamos confirmar digitando `y` e depois `Enter`.

```
Would you like to create some scaffolded views (index, add, view, edit) for this
controller?
NOTE: Before doing so, you'll need to create your controller and model classes
(including associated models). (y/n)
[n] > y
```

No momento não vamos utilizar as rotas administrativas. Por isso vamos digitar `n` e depois `Enter`.

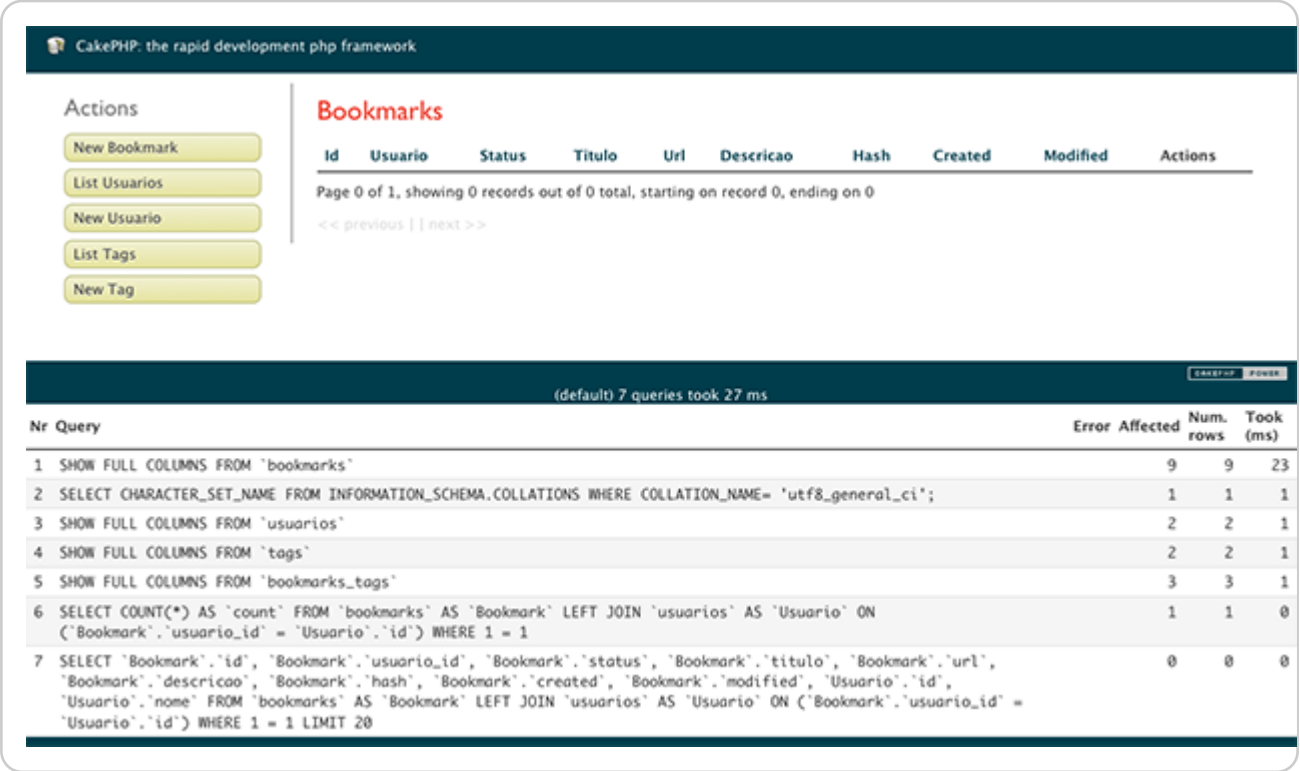
Nota

Para saber com mais detalhes sobre as rotas de administração e as rotas padrões no CakePHP veja no capítulo 3.1 - Rotas.

```
Would you like to create the views for admin routing? (y/n)
[y] > n
```

```
Creating file /projetos/bookmarks/views/bookmarks/index.ctp
Wrote /projetos/bookmarks/views/bookmarks/index.ctp

Creating file /projetos/bookmarks/views/bookmarks/view.ctp
Wrote /projetos/bookmarks/views/bookmarks/view.ctp

Creating file /projetos/bookmarks/views/bookmarks/add.ctp
Wrote /projetos/bookmarks/views/bookmarks/add.ctp

Creating file /projetos/bookmarks/views/bookmarks/edit.ctp
Wrote /projetos/bookmarks/views/bookmarks/edit.ctp
---------------------------------------------------------------

View Scaffolding Complete.

---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

A tela de listagem dos bookmarks pode ser acessada pelo endereço http://localhost/bookmarks/bookmarks/



Tela 1 – Tela da listagem dos bookmarks

Vamos repetir o processo para as outras classes de controle. Vamos digitar `v` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> v
---------------------------------------------------------------
Bake View
Path: /projetos/bookmarks/views/
---------------------------------------------------------------
Possible Controllers based on your current database:
1. Bookmarks
2. BookmarksTags
3. Tags
4. Usuarios
```

Vamos gerar as páginas da visão da classe de controle Tags. Para isso vamos digitar `3` e depois `Enter`.

```
Enter a number from the list above, type in the name of another controller, or
'q' to exit
[q] > 3
```

Vamos gerar as páginas para o índice, adicionar, ver e editar. Para isso vamos confirmar digitando `y` e depois `Enter`.

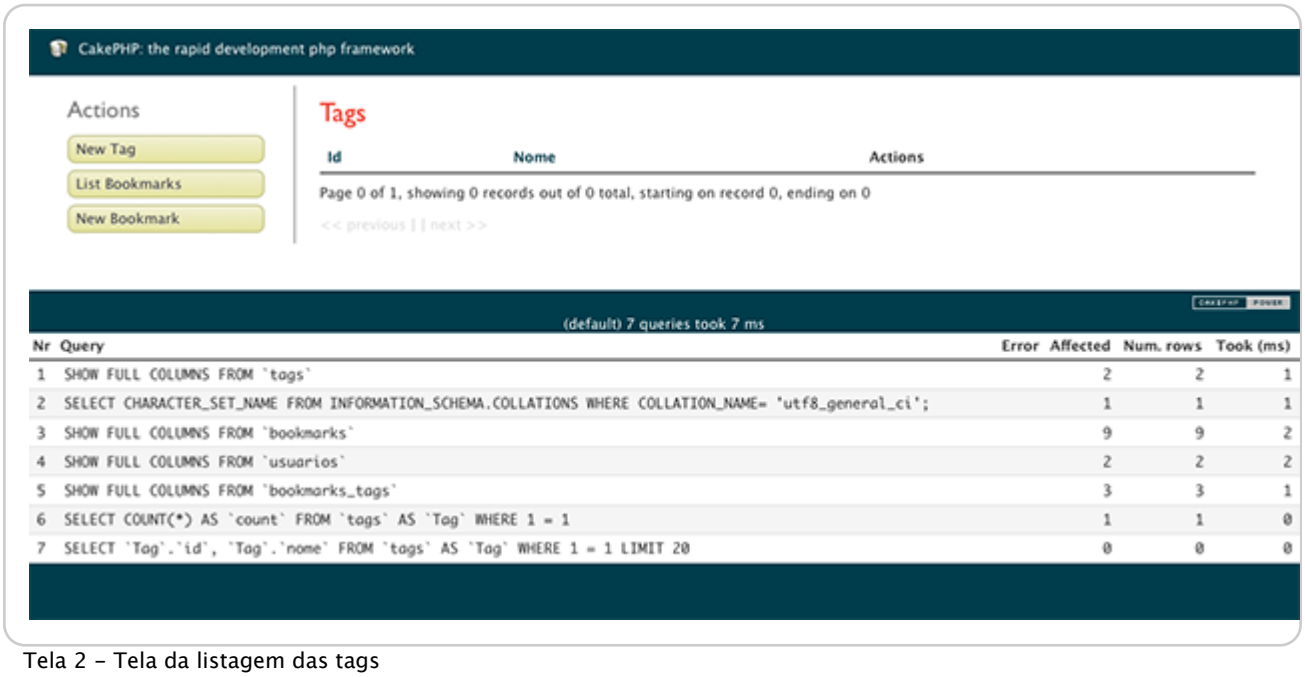
```
Would you like to create some scaffolded views (index, add, view, edit) for this
controller?
NOTE: Before doing so, you'll need to create your controller and model classes
(including associated models). (y/n)
[n] > y
```

Não vamos utilizar as rotas administrativas. Por isso vamos digitar `n` e depois `Enter`.

```
Would you like to create the views for admin routing? (y/n)
[y] > n

Creating file /projetos/bookmarks/views/tags/index.ctp
Wrote /projetos/bookmarks/views/tags/index.ctp

Creating file /projetos/bookmarks/views/tags/view.ctp
Wrote /projetos/bookmarks/views/tags/view.ctp

Creating file /projetos/bookmarks/views/tags/add.ctp
Wrote /projetos/bookmarks/views/tags/add.ctp

Creating file /projetos/bookmarks/views/tags/edit.ctp
Wrote /projetos/bookmarks/views/tags/edit.ctp
---------------------------------------------------------------

View Scaffolding Complete.

---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
```

```
[P]roject
[Q]uit
```

A tela de listagem das tags pode ser acessada pelo endereço `http://localhost/bookmarks/tags/`



Tela 2 – Tela da listagem das tags

Vamos para a última classe de controle. Vamos digitar `v` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> v
---------------------------------------------------------------
Bake View
Path: /projetos/bookmarks/views/
---------------------------------------------------------------
Possible Controllers based on your current database:
1. Bookmarks
2. BookmarksTags
3. Tags
4. Usuarios
```

Vamos gerar as páginas da visão da classe de controle `Usuarios`. Para isso vamos digitar `4` e depois `Enter`.

```
Enter a number from the list above, type in the name of another controller, or
'q' to exit
[q] > 4
```

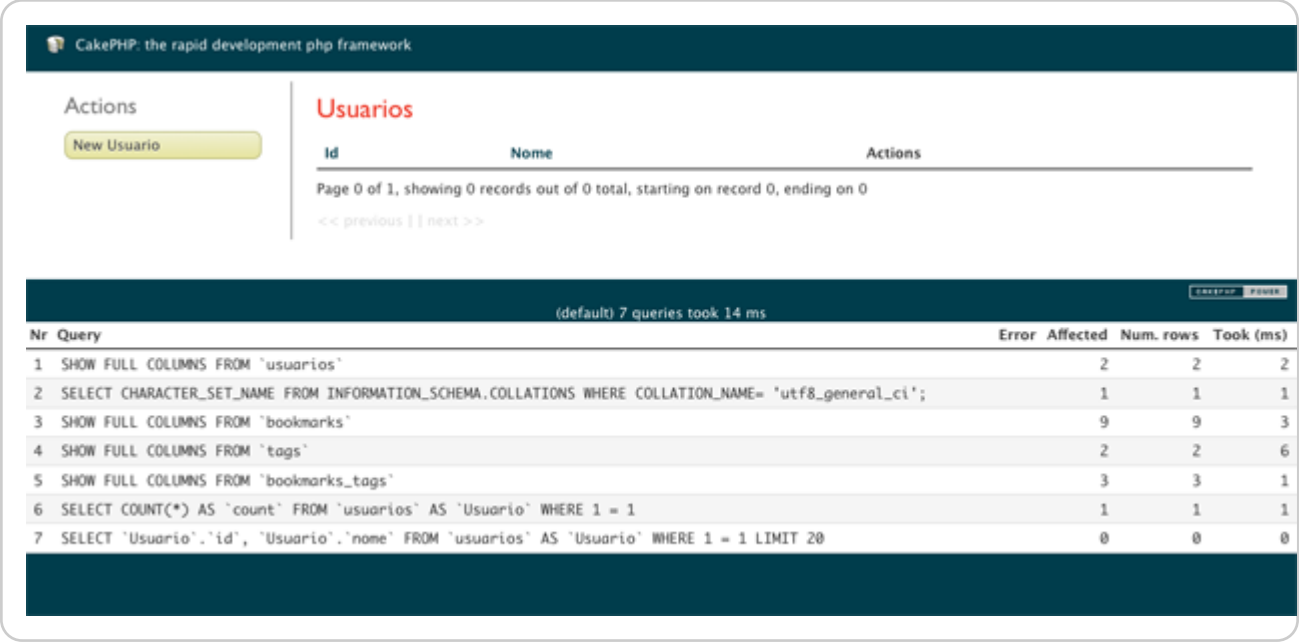
Vamos gerar as páginas para o índice, adicionar, ver e editar. Para isso vamos confirmar digitando `y` e depois `Enter`.

```
Would you like to create some scaffolded views (index, add, view, edit) for this
controller?
NOTE: Before doing so, you'll need to create your controller and model classes
(including associated models). (y/n)
[n] > y
```

Não vamos utilizar as rotas administrativas. Por isso vamos digitar `n` e depois `Enter`.

```
Would you like to create the views for admin routing? (y/n)
[y] > n

Creating file /projetos/bookmarks/views/usuarios/index.ctp
Wrote /projetos/bookmarks/views/usuarios/index.ctp

Creating file /projetos/bookmarks/views/usuarios/view.ctp
Wrote /projetos/bookmarks/views/usuarios/view.ctp

Creating file /projetos/bookmarks/views/usuarios/add.ctp
Wrote /projetos/bookmarks/views/usuarios/add.ctp

Creating file /projetos/bookmarks/views/usuarios/edit.ctp
Wrote /projetos/bookmarks/views/usuarios/edit.ctp
---------------------------------------------------------------

View Scaffolding Complete.

---------------------------------------------------------------
Interactive Bake Shell
---------------------------------------------------------------
[D]atabase Configuration
[M]odel
[V]iew
[C]ontroller
[P]roject
[Q]uit
```

A tela de listagem dos usuários pode ser acessada pelo endereço http://localhost/bookmarks/usuarios/

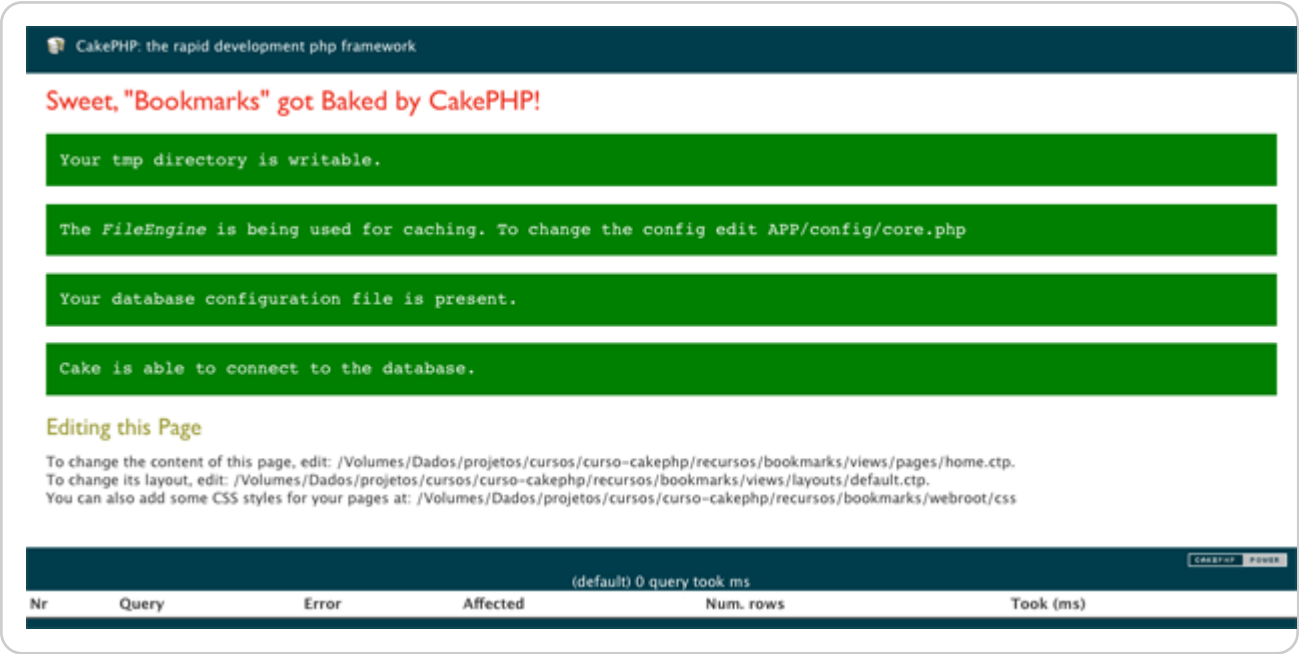


Tela 3 – Tela da listagem dos usuários

Geramos todas as páginas da visão. Agora podemos sair do shell interativo do bake digitando `q` e depois `Enter`.

```
What would you like to Bake? (D/M/V/C/P/Q)
> q
```

Pronto! A aplicação básica CRUD foi gerada com sucesso e sua página principal pode ser acessada pelo endereço http://localhost/bookmarks.



Tela 4 – Tela da página principal da aplicação *Bookmarks*

3 – Aperfeiçoando sua Aplicação

3.1 – Rotas

Este recurso foi desenvolvido para tornar os endereços (URLs) mais amigáveis. O CakePHP já vem configurado com uma rota padrão, está roda padrão define que:

- As URLs passam como primeiro parâmetro o nome do controlador. EX.: www.exemplo.com.br/**usuarios**/
- Caso não seja passado após o controlador nenhum parâmetro será executado a ação **index**.
- Após o controlador é passado a ação desejada (qual método do controle vai ser executado). Ex.: www.exemplo.com.br/usuarios/**add**
- Após a ação podem ser passado parâmetros para os métodos. Ex.:**www.exemplo.com.br/usuarios/edit/1**. No exemplo está sendo chamado a classe de controle usuarios e executando o método edit($id = null), este método está recebendo o valor 1 na variável $id.
- Podem ser passados quantos parâmetros desejar. Ex.: **www.exemplo.com.br/usuario/edit/1/2/3/4**
- Caso seja passado apenas o endereço, sem definir o *controller* e a ação, o cake irá carregar o *controller* ***pages*** e a ação **home**.

3.1.1 – Rota Administrativa

Ao desenvolver aplicações para a internet, muitas vezes é necessário separar uma área com acesso público, de uma área restrita. O Cake possibilita criar está divisão sem a necessidade de criar novos arquivos. Para isto é necessário habilitar o recurso de rotas administrativas.

```
-Endereço de acesso público:
        http://exemplo.com.br/posts/view
-Endereço de acesso restrito (administração)
        http://exemplo.com.br/admin/posts/add
```

Para utilizar o recurso de rotas administrativas em sua aplicação, o primeiro passo é descomentar a seguinte linha do arquivo **app/config/core.php**.

```
//Configure::write('Routing.prefixes', array('admin'));
```

Por padrão o nome da rota administrativa é **admin**, podendo ser alterado. Utilizando o padrão a administração será usada através do endereço: **www.exemplo.com.br/admin/usuarios/index**

O próximo passo é editar a classe de controle criando métodos que estarão disponíveis na rota administrativa, isso é feito colocando a palavra **admin_** antes do nome do método.

```
function admin_add(){...}
function admin_edit($id = null){...}
```

Deve-se agora criar as visões para os métodos, para isto, basta crair o arquivo na basta equivalente ao controller, este aqruivo deve ser iniciado também com **admin_**.

```
Para a ação add de usuarios:
- views/usuarios/admin_add.ctp
```

3.1.2 - Criando suas Próprias Rotas

Para criar suas rotas é necessário ediar o arquivo: **config/routes.php**. Este arquivo já possui algumas rotas básicas.

Para criar uma rota é necessário utilizar o método **Route::Connect()**. Este método recebe três parâmetros:

- Sting URL - endereço que deseja observar.
- Array - instruções para tratar a requisição que está sendo observada.
- Array - instruções (usando expressões regulares) para tratar a requisição.

Como exemplo de uma rota, observe o exemplo abaixo:

```
Router::connect(
        '/promocoes/*',
        array('controller'=>'produtos','action'=>'promocao')
);
```

A rota acima indica que as requisições passadas para o controlador *promocoes* devem ser encaminhadas para o controlador produtos e para ação *promocao*. Desta forma, ao acessar o site: **http://www.exemplo.com.br/promocoes**, o cake irá executar o méotodo `promocao()` do controller `produtos`.

Nota

O uso do asterisco indica que a rota deve aceitar e passar todos os parâmetros que forem passados.

Exemplo de uma Rota com expressão regular:

```
Router::connect(
    '/:controller/:id',
    array('action' => 'view'),
    array('id' => '[0-9]+')
);
```

A rota acima indica que serão passados dois elementos:

1. controlador (:controller), que é um elemento da rota padrão.

2. id (:id), que é um elemento da rota customizada.

O elemento id, por ser um elemento customizado, deve ser especificado por uma expressão regular.

A regra da rota indica que qualquer controlador passado seguido de um número deve executar a ação view passando o id. Logo usando a rota acima, ao acessar http://www.exemplo.com.br/posts/10 o cake irá executar o método `view()` do controller `posts` passando como parâmetro o valor `10`.

Nota

Ao usar :controller na rota, está sendo indicado que a rota será usada para qualquer controller.

3.1.3 - Usando Rotas com Prefixo

Assim como separar a administração do site é um recurso interessante, muitas vezes separar outras partes é interesante também, por exemplo: separar o acesso dos webservices.

Observe o exemplo abaixo:

```
Router::connect(
    '/webservice/:controller/:action/*',
    array('prefix' => 'webservice', 'webservice' => true)
);
```

Está rota irá funcionar exatamente como a rota administrativa, só que a palavra utilizada para a rota será webservice. Logo ao acessar o endereço http://www.exemplo.com.br/webservice/usuarios/index, o cake irá executar o método webservice_index do controlador usuarios.

3.1.4 - Entendendo as Rotas Reversas

Ao utilizar a rota:

```
Router::connect(
        '/promocoes/*',
        array('controller'=>'produtos','action'=>'promocao')
);
```

Ao criar links em sua aplicação, como por exemplo:

```
$html->link('veja mais',
array('controller'=>'produtos','action'=>'promocao',$produto['Produto']['id']));
```

O cake vai gerar o link com o endereço: http://www.exemplo.com.br/promocoes/10. Este comportamento é chamado rota reversa, onde o framework aplica a regra de forma inversa, no lugar de pegar a requisição e encaminhar para a rota definida, o framework verifica as rotas definidas, caso encontre uma regra compatível ele vai gerar o link baseado na regra informada.

3.2 - Vendors

O recurso vendors permite que sejam utilizadas bibliotecas externas, dentro de sua aplicação desenvolvida em CakePHP.

Para adicionar uma biblioteca externa, basta fazer o download e salvar dentro da pasta: **CAMINHO_DO_CAKE/app/vendors/**.

Nota

Por exemplo: A biblioteca phpmailer deverá ser inserida da seguinte forma: /app/vendors

Para utilizar a biblioteca em sua aplicação é necessário importa-la, seguindo os possíveis modelos descritos abaixo:

```
App::import('Vendor', 'phpThumb');
App::import('Vendor', 'phpThumb/phpThumb');
App::import('Vendor', 'Nome', array('file' => 'nome.class.php'));
App::import('Vendor', 'nome', array('file' => 'dir'.DS.'nome.class.php'));
```

Para a biblioteca phpmailer, pode utilizar o comando:

```
<?php
class MailComponent extends Object {
        function send_mail(){
                App::import('Vendor',
'phpmailer',array('file'=>'phpmailer'.DS.'class.phpmailer.php'));

                ...
        }
}
?>
```

Após importada a biblioteca ela fica disponível para ser instanciada e utilizada normalmente.

```
<?php
class MailComponent extends Object {
        function send_mail(){

                App::import('Vendor',
'phpmailer',array('file'=>'phpmailer'.DS.'class.phpmailer.php'));

                $mail = new PHPMailer();
                $mail->FromName = $fromName;
                $mail->AddAddress($to);
                $mail->Body = $body;
                $mail->Subject = $subject;
                $result = $mail->send();
        }
}
?>
```

3.3 - DRY

As aplicações desenvolvidas em CakePHP devem seguir os principios do DRY, que é uma sigla para a frase *Don't Repeat Yorself*, ou em português Não Se Repita.

Ao desenvolver uma aplicação, qualquer repetição de código-fonte é proíbida. Sempre que o desenvolvedor sentir a necessidade de copiar qualquer trecho de código, ele pode considerar a idéia de criar um componente, para que este seja reusado em qualquer local de sua camada.

Os componentes no CakePHP são divididos nas camadas que eles atuam, logo:

1. Behavior são componentes compartilhados pelas classes da camada de modelo
2. Components são componentes compartilhados pelas classes da camada de controle
3. Helpers são componentes compartilhados pela camada de visão

3.4 - Behavior

Alguns Behaviors são disponíbilizados junto ao *framework*, outros podem ser baixados pela internet, ou até mesmo criados pelo desenvolvedor, como será descrito no capitulo 7.

Este capítulo apresenta os Bahaviors que são disponibilizados junto ao *framework*.

3.4.1 - ACL

ACL é sigla para Access Control List, ou em português Lista de Controle de Acesso. É um design pattern utilizado para determinar ao que o usuário tem acesso no sistema.

Para entender este padrão de projeto é necessário entender dois conceitos:

ACO - *Access Control Object* são objetos que são controlados. Ex: adicionar usuarios, grupos, etc.

ARO - *Access Request Object* lista dos objetos que podem fazer a requisição. Ex: grupo de usuários, usuários.

Para ilustrar estes conceitos é apresentado uma hierarquia de grupos/usuários (ou AROs) de uma aplicação:

- Administradores
  - Cadu
  - Daniel
- Analista de Sistemas
  - José
  - João
- Programadores
  - Geraldo
  - Matheus

Está aplicação tem as seguintes funções (ACOs):

- Usuários
  - add
  - edit
  - index
  - view
- Especificação de Requisitos
  - add
  - edit
  - index
  - view
- API de classes
  - add
  - edit
  - index
  - view

A tabela abaixo define as permissões de acesso desta aplicação:

ARO/ACO	Usuários	Especificação de Requisitos	API de classes
Cadu	Permitido (allow)	Permitido (allow)	Permitido (allow)
Daniel	Permitido (allow)	Permitido (allow)	Permitido (allow)
José		Permitido (allow)	
João		Permitido (allow)	
Geraldo			Permitido (allow)
Matheus			Permitido (allow)

O Behavior ACL, permite adicionar o comportamento de ACOs e AROs em seu modelo. Para isto deve-se criar as tabelas usadas pelo ACL.

```
cake acl initdb
```

Após criada as tabelas, precisamos inserir o comportamento no modelo:

```
class Grupo extends AppModel {
    var $actsAs = array('Acl'=>'requester');

        function parentNode() {
                return null;
        }
}
```

Este comportamento sempre executa o método parentNote() da classe de modelo, isto se deve ao fato de que para ser possível criar uma estrutura de árvore é preciso que seja informado o nodo pai. Este caso sempre retorna null, pois os grupos sempre serão nodos pais.

A classe de modelo Usuario também terá o Behavior ACL e ela gerá os nodos filhos dos nodos de Grupo.

```
class Usuario extends AppModel {
        var $name = 'Usuario';
        var $belongsTo = array('Grupo');
        var $actsAs = array('Acl' => 'requester');

        function parentNode() {
                if (!$this->id && empty($this->data)) {
                        return null;
                }
                $data = $this->data;
                if (empty($this->data)) {
                        $data = $this->read();
                }
                if (!$data['Usuario']['grupo_id']) {
                        return null;
                } else {
                        return array('Usuario' => array('id' => $data['Usuario']
['grupo_id']));
                }
        }
}
```

O comportamento de Aco pode ser inserido de forma similar ao ACO.

```
class AlgumModel extends AppModel {
        var $name = 'AlgumModel';
        var $actsAs = array('Acl' => 'controlled');

        function parentNode() {
                return array('AlgumModel' => array('id' => $data['AlgumModel']
['parent_id']));
        }
}
```

3.4.2 - Containable

Quando um modelo tem diversos relacionamentos, o comportamento padrão do cake é trazer os dados de todos os relacionamentos, isto pode gerar um acesso lento ao banco de dados.

Este behavior visa facilitar alterações nas associações dos modelos, de forma que seja possível trazer somente as informações desejadas.

Para que este behavior seja utilizado, basta adicionar a seguinte linha ao seu modelo.

```
class Usuario extends AppModel {
    var $actsAs = array('Containable');
}
```

Uma vez que o behavior foi configurado no model é possível utilizá-lo ao fazer uma busca.

```
$this->Usuario->contain();
$this->Usuario->find('all');
```

Está busca terá o mesmo resultado da execução do find com **recursive=-1**, sendo este o resultado:

```
[0] => Array
        (
            [Usuario] => Array
                (
                    [id] => 1
                    [nome] => Primeiro Nome
                    [grupo_id] => 1
                    [created] => 2009-05-18 00:00:00
                )
        )
[1] => Array
        (
            [Usuario] => Array
                (
                    [id] => 2
                    [nome] => Segundo Nome
                    [grupo_id] => 1
                    [created] => 2009-05-19 00:00:00
                )
        )
```

Caso seja do interesse trazer algum model relacionado, basta indicar qual modelo deve ser adicionado na busca.

```
$this->Usuario->contain('Grupo');
$this->Usuario->find('all');
//ou
$this->Usuario->find('all',array('contain'=>'Grupo'));
```

A busca terá o seguinte retorno:

```
[0] => Array
        (
            [Grupo] => Array
                (
                    [id] => 1
                    [nome] => Grupo de Teste
                )
            [Usuario] => Array
                (
```

```
                    [id] => 1
                    [nome] => Primeiro Nome
                    [grupo_id] => 1
                    [created] => 2009-05-18 00:00:00
                )
        )
[1] => Array
        (
            [Grupo] => Array
                (
                    [id] => 1
                    [nome] => Grupo de Teste
                )
            [Usuario] => Array
                (
                    [id] => 2
                    [nome] => Segundo Nome
                    [grupo_id] => 1
                    [created] => 2009-05-19 00:00:00
                )
        )
```

Ainda é possível fazer uma busca com condições mais complexas.

```
$this->Post->find('all',array('contain'=>
                                    array('Post'=>
                                       array('conditions'=>
                                          array('created <'=>'2009-01-01')))));
```

3.4.3 - Translate

Aplicações que serão acessadas em diversos países, precisam ser traduzidos para diversas línguas. O processo de internacionalizar uma aplicação envolve várias camadas do desenvolvimento. O recurso de internacionalização e localização é apresentado no capítulo 4.

3.4.4 - Tree

O Behavior Tree deve ser utilizado quando é necessário criar uma estrutura de árvore hierárquica nos registros de uma tabela. Por exemplo: Categorias de produtos que possuem sub-categorias.

Para utilizar este comportamento o primeiro passo é criar, ou modificar sua tabela adicionando os seguintes campos:

```
parent_id - campo que guarda o item superior.
lft - campo que guarda o valor esquerdo do registro.
rght - campo que guarda o valor direito do registro.
```

Exemplo de tabela com os campos:

```
CREATE TABLE categorias (
        id INTEGER(10) UNSIGNED NOT NULL AUTO_INCREMENT,
        parent_id INTEGER(10) DEFAULT NULL,
        lft INTEGER(10) DEFAULT NULL,
        rght INTEGER(10) DEFAULT NULL,
        nome VARCHAR(255) DEFAULT '',
        PRIMARY KEY  (id)
```

```
);
```

Com a tabela preparada para salvar os dados corretamente, precisamos adicionar o behavior a nosso modelo.

```
class Categoria extends AppModel {
        var $name = 'Categoria';
        var $actsAs = array('Tree');
}
```

Nota

Pronto!!! As operações com o banco de dados (salvar, atualizar e apagar) já funcionam respeitando a estrutura de árvore.

Para facilitar o gerenciamento da árvore este behavior traz alguns métodos.

3.4.4.1 - Children

Este método recebe dois parâmetros:

1. id - chave primária do nodo que desejá trazer os filhos.
2. filhos diretos - caso este parâmetro não seja passado, ou seja falso a busca retornará todos os filhos. Caso seja verdadeiro, a busca traz somente os filhos diretos.

```
$todoOsFilhos = $this->Categoria->children(1);
// -- ou --
$this->Categoria->id = 1;
$todoOsFilhos = $this->Categoria->children();

$filhosDiretos = $this->Categoria->children(1, true);
```

3.4.4.2 - ChildCount

Funciona de forma similar ao método children(), sendo que retorna o número de filhos no lugar de retornar os filhos.

```
$todoOsFilhos = $this->Categoria->childCount(1);
// -- ou --
$this->Categoria->id = 1;
$todoOsFilhos = $this->Categoria->childCount();

$filhosDiretos = $this->Categoria->childCount(1, true);
```

3.4.4.3 - generatetreelist

generatetreelist (&$model, $conditions=null, $keyPath=null, $valuePath=null, $spacer= '_', $recursive=null)

Este método funciona de forma similar ao **find('list')**, mas o resultado é gerado com a estrutura de árvore usando o caracter indicado no parâmetro **$spacer**.

```
array(
        [1] =>  "Informática",
        [2] =>  "_Computadores",
        [3] =>  "__Desktops",
```

```
        [4] =>  "___HP",
        [16] => "___Dell",
        [6] =>  "__Notebooks",
        [7] =>  "___HP",
        [8] =>  "___Dell",
        [9] =>  "_Suprimentos",
        [13] => "__Impressoras",
        [14] => "___Cartuchos",
        [15] => "___Tonner",
        [17] => "Livros",
        [5] =>  "_PHP"
)
```

3.4.4.4 - getparentnode

O método retorna o nodo pai do id passado, ou falso caso ele sejá pai.

```
$pai = $this->Categoria->getparentnode(2); //<- id do item Computadores
// $pai contém os dados de Informática
```

3.4.4.5 - getpath

O método retorna todos os nodos pais do id passado.

```
$resultado = $this->Categoria->getpath(8);

->Informática
        ->Computadores
                ->Notebooks
                        ->Dell
```

3.4.4.6 - moveDown

movedown($Model, $id = NULL, $number = 1)

O método move o nodo para baixo, deve ser informado o id do nodo e quantas posições ele deve ser movido.

3.4.4.7 - moveUp

moveup($Model, $id = NULL, $number = 1)

O método move o nodo para cima, deve ser informado o id do nodo e quantas posições ele deve ser movido.

3.4.4.8 - removeFromTree

removefromtree($Model, $id = NULL, $delete = false)

Remove o nodo da árvore e move todos os filhos para um nível acima

3.4.4.9 - reorder

reorder($Model, $options = array ( ))

Reordena os nodos de uma árvore de acordo com o campo e a direção passada pelos parâmetros. Este método não altera o pai de nenhum nodo.

3.4.4.10 - recover

recover($Model, $mode = 'parent', $missingParentAction = NULL)

Recupera a estrutura de uma árvore que esteja com algum registro errado.

3.4.4.11 - verify

verify(&$model)

Retorna verdadeiro se a árvore estiver com a estrutura correta, ou um array com as chaves lft/rght erradas.

3.5 - Components

3.5.1 - ACL

Este componente foi desenvolvido para auxiliar o uso de ACL na aplicação. Este componente permite liberar, negar e verificar permissões.

O primeiro passo para utilizá-lo é incluir em sua classe de controle.

```
class UsuariosController extends AppController
{
        var $components = array('Acl');
}
```

Importante

Ao usar ACL em sua aplicação é aconselhável que ele seja inserido no AppController, desta forma o componente estará disponível em toda a aplicação.

3.5.1.1 - Atribuindo permissões

Para permitir o acesso de um usuário a um controle e/ou ação.

```
$this->Acl->allow('Cadu', 'Usuarios');
$this->Acl->allow('Daniel', 'Usuarios','*');
$this->Acl->allow('José', 'Usuarios','read');
```

Para negar o acesso de um usuário a um controle e/ou ação.

```
$this->Acl->deny('Cadu', 'Usuarios');
$this->Acl->deny('Daniel', 'Usuarios','*');
$this->Acl->deny('José', 'Usuarios','read');
```

3.5.1.2 - Verificando permissões

Para verificar se um usuário tem permissão de acesso.

```
$this->Acl->check('usuarios/Cadu', 'Usuarios');
$this->Acl->check('usuarios/Cadu', 'Usuarios','create');

$this->Acl->check(array('model' => 'Usuario', 'foreign_key' => 2356),
'Usuarios');
```

3.5.2 - Authentication

Sistema de autenticação de usuários é uma parte comum em diversos sistemas web. O Cake fornece um componente pronto para criar toda parte de autenticação, sendo possível a integração com o ACL de forma a criar níveis complexos de segurança.

Assim como todos os componentes o primeiro passo é inseri-lo no array de componentes:

```
class UsuariosController extends AppController {
    var $components = array('Auth');
```

Abaixo seguem as convenções do componente, lembrando que não são obrigatórias, mas seu uso dispensa a configuração.

Local	Regra
Banco de Dados	Nome da tabela deve ser users
Banco de Dados	Nome dos campos devem ser username e password
Banco de Dados	As senhas são criptografadas utilizando SHA1, o campo password deve possuir 40 caracteres.

Caso sua aplicação esteja respeitando as convenções o login é feito editando os arquivos:

```
//Classe de controle de User
class UsersController extends AppController {

    var $name = 'Users';
    var $components = array('Auth');

    function login() {
    }

    function logout() {
        $this->redirect($this->Auth->logout());
    }
}
```

```
//Visão para login de User
if  ($this->Session->check('Message.auth'))  //Mensagens do componente
        $this->Session->flash('auth');
echo $this->Form->create('User', array('action' => 'login'));
echo $this->Form->input('username');
echo $this->Form->input('password');
echo $this->Form->end('Login');
```

Caso não esteja sendo usada as convenções, deve se configurar o componente conforme o exemplo abaixo:

```
class UsuariosController extends AppController {
    var $components = array('Auth');

    function beforeFilter() {
        //Indica a tabela usada para a autenticação
        $this->Auth->userModel = 'Usuario';

        //Indica os campos usados para a autenticação
```

```
        $this->Auth->fields = array(
            'username' => 'login',
            'password' => 'senha'
            );
        //Alterando a criptografia usada
        Security::setHash('md5'); // ou sha1 ou sha256.
    }
}
```

Conhecendo os métodos do componente de autenticação

3.5.2.1 - action

Caso a aplicação esteja usando ACL é possível pegar o nodo do ACO (objeto a ser acessado) que deseja ser acessado.

action (string $action = ':controller/:action')

```
$aco = $this->Auth->action('usuarios/edit');
```

3.5.2.2 - allow

O método é utilizado para permitir o acesso a ações, ao utilizar o método o componente de autenticação não faz nenhuma verificação nos métodos especificados.

```
//passando somente uma acão
$this->Auth->allow('index');

//passando diversas ações
$this->Auth->allow('add', 'edit', 'delete');

//permitindo qualquer ação
$this->Auth->allow('*');
```

3.5.2.3 - deny

Caso exista a necessidade de negar um método que esteja permitido (allow).

No exemplo abaixo é liberado a ação delete, mas no método estaAutorizado ele nega a permissão para usuários que não sejam administradores.

```
    function beforeFilter() {
        $this->Auth->authorize = 'controller';
        $this->Auth->allow('delete');
    }

    function estaAutorizado() {
        if ($this->Auth->user('role') != 'admin') {
            $this->Auth->deny('delete');
        }
    }
```

3.5.2.4 - hashPasswords

hashPasswords ($data)

Este método procura os campos username e password em um array, caso seja encontrado ele aplica o hash no campo password.

Nota

Os campos username e password, assim como o modelo, podem ser alterados, como foi demonstrado no início deste tópico.

```
    $data['User']['username'] = 'contato@2km.com.br';
    $data['User']['password'] = 'senha';
    $senhaComHash = $this->Auth->hashPasswords($data);
    print_r($senhaComHash);
    /* retorno:
    Array
    (
        [User] => Array
        (
            [email] => contato@2km.com.br
            [password] => 7751a23fa55170a57e90374df13a3ab78efe0e99
        )
    )

    */
```

3.5.2.5 - mapActions

Caso esteja usando Acl com CRUD, pode ser desejável mapear ações que estejam fora do padrão CRUD.

```
$this->Auth->mapActions(
        array(
                'create' => array('adicionar'),
                'read' => array('ler', 'visualizar'),
                'update' => array('editar'),
                'delete' => array('apagar')
        )
);
```

3.5.2.6 - login

Caso esteja sendo usado um login baseado em AJAX, este método pode ser usado para criar a autenticação de forma manual, senão for infomado nenhum parâmetro o método utiliza os dados postados para o controle.

login($data = null)

3.5.2.7 - logout

Este método permite deslogar um usuário autenicado no sistema.

```
$this->redirect($this->Auth->logout());
```

3.5.2.8 - password

password (string $password)

Este método retorna o hash da string passada.

```
$this->data['Ususuario']['senha']='senha'
```

```
echo $this->Auth->password($this->data['Ususuario']['senha']);
//a saida será: 7751a23fa55170a57e90374df13a3ab78efe0e99
```

3.5.2.9 - user

user(string $key = null)

Este método retorna informações sobre o usuário logado.

```
if ($this->Auth->user('grupo') == 'admin') {
    $this->flash('O usuário é administrador');
}
//todos os dados do usuário
$data['User'] = $this->Auth->user();
//retorna null caso o usuário não esteja autenticado.
```

Atributos do componente de autenticação

3.5.2.10 - userModel

Atributo que identifica a classe de modelo que será utilizada para fazer a autenticação.

```
$this->Auth->userModel = 'Usuario';
```

3.5.2.11 - fields

Atributo que identifica os campos que serão usados na autenticação.

```
$this->Auth->fields = array('username' => 'email', 'password' => 'senha');
```

3.5.2.12 - userScope

Define alguma regra adicional para o login ser válido.

```
$this->Auth->userScope = array('Usuario.ativo' => 'S');
```

3.5.2.13 - loginAction

Define o endereço usado para a ação de login.

```
$this->Auth->loginAction = array('admin' => false,
                                 'controller' => 'usuarios',
                                 'action' => 'login');
```

3.5.2.14 - loginRedirect

Caso um usuário não autenticado tente entrar em um endereço do sistema, o componente guarda este endereço na sessão e após o login do usuário ele é redirecionado para o endereço desejado. Caso o usuário tenha entrado diretamente na tela de login, ao ser autenticado ele será redirecionado para a página configurada neste atributo.

```
$this->Auth->loginRedirect = array('controller' => 'usuarios', 'action' =>
'home');
```

3.5.2.15 - logoutRedirect

Atributo que define o redirecionamento da ação de logout.

```
$this->Auth->logoutRedirect = array(Configure::read('Routing.admin') => false,
                                'controller' => 'usuarios',
                                'action' => 'logout');
```

3.5.2.16 - loginError

Este atributo define a mensagem de erro que será apresentada ao usuário quando o login falha.

```
$this->Auth->loginError = "O login falhou, favor verificar o login/senha.";
```

3.5.2.17 - authError

Este atributo define a mensagem de erro que será apresentada ao usuário quando ele tenta acessar uma ação que não tem permissão.

```
$this->Auth->authError = "Você não tem permissão para executar a ação.";
```

3.5.2.18 - autoRedirect

Este atributo define se o componente deve redirecionar o usuário automaticamente após o login.

Deve ser utilizado nos casos onde existe interesse de adicionar alguma lógica antes do usuário ser redirecionado.

```
function beforeFilter() {
        $this->Auth->autoRedirect = false;
}

function login() {
        if ($this->Auth->user()) {
                if (!empty($this->data)) {
                        $cookie = array();
                        $cookie['username'] = $this->data['User']['username'];
                        $cookie['password'] = $this->data['User']['password'];
                        $this->Cookie->write('Auth.User', $cookie, true, '+2
weeks');
                        unset($this->data['User']['remember_me']);
                }
                $this->redirect($this->Auth->redirect());
        }
        if (empty($this->data)) {
                $cookie = $this->Cookie->read('Auth.User');
                if (!is_null($cookie)) {
                        if ($this->Auth->login($cookie)) {
                                $this->Session->del('Message.auth');
                                $this->redirect($this->Auth->redirect());
                        }
                }
        }
}
```

3.5.2.19 - authorize

Este atributo pode ter 4 valores:

1. controller
2. model
3. actions
4. crud

Este atributo identifica como será feita a verificação de permissão do usuário às partes do sistema.

Caso o authorize seja controller:

```
$this->Auth->authorize = 'controller';
```

É necessário criar o método isAuthorized() no controller para verificar a permissão de acesso.

```
function isAuthorized() {
    if ($this->action == 'delete') {
        if ($this->Auth->user('role') == 'admin') {
            return true;
        } else {
            return false;
        }
    }
    return true;
}
```

Caso o authorize seja model:

```
$this->Auth->authorize = 'model';
```

É necessário criar o método isAuthorized() no model para verificar a permissão de acesso.

```
class Usuario extends AppModel {
    function isAuthorized($usuario, $controller, $action) {
        switch ($action) {
            case 'default':
                return false;
                break;
            case 'delete':
                if ($usuario['Usuario']['grupo'] == 'admin') {
                    return true;
                }
                break;
        }
    }
}
```

Caso o authorize seja actions:

```
$this->Auth->authorize = 'actions';
```

Esta configuração não exige nenhuma codificação adicional, o componente Auth vai verificar a permissão utilizando ACL.

Caso authorize seja crud:

```
$this->Auth->authorize = 'crud';
```

Esta configuração não exige nenhuma codificação adicional, o componente Auth vai verificar a permissão utilizando o crud do ACL.

3.5.2.20 - sessionKey

Nome do array de sessão que armazena os dados do usuário autenticado. O Padrão é Auth.

```
$this->Auth->sessionKey = 'Autenticado';
```

3.5.2.21 - ajaxLogin

Caso esteja sendo utilizado requisições ajax, defina neste atributo o elemento da visão que deve ser renderizado, caso a sessão não esteja válida.

3.5.2.22 - authenticate

É uma referencia ao objeto responsável por aplicar hash ao campo senha, caso seja alterado o mecanismo de hash ele deverá ser chamado.

3.5.3 - Cookies

Este componente permite criar e gerenciar Cookies em sua aplicação.

```
var $components    = array('Cookie');

//------valores que podem ser configurados.--------
function beforeFilter() {
        //nome do cookie
        $this->Cookie->name = 'CakeCookie';

        //tempo de validade de um cookie
        $this->Cookie->time =  3600;  // em segundos

        //Parte que o cookie será aplicado. Valor padrão / (toda a aplicação)
        $this->Cookie->path = '/bakers/preferences/';

        //Domínio que pode gerenciar o cookie
        $this->Cookie->domain = 'example.com';

        //Cookie só é válido usando ssl (https)
        $this->Cookie->secure = true;

        //Chave de criptografia, é recomendado que sejá randomica
        $this->Cookie->key = 'qSI232qs*&sXOw!';
}
```

3.5.3.1 - write

Este método cria variáveis e atribui valores nos cookies de sua aplicação. O primeiro parâmetro é o nome da variável, o segundo é o valor. Por padrão todos os cookies são criptografados, caso não seja necessário criptografar basta passar falso no terceiro parâmetro. O tempo de duração do cookie pode ser informado no quarto parâmetro (em segundos).

write(mixed $key, mixed $value, boolean $encrypt, mixed $expires)

```
$this->Cookie->write('Grupo','Admin');
$this->Cookie->write('nome','Daniel',false, 3600);
$this->Cookie->write('nome','Cadu',false, '1 hour');
```

3.5.3.2 - read

Para ler as variáveis salvas no cookie deve-se utilizar o método read, que recebe como parâmetro o nome da variável.

read(mixed $key)

```
$grupo = $this->Cookie->read('Grupo');
$nome = $this->Cookie->read('nome');
```

3.5.3.3 - del

Apaga a variável informada do cookie.

del(mixed $key)

```
$this->Cookie->del('Grupo');
$this->Cookie->del('nome');
```

3.5.3.4 - destroy

Este método destroi o Cookie.

destroy()

3.5.4 - Email

Este componente permite enviar e-mails.

3.5.4.1 - Atributos

to	endereço que vai receber a mensagem
cc	array - endereços que vão receber uma cópia da mensagem
bcc	array - endereços que vão receber uma cópia oculta da mensagem
replyTo	endereço de resposta ao e-mail
from	endereço de origem do e-mail
subject	Assunto do e-mail
template	O elemento que será usado para renderizar a mensagem (localizado em app/views /elements/email/html/ e app/views/elements/email/text/)
layout	O layout utilizado para renderizar o e-mail(localizado em app/views/layouts/email /html/ e app/views/layouts/email/text/)
lineLength	Tamanho máximo que cada linha deve ter. O padrão é 70. (inteiro)
sendAs	Qual o tipo de mensagem é seu e-mail: text, html, ou both (os dois)
attachments	array - com o endereço relativo e absoluto dos arquivos
delivery	quem vai disparar o e-mail: mail, smtp e debug
smtpOptions	array - define as opções do servidor smtp (port, host, timeout, username, password)

3.5.4.2 - Enviando E-mails

```
$this->Email->from = 'Cadu ';
$this->Email->to = 'Alguém ';
$this->Email->subject = 'Teste';
$mensagem = "Olá!!!";
```

```
$this->Email->send($mensagem);
```

Ao enviar várioe e-mail em um loop deve-se chamar o método reset() após cada envio.

```
foreach($usuarios as $usuario){
        $this->Email->from = 'Cadu ';
        $this->Email->to = $usuario['Usuairo']['nome'].'<'.$usuario['Usuairo']
['email'].'>';
        $this->Email->subject = 'Teste';
        $mensagem = "Olá!!!";
        $this->Email->send($mensagem);
        $this->Email->reset();
}
```

3.5.5 - Request Handling

Este componente é utilizado para gerenciar requisições HTTP, basicamente ele obtém informações adicionais de cada requisição, desta forma é possível tomar decisões de como a requisição será tratada.

Ao utilizar o componente RequestHandler todas as requisições **ajax** já são identificadas e o **layout** é alterado automaticamente.

```
class GrupoController extends AppController {

        var $components = array('RequestHandler');

}
```

Após incluído o componente é possível obter informações da requisição utilizando os métodos:

3.5.5.1 - accepts

Este método verifica o tipo de conteúdo o cliente aceita, pode ser passado um tipo, ou um array de tipos.

accepts ($type = null)

```
class ArtigosController extends AppController {

    var $components = array('RequestHandler');

    function beforeFilter () {
        if ($this->RequestHandler->accepts('html')) {
            // Caso o cliente aceite html executa esta parte
        } elseif ($this->RequestHandler->accepts('xml')) {
            // Caso aceite xml
        }
        if ($this->RequestHandler->accepts(array('xml', 'rss', 'atom'))) {
            // caso aceite xml, rss e atom
        }
    }
}
```

3.5.5.2 - isAjax

Retorna true caso o cliente tenha enviado no header da requisição a opção X-Requested-Header igual a XMLHttpRequest.

boolean isAjax()

3.5.5.3 - isSSL

Retorna true se a requisição foi feita usando uma conexão SSl.

boolean isSSL()

3.5.5.4 - isXml

Retorna true se a requisição aceita xml como resposta.

boolean isXml()

3.5.5.5 - isRss

Retorna true se a requisição aceita rss como resposta.

boolean isRss()

3.5.5.6 - isAtom

Retorna true se a requisição aceita atom como resposta.

boolean isAtom()

3.5.5.7 - isMobile

Retorna true se a requisição foi feita por um dispositivo móvel, ou wap.

boolean isMobile()

Lista dos clientes móveis reconhecidos pelo método:

- iPhone
- MIDP
- AvantGo
- BlackBerry
- J2ME
- Opera Mini
- DoCoMo
- NetFront
- Nokia
- PalmOS
- PalmSource
- portalmmm
- Plucker
- ReqwirelessWeb
- SonyEricsson
- Symbian
- UP.Browser
- Windows CE
- Xiino

3.5.5.8 - isWap

Retorna true se o cliente aceitar wap.

boolean isWap()

É possível também detectar o tipo de requisição com os métodos:

3.5.5.9 - isPost

Retorna true a requisição HTTP for do tipo POST.

boolean isPost()

3.5.5.10 - isPut

Retorna true a requisição HTTP for do tipo PUT.

boolean isPut()

3.5.5.11 - isGet

Retorna true a requisição HTTP for do tipo GET.

boolean isGet()

3.5.5.12 - isDelete

Retorna true a requisição HTTP for do tipo DELETE.

boolean isDelete()

Nota

Detectar o tipo da requisição é importante ao utilizar *webservices*, como será demonstrado no capítulo 6.

Obtendo informações do cliente

3.5.5.13 - getClientIP

Retorna o endereço IP do cliente que está fazendo a requisição.

string getClientIP()

3.5.5.14 - getReferrer

Retorna o nome do domínio do local que está fazendo a requisição.

string getReferrer()

3.5.5.15 - getAjaxVersion

Retorna a versão da biblioteca proptotype que está sendo utilizada.

string getAjaxVersion()

O RequestHandler também oferece métodos para responder a requisição:

3.5.5.16 - setContent

Este método define o Content-type da requisição.

setContent($name, $type = null)

Listagem dos Content-types:

- javascript - text/javascript
- js - text/javascript
- json - application/json
- css - text/css
- html - text/html, */*
- text - text/plain
- txt - text/plain

- csv - application/vnd.ms-excel, text/plain
- form - application/x-www-form-urlencoded
- file - multipart/form-data
- xhtml - application/xhtml+xml, application/xhtml, text/xhtml
- xhtml-mobile - application/vnd.wap.xhtml+xml
- xml - application/xml, text/xml
- rss - application/rss+xml
- atom - application/atom+xml
- amf - application/x-amf
- wap - text/vnd.wap.wml, text/vnd.wap.wmlscript, image/vnd.wap.wbmp
- wml - text/vnd.wap.wml
- wmlscript - text/vnd.wap.wmlscript
- wbmp - image/vnd.wap.wbmp
- pdf - application/pdf
- zip - application/x-zip
- tar - application/x-tar

3.5.5.17 - prefers

Determina qual o Content-type o cliente prefere. Senão for passado nenhum parâmetro o tipo mais apropriado é passado, caso seja passado vários tipos em um array, o primeiro tipo que o cliente aceita será usado.

prefers($type = null)

3.5.5.18 - renderAs

Define o layout e o template usado para o tipo de Content-type solicitado. Ex: ajax, html, xml, json, etc.

renderAs($controller, $type)

3.5.5.19 - respondAs

Define o header de resposta.

respondAs($type, $options)

3.5.5.20 - responseType

Retorna o Content-type corrente.

3.5.5.21 - mapType($cType = array())

Mapeia um Content-type para um apelido.

3.5.6 - Security Component

Este componente é responsável por adicionar diversos recursos de segurança a sua aplicação, evitando ataques XSS, dados maliciosos, entre outros ataques. A explicação detalhada deste componente será apresentado no capítulo 5.

3.5.7 - Sessions

O componente Sessions é usado para gerenciar sessões na aplicação.

3.5.7.1 - write

write($name, $value)

Este método escreve valores na sessão, recebe como primeiro parâmetro o nome da variável e o segundo o valor.

```
$this->session->write('usuarioId',$id);
```

3.5.7.2 - read

read($name)

Este método recupera valores na sessão, recebe como primeiro parâmetro o nome da variável.

```
$id = $this->session->read('usuarioId');
```

3.5.7.3 - setFlash

setFlash($message, $layout = 'default', $params = array(), $key = 'flash')

Este método é usado para retornar mensagens para o usuário.

```
setFlash('O usuário foi salvo com sucesso!');
setFlash('O usuário foi salvo com sucesso!', 'default', array('class' =>
'success_class'))
```

3.5.7.4 - check

check($name)

Este método é usado para verificar se a variável existe na sessão.

```
if($this->Session->check('usuarioId')){
        //faz alguma coisa
} else {
        //faz outra coisa
}
```

3.5.7.5 - delete

delete($name) ou del($name)

Este método é usado para apagar uma variável da sessão.

```
$this->Session->delete('usuarioId');
```

3.5.7.6 - destroy

destroy()

Este método é usado para destruir a sessão.

```
$this->Session->destroy();
```

3.6 - Helpers

O CakePHP possui alguns helpers básicos que auxiliam no desenvolvimento de uma aplicação.

Os Helpers podem ser vistos com a camada de apresentação da aplicação. Eles possuem lógica de apresentação que é compartilhada entre várias visões, elementos ou layouts. Abaixo segue a lista dos helpers básicos utilizados no desenvolvimento de uma aplicação com o CakePHP como o Form, Html e JavaScript.

3.6.1 - AJAX

Para implementar AJAX e efeitos na aplicação, o CakePHP utiliza as bibliotecas Prototype e script.aculo.us. Portanto, antes de tudo é preciso importar essas bibliotecas para o diretório

/app/webroot/js. Depois disso é preciso incluir os helpers ajax e javascript no controller.

```
<?php
class ArtesanatosController extends AppController {
        var $name = 'Artesanatos';
        var $helpers = array('Html','Ajax','Javascript');
}
?>
```

Depois que os helpers já foram incluídos no controller é preciso importar as bibliotecas Prototype e script.aculo.us no layout ou view onde estes recursos serão utilizados.

```
<?php
        echo $javascript->link('prototype');
        echo $javascript->link('scriptaculous');
?>
```

Agora os recursos AJAX já estão disponíveis na aplicação. Se o componente RequestHandler tiver sido incluído no controller, o CakePHP irá automaticamente utilizar o layout Ajax quando uma requisição AJAX tiver sido feita.

```
<?php
class ArtesanatosController extends AppController {
        var $name = 'Artesanatos';
        var $helpers = array('Html','Ajax','Javascript');
        var $components = array( 'RequestHandler' );
}
?>
```

3.6.2 - Cache

Este helper ajuda a fazer cache em layouts e views, com isso é possível reduzir o processamento de dados. É possível escolher o tipo de armazenamento que será feito. Quando ocorre uma requisição, o CakePHP procura primeiro se a página já foi armazenada em cache. Se já tiver sido o resto do processo para ler a url é suprimido. Toda a parte do layout que não estiver em cache é processada normalmente.

A idéia de cache é a de armazenar localmente dados de forma a não sobrecarregar o servidor. Por exemplo, se houver um resultado de pesquisa ao banco de dados que consuma muito recurso e que necessite acontecer diversas vezes, e mais prático armazenar esses dados em cache, reduzindo o consumo do servidor. Ao fazer cache temos que ter em mente que deve se armazenar dados por um determinado tempo e que façam sentido de ser armazenados para serem reutilizados.

É possível definir em app/config/core.php o tipo de cache que será feito ao usar CakePHP. Na tabela a seguir estão relacionados as diferentes opções:

Tipo	Descrição
File	Este é o tipo de cache padrão utilizado pelo CakePHP. Nesse modo, o CakePHP irá escrever arquivos no sistema de arquivos. Possui diversas opções, mas funciona bem com os parâmetros padrões.
APC	Implementa um sistema alternativo de cache. Neste modo é feito cache do arquivo php opcode.
XCache	Similar com APC. Requer que o usuário utilize password para funcionar.

Tipo	Descrição
Memcache	Cria um objeto de cache na memória do sistema.

Para utilizar o helper Cache, em qualquer view ou controller, é preciso antes atribuir o valor true à constante Configure::Cache.check em app/config/core.php. Se isso não for feito o cache não poderá ser criado ou lido.

Todo controller que precisar usar este helper precisa importá-lo:

```
<?php
        var $helpers = array('Cache');
?>
```

É preciso também indicar quais ações farão uso de cache. Isso é feito através da variável $cacheAction no controller. Essa variável deve ser um array contendo as ações e o respectivo tempo de armazenamento. Este tempo pode ser em segundo ou no formato strtotime(): "1 hour" ou "3 minutes".

```
<?php
var $cacheAction = array(
        'list/' => 99000,
        'view/23/' => 55000,
        'view/102'  => 76000);

?>
```

No exemplo acima, sempre que o cliente for na view de list será feito cache. Em view apenas se for a visualização dos registros com id 23 e 102.

Existem situações onde apenas partes da págna serão necesárias para fazer cache. Para indicar os blocos de conteúdo a **não** serem feitos cache utilize <cake:nocache> </cake:nocache>.

<cake:nocache> <?php if ($this->Session->check('Usuario.nome')) : ?> Welcome, <?php echo $this->Session->read('Usuario.nome')?>. <? else: ?> <? echo $html->link('Login', 'usuario/login')?><? endif; ?></cake:nocache>

Uma vez que foi feito cache de determinada ação, o método do controller para a ação não será invocado. Por isso, não é possível utilizar <cake:nocache> </cake:nocache> com variáveis, já que estas terão valor null no controller.

Para apagar o cache basta utilizar o método Cache::clear(). Este método irá apagar todas informações que estão em cache. Se um modelo utilizado for modificado, o CakePHP irá apagar o cache feito. Se for feito alguma ação do tipo INSERT, EDIT, DELETE ou UPDATE em determinado conteúdo, este conteúdo será apagado e feito novo cache.

3.6.3 - Form

Este helper ajuda na construção de formulários de forma a trazer facilidades tanto no aspecto de quantidade de código necessária para criá-los, quanto para facilitar o tratamento dos dados posteriormente.

3.6.3.1 - Criando formulários

create(string $model = null, array $options = array())

Este método permite que se crie a tag form no seu arquivo de visão. Todos os parâmetros são opcionais, caso sejam omitidos o CakePHP vai direcionar os dados para o controller relacionada à view utilizada, para as ações add ou edit. Por padrão o método de envio de dados adotado pelo CakePHP é o POST. O formulário é montado com um ID específico. Este id contém o nome do modelo e da ação do controller relacionados, seguidos da palavra Form e no formato CamelCased. Se utilizarmos esse método sem parâmetros na tela view relacionada ao ProdutosController, o resultado será:

```
<form id="UserAddForm" method="post" action="/users/add">
```

Utilizando os parâmetros é possível costumizar os formulários. Primeiro, pode-se alterar o modelo relacionado utilizando o primeiro parâmetro.

```
<?php echo $this->Form->create('Usuario'); ?>

// Irá retornar o seguinte formulário
<form id="UsuarioAddForm" method="post" action="/usuarios/add">
```

Dessa forma, o formulário irá procurar pela ação add no controller. O FormHelper pode realizar operações de edição. Para isso, basta o array $this->data estar populado com dados e possuir um valor do ID, então o formulário irá ser criado de forma a editar alguma registro existente no banco de dados.

```
<?php echo $this->Form->create('Usuario'); ?>

// Irá retornar o seguinte formulário
<form id="UsuarioAddForm" method="post" action="/usuarios/add">

// controllers/usuarios_controller.php:
<?php
function edit($id = null) {

        if (empty($this->data)) {
                $this->data = $this->Usuario->findById($id);
        } else {
                // Lógica para editar o registro existente.
        }
}
?>

// views/usuarios/edit.ctp:
// Como $this->data['Usuario']['id'] = 7, o formulário será do tipo edit
<?php  echo $this->Form->create('Usuario'); ?>

// Irá criar o seguinte formulário
<form id="UsuarioEditForm" method="post" action="/usuarios/edit/7">
<input type="hidden" name="_method" value="PUT" >
```

3.6.3.2 - $options['type']

Através desta opção pode-se definir o tipo de envio dos dados, que pode ser 'post', 'get', 'file', 'put' ou 'delete'.

```
<?php echo $this->Form->create('Usuario', array('type' => 'get')); ?>
// Irá retornar o seguinte formulário
<form id="UsuarioAddForm" method="get" action="/usuarios/add">

<?php echo $this->Form->create('Usuario', array('type' => 'file')); ?>
// Irá retornar o seguinte formulário
<form id="UsuarioAddForm" enctype="multipart/form-data"
method="post" action="/usuarios/add">
```

3.6.3.3 - $options['action']

Através dessa opção é possível selecionar uma ação específica do seu controller para onde os dados do

formulário serão submetidos.

```
<?php echo $this->Form->create('Usuario', array('action' => 'atualizar')); ?>
// Irá retornar o seguinte formulário
<form id="UsuarioAddForm" method="post" action="/usuarios/atualizar">
```

3.6.3.4 - $options['url']

Utilizado quando se quer utilizar uma ação de um controller que não o relacionado pelo arquivo de visão utilizado.

```
<?php echo $this->Form->create('Usuario', array('url' =>
'comunidade/adicionar')); ?>
// Irá retornar o seguinte formulário
<form id="UsuarioAddForm" method="post" action="/comunidade/adicionar">
```

3.6.3.5 - $options['default']

Se for atribuído o valor *false* para essa opção, o formulário não será enviado quando o botão de envio for pressionado.

3.6.3.6 - Fechando o formulário

Para fechar o formulário basta utilizar o método end().

```
<?php echo $this->Form->create(); ?>
// Itens do formulário
<?php echo $this->Form->end(); ?>
```

Se alguma string for fornecida como parâmetro, além de criar a tag para fechar o formulário, será criado um botão do tipo submit com o utilizando a string como nome.

```
<?php echo $this->Form->end('Enviar'); ?>

//Irá retornar o seguinte código

<div class="submit">

  <input type="submit" value="Enviar" />

</div>

</form>
```

3.6.3.7 - Campos de formulário

A forma mais prática de criar itens de um formulário é através do método `input()`. Este método vai analisar o tipo de dado fornecido pelo modelo e apresentar o tipo de item de formulário mais adequada para esta.

input(string $fieldName, array $options = array())

Tipo de dado	Campo de formulário resultante
string (char, varchar, etc)	text

Tipo de dado	Campo de formulário resultante
boolean, tinyint(1)	checkbox
texto	textarea
texto com o nome password, passwd ou psword	password
data	3 selects: dia, mes e ano
datetime, timestamp	7 selects: dia, mes, ano, hora, minuto, segundo e meridiano
time	3 selects: hora, minuto e meridiano

É possível customizar os campos. Veja o exemplo a seguir:

```
<?php

echo $this->Form->input('data_nasc', array( 'label' => 'Data de nascimento'
        , 'dateFormat' => 'DMY'
        , 'minYear' => date('Y') - 70
        , 'maxYear' => date('Y') - 18 ));

?>
```

Ao utilizar o nome do campo da tabela para criar o item de formulário com o auxílio do FormHelper, este irá colocar o nome e o id do formulário da seguinte forma:

```
<input type="text" id="ModelnameFieldname" name="data[Modelname][fieldname]" />
```

É possível especificar o nome do modelo a ser utilizado manualmente.

```
<?php
        echo $this->Form->input('Modelname.fieldname');
?>
```

Se for necessário criar diversos itens de formulário com o mesmo nome para utilizar o método saveAll depois. Pode-se utilizar a seguinte convenção:

```
<?php
        echo $this->Form->input('Modelname.0.fieldname');
        echo $this->Form->input('Modelname.1.fieldname');
?>

// Irá gerar os seguintes itens de formulário

<input type="text" id="Modelname0Fieldname" name="data[Modelname][0][fieldname]">
<input type="text" id="Modelname1Fieldname" name="data[Modelname][1][fieldname]">
```

A seguir temos uma tabela com diversas outras formas de criar itens de formulário:

Item de formulário	Sintaxe
checkbox	checkbox(string $fieldName, array $options)
year	year(string $fieldName, int $minYear, int $maxYear, mixed $selected, array $attributes, boolean $showEmpty)
month	month(string $fieldName, mixed $selected, array $attributes, boolean $showEmpty)
dateTime	dateTime(string $fieldName, string $dateFormat = 'DMY', $timeFormat = '12', mixed $selected, array $attributes, boolean $showEmpty)
day	day(string $fieldName, mixed $selected, array $attributes, boolean $showEmpty)
hour	hour(string $fieldName, boolean $format24Hours, mixed $selected, array $attributes, boolean $showEmpty)
minute	minute(string $fieldName, mixed $selected, array $attributes, boolean $showEmpty)
meridian	meridian(string $fieldName, mixed $selected, array $attributes, boolean $showEmpty)
error	error(string $fieldName, string $text, array $options)
file	file(string $fieldName, array $options)
hidden	hidden(string $fieldName, array $options)
isFieldError	isFieldError(string $fieldName)
label	label(string $fieldName, string $text, array $attributes)
password	password(string $fieldName, array $options)
radio	radio(string $fieldName, array $options, array $attributes)
select	select(string $fieldName, array $options, mixed $selected, array $attributes, boolean $showEmpty)

Item de formulário	Sintaxe
submit	submit(string $caption, array $options)
submitImage	submitImage(string $path, array $options)
text	text(string $fieldName, array $options)
textarea	textarea(string $fieldName, array $options)

3.6.3.7.1 - Input do tipo file

Além das possibilidades mostradas anteriormente ainda é possível criar o item de formulário do tipo file. Para isso, é preciso antes criar o formulário da seguinte forma:

```
<?php
  $this->Form->create('Documento', array('enctype' => 'multipart/form-data') );

  // Ou

  $this->Form->create('Documento', array('type' => 'file'));
?>
```

Logo em seguida, crie o item de formulário:

```
<?php
        $this->Form->input('Documento.submittedfile', array('type'=>'file'));

        // Ou

        $this->Form->file('Documento.submittedfile');
?>
```

Quando o formulário é enviado, é criado um array com diversas informações sobre o arquivo. Veja o exemplo a seguir:

```
<?php
        $this->data['Documento']['submittedfile'] = array(
                'name' => apostilaCakePHP.pdf
                'type' => application/pdf
                'tmp_name' => C:/Projetos/tmp/php33.tmp
                'error' => 0
                'size' => 56543
        );
?>
```

3.6.4 - HTML

Este helper ajuda a produzir código html de forma prática, rápida e fácil de ser modificada. Antes de entrarmos nos métodos dessa classe, é importante ver algumas configurações. No arquivo core, localizado em app/config/core.php, existe uma constante AUTO_OUTPUT. Se essa constante estiver com o valor true, o HtmlHelper irá produzir o conteúdo das tags, ao invés de retornar o seu valor.

Funções que utilizam $return permitem sobrescrever as configurações contidas no arquivo core. Atribua o valor true para $return para utilizar o HtmlHelper corretamente

Muitos métodos do HtmlHelper possuem o parâmetro $htmlAttributes. Este parâmetro permite gerar atributos adicionais às tags. Veja a seguir um exemplo:

```
Atributo desejado: <tag class="transparente" />
Array do parâmetro: array('class'=>'transparente')
```

Por padrão o HtmlHelper está disponível nas views. Caso aconteça de ter um erro, reportando a sua ausência, verifique no arquivo do controller se está faltando sua declaração na declaração do array $helpers.

3.6.4.1 - Utilizando o HtmlHelper

Nesta sessão iremos ver as diferentes formas que este helper fornece para gerar código html, de forma a diminuir o volume de código necessário para criar as telas e facilitar sua manutenção.

3.6.4.1.1 - charset

charset(string $charset=null)

Atrvés desse método declaramos o charset que o documento utilizará. Por padrão é utilizado UTF-8. Veja o exemplo a seguir:

```
<?php echo $html->charset('ISO-8859-1');  ?>

//Retorna
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
```

3.6.4.1.2 - css

css(mixed $path, string $rel = null, array $htmlAttributes = array(), boolean $inline = true)

Cria links para arquivos de estilo CSS. Se o valor de inline for false, o arquivo é importado no $scripts_for_layout que pode ser escrito dentro da tag head do documento.

```
<?php echo $html->css(array('forms','tables'));  ?>

//Retorna
<link rel="stylesheet" type="text/css" href="/test/css/forms.css" />
<link rel="stylesheet" type="text/css" href="/test/css/tables.css" />
```

3.6.4.1.3 - meta

meta(string $type, string $url = null, array $attributes = array(), boolean $inline = true)

Este método é útil para para linkar recursos externos como favicon ou feeds RSS. O parâmetro inline funciona da mesma forma do método `css`. Para definir o tipo de metatag a ser gerada, utilize o parâmetro type. Na tabela seguinte, veja as opções de type e o que elas renderizam na tela.

Type	Resultado na tela
html	text/html
rss	application/rss+xml

Type	Resultado na tela
atom	application/atom+xml
icon	image/x-icon

```
<?php
echo $html->meta(
        'Últimas notícias',
        '/noticias/index.rss',
        array('type' => 'rss')
);
?>
//Retorna (obs: para efeito ilustrativo foi adiconado quebra-de-linha)
<link
        href="http://exemplo.com/noticias/index.rss"
        title="Últimas notícias"
        type="application/rss+xml"
        rel="alternate"
/>
```

Listagem 49 - Exemplo de uso do método `meta()`

3.6.4.1.4 - docType

docType(string $type = 'xhtml-strict')

Utilizado para declarar o tipo de documento. Veja a tabela a seguir com os tipos possíveis.

Type	Resultado na tela
html	text/html
html4-strict	HTML4 Strict
html4-trans	HTML4 Transitional
html4-frame	HTML4 Frameset
xhtml-strict	XHTML1 Strict
xhtml-trans	XHTML1 Transitional
xhtml-frame	XHTML1 Frameset
xhtml11	XHTML 1.1

3.6.4.1.5 - style

style(array $data, boolean $inline = true)

Este método permite gerar estilo CSS utilizando os índices e valores do array $data para gerar a saída.

```
<?php
echo $html->style(
        array(
                'position' => 'relative',
                'top' => '10px',
                'left' => '10px'
        )
);
?>

//Retorna (obs: para efeito ilustrativo foi adiconado quebra-de-linha)

position:relative;
top:10px;
left:10px;
```

3.6.4.1.6 - image

image(string $path, array $htmlAttributes = array())

Cria a tag de imagem. As imagens serão procuradas dentro do diretório /app/webroot/img/.

```
<?php echo $html->image('cake_logo.png', array('alt' => 'CakePHP'));  ?>

//Retorna
<img src="/img/cake_logo.png" alt="CakePHP" />
```

3.6.4.1.7 - link

link(string $title, mixed $url = null, array $htmlAttributes = array(), string $confirmMessage = false, boolean $escapeTitle = true)

Método utilizado para criar links.

```
<?php
echo $html->link( 'Apagar usuário',
        array('controller'=>'usuario', 'action'=>'delete', 121),
        array(),
        "Você tem certeza que deseja apagar este usuário?"
);
?>

//Retorna
<a href="/usuario/delete/121"
onclick="return confirm('Você tem certeza que deseja apagar este usuário?');">
Apagar usuário</a>
```

3.6.4.1.8 - tag

tag(string $tag, string $text, array $htmlAttributes, boolean $escape = false)

Retorna determinado texto definido no parâmetro $text dentro da tag definida no parâmetro $tag. Se não é fornecido valor no parâmetro $tag é retornado tag.

```
<?php echo $html->tag('span', 'Algum texto.', array('class' => 'texto1')); ?>

//Retorna
<span class="texto1" >Algum texto. </span>
```

3.6.4.1.9 - div

div(string $class, string $text, array $htmlAttributes, boolean $escape = false)

Usado para criar Div's. No primeiro parâmetro define-se uma classe CSS, no segundo o conteúdo da div e se o parâmetro $escape estiver como true, o conteúdo de $text é impresso transformando as tags em textos.

```
<?php echo $html->div('erro', 'Login incorreto.'); ?>

//Retorna
<div class="erro" >Login incorreto. </div>
```

3.6.4.1.10 - para

para(string $class, string $text, array $htmlAttributes, boolean $escape = false)

Este método é utilizado para criar parágrafos.

```
<?php echo $html->para(null, 'Algum texto.'); ?>

//Retorna
<p>Algum texto. </p>
```

3.6.4.1.11 - tableHeaders

tableHeaders(array $names, array $trOptions = null, array $thOptions = null)

3.6.4.1.12 - tableCells

tableCells(array $data, array $oddTrOptions = null, array $evenTrOptions = null)

Cria linhas no cabeçalho da tabela.

```
<?php echo $html->tableHeaders(array('Data de nascimento','Nome','Telefone')); ?>

//Retorna
<tr><th>Data de nascimento</th><th>Nome</th><th>Telefone</th></tr>
```

3.6.5 - Javascript

Este helper foi desenvolvido para ajudar a inserir códigos javascript na aplicação. Muitos métodos necessitam da biblioteca Prototype.

3.6.5.1 - Métodos da classe Javascript

3.6.5.1.1 - codeBlock

codeBlock($string, $options, $safe)

Este método retorna o código javascript contido no parâmetro $string. Este código vai ser posto entre as tags script. O parâmetro $options pode receber os valores allowCache, safe ou inline. O último parâmetro, $safe, não deve ser utilizado. No lugar dele, deve-se usar $options['inline'].

3.6.5.1.2 - blockEnd

blockEnd()

Método usado para terminar um bloco de código javascript.

3.6.5.1.3 - link

link($url, $inline)

Método utilizado para incorporar arquivos javascript ao documento. No parâmetro deve-se colocar a url do arquivo javascript, sendo que pode ser usado um array para importar mais de um arquivo. Se for atribuído o valor true para o parâmetro $inline, será inserida a tag script no documento, senão o arquivo será incorporar em $scripts_for_layout. Por padrõa recebe o valor true.

3.6.5.1.4 - escapeString

escapeString($string)

Transforma a string passada pelo parâmetro $string em um formato que não gere conflito com a linguagem javascript. A lista a seguir contém as formas como os caracteres são alterados.

- "\r\n" => '\n'
- "\r" => '\n'
- "\n" => '\n'
- "'" => '\'"
- "'" => "\\'"

3.6.5.1.5 - event

event($object, $event, $observer, $useCapture)

Insere um evento, definido pelo parâmetro $event, a um objeto DOM, definido por $object. O objeto pode ser um Id, um seletor CSS ou um objeto javascript. Este método requer a biblioteca Prototype. O parâmetro $observer deve conter o método a ser executado quando o evento ('click', 'over', 'out', etc definido em $event) ocorrer.

3.6.5.1.6 - cacheEvents

cacheEvents($file, $all)

Ajuda a controlar como o JavascriptHelper lida com o cache feito com o método event. Se o parâmetro $all estiver com o valor true, todo o código gerado pelo helper vai ficar em cache e poderá ser recuperado com o método getCache ou escrito num arquivo ou página através do método writeCache. Se o parâmetro $file for true o código será escrito em um arquivo.

3.6.5.1.7 - getCache

getCache($clear)

Pega ou apaga o evento javascript atual que estiver em cache. Se o parâmetro $clear for true, então será apagado. Por padrão $clear é true.

3.6.5.1.8 - writeEvents

writeEvents($inline)

Retorna códigos javascript que estão em cache. Se o parâmetro $inline estiver como true o código é retornado no próprio documento, senão é adicionado ao $scripts_for_layout.

3.6.5.1.9 - includeScript

includeScript($script)

Inclui o script javascript definido pelo parâmetro $script. Se este parâmetro estiver em branco será incluído todos scripts dentro de webroot/js.

3.6.5.1.10 - object

object($data, $options)

Cria um objeto javascript. O parâmetro $data contém os dados a serem convertidos. No parâmetro $options pode ser utilizadas as seguintes opções:

- boolean $options['block'] - o código é retornado dentro do bloco script se tiver o valor como true. Por padrão recebe o valor false.
- string $options['prefix'] - String que irá ser colocada antes dos dados retornados.
- string $options['postfix'] - o mesmo que prefix, mas coloca a string após.
- array $options['stringKeys'] - lista de índices de array a serem tratadas como string.
- boolean $options['quoteKeys'] - Se for false trata $stringKey como uma lista de índices a não serem postos entre aspas.
- string $options['q'] - o tipo de aspas a ser usado. O padrão é "

3.6.6 - Number

Este helper ajuda a converter e apresentar dados numéricos em diferentes formatos. Os métodos incluem conversão de moedas, porcentagem, precisão numérica, etc. Todos os métodos retornam o valor numérico através de echo. Ou seja, ao chamar um método o resultado será um valor impresso na tela.

3.6.6.1 - Métodos do helper Number

3.6.6.1.1 - currency

currency(mixed $number, string $currency= 'USD')

Este método é utilizado para mostrar valores em diferentes moedas (dólar, euro, etc).

```
<?php echo $number->currency($number,$currency); ?>
```

Listagem 50 - Exemplo de uso do método currency()

O parâmetro $number deve ter um número do tipo float (decimal). O parâmetro $currency deve conter o tipo de moeda. Este parâmetro pode conter os valores: USD, EUR ou GBP.

Moeda	Formato
USD	1,236.33
GBP	1,236.33

Moeda	Formato
EUR	1.236,33

3.6.6.1.2 - precision

precision (mixed $number, int $precision = 3)

Este método mostra um número, definido pelo parâmetro $number, com a precisão determinada pelo parâmetro $precision. O valor de $precision é igual ao número de casas decimais a serem mostradas.

```
<?php echo $number->precision(12.86545689111, 3); ?>

//Retorna
12.865
```

Listagem 51 - Exemplo de uso do método `precision()`

3.6.6.1.3 - toPercentage

toPercentage(mixed $number, int $precision = 2)

Método semelhante ao precision. Deve-se fornecer um número no parâmetro $number e o grau de precisão pelo parâmetro $precision. O valor é retornado em porcentagem.

```
<?php echo $number->toPercentage(12.86545689111); ?>

//Retorna
12.86%
```

Listagem 52 - Exemplo de uso do método `toPercentage()`

3.6.6.1.4 - toReadableSize

toReadableSize(string $data_size)

Este método transforma tamanho de arquivos em formatos mais compreensíveis. Os valores são mostrados com precisão de duas casas decimais e transformados para Bytes, KB, MB, GB ou TB de acordo com a conveniência.

```
echo $number->toReadableSize(0);  // 0 Bytes
echo $number->toReadableSize(1024); // 1 KB
echo $number->toReadableSize(1321111.11); // 1.26 MB
echo $number->toReadableSize(4159215222); // 5.00 GB
```

Listagem 53 - Exemplo de uso do método `toReadableSize()`

3.6.6.1.5 - format

format (mixed $number, mixed $options=false)

Este método fornece maior controle sobre o formato dos números. No parâmetro $number deve ser fornecido o número e no parâmetro $options as opções de formatação. Se nenhum parâmetro é passado em $options a formatação padrão será sem casas decimais. Em opções podem ser usadas as seguintes opções:

- places (integer): precisão decimal do resultado
- before (string): string a ser adicionada antes do número retornado

- escape (boolean): se for true a string em before vai ser escapada
- decimals (string): usado para definir o caractere que separa as casas decimais
- thousands (string): usado para definir o caractere que separa milhares, milhões.

```
echo $number->format('556677.8899', array(
                'places' => 2,
                'before' => '$ ',
                'escape' => false,
                'decimals' => ',',
                'thousands' => '.'
        )
);

// Retorna
'$ 556.677,88'
```

Listagem 54 - Exemplo de uso de format

3.6.7 - Paginator

Este helper ajuda a gerar paginação dos resultados. Através dele é possível obter o número de páginas e links para página anterior e próxima.

3.6.7 - Métodos do helper Paginator

3.6.7.1 - options

options($options = array())

Define todas as opções para a paginação. O array $options recebe como índice format que pode conter os seguintes valores 'range' and 'pages' e 'custom'. Sendo 'custom' o padrão. Se o valor for custom é possível utilizar diversas strings que serão substituídas por valores. Essas strings são as seguintes:

- %page% - o número da página atual.
- %pages% - total de páginas.
- %current% - número de resultados atuais sendo mostrados.
- %count% - número total de resultados.
- %start% - número do primeiro resultado sendo mostrado.
- %end% - número do último resultado sendo mostrado.

Utilizando o método counter() é possível utilizar essas opções. Veja o exemplo a seguir:

```
echo $paginator->counter(
        array(
                'format' => 'Página %page% de %pages%,
                        mostrando %current% resultados de %count% total,
                        começando no resultado %start%, terminando em %end%'
        )
);
```

separator

A string utilizada para separar a página atual do número das páginas. O padrão é 'of'. Geralmente é utlizado junto com format = 'pages'.

url

A url da ação da paginação. Url possui as seguintes opções:

- sort - o campo que servirá como referência para a ordenação
- direction - A direção da ordenação. O padrão é ascendente - 'ASC'

- page – O número de páginas a mostrar

model

O nome do modelo usado para gerar a paginação

escape

Define se o nome dos campos deve ser escapado. O padrão é true.

update

O id do objeto a ser atualizado quando se está usando AJAX. Links serão criados se este item não for definido.

indicator

Id do objeto que irá mostrar o status (lendo, ou trablhando) da requisição AJAX.

3.6.7 – link

link($title, $url = array(), $options = array())

Cria um link normal ou do tipo AJAX com parâmetros de paginação.

- string $title – Título do link.
- mixed $url – Url para a ação.
- array $options – Opções para o link.

Exemplo de uso de link

```
echo $paginator->link('Ordenação por título na página 5', array(
                'sort' => 'titulo',
                'page' => 5,
                'direction' => 'desc'
        )
);
```

3.6.8 – RSS

Este helper possibilita a criação de arquivo com feed RSS de forma prática.

3.6.8.1 – Criando um feed

Nesse exemplo, partimos do pressuposto que já se possui um model e um controller de posts. É preciso ir em app/config/routes.php e alterar o parseExtensions da seguinte forma:

```
Router::parseExtensions('rss');
```

Agora inserimos o RequestHandler no controller no array $components.

3.6.8.1.1 – Código no controller

Para fazer uma versão rss de posts/index devemos alterar o código do PostsController da seguinte forma:

```
// O código é uma alteração para gerar saída em RSS
public function index(){
        if( $this->RequestHandler->isRss() ){
                // Neste caso a requisição é por uma página RSS
```

```
                $posts = $this->Post->find('all', array(
                        'limit' => 20,
                        'order' => 'Post.created DESC'));
                $this->set(compact('posts'));
        } else {
                // Aqui não é uma requisição RSS então deve ser fornecido dados
                // para gerar uma interface pertinente ao website's
                $this->paginate['Post'] = array(
                        'order' = 'Post.created DESC',
                        'limit' => 10);
                $posts = $this->paginate();
                $this->set(compact('posts'));

        }
}
```

Para criar o layout, basta salvar o código abaixo no diretório app/views/layouts/rss/default.ctp

```
echo $rss->header();
if (!isset($documentData)) {
        $documentData = array();
}
if (!isset($channelData)) {
        $channelData = array();
}
if (!isset($channelData['title'])) {
        $channelData['title'] = $title_for_layout;
}
$channel = $rss->channel(array(), $channelData, $content_for_layout);
echo $rss->document($documentData,$channel);
```

As variáveis $documentData e $channelData não foram definidas no controller. Como é possível passar variáveis entre a view e o layout, estas variáveis serão definidas na view.

O arquivo da view deverá ser criado no seguinte caminho: app/views/posts/rss/index.ctp. No início do código definimos as variáveis $documentData e $channelData que deverão conter metadados para a construção do feed.

```
$this->set('documentData', array(
        'xmlns:dc' => 'http://teste.com.br/'));
$this->set('channelData', array(
        'title' => __("Últimas notícias", true),
        'link' => $html->url('/', true),
        'description' => __("Contém as últimas notícias.", true),
        'language' => 'pt-br'));
```

Agora definimos todo o corpo do feed. Através de um loop pelo array com a listagem das notícias vamos

```
foreach ($entries as $entry) {
        $postTime = strtotime($entry['Entry']['created']);

        $entryLink = array(
                'controller' => 'entries',
                'action' => 'view',
                'year' => date('Y', $postTime),
```

```
                'month' => date('m', $postTime),
                'day' => date('d', $postTime),
                $entry['Entry']['slug']
        );

        App::import('Sanitize');

        // O texto e descrição do feed deve conter apenas texto para serem
válidos.
        $bodyText = preg_replace('=\(.*?)\=is', '', $entry['Entry']['body']);
        $bodyText = $text->stripLinks($bodyText);
        $bodyText = Sanitize::stripAll($bodyText);
        $bodyText = $text->truncate($bodyText, 400, '...', true, true);

        echo $rss->item(array(), array(
                        'title' => $entry['Entry']['title'],
                        'link' => $entryLink,
                        'guid' => array('url' => $entryLink, 'isPermaLink' =>
'true'),
                        'description' => $bodyText,
                        'dc:creator' => $entry['Entry']['author'],
                        'pubDate' => $entry['Entry']['created']
                )
        );
}
```

3.6.9 - Session

O helper Session é muito parecido com o componente Session. A grande diferença é que não é possível escrever uma sessão através do helper. Por padrão este helper já está disponível em suas views.

Método	Descrição
read($key)	Este método permite ler alguma variável de Sessão. Retorna uma string ou array, dependendo do conteudo da sessão.
id()	Retorna o id da sessão atual.
check($key)	Utilizado para checar se a chave, definida no parâmetro $key, existe na sessão. Retorna um valor booleano, 0 se não existir, 1 se existir.
flash($key)	Irá imprimir o conteúdo de $_SESSION.Message. Geralmente utilizado junto com o método setFlash() do componente Session.
error()	Retorna o último erro da sessão, se existir.

Tabela 16 - Métodos do helper Session

3.6.10 - Text

Este helper possibilita que os textos fiquem mais práticos de serem usados nas views. Dentre outras facilidades, podemos citar a adição de links, criar destaques em palavras ou frases e formatar urls.

3.6.10.1 - Métodos da classe Text

3.6.10.1.1 - autoLinkEmails

autoLinkEmails (string $text, array $htmlOptions=array())

Este método adiciona links de e-mails a todos endereços de e-mail contidos dentro do texto passado pelo parâmetro $text. No parâmetro $htmlOptions é possível passar opções iguais as utilizadas no método link() do helper Html.

```
$my_text = 'Se você gostou do nosso curso, entre em contato com contato@2km.com';
$linked_text = $text->autoLinkEmails($my_text);

//$linked_text:
Se você gostou do nosso curso, entre em contato com
<a href="mailto:contato@2km.com"><u>contato@2km.com</u></a>
```

3.6.10.1.2 - autoLinkUrls

autoLinkUrls (string $text, array $htmlOptions=array())

O mesmo que autoLinkEmails, a diferença é que este método irá procurar por strings iniciadas por https, http, ftp, ou nntp no texto fornecido pelo parâmetro $text e montar links para estes itens.

3.6.10.1.3 - autoLink

autoLink (string $text, array $htmlOptions=array())

Este método faz o que autoLinkUrls e autoLinkEmails fazem ao mesmo tempo.

3.6.10.1.4 - excerpt

excerpt (string $haystack, string $needle, int $radius=100, string $ending="...")

Método útil na construção de resultados de buscas. Extrai um trecho do text fornecido pelo parâmetro $haystack. No parâmetro $radius é definido o número de caracteres a ser retornado. Em $needle define-se a string de pesquisa e em $ending define-se a string que deve existir no fim do texto retonado.

```
echo $text->excerpt($paragrafo, 'método', 50); ?>
// Retorna
Devem haver cinquenta caracteres antes da palavra método, depois dela não
importa.
```

3.6.10.1.5 - highlight

highlight (string $haystack, string $needle, $highlighter= '< span class="highlight">\1')

Pesquisa a string $needle no texto passado no parâmetro $haystack e dá o destaque a essa string através do que for definido no parâmetro $highlighter.

```
echo $text->highlight($artigo, 'usando');
// Retorna
O texto está <span class="highlight">usando</span> o recurso de highlight.
```

3.6.10.1.6 - stripLinks

stripLinks ($text)

Extrai os textos dos links existentes dentro do texto fornecido pelo parâmetro $text.

3.6.10.1.7 - toList

toList (array $list, $and= 'and')

Organiza os itens do array fornecido pelo parâmetro $list separando-os por vírgulas. Entre o penúltimo e último item é possível definir o separador pelo parâmetro $and.

```
echo $text->toList($animais, 'e'); ?>
// Retorna: macaco, leão, onça, cachorro e gato
```

3.6.10.1.8 - truncate

truncate(string $text, int $length=100, array $options)

Corta o texto, definido pelo parâmetro $text, até o tamanho definido pelo parâmetro $length. Se o texto é maior que o tamanho de $lenght é posto no fim do texto retornado a string definida pelo parâmetro $ending.

```
echo $text->truncate(
        'Falta uma semana para a viagem.',
        9,
        array(
                'ending'=>'...',
                'exact'=>false
        )
);

// Retorna
Falta uma...
```

O método trim() pode ser usado como um alias para truncate.

3.6.11 - Time

Este helper ajuda a formatar as horas e testar as horas.

3.6.11.1 - Métodos de formatação de tempo

3.6.11.1.1 - fromString

fromString($date_string)

Pega uma string e utiliza a função strtotime para convertê-la em um objeto de data. Se a string fornecida for um número, ela primeiro é convertida em inteiro, armazenando o valor em segundos.

3.6.11.1.2 - toQuarter

toQuarter($date_string, $range = false)

Irá retornar 1,2,3 ou 4 dependendo em qual das épocas do ano a data passada pelo parâmetro $date estiver. Se o parâmetro $range estiver com o valor true, será retornado um array com as datas de início e fim.

3.6.11.1.3 - toUnix

toUnix($date_string)

3.6.11.1.4 - toAtom

toAtom($date_string)

Retorna uma data (string) no formato Atom: "2008-01-12T00:00:00Z".

3.6.11.1.5 - toRSS

toRSS($date_string)

Retorna uma data (string) no formato de RSS: "Sat, 12 Jan 2008 00:00:00 -0500"

3.6.11.1.6 - nice

nice($date_string = null)

Pega uma string de data e converte para o formato "Tue, Jan 1st 2008, 19:25".

3.6.11.1.7 - niceShort

niceShort($date_string = null)

Faz o mesmo que o método nice. Se a data for do dia atual, o resultado será algo do tipo: "Today, 19:25". Se o dia fornecido for ontem, o resultado será: "Yesterday, 19:25".

2.5.11.1.8 - daysAsSql

daysAsSql($begin, $end, $field_name)

Retorna uma data no seguinte formato: "($field_name >= '2008-01-21 00:00:00') AND ($field_name <= '2008-01-25 23:59:59')". Útil caso seja necessário buscar por resultados entre duas datas.

2.5.11.1.9 - daysAsSql

ayAsSql($date_string, $field_name)

Igual o método daysAsSql, só que precisa ser fornecido apenas um objeto de data.

3.6.11.1.10 - timeAgoInWords

timeAgoInWords($date_string, $options = array(), $backwards = null)

Pega uma string de data e converte para um formato mais amigável, do tipo: "3 weeks, 3 days ago". Se o parâmetro $backwards for true a data será tratada como sendo do futuro, o que irá trazer o resultado no formato: "on 31/12/08".

3.6.11.1.11 - relativeTime

relativeTime($date_string, $format = 'j/n/y')

Alias para o método timeAgoInWords.

Parâmetro $format	Parâmetro $end
Formato que a data será apresentada. O padrão é "on 31/12/08"	Define o ponto onde não é mais usado palavras e sim um formato de data. O padrão é "+1 month"

3.6.11.1.12 - gmt

gmt($date_string = null)

Retorna uma data no horário de Greenwich.

3.6.11.1.13 - format

format($format = 'd-m-Y', $date_string)

3.6.11.1.14 - Testes de tempo

- isToday
- isThisWeek
- isThisMonth
- isThisYear
- wasYesterday
- isTomorrow
- wasWithinLast

Todas fuções acima retornam um valor true ou false de acordo com a data fornecida pelo parâmetro $date. No método wasWithinLast, existe um parâmetro a mais:

wasWithinLast($time_interval, $date_string)

Este parâmetro deve conter uma string no format "3 months" e aceita um intervalo de tempo em segundos, minutos, horas, dias, semanas, meses e anos. O valor padrão é dias, então se o valor passado para este parâmetro não for reconhecido será utilizado dias.

3.6.12 - XML

Este helper facilita a criação de documentos XML.

3.6.12.1 - Métodos do helper XML

3.6.12.1.1 - serialize

Este método transforma um array em uma saída XML.

```
echo $xml->serialize($data);

// irá retornar os dados seguindo este modelo:
<nomeDoModelo id="1" nomeDoCampo="content" />
```

3.6.12.1.2 - elem

string elem (string $name, $attrib = array(), mixed $content = null, $endTag = true)

Este método permite construir um nó XML com atributos e conteúdo.

```
echo $xml->elem('cont', array('namespace' => 'cronometro'), '10:10');
// retorna: <cronometro:cont>10:10</count>
```

Se for necessário utilizar CDATA entre o conteúdo, então o terceiro parâmetro deve ser um array com os índices cdata e value. Veja o exemplo a seguir:

```
echo $xml->elem('contador', null, array('cdata'=>true,'value'=>'conteudo');
// retorna: <contador><![CDATA[conteudo]]></contador>
```

3.6.12.1.3 - header

Este método é utilizado para gerar a declaração do documento XML.

```
echo $xml->header();
// retorna: <?xml version="1.0" encoding="UTF-8" ?>
```

É possível passar diferentes números de versão e tipos de codificação. Veja o exemplo a seguir:

```
echo $xml->header(array('version'=>'1.1'));
// retorna: <?xml version="1.1" encoding="UTF-8" ?>
```

3.7 - Plugins

Este recurso é utilizado para reutilizar toda uma aplicação, a idéia basica deste método é empacotar sua aplicação de forma que seja possível utilizá-la em outras aplicações. Por exemplo: um sistema de busca de ceps que pode ser utilizado em diversas aplicações.

3.7.1 - Criando um Plugin

Será mostrado os passos para criar um plugin para um sistema de busca de ceps. Segue abaixo a hierarquia das pastas:

```
/app
     /plugins
         /cep
             /controllers                <- os controles do plugin ficam aqui
             /models                     <- os modelos do plugin ficam aqui
             /views                      <- as visões do plugin ficam aqui
             /cep_app_controller.php   <- AppController
             /cep_app_model.php        <- AppModel
```

O CepAppController deve ficar assim:

```
// /app/plugins/cep/cep_app_controller.php:

class CepAppController extends AppController {
     //... Suas regras...
}
```

O CepAppModel deve ficar assim:

```
// /app/plugins/cep/cep_app_model.php:

class CepAppModel extends AppModel {
     //... Suas regras...
}
```

3.7.2 - Camada de Controle

Os arquivos da camada de controle devem ser salvos na pasta: **/app/plugins/cep/controllers/**.

Sua estrutura básica será:

```
class CepRuasController extends CepAppController {
        var $name = 'CepRuas';
        var $uses = array('Cep.CepRua');
        function index() {
```

```
                //...
        }
}
```

3.7.3 - Camada de Modelo

Os arquivos da camada de modelo devem ser salvos na pasta: **/app/plugins/cep/models/**.

Sua estrutura básica será:

```
class CepRua extends CepAppModel {
        var $name = 'CepRua';
}
```

3.7.4 - Camada de Visão

Os arquivos da camada de visão devem ser salvos na pasta: **/app/plugins/cep/views/**.

Sua estrutura básica será:

```
// /app/plugins/cep/views/cep_ruas/index.ctp:
Listagem de Ruas
Rua: teste 1
```

3.7.5 - Components, Helpers e Behaviors

A utilização de Components, Helpers e Behaviors no plugin é exatamente igual a aplicação.

```
// Component
class ExampleComponent extends Object {

}

// no controller de seu plugin:
var $components = array('Examplo');
```

Caso seja necessário a aplicação utilizar um componente do plugin, isto deverá ser feito da seguinte maneira:

```
//Exemplo de uso na aplicação
var $components = array('NomeDoPlugin.Exemplo');
var $components = array('Cep.Exemplo');
```

3.7.6 - Imagens, CSS, Javascript

Para inserir imagens, CSS e Javascript em seu plugin:

```
echo $html->image('/cep/img/minha_imagem.png');
echo $html->css('/cep/css/meu_css');
echo $javascript->link('/cep/js/biblioteca');
```

3.8 - Core Utility Libraries

3.8.1 - Inflector

3.8.1.1 - Métodos da Classe

	Entrada	Saída
pluralize	Apple, Orange, Person, Man	Apples, Oranges, People. Men
singularize	Apples, Oranges, People. Men	Apple, Orange, Person, Man
camelize	Apple_pie, some_thing, people_person	ApplePie, SomeThing, PeoplePerson
underscore	applePie, someThing	apple_pie, some_thing
humanize	apple_pie, some_thing, people_person	Apple Pie, Some Thing, People Person
tableize	Apple, UserProfileSetting, Person	apples, user_profile_settings, people
classify	apples, user_profile_settings, people	Apple, UserProfileSetting, Person
variable	apples, user_result, people_people	apples, userResult, peoplePeople
slug	apple purée	apple_puree

3.8.2 - Strings

3.8.2.1 - uuid

Este método é usado para gerar identificadores únicos, baseado no RFC 4122. O uuid é uma string de 128 bits com o formato: 485fc381-e790-47a3-9794-1337c0a8fe68

```
String::uuid(); // 485fc381-e790-47a3-9794-1337c0a8fe68
```

3.8.2.2 - tokenize

Cria o token de uma string usando o separador definido, ignorando qualquer seprador encontrado entre o $leftBound e $rightBound.

```
string tokenize ($data, $separator = ',', $leftBound = '(', $rightBound = ')')
```

3.8.2.3 - insert

O método é usado para trocar chaves em uma string para um valor.

```
String::insert('Meu nome é :nome e eu tenho :idade anos.', array('nome' =>
'José', 'age' => '35'));
// Retorna: "Meu nome é José e eu tenho 35 anos."
```

3.8.2.4 - cleanInsert

Este método é usado para limpar espaços e marcações não necessárias de uma string.

3.8.3 - Xml

Esta biblioteca é utilizada para fazer a transformação de um arquivo XML.

3.8.3.1 - Parse Xml

Baseado no exemplo abaixo:

```
$input = '<' . '?xml version="1.0" encoding="UTF-8" ?' . '>
    <funcionario>
        <elemento id="primeiro">
            <nome>João</nome>
```

```
            <cargo>Assitente</cargo>
        </elemento>
        <elemento>
            <nome>José</nome>
            <cargo>Gerente</cargo>
        </elemento>
    </funcionario>';
$xml = new Xml($input)
```

Após a transformação o objeto estará assim:

```
echo $xml->children[0]->children[0]->name;
//saída: 'elemento'

echo $xml->children[0]->children[0]->children[0]->children[0]->value;
//saída: 'João'

echo $xml->children[0]->child('elemento')->attributes['id'];
//saída: 'primeiro'
```

3.8.4 - Set

Esta biblioteca deve ser utilizada para gerenciamento de arrays.

3.8.4.1 - insert

Insere dados em um array, baseado na $path.

array Set::insert ($list, $path, $data = null)

```
$a = array('paginas' => array('nome' => 'pagina'));
$resultado = Set::insert($a, 'arquivos', array('nome' => 'teste'));
/* $resultado:
Array
(
        [paginas] => Array
        (
                [nome] => pagina
        )
        [arquivos] => Array
        (
                [nome] => teste
        )
)
*/

$a = array('paginas' => array('nome' => 'pagina'));
$result = Set::insert($a, 'paginas.nome', array());
/* $resultado:
Array
(
        [paginas] => Array
        (
                [nome] => Array
                (
                )
```

```
        )
)
*/

$a = array(
        'paginas' => array(
                0 => array('nome' => 'principal'),
                1 => array('nome' => 'ajuda')
)
);
$resultado = Set::insert($a, 'paginas.1.vars', array('titulo' => 'Titulo:
Ajuda'));
/* $resultado:
Array
(
        [paginas] => Array
        (
                [0] => Array
                (
                        [nome] => principal
                )
                [1] => Array
                (
                        [nome] => ajuda
                        [vars] => Array
                        (
                                [titulo] => Titulo:Ajuda
                        )
                )
        )
)
*/
```

3.8.4.2 - sort

Ordena um array pelo valor, baseado no path.

array Set::sort ($data, $path, $dir)

```
$a = array(
        0 => array('Pessoa' => array('nome' => 'Cadu')),
        1 => array('Camisa' => array('cor' => 'azul'))
);
$resultado = Set::sort($a, '{n}.Pessoa.nome', 'asc');
/* $resultado:
Array
(
        [0] => Array
        (
                [Camisa] => Array
                (
                        [cor] => azul
                )
        )
        [1] => Array
```

```
        (
                [Pessoa] => Array
                (
                        [name] => Cadu
                )
        )
)
*/

$resultado = Set::sort($a, '{n}.Camisa', 'asc');
/* $resultado:
Array
(
        [0] => Array
        (
                [Pessoa] => Array
                (
                        [nome] => Cadu
                )
        )
        [1] => Array
        (
                [Camisa] => Array
                (
                        [cor] => azul
                )
        )
)
*/

$resultado = Set::sort($a, '{n}', 'desc');
/* $resultado:
Array
(
        [0] => Array
        (
                [Camisa] => Array
                (
                        [cor] => azul
                )
        )
        [1] => Array
        (
                [Pessoa] => Array
                (
                        [nome] => Cadu
                )
        )
)
*/

$a = array(
        array(7,6,4),
        array(3,4,5),
        array(3,2,1),
```

```
);

$resultado = Set::sort($a, '{n}.{n}', 'asc');
/* $resultado:
Array
(
        [0] => Array
        (
                [0] => 3
                [1] => 2
                [2] => 1
        )
        [1] => Array
        (
                [0] => 3
                [1] => 4
                [2] => 5
        )
        [2] => Array
        (
                [0] => 7
                [1] => 6
                [2] => 4
        )
)
*/
```

3.8.4.3 - reverse

Converte um objeto para um array. Este método é o oposto do método map.

array Set::reverse ($object)

```
$resultado = Set::reverse(null);
// Null
$resultado = Set::reverse(false);
// false
$a = array(
        'Artigo' => array('id'=> 1, 'titulo' => 'Primeiro Artigo'),
        'Comentario' => array(
                array('id'=> 1, 'titulo' => 'Primeiro Comentatio'),
                array('id'=> 2, 'titulo' => 'Segundo Comentatio')
        ),
        'Tag' => array(
                array('id'=> 1, 'titulo' => 'Primeira Tag'),
                array('id'=> 2, 'titulo' => 'Segunda Tag')
        ),
);
$map = Set::map($a); // Transforma $a em uma classe object
/* $map:
stdClass Object
(
        [_name_] => Artigo
        [id] => 1
        [titulo] => Primeiro Artigo
```

```
        [Comentario] => Array
        (
                [0] => stdClass Object
                (
                        [id] => 1
                        [titulo] => Primeiro Comentatio
                )
                [1] => stdClass Object
                (
                        [id] => 2
                        [titulo] => Segundo Comentatio
                )
        )
        [Tag] => Array
        (
                [0] => stdClass Object
                (
                        [id] => 1
                        [titulo] => Primeira Tag
                )
                [1] => stdClass Object
                (
                        [id] => 2
                        [titulo] => Segunda Tag
                )
        )
)
*/

$resultado = Set::reverse($map);
/* $resultado:
Array
(
        [Artigo] => Array
        (
                [id] => 1
                [titulo] => Primeiro Artigo
                [Comentario] => Array
                (
                        [0] => Array
                        (
                                [id] => 1
                                [titulo] => Primeiro Comentatio
                        )
                        [1] => Array
                        (
                                [id] => 2
                                [titulo] => Segundo Comentatio
                        )
                )
                [Tag] => Array
                (
                        [0] => Array
                        (
                                [id] => 1
```

```
                                [titulo] => Primeira Tag
                        )
                        [1] => Array
                        (
                                [id] => 2
                                [titulo] => Segunda Tag
                        )
                )
        )
)
*/

$resultado = Set::reverse($a['Artigo']);
/* $resultado:
Array
(
        [id] => 1
        [titulo] => Primeiro Artigo
)
*/
```

3.8.4.4 - combine

Retorna um array a partir de um array ($data), usa o $path1 para gerar as chaves e o $path2 para gerar os valores do array que será retornado.

array Set::combine ($data, $path1 = null, $path2 = null, $groupPath = null)

```
$a = array(
        array('Usuario' => array(
                'id' => 2,
                'grupo_id' => 1,
                'Dados' => array(
                        'usuario' => 'cadu',
                        'nome' => 'Carlos Pires')
                )
        ),
        array('Usuario' => array(
                'id' => 14,
                'grupo_id' => 2,
                'Dados' => array(
                        'usuario' => 'daniel',
                        'nome' => 'Daniel Golgher')
                )
        ),
        array('Usuario' => array(
                'id' => 25,
                'grupo_id' => 1,
                'Dados' => array(
                        'usuario' => 'leo',
                        'nome' => 'Leonardo Golgher')
                )
        )
);
```

```
$resultado = Set::combine($a, '{n}.Usuario.id');
/* $resultado:
Array
(
        [2] =>
        [14] =>
        [25] =>
)
*/

$resultado = Set::combine($a, '{n}.Usuario.id', '{n}.Usuario.campo-inexistente');
/* $resultado:
Array
(
        [2] =>
        [14] =>
        [25] =>
)
*/

$resultado = Set::combine($a, '{n}.Usuario.id', '{n}.Usuario.Dados');
/* $resultado:
Array
(
        [2] => Array
        (
                [usuario] => cadu
                [nome] => Carlos Pires
        )
        [14] => Array
        (
                [usuario] => daniel
                [nome] => Daniel Golgher
        )
        [25] => Array
        (
                [usuario] => leonardo
                [nome] => Leonardo Golgher
        )
)
*/

$resultado = Set::combine($a, '{n}.Usuario.id', '{n}.Usuario.Dados.nome');
/* $resultado:
Array
(
        [2] => Carlos Pires
        [14] => Daniel Golgher
        [25] => Leonardo Golgher
)
*/

$resultado = Set::combine($a,
        '{n}.Usuario.id',
        '{n}.Usuario.Dados',
```

```
        '{n}.Usuario.grupo_id'
);
/* $resultado:
Array
(
        [1] => Array
        (
                [2] => Array
                (
                        [usuario] => cadu
                        [nome] => Carlos Pires
                )
                [25] => Array
                (
                        [usuario] => leonardo
                        [nome] => Leonardo Golgher
                )
        )
        [2] => Array
        (
                [14] => Array
                (
                        [usuario] => daniel
                        [nome] => Daniel Golgher
                )
        )
)
*/

$resultado = Set::combine($a,
        '{n}.Usuario.id',
        '{n}.Usuario.Dados.nome',
        '{n}.Usuario.grupo_id'
);
/* $resultado:
        Array
        (
            [1] => Array
                (
                    [2] => Carlos Pires
                    [25] => Leonardo Golgher
                )
            [2] => Array
                (
                    [14] => Daniel Golgher
                )
        )
*/

$resultado = Set::combine($a,
        '{n}.Usuario.id',
        array('{0}: {1}', '{n}.Usuario.Dados.usuario',
'{n}.Usuario.Dados.nome'),
        '{n}.Usuario.grupo_id'
);
```

```
/* $resultado:
Array
(
    [1] => Array
        (
            [2] => cadu: Carlos Pires
            [25] => leo: Leonardo Golgher
        )
    [2] => Array
        (
            [14] => daniel: Daniel Golgher
        )
)
*/

$resultado = Set::combine($a,
        array('{0}: {1}','{n}.Usuario.Dados.usuario','{n}.Usuario.Dados.nome'),
        '{n}.Usuario.id'
);
/* $resultado:
Array
(
    [cadu: Carlos Pires] => 2
    [daniel: Daniel Golgher] => 14
    [leo: Leonardo Golgher] => 25
)
*/

$resultado = Set::combine($a,
        array('{1}: {0}', '{n}.Usuario.Dados.usuario','{n}.Usuario.Dados.nome'),
        '{n}.Usuario.id'
);
/* $resulado:
Array
(
    [Carlos Pires: cadu] => 2
    [Daniel Golgher: daniel] => 14
    [Leonardo Golgher: leo] => 25
)
*/

$resultado = Set::combine($a,
        array('%1$s: %2$d', '{n}.Usuario.Dados.usuario', '{n}.Usuario.id'),
        '{n}.Usuario.Dados.nome'
);

/* $resultado:
Array
(
    [cadu: 2] => Carlos Pires
    [daniel: 14] => Daniel Golgher
    [leo: 25] => Leonardo Golgher
)
*/
```

```
$resultado = Set::combine($a,
        array('%2$d: %1$s', '{n}.Usuario.Dados.usuario', '{n}.Usuario.id'),
        '{n}.Usuario.Dados.nome'
);
/* $resultado:
Array
(
    [2: cadu] => Carlos Pires
    [14: daniel] => Daniel Golgher
    [25: leo] => Leonardo Golgher
)
*/
```

3.8.4.5 - normalize

Normaliza um array.

array Set::normalize ($list, $assoc = true, $sep = ',', $trim = true)

```
$a = array('Opção 1', 'Opção 2','Opção 3' => array(
                'teste 1','teste 2','teste 3'
        )
);

$resultado = Set::normalize($a);
/* $resultado:
Array
(
    [Opção 1] =>
    [Opção 2] =>
    [Opção 3] => Array
        (
            [0] => teste 1
            [1] => teste 2
            [2] => teste 3
        )

)
*/
```

3.8.4.6 - countDim

Conta as dimensões de um array.

integer Set::countDim ($array = null, $all = false, $count = 0)

```
$data = array('um', '2', 'três');
$resultado = Set::countDim($data);
// $resultado == 1
$data = array(
        '1' => array(
                '1.1' => '1.1.1'
        ),
        '2',
        '3' => array(
                '3.1' => array(
```

```
                        '3.1.1' => '3.1.1.1'
                )
        )
);
$resultado = Set::countDim($data, true);
// $resultado == 3
```

3.8.4.7 - isEqual

Verifica se dois array são iguais.

boolean Set::isEqual ($val1, $val2 = null)

```
$resultado = Set::isEqual(array(1), array(1,1));
// False
$resultado = Set::isEqual(array(1), array(1));
// True
```

3.8.4.8 - diff

Retorna um array contendo a diferença entre dois arrays.

array Set::diff ($val1, $val2 = null)

```
$a = array(
        0 => array('nome' => 'principal'),
        1 => array('nome' => 'ajuda')
);
$b = array(
        0 => array('nome' => 'principal'),
        1 => array('nome' => 'ajuda'),
        2 => array('nome' => 'contato')
);

$resultado = Set::diff($a, $b);
/* $resultado:
        Array
        (
            [2] => Array
                (
                    [nome] => contato
                )
        )
*/
```

3.8.4.9 - check

Verifica se uma determinada parte do array existe.

boolean Set::check ($data, $path = null)

```
$set = array(
        'Primeiro Item' => array('Primeiro' => 'valor')
);
$resultado = Set::check($set, 'Primeiro Item.Primeiro');
// $resultado == True
```

```
$resultado = Set::check($set, 'Primeiro Item');
// $resultado == True
$resultado = Set::check($set, array());
// $resultado == array('Primeiro Item' => array('Primeiro' => 'valor'))
$resultado = Set::check($set, 'Primeiro Item.Segundo');
// $resultado == false
```

3.8.4.10 - remove

Remove o elemento definido no $path de um array.

boolean Set::remove ($list, $path = null)

```
$a = array(
        'pagina' => array('nome' => 'pagina'),
        'arquivos' => array('nome' => 'arquivos')
);

$resultado = Set::remove($a, 'arquivos');
/* $resultado:
Array
(
    [paginas] => Array
        (
            [nome] => pagina
        )

)
*/
```

3.8.4.11 - classicExtract

Retorna um array extraído de outro array, baseado no $path.

array Set::classicExtract ($data, $path = null)

```
$a = array(
        array('Artigo' => array('id' => 1, 'titulo' => 'Artigo 1')),
        array('Artigo' => array('id' => 2, 'titulo' => 'Artigo 2')),
        array('Artigo' => array('id' => 3, 'titulo' => 'Artigo 3')));
$resultado = Set::classicExtract($a, '{n}.Article.id');
/* $resultado now looks like:
Array
(
        [0] => 1
        [1] => 2
        [2] => 3
)
*/
```

3.8.4.12 - matches

Verifica se uma condição é verdadeira no conteúdo de um array.

boolean Set::matches ($conditions, $data=array(), $i = null, $length=null)

```
$a = array(
```

```
        array('Artigo' => array('id' => 1, 'titulo' => 'Artigo 1')),
        array('Artigo' => array('id' => 2, 'titulo' => 'Artigo 2')),
        array('Artigo' => array('id' => 3, 'titulo' => 'Artigo 3')));
$res=Set::matches(array('id>2'), $a[1]['Artigo']);
// retorna false
$res=Set::matches(array('id>=2'), $a[1]['Artigo']);
// retorna true
$res=Set::matches(array('id>=3'), $a[1]['Artigo']);
// retorna false
$res=Set::matches(array('id<=2'), $a[1]['Artigo']);
// retorna true
$res=Set::matches(array('id<2'), $a[1]['Artigo']);
// retorna false
$res=Set::matches(array('id>1'), $a[1]['Artigo']);
// retorna true
$res=Set::matches(array('id>1', 'id<3', 'id!=0'), $a[1]['Artigo']);
// retorna true
$res=Set::matches(array('3'), null, 3);
// retorna true
$res=Set::matches(array('5'), null, 5);
// retorna true
$res=Set::matches(array('id'), $a[1]['Artigo']);
// retorna true
$res=Set::matches(array('id', 'title'), $a[1]['Artigo']);
// retorna true
$res=Set::matches(array('non-existant'), $a[1]['Artigo']);
// retorna false
$res=Set::matches('/Artigo[id=2]', $a);
// retorna true
$res=Set::matches('/Artigo[id=4]', $a);
// retorna false
$res=Set::matches(array(), $a);
// retorna true
```

3.8.4.13 - extract

Retorna um array extraído de outro array, baseado no $path. Similar ao classicExtract.

Seletor	Nota
/Usuario/id	Similar ao classicExtract {n}.Usuario.id
/Usuario[2]/name	Seleciona o nome do segundo usuário
/Usuario[id<2]	Seleciona todos os usuários com id < 2
/Usuario[id>2][<5]	Seleciona os usuários entre id>2 e id<5
/Artigo/Comentario[nome_do_autor=joão]/../nome	Seleciona o nome de todos os Artigos que tem pelo menos um comentário escrito por joão
/Artigos[titulo]	Seleciona todos os artigos que possuem a chave titulo
/Comentario/.[1]	Seleciona todo o conteúdo do primeiro cometário
/Comentario/.[:last]	Seleciona o último cometário
/Comentario/.[:first]	Seleciona o primeiro cometário
/Comentario[texto=/cakephp/i]	Seleciona todos os comentários cujo o texto contenha a expressão regular /cake/i

Seletor	Nota
/Comentario/@*	Seleciona a chave name de todos os cometários

3.8.4.14 - format

Extrai valores de um array formatado de acordo com as chaves passadas.

array Set::format ($data, $format, $keys)

```
$data = array(
        array(
                'Usuario' => array(
                        'nome' => 'Carlos',
                        'sobrenome' => 'Pires'
                )
        ),
        array(
                'Usuario' => array(
                        'nome' => 'Daniel',
                        'sobrenome' => 'Golgher'
                )
        ),
        array(
                'Usuario' => array(
                        'nome' => 'Leonardo',
                        'sobrenome' => 'Golgher'
                )
        )
);

$res = Set::format($data, '{1}, {0}', array('{n}.Usuario.nome',
'{n}.Usuario.sobrenome'));
/*
Array
(
    [0] => Pires, Carlos
    [1] => Golgher, Daniel
    [2] => Golgher, Leonardo
)
*/
```

3.8.4.15 - enum

Retorna o valor de um array se a chave passada existir.

string Set::enum ($select, $list=null)

```
$res = Set::enum(1, 'um, dois');
// $res é 'dois'

$res = Set::enum('nao', array('nao' => 0, 'sim' => 1));
// $res é 0

$res = Set::enum('primeiro', array('primeiro' => 'um', 'segundo' => 'dois'));
// $res é 'um'
```

3.8.4.16 - numeric

Verifica se um valor no array é numérico.

array Set::numeric ($array=null)

```
$data = array('um');
$res = Set::numeric(array_keys($data));
// $res é true

$data = array(1 => 'um');
$res = Set::numeric($data);
// $res é false

$data = array('um');
$res = Set::numeric($data);
// $res é false

$data = array('um' => 'dois');
$res = Set::numeric($data);
// $res é false

$data = array('um' => 1);
$res = Set::numeric($data);
// $res é true
```

3.8.4.17 - map

A Função map transforma arrays em objetos, por padrão da classe stdClass.

object Set::map ($class = 'stdClass', $tmp = 'stdClass')

```
$data = array(
        array(
                "IndexedPage" => array(
                        "id" => 1,
                        "url" => 'http://blah.com/',
                        'hash' => '68a9f053b19526d08e36c6a9ad150737933816a5',
                        'get_vars' => '',
                        'redirect' => '',
                        'created' => "1195055503",
                        'updated' => "1195055503",
                )
        ),
        array(
                "IndexedPage" => array(
                        "id" => 2,
                        "url" => 'http://blah.com/',
                        'hash' => '68a9f053b19526d08e36c6a9ad150737933816a5',
                        'get_vars' => '',
                        'redirect' => '',
                        'created' => "1195055503",
                        'updated' => "1195055503",
                ),
        )
);
```

```
$mapped = Set::map($data);

/* $mapped:

Array
(
    [0] => stdClass Object
        (
            [_name_] => IndexedPage
            [id] => 1
            [url] => http://blah.com/
            [hash] => 68a9f053b19526d08e36c6a9ad150737933816a5
            [get_vars] =>
            [redirect] =>
            [created] => 1195055503
            [updated] => 1195055503
        )

    [1] => stdClass Object
        (
            [_name_] => IndexedPage
            [id] => 2
            [url] => http://blah.com/
            [hash] => 68a9f053b19526d08e36c6a9ad150737933816a5
            [get_vars] =>
            [redirect] =>
            [created] => 1195055503
            [updated] => 1195055503
        )

)

*/
```

3.8.4.18 - pushDiff

A função faz um *merge* de dois arrays e retorna um array com todos os dados, ignorando as chaves iguais.

```
$a = array(
        'a'=>'Este entra',
        'b'=>'Este entra',
        'c'=>'Este entra');
$b = array(
        'a'=>'Este nao entra',
        'b'=>'Este tb fica de fora',
        'd'=>'Este entra');
$resultado = Set::pushDiff($a,$b);
/*
$resultado:
Array
(
    [a] => Este entra
    [b] => Este entra
    [c] => Este entra
```

```
    [d] => Este entra
)
*/
```

3.8.4.19 - filter

A função limpa elementos de um array de tota, exclindo '0'.

array Set::filter ($var, $isArray=null)

```
$res = Set::filter(array(
        '0',
        false,
        true,
        0,
        array('um', 'dois', 'três', false)));

/* $res:
        Array (
            [0] => 0
            [2] => 1
            [3] => 0
            [4] => Array
                (
                    [0] => um
                    [1] => dois
                    [2] => três
                    [3] =>
                )
        )
*/
```

3.8.4.20 - merge

Faz o *merge* de dois arrays.

array Set::merge ($arr1, $arr2=null)

```
$a = array(
        'a'=>'Este nao entra',
        'b'=>'Este tb nao entra',
        'c'=>'Este entra');
$b = array(
        'a'=>'Este entra',
        'b'=>'Este entra',
        'd'=>'Este entra');
$resultado = Set::merge($a,$b);
/*
$resultado:
Array
(
    [a] => Este entra
    [b] => Este entra
    [c] => Este entra
    [d] => Este entra
)
```

```
*/
```

3.8.4.21 - contains

Verifica se oprimeiro array contém as mesmas chaves e valores do segundo.

boolean Set::contains ($val1, $val2 = null)

```
$a = array(
        'a'=>'Este nao entra',
        'b'=>'Este tb nao entra');
$b = array(
        'a'=>'Este entra',
        'b'=>'Este entra',
        'd'=>'Este entra');
$resultado = Set::contains($a,$a);
//true
$resultado = Set::contains($a,$b);
//false
$resultado = Set::contains($b,$a);
//true
```

3.8.5 - Cache

3.8.5.1 - Cache::read()

Este método é usado para recuperar um valor que está em cache.

Cache::read($key, $config = null)

3.8.5.2 - Cache::write()

Este método salva um valor em cache.

Cache::write($key, $value, $config = null);

```
if (($artigos = Cache::read('artigos')) === false) {
        $artigos = $this->Artigo->find('all');
        Cache::write('artigos', $artigos);
}
```

3.8.5.3 - Cache::delete()

Este método apaga um cache.

Cache::delete($key, $config = null)

```
Cache::delete('artigos');
```

3.8.5.4 - Cache::config()

Este método define algumas configurações do cache, as configurações podem mudar dependendo do mecanismo usado para fazer o cache.

config($name = NULL,$settings = array())

```
//O nome gerado nesta configuração pode ser passado nos métodos:
//write, read, delete
```

```
Cache::config('short', array(
    'engine' => 'File',
    'duration'=> '+1 hours',
    'path' => CACHE,
    'prefix' => 'cake_short_'
));


Cache::config('long', array(
    'engine' => 'File',
    'duration'=> '+1 week',
    'probability'=> 100,
    'path' => CACHE . 'long' . DS,
));
```

3.8.5.5 - Cache::set()

Altera temporariamente uma configuração do cache.

```
Cache::set(array('duration' => '+30 days'));
Cache::write('results', $data);

Cache::set(array('duration' => '+30 days'));
$results = Cache::read('results');
```

3.8.6 - HTTPSocket

3.8.6.1 - get

O método get faz uma requisição do tipo HTTP GET.

```
App::import('Core', 'HttpSocket');
$HttpSocket = new HttpSocket();
$resultado = $HttpSocket->get('www.google.com/search', 'q=cakephp');
//$resultado é o html retornado pela busca no google.
```

3.8.6.2 - post

O método post faz uma requisição do tipo HTTP POST.

string function post ($uri, $data, $request)

```
App::import('Core', 'HttpSocket');
$HttpSocket = new HttpSocket();
$resultado = $HttpSocket->post('www.exemplo.com/add', array(
        'nome' => 'cadu',
        'tipo' => 'admin'));
//$resultado contém a resposta da requisição.
```

3.8.6.3 - delete

O método delete faz uma requisição do tipo HTTP DELETE.

string function delete($uri, $data, $request)

```
App::import('Core', 'HttpSocket');
```

```
$HttpSocket = new HttpSocket();
$resultado = $HttpSocket->delete('www.exemplo.com/delete', array('id' => '7'));
//$resultado contém a resposta da requisição.
```

3.8.6.4 put

O método put faz uma requisição do tipo HTTP PUT.

string function put($uri, $data, $request)

```
App::import('Core', 'HttpSocket');
$HttpSocket = new HttpSocket();
$resultado = $HttpSocket->put('www.exemplo.com/edit', array(
        'id' => '7',
        'nome' => 'cadu',
        'tipo' => 'admin'));
//$resultado contém a resposta da requisição.
```

3.8.6.5 - request

Este método permite fazer qualquer requisição HTTP (POST, PUT, GET, DELETE)

string function request($request)

```
var $request = array(
        'method' => 'GET',
        'uri' => array(
                'scheme' => 'http',
                'host' => null,
                'port' => 80,
                'user' => null,
                'pass' => null,
                'path' => null,
                'query' => null,
                'fragment' => null
        ),
        'auth' => array(
                'method' => 'Basic',
                'user' => null,
                'pass' => null
        ),
        'version' => '1.1',
        'body' => '',
        'line' => null,
        'header' => array(
                'Connection' => 'close',
                'User-Agent' => 'CakePHP'
        ),
        'raw' => null,
        'cookies' => array()
);
```

3.9 - Debugging

3.9.1 - debug

Este é um método global e produz um resultado similar a função print_r() do php

debug($var, $showHTML = false, $showFrom = true)

Os parâmetros são:

1. A variável que será imprimida
2. A formatação do resultado se deve ser html ou textual.
3. Imprimir apenas quando a aplicação está com debug(0).

3.9.2 - Debugger::dump

Nota

Para utilizar a classe Debbuger verifique se a configuração de debug da aplicação é maior que 0 (zero).

Imprime o conteúdo de uma variável.

```
$array = array(1,2,3);

Debugger::dump($array);

//saída
array(
        1,
        2,
        3
)

//Objeto
$usuarios = new Usuario();

Debugger::dump($usuarios);

//outputs
Usuario::
Usuario::idade = '20'
Usuario::altura = '1.70'
Usuario::peso = '75'
Usuario::insere()
Usuario::getPeso()
Usuario::getIdade()
```

3.9.3 - Debugger::log

log($var, $level = 7)

Este método grava no arquivo /app/tmp/debug.log uma saída similar ao do dump().

3.9.4 - Debugger::trace

Cria a *stack trace* da requisição.

```
        //PostsController::index()
        pr( Debugger::trace() );
```

```
        //saída
        PostsController::index() - APP/controllers/downloads_controller.php,
line 48
        Dispatcher::_invoke() - CORE/cake/dispatcher.php, line 265
        Dispatcher::dispatch() - CORE/cake/dispatcher.php, line 237
        [main] - APP/webroot/index.php, line 84
```

3.9.5 - Debugger::excerpt

Exibe um trecho de um arquivo. Este método é usado internamente no Cake para exibir algum trecho de cógigo errado.

excerpt($file, $line, $context)

O primeiro parâmetro é o arquivo, o segundo a linha que deseja exibir e o terceiro é quantas linhas antes e depois devem ser mostradas.

3.9.6 - Debugger::exportVar

Transforma qualquer tipo de variável em uma string.

3.9.7 - Debugger::invoke

Substitui o Debbuger do cakephp para uma nova gestão de de erros (Error Handler).

3.10 - Logging

O Cake permite gravar mensagens de log de forma fácil, usando o método **log**. Este método recebe dois parâmetros, o primeiro é a mensagem, o segundo é o arquivo onde será gravado.

```
$this->log("Alguma coisa não funcionou bem!");

//O resultado será adicionado no arquivo: app/tmp/logs/error.log
2007-11-02 10:22:02 Error: Alguma coisa não funcionou bem!
```

```
$this->log('Mensagem para debug.', LOG_DEBUG);

//O resultado será adicionado no arquivo: app/tmp/logs/debug.log
2007-11-02 10:22:02 Error: Mensagem para debug.
```

```
$this->log('Log em um outro arquivo', 'teste');

//O resultado será adicionado no arquivo: app/tmp/logs/teste.log
2007-11-02 10:22:02 Activity: Log em um outro arquivo
```

4 - Localização e Internacionalização

Uma das melhores maneiras para que uma aplicação atinja o maior número de usuários é codificá-la de tal forma que ela dê suporte a diferentes idiomas. Esta tarefa, em um primeiro momento, pode lhe desencorajar, mas com os recursos de localização e internacionalização do CakePHP ela torna-se muito mais fácil.

4.1 - Localização (l10n)

O termo *localization* refere-se à adaptação de uma aplicação para atender a requisitos específicos da

língua (ou cultura). Ele é geralmente abreviado como l10n. O número 10 é o número de caracteres entre o primeiro e o último caracter da palavra *localization*.

Toda classe de controle que utilizar um conteúdo localizado deve incluir a classe utilitária L10N do CakePHP.

```
// Inclui a classe utilitária L10n:
App::import('Core', 'l10n');
class UsuariosController extends AppController {
    ...
}
```

4.1.1 - Extraindo o arquivo POT dos arquivos do projeto

Para fazer a tradução dos arquivos dos projeto, é preciso gerar um arquivo POT. O arquivo POT é um arquivo tipo texto que irá conter todos pares de strings encontrados nos arquivos do projeto que estejam sendo exibidos através da função `__()`.

Vamos utilizar o console do CakePHP para gerar o arquivo POT:

```
$ cake i18n
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
I18n Shell
---------------------------------------------------------------
[E]xtract POT file from sources
[I]nitialize i18n database table
[H]elp
[Q]uit
```

Vamos selecionar a opção para extrair o arquivo POT dos arquivos fonte do projeto pressionando a tecla `e` e depois `Enter`.

```
What would you like to do? (E/I/H/Q)
> e
```

Vamos conferir o caminho completo de onde ele irá extrair as strings e confirmar pressionando `Enter`.

```
What is the full path you would like to extract?
Example: /projetos/myapp
[Q]uit
[/projetos/bookmarks] >
```

Vamos confirmar o caminho para onde ele irá salvar o arquivo gerado pressionando a `Enter`.

```
What is the full path you would like to output?
Example: /projetos/bookmarks/locale
[Q]uit
[/projetos/bookmarks/locale] >
```

```
Extracting...
---------------------------------------------------------------
Path: /projetos/bookmarks
Output Directory: /projetos/bookmarks/locale/
---------------------------------------------------------------
```

Vamos confirmar a opção para que ele combine todos as strings em um único arquivo pressinando `Enter`.

```
Would you like to merge all translations into one file? (y/n)
[y] >
```

Vamos confirmar o nome padrão do arquivo a ser gerado (`default.pot`) pressionando `Enter`.

```
What should we name this file?
[default] >
Processing /projetos/bookmarks/app_controller.php...
Processing /projetos/bookmarks/app_helper.php...
Processing /projetos/bookmarks/app_model.php...
Processing /projetos/bookmarks/index.php...
Processing /projetos/bookmarks/config/acl.ini.php...
Processing /projetos/bookmarks/config/bootstrap.php...
Processing /projetos/bookmarks/config/core.php...
Processing /projetos/bookmarks/config/database.php...
Processing /projetos/bookmarks/config/inflections.php...
Processing /projetos/bookmarks/config/routes.php...
Processing /projetos/bookmarks/config/sql/db_acl.php...
Processing /projetos/bookmarks/config/sql/i18n.php...
Processing /projetos/bookmarks/config/sql/sessions.php...
Processing /projetos/bookmarks/controllers/bookmarks_controller.php...
Processing /projetos/bookmarks/controllers/pages_controller.php...
Processing /projetos/bookmarks/controllers/tags_controller.php...
Processing /projetos/bookmarks/controllers/usuarios_controller.php...
Processing /projetos/bookmarks/models/bookmark.php...
Processing /projetos/bookmarks/models/tag.php...
Processing /projetos/bookmarks/models/usuario.php...
Processing /projetos/bookmarks/views/bookmarks/add.ctp...
Processing /projetos/bookmarks/views/bookmarks/edit.ctp...
Processing /projetos/bookmarks/views/bookmarks/index.ctp...
Processing /projetos/bookmarks/views/bookmarks/view.ctp...
Processing /projetos/bookmarks/views/elements/email/html/default.ctp...
Processing /projetos/bookmarks/views/elements/email/text/default.ctp...
Processing /projetos/bookmarks/views/layouts/ajax.ctp...
Processing /projetos/bookmarks/views/layouts/default.ctp...
Processing /projetos/bookmarks/views/layouts/flash.ctp...
Processing /projetos/bookmarks/views/layouts/email/html/default.ctp...
Processing /projetos/bookmarks/views/layouts/email/text/default.ctp...
Processing /projetos/bookmarks/views/layouts/js/default.ctp...
Processing /projetos/bookmarks/views/layouts/rss/default.ctp...
Processing /projetos/bookmarks/views/layouts/xml/default.ctp...
Processing /projetos/bookmarks/views/pages/home.ctp...
Processing /projetos/bookmarks/views/tags/add.ctp...
Processing /projetos/bookmarks/views/tags/edit.ctp...
Processing /projetos/bookmarks/views/tags/index.ctp...
Processing /projetos/bookmarks/views/tags/view.ctp...
```

```
Processing /projetos/bookmarks/views/usuarios/add.ctp...
Processing /projetos/bookmarks/views/usuarios/edit.ctp...
Processing /projetos/bookmarks/views/usuarios/index.ctp...
Processing /projetos/bookmarks/views/usuarios/view.ctp...
Processing /projetos/bookmarks/webroot/css.php...
Processing /projetos/bookmarks/webroot/index.php...
Processing /projetos/bookmarks/webroot/test.php...
Processing /projetos/bookmarks/webroot/js/vendors.php...
Done.
---------------------------------------------------------------
I18n Shell
---------------------------------------------------------------
[E]xtract POT file from sources
[I]nitialize i18n database table
[H]elp
[Q]uit
```

Observe que o shell percorreu todos os arquivos do projeto extraindo as strings e gerou um único arquivo no caminho indicado */projetos/bookmarks/locale/default.pot*. Agora podemos sair do console pressionando a tecla `q` e depois `Enter`.

```
What would you like to do? (E/I/H/Q)
> q
```

Em seguida você deverá copiar os arquivos que irão guardar as strings localizadas para cada idioma. Cada idioma deve possuir um arquivo `default.po` em um diretório com o nome do idioma a que se refere. Veja alguns exemplos:

```
/bookmarks/locale/fre/LC_MESSAGES/default.po (Francês)
/bookmarks/locale/ita/LC_MESSAGES/default.po (Italiano)
/bookmarks/locale/spa/LC_MESSAGES/default.po (Espanhol)
```

A pasta *locale* encontra-se na pasta raíz da aplicação *bookmarks*. O código de três caracteres do *locale* está de acordo com o padrão ISO 639-2 (outros exemplos podem ser encontrados no website da biblioteca do Congresso em http://www.loc.gov/standards/iso639-2/php/code_list.php) [em inglês].

Nota

O locale padrão do CakePHP é o `en_us`. Para as aplicações na nossa língua, o português do Brasil, o *locale* indicado é o `pt_br` e o nome da pasta a ser criada dentro da pasta *locale* é *por*. O caminho completo para o arquivo `default.po` deverá ser o seguinte:

```
/bookmarks/locale/por/LC_MESSAGES/default.po
```

Cada string no arquivo `default.po` deve ser única e possuir um valor correspondente. Um exemplo básico de o que se espera estar em um arquivo .po do idioma Português:

```
msgid "help"
msgstr "Clique no ícone para obter ajuda"
msgid "save"
msgstr "Clique no botão salvar para salvar os dados."
```

Após a criação correta dos arquivos .po, sua aplicação pode ser considerada localizada.

4.2 - Internacionalização (i18n)

O termo *internationalization* refere-se à habilidade de uma aplicação ser localizada.

O primeiro passo para se utilizar dados localizados é informar ao CakePHP qual a língua a aplicação deverá utilizar para servir o conteúdo. Isto pode ser feito detectando sub-domínios (pt.exemplo.com versus en.exemplo.com), ou verificando o *user-agent* do navegador dentre outras formas.

A troca de linguagem deve ser feita no *controller*:

```
$this->L10n = new L10n();
$this->L10n->get("eng");
```

Você pode querer colocar este código no *beforeFilter* assim toda ação no *controller* será exibida utilizando a língua correta, ou você pode querer colocá-lo em uma ação que controla a autenticação ou uma maneira para alterar o padrão.

A exibição do conteúdo localizado é feita utilizando a função __(). Esta função está disponível globalmente, mas ela será melhor utilizada nas suas classes de visão. O primeiro parâmetro da função é a chave msgid definida no arquivo .po. O conteúdo localizado é exibido por padrão, mas um segundo parâmetro opcional pode ser utilizado para que o valor seja retornado. (Útil para marcar ou criar links utilizando o TextHelper, por exemplo). O pequeno exemplo abaixo mostra como dar saída a um conteúdo localizado utilizando a função __(). A saída atual depende do locale que foi selecionado utilizando a classe L10n e a configuração ativa que foi definida.

```
<?php __("carrinho"); ?>
```

Lembre-se de utilizar o parâmetro *escape* nos métodos dos helpers caso você queira utilizar o conteúdo localizado como parte da chamada de método do helper. Note que o uso do segundo parâmetro do __() é para retornar os dados ao invés de imprimí-los (echo):

```
<?php
        echo $form->error(
            'Card.cardNumber',
            __("errorCardNumber", true),
            array('escape' => false)
        );
?>
```

Se você quiser que todas as suas mensagens de validação de erro sejam traduzidas por padrão, uma simples solução seria adicionar o seguinte código no app_model.php:

```
function invalidate($field, $value = true) {
        return parent::invalidate($field, __($value, true));
}
```

4.2.1 - Utilizando o behavior Translate

O CakePHP possui um behavior para facilitar a internacionalização das aplicações. Vamos utilizar o shell para inicializar as tabelas que irão conter os dados traduzidos. Para isto vamos digitar:

```
$ cake i18n
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
```

```
---------------------------------------------------------------
I18n Shell
---------------------------------------------------------------
[E]xtract POT file from sources
[I]nitialize i18n database table
[H]elp
[Q]uit
```

Vamos selecionar o opção `Initialize i18n database table` pressionando a tecla `i` e depois `Enter`.

```
What would you like to do? (E/I/H/Q)
> i

Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
Cake Schema Shell
---------------------------------------------------------------

The following table(s) will be dropped.
i18n
```

Vamos confirmar a remoção da tabela i18n pressionando a tecla `y` e depois `Enter`.

```
Are you sure you want to drop the table(s)? (y/n)
[n] > y
Dropping table(s).
i18n updated.

The following table(s) will be created.
i18n
```

Vamos confirmar a criação da tabela pressionando `Enter`.

```
Are you sure you want to create the table(s)? (y/n)
[y] >
Creating table(s).
i18n updated.
End create.
---------------------------------------------------------------
I18n Shell
---------------------------------------------------------------
[E]xtract POT file from sources
[I]nitialize i18n database table
[H]elp
[Q]uit
```

Vamos sair do shell digitando `q` e depois `Enter`.

```
What would you like to do? (E/I/H/Q)
> q
```

Para utilizar o behavior na aplicação, é preciso utilizar a variável `$actAs` e definir quais campos serão traduzidos, como no exemplo abaixo:

```
<?php
class Bookmark extends AppModel {
        var $name = 'Bookmark';
        var $actsAs = array(
                'Translate' => array('titulo','descricao')
        );
}
?>
```

Desta maneira, toda vez que um registro for atualizado ou criado, os campos `titulo` e `descricao` serão salvos na tabela `i18n` com o `locale` atual.

Nota

O `locale` atual é definido através do método `Configure::read('Config.language')`. Este valor é setado na classe L10n - a não ser que já tenha sido definido. De qualquer forma, o behavior Translate permite que este valor seja sobrescrito em tempo de execução.

Nota

As classes de modelo possuem uma variável *virtual* chamada `$locale` que indica qual *locale* será utilizado para recuperar/salvar os dados.

4.2.1.1 - Salvando em outras línguas

Para salvar os campos em outra língua é preciso alterar o valor da variável `$locale` no modelo antes de salvar os dados no banco. Isto pode ser feito na classe de controle ou diretamente na classe de modelo:

```
<?php
class Bookmark extends AppModel {

        var $name = 'Bookmark';

        function add() {
                if (!empty($this->data)) {
                        // Aqui vamos salvar a versão em inglês
                        $this->Bookmark->locale = 'en_us';
                        $this->Bookmark->create();
                        if ($this->Bookmark->save($this->data)) {
                                $this->Session->setFlash(__('Bookmark foi
salvo', true));
                                $this->redirect(array('action'=>'index'));
                        } else {
                                $this->Session->setFlash(__('The Bookmark could
not be saved. Please, try again.', true));
                        }
                }
                $tags = $this->Bookmark->Tag->find('list');
```

```
                $usuarios = $this->Bookmark->Usuario->find('list');
                $this->set(compact('tags', 'usuarios'));
        }
}
?>
```

Listagem 55 - Alterando o valor da variável `locale` na classe de controle.

```
<?php
class Bookmark extends AppModel {
        var $name = 'Bookmark';
        var $actsAs = array(
                'Translate' => array('titulo','descricao')
        );

        // Opção 1: Defina a propriedade diretamente
        var $locale = 'en_us';

        // Opção 2: Crie um método simples para alterá-la
        function setLanguage($locale) {
                $this->locale = $locale;
        }
}
?>
```

Listagem 56 - Alterando o valor da variável `locale` diretamente na classe de modelo.

4.2.1.2 - Recuperando todas as línguas traduzidas de um campo

Para recuperar as traduções salvas na tabela `i18n` é preciso especificar um nome para cada item do array *Translate* da variável `$actsAs`, como no exemplo abaixo:

```
var $actsAs = array(
        'Translate' => array(
                'titulo' => 'titulo_traduzido',
                'descricao' => 'descricao_traduzida'
        )
);
```

Desta forma, ao se recuperar os dados do modelo vamos ter o seguinte resultado:

```
Array
(
    [0] => Array
        (
            [Bookmark] => Array
                (
                    [id] => 1
                    [usuario_id] => 2
                    [status] => 1
                    [titulo] => Website da 2km interativa!
                    [url] => http://www.2km.com.br
                    [descricao] => Soluções livres para um mundo livre!
                    [hash] =>
                    [created] => 2009-04-20 15:51:19
                    [modified] => 2009-04-20 16:10:06
```

```
                    [locale] => pt_br
                )

            [titulo_traduzido] => Array
                (
                    [0] => Array
                        (
                            [id] => 1
                            [locale] => pt_br
                            [model] => Bookmark
                            [foreign_key] => 5
                            [field] => titulo
                            [content] => Website da 2km interativa!
                        )

                    [1] => Array
                        (
                            [id] => 2
                            [locale] => en_us
                            [model] => Bookmark
                            [foreign_key] => 5
                            [field] => titulo
                            [content] => 2km interativa!'s website
                        )

                )

            [descricao_traduzida] => Array
                (
                    [0] => Array
                        (
                            [id] => 3
                            [locale] => pt_br
                            [model] => Bookmark
                            [foreign_key] => 5
                            [field] => descricao
                            [content] => Soluções livres para um mundo livre!
                        )

                    [1] => Array
                        (
                            [id] => 4
                            [locale] => en_us
                            [model] => Bookmark
                            [foreign_key] => 5
                            [field] => descricao
                            [content] => Free solutions for a free world!
                        )

                )

        )

)
```

Importante

Se você deseja utilizar o recurso de localização da classe `l10n` e o behavior translate para o português do Brasil, você deve definir o locale no arquivo `core.php` assim:

```
Configure::write('Config.language','pt-br');
```

A língua também pode ser definida no arquivo `app_controller.php`:

```
App::import('Core', 'l10n');

class AppController extends Controller {
        function beforeFilter() {
                $this->L10n = new L10n();
                $this->L10n->get("pt-br");
        }
}
```

5 - Segurança

5.1 - Componente de Segurança

Através deste componente é possível integrar segurança à aplicação. É possível criar autenticação de requisições HTTP. Essa validação deve ser configurada no beforeFilter() dentro do controller. Existem diversos parâmetros que permitem trazer maior customização para a segurança de sua aplicação. Estas propriedades podem ser utilizadas diretamente ou através de métodos que possuem o mesmo nome. A seguir temos as propriedades que existem para o componente de segurança:

Propriedade	Descrição
$allowedActions	Ações de quais ações da classe de controle atual podem receber requisições. Ele pode ser utilizado para controlar requisições a mais de uma classe de controle.
$allowedControllers	Lista de controllers que as ações do controller atual permite receber requisições.
$blackHoleCallback	Callback do controller que irá lidar com requisições do tipo *blackholed*
$disabledFields	Lista de campos de formulários que devem ser ignorados em uma validação POST. O valor, presença ou ausência desses campos não será levada em consideração quando o envio de um formulário é válido.
$loginOptions	Opções para as requisições que precisam de validação de login. Permite que se escolha o tipo de validação e o controller callback para o processo de autenticação.
$loginUsers	Um array associativo para usuários => senhas, que são utilizados para a autenticação HTTP de logins. Se estiver sendo usado digest, o password deve ser encripitado com MD5.

Tabela 17 - Propriedades do componente de segurança

5.1.1 - Métodos do componente de Segurança

5.1.1.1 - blackHole(object $controller, string $error)

Este método deve ser chamado quando for necessário um *blackhole* para uma requisição inválida com um erro 404 ou uma função de *callback* customizada. Como parâmetros, devem ser passados o objeto da classe de controle e a string com a mensagem de erro. Caso nenhuma função de *callback* seja informada a requisição será terminada, caso contrário a função de *callback* definida será chamada com os parâmetros informados.

5.1.1.2 - generateDigestResponseHash(array $data)

Método que gera a resposta *hash* do HTTP *digest-authenticated*. O array $data deve ter sido gerado pelo método `SecurityComponent::parseDigestAuthData()`.

5.1.1.3 - loginCredentials(string $type)

Faz a validação de login na requisição. O parâmetro $type refere-se ao tipo de autenticação. Pode conter os valores 'basic' ou 'digest'. Se não for passado parâmetro algum ou o valor null, serão verificadas as duas formas. Se a validação for feita com sucesso, retorna um array com login e password.

5.1.1.4 - loginRequest(array $options)

Método utilizado para fazer login com autenticação HTTP. Em $options geralmente é utilizado os parâmetros 'type' e 'realm'. Em *Type* é especificado qual método de autenticação HTTP será utilizado. O padrão de *Realm* é o ambiente HTTP.

5.1.1.5 - parseDigestAuthData(string $digest)

Realiza o *parse* na requisição HTTP de autenticação digest *(HTTP digest authentication request)*. Retorna um array associativo caso tenha sucesso e null no caso de falha.

5.1.1.6 - requireAuth()

Define as ações que precisam de usar um token. Pode ter qualquer número de argumentos. Pode ser chamada sem argumento algum para forçar todas ações precisarem de validação com o token.

5.1.1.7 - requireLogin()

Define as ações que precisam de validação de login. Pode ter qualquer número de argumentos. Pode ser chamada sem argumento algum para forçar todas ações precisarem da validação de login.

5.1.1.8 - requirePost()

Define as ações que precisam de uma requisição POST. Pode ter qualquer número de argumentos. Pode ser chamada sem argumento algum para forçar todas ações precisarem da requisição POST.

5.1.1.9 - requireSecure()

Define as ações que precisam de uma requisição SSL. Pode ter qualquer número de argumentos. Pode ser chamada sem argumento algum para forçar todas ações precisarem da requisição SSL.

5.1.2 - Uso do componente de segurança

Geralmente o componente de segurança é usado no beforeFilter() do controller. Deve-se especificar as restrições de segurança e o componente de segurança que será usado.

```
<?php
class TagsController extends AppController {
```

```
        var $components = array('Security');

        function beforeFilter() {
                $this->Security->requirePost('edit');
        }
}
?>
```

Listagem 57 - Exemplo da utilização do método `requirePost()` do componente de segurança

No exemplo acima, a ação edit só ocorrerá se houver uma requisição do tipo POST.

```
<?php
class UsuariosController extends AppController {

        var $components = array('Security');

    function beforeFilter() {
        if(isset($this->params[Configure::read('Routing.admin')])){
              $this->Security->requireSecure();
        }
    }
}
?>
```

Listagem 58 - Exemplo da utilização do método `requireSecure()` do componente de segurança

No exemplo acima, todas as ações, que tem admin como rota, precisam de uma requisição SSL para ocorrerem.

5.1.3 - Sistema básico de autenticação HTTP

Existem situações onde determinados conteúdos devem ficar disponíveis a apenas um grupo restrito de clientes. A seguir temos um exemplo de como construir um sistema de autenticação básico. O termo básico foi usado porque os dados utilizados para validação são compostos de texto.

```
<?php
class UsuariosController extends AppController {

    var $name = 'Usuarios';
    var $uses = array();
    var $components = array('Security');

    function beforeFilter() {
       $this->Security->loginOptions = array(
                   'type'=>'basic',
                   'realm'=>'Area restrita'
                   );
       $this->Security->loginUsers = array(
                   'Daniel'=>'123456',
                   'Carlos'=>'654321'
                   );
       $this->Security->requireLogin();
   }

   function index() {
        //lógica de proteção da aplicação
```

```
    }
}
?>
```

A propriedade loginOptions do componente de segurança contém um array que determina como deve ser feita a verificação do login. O parâmetro *type* pode ter o valor *basic*. É possível especificar o parâmetro *realm* para mostrar uma mensagem para o cliente que estiver tentando logar, ou então, se hovuer diversas sessões de autenticação distintas então pode-se utilizar este parâmetro para separar as sessões. (realm = sessão)

A propriedade loginUsers contéum um array com o login e senha dos usuários que possuem permssão para acessar a área restrita. No exemplo foi usada uma lista estática, mas na maioria das vezes será necessário utilizar o model para buscar as informações de autenticação no banco de dados.

A propriedade requireLogin determina que este controller precisa de autenticação para ser acessado. É possível passar uma lista de métodos do controller como parâmetros, assim apenas os métodos listados serão alvo de autenticação. No nosso exemplo, como não foi passado parâmetro algum, todos os métodos terão necessidade de autenticação.

5.2 - Data Sanitization

Esta classe é utilizada para previnir o envio de dados maliciosos e outras informações indesejadas. Como se trata de uma biblioteca, pode ser utilizada em qualquer lugar do código, entretanto é recomendável utilizar no *controller* ou no *model*.

Para utilizar essa classe, basta importar a biblioteca para o seu código:

```
<?php

App::import('Sanitize');

?>
```

5.2.1 - Paranoid

paranoid(string $string, array $allowedChars);

Essa função tira tudo que estiver na variável $string que não for um caractere alphanumérico ou se for tag. É possível incluir excessões através do parâmetro $allowedChars.

```
<?php

  $badString = ";:<p><strong>Oi !#%</strong></p>";
        echo Sanitize::paranoid($badString);
        // output: pstrongOistrongp
        echo Sanitize::paranoid($badString, array(' ', '#'));
        // output: pstrongOi #strongp

?>
```

5.2.2 - html

html(string $string, boolean $remove = false);

Essa função é interessante de ser utilizada quando o usuário pode postar mensagens que serão publicadas. Para evitar que seja possível o layout ser quebrado. Assim se o usuário tentar inserir tags, imagens ou scripts na mensagem, estas serão retiradas. Se o parâmetro $remove for true, as tags serão totalmente retiradas da mensagem, se for false serão renderizadas com HTML entities.

```
<?php
```

```
    $badString = '<a href="#">Teste</a><script></script>';

    echo Sanitize::html($badString);
    // output: &lt;a href='#'&gt;Teste&lt;/a&gt;&lt;script&gt;&lt;/script&gt;

    echo Sanitize::html($badString, true);
    // output: Teste

?>
```

5.2.3 - escape

escape(string $string, string $connection)

Adiciona aspas a declarações do tipo SQL que existam em $string. O parâmetro $connection contém o nome do banco de dados, como definido no arquivo app/config/database.php.

5.2.4 - clean

Sanitize::clean(mixed $data, mixed $options)

Essa função tem como propósito limpar grande volumes de dados. Pode ser usada em arrays inteiros, como o $this->data. A função pega a string ou array passada no parâmetro $data e retorna sua versão limpa. Essa função executa os seguintes procedimentos:

- Mais de um espaço em branco ou 0xCA são substituídos por um espaço apenas.
- Confere se existem caracteres especiais e \n e os remove. Aumenta a segurança com códigos SQL maliciosos.
- Adiciona aspas para evitar códigos SQL maliciosos.
- Troca as barras digitadas pelo usuário por barras confiáveis.

O parâmetro $options pode ser uma string ou um array. Quando é fornecida apenas uma string, trata-se do nome da conexão com o banco de dados. Se for um array, as seguintes opções podem ser usadas:

- connection
- odd_spaces
- encode
- dollar
- carriage
- unicode
- escape
- backslash

A seguir temos um exemplo do uso de clean:

```
<?php

  Sanitize::clean($this->data, array('encode' => false));

?>
```

6 - Integrando aplicações

6.1 - Web Services

Os *web services* são soluções criadas para a comunicação de aplicações distintas. Basicamente são criadas funções, que podem ser acessadas remotamente, estas recebem os dados em formato XML, o processa e retorna uma resposta formatada em XML também.

6.1.1 - Utilizando um Web Service REST

O REST se tornou uma alternativa simples e viável para construções de *web services* na internet. Sua arquitetura é totalmente baseada em requisições HTTP.

Método HTTP	Ação CRUD	Descrição
POST	CREATE	Cria dados
GET	RETRIEVE	Recupera dados
PUT	UPDATE	Atualiza dados
DELETE	DELETE	Apaga dados

Tabela 18 - Métodos HTTP utilizados por um web service REST

Para fazer requisições em um *web service* REST o CakePHP utiliza sua biblioteca HTTPSocket.

Serviços baseados na web são disponibilizados por meio de autenticação, existem diversas formas de criar uma autenticação, dentre elas uma forma bem difundida entre os serviços é o *BASIC AUTHENTICATION*.

A autenticação básica consiste em enviar o usuário e a senha no *header* de todas as requisições.

```
$url = "http://twitter.com/statuses/update.xml";
$header = array('auth' => array(
        'method' => 'Basic',
        'user' => 'usuarioDoTwitter',
        'pass' => 'senhaDoTwitter'
));
$mensagem = 'Nova mensagem no meu twitter';
$resposta = $this->Http->post($url,
        array('status' => $mensagem),
        $header);
```

Listagem 59 - Enviando o usuário e senha no *header* da requisição

O exemplo acima faz uma requisição a um serviço do site twitter, esta requisição cria uma nova mensagem no twitter do usuário usado na autenticação.

As requisições a um Web Service trazem uma resposta, geralmente XML, que podem ser tratadas com a biblioteca XML.

```
$url = "http://twitter.com/account/verify_credentials.xml";
$resposta =$this->Http->get($url,
        null,
        $header);

$xml = new XML($resposta);
$array = $xml->toArray();
/*
Array
(
    [User] => Array
        (
```

```
            [id] => 26864385
            [name] => Daniel Golgher
            [screen_name] => golgher
            [location] => Belo Horizonte - MG - Brasil
            ...
        )
)
*/
```

Listagem 60 – Tratando a resposta do *webservice*

Nota

Mais detalhes sobre a biblioteca XML podem ser encontrados no tópico 3.6.12 - XML do Capítulo 3 - Aperfeiçoando sua Aplicação.

6.1.2 – Criando um Web Service REST

Para disponibilizar alguma parte de sua aplicação como um *web service* o primeiro passo é criar uma rota no arquivo app/config/routes.php para identificar as requisições REST.

```
Router::mapResources('posts');
Router::parseExtensions();
```

Listagem 61 – Configuração da rota para o *web service*

O comando **mapResources**, configura uma série de rotas para o suporte ao REST na aplicação. A tabela abaixo demonstra as rotas criadas.

Método HTTP	URL	Controle e ação
GET	/posts	PostsController::index()
GET	/posts/1	PostsController::view(1)
POST	/posts	PostsController::add()
PUT	/posts/1	PostsController::edit(1)
DELETE	/posts/1	PostsController::delete(1)
POST	/posts/1	PostsController::edit(1)

Tabela 19 – Rotas criadas pelo mapResources

Após configurar as rotas, o próximo passo é criar os métodos em seu controller. No caso deste exemplo, o PostsControler.

```
class PostsController extends AppController {

        var $components = array('RequestHandler');

        function index() {
                $posts = $this->Post->find('all');
                $this->set(compact('posts'));
        }

        function view($id) {
                $post = $this->Post->findById($id);
                $this->set(compact('post'));
```

```
        }

        function edit($id) {
                $this->Post->id = $id;
                if ($this->Post->save($this->data)) {
                        $message = 'Salvo';
                } else {
                        $message = 'Erro';
                }
                $this->set(compact("message"));
        }

        function add() {
                $this->Post->create();
                if ($this->Post->save($this->data)) {
                        $message = 'Salvo';
                } else {
                        $message = 'Erro';
                }
                $this->set(compact("message"));
        }

        function delete($id) {
                if($this->Post->del($id)) {
                        $message = 'Apagado';
                } else {
                        $message = 'Erro';
                }
                $this->set(compact("message"));
        }
}
```

Listagem 62 - Métodos na classe de controle controllers/posts_controller.php

Ao adicionar **Router::parseExtensions()** no arquivo de rotas, você está informando ao CakePHP, que ele pode lidar com diversas extensões para tratar a requisição, no caso de uma requisição REST é esperado uma resposta XML.

Com a classe de controle pronta, devemos codificar a página da visão de cada ação. Este arquivo deverá ficar localizado em: app/views/post/xml/nome_da_acao.ctp.

```
<posts>
        <?php echo $xml->serialize($posts); ?>
</posts>
```

Listagem 63 - Página da visão app/views/posts/xml/index.ctp

Ao fazer uma requisição ao endereço: **/posts/index.xml** será exibida a seguinte resposta:

```
<posts>
        <post id="1" titulo="teste"
                texto="teste"
                created="2008-06-13"
                modified="2008-06-14">
        </post>
        <post id="2" titulo="teste"
                texto="teste"
```

```
                created="2008-06-15"
                modified="2008-06-15">
        </post>
</posts>
```

Listagem 64 - Resposta da requisição ao endereço `/posts/index.xml`

Importante

Quando utilizamos o RequestHandler e Route, o CakePHP é capaz de identificar requisições to tipo XML e tratar de forma a gerar automaticamente o array $this->data, para ser manipulado dentro da ação requisitada.
O exemplo abaixo demonstra como inserir um novo post:

```
function envia_post(){
        App::import('Core', array('Xml', 'HttpSocket'));
        $this->Http =& new HttpSocket();
        $post['Post']['title']='teste feito pelo REST';
        $post['Post']['body']='teste feito pelo REST';
//Cria o XML a partir do array de Post.
        $xml = new XML($post);
//Faz uma requisição do tipo POST.
        $resposta = $this->Http->post(
                "http://projetos/array_test/posts/add.xml",
                $xml,
                array('header'=>array('Content-Type'=>'text/xml')
        );
        $xml = new XML($resposta);
        $array = $xml->toArray();
        pr($array);
}
```

CUIDADO: ao utilizar o método $this->Http->post deve-se informar o Content-type, caso o Content-type não seja informado o RequestHandler não será capaz de transformar o XML no array $this->data.

O array $this->data é carregado automaticamente a partir do XML enviado.

```
function add() {
        $this->Post->create();
        if(!empty($this->data)){
                if ($this->Post->save($this->data)) {
                        $message = 'Salvo';
                } else {
                        $message = 'Erro';
                }
                $this->set(compact("message"));
        }
}
```

Listagem 65 - Método add

Caso seja necessário criar mais alguma rota para ser acessível por REST, basta seguir o exemplo abaixo:

```
Router::connect(
        "/:controller/:id",
        array("action" => "edit", "[method]" => "PUT"),
        array("id" => "[0-9+]")
)
```

Listagem 66 - Criação de uma nova rota no arquivo app/config/routes.php

6.2 - Data Sources

O *data source* é o repositório de dados da aplicação. Este repositório pode ser:

- Banco de dados
- Arquivos de computador
- Arquivo XML
- Webservice
- Ldap

A intenção de utilizar *data sources* é abstrair o acesso aos diversos tipos de repositórios, de forma que estes possam ser associados.

Por exemplo: Uma aplicação que autentica seus usuários em um servidor Ldap, logo a aplicação utiliza dois *data sources*, o banco de dados dos usuários da aplicação e o servidor LDAP que autentica os usuários.

Utilizando o CakePHP é possível criar um modelo para representar os usuários da aplicação (banco de dados) e outro modelo representando os usuários LDAP, é possível também, relacionar os dois modelos, de forma, que o *framework* recupere registros relacionados nos dois *data sources*.

6.2.1 - Utilizando um Data Source LDAP

O LDAP é um protocolo para atualizar e pesquisar diretórios rodando sobre TCP/IP. O LDAP é amplamente utilizado para centralizar serviço de autenticação.

Este tópico foi baseado no artigo: http://bakery.cakephp.org/articles/view/using-ldap-as-a-datasource-basic-find-example, do site bakery (http://bakery.cakephp.org).

A aplicação construída neste artigo tem dois modelos relacionados. O modelo User faz acesso a um banco de dados e o LDAPUser faz acesso a um servidor LDAP.

Para a construção da aplicação o primeiro passo é fazer o *download* da classe do *data source*.

```
O projeto responsável pelo desenvolvimeto é:
        -http://ldapsource.googlecode.com/
```

O arquivo ldap_source.php deve ser salvo em: `app/models/datasources/`.

O próximo passo é configurar o acesso ao servidor LDAP, para isto é necessário criar um novo array no arquivo `app/config/database.php`, conforme o código mostrado abaixo:

```
 var $ldap = array (
        'datasource' => 'ldap',
        'host' => 'localhost',
        'port' => 389,
        'basedn' => 'dc=example,dc=org',
        'login' => 'cn=developer,dc=example,dc=org',
        'password' => 'xxxx',
        'version' => 3
    );
```

Listagem 67 - Configuração do acesso ao servidor LDAP

Criar a classe de modelo que utiliza como data source o banco de dados da aplicação.

```
<?php
class User extends AppModel {
    var $name = 'User';

    var $belongsTo = array (
        'LdapUser' => array (
            'className' => 'LdapUser',
```

```
            'foreignKey' => 'username'
        )
    );
}
?>
```

Listagem 68 – Classe de modelo que utiliza como data source o banco de dados

Criar a classe de modelo que utiliza como data source o servidor LDAP.

```
<?php
class LdapUser extends AppModel {

    var $name = 'LdapUser';

    var $useDbConfig = 'ldap';  //Este atributo informa ao cake
                                //que deverá ser utilizado o array $ldap

    var $primaryKey = 'cn';     //Este atributo deve ser alterado para sua
aplicação

    var $useTable = 'ou=person'; //Este atributo deve ser alterado para sua
aplicação

    var $hasOne = array (
        'User' => array (
            'className' => 'User',
            'foreignKey'=> 'username'//Este atributo deve ser alterado para sua
aplicação
        )

    );
}
?>
```

Listagem 69 – Classe de modelo que utiliza como data source o servidor LDAP

Após a criação da classe de modelo a classe de controle deve ser criada.

```
<?php
class UsersController extends AppController {

    var $name = 'Users';

    function index() {
        $conditions = "id=1"; //Este atributo deve ser alterado para sua
necessidade
        $this->User->recursive = 1;

        $data = $this->User->find('first',array('conditions'=>$conditions));
        $this->set('data', $data);
    }
}
?>
```

Listagem 70 – Classe de controle de Usuarios

A variável $data terá o seguinte conteúdo:

```
Array
(
    [User] => Array
        (
            [id] => 1
            [username] => jean
            [password] => xxxxxxxxxxxxxxxxxxxxxxxxx
            [mail] => jean@example.org
            [created] => 2007-01-13 12:16:09
            [modified] => 2007-05-03 15:21:12
        )

    [LdapUser] => Array
        (
            [mail] => jean@example.org
            [objectclass] => Person
            [telephonenumber] => 0000
            [cn] => jean
        )
)
```

Listagem 71 – Conteúdo da variável `$data` após a procura por um Usuario

6.2.2 Criando um Data Source

Os Data Sources são classes que estendem a classe Data Source e devem ser gravadas em: `app/models/datasources/`.

A classe criada precisa implementar alguns métodos da api da classe mãe (DataSource). A implementação dos métodos contém as rotinas para salvar, recuperar, apagar e atualizar os dados no ambiente que o *data source* utiliza.

A seguir segue o exemplo de um pequeno data source utilizado para adicionar mensagens no twitter.

```
App::import('Core', array('Xml', 'HttpSocket'));
class TwitterSource extends DataSource
{
        var $username = "";
        var $password = "";
        var $description = "Twitter API";
        var $Http = null;
        var $alias = null;
        function __construct($config) {
                $config['database']=false;
                parent::__construct($config);
                $this->Http =& new HttpSocket();
                $this->username = $this->config['username'];
                $this->password = $this->config['password'];
        }
        function __destruct() {
                parent :: __destruct();
        }
//Sempre retorna verdadeiro, pois a API do Twitter é baseada em BASIC
AUTHENTICATION
        function connect() {
```

```
                return true;
        }
//Método responsável em salvar dados novos no data source
        function create(&$model, $fields = null, $values = null) {
                $id = null;
                if ($fields == null) {
                        unset($fields, $values);
                        $fields = array_keys($model->data);
                        $values = array_values($model->data);
                }
                $status = $values[0];
                $url = "http://twitter.com/statuses/update.xml";
                $resposta = $this->Http->post($url,
                        array('status' => $status),
                        $this->__getAuthHeader());
                $xml = new XML($resposta);
                $array = $xml->toArray();
                return isset($array['Status']['id']);
        }
        function __getAuthHeader() {
                return array('auth' => array('method' => 'Basic',
                        'user' => $this->username,
                        'pass' => $this->password));
        }
}
```

Listagem 72 - Data Source do twitter app/models/datasources/twitter_source.php

Criado o *data source* do Twitter é preciso configurar a aplicação para que este *data source* esteja disponível para as classes de modelo.

```
var $twitter = array(
        'datasource' => 'twiter',
        'username' => 'usuarioDoTwitter',
        'password' => 'senhaDoTwitter',
);
```

Listagem 73 - Configuração do data source no arquivo de configuração app/config/database.php

Criando uma classe de modelo para utilizar o *data source* do Twitter.

```
class Message extends AppModel {

        var $name = 'Message';
        var $useTable=false;
//indica qual data source deve ser utilizado.
        var $useDbConfig = 'twitter';
//como não está sendo utilizado banco, o array descreve os campos
        var $_schema = array('mensagem' => array(
                'type'=>'string',
                'null' => true,
                'default' => NULL,
                'length' => 140)
        );
        var $validate=array('mensagem'=>
                array('rule'=>
                        array('between',1,140),
```

```
                        'message'=>'O campo deve ter de 1 a 140 caracteres.'
                )
        );
}
```

Listagem 74 - Classe de modelo app/model/message.php

A classe de controle não tem nenhuma alteração.

```
class MessagesController extends AppController {
        var $name = 'Messages';
        function add(){
                if (!empty($this->data)) {
                        if ($this->Message->save($this->data)) {
                                $this->Session->setFlash(__('Dado salvo!',
true));
                                $this->redirect(array('action'=>'index'));
                        } else {
                                $this->Session->setFlash(__('Verifique os
erros.', true));
                        }
                }
        }
}
```

Listagem 75 - Classe de controle app/controller/messages_controller.php

A camada de visão também não tem nenhuma alteração.

```
<div class="channels form">
<?php echo $form->create('Message');?>
        <fieldset>
                <legend><?php __('Add Message');?></legend>
        <?php echo $form->input('mensagem'); ?>
<?php echo $form->end('Submit');?>
        </fieldset>
</div>
```

Listagem 76 - Página da visão app/view/messages/add.ctp

7 - Customizando Componentes

7.1 - Behaviors

7.1.1 - Criando Behaviors

Para criar um Behaivor é necessário criar um arquivo PHP no caminho: **/app/models/behaviors/**. Este arquivo será uma classe que herdará da classe ModelBehavior.

Por exemplo: Deseja-se criar um behavior que altere a data do padrão DD/MM/YYYY para YYYY-MM-DD, antes de salvar a data no banco de dados. Este behavior chamará date_formatter, logo será criado o arquivo: **/app/models/behaviors/date_formatter.php**, com o conteúdo:

```
// app/models/behaviors/date_formater.php
<?php
class DateFormatterBehavior extends ModelBehavior {
        function setup(&$model) {
```

```
                $this->model = $model;
        }
        function alteraDataParaOBanco($dados){
                $campos = $this->model->getColumnTypes();
                foreach($dados[$this->model->name] as $chave=>$valor){
                        foreach($valor as $nomeDoCampo=>valorDoCampo){
                                if($campos[$nomeDoCampo] == 'date'){
                                        list($dia,$mes,$ano) =
split('/',$valorDoCampo);
                                        $novaData = $ano."-".$mes."-".$dia;
                                        $modelo=$this->model->name;
                                        $dados[$modelo][$chave]
[$nomeDoCampo]=$novaData;
                                }
                        }
                }
                return $dados;
        }
        function alteraDataParaOUsuario($dados){
          $campos = $this->model->getColumnTypes();
          foreach($dados as $chave=>$valor){
           foreach($valor as $model=>$campo){
             foreach($campo as $nomeDoCampo=>$valorDoCampo){
               if($campos[$nomeDoCampo] == 'date'){
                 list($ano,$mes,$dia)=split('-',$valorDoCampo);
                                $novaData=$dia."/".$mes."/".$ano;
                                $dados[$chave][$model][$nomeDoCampo]=$novaData;
               }
              }
            }
          }
          return $dados;
        }
//Funcão chamada antes de fazer a validação
    function beforeValidate($model) {
        $model->data = $this->alteraDataParaOBanco($model->data);
        return true;
    }
//Função chamada sempre depois de fazer uma busca ao banco de dados
    function afterFind(&$model, $results){
        $results = $this->alteraDataParaOUsuario($results, 2);
        return $results;
    }
}
?>
```

Para utilizar este Behavior criado pelo próprio desenvolvedor, basta adicioná-lo ao array de **$actsAs** da classe de modelo desejada.

```
// app/models/post.php
class Post extends AppModel {
        var $name = 'Post';
        var $actsAs = array('DateFormatter');
}
```

7.2 - Components

Para criar um Component é necessário criar um arquivo PHP no caminho: **/app/controllers/components/**. Este arquivo será uma classe que herdará da classe Object.

Por exemplo: Deseja-se criar um Component que envia e-mails utilizando a classe phpmailer. Este Component chamará mail, portanto será criado o arquivo: **/app/controllers/components/mail.php**.

Para este componente será necessário fazer o download da classe phpmailer no endereço:

```
http://phpmailer.codeworxtech.com/index.php?pg=sf&p=dl
```

A biblioteca phpmailer deve ser descompactada e a pasta phpmailer deve ser copiada para o diretório **vendors** da aplicação, a estrutura de diretórios deve ficar assim:

```
app/vendors/phpmailer/
app/vendors/phpmailer/class.phpmailer.php
```

Com a classe disponível para a aplicação podemos codificar o componente:

```
// app/controllers/components/mail.php
class MailComponent extends Object {

        function send_mail($to, $body, $subject,$fromName="Contato",
                        $attachment=null,$nomeDoArquivo=null){

//Importa a biblioteca phpmailer
                App::import('Vendor', 'phpmailer',
                        array('file' => 'phpmailer'.DS.'class.phpmailer.php'));
//Cria o objeto de phpmailer
                $mail = new PHPMailer();
//Configura o objeto do phpmailer
                $mail->IsHTML(true); // envia como HTML se 'true'
                $mail->WordWrap = 50; // Definição de quebra de linha
                $mail->Mailer = "smtp"; //Usando protocolo SMTP
                $mail->Host = "smtp.exemplo.com.br"; //seu servidor SMTP
                $mail->Username = "usuario"; //usuário do SMTP
                $mail->Password = "senha"; // senha do SMTP
//Insere anexo
                if(!empty($attachment)){
                        if(!$mail->AddAttachment($attachment,$nomeDoArquivo)){
                                return false;
                        }
                }
//Dados do e-mail que será enviado
                $mail->FromName = $fromName;
                $mail->AddAddress($to);
                $mail->Body = $body;
                $mail->Subject = $subject;
                return $mail->send();
        }
}
```

Para utilizar este Component criado pelo próprio desenvolvedor, basta adicioná-lo ao array de **$components** da classe de controle desejada. A partir dai os métodos do componente ficam disponíveis para utilização na classe de controle.

```
// app/controllers/fale_controller.php
class FaleController extends AppController {
        var $name = 'Fale';
        var $helpers = array('Html', 'Form' );
        var $components = array('Mail');

        function conosco() {
                if (!empty($this->data)) {
                        $message='Nome: '.$this->data['Fale']['nome'];
                        $message.='Empresa: '.$this->data['Fale']['empresa'];
                        $message.='Email: '.$this->data['Fale']['email'];
                        $message.='Endereco: '.$this->data['Fale']['endereco'];
                        $message.='Bairro: '.$this->data['Fale']['bairro'];
                        $message.='Cidade: '.$this->data['Fale']['cidade'];
                        $message.='Estado: '.$this->data['Fale']['uf'];
                        $message.='Telefone: '.$this->data['Fale']['ddd']."-".
                                $this->data['Fale']['tel'];
                        $message.='Mensagem: '.nl2br($this->data['Fale']
['mensagem']);

if($this->Mail->send_mail('site@exemplo.com.br',
                                        $message, 'FALE CONOSCO')){
                                $this->Session->setFlash('Dados enviados com
sucesso!');
                                $this->redirect('/fale/conosco/');
                        }
                        else{
                                $this->Session->setFlash('Falha ao enviar
email.');
                        }
                }
        }
}
```

7.3 - Helpers

Para criar um Helper é necessário criar um arquivo PHP no caminho: **/app/views/helpers/**. Este arquivo será uma classe que herdará da classe AppHelper.

Por exemplo: Deseja-se criar um Helper que gera imagens de mapas usando o *google maps*. Este Helper chamará maps, portanto será criado o arquivo: **/app/views/helpers/maps.php**.

Para este componente será necessário fazer o download da classe Google Maps API no endereço:

```
http://www.phpinsider.com/php/code/GoogleMapAPI/
```

A biblioteca Google Maps deve ser descompactada e a pasta GoogleMapAPI-2.5 deve ser copiada para o diretório **vendors** da aplicação, a estrutura de diretórios deve ficar assim:

```
app/vendors/GoogleMapAPI-2.5/
app/vendors/GoogleMapAPI-2.5/GoogleMapAPI.class.php
```

Importante

A API do google maps exige que seja informado uma chave, esta chave pode ser gerada

no endereço: http://maps.google.com/apis/maps/signup.html

Com a classe disponível para a aplicação podemos codificar o Helper:

```
// app/views/helpers/maps.php
class MapsHelper extends AppHelper
{
    var $helpers = array('Html');
    function displaymap($localizacoes=false,$width=500,$height=500)
    {
                App::import('Vendor', 'GoogleMapAPI',
                        array('file' => 'GoogleMapAPI-
2.5'.DS.'GoogleMapAPI.class.php'));

                $map = new GoogleMapAPI('map');
                if($localizacoes){
                        foreach($localizacoes as $localizacao){
                                $map->addMarkerByAddress(
$localizacao['endereco'],
                                        strip_tags($localizacao['titulo']),
                                        $localizacao['titulo']);
                        }
                }
                else{
//Centraliza o mapa no Brasil
                        $map->setCenterCoords(-45.815731,-33.954223);
                        $map->setZoomLevel(3);
                }

                $map->setWidth($width);
                $map->setHeight($height);
                $map_content=$map->getHeaderJS().$map->getMapJS().$map->getMap();
                return $this->output($map_content);
    }
}
```

Para utilizar este Helper criado pelo próprio desenvolvedor, basta adicioná-lo ao array de **$helpers** da classe de controle desejada. A partir dai os métodos deste Helper ficam disponíveis para utilização na camada de visão.

```
// app/controllers/maps_controller.php
class MapsController extends AppController {

        var $name = 'Maps';
        var $uses = null;
        var $helpers=array('Maps');

        function index(){
                $local=array();
                $local[1]['endereco']='Comendador Viana 720 Belo Horizonte Minas
Gerais Brasil';
                $local[1]['titulo']='2km';

                $local[2]['endereco']='Rua Penambuco 100 Belo Horizonte Minas
```

```
Gerais Brasil';
                $local[2]['titulo']='Savassi';
                $this->set(compact('local'));
        }
}
```

No arquivo index.ctp (camada de visão) é possível acessar o helper.

```
// app/views/maps/index.ctp
<?php
echo $maps->displaymap($local,500,500);
?>
```

8 - Ajax no CakePHP

8.1 - O Helper Js

O CakePHP possibilita a implementação de recursos Ajax nas aplicações de forma prática. Para poder implementar estes recursos e variados efeitos. Nesta versão, o framework dá suporte às bibliotecas jQuery, Mootools e Prototype/Scriptaculous. A biblioteca padrão é a jQuery.

Para utilizar os recursos da biblioteca é preciso colocá-la no diretório app/webroot/js. Além disso, é necessário importar essa biblioteca em todas as classes de visão que vão utilizar recursos Ajax. Por fim, é necessário incluir o *helper* Js na classe de controle. A seguir temos os exemplos de como importar a biblioteca nos arquivos da camada de visão e setar o *helper* na classe de controle:

```
//Na página da visão:
echo $this->Html->script('jquery');

//Na classe de controle:
var $helpers = array('Js');
```

Listagem 77 - Código para utilizar o helper `JsHelper`

Pode-se também utilizar o componente RequestHandler, neste caso sempre será renderizado o *layout* Ajax.

```
//Na classe de controle:
var $components = array( 'RequestHandler' );
```

Listagem 78 - Código para utilizar o componente `RequestHandler`

8.1.1 - Utilizando o mecanismo padrão de Javascript (jQuery)

o *JsHelper* fornece alguns métodos e atua como uma fachada para o mecanismo padrão. É recomendável que você utilize as funcionalidades de fachada do JsHelper. Elas permitem que você utilize os recursos de *buffering* e encadeamento (os métodos de encadeamento funcionam apenas no PHP5).

8.1.2 - Trabalhando com scripts no *buffer*

Um inconveniente na implementação anterior das funcionalidades de 'Ajax' era o espalhamento de tags `<script>` pelo documento e a impossibilidade de fazer o *buffer* de scripts adicionados por elementos no layout. O novo `JsHelper`, se utilizado de forma correta evita ambos os problemas. É recomendável que você coloque o `$this->Js->writeBuffer()` no final do seu arquivo de layout e acima da tag `</body>`. Isto irá permitir que todos os scripts gerados nos elementos do layout sejam escritos no mesmo lugar.

É importante ressaltar que os scripts em *buffer* são tratados separadamente dos arquivos de script que são incluídos.

O método `writeBuffer()` possui algumas opções:

- **inline** – Defina como `true` para ter os scripts escritos como um bloco inline. Caso a opção `cache` também for setada como `true`, uma tag de script com link será gerada. (padrão `true`)
- **cache** – Defina como `true` para fazer o cache dos scripts em um arquivo e fazer com que seja criado um link para o arquivo de cache (padrão `false`)
- **clear** – Defina como `false` para evitar que o cache do script seja limpado (padrão `true`)
- **onDomReady** – envolve os scripts em *cache* no evento *domready* (padrão `true`)
- **safe** – caso um bloco inline seja gerado ele será envolvido em `<![CDATA[ ... ]]>` (padrão `true`)

Importante

Para criar um arquivo de cache com o `writeBuffer()` é necessário que o diretório `webroot/js` tenha permissões de escrita global, permitindo assim que o browser faça o cache do script gerado por qualquer página.

8.1.3 – Métodos

8.1.3.1 – link

link($title, $url = null, $options = array())

Retorna um link para uma ação remota definida em $url. Essa url é acessada utlizando o XMLHttpRequest quando o link é clicado. O resultado gerado pode ser então inserido no objeto DOM definido em $options['update'].

Opções:

- **confirm** – Gera uma caixa de diálogo confirm() antes de enviar o evento.
- **id** – utiliza um `id` customizado.
- **htmlAttributes** – atributos HTML não convencionais. Atributos convencionais são `class`, `id`, `rel`, `title`, `escape`, `onblur` e `onfocus`.
- **buffer** – Se `false` desabilita o *buffer* e retorna a tag `script` além do link.

```
<div id="resposta"></div>

<?php echo $this->Js->link('Método link',
        array(
                'controller'=>'bookmarks',
                'action'=>'link_ajax'
        ),
        array(
                'update' => '#resposta'
        )
);
?>
```

Listagem 79 – Exemplo de uso do método `link`

É possível também utilizar a opção `confirm` para gerar uma caixa de diálogo javascript de confirmação. A requisição só ocorrerá se esse pop up retornar true. Na chave `confirm` deve-se colocar o texto que irá aparecer na janela.

```
<div id="resposta"></div>

<?php echo $this->Js->link('Método link',
        array(
                'controller'=>'bookmarks',
                'action'=>'link_ajax'
        ),
        array(
                'confirm'=>'Tem certeza?',
                'update' => '#resposta'
        )
);
?>
```

Listagem 80 – Exemplo de uso do método `link` com a opção `confirm`

8.1.3.2 – effect

effect($name, $options = array())

Cria um efeito básico. Por padrão não é feito o *buffer* deste método e ele retorna o seu resultado.

Os seguintes efeitos são suportados por todas os mecanimos de Js:

- **show** – mostra o elemento.
- **hide** – esconde o elemento.
- **fadeIn** – mostra o elemento com *Fade*.
- **fadeOut** – esconde o elemento com *Fade*.
- **slideIn** – mostra o elemento com *Slide*.
- **slideOut** – esconde o elemento com *Slide*.

Opções

- **speed** – Velocidade com que a animação deve ocorrer. Valores válidos são: `'slow'`, `'fast'`. Nem todos dos efeitos utilizam a opção `speed`.

8.1.3.3 – object

object($data, $options = array())

Converte valores em JSON. Existem algumas diferenças entre este método e o `JavascriptHelper::object()`. A principal diferença é que este método não pode criar tags `script`.

Opções:

- **prefix** – Prefixo (string) adicionado aos dados retornados.
- **postfix** – Sufixo (string) adicionado aos dados retornados.

8.1.3.4 – submit

submit(string $title, array $options)

Renderiza um botão do tipo submit que ao clicado enviará os dados via XMLHttpRequest. As opções incluem `FormHelper::submit()`, `JsBaseEngine::request()` e `JsBaseEngine::event();`

```
echo $this->Js->submit('Salvar', array('update' => '#resposta'));
```

Listagem 81 – Exemplo de uso do método `submit()`

8.1.3.5 – Drag & Drop

drag($options=array())

Torna um elemento arrastável na tela, especificado no parâmetro $id. Veja a seguir algumas opções que podem ser utilizadas:

Opção	Descrição
handle	Define quando um objeto será arrastável por um *handle* incorporado. O valor deve ser uma referência a um elemento, o id deste elemento, ou uma string que referencia uma classe CSS. O primeiro filho, neto, etc do elemento encontrado que tiver essa classe CSS será usado como o *handle*.
snapGrid	O grid de pixels que o movimento estará restrito, um array(x, y).
container	O elemento que atua como uma caixa de contorno para o elemento arrastável.

Tabela 20 - Opções do método drag()

Confira as opções para o evento:

Opção	Descrição
start	Evento disparado quando o a ação inicia.
drag	Evento disparado em cada passo da ação.
stop	Evento disparado quando a ação para (mouse é liberado).

Tabela 21 - Opções de evento do método drag()

```
$this->Js->get('#elemento');
$this->Js->drag(array(
    'container' => '#conteudo',
    'start' => 'onStart',
    'drag' => 'onDrag',
    'stop' => 'onStop',
    'snapGrid' => array(10, 10),
    'wrapCallbacks' => false
));
```

Listagem 82 - Exemplo do uso do método drag()

drop($options = array())

Faz o elemento poder receber elementos que foram arrastados para ele. Veja a seguir algumas opções que podem ser utilizadas no parâmetro $options:

Opção	Descrição
accept	Pode conter uma string ou um array javascript contendo strings contendo as classes CSS que o elemento irá aceitar como elementos arrastáveis a ele.
hoverclass	Classe a ser adicionada ao *droppable* quando um *draggable* está sobre ele.

Tabela 22 - Exemplo do uso do método drop()

Confira as opções para o evento:

Opção	Descrição
drop	Evento disparado quando um elemento é 'solto' na zona de *drop*.
hover	Evento disparado quando um *drag* entra na zona de *drop*.
leave	Evento disparado quando um *drag* é removido da zona *drop* sem ser 'solto'.

Tabela 23 - Opções de evento do método drop()

```
$this->Js->get('#elemento');
$this->Js->drop(array(
    'accept' => '.itens',
    'hover' => 'onHover',
    'leave' => 'onExit',
    'drop' => 'onDrop',
    'wrapCallbacks' => false
));
```

8.1.3.6 - slider

slider($options=array())

Cria um slider de controle. Pode-se usar as seguintes opções no parâmetro $options:

Opção	Descrição
$options['axis']	Define a direção que o slider irá se mover. Pode ser 'horizontal' ou 'vertical'.
$options['handleImage']	O id da imagem que será arrastável no slider. É usado para a imagem (src) quando está habilitada. É usado em conjunto com handleDisabled.
$options['increment']	Define a relação entre pixels e valores. Se colocar 1, para cada pixel que o slider mover vai ser atribuído o valor 1.
$options['handleDisabled']	O id da imagem (src) quando está desabilitada. É usado em conjunto com handleImage.
$options['change'] $options['onChange']	Callback em javascript disparado quando o slider para de mover. O método recebe o valor do slider como parâmetro.
$options['slide'] $options['onSlide']	Callback em javascript disparado quando o slider se move. O método recebe o valor do slider como parâmetro.

8.1.3.7 - sortable

sortable($options = array())

Faz uma lista ou um grupo de objetos flutuantes ordenáveis. Veja a seguir algumas opções que podem ser usadas no array `$options`:

Opção	Descrição
containment	O container para a ação de mover.
handle	Seletor do elemento *handle*. Somente este elemento irá iniciar a ação de ordernar.
revert	Define se irá utilizar ou não um efeito para mover o elemento para a posição final.
opacity	Opacidade do *placeholder*.
distance	Distância que o *sortable* deve ser arrastado antes que a ordenação seja iniciada.

9 - Ajax no CakePHP com a jQuery

A jQuery é uma biblioteca javascript simples e rápida que provê funcionalidades para trabalhar com eventos, animações, arrastar e soltar (*drag and drop*), manipulação do DOM, AJAX e muito mais. Ela é um *software* livre (está sob a licença MIT) e dá suporte a praticamente todos os navegadores atuais.

Mais informações sobre a biblioteca podem ser encontradas no seu site oficial: `http://www.jquery.com`. Se você não tem nenhum conhecimento sobre esta excelente biblioteca você pode começar com o tutorial oficial traduzido para o português do Brasil em `http://i18n.2kminterativa.com.br/jquery/jquery-getting-started-pt_br.html`.

Para utilizar os recursos de AJAX da jQuery no CakePHP é muito simples, vamos aos passos.

Importante

O *helper* AJAX padrão do CakePHP utiliza as bibliotecas `prototype` e `scriptaculous`. Para utilizar a bibliteca `jQuery` em conjunto com as bibliotecas padrão é preciso definir o modo `noConflict()` da `jQuery`.

9.1 - Fazendo o download do helper

Primeiramente é preciso fazer o download da biblioteca jQuery, do plugin form (para envio do formulário via AJAX) e do helper para a jQuery. Todos estes arquivos podem ser baixados no endereço:

```
http://cakephp.2km.com.br/helpers/jquery.zip
```

Após o download vamos descompactar o arquivo e a seguinte estrutura deverá ser criada:

```
/vendors/js/jquery/jquery.js
/vendors/js/jquery/jquery.form.js
/vendors/js/jquery/jquery.jeditable.mini.js
/views/helpers/ajax.php
```

9.2 - Configurando o helper

Vamos copiar o conteúdo da pasta *vendors* para a pasta *vendors* da nossa aplicação. O conteúdo da pasta ficará assim:

```
/projetos/bookmarks/vendors/js/jquery/jquery.js
/projetos/bookmarks/vendors/js/jquery/jquery.form.js
/projetos/bookmarks/vendors/js/jquery/jquery.jeditable.mini.js
```

Depois disso, vamos copiar o arquivo ajax.php para a pasta */projetos/bookmarks/views/helpers*. O conteúdo da pasta ficará assim:

```
/projetos/bookmarks/views/helpers/ajax.php
```

Após a cópia dos arquivos vamos configurar no arquivo app_controller.php para utilizar o helper Ajax.

```
<?php
class AppController extends Controller {

    var $helpers = array('Ajax');
    ...
?>
```

Listagem 83 - Configuração do helper Ajax no app_controller.php

9.2.1 - Link AJAX

Depois de configurado o helper, você poderá utilizá-lo em qualquer página da visão. Vamos ver um exemplo da utilização do helper para criar um link AJAX:

```
<?php
 echo $ajax->link('Link AJAX', '/bookmarks/link_ajax', array(
        'update' => 'resposta_ajax'
  ));
    ...
?>
```

Listagem 84 - Utilização do helper Ajax na camada da visão para criar um link

No exemplo mostrado anteriormente, será feita uma requisição para a URL /bookmarks/link_ajax e a sua resposta será impressa no elemento que possui o ID "resposta_ajax".

9.2.2 - Envio de formulário por AJAX

Vamos para um exemplo do envio de um formulário utilizando o helper AJAX

```
<?php
        echo $ajax->form('Bookmarks', 'post', array('model' => 'Bookmarks',
'url' => '/bookmarks/post_test', 'update' => 'resposta_ajax'));
        echo $form->input('Bookmark.hash', array('id' => 'test_observe'));
        echo $form->end('Enviar');
?>
```

No exemplo acima o formulário será submetido via POST para a *URL* /bookmarks/post_via_ajax juntamente com o parâmetro hash. A resposta será impressa no elemento que possui o ID "resposta_ajax".

10 - Testes

O CakePHP, versão 1.3, fornece um sistema abrangente de teste. O CakePHP possui um *framework* de teste que é uma extensão do *framework* SimpleTest para PHP. Nesse módulo iremos ver como preparar os testes e depois como fazer e rodar esses testes.

10.1 - Preparando o ambiente

10.1.1 - Instalando o SimpleTest

O *framework* de teste fornecido pelo CakePHP 1.3 é construído utilizando o SimpleTest testing *framework*. Esse *framework* não vem instalado junto ao CakePHP, por isso é preciso instalá-lo antes de qualquer coisa. Para baixar o *framework* basta entrar no endereço http://simpletest.sourceforge.net/.

Puxe a última versão, descompacte o arquivo e coloque na pasta *cake/vendors* ou app/vendors. Agora temos o diretório vendors/simpletest com todos arquivos e pastas do *framework* dentro. Antes de rodar qualquer teste, lembre-se de ter o nível de DEBUG em pelo menos 1 no arquivo app/config/core.php.

Se não existe nenhum teste de conexão com o banco de dados no arquivo app/config/database.php, os testes das tabelas serão criados com test_suite_ prefix. Você pode criar $test para testar a conexão com o banco de dados da seguinte forma:

```
var $test = array(
        'driver' => 'mysql',
        'persistent' => false,
        'host' => 'teste',
        'login' => 'teste',
        'password' => '12345',
        'database' => 'teste'
);
```

10.1.2 - Executando o teste de casos Core

O CakePHP 1.2 fornece diversos cases que envolvem a funcionalidade Core. Esses testes podem ser acessados através da url `http://seudominio/pasta_do_cake/test.php`. Tente executar um dos testes clicando nos respectivos links. A execução de teste de grupo pode durar um tempo, mas no fim você deverá ver a mensagem "2/2 test cases complete: 49 passes, 0 fails and 0 exceptions.".

10.2 - Preparando os dados

O *framework* de teste do CakePHP oferece dois tipos de testes. O primeiro, denominado Unit Testing, irá testar pequenas partes do seu código, que pode ser um método de um componente ou uma ação de uma classe de controle. O outro, denominado Web Testing, é uma forma de automatizar seus testes através da navegação de páginas, preenchendo formulários, clicando em links, e assim por diante.

10.2.1 - Sobre *fixtures*

Quando testando códigos que dependem de classes de modelos e dados, pode-se utilizar *fixtures* para gerar dados temporários nas tabelas. O benefício de utilizar *fixtures* é que não haverá como distorcer dados que estejam sendo utilizados em uma aplicação real.

Por padrão, o CakePHP irá tentar estabelecer uma conexão com o banco de dados definido no array $tests do arquivo app/config/database.php. Se essa conexão não pode ser estabelecida, ele irá estabelecer a conexão $default e criar os testes das tabelas do banco de dados definidos nessa configuração. Nos dois casos será criado "test_suite_" para evitar colisão com tabelas existentes.

Durante o processo de teste com *fixtures*, o CakePHP irá fazer os seguintes processos:

1. Criar tabelas para cada *fixture* necessária
2. Criar tabelas com dados, se os dados forem fornecidos em *fixtures*

3. Rodar os métodos de testes
4. Esvaziar as tabelas de *fixtures*
5. Remover as tabelas de *fixtures* do banco de dados

10.2.2 - Criando *fixtures*

Ao criar um fixture será necessário definir como as tabelas são criadas - quais campos a tabela possui - e quais registros serão utilizados na tabela de teste. No exemplo a seguir iremos criar o teste da classe de modelo Artigo. Para isso, será necessário criar o arquivo artigo_fixture.php dentro da pasta app/tests/*fixtures*. Este arquivo deverá ter o seguinte código:

```
<?php

class ArtigoFixture extends CakeTestFixture {
        var $name = 'Artigo';
        var $fields = array(
                'id' => array('type' => 'integer', 'key' =>
'primary'),
                'titulo' => array('type' => 'string', 'length' => 255, 'null' =>
false),
                'texto' => 'text',
                'publicado' => array('type' => 'integer', 'default' => '0',
'null' => false),
                'criado' => 'datetime',
                'atualizado' => 'datetime'
        );

        var $records = array(
                array (
                        'id' => 1,
                        'titulo' => 'Artigo de teste',
                        'texto' => 'Aqui vem o texto completo do artigo.',
                        'publicado' => '1',
                        'criado' => '2009-04-16 18:19:45',
                        'atualizado' => '2009-04-18 10:41:31'
                ),

                array (
                        'id' => 2,
                        'titulo' => 'Artigo de teste2',
                        'texto' => 'Aqui vem o texto completo do artigo.',
                        'publicado' => '1',
                        'criado' => '2008-10-08 10:41:23',
                        'atualizado' => '2008-11-20 09:23:21
                ),

                array (
                        'id' => 3,
                        'titulo' => 'Artigo de teste3',
                        'texto' => 'Aqui vem o texto completo do artigo.',
                        'publicado' => '0',
                        'criado' => '2006-01-20 11:43:29',
                        'atualizado' => '2006-01-21 20:15:39'
                )
        );
}
```

```
?>
```

É usado o array $fields para indicar quais campos fazem parte dessa tabela e como esses campos estão definidos. O formato usado para definir esses campos é o mesmo usado em generateColumnSchema() na classe de engenharia de banco de dados do CakePHP. A seguir temos uma listagem dos atributos que os campos em $fields podem receber e seus respectivos significados:

Atributo	Descrição
type	Tipo de dado interno do CakePHP. Pode ser string (VARCHAR no banco de dados), text (TEXT), integer (INT), float (FLOAT), datetime (DATETIME), timestamp (TIMESTAMP), time (TIME), date (DATE) e binary (BLOB).
key	Define a chave primária da tabela, faz o campo com AUTO_INCREMENT e PRIMARY KEY.
length	O tamanho do campo. O número de caracteres que pode receber.
null	Se for *true* permite que o campo seja null, se for *false* o null não é permitido.
default	O valor padrão que o campo recebe.

10.2.3 - Importando informações de tabelas e seus registros

Muitas vezes a aplicação pode já ter classes de modelos com dados associados a eles. E, nestas situações, pode-se querer testar essas classes de modelos com seus dados. Ter que recriar as tabelas e os dados no arquivo de fixture seria muito trabalhoso. Existe um jeito onde se pode pegar as definições dessa tabela e seus dados da classe de modelo e enviar para uma fixture particular.

Vamos supor que exista uma classe de modelo chamada Artigo. A princípio iremos alterar o arquivo app/tests/*fixtures*/artigo_fixture.php para:

```
<?php

class ArtigoFixture extends CakeTestFixture {
        var $name = 'Artigo';
        var $import = 'Artigo';
}

?>
```

O $import faz com que o fixture importe as definições da tabela da classe de modelo Artigo. Por padrão, os dados não são importados. Para que isso ocorra deve-se alterar o código da seguinte forma:

```
<?php

class ArtigoFixture extends CakeTestFixture {
        var $name = 'Artigo';
        var $import = array('model' => 'Artigo', 'records' => true);
}
```

```
?>
```

Existem casos onde se têm uma tabela criada mas não se têm uma classe de modelo. Neste caso, deve-se proceder da seguinte forma:

```
<?php

class ArtigoFixture extends CakeTestFixture {
        var $name = 'Artigo';
        var $import = array('table' => 'artigos');
}

?>
```

Neste caso, serão utilizadas as definições da tabela artigos, com a conexão padrão ('default') da aplicação. Se for necessário mudar a conexão de banco de dados, deve-se proceder da seguinte forma:

```
<?php

class ArtigoFixture extends CakeTestFixture {
        var $name = 'Artigo';
        var $import = array('table' => 'artigos', 'connection' => 'outra');
}

?>
```

Se houver algum prefixo em sua tabela, este será automaticamente usado. Para poder importar os dados da tabela, deve-se proceder da seguinte forma:

```
<?php

class ArtigoFixture extends CakeTestFixture {
        var $name = 'Artigo';
        var $import = array('table' => 'artigos', 'records' =>
true);
}

?>
```

Por fim, é possível importar as definições da tabela ou classe de modelo no *fixtures* e depois inserir dados manualmente:

```
<?php

class ArtigoFixture extends CakeTestFixture {

        var $name = 'Artigo';
        var $import = 'Artigo';
        var $records = array(
                array (
                        'id' => 1,
                        'titulo' => 'Artigo de teste',
                        'texto' => 'Aqui vem o texto completo do artigo.',
                        'publicado' => '1',
                        'criado' => '2009-04-16 18:19:45',
                        'atualizado' => '2009-04-18 10:41:31'
```

```
                ),

                array (
                        'id' => 2,
                        'titulo' => 'Artigo de teste2',
                        'texto' => 'Aqui vem o texto completo do artigo.',
                        'publicado' => '1',
                        'criado' => '2008-10-08 10:41:23',
                        'atualizado' => '2008-11-20 09:23:21
                ),

                array (
                        'id' => 3,
                        'titulo' => 'Artigo de teste3',
                        'texto' => 'Aqui vem o texto completo do artigo.',
                        'publicado' => '0',
                        'criado' => '2006-01-20 11:43:29',
                        'atualizado' => '2006-01-21 20:15:39'
                )
        );
}

?>
```

10.3 - Criando os testes

Antes de criar os testes, devemos ficar atentos às seguintes regras:

- Os arquivos php de teste devem ser salvos em app/tests/cases/[alguma_pasta]
- Os nomes desses arquivos devem terminar com .test.php no lugar de apenas .php
- As classes contendo testes devem herdar CakeTestCase ou CakeWebTestCase.
- O nome de qualquer método contendo um teste deve começar com test. Ex: testArtigo.

Tendo essas regras em mente, vamos partir para exemplos práticos.

10.4 - Testando as classes de modelos

10.4.1 - Criando um teste

Vamos partir do pressuposto que já temos a seguinte classe de modelo:

```
<?php

class Artigo extends AppModel {
        var $name = 'Artigo';
        function publicado($fields = null) {
                $conditions = array(
                        $this->name . '.publicado' => 1
                );
                return $this->findAll($conditions, $fields);
        }
}

?>
```

Para testar a classe de modelo de Artigo através de *fixtures*, vamos ao arquivo *fixtures* e primeiro lemos a classe de modelo Artigo. Depois definimos qual conexão de banco de dados será usada para fazer o teste. O CakePHP test suite permite que o nome da configuração de banco de dados seja test_suite, que serão utilizados em todas classes de modelos de *fixtures*. Usando $useDbConfig a essa configuração fará com que o CakePHP saiba que essa classe de modelo usa a conexão de banco de dados test suite.

Como nós queremos reutilizar a classe de modelo de Artigo existente, iremos criar uma classe de modelo de teste que herda a classe de modelo Artigo. Para isso, criamos o arquivo artigo.test.php, dentro da pasta app/tests/cases/models, com o seguinte código:

```
<?php

App::import('model','Artigo');
class ArtigoTestCase extends CakeTestCase {
        var $fixtures = array( 'app.artigo' );
}

?>
```

Importante

Na variável `$fixtures` definimos qual *fixtures* iremos utilizar. Se a classe de modelo possui associação com outras classes de modelos, será necessário todas as *fixtures* para cada classe de modelo associada. Mesmo se estes não forem utilizados.

10.4.2 - Criando um método de teste

Para criar um método que teste o método publicado() na classe de modelo Artigo, devemos alterar o arquivo app/tests/cases/models/artigo.test.php da seguinte maneira:

```
<?php

App::import('model', 'Artigo');

class ArtigoTestCase extends CakeTestCase {

        var $fixtures = array('app.artigo');

        function testPublicado() {
                $this->Artigo =& ClassRegistry::init('Artigo');
                $result = $this->Artigo->publicado(array('id', 'titulo'));
                $expected = array(
                        array('Artigo' => array( 'id' => 1, 'titulo' => 'Artigo
de teste' )),
                        array('Artigo' => array( 'id' => 2, 'titulo' => 'Artigo
de teste2' )),
                        array('Artigo' => array( 'id' => 3, 'titulo' => 'Artigo
de teste3' ))
                );
```

```
                $this->assertEqual($result, $expected);
        }
}

?>
```

O método testPublicado() primeiro cria uma instância da classe de modelo Artigo, depois executa a função publicado. No array $expected é gerado um array com os dados que se espera que a função publicado retorne. Para verificar se o resultado ocorreu como esperado é utilizado o método assertEqual().

10.5 - Testando as classes de controle

10.5.1 - Criando um teste

Continuando com o exemplo de Artigos, imagine a seguinte classe de controle:

```
<?php

class ArtigosController extends AppController {

        var $name = 'Artigos';
        var $helpers = array('Ajax', 'Form', 'Html');

        function index($short = null) {
                if (!empty($this->data)) {
                        $this->Artigo->save($this->data);
                }
                if (!empty($short)) {
                        $result = $this->Artigo->findAll(null, array('id',
'titulo'));
                } else {
                        $result = $this->Artigo->findAll();
                }
                if (isset($this->params['requested'])) {
                        return $result;
                }
                $this->set('titulo', 'Artigos');
                $this->set('artigos', $result);
        }
}

?>
```

Crie um arquivo artigos_controller.test.php na pasta your app/tests/cases/controllers, com o seguinte código:

```
<?php

class ArtigosControllerTest extends CakeTestCase {
```

```
function startCase() {
        echo '<h1>Começando o teste</h1>';
}

function endCase() {
        echo '<h1>Fim do teste</h1>';
}

function startTest($method) {
        echo '<h3>Começando o método ' . $method . '</h3>';
}

function endTest($method) {
        echo '<hr />';
}

function testIndex() {
        $result = $this->testAction('/artigos/index');
        debug($result);
}

function testIndexShort() {
        $result = $this->testAction('/artigos/index/short');
        debug($result);
}

function testIndexShortGetRenderedHtml() {
        $result = $this->testAction('/artigos/index/short',
                array('return' => 'render'));
                debug(htmlentities($result));
}

function testIndexShortGetViewVars() {
        $result = $this->testAction('/artigos/index/short',
                array('return' => 'vars'));
                debug($result);
}

function testIndexFixturized() {
        $result = $this->testAction('/artigos/index/short',
                array('fixturize' => true));
                debug($result);
}

function testIndexPostFixturized() {
        $data = array('Artigo' => array(
                'user_id' => 1,
                'publicado' => 1,
                'slug'=>'novo-artigo',
                'titulo' => 'Novo Artigo',
                'texto' => 'Novo corpo do artigo'
        ));
        $result = $this->testAction('/artigos/index',
                array('fixturize' => true, 'data' => $data, 'method' =>
```

```
'post'));
                        debug($result);
        }
}

?>
```

10.5.2 - O método testAction

O método testAction recebe como primeiro parâmetro a url da ação da classe de controle a ser testado. Ex: '/artigos/index/short'. O segundo parâmetro é um array que pode receber os seguintes índices:

Indice	Descrição
return	Define o que irá ser retornado. Os valores válidos são: • 'vars' - pega as variáveis da classe de visão após executar a ação; • 'view' - pega a classe de visão renderizada, sem o *layout*; • 'contents' - pega a classe de visão completa, com todo o *html* e *layout*; • 'result' - pega o valor retornado quando a ação utiliza $this->params['requested']. O padrão é 'result'.
fixturize	Se possuir o valor *true* as classes de modelos serão auto-fixturized. As tabelas da aplicação serão copiados com os seus registros. Se os dados forem alterados no teste isso não afetará os dados da aplicação real. Se *fixturize* possuir um array com as classes de modelos, apenas estas classes de modelos serão auto-fixturized.Se você quiser utilizar arquivos fixtures com testAction() não utilize *fixturize*, no lugar use *fixtures* como você normalmente iria utilizar.
method	Pode receber os valores 'get' ou 'post', caso seja necessário passar dados para a classe de controle.
data	Os dados a serem passados. Seu formato é de um array associativo do tipo campo => valor. No exemplo passado a função testIndexPostFixturized() exemplifica o uso de data.

10.5.3 - Pitfalls

Se for usado testAction para testar um método de uma classe de controle que utiliza redirect, o teste será finalizado imediatamente, sem gerar resultado algum.

Importante

Para que o teste não seja finalizado imediatamente, siga os seguintes passos:
No arquivo `test.php` é preciso definir a seguinte linha:

```
define('CAKEPHP_UNIT_TEST_EXECUTION',true);
```

e no arquivo `app_controller.php` é preciso redefinir o método redirect:

```
function redirect($url=NULL,$code=NULL){
       if(defined('CAKEPHP_UNIT_TEST_EXECUTION')){
           $this->set(compact('url'));
       } else {
           parent::redirect($url,$code);
       }
   }
```

10.6 - Testando Helpers

Como existe muita lógica que ocorre nas classes dos Helpers, é importante ter certeza que todas essas classes serão testadas. O procedimento de teste de Helpers é similar ao teste de componentes. Vamos supor que temos um helper chamado cambioHelper localizado em app/views/helpers/cambio.php. O arquivo de teste deve ficar localizado em app/tests/cases/helpers/cambio.test.php

10.6.1 - Criando teste de Helpers

Primeiro iremos definir as funcionalidades do helper cambio. Para efeito de demonstração este helper irá ter dois métodos:

function usd($amount)

Essa função irá receber o valor em $amount. E irá converter o valor para duas casas decimais e com o prefixo 'USD'.

function euro($amount)

Irá fazer o mesmo que a função usd, entretanto o pefixo será 'EUR'. Irá também inserir o resultado em um tag span:

```
<span class="euro"></span>
```

Para criar o teste, inserimos, no arquivo app/tests/cases/helpers/cambio.test.php, o seguinte código :

```
<?php

//Import the helper to be tested.
//If the tested helper were using some other helper, like Html,
//it should be impoorted in this line, and instantialized in startTest().

App::import('Helper', 'CurrencyRenderer');

class CurrencyRendererTest extends CakeTestCase {

        private $currencyRenderer = null;
        //Here we instantiate our helper, and all other helpers we need.

        public function startTest() {
                $this->currencyRenderer = new CurrencyRendererHelper();
        }

        //testing usd() function.
        public function testUsd() {
                $this->assertEqual('USD 5.30',
$this->currencyRenderer->usd(5.30));
```

```
                //We should always have 2 decimal digits.
                $this->assertEqual('USD 1.00', $this->currencyRenderer->usd(1));
                $this->assertEqual('USD 2.05',
$this->currencyRenderer->usd(2.05));

                //Testing the thousands separator
                $this->assertEqual('USD 12,000.70',
$this->currencyRenderer->usd(12000.70));
        }
?>
```

Aqui pode-se utilizar diferentes valores a serem testados em usd() e avaliar se os resultados são como o esperado. Se rodarmos o teste irá ocorrer uma falha, já que o Helper em si não existe ainda. Podemos então criar o Helper:

```
<?php

class CambioHelper extends AppHelper {
        public function usd($amount) {
                return 'USD ' . number_format($amount, 2, ',', '.');
        }
}

?>
```

Definimos com numer_format() que o valor passado pelo parâmetro $amount será convertido para duas casas decimais sendo estes separados por vírgula e os milhares por ponto. Além disso, o valor final conterá o prefixo 'USD'. Este arquivo deve ser salvo em app/views/helpers/cambio.php. Se rodarmos o teste agora ele irá funcionar.

10.7 - Testando componentes

Vamos testar um componente chamado TransporterComponent, que utiliza uma classe de modelo chamada Transporter que irá forncer funcionalidades para outras classes de controles. Serão necessários os seguintes arquivos:

- Um componente chamado Transporters localizado em app/controllers/components /transporter.php
- Uma classe de modelo chamada Transporter localizado em app/models/transporter.php
- Um fixture chamado TransporterTestFixture localizado em app/tests/fixtures /transporter_fixture.php
- O código de teste localizado em app/tests/cases/transporter.test.php

10.7.1 - Iniciando o componente

Como não se recomenda importar classes de modelos diretamente nos componentes será necessário uma classe de controle para acessar os dados da classe de modelo. Se a função startup() do componente for assim:

```
public function startup(&$controller){
        $this->Transporter = $controller->Transporter;
}
```

Então é possível desenvolver uma classe falsa:

```
class FakeTransporterController {}
```

E associar valores dentro dela, da seguinte forma:

```
$this->TransporterComponentTest = new TransporterComponent();
$controller = new FakeTransporterController();
$controller->Transporter = new TransporterTest();
$this->TransporterComponentTest->startup(&$controller);
```

10.7.2 - Criando o método de teste

Para criar um método de teste basta criar uma classe que herda a CakeTestCase e escrever os testes:

```
class TransporterTestCase extends CakeTestCase {

        var $fixtures = array('transporter');

        function testGetTransporter() {

                $this->TransporterComponentTest = new TransporterComponent();
                $controller = new FakeTransporterController();
                $controller->Transporter = new TransporterTest();
                $this->TransporterComponentTest->startup(&$controller);
                $result =
$this->TransporterComponentTest->getTransporter("12345", "Sweden", "54321",
"Sweden");
                $this->assertEqual($result, 1, "SP is best for 1xxxx-5xxxx");
                $result =
$this->TransporterComponentTest->getTransporter("41234", "Sweden", "44321",
"Sweden");
                $this->assertEqual($result, 2, "WSTS is best for 41xxx-44xxx");
                $result =
$this->TransporterComponentTest->getTransporter("41001", "Sweden", "41870",
"Sweden");
                $this->assertEqual($result, 3, "GL is best for 410xx-419xx");
                $result =
$this->TransporterComponentTest->getTransporter("12345", "Sweden", "54321",
"Norway");
                $this->assertEqual($result, 0, "Noone can service Norway");
        }
}
```

10.8 - Testando a classe de visão (Web Testing)

Na maioria das vezes, o CakePHP gera resultados em uma aplicação Web. Testes de unidades são eficientes para testar pequentas partes funcionais da aplicação, mas existem situações onde pode-se precisar fazer testes em maior escala. A classe CakeWebTestCase fornece uma forma de fazer este tipo de teste, pelo ponto de vista do usuário.

10.8.1 - Sobre o CakeWebTestCase

O CakeWebTestCase é uma extensão do SimpleTest WebTestCase, sem nenhuma funcionalidade extra. Isso quer dizer que nenhuma funcionalidade além das encontradas na SimpleTest está disponível. Portanto, não é possível utilizar fixtures. Isso implica que os testes envolvendo a atualização ou gravação de dados irá permanentemente alterar os dados armazenados no banco de dados. Os testes de resultados são baseados em quais dados o banco de dados tem armazenado. Então, fazer o banco de dados possuir dados é parte do procedimento de teste.

10.8.2 - Criando um teste

Os testes das classes de visão devem ser localizados no diretório tests/cases/views. Na prática, é possível colocar esses testes em qualquer lugar, mas é sempre recomendável seguir os padrões e ter uma organização lógica em suas aplicações.

Para criar um teste das visões, vamos criar um arquivo de teste no seguinte caminho: tests/cases/views /complete_web.test.php. Dentro deste arquivo deve-se declarar uma classe que herda a CakeWebTestCase, como no exemplo a seguir:

```
class CompleteWebTestCase extends CakeWebTestCase
```

Se for necessário algum preparo, pode-se criar um métdo construtor:

```
function CompleteWebTestCase(){
        //Códigos vêm aqui
}
```

Quando fazendo um teste, a primeira coisa a se fazer é conseguir alguma tela renderizada. Isso pode ser feito através de uma requisição *get* ou *post*, usando get() ou post(). Estes dois métodos pegam a url como primeiro parâmetro. Isto pode ser feito dinamicamente. Se partirmos do pressuposto que o código de teste está localizado em http://seu.dominio/cake/folder/webroot/test.php, podemos fazer da seguinte forma:

```
$this->baseurl = current(split("webroot", $_SERVER['PHP_SELF']));}
```

Depois, pode-se utilizar gets and posts, da seguinte forma:

```
$this->get($this->baseurl."/products/index/");
$this->post($this->baseurl."/customers/login", $data);
```

O segundo parâmetro do método post, $data, é um array associativo com dados post no formato do CakePHP.

```
"data[Usuario][nome]" => "José da Silva",
"data[Usuario][senha]" => "teste");
```

10.8.3 - Testando uma página

CakeWebTest fornece uma forma de navegar pela página através do click de links e imagens, preenchendo formulários e clicando em botões.

10.9 - Testando pluggins

Testes de plugins são cridos dentro de seus próprios diretórios, dentro da pasta plugin:

```
/app
        /plugins
                /pizza
                        /tests
                                /cases
                                /fixtures
                                /groups
```

Funcionam como testes normais, só é preciso ficar atento às convenções dos nomes quando for importar as classes. Veja o exemplo a seguir:

```
App::import('Model', 'Pizza.PizzaOrder');
```

```
class PizzaOrderCase extends CakeTestCase {
        // Plugin fixtures located in /app/plugins/pizza/tests/fixtures/
        var $fixtures = array('plugin.pizza.pizza_order');
        var $PizzaOrderTest;

        function testSomething() {
                // ClassRegistry makes the model use the test database connection
                $this->PizzaOrderTest =& ClassRegistry::init('PizzaOrder');
                // do some useful test here
                $this->assertTrue(is_object($this->PizzaOrderTest));
        }
}
```

Se for utilizar fixtures no plugin no app de testes, pode-se utilizar uma referência a ele da seguinte forma: 'plugin.pluginName.fixtureName' no array $fixtures.

10.10 - Gerando relatórios

Os relatórios padrões geram resultados mais simples. De todo jeito, é simples gerar resultados mais complexos. Para mudar a saída dos testes, basta ir no arquivo /cake/tests/libs/cake_reporter.php e sobrescrever os seguintes métodos:

Método	Descrição
paintHeader()	Imprime antes do teste ter sido iniciado
paintPass()	Imprime toda vez que um teste passa com êxito. Para recuperar um array com informações sobre o teste, use o método $this->getTestList(). O método $message é usado para recuperar o resultado. Lembre-se de chamar parent::paintPass($message).
paintFail()	Imprime sempre que um teste falhar. Lembre-se de chamar parent::paintFail($message).
paintFooter()	Imprime quando o teste termina.

Se quando estiver rodando paintPass() ou o paintFail(), for necessário esconder as saídas parentes, basta anexar os comentários html, da seguinte forma:

```
echo "\n<!-- ";
parent::paintFail($message);
echo " -->\n";
```

A seguir temos um exemplo de configurações do arquivo cake_reporter.php para criar uma tabela com os resultados do teste:

```
<?php

/**
 * CakePHP(tm) Tests <https://trac.cakephp.org/wiki/Developement/TestSuite>
 * Copyright 2005-2008, Cake Software Foundation, Inc.
 *                                1785 E. Sahara Avenue, Suite 490-204
 *                                Las Vegas, Nevada 89104
```

```
 *
 *  Licensed under The Open Group Test Suite License
 *  Redistributions of files must retain the above copyright notice.
 */

 class CakeHtmlReporter extends HtmlReporter {

        function CakeHtmlReporter($characterSet = 'UTF-8') {
                parent::HtmlReporter($characterSet);
        }

        function paintHeader($testName) {
                $this->sendNoCacheHeaders();
                $baseUrl = BASE;
                print "<h2>$testName</h2>\n";
                print "<table style=\"\"><th>Res.</th><th>Test case</th>
<th>Message</th>\n";
                flush();
        }

        function paintFooter($testName) {
                $colour = ($this->getFailCount() + $this->getExceptionCount() >
0 ? "red" : "green");
                print "</table>\n";
                print "<div style=\"";
                print "padding: 8px; margin-top: 1em; background-color: $colour;
color: white;";
                print "\">";
                print $this->getTestCaseProgress() . "/" .
$this->getTestCaseCount();
                print " test cases complete:\n";
                print "<strong>" . $this->getPassCount() . "</strong> passes, ";
                print "<strong>" . $this->getFailCount() . "</strong> fails and
";
                print "<strong>" . $this->getExceptionCount() . "</strong>
exceptions.";
                print "</div>\n";
        }

        function paintPass($message) {
                parent::paintPass($message);
                echo "<tr>\n\t<td width=\"20\" style=\"border: dotted 1px;
border-top: hidden; border-left: hidden;
                border-right: hidden\">\n";
                print "\t\t<span style=\"color: green;\">Pass</span>: \n";
                echo "\t</td>\n\t<td width=\"40%\" style=\"border: dotted 1px;
border-top: hidden; border-left: hidden; border-right: hidden\">\n";
                $breadcrumb = $this->getTestList();
                array_shift($breadcrumb);
                array_shift($breadcrumb);
                print implode("->", $breadcrumb);
                echo "\n\t</td>\n\t<td width=\"40%\" style=\"border: dotted 1px;
border-top: hidden; border-left: hidden; border-right: hidden\">\n";
                $message = split('at \[', $message);   print "->$message[0]<br
/>\n\n";
```

```
                echo "\n\t</td>\n</tr>\n\n"; }

        function paintFail($message) {
                echo "\n<!-- ";
                parent::paintFail($message);
                echo " -->\n";
                echo "<tr>\n\t<td width=\"20\" style=\"border: dotted 1px;
border-top: hidden; border-left: hidden; border-right: hidden\">\n";
                print "\t\t<span style=\"color: red;\">Fail</span>: \n";
                echo "\n\t</td>\n\t<td width=\"40%\" style=\"border: dotted 1px;
border-top: hidden; border-left: hidden; border-right: hidden\">\n";
                $breadcrumb = $this->getTestList();
                print implode("->", $breadcrumb);
                echo "\n\t</td>\n\t<td width=\"40%\" style=\"border: dotted 1px;
border-top: hidden; border-left: hidden; border-right: hidden\">\n";
                print "$message";
                echo "\n\t</td>\n</tr>\n\n";
        }

        function _getCss() {
                return parent::_getCss() . ' .pass { color: green; }';
        }
}
?>
```

10.11 - Agrupando testes

Se for necessário rodar diversos testes ao mesmo tempo, pode-se criar um grupo de teste. Crie um arquivo no diretório /app/tests/groups/ e o nomeie com algo do tipo nome_do_seu_grupo_de_teste.group.php. A classe desse arquivo herda a classe GroupTest e importa os testes da seguinte forma:

```
<?php

        class TryGroupTest extends GroupTest {

                var $label = 'try';

                function tryGroupTest() {
                        TestManager::addTestCasesFromDirectory($this,
APP_TEST_CASES . DS . 'models');
                }
        }
?>
```

Neste exemplo, todos os testes encontrados no diretório /app/tests/cases/models/ serão agrupados. Para agrupar um arquivo individual, utilize o método TestManager::addTestFile($this, nome_do_arquivo).

11 - O Console do CakePHP

O CakePHP possui diversas aplicações que executam na linha de comando. Algumas delas possuem funcionalidades específicas (como ACL ou i18n) e outras são para uso geral.

O Console do CakePHP possui um framework para a criação de shell scripts. O Console utiliza um configuração semelhante a de um *dispatcher* para carregar um shell ou uma task e tratar seus

parâmetros.

Importante

O PHP deve estar instalado como (CLI) se você pretende utilizar o Console.

11.1 - Conhecendo os shells padrões

A instalação padrão do Console do CakePHP possui os shells **api**, **bake**, **console**, **i18n**, **schema** e **testsuite**.

Nota

Para saber obter ajuda na utilização dos shells, execute o shell passando o parâmetro `help`:

```
$ cake api help

Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
Usage: cake api [<type>] <className> [-m <method>]
-----------------------------------------------
Parameters:


<type>
        Either a full path or type of class (model, behavior,
controller, component, view, helper).
        Available values:

        behavior        Look for class in CakePHP behavior path
        cache           Look for class in CakePHP cache path
        controller      Look for class in CakePHP controller path
        component       Look for class in CakePHP component path
        helper          Look for class in CakePHP helper path
        model           Look for class in CakePHP model path
        view            Look for class in CakePHP view path


<className>
        A CakePHP core class name (e.g: Component, HtmlHelper).
```

11.1.1 - api

Este shell possibilita a verificação da documentação (API) do CakePHP.

Para utilizá-lo, basta executar o comando abaixo passando como parâmetro o tipo da classe (behavior, controller, helper, etc):

```
$ cake api controller
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
Controller
---------------------------------------------------------------
1. flash($message, $url, $pause = 1)
2. header($status)
3. loadModel($modelClass = null, $id = null)
4. paginate($object = null, $scope = array(), $whitelist = array())
5. postConditions($data = array(), $op = null, $bool = 'AND', $exclusive =
false)
6. redirect($url, $status = null, $exit = true)
7. referer($default = null, $local = false)
8. render($action = null, $layout = null, $file = null)
9. set($one, $two = null)
10. setAction($action)
```

Vamos selecionar a opção 6 para ver os detalhes sobre o método `redirect` pressionando a tecla `6` e depois `Enter`.

```
Select a number to see the more information about a specific method. q to
quit. l to list.
[q] > 6
---------------------------------------------------------------
Controller::redirect($url, $status = null, $exit = true)
---------------------------------------------------------------

  Redirects to given $url, after turning off $this->autoRender.
  Script execution is halted after the redirect.

  @param mixed $url A string or array-based URL pointing to another location
within the app, or an absolute URL
  @param integer $status Optional HTTP status code (eg: 404)
  @param boolean $exit If true, exit() will be called after the redirect
  @return mixed void if $exit = false. Terminates script if $exit = true
  @access public
  @link http://book.cakephp.org/view/425/redirect
```

Você pode selecionar outro número da lista para ver os detalhes. Para listar todos os métodos digite `l` e depois `Enter`. Para sair digite apenas `Enter`

```
Select a number to see the more information about a specific method. q to
quit. l to list.
[q] > q
```

11.1.2 - bake

Este shell possibilita a geração de códigos. Classes da camada de modelo e de controle e as páginas da visão podem ser geradas. Os detalhes da utilização deste *shell* podem ser encontrados no Capítulo 2: Utilizando o bake.

11.1.2.1 - Customizando o bake

Para customizar os arquivos da visão gerados pelo bake é preciso criar a seguinte estrutura no diretório *DIR_NOVO_PROJETO/vendors/shells/templates/views* da sua aplicação antes de executar o bake. Vamos copiar os arquivos originais utilizados pelo cake para a nossa aplicação:

```
cp DIR_INSTALAÇÃO_CAKE/cake/console/libs/templates/views/*.ctp
DIR_NOVO_PROJETO/vendors/shells/templates/views
```

No diretório *DIR_INSTALAÇÃO_CAKE/cake/console/libs/templates/views/* existem quatro arquivos: form.ctp, home.ctp, index.ctp e view.ctp. Estes quatro arquivos são, respectivamente, o template para: O formulário de adicionar e editar os dados, a página principal da aplicação, a página de listagem e a página de visualização.

Para customizar os arquivos utilizados no scaffolding, basta copiar os arquivos do diretório *DIR_INSTALAÇÃO_CAKE/cake/libs/view/scaffolds* para o diretório *DIR_NOVO_PROJETO/views/scaffolds*:

```
cp DIR_INSTALAÇÃO_CAKE/cake/libs/views/scaffolds/*.ctp DIR_NOVO_PROJETO/views
/scaffolds
```

Após a cópia dos arquivos, o diretório *DIR_NOVO_PROJETO/views/scaffolds* deverá conter os seguintes arquivos:

```
edit.ctp
index.ctp
view.ctp
```

Listagem 85 - Arquivos do diretório *DIR_NOVO_PROJETO/views/scaffolds*

Feita a cópia dos arquivos, faça as alterações que desejar.

Para alterar o "esqueleto" da aplicação que será copiada na geração de um novo projeto, basta fazer as alterações no diretório DIR_INSTALAÇÃO_CAKE/cake/console/libs/templates/skel.

Após todas as alterações desejadas execute o bake para gerar seu novo projeto customizado.

11.1.3 - console

O *console* interativo do CakePHP é uma ferramenta que pode ser utilizada para testar partes da aplicação antes de realizar o *commit* do código.

Com ele, é possível testar as classes de modelo e as rotas. Nas classes de modelo é possível fazer buscas, salvar dados, remover e adicionar associações com outros modelos dinamicamente.

Veja o exemplo abaixo onde vamos fazer uma busca pelos Bookmarks. Observe que o shell retorna, além das informações do Bookmark, o Usuário e as Tags associadas:

```
$ cake console
```

```
> Bookmark->find('all')
Bookmark
        id: 1
        usuario_id: 1
        status: 1
        titulo: PRODEMGE
        url: http://www.prodemge.gov.br/
        descricao: Website da PRODEMGE
        hash:
```

```
        created: 2009-03-05 12:18:31
        modified: 2009-03-05 12:38:49
Usuario
        id: 1
        login: joao
        senha: dccd96c256bc7dd39bae41a405f25e43
        nome: João da Silva
        email: joao@email.com
        site: http://www.exemplo.com
        created: 2009-01-05 12:16:42
        modified: 2009-02-05 12:16:42
Tag
        id: 1
        nome: desenvolvimento
        created: 2009-02-04 12:09:11
        modified: 2009-02-04 12:09:11
        BookmarksTag: Array

        id: 2
        nome: certificaçãodigital
        created: 2009-02-05 12:36:44
        modified: 2009-02-05 12:36:55
        BookmarksTag: Array

        id: 3
        nome: datacenter
        created: 2009-02-05 12:38:24
        modified: 2009-02-05 12:38:24
        BookmarksTag: Array

Bookmark
        id: 2
        usuario_id: 1
        status: 1
        titulo: 2km interativa!
        url: http://www.2km.com.br
        descricao: Website da 2km interativa!
        hash:
        created: 2009-05-05 12:17:54
        modified: 2009-05-05 12:25:38
Usuario
        id: 1
        login: joao
        senha: dccd96c256bc7dd39bae41a405f25e43
        nome: João da Silva
        email: joao@email.com
        site: http://www.exemplo.com
        created: 2009-05-05 12:16:42
        modified: 2009-05-05 12:16:42
Tag
        id: 4
        nome: treinamento
        created: 2009-02-05 12:25:11
        modified: 2009-02-05 12:25:11
        BookmarksTag: Array
```

```
        id: 5
        nome: cakephp
        created: 2009-02-05 12:25:03
        modified: 2009-02-05 12:25:03
        BookmarksTag: Array

        id: 1
        nome: desenvolvimento
        created: 2009-02-04 12:09:11
        modified: 2009-02-04 12:09:11
        BookmarksTag: Array


>
```

Nota

Para sair do console basta digitar `quit` ou `exit` e depois `Enter`.

11.1.4 - i18n

Este shell é utilizado na internacionalização da aplicação. Os detalhes de sua utilização podem ser encontrados no `Capítulo 4: Localização e Internacionalização` deste manual.

11.1.5 - schema

O *Shell schema* pode ser utilizado para gerar um objeto do banco de dados e atualizar o banco a partir de um *schema*. Ele também pode criar e restaurar *snapshots* do banco.

Para gerar um objeto do banco de dados vamos utilizar a opção `generate`:

```
$ cake schema generate
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
Cake Schema Shell
---------------------------------------------------------------
Generating Schema...
Schema file: schema.php generated
```

O arquivo `schema.php` foi gerado na pasta */projetos/bookmarks/config/sql* e o seu conteúdo deve ser algo assim:

```
<?php
/* SVN FILE: $Id$ */
/* Bookmarks schema generated on: 2009-05-05 12:05:56 : 1241538476*/
class BookmarksSchema extends CakeSchema {
        var $name = 'Bookmarks';

        function before($event = array()) {
```

```
                return true;
        }

        function after($event = array()) {
        }

        var $bookmarks = array(
                'id' => array('type' => 'integer', 'null' => false, 'default' =>
NULL, 'key' => 'primary'),
                'usuario_id' => array('type' => 'integer', 'null' => false,
'default' => '0', 'key' => 'index'),
                'status' => array('type' => 'boolean', 'null' => false,
'default' => '0'),
                'titulo' => array('type' => 'string', 'null' => false),
                'url' => array('type' => 'text', 'null' => false, 'default' =>
NULL),
                'descricao' => array('type' => 'text', 'null' => true, 'default'
=> NULL),
                'hash' => array('type' => 'string', 'null' => false, 'length' =>
32),
                'created' => array('type' => 'datetime', 'null' => false,
'default' => NULL),
                'modified' => array('type' => 'datetime', 'null' => false,
'default' => NULL),
                'indexes' => array('PRIMARY' => array('column' => 'id', 'unique'
=> 1), 'usuario_id' => array('column' => 'usuario_id', 'unique' => 0))
        );
        var $bookmarks_tags = array(
                'id' => array('type' => 'integer', 'null' => false, 'default' =>
NULL, 'key' => 'primary'),
                'tag_id' => array('type' => 'integer', 'null' => false,
'default' => NULL, 'key' => 'index'),
                'bookmark_id' => array('type' => 'integer', 'null' => false,
'default' => NULL, 'key' => 'index'),
                'indexes' => array('PRIMARY' => array('column' => 'id', 'unique'
=> 1), 'tag_id' => array('column' => 'tag_id', 'unique' => 0), 'bookmark_id' =>
array('column' => 'bookmark_id', 'unique' => 0))
        );
        var $tags = array(
                'id' => array('type' => 'integer', 'null' => false, 'default' =>
NULL, 'key' => 'primary'),
                'nome' => array('type' => 'string', 'null' => false, 'default'
=> NULL, 'length' => 32, 'key' => 'unique'),
                'created' => array('type' => 'datetime', 'null' => false,
'default' => NULL),
                'modified' => array('type' => 'datetime', 'null' => false,
'default' => NULL),
                'indexes' => array('PRIMARY' => array('column' => 'id', 'unique'
=> 1), 'tag_bookmark' => array('column' => 'nome', 'unique' => 1))
        );
        var $usuarios = array(
                'id' => array('type' => 'integer', 'null' => false, 'default' =>
NULL, 'key' => 'primary'),
                'login' => array('type' => 'string', 'null' => false, 'length'
=> 25),
```

```
                'senha' => array('type' => 'string', 'null' => false, 'length'
=> 40),
                'nome' => array('type' => 'string', 'null' => true, 'default' =>
NULL, 'length' => 50),
                'email' => array('type' => 'string', 'null' => false, 'length'
=> 50),
                'site' => array('type' => 'string', 'null' => true, 'default' =>
NULL),
                'created' => array('type' => 'datetime', 'null' => false,
'default' => NULL),
                'modified' => array('type' => 'datetime', 'null' => false,
'default' => NULL),
                'indexes' => array('PRIMARY' => array('column' => 'id', 'unique'
=> 1))
        );
}
?>
```

Você pode gerar o schema e adicioná-lo ao seu SCM (Sistema de controle de versão, como o SVN ou GIT). Para gerar o script sql a partir do schema basta digitar:

```
$ cake schema dump bookmarks.sql
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
Cake Schema Shell
---------------------------------------------------------------
SQL dump file created in /projetos/bookmarks/config/sql/dumpbd.sql
```

O arquivo com o script do banco foi gerado com sucesso. O seu conteúdo será algo assim:

```
#Bookmarks sql generated on: 2009-05-05 17:05:21 : 1241555361

DROP TABLE IF EXISTS `bookmarks`;
DROP TABLE IF EXISTS `bookmarks_tags`;
DROP TABLE IF EXISTS `tags`;
DROP TABLE IF EXISTS `usuarios`;


CREATE TABLE `bookmarks` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `usuario_id` int(11) DEFAULT 0 NOT NULL,
        `status` tinyint(1) DEFAULT 0 NOT NULL,
        `titulo` varchar(255) NOT NULL,
        `url` text NOT NULL,
        `descricao` text DEFAULT NULL,
        `hash` varchar(32) NOT NULL,
        `created` datetime NOT NULL,
        `modified` datetime NOT NULL,   PRIMARY KEY  (`id`),
        KEY usuario_id (`usuario_id`));

CREATE TABLE `bookmarks_tags` (
```

```
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `tag_id` int(11) NOT NULL,
        `bookmark_id` int(11) NOT NULL, PRIMARY KEY  (`id`),
        KEY tag_id (`tag_id`),
        KEY bookmark_id (`bookmark_id`));

CREATE TABLE `tags` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `nome` varchar(32) NOT NULL,
        `created` datetime NOT NULL,
        `modified` datetime NOT NULL,   PRIMARY KEY  (`id`),
        UNIQUE KEY tag_bookmark (`nome`));

CREATE TABLE `usuarios` (
        `id` int(11) NOT NULL AUTO_INCREMENT,
        `login` varchar(25) NOT NULL,
        `senha` varchar(40) NOT NULL,
        `nome` varchar(50) DEFAULT NULL,
        `email` varchar(50) NOT NULL,
        `site` varchar(255) DEFAULT NULL,
        `created` datetime NOT NULL,
        `modified` datetime NOT NULL,   PRIMARY KEY  (`id`));
```

Script 2 - Script do banco de dados da aplicação bookmarks gerado a partir do *schema*.

Nota

É recomendável executar o shell *schema* a partir do diretório raiz da sua aplicação (APP). Caso contrário, você terá que informar o caminho completo para a pasta onde encontra-se o arquivo default.php através do parâmetro -path.

11.1.6 - testsuite

Este shell possibilita a execução dos testes através da linha de comando.

Importante

Para utilizar o shell testsuite é preciso ter o SimpleTest instalado. Os detalhes da instalação podem ser vistos no Capítulo 10: Testes.

Você pode executar testes do core do CakePHP e da sua aplicação. Para exemplificar vamos executar o teste da biblioteca File. para isso vamos digitar:

```
$ cake testsuite core case file
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
CakePHP Test Shell
---------------------------------------------------------------
Running core case file
```

```
Individual test case: libs/file.test.php
1/1 test cases complete: 93 passes.
```

Para executar o teste da classe de modelo `Bookmark` criado pelo bake, vamos digitar:

```
$ cake testsuite app case models/bookmark
```

```
Welcome to CakePHP v1.3.10 Console
---------------------------------------------------------------
App : bookmarks
Path: /projetos/bookmarks
---------------------------------------------------------------
CakePHP Test Shell
---------------------------------------------------------------
Running app case models/bookmark
Individual test case: models/bookmark.test.php
1/1 test cases complete: 3 passes.
```

11.2 - Criando shells e tasks

O CakePHP permite a criação de tasks e shells customizadas.

11.2.1 - Criando um shell customizado

Criar um shell customizado no CakePHP é uma tarefa muito simples. Basta extender a classe `Shell` e criar o método `main` com o código desejado. Vamos criar o *shell* `relatorio` para exemplificar.

Primeiro, vamos criar o arquivo `relatorio.php` na pasta */vendors/shells*.

```
<?php
class RelatorioShell extends Shell {
    var $uses = array('Bookmark');

    function main() {
        // Recupera os bookmarks criados a partir da última semana
        $semana_passada = date('Y-m-d H:i:s', strtotime('-1 week'));
        $bookmarks = $this->Bookmark->find('all',array(
            'conditions' => array(
                'Bookmark.created > ' =>  $semana_passada
            )
        ));

        // Imprime os dados do bookmark
        foreach($bookmarks as $bookmark) {
            $tags = "";
            $this->out('Nome:       ' . $bookmark['Bookmark']['titulo'] . "\n");
            $this->out('Criado em:  ' . $bookmark['Bookmark']['created'] . "\n");
            foreach($bookmark['Tag'] as $tag) {
                $tags[] = $tag['nome'];
            }
            $this->out('Tags:        '.implode(", ",$tags));
            $this->out('---------------------------------------' .    "\n");
        }
    }
}
```

```
?>
```

11.2.2 - Criando uma task

Tasks são pequenas extensões para os shells. Elas permitem que sua lógica seja compartilhada por mais de um *shell*. Para isso você deve utilizar a variável `$tasks`. As tasks devem extender a classe Shell e devem ser salvas no diretório */vendors/shells/tasks*. Veja a seguir o exemplo da task formata_tags.php.

```
<?php
class FormataTagsTask extends Shell {
    // Esta é a mesma variável utilizada nas classes de controle $uses
    var $uses = array('Bookmark');
    function execute($tags) {
        $nome_tags = array();
        if (is_array($tags) && !empty($tags)) {
            foreach($tags as $tag) {
                $nome_tags[] = $tag['nome'];
            }
        }
        return $nome_tags;
    }
}
?>
```

Listagem 86 - Exemplo da task formata_tags.php

Para utilizar uma task, basta declarar a variável `$tasks` com um array contendo o nome de cada uma, separadas por vírgula.

O shell que criamos anteriormente se fosse utilizar a task FormataTagsTask, ficaria assim:

```
<?php
class RelatorioShell extends Shell {
    var $uses = array('Bookmark');
    var $tasks = array('FormataTags');
    function main() {
        // Recupera os bookmarks criados a partir da última semana
        $semana_passada = date('Y-m-d H:i:s', strtotime('-1 week'));
        $bookmarks = $this->Bookmark->find('all',array(
            'conditions' => array(
                'Bookmark.created > ' =>  $semana_passada
            )
        ));
        // Imprime os dados do bookmark
        foreach($bookmarks as $bookmark) {
            $tags = "";
            $this->out('Nome:        ' . $bookmark['Bookmark']['titulo'] . "\n");
            $this->out('Criado em:  ' . $bookmark['Bookmark']['created'] . "\n");
            $tags = $this->FormataTags->execute($bookmark['Tag']);
            if (is_array($tags) && !empty($tags)) {
                $this->out('Tags:        '.implode(", ",$tags));
            }
            $this->out('---------------------------------------' .    "\n");
        }
    }
```

```
}
?>
```