CakePHP Cookbook Documentation

Versão 3.x

Cake Software Foundation

03 May, 2016

Conteúdo

CAPÍTULO 1

CakePHP num piscar de olhos

O CakePHP é concebido para tornar tarefas de desenvolvimento web mais simples e fáceis. Por fornecer uma caixa de ferramentas completa para você poder começar, o CakePHP funciona bem em conjunto ou isoladamente.

O objetivo desta análise é introduzir os conceitos gerais presentes no CakePHP, e lhe dar uma rápida visão geral de como estes conceitos são implementados. Se você está ávido para começar um projeto, você pode *começar com o tutorial*, ou `mergulhar na documentação`.

Convenções Sobre Configuração

O CakePHP provê uma estrutura organizacional básica que cobre nomenclaturas de classes, nomenclaturas de arquivos, nomenclaturas de banco de dados, e outras convenções. Apesar das convenções levarem algum tempo para serem assimiladas, ao seguí-las o CakePHP evita configuração desnecessário e cria uma estrutura de aplicação uniforme que faz trabalhar com vários projetos uma tarefa suave. O *capítulo de convenções* cobre as variadas convenções que o CakePHP utiliza.

A camada Model

A camada Model representa a parte da sua aplicação que implementa a lógica de negócio. Ela é responsável por recuperar dados e convertê-los nos conceitos significativos primários na sua aplicação. Isto inclui processar, validar, associar ou qualquer outra tarefa relacionada à manipulação de dados.

No caso de uma rede social, a camada Model deveria tomar cuidado de tarefas como salvar os dados do usuário, salvar as associações entre amigos, salvar e recuperar fotos de usuários, localizar sugestões para novos amigos, etc. Os objetos de modelo podem ser pensados como "Friend", "User", "Comment", ou "Photo". Se nós quiséssemos carregar alguns dados da nossa tabela `users` poderiamos fazer:

```
use Cake\ORM\TableRegistry;

$users = TableRegistry::get('Users');
$query = $users->find();
```

```
foreach ($query as $row) {
    echo $row->username;
}
```

Você pode notar que não precisamos escrever nenhum código antes de podermos começar a trabalhar com nossos dados. Por usar convenções, o CakePHP irá utilizar classes padrão para tabelas e entidades ainda não definidas.

Se nós quiséssemos criar um usuário e salvá-lo (com validação) fariamos algo assim:

```
use Cake\ORM\TableRegistry;

$users = TableRegistry::get('Users');
$user = $users->newEntity(['email' => 'mark@example.com']);
$users->save($user);
```

A camada View

A View renderiza uma apresentação de dados modelados. Estando separada dos objetos da Model, é responsável por utilizar a informação que tem disponível para produzir qualquer interface de apresentação que a sua aplicação possa precisar.

Por exemplo, a view pode usar dados da model para renderizar uma página HTML que os conhtenha, ou um resultado formatado como XML:

```
// No arquivo view, nós renderizaremos um 'elemento' para cada usuário.
<?php foreach ($users as $user): ?>
    <div class="user">
        <?= $this->element('user', ['user' => $user]) ?>
    </div>
<?php endforeach; ?>
```

A camada View provê alguma variedade de extensões como *Elements* e *View Cells (Células de Visualização)* para permitir que você reutilize sua lógica de apresentação.

A camada View não está limitada somente a HTML ou apresentação textual dos dados. Ela pode ser usada para entregar formatos de dado comuns como JSON, XML, e através de uma arquitetura encaixável qualquer outro formato que você venha precisar.

A camada Controller

A camada Controller manipula requisições dos usuários. É responsável por renderizar uma resposta com o auxílio de ambas as camadas, Model e View respectivamente.

Um controller pode ser visto como um gerente que certifica-se que todos os recursos necessários para completar uma tarefa sejam delegados aos trabalhadores corretos. Ele aguarda por petições dos clientes, checa suas validades de acordo com autenticação ou regras de autorização, delega requisições ou processamento de dados da camada Model, selecciona o tipo de dados de apresentação que os clientes estão aceitando, e

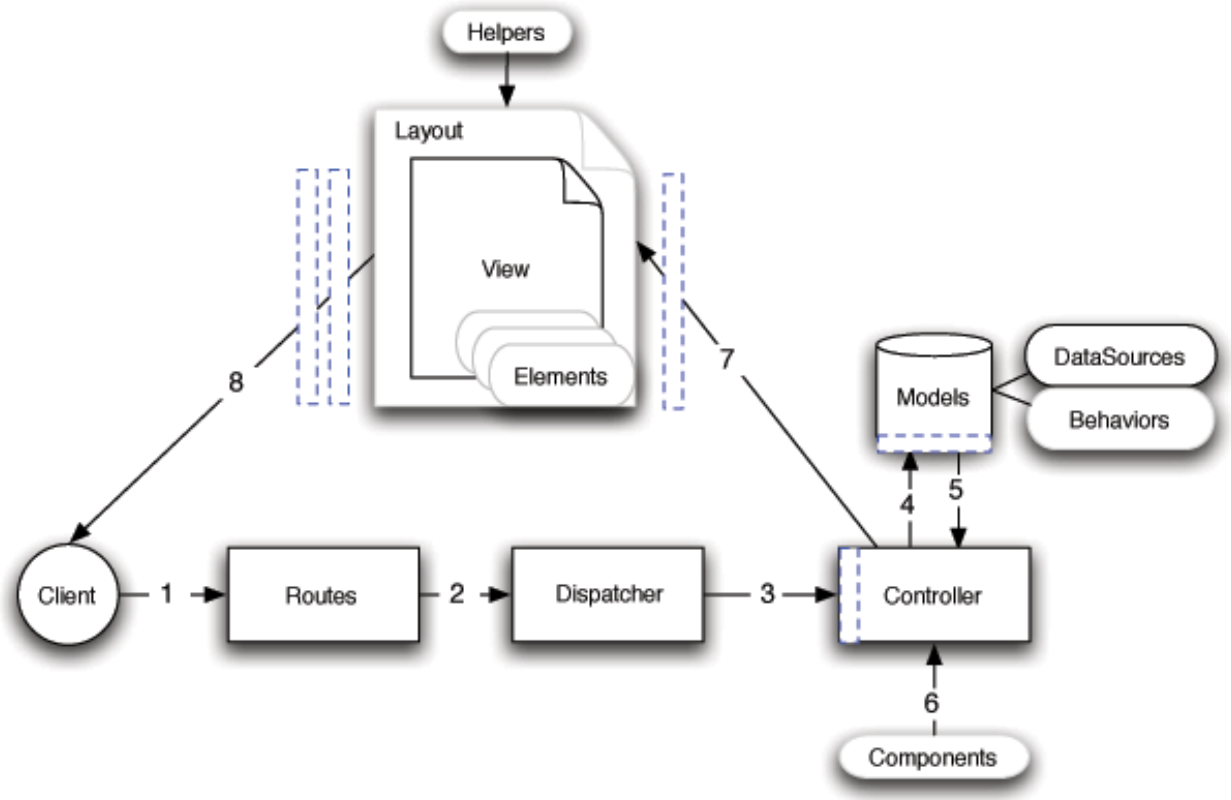
finalmente delega o processo de renderização para a camada View. Um exemplo de controller para registro de usuário seria:

```
public function add()
{
    $user = $this->Users->newEntity();
    if ($this->request->is('post')) {
        $user = $this->Users->patchEntity($user, $this->request->data);
        if ($this->Users->save($user, ['validate' => 'registration'])) {
            $this->Flash->success(__('Você está registrado.'));
        } else {
            $this->Flash->error(__('Houve algum problema.'));
        }
    }
    $this->set('user', $user);
}
```

Você pode perceber que nós nunca renderizamos uma view explicitamente. As convenções do CakePHP tomarão cuidado de selecionar a view correta e renderizá-la como os dados definidos com `set()`.

Ciclo de Requisições do CakePHP

Agora que você é familiar com as diferentes camadas no CakePHP, vamos revisar como um cíclo de requisição funciona no CakePHP:



O cíclo de requisição típico do CakePHP começa com um usuário solicitando uma página ou recurso na sua aplicação. Em alto nível cada requisição vai através dos seguintes passos:

1. A requisição é primeiramente processada pela suas rotas.
2. Depois da requisição ter sido roteada, o despachante irá selecionar o objeto de controller correto para manipulá-la.
3. A action do controller é chamada e o controller interage com os models e components requisitados.
4. O controller delega a criação de resposta à view para gerar os dados de saída resultantes dos dados do model.

Apenas o Começo

Esperamos que essa rápida visão geral tenha despertado seu interesse. Alguns outros grandes recursos no CakePHP são:

- *Framework de cache* que integra com Memcached, Redis e outros backends.
- Poderosas *ferramentas de geração de código* para você sair em disparada.
- *Framework de teste integrado* para você assegurar-se que seu código funciona perfeitamente.

Os próximos passos óbvios são *baixar o CakePHP*, ler o *tutorial e construir algo fantástico*.

Leitura adicional

Onde Conseguir Ajuda

O website oficial do CakePHP

http://www.cakephp.org

O website oficial do CakePHP é sempre um ótimo lugar para visitar. Ele provê links para ferramentas comunmente utilizadas por desenvolvedores, screencasts, oportunidades de doação e downloads.

O Cookbook

http://book.cakephp.org

Esse manual deveria ser o primeiro lugar para onde você iria afim de conseguir respostas. Assim como muitos outros projetos de código aberto, nós conseguimos novos colaboradores regularmente. Tente o seu melhor para responder suas questões por si só. Respostas vão vir lentamente, e provavelmente continuarão longas. Você pode suavizar nossa carga de suporte. Tanto o manual quanto a API possuem um componente online.

A Bakery

http://bakery.cakephp.org

A "padaria" do CakePHP é um local para todas as coisas relacionadas ao CakePHP. Visite-a para tutoriais, estudos de caso e exemplos de código. Uma vez que você tenha se familiarizado com o CakePHP, autentique-se e compartilhe seu conhecimento com a comunidade, ganhe instantaneamente fama e fortuna.

A API

http://api.cakephp.org/

Diretamente ao ponto, dos desenvolvedores do núcleo do CakePHP, a API (Application Programming Interface) do CakePHP é a mais compreensiva documentação sobre os detalhes técnicos e minuciosos sobre do funcionamento interno do framework.

Os Testes de Caso

Se você sente que a informação provida pela API não é suficiente, verifique os códigos de testes de caso do CakePHP. Eles podem servir como exemplos práticos para funções e e utilização de dados referentes a uma classe.:

```
tests/TestCase/
```

O canal de IRC

Canal de IRC na irc.freenode.net:

- #cakephp – Discussão geral
- #cakephp-docs – Documentação
- #cakephp-bakery – Bakery
- #cakephp-fr – Canal francês.

Se você está travado, nos faça uma visita no canal de IRC do CakePHP. Alguém do time de desenvolvimento[1] normalmente está conectado, especiamente nos horários diurnos da América do Sul e América do Norte. Nós apreciaríamos ouví-lo se você precisar de ajuda, se quiser encontrar usuários da sua área ou ainda se quiser doar seu novo carro esporte.

Grupo oficial de discussão do CakePHP

Grupo de discussão do Google[2]

[1] https://github.com/cakephp?tab=members
[2] http://groups.google.com/group/cake-php

O CakePHP também possui seu grupo de discussão oficial no Google Grupos. Existem milhares de pessoas discutindo projetos CakePHP, ajudando uns aos outros, resolvendo problemas, construindo projetos e compartilhando idéias. Pode ser uma grande fonte para encontrar respostas arquivadas, perguntas frequentes e conseguir respostas para problemas imediatos. Junte-se a outros usuários do CakePHP e comece a conversar.

Stackoverflow

http://stackoverflow.com/[3]

Marque suas questões com a tag `cakephp` e especifique a versão que você está utilizando para permitir que usuários do stackoverflow achem suas questões.

Onde conseguir ajuda em sua língua

Francês

- Comunidade CakePHP francesa[4]

Português brasileiro

- Comunidade CakePHP brasileira[5]

Convenções do CakePHP

Nós somos grandes fãs de convenção sobre configuração. Apesar de levar um pouco de tempo para aprender as convenções do CakePHP, você economiza tempo a longo prazo. Ao seguir as convenções, você ganha funcionalidades instantaneamente e liberta-se do pesadelo de manutenção e rastreamento de arquivos de configuração. Convenções também prezam por uma experiência de desenvolvimento uniforme, permitindo que outros desenvolvedores ajudem mais facilmente.

Convenções para Controllers

Os nomes das classes de Controllers são pluralizados, CamelCased, e terminam em `Controller`. `PeopleController` e `LatestArticlesController` são exemplos de nomes convencionais para controllers.

Métodos públicos nos Controllers são frequentemente referenciados como 'actions' acessíveis através de um navegador web. Por exemplo, o `/articles/view` mapeia para o método `view()` do `ArticlesController` sem nenhum esforço. Métodos privados ou protegidos não podem ser acessados pelo roteamento.

[3] http://stackoverflow.com/questions/tagged/cakephp/
[4] http://cakephp-fr.org
[5] http://cakephp-br.org

Considerações de URL para nomes de Controller

Como você acabou de ver, controllers singulares mapeiam facilmente um caminho simples, todo em minúsculo. Por exemplo, `ApplesController` (o qual deveria ser definido no arquivo de nome 'ApplesController.php') é acessado por http://example.com/apples.

Controllers com múltiplas palavras *podem* estar em qualquer forma 'flexionada' igual ao nome do controller, então:

- /redApples
- /RedApples
- /Red_apples
- /red_apples

Todos resolverão para o index do controller RedApples. Porém, a forma correta é que suas URLs sejam minúsculas e separadas por sublinhado, portanto /red_apples/go_pick é a forma correta de acessar a action `RedApplesController::go_pick`.

When you create links using `this->Html->link()`, you can use the following conventions for the url array:

```
$this->Html->link('link-title', [
    'prefix' => 'MyPrefix' // CamelCased
    'plugin' => 'MyPlugin', // CamelCased
    'controller' => 'ControllerName', // CamelCased
    'action' => 'actionName' // camelBacked
]
```

Para mais informações sobre o manuseio de URLs e parâmetros do CakePHP, veja *Connecting Routes*.

Convenções para nomes de Classes e seus nomes de arquivos

No geral, nomes de arquivos correspondem aos nomes das classes, e seguem os padrões PSR-0 ou PSR-4 para auto-carregamento. A seguir seguem exemplos de nomes de classes e de seus arquivos:

- A classe de Controller **KissesAndHugsController** deveria ser encontrada em um arquivo nomeado **KissesAndHugsController.php**
- A classe de Component **MyHandyComponent** deveria ser encontrada em um arquivo nomeado **MyHandyComponent.php**
- A classe de Table **OptionValuesTable** deveria ser encontrada em um arquivo nomeado **OptionValuesTable.php**.
- A classe de Entity **OptionValue** deveria ser encontrada em um arquivo nomeado **OptionValue.php**.
- A classe de Behavior **EspeciallyFunkableBehavior** deveria ser encontrada em um arquivo nomeado **EspeciallyFunkableBehavior.php**
- A classe de View **SuperSimpleView** deveria ser encontrada em um arquivo nomeado **SuperSimpleView.php**

- A classe de Helper **BestEverHelper** deveria ser encontrada em um arquivo nomeado **BestEverHelper.php**

Cada arquivo deveria estar localizado no diretório/namespace apropriado de sua aplicação.

Convenções para Models e Databases

Os nomes de classe de Tables são pluralizadas e CamelCased. People, BigPeople, and ReallyBigPeople são todos exemplos convencionais de models.

Os nomes de Tables correspondentes aos models do CakePHP são pluralizadas e separadas por sublinhado. As tables sublinhadas para os models mencionados acima seriam `people`, `big_people`, e `really_big_people`, respectively.

Você pode utilizar a biblioteca utility `Cake\Utility\Inflector` para checar o singular/plural de palavras. Veja o *Inflector* para mais informações. Recomenda-se que as tables sejam criadas e mantidas na língua inglesa.

Campos com duas ou mais palavras são separados por sublinhado: first_name.

Chaves estrangeiras nos relacionamentos hasMany, belongsTo ou hasOne são reconhecidas por padrão como o nome (singular) da table relacionada seguida por _id. Então se Bakers hasMany Cakes, a table cakes irá referenciar-se para a table bakers através da chave estrangeira baker_id. Para uma tabela como category_types a qual o nome contém mais palavras, a chave estrangeira seria a category_type_id.

tables de união, usadas no relacionamento BelongsToMany entre models, devem ser nomeadas depois das tables que ela está unindo, ordenadas em ordem alfabética (apples_zebras ao invés de zebras_apples).

Convenções para Views

Arquivos de template views são nomeadas seguindo as funções que a exibem do controller, separadas por sublinhado. A função getReady() da classe PeopleController buscará por um template view em **src/Template/People/get_ready.ctp**. O padrão é **src/Template/Controller/underscored_function_name.ctp**.

Por nomear as partes de sua aplicação utilizando as convenções do CakePHP, você ganha funcionalidades sem luta e sem amarras de configuração. Aqui está um exemplo final que enlaça as convenções juntas:

- Table: "people"
- Classe Table: "PeopleTable", encontrada em **src/Model/Table/PeopleTable.php**
- Classe Entity: "Person", encontrada em **src/Model/Entity/Person.php**
- Classe Controller: "PeopleController", encontrada em **src/Controller/PeopleController.php**
- View template, encontrado em **src/Template/People/index.ctp**

Utilizando estas convenções, o CakePHP sabe que uma requisição para http://example.com/people/ mapeia para uma chamada da função `index()` do PeopleController, onde o model Person é automaticamente disponbilizado (e automaticamente amarrado à table 'people' no banco de dados), e então renderiza-se um arquivo view template. Nenhuma destes relacionamentos foi configurado de qualquer forma se não por criar classes e arquivos que você precisaria criar de qualquer forma.

Agora que você foi introduzido aos fundamentos do CakePHP, você pode tentar seguir através do *Tutorial - Criando um Blog - Parte 1* para ver como as coisas se encaixam juntas.

Estrutura de pastas do CakePHP

Depois de você ter baixado e extraído o CakePHP, aí estão os arquivos e pastas que você deve ver:

- bin
- config
- logs
- plugins
- src
- tests
- tmp
- vendor
- webroot
- .htaccess
- composer.json
- index.php
- README.md

Você notará alguns diretórios principais:

- The *bin* folder holds the Cake console executables.
- O diretório *config* contem os (poucos) *Configuração* arquivos de configuração que o CakePHP utiliza. Detalhes de conexão com banco de dados, inicialização, arquivos de configuração do núcleo da aplicação, e relacionados devem ser postos aqui.
- O diretório *logs* será normalmente onde seus arquivos de log ficarão, dependendo das suas configurações.
- O diretório *plugins* será onde *Plugins* que sua aplicação utiliza serão armazenados.
- O diretório *src* será onde você fará sua mágica: é onde os arquivos da sua aplicação serão colocados.
- O diretório *tests* será onde você colocará os testes de caso para sua aplicação.
- O diretório *tmp* será onde o CakePHP armazenará dados temporários. O modo como os dados serão armazenados depende da configuração do CakePHP, mas esse diretório é comunmente usado para armazenar descrições de modelos e algumas vezes informação de sessão.
- O diretório *vendor* será onde o CakePHP e outras dependências da aplicação serão instalados. Faça uma nota pessoal para **não** editar arquivos deste diretório. Nós não podemos ajudar se você tivé-lo feito.

- O diretório *webroot* será a raíz pública de documentos da sua aplicação. Ele contem todos os arquivos que você gostaria que fossem públicos.

  Certifique-se que os diretórios *tmp* e *logs* existem e são passíveis de escrita, senão a performance de sua aplicação será severamente impactada. Em modo de debug, o CakePHP irá alertá-lo se este for o caso.

O diretório src

O diretório *src* do CakePHP é onde você fará a maior parte do desenvolvimento de sua aplicação. Vamos ver mais de perto a estrutura de pastas dentro de *src*.

Console Contém os comandos e tarefas de console para sua aplicação. Para mais informações veja *Console e Shells*.

Controller Contém os controllers de sua aplicação e seus componentes.

Locale Armazena arquivos textuais para internacionalização.

Model Contém as tables, entities e behaviors de sua aplicação.

View Classes de apresentação são alocadas aqui: cells, helpers, e arquivos view.

Template Arquivos de apresentação são alocados aqui: elements, páginas de erro, layouts, e templates view.

CAPÍTULO 2

Guia de Início Rápido

A melhor forma de viver experiências e aprender sobre CakePHP é sentar e construir algo. Para começar nós iremos construir uma aplicação simples de blog.

Tutorial - Criando um Bookmarker - Parte 1

Esse tutorial vai guiar você através da criação de uma simples aplicação de marcação (bookmarker). Para começar, nós vamos instalar o CakePHP, criar nosso banco de dados, e usar as ferramentas que o CakePHP fornece para obter nossa aplicação de pé rápido.

Aqui está o que você vai precisar:

1. Um servidor de banco de dados. Nós vamos usar o servidor MySQL neste tutorial. Você precisa saber o suficiente sobre SQL para criar um banco de dados: O CakePHP vai tomar as rédeas a partir daí. Por nós estarmos usando o MySQL, também certifique-se que você tem a extensão `pdo_mysql` habilitada no PHP.
2. Conhecimento básico sobre PHP.

Vamos começar!

Instalação do CakePHP

A maneira mais fácil de instalar o CakePHP é usando Composer, um gerenciador de dependências para o PHP. É uma forma simples de instalar o CakePHP a partir de seu terminal ou prompt de comando. Primeiro, você precisa baixar e instalar o Composer. Se você tiver instalada a extensão cURL do PHP, execute o seguinte comando:

```
curl -s https://getcomposer.org/installer | php
```

Ao invés disso, você também pode baixar o arquivo `composer.phar` do site[1] oficial.

[1] https://getcomposer.org/download/

Em seguida, basta digitar a seguinte linha no seu terminal a partir do diretório onde se localiza o arquivo `composer.phar` para instalar o esqueleto de aplicações do CakePHP no diretório `bookmarker`.

```
php composer.phar create-project --prefer-dist cakephp/app bookmarker
```

A vantagem de usar Composer é que ele irá completar automaticamente um conjunto importante de tarefas, como configurar as permissões de arquivo e criar a sua **config/app.php**.

Há outras maneiras de instalar o CakePHP. Se você não puder ou não quiser usar Composer, veja a seção *Instalação*.

Independentemente de como você baixou o CakePHP, uma vez que sua instalação for concluída, a estrutura dos diretórios deve ficar parecida com o seguinte:

```
/bookmarker
    /bin
    /config
    /logs
    /plugins
    /src
    /tests
    /tmp
    /vendor
    /webroot
    .editorconfig
    .gitignore
    .htaccess
    .travis.yml
    composer.json
    index.php
    phpunit.xml.dist
    README.md
```

Agora pode ser um bom momento para aprender sobre como a estrutura de diretórios do CakePHP funciona: Confira a seção *Estrutura de pastas do CakePHP*.

Verificando nossa instalação

Podemos checar rapidamente que a nossa instalação está correta, verificando a página inicial padrão. Antes que você possa fazer isso, você vai precisar iniciar o servidor de desenvolvimento:

```
bin/cake server
```

Isto irá iniciar o servidor embutido do PHP na porta 8765. Abra `http://localhost:8765` em seu navegador para ver a página de boas-vindas. Todas as verificações devem estar checadas corretamente, a não ser a conexão com banco de dados do CakePHP. Se não, você pode precisar instalar extensões do PHP adicionais, ou definir permissões de diretório.

Criando o banco de dados

Em seguida, vamos criar o banco de dados para a nossa aplicação. Se você ainda não tiver feito isso, crie um banco de dados vazio para uso nesse tutorial, com um nome de sua escolha, por exemplo,

cake_bookmarks. Você pode executar o seguinte SQL para criar as tabelas necessárias:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created DATETIME,
    modified DATETIME
);

CREATE TABLE bookmarks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(50),
    description TEXT,
    url TEXT,
    created DATETIME,
    modified DATETIME,
    FOREIGN KEY user_key (user_id) REFERENCES users(id)
);

CREATE TABLE tags (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    created DATETIME,
    modified DATETIME,
    UNIQUE KEY (title)
);

CREATE TABLE bookmarks_tags (
    bookmark_id INT NOT NULL,
    tag_id INT NOT NULL,
    PRIMARY KEY (bookmark_id, tag_id),
    INDEX tag_idx (tag_id, bookmark_id),
    FOREIGN KEY tag_key(tag_id) REFERENCES tags(id),
    FOREIGN KEY bookmark_key(bookmark_id) REFERENCES bookmarks(id)
);
```

Você deve ter notado que a tabela bookmarks_tags utilizada uma chave primária composta. O CakePHP suporta chaves primárias compostas quase todos os lugares, tornando mais fácil construir aplicações multi-arrendados.

Os nomes de tabelas e colunas que usamos não foram arbitrárias. Usando *convenções de nomenclatura* do CakePHP, podemos alavancar o desenvolvimento e evitar ter de configurar o framework. O CakePHP é flexível o suficiente para acomodar até mesmo esquemas de banco de dados legados inconsistentes, mas aderir às convenções vai lhe poupar tempo.

Configurando o banco de dados

Em seguida, vamos dizer ao CakePHP onde o nosso banco de dados está como se conectar a ele. Para muitos, esta será a primeira e última vez que você vai precisar configurar qualquer coisa.

A configuração é bem simples: basta alterar os valores do array Datasources.default no arquivo

config/app.php pelos que se aplicam à sua configuração. A amostra completa da gama de configurações pode ser algo como o seguinte:

```
return [
    // Mais configuração acima.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_bookmarks',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Mais configuração abaixo.
];
```

Depois de salvar o seu arquivo **config/app.php**, você deve notar que a mensagem 'CakePHP is able to connect to the database' tem uma marca de verificação.

Nota: Uma cópia do arquivo de configuração padrão do CakePHP é encontrado em **config/app.default.php**.

Gerando o código base

Devido a nosso banco de dados seguir as convenções do CakePHP, podemos usar o *bake console* para gerar rapidamente uma aplicação básica . Em sua linha de comando execute:

```
bin/cake bake all users
bin/cake bake all bookmarks
bin/cake bake all tags
```

Isso irá gerar os controllers, models, views, seus casos de teste correspondentes, e fixtures para os nossos users, bookmarks e tags. Se você parou seu servidor, reinicie-o e vá para `http://localhost:8765/bookmarks`.

Você deverá ver uma aplicação que dá acesso básico, mas funcional a tabelas de banco de dados. Adicione alguns users, bookmarks e tags.

Adicionando criptografia de senha

Quando você criou seus users, você deve ter notado que as senhas foram armazenadas como texto simples. Isso é muito ruim do ponto de vista da segurança, por isso vamos consertar isso.

Este também é um bom momento para falar sobre a camada de modelo. No CakePHP, separamos os métodos que operam em uma coleção de objetos, e um único objeto em diferentes classes. Métodos que operam na recolha de entidades são colocadas na classe *Table*, enquanto as características pertencentes a um único registro são colocados na classe *Entity*.

Por exemplo, a criptografia de senha é feita no registro individual, por isso vamos implementar esse comportamento no objeto entidade. Dada a circunstância de nós querermos criptografar a senha cada vez que é definida, vamos usar um método modificador/definidor. O CakePHP vai chamar métodos de definição baseados em convenções a qualquer momento que uma propriedade é definida em uma de suas entidades. Vamos adicionar um definidor para a senha. Em **src/Model/Entity/User.php** adicione o seguinte:

```
namespace App\Model\Entity;

use Cake\ORM\Entity;
use Cake\Auth\DefaultPasswordHasher;

class User extends Entity
{

    // Code from bake.

    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();
        return $hasher->hash($value);
    }
}
```

Agora atualize um dos usuários que você criou anteriormente, se você alterar sua senha, você deve ver um senha criptografada ao invés do valor original nas páginas de lista ou visualização. O CakePHP criptografa senhas com bcrypt[2] por padrão. Você também pode usar sha1 ou md5 caso venha a trabalhar com um banco de dados existente.

Recuperando bookmarks com uma tag específica

Agora que estamos armazenando senhas com segurança, podemos construir algumas características mais interessantes em nossa aplicação. Uma vez que você acumulou uma coleção de bookmarks, é útil ser capaz de pesquisar através deles por tag. Em seguida, vamos implementar uma rota, a ação do controller, e um método localizador para pesquisar através de bookmarks por tag.

Idealmente, nós teríamos uma URL que se parece com `http://localhost:8765/bookmarks/tagged/funny/ca`
Isso deveria nos permitir a encontrar todos os bookmarks que têm as tags 'funny', 'cat' e 'gifs'. Antes de podermos implementar isso, vamos adicionar uma nova rota. Em **config/routes.php**, adicione o seguinte na parte superior do arquivo:

```
Router::scope(
    '/bookmarks',
    ['controller' => 'Bookmarks'],
    function ($routes) {
        $routes->connect('/tagged/*', ['action' => 'tags']);
```

[2] http://codahale.com/how-to-safely-store-a-password/

```
    }
);
```

O acima define uma nova "rota" que liga o caminho `/bookmarks/tagged/*`, a `BookmarksController::tags()`. Ao definir rotas, você pode isolar como suas URLs parecerão, de como eles são implementadas. Se fôssemos visitar `http://localhost:8765/bookmarks/tagged`, deveriamos ver uma página de erro informativa do CakePHP. Vamos implementar esse método ausente agora. Em **src/Controller/BookmarksController.php** adicione o seguinte:

```
public function tags()
{
    $tags = $this->request->params['pass'];
    $bookmarks = $this->Bookmarks->find('tagged', [
        'tags' => $tags
    ]);
    $this->set(compact('bookmarks', 'tags'));
}
```

Criando o método localizador

No CakePHP nós gostamos de manter as nossas ações do controller enxutas, e colocar a maior parte da lógica de nossa aplicação nos modelos. Se você fosse visitar a URL `/bookmarks/tagged` agora, você veria um erro sobre o método `findTagged` não estar implementado ainda, então vamos fazer isso. Em **src/Model/Table/BookmarksTable.php** adicione o seguinte:

```
public function findTagged(Query $query, array $options)
{
    $fields = [
        'Bookmarks.id',
        'Bookmarks.title',
        'Bookmarks.url',
    ];
    return $this->find()
        ->distinct($fields)
        ->matching('Tags', function ($q) use ($options) {
            return $q->where(['Tags.title IN' => $options['tags']]);
        });
}
```

Nós implementamos um método *localizador customizado*. Este é um conceito muito poderoso no CakePHP que lhe permite construir consultas reutilizáveis. Em nossa pesquisa, nós alavancamos o método `matching()` que nos habilita encontrar bookmarks que têm uma tag 'correspondente'.

Criando a view

Agora, se você visitar a URL `/bookmarks/tagged`, o CakePHP irá mostrar um erro e deixá-lo saber que você ainda não fez um arquivo view. Em seguida, vamos construir o arquivo view para a nossa ação `tags`. Em **src/Template/Bookmarks/tags.ctp** coloque o seguinte conteúdo:

```
<h1>
    Bookmarks tagged with
    <?= $this->Text->toList($tags) ?>
</h1>

<section>
<?php foreach ($bookmarks as $bookmark): ?>
    <article>
        <h4><?= $this->Html->link($bookmark->title, $bookmark->url) ?></h4>
        <small><?= h($bookmark->url) ?></small>
        <?= $this->Text->autoParagraph($bookmark->description) ?>
    </article>
<?php endforeach; ?>
</section>
```

O CakePHP espera que os nossos templates sigam a convenção de nomenclatura onde o nome do template é a versão minúscula e grifada do nome da ação do controller.

Você pode perceber que fomos capazes de utilizar as variáveis `$tags` e `bookmarks` em nossa view. Quando usamos o método `set()` em nosso controller, automaticamente definimos variáveis específicas que devem ser enviadas para a view. A view vai tornar todas as variáveis passadas disponíveis nos templates como variáveis locais.

Em nossa view, usamos alguns dos *helpers* nativos do CakePHP. Helpers são usados para criar lógica reutilizável para a formatação de dados, a criação de HTML ou outra saída da view.

Agora você deve ser capaz de visitar a URL `/bookmarks/tagged/funny` e ver todas os bookmarks com a tag 'funny'.

Até agora, nós criamos uma aplicação básica para gerenciar bookmarks, tags e users. No entanto, todos podem ver as tags de toda a gente. No próximo capítulo, vamos implementar a autenticação e restringir os bookmarks visíveis para somente aqueles que pertencem ao usuário atual.

Agora vá a *Tutorial - Criando um Bookmarker - Parte 2* para continuar a construir sua aplicação ou `mergulhe na documentação` para saber mais sobre o que CakePHP pode fazer por você.

Tutorial - Criando um Bookmarker - Parte 2

Depois de terminar a *primeira parte deste tutorial*, você deve ter uma aplicação muito básica. Neste capítulo iremos adicionar autenticação e restringir as bookmarks para que cada usuário possa ver/modificar somente aquelas que possuam.

Adicionando login

No CakePHP, a autenticação é feita por *Components (Componentes)*. Os Components podem ser considerados como formas de criar pedaços reutilizáveis de código relacionado a controllers com uma característica específica ou conceito. Os components também podem ligar-se ao evento do ciclo de vida do controller e interagir com a sua aplicação. Para começar, vamos adicionar o AuthComponent a nossa aplicação. Nós vamos querer muito que cada método exija autenticação, por isso vamos acrescentar o *AuthComponent* em nosso AppController:

```
// Em src/Controller/AppController.php
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
    public function initialize()
    {
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authenticate' => [
                'Form' => [
                    'fields' => [
                        'username' => 'email',
                        'password' => 'password'
                    ]
                ]
            ],
            'loginAction' => [
                'controller' => 'Users',
                'action' => 'login'
            ]
        ]);

        // Permite a ação display, assim nosso pages controller
        // continua a funcionar.
        $this->Auth->allow(['display']);
    }
}
```

Acabamos de dizer ao CakePHP que queremos carregar os components `Flash` e `Auth`. Além disso, temos a configuração personalizada do AuthComponent, assim a nossa tabela users pode usar `email` como username. Agora, se você for a qualquer URL, você vai ser chutado para `/users/login`, que irá mostrar uma página de erro já que não escrevemos o código ainda. Então, vamos criar a ação de login:

```
// Em src/Controller/UsersController.php

public function login()
{
    if ($this->request->is('post')) {
        $user = $this->Auth->identify();
        if ($user) {
            $this->Auth->setUser($user);
            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Flash->error('Your username or password is incorrect.');
    }
}
```

E em **src/Template/Users/login.ctp** adicione o seguinte:

```
<h1>Login</h1>
<?= $this->Form->create() ?>
<?= $this->Form->input('email') ?>
<?= $this->Form->input('password') ?>
<?= $this->Form->button('Login') ?>
<?= $this->Form->end() ?>
```

Agora que temos um simples formulário de login, devemos ser capazes de efetuar login com um dos users que tenham senha criptografada.

Nota: Se nenhum de seus users tem senha criptografada, comente a linha `loadComponent('Auth')`. Então vá e edite o user, salvando uma nova senha para ele.

Agora você deve ser capaz de entrar. Se não, certifique-se que você está usando um user que tenha senha criptografada.

Adicionando logout

Agora que as pessoas podem efetuar o login, você provavelmente vai querer fornecer uma maneira de encerrar a sessão também. Mais uma vez, no `UsersController`, adicione o seguinte código:

```
public function logout()
{
    $this->Flash->success('You are now logged out.');
    return $this->redirect($this->Auth->logout());
}
```

Agora você pode visitar `/users/logout` para sair e ser enviado à página de login.

Ativando inscrições

Se você não estiver logado e tentar visitar / usuários / adicionar você vai ser expulso para a página de login. Devemos corrigir isso se quisermos que as pessoas se inscrevam em nossa aplicação. No UsersController adicione o seguinte:

```
public function beforeFilter(\Cake\Event\Event $event)
{
    $this->Auth->allow(['add']);
}
```

O texto acima diz ao `AuthComponent` que a ação `add` não requer autenticação ou autorização. Você pode querer dedicar algum tempo para limpar a `/users/add` e remover os links enganosos, ou continuar para a próxima seção. Nós não estaremos construindo a edição do usuário, visualização ou listagem neste tutorial, então eles não vão funcionar, já que o `AuthComponent` vai negar-lhe acesso a essas ações do controller.

Restringindo acesso

Agora que os usuários podem conectar-se, nós vamos querer limitar os bookmarks que podem ver para aqueles que fizeram. Nós vamos fazer isso usando um adaptador de 'autorização'. Sendo os nossos requisitos bastante simples, podemos escrever um código em nossa `BookmarksController`. Mas antes de fazer isso, vamos querer dizer ao AuthComponent como nossa aplicação vai autorizar ações. Em seu `AppController` adicione o seguinte:

```
public function isAuthorized($user)
{
    return false;
}
```

Além disso, adicione o seguinte à configuração para `Auth` em seu `AppController`:

```
'authorize' => 'Controller',
```

Seu método `initialize` agora deve parecer com:

```
public function initialize()
{
    $this->loadComponent('Flash');
    $this->loadComponent('Auth', [
        'authorize'=> 'Controller',//added this line
        'authenticate' => [
            'Form' => [
                'fields' => [
                    'username' => 'email',
                    'password' => 'password'
                ]
            ]
        ],
        'loginAction' => [
            'controller' => 'Users',
            'action' => 'login'
        ],
        'unauthorizedRedirect' => $this->referer()
    ]);

    // Permite a ação display, assim nosso pages controller
    // continua a funcionar.
    $this->Auth->allow(['display']);
}
```

Vamos usar como padrão, negação do acesso, e de forma incremental conceder acesso onde faça sentido. Primeiro, vamos adicionar a lógica de autorização para os bookmarks. Em seu `BookmarksController` adicione o seguinte:

```
public function isAuthorized($user)
{
    $action = $this->request->params['action'];

    // As ações add e index são permitidas sempre.
    if (in_array($action, ['index', 'add', 'tags'])) {
```

```
            return true;
        }
        // Todas as outras ações requerem um id.
        if (empty($this->request->params['pass'][0])) {
            return false;
        }

        // Checa se o bookmark pertence ao user atual.
        $id = $this->request->params['pass'][0];
        $bookmark = $this->Bookmarks->get($id);
        if ($bookmark->user_id == $user['id']) {
            return true;
        }
        return parent::isAuthorized($user);
    }
```

Agora, se você tentar visualizar, editar ou excluir um bookmark que não pertença a você, você deve ser redirecionado para a página de onde veio. No entanto, não há nenhuma mensagem de erro sendo exibida, então vamos corrigir isso a seguir:

```
// In src/Template/Layout/default.ctp
// Under the existing flash message.
<?= $this->Flash->render('auth') ?>
```

Agora você deve ver as mensagens de erro de autorização.

Corrigindo a view de listagem e formulários

Enquanto view e delete estão trabalhando, edit, add e index tem alguns problemas:

1. Ao adicionar um bookmark, você pode escolher o user.
2. Ao editar um bookmark, você pode escolher o user.
3. A página de listagem mostra os bookmarks de outros users.

Vamos enfrentar o formulário de adição em primeiro lugar. Para começar remova o `input('user_id')` a partir de **src/Template/Bookmarks/add.ctp**. Com isso removido, nós também vamos atualizar o método add:

```
public function add()
{
    $bookmark = $this->Bookmarks->newEntity();
    if ($this->request->is('post')) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('The bookmark has been saved.');
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('The bookmark could not be saved. Please, try again.');
    }
    $tags = $this->Bookmarks->Tags->find('list');
```

```
    $this->set(compact('bookmark', 'tags'));
}
```

Ao definir a propriedade da entidade com os dados da sessão, nós removemos qualquer possibilidade do user modificar de que outro user um bookmark seja. Nós vamos fazer o mesmo para o formulário edit e action edit. Sua ação edit deve ficar assim:

```
public function edit($id = null)
{
    $bookmark = $this->Bookmarks->get($id, [
        'contain' => ['Tags']
    ]);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('The bookmark has been saved.');
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('The bookmark could not be saved. Please, try again.');
    }
    $tags = $this->Bookmarks->Tags->find('list');
    $this->set(compact('bookmark', 'tags'));
}
```

View de listagem

Agora, nós precisamos apenas exibir bookmarks para o user logado. Nós podemos fazer isso ao atualizar a chamada para `paginate()`. Altere sua ação index:

```
public function index()
{
    $this->paginate = [
        'conditions' => [
            'Bookmarks.user_id' => $this->Auth->user('id'),
        ]
    ];
    $this->set('bookmarks', $this->paginate($this->Bookmarks));
}
```

Nós também devemos atualizar a action `tags()` e o método localizador relacionado, mas vamos deixar isso como um exercício para que você conclua por sí.

Melhorando a experiência com as tags

Agora, adicionar novas tags é um processo difícil, pois o `TagsController` proíbe todos os acessos. Em vez de permitir o acesso, podemos melhorar a interface do usuário para selecionar tags usando um campo de texto separado por vírgulas. Isso permitirá dar uma melhor experiência para os nossos usuários, e usar mais alguns grandes recursos no ORM.

Adicionando um campo computado

Porque nós queremos uma maneira simples de acessar as tags formatados para uma entidade, podemos adicionar um campo virtual/computado para a entidade. Em **src/Model/Entity/Bookmark.php** adicione o seguinte:

```
use Cake\Collection\Collection;

protected function _getTagString()
{
    if (isset($this->_properties['tag_string'])) {
        return $this->_properties['tag_string'];
    }
    if (empty($this->tags)) {
        return '';
    }
    $tags = new Collection($this->tags);
    $str = $tags->reduce(function ($string, $tag) {
        return $string . $tag->title . ', ';
    }, '');
    return trim($str, ', ');
}
```

Isso vai nos deixar acessar a propriedade computada `$bookmark->tag_string`. Vamos usar essa propriedade em inputs mais tarde. Lembre-se de adicionar a propriedade `tag_string` a lista `_accessible` em sua entidade.

Em **src/Model/Entity/Bookmark.php** adicione o `tag_string` ao `_accessible` desta forma:

```
protected $_accessible = [
    'user_id' => true,
    'title' => true,
    'description' => true,
    'url' => true,
    'user' => true,
    'tags' => true,
    'tag_string' => true,
];
```

Atualizando as views

Com a entidade atualizado, podemos adicionar uma nova entrada para as nossas tags. Nas views add e edit, substitua `tags._ids` pelo seguinte:

```
<?= $this->Form->input('tag_string', ['type' => 'text']) ?>
```

Persistindo a string tag

Agora que podemos ver as tags como uma string existente, vamos querer salvar os dados também. Por marcar o `tag_string` como acessível, o ORM irá copiar os dados do pedido em nossa entidade. Podemos

usar um método `beforeSave` para analisar a cadeia tag e encontrar/construir as entidades relacionadas. Adicione o seguinte em **src/Model/Table/BookmarksTable.php**:

```
public function beforeSave($event, $entity, $options)
{
    if ($entity->tag_string) {
        $entity->tags = $this->_buildTags($entity->tag_string);
    }
}

protected function _buildTags($tagString)
{
    $new = array_unique(array_map('trim', explode(',', $tagString)));
    $out = [];
    $query = $this->Tags->find()
        ->where(['Tags.title IN' => $new]);

    // Remove tags existentes da lista de novas tags.
    foreach ($query->extract('title') as $existing) {
        $index = array_search($existing, $new);
        if ($index !== false) {
            unset($new[$index]);
        }
    }
    // Adiciona tags existentes.
    foreach ($query as $tag) {
        $out[] = $tag;
    }
    // Adiciona novas tags.
    foreach ($new as $tag) {
        $out[] = $this->Tags->newEntity(['title' => $tag]);
    }
    return $out;
}
```

Embora esse código seja um pouco mais complicado do que o que temos feito até agora, ele ajuda a mostrar o quão poderosa a ORM do CakePHP é. Você pode facilmente manipular resultados da consulta usando os métodos de *Collections (Coleções)*, e lidar com situações em que você está criando entidades sob demanda com facilidade.

Terminando

Nós expandimos nossa aplicação bookmarker para lidar com situações de autenticação e controle de autorização/acesso básico. Nós também adicionamos algumas melhorias agradáveis à UX, aproveitando os recursos FormHelper e ORM.

Obrigado por dispor do seu tempo para explorar o CakePHP. Em seguida, você pode saber mais sobre o *Models (Modelos)*, ou você pode ler os `/topics`.

CAPÍTULO 3

3.0 - Guia de migração

Esta página resume as alterações do CakePHP 2.x e irá auxiliar na migração do seu projeto para a versão 3.0, e também será uma referência para atualizá-lo quanto às principais mudanças do branch 2.x. Certifique-se de ler também as outras páginas nesse guia para conhecer todas as novas funcionalidades e mudanças na API.

Requerimentos

- O CakePHP 3.x suporta o PHP 5.4.16 e acima.
- O CakePHP 3.x precisa da extensão mbstring.
- O CakePHP 3.x precisa da extensão intl.

Aviso: O CakePHP 3.0 não irá funcionar se você não atender aos requisitos acima.

Ferramenta de atualização

Enquanto este documento cobre todas as alterações e melhorias feitas no CakePHP 3.0, nós também criamos uma aplicação de console para ajudar você a completar mais facilmente algumas das alterações mecânicas que consomem tempo. Você pode pegar a ferramenta de atualização no github[1].

Layout do diretório da aplicação

O Layout do diretório da aplicação mudou e agora segue o PSR-4[2]. Você deve usar o projeto do esqueleto da aplicação[3] como um ponto de referência quando atualizar sua aplicação.

[1] https://github.com/cakephp/upgrade
[2] http://www.php-fig.org/psr/psr-4/
[3] https://github.com/cakephp/app

O CakePHP deve ser instalado via Composer

Como o CakePHP não pode mais ser instalado facilmente via PEAR, ou em um diretório compartilhado, essas opções não são mais suportadas. Ao invés disso, você deve usar o Composer[4] para instalar o CakePHP em sua aplicação.

Namespaces

Todas as classes do core do CakePHP agora usam namespaces e seguem as especificações de autoload (autocarregamento) do PSR-4. Por exemplo **src/Cache/Cache.php** tem o namespace `Cake\Cache\Cache`. Constantes globais e métodos de helpers como `__()` e `debug()` não usam namespaces por questões de conveniência.

Constantes removidas

As seguintes constantes obsoletas foram removidas:

- `IMAGES`
- `CSS`
- `JS`
- `IMAGES_URL`
- `JS_URL`
- `CSS_URL`
- `DEFAULT_LANGUAGE`

Configuração

As configurações no CakePHP 3.0 estão significativamente diferentes que nas versões anteriores. Você deve ler a documentação *Configuração* para ver como a configuração é feita.

Você não pode mais usar o `App::build()` para configurar caminhos adicionais de classes. Ao invés disso, você deve mapear caminhos adicionais usando o autoloader da sua aplicação. Veja a seção *Caminhos de Classes Adicionais* para mais informações.

Três novas variáveis de configuração fornecem o caminho de configuração para plugins, views e arquivos de localização. Você pode adicionar vários caminhos em `App.paths.templates`, `App.paths.plugins`, `App.paths.locales` para configurar múltiplos caminhos para templates, plugins e arquivos de localização respectivamente.

A chave de configuração `www_root` mudou para `wwwRoot` devido a consistência. Por favor, ajuste seu arquivo de configuração `app.php` assim como qualquer uso de `Configure::read('App.wwwRoot')`.

[4] http://getcomposer.org

Novo ORM

O CakePHP 3.0 possui um novo ORM que foi refeito do zero. O novo ORM é significativamente diferente e incompatível com o anterior. Migrar para o novo ORM necessita de alterações extensas em qualquer aplicação que esteja sendo atualizada. Veja a nova documentação *Models (Modelos)* para informações de como usar o novo ORM.

Básico

- O `LogError()` foi removido, ele não tinha vantagens e era raramente ou mesmo, nunca usado.
- As seguintes funções globais foram removidas: `config()`, `cache()`, `clearCache()`, `convertSlashes()`, `am()`, `fileExistsInPath()`, `sortByKey()`.

Debug

- A função `Configure::write('debug', $bool)` não suporta mais 0/1/2. Um boleano simples é usado para mudar o modo de debug para ligado ou desligado.

Especificações/Configurações de objetos

- Os objetos usados no CakePHP agora tem um sistema consistente de armazenamento/recuperação de configuração-de-instância. Os códigos que anteriormente acessavam, por exemplo `$object->settings`, devem ser atualizados para usar `$object->config()` alternativamente.

Cache

- `Memcache` foi removido, use `Cake\Cache\Cache\Engine\Memcached` alternativamente.
- Cache engines são carregados sob demanda no primeiro uso.
- `Cake\Cache\Cache::engine()` foi adicionado.
- `Cake\Cache\Cache::enabled()` foi adicionado. Substituindo a opção de configuração `Cache.disable`.
- `Cake\Cache\Cache::enable()` foi adicionado.
- `Cake\Cache\Cache::disable()` foi adicionado.
- Configuração de cache agora é imutável. Se você precisa alterar a configuração, será necessário desfazer-se da configuração e recriá-la. Isso previne problemas de sincronização com as opções de configuração.

- `Cache::set()` foi removido. É recomendado criar múltiplas configurações de cache para substituir ajustes de configuração em tempo de execução anteriormente possíveis com `Cache::set()`.
- Todas as subclasses `CacheEngine` agora implementam um método `config()`.
- `Cake\Cache\Cache::readMany()`, `Cake\Cache\Cache::deleteMany()`, e `Cake\Cache\Cache::writeMany()` foram adicionados.

Todos os métodos `Cake\Cache\Cache\CacheEngine` agora são responsáveis por manipular o prefixo chave configurado. O `Cake\Cache\CacheEngine::write()` não mais permite definir a duração na escrita, a duração é captada pela configuração de tempo de execução do mecanismo de cache. Chamar um método cache com uma chuva vazia irá lançar uma `InvalidArgumentException` ao invés de retornar `false`.

Core

App

- `App::pluginPath()` foi removido. Use `CakePlugin::path()` alternativamente.
- `App::build()` foi removido.
- `App::location()` foi removido.
- `App::paths()` foi removido.
- `App::load()` foi removido.
- `App::objects()` foi removido.
- `App::RESET` foi removido.
- `App::APPEND` foi removido.
- `App::PREPEND` foi removido.
- `App::REGISTER` foi removido.

Plugin

- O `Cake\Core\Plugin::load()` não configura a carga automática a menos que você defina a opção `autoload` como `true`.
- Quanto estiver carregando plugins você não pode mais fornecer um `callable`.
- Quanto estiver carregando plugins você não pode mais fornecer um array de arquivos de configuração para carregar.

Configure

- O `Cake\Configure\PhpReader` foi renomeado para `Cake\Core\Configure\EnginePhpConfig`

- O `Cake\Configure\IniReader` foi renomeado para `Cake\Core\Configure\EngineIniConfig`
- O `Cake\Configure\ConfigReaderInterface` foi renomeado para `Cake\Core\Configure\ConfigEngineInterface`
- O `Cake\Core\Configure::consume()` foi adicionado.
- O `Cake\Core\Configure::load()` agora espera o nome de arquivo sem o sufixo de extensão como isso pode ser derivado do mecanismo. Ex.: para usar o PhpConfig use `app` para carregar `app.php`.
- Definir uma variável `$config` no arquivo PHP config está obsoleto. `Cake\Core\Configure\EnginePhpConfig` agora espera que o arquivo de configuração retorne um array.
- Um novo mecanismo de configuração `Cake\Core\Configure\EngineJsonConfig` foi adicionado.

Object

A classe `Object` foi removida. Ela anteriormente continha um monte de métodos que eram utilizados em vários locais no framework. O mais útil destes métodos foi extraido como um `trait`. Você pode usar o `Cake\Log\LogTrait` para acessar o método `log()`. O `Cake\Routing\RequestActionTrait` fornece o método `requestAction()`.

Console

O executável `cake` foi movido do diretório `app/Console` para o diretório `bin` dentro do esqueleto da aplicação. Você pode agora invocar o console do CakePHP com `bin/cake`.

TaskCollection Substituído

Essa classe foi renomeada para `Cake\Console\TaskRegistry`. Veja a seção em *Objetos de Registro* para mais informações sobre funcionalidades fornecidas pela nova classe. Você pode usar o `cake upgrade rename_collections` para ajuda ao atualizar seu código. Tarefas não tem mais acesso a callbacks, como nunca houve nenhum callback para se usar.

Shell

- O `Shell::__construct()` foi alterado. Ele agora usa uma instância de `Cake\Console\ConsoleIo`.
- O `Shell::param()` foi adicionado como um acesso conveniente aos parâmetros.

Adicionalmente todos os métodos shell serão transformados em camel case quando invocados. Por exemplo, se você tem um método `hello_world()` dentro de um shell e chama ele com `bin/cake my_shell hello_world`, você terá que renomear o método para `helloWorld`. Não há necessidade de mudanças no modo que você chama os métodos/comandos.

ConsoleOptionParser

- O `ConsoleOptionParser::merge()` foi adicionado para mesclar os parsers.

ConsoleInputArgument

- O `ConsoleInputArgument::isEqualTo()` foi adicionado para comparar dois argumentos.

Shell / Tarefa

Os Shells e Tarefas foram movidas de `Console/Command` e `Console/Command/Task` para `Shell` e `Shell/Task`, respectivamente.

ApiShell Removido

O ApiShell foi removido pois ele não fornecia nenhum beneficio além do próprio arquivo fonte e da documentação/API[5] online.

SchemaShell Removido

O SchemaShell foi removido como ele nunca foi uma implementação completa de migração de banco de dados e surgiram ferramentas melhores como o Phinx[6]. Ele foi substituído pelo CakePHP Migrations Plugin[7] que funciona como um empacotamento entre o CakePHP e o Phinx[8].

ExtractTask

- O `bin/cake i18n extract` não inclui mais mensagens de validação sem tradução. Se você quiser mensagens de validação traduzidas você deve encapsula-las com chamadas *__()* como qualquer outro conteúdo.

BakeShell / TemplateTask

- O Bake não faz mais parte do fonte do núcleo e é suplantado pelo CakePHP Bake Plugin[9]
- Os templates do Bake foram movidos para **src/Template/Bake**.
- A sintaxe dos templates do Bake agora usam tags estilo erb (`<% %>`) para denotar lógica de template, permitindo código php ser tratado como texto plano.
- O comando `bake view` foi renomeado para `bake template`.

[5] http://api.cakephp.org/
[6] https://phinx.org/
[7] https://github.com/cakephp/migrations
[8] https://phinx.org/
[9] https://github.com/cakephp/bake

Eventos

O método `getEventManager()`, foi removido de todos os objetos que continham. Um método `eventManager()` é agora fornecido pelo `EventManagerTrait`. O `EventManagerTrait` contém a lógica de instanciação e manutenção de uma referência para um gerenciador local de eventos.

O subsistema `Event` teve um monte de funcionalidades opcionais removidas. Quando despachar eventos você não poderá mais usar as seguintes opções:

- `passParams` Essa opção está agora ativada sempre implicitamente. Você não pode desliga-la.
- `break` Essa opção foi removida. Você deve agora parar os eventos.
- `breakOn` Essa opção foi removida. Você deve agora parar os eventos.

Log

- As configurações do Log agora não imutáveis. Se você precisa alterar a configuração você deve primeiro derrubar a configuração e então recria-la. Isso previne problemas de sincronização com opções de configuração.
- Os mecanismos de Log agora são carregados tardiamente após a primeira escrita nos logs.
- O `Cake\Log\Log::engine()` foi adicionado.
- Os seguintes métodos foram removidos de `Cake\Log\Log` :: `defaultLevels()`, `enabled()`, `enable()`, `disable()`.
- Você não pode mais criar níveis personalizados usando `Log::levels()`.
- Quando configurar os loggers você deve usar `'levels'` ao invés de `'types'`.
- Você não pode mais especificar níveis personalizados de log. Você deve usar o conjunto padrão de níveis de log. Você deve usar escopos de log para criar arquivos de log personalizados ou manipulações específicas para diferentes seções de sua aplicação. Usando um nível de log não padrão irá lançar uma exceção.
- O `Cake\Log\LogTrait` foi adicionado. Você pode usar este trait em suas classes para adicionar o método `log()`.
- O escopo de log passado para `Cake\Log\Log::write()` é agora encaminhado para o método `write()` dos mecanismos de log de maneira a fornecer um melhor contexto para os mecanismos.
- Os mecanismos de Log agora são necessários para implementar `Psr\Log\LogInterface` invés do próprio `LogInterface` do Cake. Em geral, se você herdou o `Cake\Log\Engine\BaseEngine` você só precisa renomear o método `write()` para `log()`.
- O `Cake\Log\Engine\FileLog` agora grava arquivos em `ROOT/logs` no lugar de `ROOT/tmp/logs`.

Roteamento

Parâmetros Nomeados

Os parâmetros nomeados foram removidos no 3.0. Os parâmetros nomeados foram adicionados no 1.2.0 como uma versão 'bonita' de parâmetros de requisição. Enquanto o benefício visual é discutível, os problemas criados pelos parâmetros nomeados não são.

Os parâmetros nomeados necessitam manipulação especial no CakePHP assim como em qualquer biblioteca PHP ou JavaScript que necessite interagir com eles, os parâmetros nomeados não são implementados ou entendidos por qualquer biblioteca *exceto* o CakePHP. A complexidade adicionada e o código necessário para dar suporte aos parâmetros nomeados não justificam a sua existência, e eles foram removidos. No lugar deles, você deve agora usar o padrão de parâmetros de requisição (querystring) ou argumentos passados configurados nas rotas. Por padrão o `Router` irá tratar qualquer parâmetro adicional ao `Router::url()` como argumentos de requisição.

Como muitas aplicações ainda precisarão analisar URLs contendo parâmetros nomeados, o `Cake\Routing\Router::parseNamedParams()` foi adicionado para permitir compatibilidade com URLs existentes.

RequestActionTrait

- O `Cake\Routing\RequestActionTrait::requestAction()` teve algumas de suas opções extras alteradas:
  - o `options[url]` é agora `options[query]`.
  - o `options[data]` é agora `options[post]`.
  - os parâmetros nomeados não são mais suportados.

Roteador

- Os parâmetros nomeados foram removidos, veja acima para mais informações.
- A opção `full_base` foi substituída com a opção `_full`.
- A opção `ext` foi substituída com a opção `_ext`.
- As opções `_scheme`, `_port`, `_host`, `_base`, `_full`, `_ext` foram adicionadas.
- As URLs em strings não são mais modificados pela adição de plugin/controller/nomes de prefixo.
- A manipulação da rota padrão de `fallback` foi removida. Se nenhuma rota combinar com o conjunto de parâmetros, o `/` será retornado.
- As classes de rota são responsáveis por *toda* geração de URLs incluindo parâmetros de requisição (query string). Isso faz com que as rotas sejam muito mais poderosas e flexíveis.
- Parâmetros persistentes foram removidos. Eles foram substituídos pelo `Cake\Routing\Router::urlFilter()` que permite um jeito mais flexível para mudar URLs sendo roteadas reversamente.

- O `Router::parseExtensions()` foi removido. Use o `Cake\Routing\Router::extensions()` no lugar. Esse método **deve** ser chamado antes das rotas serem conectadas. Ele não irá modificar rotas existentes.
- O `Router::setExtensions()` foi removido. Use o `Cake\Routing\Router::extensions()` no lugar.
- O `Router::resourceMap()` foi removido.
- A opção `[method]` foi renomeada para `_method`.
- A habilidade de combinar cabeçalhos arbitrários com parâmetros no estilo `[]` foi removida. Se você precisar combinar/analisar em condições arbitrárias considere usar classes personalizadas de roteamento.
- O `Router::promote()` foi removido.
- O `Router::parse()` irá agora lançar uma exceção quando uma URL não puder ser atendida por nenhuma rota.
- O `Router::url()` agora irá lançar uma exceção quando nenhuma rota combinar com um conjunto de parâmetros.
- Os escopos de rotas foram adicionados. Escopos de rotas permitem você manter seu arquivo de rotas limpo e dar dicas de rotas em como otimizar análise e reversão de rotas de URL.

Route

- O `CakeRoute` foi renomeado para `Route`.
- A assinatura de `match()` mudou para `match($url, $context = [])`. Veja `Cake\Routing\Route::match()` para mais informações sobre a nova assinatura.

Configuração de Filtros do Despachante Mudaram

Os filtros do despachante não são mais adicionados em sua aplicação usando o `Configure`. Você deve agora anexa-los com `Cake\Routing\DispatcherFactory`. Isso significa que sua aplicação usava `Dispatcher.filters`, você deve usar agora o método `Cake\Routing\DispatcherFactory::add()`.

Além das mudanças de configuração, os filtros do despachante tiveram algumas convenções atualizadas e novas funcionalidades. Veja a documentação em *Filtros do Dispatcher* para mais informações.

FilterAssetFilter

- Os itens de plugins e temas manipulados pelo AssetFilter não são mais lidos via `include`, ao invés disso eles são tratados como arquivos de texto plano. Isso corrige um número de problemas com bibliotecas javascript como TinyMCE e ambientes com short_tags ativadas.
- O suporte para a configuração `Asset.filter` e ganchos foram removidos. Essa funcionalidade pode ser facilmente substituída com um plugin ou filtro de despachante.

Rede

Requisição

- O `CakeRequest` foi renomeada para `Cake\Network\Request`.
- O `Cake\Network\Request::port()` foi adicionado.
- O `Cake\Network\Request::scheme()` foi adicionado.
- O `Cake\Network\Request::cookie()` foi adicionado.
- O `Cake\Network\Request::$trustProxy` foi adicionado. Isso torna mais fácil colocar aplicações CakePHP atrás de balanceadores de carga.
- O `Cake\Network\Request::$data` não é mais mesclado com a chave de dados prefixada, pois esse prefixo foi removido.
- O `Cake\Network\Request::env()` foi adicionado.
- O `Cake\Network\Request::acceptLanguage()` mudou de um método estático para não-estático.
- O detector de requisição para dispositivos móveis foi removido do núcleo. Agora o app template adiciona detectores para dispositivos móveis usando a biblioteca `MobileDetect`.
- O método `onlyAllow()` foi renomeado para `allowMethod()` e não aceita mais "argumentos var". Todos os nomes de métodos precisam ser passados como primeiro argumento, seja como string ou como array de strings.

Resposta

- O mapeamento do mimetype `text/plain` para extensão `csv` foi removido. Como consequência o `Cake\Controller\Component\RequestHandlerComponent` não define a extensão para `csv` se o cabeçalho `Accept` tiver o mimetype `text/plain` que era um problema comum quando recebia uma requisição XHR do jQuery.

Sessões

A classe de sessão não é mais estática, agora a sessão (session) pode ser acessada através do objeto de requisição (request). Veja a documentação em *Sessions* para ver como usar o objeto de sessão.

- O `Cake\Network\Session` e classes de sessão relacionadas foram movidas para o namespace `Cake\Network`.
- O `SessionHandlerInterface` foi removido em favor ao fornecido pelo próprio PHP.
- A propriedade `Session::$requestCountdown` foi removida.
- O funcionalidade de sessão `checkAgent` foi removida. Ela causava um monte de bugs quando quadros do chrome e o flash player estavam envolvidos.

- A convenção de nome para a tabela de sessão no banco de dados agora é `sessions` ao invés de `cake_sessions`.
- O cookie de tempo limite da sessão é atualizado automaticamente em conjunto com o tempo limite dos dados de sessão.
- O caminho padrão para o cookie de sessão agora é o caminho base da aplicação, ao invés de "/". Além disso, uma nova variável de configuração `Session.cookiePath` foi adicionada para facilitar a personalização do caminho para os cookies.
- Um novo método conveniente `Cake\Network\Session::consume()` foi adicionado para permitir a leitura e exclusão de dados de sessão em um único passo.
- O valor padrão do argumento `$renew` de `Cake\Network\Session::clear()` mudou de `true` para `false`.

Network\Http

- O `HttpSocket` agora é `Cake\Network\Http\Client`.
- O HttpClient foi reescrito do zero. Ele tem uma API mais simples/fácil de usar, suporta novos sistemas de autenticação como OAuth, e uploads de arquivos. Ele usa as API de stream do PHP de modo que não há requerimentp para o cURL. Veja a documentação *Http Client* para mais informações.

Network\Email

- O `Cake\Network\Email\Email::config()` agora é usado para definir perfis de configuração. Isso substitui as classes `EmailConfig` nas versões anteriores.
- O `Cake\Network\Email\Email::profile()` substitui o `config()` como modo de modificar opções de configuração por instância.
- O `Cake\Network\Email\Email::drop()` foi adicionado para permitir a remoção de configurações de email.
- O `Cake\Network\Email\Email::configTransport()` foi adicionado para permitir a definição de configurações de transporte. Essa mudança retira as opções de transporte dos perfis de entrega e permite a você reusar facilmente os transportes através de perfis de e-mails.
- O `Cake\Network\Email\Email::dropTransport()` foi adicionado para permitir a remoção de configurações de transporte.

Controller

Controller

- As propriedades `$helpers` e `$components` agora estão mescladas com **todas** classes pai, não apenas a `AppController` e o plugin de AppController. As propriedades são mescladas de modo

diferente agora também. No lugar de todas as configurações em todas as classes serem mescladas juntas, as configurações definidas nas classes filho serão usadas. Isso quer dizer que se você tem alguma configurações definida no seu AppController, e alguma configuração definida em uma a subclasse, apenas a configuração na subclasse será usada.

- O `Controller::httpCodes()` foi removido, use o `Cake\Network\Response::httpCodes()` no lugar.
- O `Controller::disableCache()` foi removido, use o `Cake\Network\Response::disableCache()` no lugar.
- O `Controller::flash()` foi removido. Esse método era raramente usado em aplicações reais e não tinha mais propósito algum.
- O `Controller::validate()` e `Controller::validationErrors()` foram removidos. Eles eram restos dos dias do 1.x onde as preocupações com os models + controllers eram muito mais entrelaçados.
- O `Controller::loadModel()` agora carrega uma tabela de objetos.
- A propriedade `Controller::$scaffold` foi removida. O scaffolding dinâmico foi removido do núcleo do CakePHP. Um plugin de scaffolding melhorado, chamado CRUD, pode ser encontrado em: https://github.com/FriendsOfCake/crud
- A propriedade `Controller::$ext` foi removida. Você deve agora estender e sobrescrever a propriedade `View::$_ext` se você deseja usar uma extensão de arquivo de visão não padrão.
- A propriedade `Controller::$methods` foi removida. Você deve usar o `Controller::isAction()` para determinar quando ou não um nome de método é uma ação. Essa mudança foi feita para permitir personalizações mais fáceis do que vai contar ou não como uma ação.
- A propriedade `Controller::$Components` foi removida e substituída pelo `_components`. Se você precisar carregar componentes em tempo de execução você deve usar o `$this->loadComponent()` em seu controller.
- A assinatura do `Cake\Controller\Controller::redirect()` mudou para `Controller::redirect(string|array $url, int $status = null)`. O terceiro argumento `$exit` foi removido. O método não pode mais enviar resposta e sair do script, no lugar ele retorna uma instância de `Response` com os cabeçalhos apropriados definidos.
- As propriedades mágicas `base`, `webroot`, `here`, `data`, `action`, e `params` foram removidas. Você deve acessar todas essas propriedades em `$this->request` no lugar.
- Métodos de controlar prefixados com sublinhado como `_someMethod()` não são mais tratados como métodos privados. Use as palavras chaves de visibilidade apropriadas no lugar. Somente métodos públicos podem ser usados como ação de controllers.

Scaffold Removido

O scaffolding dinâmico no CakePHP foi removido do núcleo do CakePHP. Ele não era usado com frequência, e não era voltado para uso em produção. Um plugin melhorado de scaffolding, chamado CRUD, pode ser encontrado em: https://github.com/FriendsOfCake/crud

ComponentCollection Substituído

Essa classe foi renomeada para `Cake\Controller\ComponentRegistry`. Veja a seção em *Objetos de Registro* para mais informações sobre as funcionalidades fornecidas pela nova classe. Você pode usar o `cake upgrade rename_collections` para ajudar você a atualizar o seu código.

Components

- A propriedade `_Collection` é agora `_registry`. Ela contém uma instância do `Cake\Controller\ComponentRegistry` agora.
- Todos components devem agora usar o método `config()` para obter/definir configurações.
- A configuração padrão para components deve ser definido na propriedade `$_defaultConfig`. Essa propriedade é automaticamente mesclada com qualquer configuração fornecida pelo construtor.
- Opções de configuração não são mais definidas como propriedades públicas.
- O método `Component::initialize()` não é mais um `event listener` (ouvinte de eventos). Ao invés disso, ele é um gancho pós-construtor como o `Table::initialize()` e `Controller::initialize()`. O novo método `Component::beforeFilter()` é ligado ao mesmo evento que o `Component::initialize()` costumava ser. O método de inicialização deve ter a seguinte assinatura `initialize(array $config)`.

Controller\Components

CookieComponent

- Ele usa o `Cake\Network\Request::cookie()` para ler os dados de cookies, isso facilita os testes, e permite o ControllerTestCase definir os cookies.
- Os Cookies encriptados pelas versões anteriores do CakePHP usando o método `cipher()`, agora não podem ser lidos, pois o `Security::cipher()` foi removido. Você precisará reencriptar os cookies com o método `rijndael()` ou `aes()` antes de atualizar.
- O `CookieComponent::type()` foi removido e substituído com dados de configuração acessados através de `config()`.
- O `write()` não aceita mais os parâmetros `encryption` ou `expires`. Os dois agora são gerenciados através de dados de configuração. Veja *CookieComponent* para mais informações.
- O caminho padrão para os cookies agora é o caminho base da aplicação, ao invés de "/".

AuthComponent

- O `Default` é agora o hasher de senhas padrão usado pelas classes de autenticação. Ele usa exclusivamente o algoritmo de hash bcrypt. Se você desejar continuar usando o hash SHA1 usado no 2.x, use `'passwordHasher' => 'Weak'` nas configurações de seu autenticador.

- O novo `FallbackPasswordHasher` foi adicionado para ajudar os usuários migrar senhas antigas de um algoritmo para o outro. Veja a documentação do AuthComponent para mais informações.
- A classe `BlowfishAuthenticate` foi removida. Apenas use `FormAuthenticate`.
- A classe `BlowfishPasswordHasher` foi removida. Use o `DefaultPasswordHasher` no lugar.
- O método `loggedIn()` foi removido. Use o `user()` no lugar.
- As opções de configuração não são mais definidas como propriedades públicas.
- Os métodos `allow()` e `deny()` não aceitam mais "var args". Todos os nomes de métodos precisam ser passados como primeiro argumento, seja como string ou array de strings.
- O método `login()` foi removido e substituído por `setUser()`. Para logar um usuário agora você deve chamar `identify()` que retorna as informações do usuário caso identificado com sucesso e então usar `setUser()` para salvar as informações na sessão de maneira persistente entre as requisições.
- O `BaseAuthenticate::_password()` foi removido. Use a classe `PasswordHasher` no lugar.
- O `BaseAuthenticate::logout()` foi removido.
- O `AuthComponent` agora dispara dois eventos `Auth.afterIdentify` e `Auth.logout` após um usuário ser identificado e antes de um usuário ser deslogado respectivamente. Você pode definir funções de callback para esses eventos retornando um array mapeado no método `implementedEvents()` de sua classe de autenticação.

Classes relacionadas a ACL foram movidas para um plugin separado. Hashers de senha, fornecedores de Autenticação e Autorização foram movidos para o namespace `\Cake\Auth`. Você DEVE mover seus fornecedores e hashers para o namespace `App\Auth` também.

RequestHandlerComponent

- Os seguintes métodos foram removidos do componente RequestHandler: `isAjax()`, `isFlash()`, `isSSL()`, `isPut()`, `isPost()`, `isGet()`, `isDelete()`. Use o método `Cake\Network\Request::is()` no lugar com o argumento relevante.
- O `RequestHandler::setContent()` foi removido, use `Cake\Network\Response::type()` no lugar.
- O `RequestHandler::getReferer()` foi removido, use `Cake\Network\Request::referer()` no lugar.
- O `RequestHandler::getClientIP()` foi removido, use `Cake\Network\Request::clientIp()` no lugar.
- O `RequestHandler::getAjaxVersion()` foi removido.
- O `RequestHandler::mapType()` foi removido, use `Cake\Network\Response::mapType()` no lugar.
- As opções de configuração não são mais definidas como propriedades públicas.

SecurityComponent

- Os seguintes métodos e as propriedades relacionadas foram removidas do componente Security: `requirePost()`, `requireGet()`, `requirePut()`, `requireDelete()`. Use o `Cake\Network\Request::allowMethod()` no lugar.
- `SecurityComponent::$disabledFields()` foi removido, use o `SecurityComponent::$unlockedFields()`.
- As funções relacionadas ao CSRF no SecurityComponent foram extraídas e movidas em separado no CsrfComponent. Isso permite que você use a proteção CSRF facilmente sem ter que usar prevenção de adulteração de formulários.
- As opções de configuração não são mais definidas como propriedades públicas.
- Os métodos `requireAuth()` e `requireSecure()` não aceitam mais "var args". Todos os nomes de métodos precisam ser passados como primeiro argumento, seja como string ou array de strings.

SessionComponent

- O `SessionComponent::setFlash()` está obsoleto. Você deve usar o *Flash* no lugar.

Error

ExceptionRenderers personalizados agora espera-se que retornem ou um objeto `Cake\Network\Response` ou uma string quando renderizando erros. Isso significa que qualquer método que manipule exceções específicas devem retornar uma resposta ou valor de string.

Model

A camada de model do 2.x foi completamente reescrita e substituída. Você deve revisar o *Guia de atualização para o novo ORM* para saber como usar o novo ORM.

- A classe `Model` foi removida.
- A classe `BehaviorCollection` foi removida.
- A classe `DboSource` foi removida.
- A classe `Datasource` foi removida.
- As várias classes de fonte de dados foram removidas.

ConnectionManager

- O ConnectionManager (gerenciador de conexão) foi movido para o namespace `Cake\Datasource`.

- O ConnectionManager teve os seguintes métodos removidos:
  - `sourceList`
  - `getSourceName`
  - `loadDataSource`
  - `enumConnectionObjects`
- O `Database\ConnectionManager::config()` foi adicionado e é agora o único jeito de configurar conexões.
- O `Database\ConnectionManager::get()` foi adicionado. Ele substitui o `getDataSource()`.
- O `Database\ConnectionManager::configured()` foi adicionado. Ele junto com `config()` substitui o `sourceList()` e `enumConnectionObjects()` com uma API mais padrão e consistente.
- O `ConnectionManager::create()` foi removido. Ele pode ser substituído por `config($name, $config)` e `get($name)`.

Behaviors

- Os métodos de comportamentos (behaviors) prefixados com sublinhado como `_someMethod()` não são mais tratados como métodos privados. Use as palavras chaves de visibilidade.

TreeBehavior

O TreeBehavior foi completamente reescrito para usar o novo ORM. Embora ele funcione do mesmo modo que no 2.x, alguns métodos foram renomeados ou removidos:

- `TreeBehavior::children()` é agora uma busca personalizada `find('children')`.
- `TreeBehavior::generateTreeList()` é agora uma busca personalizada `find('treeList')`.
- `TreeBehavior::getParentNode()` foi removido.
- `TreeBehavior::getPath()` é agora uma busca personalizada `find('path')`.
- `TreeBehavior::reorder()` foi removido.
- `TreeBehavior::verify()` foi removido.

Suíte de Testes

Casos de Teste

- O `_normalizePath()` foi adicionado para permitir testes de comparação de caminhos para executar em todos os sistemas operacionais, independente de sua configuração (`\` no Windows vs `/` no

UNIX, por exemplo).

Os seguintes métodos de asserção foram removidos já que eles estavam há muito obsoletos e foram substituídos pelo seu equivalente no PHPUnit:

- `assertEqual()` é substituído por `assertEquals()`
- `assertNotEqual()` é substituído por `assertNotEquals()`
- `assertIdentical()` é substituído por `assertSame()`
- `assertNotIdentical()` é substituído por `assertNotSame()`
- `assertPattern()` é substituído por `assertRegExp()`
- `assertNoPattern()` é substituído por `assertNotRegExp()`
- `assertReference()` é substituído por `assertSame()`
- `assertIsA()` é substituído por `assertInstanceOf()`

Note que alguns métodos tiveram a ordem dos argumentos trocada, ex. `assertEqual($is, $expected)` deve ser agora `assertEquals($expected, $is)`.

Os seguintes métodos de asserção estão obsoletos e serão removidos no futuro:

- `assertWithinMargin()` é substituído por `assertWithinRange()`
- `assertTags()` é substituído por `assertHtml()`

Em ambas as substituições dos métodos também mudaram a ordem dos argumentos para manter a consistência na API com `$expected` como primeiro argumento.

Os seguintes métodos de asserção foram adicionados:

- `assertNotWithinRange()` em contrapartida ao `assertWithinRange()`

View

Temas são agora Plugins Básicos

Ter os temas e plugins de modo a criar components modulares da aplicação se provou limitado e confuso. No CakePHP 3.0, temas não residem mais **dentro** da aplicação. Ao invés disso, eles são plugins independentes. Isso resolveu alguns problemas com temas:

- Você não podia colocar temas *nos* plugins.
- Temas não podiam fornecer helpers (helpers), ou classes de visão personalizadas.

Esses dois problemas foram resolvidos ao converter os temas em plugins.

Pasta das views renomeada

As pastas contendo os arquivos de views agora ficam em **src/Template** no lugar de **src/View**. Isso foi feito para separar os arquivos de visão dos arquivos contendo classes php. (ex. helpers, Classes de visão).

As seguintes pastas de Visão foram renomeadas para evitar colisão de nomes com nomes de controllers:

- `Layouts` agora é `Layout`
- `Elements` agora é `Element`
- `Errors` agora é `Error`
- `Emails` agora é `Email` (o mesmo para `Email` dentro de `Layout`)

Coleção de Helpers Substituída

Essa classe foi renomeada para `Cake\View\HelperRegistry`. Veja a seção em *Objetos de Registro* para mais informações sobre as funcionalidades fornecidas pela nova classe. Você pode usar o `cake upgrade rename_collections` para ajudar você a atualizar seu código.

Classe View

- A chave `plugin` foi removida do argumento `$options` de `Cake\View\View::element()`. Especifique o nome do elemento como `AlgumPlugin.nome_do_elemento` no lugar.
- O `View::getVar()` foi removido, use o `Cake\View\View::get()` no lugar.
- O `View::$ext` foi removido e no lugar uma propriedade protegida `View::$_ext` foi adicionada.
- O `View::addScript()` foi removido. Use o *Using View Blocks* no lugar.
- As propriedades mágicas `base`, `webroot`, `here`, `data`, `action`, e `params` foram removidas. Ao invés disso, você deve acessar todas essas propriedades no `$this->request`.
- O `View::start()` não se liga mais a um bloco existente. Ao invés disso ele irá sobrescrever o conteúdo do bloco quando o `end()` for chamado. Se você precisa combinar o conteúdo de um bloco você deverá buscar o conteúdo do bloco quando chamar o start uma segunda vez, ou usar o modo de captura de `append()`.
- O `View::prepend()` não tem mais um modo de captura.
- O `View::startIfEmpty()` foi removido. Agora que o start() sempre sobrescreve, o startIfEmpty não tem mais propósito.
- A propriedade `View::$Helpers` foi removida e substituída com `_helpers`. Se você precisar carregar helpers em tempo de execução você deve usar o `$this->addHelper()` em seus arquivos de visão.
- O `View` agora irá lançar `Cake\View\Exception\MissingTemplateException` quando templates estiverem faltando, ao invés de `MissingViewException`.

ViewBlock

- O `ViewBlock::append()` foi removido, use o `Cake\ViewViewBlock::concat()` no lugar. Entretanto o `View::append()` ainda existe.

JsonView

- Agora os dados JSON terão as entidades HTML codificadas por padrão. Isso previne possíveis problemas de XSS quando o conteúdo de visão JSON está encapsulado em arquivos HTML.
- O `Cake\View\JsonView` agora suporta a variável de visão `_jsonOptions`. Isso permite a você configurar as opções de máscara de bits usadas ao gerar JSON.

XmlView

- A `Cake\View\XmlView` agora suporta a variável de visão `_xmlOptions`. Isso permite a você configurar as opções usadas quando gerar XML.

View\Helper

- A propriedade `$settings` é agora chamada `$_config` e deve ser acessada através do método `config()`.
- As opções de configuração não são mais definidas como propriedades públicas.
- O `Helper::clean()` foi removido. Ele nunca foi robusto o suficiente para prevenir completamente XSS. Ao invés disso você deve escapar o conteúdo com `h` ou ou usar uma biblioteca dedicada como o htmlPurifier.
- O `Helper::output()` foi removido. Esse método estava obsoleto no 2.x.
- Os métodos `Helper::webroot()`, `Helper::url()`, `Helper::assetUrl()`, `Helper::assetTimestamp()` foram movidos para o novo ajudante `Cake\View\Helper\UrlHelper`. O `Helper::url()` está agora disponível como `Cake\View\Helper\UrlHelper::build()`.
- Os Assessores Mágicos a propriedades obsoletas foram removidos. A seguinte propriedade agora deve ser acessada a partir do objeto de requisição:
  - base
  - here
  - webroot
  - data
  - action
  - params

Helpers

A classe Helper teve os seguintes métodos removidos:

- `Helper::setEntity()`

- `Helper::entity()`
- `Helper::model()`
- `Helper::field()`
- `Helper::value()`
- `Helper::_name()`
- `Helper::_initInputField()`
- `Helper::_selectedArray()`

Esses métodos eram partes usadas apenas pelo FormHelper, e parte de uma funcionalidade de persistência de campos que se mostrou problemática com o tempo. O FormHelper não precisa mais destes métodos e a complexidades que eles provêm não é mais necessária.

Os seguintes métodos foram removidos:

- `Helper::_parseAttributes()`
- `Helper::_formatAttribute()`

Esses métodos podem agora ser encontrados na classe `StringTemplate` que os helpers usam com frequência. Veja o `StringTemplateTrait` para um jeito fácil de integrar os templates de string em seus próprios helpers.

FormHelper

O FormHelper foi completamente reescrito para o 3.0. Ele teve algumas grandes mudanças:

- O FormHelper trabalha junto com o novo ORM. Mas também possui um sistema extensível para integrar com outros ORMs e fontes de dados.
- O FormHelper possui um sistema de widgets extensível que permite a você criar novos widgets de entrada personalizados e expandir facilmente aqueles inclusos no framework.
- Os Templates de String são a fundação deste ajudante. Ao invés de encher de arrays por toda parte, a maioria do HTML que o FormHelper gera pode ser personalizado em um lugar central usando conjuntos de templates.

Além dessas grandes mudanças, foram feitas algumas mudanças menores que causaram rompendo algumas coisas da versão anterior. Essas mudanças devem simplificar o HTML que o FormHelper gera e reduzir os problemas que as pessoas tinham no passado:

- O prefixo `data[` foi removido de todas as entradas geradas. O prefixo não tem mais propósito.
- Os vários métodos de entradas independentes, como `text()`, `select()` e outros, não geram mais atributos id.
- A opção `inputDefaults` foi removida de `create()`.
- As opções `default` e `onsubmit` do `create()` foram removidas. No lugar você deve usar JavaScript event binding ou definir todos os códigos js necessários para o `onsubmit`.
- O `end()` não gerará mais botões. Você deve criar botões com `button()` ou `submit()`.

- O `FormHelper::tagIsInvalid()` foi removido. Use `isFieldError()` no lugar.
- O `FormHelper::inputDefaults()` foi removido. Você pode usar `templates()` para definir/expandir os templates que o FormHelper usa.
- As opções `wrap` e `class` foram removidas do método `error()`.
- A opção `showParents` foi removida do select().
- As opções `div`, `before`, `after`, `between` e `errorMessage` foram removidas do `input()`. Você pode usar templates para atualizar o HTML envoltório. A opção `templates` permite você sobrescrever os templates carregados para uma entrada.
- As opções `separator`, `between`, e `legend` foram removidas do `radio()`. Você pode usar templates para mudar o HTML envoltório agora.
- O parâmetro `format24Hours` foi removido de `hour()`. Ele foi substituído pela opção `format`.
- Os parâmetros `minYear` e `maxYear` foram removidos do `year()`. Ambos podem ser fornecidos como opções.
- Os parâmetros `dateFormat` e `timeFormat` foram removidos do `datetime()`. Você pode usar o template para definir a ordem que as entradas devem ser exibidas.
- O `submit()` teve as opções `div`, `before` e `after` removidas. Você pode personalizar o template `submitContainer` para modificar esse conteúdo.
- O método `inputs()` não aceita mais `legend` e `fieldset` no parâmetro `$fields`, você deve usar o parâmetro `$options`. Ele também exige que o parâmetro `$fields` seja um array. O parâmetro `$blacklist` foi removido, a funcionalidade foi substituída pela especificação de `'field' => false` no parâmetro `$fields`.
- O parâmetro `inline` foi removido do método postLink(). Você deve usar a opção `block` no lugar. Definindo `block => true` irá emular o comportamento anterior.
- O parâmetro `timeFormat` para `hour()`, `time()` e `dateTime()` agora é 24 por padrão, em cumprimento ao ISO 8601.
- O argumento `$confirmMessage` de `Cake\View\Helper\FormHelper::postLink()` foi removido. Você deve usar agora a chave `confirm` no `$options` para especificar a mensagem.
- As entradas do tipo Checkbox e radio são agora renderizadas *dentro* de elementos do tipo label por padrão. Isso ajuda a aumentar a compatibilidade com bibliotecas CSS populares como Bootstrap[10] e Foundation[11].
- As tags de template agora são todas camelBacked (primeira letra minúscula e inicio de novas palavras em maiúsculo). As tags pré-3.0 `formstart`, `formend`, `hiddenblock` e `inputsubmit` são agora `formStart`, `formEnd`, `hiddenBlock` e `inputSubmit`. Certifique-se de altera-las se elas estiverem personalizando sua aplicação.

É recomendado que você revise a documentação *Form* para mais detalhes sobre como usar o FormHelper no 3.0.

[10] http://getbootstrap.com/
[11] http://foundation.zurb.com/

HtmlHelper

- O `HtmlHelper::useTag()` foi removido, use `tag()` no lugar.
- O `HtmlHelper::loadConfig()` foi removido. As tags podem ser personalizadas usando `templates()` ou as configurações de `templates`.
- O segundo parâmetro `$options` para `HtmlHelper::css()` agora sempre irá exigir um array.
- O primeiro parâmetro `$data` para `HtmlHelper::style()` agora sempre irá exigir um array.
- O parâmetro `inline` foi removido dos métodos meta(), css(), script() e scriptBlock(). Ao invés disso, você deve usar a opção `block`. Definindo `block => true` irá emular o comportamento anterior.
- O `HtmlHelper::meta()` agora exige que o `$type` seja uma string. Opções adicionais podem ser passadas como `$options`.
- O `HtmlHelper::nestedList()` agora exige que o `$options` seja um array. O quarto argumento para o tipo tag foi removido e incluido no array `$options`.
- O argumento `$confirmMessage` de `Cake\View\Helper\HtmlHelper::link()` foi removido. Você deve usar agora a chave `confirm` no `$options` para especificar a menssagem.

PaginatorHelper

- O `link()` foi removido. Ele não era mais usado internamente pelo ajudante. Ele era pouco usado em códigos de usuários e não se encaixava mais nos objetivos do ajudante.
- O `next()` não tem mais as opções 'class' ou 'tag'. Ele não tem mais argumentos desabilitados. Ao invés disso são usados templates.
- O `prev()` não tem mais as opções 'class' ou 'tag'. Ele não tem mais argumentos desabilitados. Ao invés disso são usados templates.
- O `first()` não tem mais as opções 'after', 'ellipsis', 'separator', 'class' ou 'tag'.
- O `last()` não tem mais as opções 'after', 'ellipsis', 'separator', 'class' ou 'tag'.
- O `numbers()` não tem mais as opções 'separator', 'tag', 'currentTag', 'currentClass', 'class', 'tag' e 'ellipsis'. Essas opções são agora facilitadas pelos templates. Ele também exige que agora o parâmetro `$options` seja um array.
- O espaço reservado de estilo `%page%` foi removido de `Cake\View\Helper\PaginatorHelper::counter()`. Use o espaço reservado de estilo `{{page}}` no lugar.
- O `url()` foi renomeada para `generateUrl()` para evitar colisão de declaração de método com `Helper::url()`.

Por padrão todos os links e textos inativos são encapsulados em elementos `<li>`. Isso ajuda a fazer o CSS mais fácil de escrever, e aumenta a compatibilidade com frameworks de CSS populares.

Ao invés de várias opções em cada método, você deve usar a funcionalidade de templates. Veja a documentação *PaginatorHelper Templates* para informações de como se usar templates.

TimeHelper

- `TimeHelper::__set()`, `TimeHelper::__get()`, e `TimeHelper::__isset()` foram removidos. Eles eram métodos mágicos para atributos obsoletos.
- O `TimeHelper::serverOffset()` foi removido. Ele provia práticas incorretas de operações com tempo.
- O `TimeHelper::niceShort()` foi removido.

NumberHelper

- O `NumberHelper::format()` agora exige que `$options` seja um array.

SessionHelper

- O `SessionHelper` está obsoleto. Você pode usar `$this->request->session()` diretamente, e a funcionalidade de mensagens flash foi movida para *Flash*.

JsHelper

- O `JsHelper` e todos motores associados foram removidos. Ele podia gerar somente um subconjunto muito pequeno de códigos JavaScript para biblioteca selecionada e consequentemente tentar gerar todo código JavaScript usando apenas o ajudante se tornava um impedimento com frequência. É recomendado usar diretamente sua biblioteca JavaScript preferida.

CacheHelper Removido

O CacheHelper foi removido. A funcionalidade de cache que ele fornecia não era padrão, limitada e incompatível com layouts não-HTML e views de dados. Essas limitações significavam que uma reconstrução completa era necessária. O ESI (Edge Side Includes) se tornou uma maneira padronizada para implementar a funcionalidade que o CacheHelper costumava fornecer. Entretanto, implementando Edge Side Includes[12] em PHP tem várias limitações e casos. Ao invés de construir uma solução ruim, é recomendado que os desenvolvedores que precisem de cache de resposta completa use o Varnish[13] ou Squid[14] no lugar.

I18n

O subsistema de internacionalização foi completamente reescrito. Em geral, você pode esperar o mesmo comportamento que nas versões anteriores, especialmente se você está usando a família de funções `__()`.

[12] http://en.wikipedia.org/wiki/Edge_Side_Includes
[13] http://varnish-cache.org
[14] http://squid-cache.org

Internamente, a classe `I18n` usa `Aura\Intl`, e métodos apropriados são expostos para dar acesso a funções específicas da biblioteca. Por esta razão a maior parte dos métodos dentro de `I18n` foram removidos ou renomeados.

Devido ao uso do `ext/intl`, a classe L10n foi removida completamente. Ela fornecia dados incompletos e desatualizados em comparação com os dados disponíveis na classe `Locale` do PHP.

O idioma padrão da aplicação não será mais alterado automaticamente pelos idiomas aceitos pelo navegador nem por ter o valor `Config.language` definido na sessão do navegador. Você pode, entretanto, usar um filtro no despachante para trocar o idioma automaticamente a partir do cabeçalho `Accept-Language` enviado pelo navegador:

```
// No config/bootstrap.php
DispatcherFactory::addFilter('LocaleSelector');
```

Não há nenhum substituto incluso para selecionar automaticamente o idioma a partir de um valor configurado na sessão do usuário.

A função padrão para formatação de mensagens traduzidas não é mais a `sprintf`, mas a mais avançada e funcional classe `MessageFormatter`. Em geral você pode reescrever os espaços reservados nas mensagens como segue:

```
// Antes:
__('Hoje é um dia %s na %s', 'Ensolarado', 'Espanha');

// Depois:
__('Hoje é um dia {0} na {1}', 'Ensolarado', 'Espanha');
```

Você pode evitar ter de reescrever suas mensagens usando o antigo formatador `sprintf`:

```
I18n::defaultFormatter('sprintf');
```

Adicionalmente, o valor `Config.language` foi removido e ele não pode mais ser usado para controlar o idioma atual da aplicação. Ao invés disso, você pode usar a classe `I18n`:

```
// Antes
Configure::write('Config.language', 'fr_FR');

// Agora
I18n::locale('en_US');
```

- Os métodos abaixo foram movidos:
  - De `Cake\I18n\Multibyte::utf8()` para `Cake\Utility\Text::utf8()`
  - De `Cake\I18n\Multibyte::ascii()` para `Cake\Utility\Text::ascii()`
  - De `Cake\I18n\Multibyte::checkMultibyte()` para `Cake\Utility\Text::isMultibyte()`
- Como agora o CakePHP requer a extensão mbstring, a classe `Multibyte` foi removida.
- As mensagens de erro por todo o CakePHP não passam mais através das funções de internacionalização. Isso foi feito para simplificar o núcleo do CakePHP e reduzir a sobrecarga. As mensagens

apresentadas aos desenvolvedores são raramente, isso quando, são de fato traduzidas - de modo que essa sobrecarga adicional trás pouco beneficio.

Localização

- Agora o construtor de `Cake\I18n\L10n` recebe uma instância de `Cake\Network\Request` como argumento.

Testes

- O `TestShell` foi removido. O CakePHP, o esqueleto da aplicação e novos plugins "cozinhados", todos usam o `phpunit` para rodar os testes.
- O webrunner (webroot/test.php) foi removido. A adoção do CLI aumentou grandemente desde o release inicial do 2.x. Adicionalmente, os CLI de execução oferecem integração superior com IDE's e outras ferramentas automáticas.

  Se você sentir necessidade de um jeito de executar os testes a partir de um navegador, você deve verificar o VisualPHPUnit[15]. Ele oferece muitas funcionalidades adicionais que o antigo webrunner.
- O `ControllerTestCase` está obsoleto e será removido no CakePHP 3.0.0. Ao invés disso, você deve usar a nova funcionalidade *Controller Integration Testing*.
- As Fixtures devem agora ser referenciadas usando sua forma no plural:

```
// No lugar de
$fixtures = ['app.artigo'];

// Você deve usar
$fixtures = ['app.artigos'];
```

Utilitários

Classe Set Removida

A classe Set foi removida, agora você deve usar a classe Hash no lugar dela.

Pastas & Arquivos

As classes de pastas e arquivos foram renomeadas:

- O `Cake\Utility\File` foi renomeado para `Cake\Filesystem\File`
- O `Cake\Utility\Folder` foi renomeado para `Cake\Filesystem\Folder`

[15] https://github.com/NSinopoli/VisualPHPUnit

Inflexão

- O valor padrão para o argumento `$replacement` do `Cake\Utility\Inflector::slug()` foi alterado do sublinhado (_) para o traço (-). Usando traços para separar palavras nas URLs é a escolha popular e também recomendada pelo Google.
- As transliterações para `Cake\Utility\Inflector::slug()` foram alteradas. Se você usa transliterações personalizadas você terá que atualizar seu código. No lugar de expressões regulares, as transliterações usam simples substituições de string. Isso rendeu melhorias de performance significativas:

```
// No lugar de
Inflector::rules('transliteration', [
    '/ä|æ/' => 'ae',
    '/å/' => 'aa'
]);

// Você deve usar
Inflector::rules('transliteration', [
    'ä' => 'ae',
    'æ' => 'ae',
    'å' => 'aa'
]);
```

- Os conjuntos distintos de regras de não-inflexões e irregulares para pluralização e singularização foram removidos. No lugar agora temos uma lista comum para cada. Quando usar `Cake\Utility\Inflector::rules()` com o tipo 'singular' e 'plural' você não poderá mais usar chaves como 'uninflected' e 'irregular' no array de argumentos `$rules`.

  Você pode adicionar / sobrescrever a lista de regras de não-inflexionados e irregulares usando `Cake\Utility\Inflector::rules()` com valores 'uninflected' e 'irregular' para o argumento `$type`.

Sanitize

- A classe `Sanitize` foi removida.

Segurança

- O `Security::cipher()` foi removido. Ele era inseguro e promovia práticas ruins de criptografia. Você deve usar o `Security::encrypt()` no lugar.
- O valor de configuração `Security.cipherSeed` não é mais necessário. Com a remoção de `Security::cipher()` ele não tem utilidade.
- A retrocompatibilidade do `Cake\Utility\Security::rijndael()` para valores encriptados antes do CakePHP 2.3.1 foi removido. Você deve reencriptar os valores usando `Security::encrypt()` e uma versão recente do CakePHP 2.x antes de migrar.
- A habilidade para gerar um hash do tipo blowfish foi removido. Você não pode mais usar o tipo "blowfish" em `Security::hash()`. Deve ser usado apenas o *password_hash()* do

PHP e *password_verify()* para gerar e verificar hashes blowfish. A compabilidade da biblioteca ircmaxell/password-compat[16] que é instalado junto com o CakePHP fornece essas funções para versões de PHP menor que 5.5.

- O OpenSSL é usado agora no lugar do mcrypt ao encriptar/desencriptar dados. Esse alteração fornece uma melhor performance e deixa o CakePHP a prova de futuros abandonos de suporte das distribuições ao mcrypt.
- O `Security::rijndael()` está obsoleto e apenas disponível quando se usa o mcrypt.

Aviso: Dados encriptados com Security::encrypt() em versões anteriores não são compatíveis com a implementação openssl. Você deve *definir a implementação como mcrypt* quando fizer atualização.

Data e Hora

- O `CakeTime` foi renomeado para `Cake\I18n\Time`.
- O `CakeTime::serverOffset()` foi removido. Ele provia práticas incorretas de operações com tempo.
- O `CakeTime::niceShort()` foi removido.
- O `CakeTime::convert()` foi removido.
- O `CakeTime::convertSpecifiers()` foi removido.
- O `CakeTime::dayAsSql()` foi removido.
- O `CakeTime::daysAsSql()` foi removido.
- O `CakeTime::fromString()` foi removido.
- O `CakeTime::gmt()` foi removido.
- O `CakeTime::toATOM()` foi renomeado para `toAtomString`.
- O `CakeTime::toRSS()` foi renomeado para `toRssString`.
- O `CakeTime::toUnix()` foi renomeado para `toUnixString`.
- O `CakeTime::wasYesterday()` foi renomeado para `isYesterday` para combinar com o resto da renomeação de métodos.
- O `CakeTime::format()` não usa mais o formato do `sprintf`, ao invés disso você deve usar o formato `i18nFormat`.
- O `Time::timeAgoInWords()` agora exige que o `$options` seja um array.

A classe Time não é mais uma coleção de métodos estáticos, ela estende o `DateTime` para herdar todos seus métodos e adicionar funções de formatação baseado em localização com ajuda da extensão `intl`.

Em geral, expressões assim:

[16] https://packagist.org/packages/ircmaxell/password-compat

```
CakeTime::aMethod($date);
```

Podem ser migradas reescrevendo para:

```
(new Time($date))->aMethod();
```

Números

A biblioteca Number foi reescrita para usar internamente a classe `NumberFormatter`.

- O `CakeNumber` foi renomeada para `Cake\I18n\Number`.
- O `Number::format()` agora exige que o `$options` seja um array.
- O `Number::addFormat()` foi removido.
- O `Number::fromReadableSize()` foi movido para `Cake\Utility\Text::parseFileSize()`.

Validação

- A faixa de valores para `Validation::range()` agora é inclusiva se `$lower` e `$upper` forem fornecidos.
- O `Validation::ssn()` foi removido.

Xml

- O `Xml::build()` agora exige que o `$options` seja um array.
- O `Xml::build()` não aceita mais uma URL. Se você precisar criar um documento XML a partir de uma URL, use o

CAPÍTULO 4

Tutoriais & Exemplos

Nesta seção, você poderá caminhar através de típicas aplicações CakePHP para ver como todas as peças se encaixam.

Como alternativa, você pode preferir visitar o repositório não oficial de plugins para o CakePHP CakePackages[1] e a Bakery (Padaria)[2] para conhecer aplicações e componentes existentes.

Tutorial - Criando um Bookmarker - Parte 1

Esse tutorial vai guiar você através da criação de uma simples aplicação de marcação (bookmarker). Para começar, nós vamos instalar o CakePHP, criar nosso banco de dados, e usar as ferramentas que o CakePHP fornece para obter nossa aplicação de pé rápido.

Aqui está o que você vai precisar:

1. Um servidor de banco de dados. Nós vamos usar o servidor MySQL neste tutorial. Você precisa saber o suficiente sobre SQL para criar um banco de dados: O CakePHP vai tomar as rédeas a partir daí. Por nós estarmos usando o MySQL, também certifique-se que você tem a extensão `pdo_mysql` habilitada no PHP.
2. Conhecimento básico sobre PHP.

Vamos começar!

Instalação do CakePHP

A maneira mais fácil de instalar o CakePHP é usando Composer, um gerenciador de dependências para o PHP. É uma forma simples de instalar o CakePHP a partir de seu terminal ou prompt de comando. Primeiro, você precisa baixar e instalar o Composer. Se você tiver instalada a extensão cURL do PHP, execute o seguinte comando:

[1] http://plugins.cakephp.org/
[2] http://bakery.cakephp.org/

```
curl -s https://getcomposer.org/installer | php
```

Ao invés disso, você também pode baixar o arquivo `composer.phar` do site[3] oficial.

Em seguida, basta digitar a seguinte linha no seu terminal a partir do diretório onde se localiza o arquivo `composer.phar` para instalar o esqueleto de aplicações do CakePHP no diretório `bookmarker`.

```
php composer.phar create-project --prefer-dist cakephp/app bookmarker
```

A vantagem de usar Composer é que ele irá completar automaticamente um conjunto importante de tarefas, como configurar as permissões de arquivo e criar a sua **config/app.php**.

Há outras maneiras de instalar o CakePHP. Se você não puder ou não quiser usar Composer, veja a seção *Instalação*.

Independentemente de como você baixou o CakePHP, uma vez que sua instalação for concluída, a estrutura dos diretórios deve ficar parecida com o seguinte:

```
/bookmarker
    /bin
    /config
    /logs
    /plugins
    /src
    /tests
    /tmp
    /vendor
    /webroot
    .editorconfig
    .gitignore
    .htaccess
    .travis.yml
    composer.json
    index.php
    phpunit.xml.dist
    README.md
```

Agora pode ser um bom momento para aprender sobre como a estrutura de diretórios do CakePHP funciona: Confira a seção *Estrutura de pastas do CakePHP*.

Verificando nossa instalação

Podemos checar rapidamente que a nossa instalação está correta, verificando a página inicial padrão. Antes que você possa fazer isso, você vai precisar iniciar o servidor de desenvolvimento:

```
bin/cake server
```

Isto irá iniciar o servidor embutido do PHP na porta 8765. Abra `http://localhost:8765` em seu navegador para ver a página de boas-vindas. Todas as verificações devem estar checadas corretamente, a não ser a conexão com banco de dados do CakePHP. Se não, você pode precisar instalar extensões do PHP adicionais, ou definir permissões de diretório.

[3] https://getcomposer.org/download/

Criando o banco de dados

Em seguida, vamos criar o banco de dados para a nossa aplicação. Se você ainda não tiver feito isso, crie um banco de dados vazio para uso nesse tutorial, com um nome de sua escolha, por exemplo, `cake_bookmarks`. Você pode executar o seguinte SQL para criar as tabelas necessárias:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created DATETIME,
    modified DATETIME
);

CREATE TABLE bookmarks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(50),
    description TEXT,
    url TEXT,
    created DATETIME,
    modified DATETIME,
    FOREIGN KEY user_key (user_id) REFERENCES users(id)
);

CREATE TABLE tags (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    created DATETIME,
    modified DATETIME,
    UNIQUE KEY (title)
);

CREATE TABLE bookmarks_tags (
    bookmark_id INT NOT NULL,
    tag_id INT NOT NULL,
    PRIMARY KEY (bookmark_id, tag_id),
    INDEX tag_idx (tag_id, bookmark_id),
    FOREIGN KEY tag_key(tag_id) REFERENCES tags(id),
    FOREIGN KEY bookmark_key(bookmark_id) REFERENCES bookmarks(id)
);
```

Você deve ter notado que a tabela `bookmarks_tags` utilizada uma chave primária composta. O CakePHP suporta chaves primárias compostas quase todos os lugares, tornando mais fácil construir aplicações multi-arrendados.

Os nomes de tabelas e colunas que usamos não foram arbitrárias. Usando *convenções de nomenclatura* do CakePHP, podemos alavancar o desenvolvimento e evitar ter de configurar o framework. O CakePHP é flexível o suficiente para acomodar até mesmo esquemas de banco de dados legados inconsistentes, mas aderir às convenções vai lhe poupar tempo.

Configurando o banco de dados

Em seguida, vamos dizer ao CakePHP onde o nosso banco de dados está como se conectar a ele. Para muitos, esta será a primeira e última vez que você vai precisar configurar qualquer coisa.

A configuração é bem simples: basta alterar os valores do array `Datasources.default` no arquivo **config/app.php** pelos que se aplicam à sua configuração. A amostra completa da gama de configurações pode ser algo como o seguinte:

```
return [
    // Mais configuração acima.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_bookmarks',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Mais configuração abaixo.
];
```

Depois de salvar o seu arquivo **config/app.php**, você deve notar que a mensagem 'CakePHP is able to connect to the database' tem uma marca de verificação.

Nota: Uma cópia do arquivo de configuração padrão do CakePHP é encontrado em **config/app.default.php**.

Gerando o código base

Devido a nosso banco de dados seguir as convenções do CakePHP, podemos usar o *bake console* para gerar rapidamente uma aplicação básica . Em sua linha de comando execute:

```
bin/cake bake all users
bin/cake bake all bookmarks
bin/cake bake all tags
```

Isso irá gerar os controllers, models, views, seus casos de teste correspondentes, e fixtures para os nossos users, bookmarks e tags. Se você parou seu servidor, reinicie-o e vá para `http://localhost:8765/bookmarks`.

Você deverá ver uma aplicação que dá acesso básico, mas funcional a tabelas de banco de dados. Adicione alguns users, bookmarks e tags.

Adicionando criptografia de senha

Quando você criou seus users, você deve ter notado que as senhas foram armazenadas como texto simples. Isso é muito ruim do ponto de vista da segurança, por isso vamos consertar isso.

Este também é um bom momento para falar sobre a camada de modelo. No CakePHP, separamos os métodos que operam em uma coleção de objetos, e um único objeto em diferentes classes. Métodos que operam na recolha de entidades são colocadas na classe *Table*, enquanto as características pertencentes a um único registro são colocados na classe *Entity*.

Por exemplo, a criptografia de senha é feita no registro individual, por isso vamos implementar esse comportamento no objeto entidade. Dada a circunstância de nós querermos criptografar a senha cada vez que é definida, vamos usar um método modificador/definidor. O CakePHP vai chamar métodos de definição baseados em convenções a qualquer momento que uma propriedade é definida em uma de suas entidades. Vamos adicionar um definidor para a senha. Em **src/Model/Entity/User.php** adicione o seguinte:

```
namespace App\Model\Entity;

use Cake\ORM\Entity;
use Cake\Auth\DefaultPasswordHasher;

class User extends Entity
{

    // Code from bake.

    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();
        return $hasher->hash($value);
    }
}
```

Agora atualize um dos usuários que você criou anteriormente, se você alterar sua senha, você deve ver um senha criptografada ao invés do valor original nas páginas de lista ou visualização. O CakePHP criptografa senhas com bcrypt[4] por padrão. Você também pode usar sha1 ou md5 caso venha a trabalhar com um banco de dados existente.

Recuperando bookmarks com uma tag específica

Agora que estamos armazenando senhas com segurança, podemos construir algumas características mais interessantes em nossa aplicação. Uma vez que você acumulou uma coleção de bookmarks, é útil ser capaz de pesquisar através deles por tag. Em seguida, vamos implementar uma rota, a ação do controller, e um método localizador para pesquisar através de bookmarks por tag.

Idealmente, nós teríamos uma URL que se parece com `http://localhost:8765/bookmarks/tagged/funny/ca`
Isso deveria nos permitir a encontrar todos os bookmarks que têm as tags 'funny', 'cat' e 'gifs'. Antes de podermos implementar isso, vamos adicionar uma nova rota. Em **config/routes.php**, adicione o seguinte na parte superior do arquivo:

[4] http://codahale.com/how-to-safely-store-a-password/

```
Router::scope(
    '/bookmarks',
    ['controller' => 'Bookmarks'],
    function ($routes) {
        $routes->connect('/tagged/*', ['action' => 'tags']);
    }
);
```

O acima define uma nova "rota" que liga o caminho `/bookmarks/tagged/*`, a `BookmarksController::tags()`. Ao definir rotas, você pode isolar como suas URLs parecerão, de como eles são implementadas. Se fôssemos visitar `http://localhost:8765/bookmarks/tagged`, deveriamos ver uma página de erro informativa do CakePHP. Vamos implementar esse método ausente agora. Em **src/Controller/BookmarksController.php** adicione o seguinte:

```
public function tags()
{
    $tags = $this->request->params['pass'];
    $bookmarks = $this->Bookmarks->find('tagged', [
        'tags' => $tags
    ]);
    $this->set(compact('bookmarks', 'tags'));
}
```

Criando o método localizador

No CakePHP nós gostamos de manter as nossas ações do controller enxutas, e colocar a maior parte da lógica de nossa aplicação nos modelos. Se você fosse visitar a URL `/bookmarks/tagged` agora, você veria um erro sobre o método `findTagged` não estar implementado ainda, então vamos fazer isso. Em **src/Model/Table/BookmarksTable.php** adicione o seguinte:

```
public function findTagged(Query $query, array $options)
{
    $fields = [
        'Bookmarks.id',
        'Bookmarks.title',
        'Bookmarks.url',
    ];
    return $this->find()
        ->distinct($fields)
        ->matching('Tags', function ($q) use ($options) {
            return $q->where(['Tags.title IN' => $options['tags']]);
        });
}
```

Nós implementamos um método *localizador customizado*. Este é um conceito muito poderoso no CakePHP que lhe permite construir consultas reutilizáveis. Em nossa pesquisa, nós alavancamos o método `matching()` que nos habilita encontrar bookmarks que têm uma tag 'correspondente'.

Criando a view

Agora, se você visitar a URL `/bookmarks/tagged`, o CakePHP irá mostrar um erro e deixá-lo saber que você ainda não fez um arquivo view. Em seguida, vamos construir o arquivo view para a nossa ação `tags`. Em **src/Template/Bookmarks/tags.ctp** coloque o seguinte conteúdo:

```
<h1>
    Bookmarks tagged with
    <?= $this->Text->toList($tags) ?>
</h1>

<section>
<?php foreach ($bookmarks as $bookmark): ?>
    <article>
        <h4><?= $this->Html->link($bookmark->title, $bookmark->url) ?></h4>
        <small><?= h($bookmark->url) ?></small>
        <?= $this->Text->autoParagraph($bookmark->description) ?>
    </article>
<?php endforeach; ?>
</section>
```

O CakePHP espera que os nossos templates sigam a convenção de nomenclatura onde o nome do template é a versão minúscula e grifada do nome da ação do controller.

Você pode perceber que fomos capazes de utilizar as variáveis `$tags` e `bookmarks` em nossa view. Quando usamos o método `set()` em nosso controller, automaticamente definimos variáveis específicas que devem ser enviadas para a view. A view vai tornar todas as variáveis passadas disponíveis nos templates como variáveis locais.

Em nossa view, usamos alguns dos *helpers* nativos do CakePHP. Helpers são usados para criar lógica reutilizável para a formatação de dados, a criação de HTML ou outra saída da view.

Agora você deve ser capaz de visitar a URL `/bookmarks/tagged/funny` e ver todas os bookmarks com a tag 'funny'.

Até agora, nós criamos uma aplicação básica para gerenciar bookmarks, tags e users. No entanto, todos podem ver as tags de toda a gente. No próximo capítulo, vamos implementar a autenticação e restringir os bookmarks visíveis para somente aqueles que pertencem ao usuário atual.

Agora vá a *Tutorial - Criando um Bookmarker - Parte 2* para continuar a construir sua aplicação ou `mergulhe na documentação` para saber mais sobre o que CakePHP pode fazer por você.

Tutorial - Criando um Bookmarker - Parte 2

Depois de terminar a *primeira parte deste tutorial*, você deve ter uma aplicação muito básica. Neste capítulo iremos adicionar autenticação e restringir as bookmarks para que cada usuário possa ver/modificar somente aquelas que possuam.

Adicionando login

No CakePHP, a autenticação é feita por *Components (Componentes)*. Os Components podem ser considerados como formas de criar pedaços reutilizáveis de código relacionado a controllers com uma característica específica ou conceito. Os components também podem ligar-se ao evento do ciclo de vida do controller e interagir com a sua aplicação. Para começar, vamos adicionar o AuthComponent a nossa aplicação. Nós vamos querer muito que cada método exija autenticação, por isso vamos acrescentar o *AuthComponent* em nosso AppController:

```
// Em src/Controller/AppController.php
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
    public function initialize()
    {
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authenticate' => [
                'Form' => [
                    'fields' => [
                        'username' => 'email',
                        'password' => 'password'
                    ]
                ]
            ],
            'loginAction' => [
                'controller' => 'Users',
                'action' => 'login'
            ]
        ]);

        // Permite a ação display, assim nosso pages controller
        // continua a funcionar.
        $this->Auth->allow(['display']);
    }
}
```

Acabamos de dizer ao CakePHP que queremos carregar os components `Flash` e `Auth`. Além disso, temos a configuração personalizada do AuthComponent, assim a nossa tabela users pode usar `email` como username. Agora, se você for a qualquer URL, você vai ser chutado para `/users/login`, que irá mostrar uma página de erro já que não escrevemos o código ainda. Então, vamos criar a ação de login:

```
// Em src/Controller/UsersController.php

public function login()
{
    if ($this->request->is('post')) {
        $user = $this->Auth->identify();
        if ($user) {
            $this->Auth->setUser($user);
```

```
            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Flash->error('Your username or password is incorrect.');
    }
}
```

E em **src/Template/Users/login.ctp** adicione o seguinte:

```
<h1>Login</h1>
<?= $this->Form->create() ?>
<?= $this->Form->input('email') ?>
<?= $this->Form->input('password') ?>
<?= $this->Form->button('Login') ?>
<?= $this->Form->end() ?>
```

Agora que temos um simples formulário de login, devemos ser capazes de efetuar login com um dos users que tenham senha criptografada.

Nota: Se nenhum de seus users tem senha criptografada, comente a linha `loadComponent('Auth')`. Então vá e edite o user, salvando uma nova senha para ele.

Agora você deve ser capaz de entrar. Se não, certifique-se que você está usando um user que tenha senha criptografada.

Adicionando logout

Agora que as pessoas podem efetuar o login, você provavelmente vai querer fornecer uma maneira de encerrar a sessão também. Mais uma vez, no `UsersController`, adicione o seguinte código:

```
public function logout()
{
    $this->Flash->success('You are now logged out.');
    return $this->redirect($this->Auth->logout());
}
```

Agora você pode visitar `/users/logout` para sair e ser enviado à página de login.

Ativando inscrições

Se você não estiver logado e tentar visitar / usuários / adicionar você vai ser expulso para a página de login. Devemos corrigir isso se quisermos que as pessoas se inscrevam em nossa aplicação. No UsersController adicione o seguinte:

```
public function beforeFilter(\Cake\Event\Event $event)
{
    $this->Auth->allow(['add']);
}
```

O texto acima diz ao `AuthComponent` que a ação `add` não requer autenticação ou autorização. Você pode querer dedicar algum tempo para limpar a `/users/add` e remover os links enganosos, ou continuar para a

próxima seção. Nós não estaremos construindo a edição do usuário, visualização ou listagem neste tutorial, então eles não vão funcionar, já que o `AuthComponent` vai negar-lhe acesso a essas ações do controller.

Restringindo acesso

Agora que os usuários podem conectar-se, nós vamos querer limitar os bookmarks que podem ver para aqueles que fizeram. Nós vamos fazer isso usando um adaptador de 'autorização'. Sendo os nossos requisitos bastante simples, podemos escrever um código em nossa `BookmarksController`. Mas antes de fazer isso, vamos querer dizer ao AuthComponent como nossa aplicação vai autorizar ações. Em seu `AppController` adicione o seguinte:

```
public function isAuthorized($user)
{
    return false;
}
```

Além disso, adicione o seguinte à configuração para `Auth` em seu `AppController`:

```
'authorize' => 'Controller',
```

Seu método `initialize` agora deve parecer com:

```
public function initialize()
{
    $this->loadComponent('Flash');
    $this->loadComponent('Auth', [
        'authorize'=> 'Controller',//added this line
        'authenticate' => [
            'Form' => [
                'fields' => [
                    'username' => 'email',
                    'password' => 'password'
                ]
            ]
        ],
        'loginAction' => [
            'controller' => 'Users',
            'action' => 'login'
        ],
        'unauthorizedRedirect' => $this->referer()
    ]);

    // Permite a ação display, assim nosso pages controller
    // continua a funcionar.
    $this->Auth->allow(['display']);
}
```

Vamos usar como padrão, negação do acesso, e de forma incremental conceder acesso onde faça sentido. Primeiro, vamos adicionar a lógica de autorização para os bookmarks. Em seu `BookmarksController` adicione o seguinte:

```
public function isAuthorized($user)
{
    $action = $this->request->params['action'];

    // As ações add e index são permitidas sempre.
    if (in_array($action, ['index', 'add', 'tags'])) {
        return true;
    }
    // Todas as outras ações requerem um id.
    if (empty($this->request->params['pass'][0])) {
        return false;
    }

    // Checa se o bookmark pertence ao user atual.
    $id = $this->request->params['pass'][0];
    $bookmark = $this->Bookmarks->get($id);
    if ($bookmark->user_id == $user['id']) {
        return true;
    }
    return parent::isAuthorized($user);
}
```

Agora, se você tentar visualizar, editar ou excluir um bookmark que não pertença a você, você deve ser redirecionado para a página de onde veio. No entanto, não há nenhuma mensagem de erro sendo exibida, então vamos corrigir isso a seguir:

```
// In src/Template/Layout/default.ctp
// Under the existing flash message.
<?= $this->Flash->render('auth') ?>
```

Agora você deve ver as mensagens de erro de autorização.

Corrigindo a view de listagem e formulários

Enquanto view e delete estão trabalhando, edit, add e index tem alguns problemas:

1. Ao adicionar um bookmark, você pode escolher o user.
2. Ao editar um bookmark, você pode escolher o user.
3. A página de listagem mostra os bookmarks de outros users.

Vamos enfrentar o formulário de adição em primeiro lugar. Para começar remova o `input('user_id')` a partir de **src/Template/Bookmarks/add.ctp**. Com isso removido, nós também vamos atualizar o método add:

```
public function add()
{
    $bookmark = $this->Bookmarks->newEntity();
    if ($this->request->is('post')) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
```

```
            $this->Flash->success('The bookmark has been saved.');
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('The bookmark could not be saved. Please, try again.');
    }
    $tags = $this->Bookmarks->Tags->find('list');
    $this->set(compact('bookmark', 'tags'));
}
```

Ao definir a propriedade da entidade com os dados da sessão, nós removemos qualquer possibilidade do user modificar de que outro user um bookmark seja. Nós vamos fazer o mesmo para o formulário edit e action edit. Sua ação edit deve ficar assim:

```
public function edit($id = null)
{
    $bookmark = $this->Bookmarks->get($id, [
        'contain' => ['Tags']
    ]);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('The bookmark has been saved.');
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('The bookmark could not be saved. Please, try again.');
    }
    $tags = $this->Bookmarks->Tags->find('list');
    $this->set(compact('bookmark', 'tags'));
}
```

View de listagem

Agora, nós precisamos apenas exibir bookmarks para o user logado. Nós podemos fazer isso ao atualizar a chamada para `paginate()`. Altere sua ação index:

```
public function index()
{
    $this->paginate = [
        'conditions' => [
            'Bookmarks.user_id' => $this->Auth->user('id'),
        ]
    ];
    $this->set('bookmarks', $this->paginate($this->Bookmarks));
}
```

Nós também devemos atualizar a action `tags()` e o método localizador relacionado, mas vamos deixar isso como um exercício para que você conclua por sí.

Melhorando a experiência com as tags

Agora, adicionar novas tags é um processo difícil, pois o `TagsController` proíbe todos os acessos. Em vez de permitir o acesso, podemos melhorar a interface do usuário para selecionar tags usando um campo de texto separado por vírgulas. Isso permitirá dar uma melhor experiência para os nossos usuários, e usar mais alguns grandes recursos no ORM.

Adicionando um campo computado

Porque nós queremos uma maneira simples de acessar as tags formatados para uma entidade, podemos adicionar um campo virtual/computado para a entidade. Em **src/Model/Entity/Bookmark.php** adicione o seguinte:

```
use Cake\Collection\Collection;

protected function _getTagString()
{
    if (isset($this->_properties['tag_string'])) {
        return $this->_properties['tag_string'];
    }
    if (empty($this->tags)) {
        return '';
    }
    $tags = new Collection($this->tags);
    $str = $tags->reduce(function ($string, $tag) {
        return $string . $tag->title . ', ';
    }, '');
    return trim($str, ', ');
}
```

Isso vai nos deixar acessar a propriedade computada `$bookmark->tag_string`. Vamos usar essa propriedade em inputs mais tarde. Lembre-se de adicionar a propriedade `tag_string` a lista `_accessible` em sua entidade.

Em **src/Model/Entity/Bookmark.php** adicione o `tag_string` ao `_accessible` desta forma:

```
protected $_accessible = [
    'user_id' => true,
    'title' => true,
    'description' => true,
    'url' => true,
    'user' => true,
    'tags' => true,
    'tag_string' => true,
];
```

Atualizando as views

Com a entidade atualizado, podemos adicionar uma nova entrada para as nossas tags. Nas views add e edit, substitua `tags._ids` pelo seguinte:

```
<?= $this->Form->input('tag_string', ['type' => 'text']) ?>
```

Persistindo a string tag

Agora que podemos ver as tags como uma string existente, vamos querer salvar os dados também. Por marcar o `tag_string` como acessível, o ORM irá copiar os dados do pedido em nossa entidade. Podemos usar um método `beforeSave` para analisar a cadeia tag e encontrar/construir as entidades relacionadas. Adicione o seguinte em **src/Model/Table/BookmarksTable.php**:

```
public function beforeSave($event, $entity, $options)
{
    if ($entity->tag_string) {
        $entity->tags = $this->_buildTags($entity->tag_string);
    }
}

protected function _buildTags($tagString)
{
    $new = array_unique(array_map('trim', explode(',', $tagString)));
    $out = [];
    $query = $this->Tags->find()
        ->where(['Tags.title IN' => $new]);

    // Remove tags existentes da lista de novas tags.
    foreach ($query->extract('title') as $existing) {
        $index = array_search($existing, $new);
        if ($index !== false) {
            unset($new[$index]);
        }
    }
    // Adiciona tags existentes.
    foreach ($query as $tag) {
        $out[] = $tag;
    }
    // Adiciona novas tags.
    foreach ($new as $tag) {
        $out[] = $this->Tags->newEntity(['title' => $tag]);
    }
    return $out;
}
```

Embora esse código seja um pouco mais complicado do que o que temos feito até agora, ele ajuda a mostrar o quão poderosa a ORM do CakePHP é. Você pode facilmente manipular resultados da consulta usando os métodos de *Collections (Coleções)*, e lidar com situações em que você está criando entidades sob demanda com facilidade.

Terminando

Nós expandimos nossa aplicação bookmarker para lidar com situações de autenticação e controle de autorização/acesso básico. Nós também adicionamos algumas melhorias agradáveis à UX, aproveitando os

recursos FormHelper e ORM.

Obrigado por dispor do seu tempo para explorar o CakePHP. Em seguida, você pode saber mais sobre o *Models (Modelos)*, ou você pode ler os `/topics`.

Tutorial - Criando um Blog - Parte 1

Este tutorial irá orientá-lo através da criação de um simples blog. Faremos a instalação do CakePHP, criaremos um banco de dados e implementaremos a lógica capaz de listar, adicionar, editar e apagar postagens do blog.

Aqui está o que você vai precisar:

1. Um servidor web em funcionamento. Nós iremos assumir que você esteja usando o Apache, embora as instruções para outros servidores sejam bem similares. Talvez seja preciso alterar um pouco a configuração do servidor, mas a maioria das pessoas pode ter o CakePHP instalado e funcionando sem qualquer trabalho extra. Certifique-se de que você tem o PHP 5.5.9 ou superior, e que as extensões *mbstring* e *intl* estejam habilitadas no PHP. Caso não saiba a versão do PHP que está instalada, utilize a função `phpinfo()` ou digite `php -v` no seu terminal de comando.
2. Um servidor de banco de dados. Nós vamos usar o servidor *MySQL* neste tutorial. Você precisa saber o mínimo sobre SQL para então criar um banco de dados, depois disso o CakePHP vai assumir as rédeas. Já que usaremos o *MySQL*, também certifique-se que a extensão `pdo_mysql` está habilitada no PHP.
3. Conhecimento básico sobre PHP.

Vamos começar!

Instalação do CakePHP

A maneira mais fácil de instalar o CakePHP é usando Composer, um gerenciador de dependências para o PHP. Se trata de uma forma simples de instalar o CakePHP a partir de seu terminal ou prompt de comando. Primeiro, você precisa baixar e instalar o Composer. Se possuir instalada a extensão *cURL* do PHP, execute o seguinte comando:

```
curl -s https://getcomposer.org/installer | php
```

Você também pode baixar o arquivo `composer.phar` do site[5] oficial do Composer.

Em seguida, basta digitar a seguinte linha de comando no seu terminal a partir do diretório onde se localiza o arquivo `composer.phar` para instalar o esqueleto da aplicação do CakePHP no diretório [nome_do_app].

```
php composer.phar create-project --prefer-dist cakephp/app [nome_do_app]
```

A vantagem de usar o Composer é que ele irá completar automaticamente um conjunto importante de tarefas, como configurar corretamente as permissões de pastas e criar o **config/app.php** para você.

[5] https://getcomposer.org/download/

Há outras maneiras de instalar o CakePHP. Se você não puder ou não quiser usar o Composer, confira a seção *Instalação*.

Independentemente de como você baixou o CakePHP, uma vez que sua instalação for concluída, a estrutura dos diretórios deve ficar parecida com o seguinte:

```
/nome_do_app
    /bin
    /config
    /logs
    /plugins
    /src
    /tests
    /tmp
    /vendor
    /webroot
    .editorconfig
    .gitignore
    .htaccess
    .travis.yml
    composer.json
    index.php
    phpunit.xml.dist
    README.md
```

Agora pode ser um bom momento para aprender sobre como a estrutura de diretórios do CakePHP funciona: Confira a seção *Estrutura de pastas do CakePHP*.

Permissões dos diretórios tmp e logs

Os diretórios **tmp** e **logs** precisam ter permissões adequadas para que possam ser alterados pelo seu servidor web. Se você usou o Composer na instalação, ele deve ter feito isso por você e confirmado com uma mensagem "Permissions set on <folder>". Se você ao invés disso, recebeu uma mensagem de erro ou se quiser fazê-lo manualmente, a melhor forma seria descobrir por qual usuário o seu servidor web é executado (`<?= 'whoami'; ?>`) e alterar o proprietário desses dois diretórios para este usuário. Os comandos finais a serem executados (em *nix) podem ser algo como:

```
chown -R www-data tmp
chown -R www-data logs
```

Se por alguma razão o CakePHP não puder escrever nesses diretórios, você será informado por uma advertência enquanto não estiver em modo de produção.

Embora não seja recomendado, se você é incapaz de redefinir as permissões do seu servidor web, você pode simplesmente alterar as permissões de gravação diretamente nos diretórios, executando os seguintes comandos:

```
chmod 777 -R tmp
chmod 777 -R logs
```

Criando o banco de dados do Blog

Em seguida, vamos configurar o banco de dados para o nosso blog. Se você ainda não tiver feito isto, crie um banco de dados vazio para usar neste tutorial, com um nome de sua escolha, por exemplo, `cake_blog`. Agora, vamos criar uma tabela para armazenar nossos artigos:

```
/* Primeiro, criamos a tabela articles: */
CREATE TABLE articles (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(50),
    body TEXT,
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);
```

Nós vamos também inserir alguns artigos para usarmos em nossos testes. Execute os seguintes comandos SQL em seu banco de dados:

```
/* Então inserimos articles para testes: */
INSERT INTO articles (title,body,created)
    VALUES ('The title', 'This is the article body.', NOW());
INSERT INTO articles (title,body,created)
    VALUES ('A title once again', 'And the article body follows.', NOW());
INSERT INTO articles (title,body,created)
    VALUES ('Title strikes back', 'This is really exciting! Not.', NOW());
```

Os nomes de tabelas e colunas que usamos não foram arbitrárias. Usando *convenções de nomenclatura* do CakePHP, podemos alavancar o desenvolvimento e acelerar a configuração do framework. O CakePHP é flexível o suficiente para acomodar até mesmo esquemas de banco de dados legados inconsistentes, mas aderir às convenções vai lhe poupar tempo.

Configurando o banco de dados do Blog

Em seguida, vamos dizer ao CakePHP onde nosso banco de dados está e como se conectar a ele. Para muitos, esta será a primeira e última vez que será necessário configurar algo.

A configuração é bem simples e objetiva: basta alterar os valores no array `Datasources.default` localizado no arquivo **config/app.php**, pelos valores que se aplicam à sua configuração. Um exemplo completo de configurações deve se parecer como o seguinte:

```
return [
    // Mais configurações acima.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_blog',
            'encoding' => 'utf8',
```

```
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Mais configurações abaixo.
];
```

Depois de salvar o arquivo **config/app.php**, você deve notar a mensagem *CakePHP is able to connect to the database* ao acessar o Blog pelo seu navegador.

Nota: Uma cópia do arquivo de configuração padrão do CakePHP pode ser encontrada em **config/app.default.php**.

Configurações opcionais

Há alguns outros itens que podem ser configurados. Muitos desenvolvedores completam esta lista de itens, mas os mesmos não são obrigatórios para este tutorial. Um deles é definir uma sequência personalizada (ou "salt") para uso em hashes de segurança.

A sequência personalizada (ou salt) é utilizada para gerar hashes de segurança. Se você utilizou o Composer, ele cuidou disso para você durante a instalação. Apesar disso, você precisa alterar a sequência personalizada padrão editando o arquivo **config/app.php**. Não importa qual será o novo valor, somente deverá ser algo difícil de descobrir:

```
'Security' => [
    'salt' => 'algum valor longo contendo uma mistura aleatória de valores.',
],
```

Observação sobre o mod_rewrite

Ocasionalmente, novos usuários irão se atrapalhar com problemas de mod_rewrite. Por exemplo, se a página de boas vindas do CakePHP parecer estranha (sem imagens ou estilos CSS). Isto provavelmente significa que o mod_rewrite não está funcionando em seu servidor. Por favor, verifique a seção *Reescrita de URL* para obter ajuda e resolver qualquer problema relacionado.

Agora continue o tutorial em *Tutorial - Criando um Blog - Parte 2* e inicie a construção do seu Blog com o CakePHP.

Tutorial - Criando um Blog - Parte 2

Criando o model

Após criar um model (modelo) no CakePHP, nós teremos a base necessária para interagirmos com o banco de dados e executar operações.

Os arquivos de classes, correspondentes aos models, no CakePHP estão divididos entre os objetos `Table` e `Entity`. Objetos `Table` provêm acesso à coleção de entidades armazenada em uma tabela e são alocados em **src/Model/Table**.

O arquivo que criaremos deverá ficar salvo em **src/Model/Table/ArticlesTable.php**:

```
// src/Model/Table/ArticlesTable.php

namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
    }
}
```

Convenções de nomenclatura são muito importantes no CakePHP. Ao nomear nosso objeto como `ArticlesTable`, o CakePHP automaticamente deduz que o mesmo utilize o `ArticlesController` e seja relacionado à tabela `articles`.

Nota: O CakePHP criará automaticamente um objeto model se não puder encontrar um arquivo correspondente em **src/Model/Table**. Se você nomear incorretamente seu arquivo (isto é, artciclestable.php ou ArticleTable.php), o CakePHP não reconhecerá suas definições e usará o model gerado como alternativa.

Para mais informações sobre models, como callbacks e validação, visite o capítulo *Models (Modelos)* do manual.

Nota: Se você completou a *primeira parte* do tutorial e criou a tabela `articles`, você pode tomar proveito da capacidade de geração de código do bake através do console do CakePHP para criar o model `ArticlesTable`:

```
bin/cake bake model Articles
```

Para mais informações sobre o bake e suas características relacionadas a geração de código, visite o capítulo *Geração de código com o Bake* do manual.

Criando o controller

A seguir, criaremos um controller (controlador) para nossos artigos. O controller é responsável pela lógica de interação da aplicação. É o lugar onde você utilizará as regras contidas nos models e executará tarefas relacionadas aos artigos. Criaremos um arquivo chamado **ArticlesController.php** no diretório **src/Controller**:

```
// src/Controller/ArticlesController.php

namespace App\Controller;
```

```
class ArticlesController extends AppController
{
}
```

Agora, vamos adicionar uma action (ação) ao nosso controller. Actions frequentemente, representam uma função ou interface em uma aplicação. Por exemplo, quando os usuários requisitarem www.example.com/articles/index (sendo o mesmo que www.example.com/articles/), eles esperam ver uma lista de artigos:

```
// src/Controller/ArticlesController.php

namespace App\Controller;

class ArticlesController extends AppController
{

    public function index()
    {
        $articles = $this->Articles->find('all');
        $this->set(compact('articles'));
    }
}
```

Ao definir a função `index()` em nosso `ArticlesController`, os usuários podem acessá-la requisitando www.example.com/articles/index. Similarmente, se definíssemos uma função chamada `foobar()`, os usuários poderiam acessá-la em www.example.com/articles/foobar.

Aviso: Vocês podem ser tentados a nomear seus controllers e actions para obter uma certa URL. Resista a essa tentação. Siga as *Convenções do CakePHP* e crie nomes de action legíveis e compreensíveis. Você pode mapear URLs para o seu código utilizando *Roteamento*.

A instrução na action usa `set()` para passar dados do controller para a view. A variável é definida como 'articles', sendo igual ao valor retornado do método `find('all')` do objeto `ArticlesTable`.

Nota: Se você completou a *primeira parte* do tutorial e criou a tabela `articles`, você pode tomar proveito da capacidade de geração de código do bake através do console do CakePHP para criar o controller `ArticlesController`:

```
bin/cake bake controller Articles
```

Para mais informações sobre o bake e suas características sobre geração de código, visite o capítulo *Geração de código com o Bake* do manual.

Criando as views

Agora que nós temos os dados fluindo pelo nosso model, e nossa lógica da aplicação definida em nosso controller, vamos criar uma view (visualização) para a action `index()`.

As views do CakePHP são camadas de apresentação que se encaixam nos layouts da aplicação. Para a maioria das aplicações, elas são uma mescla entre HTML e PHP, mas também podem ser distribuídas como

XML, CSV, ou ainda dados binários.

Um layout é um conjunto de códigos encontrado ao redor das views. Múltiplos layouts podem ser definidos, e você pode alterar entre eles, mas agora, vamos usar o default, localziado em **src/Template/Layout/default.ctp**.

Lembra que na última sessão atribuímos a variável 'articles' à view usando o método `set()`? Isso levará a coleção de objetos gerada pela query a ser invocada numa iteração `foreach`.

Arquivos de template do CakePHP são armazenados em **src/Template** dentro de uma pasta com o nome do controller correspondente (nós teremos que criar a pasta 'Articles' nesse caso). Para distribuir os dados de artigos em uma tabela, precisamos criar uma view assim:

```
<!-- File: src/Template/Articles/index.ctp -->

<h1>Blog articles</h1>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Created</th>
    </tr>

    <!-- Aqui é onde iremos iterar nosso objeto de solicitação $articles, exibindo informac

    <?php foreach ($articles as $article): ?>
    <tr>
        <td><?= $article->id ?></td>
        <td>
            <?= $this->Html->link($article->title, ['action' => 'view', $article->id]) ?>
        </td>
        <td>
            <?= $article->created->format(DATE_RFC850) ?>
        </td>
    </tr>
    <?php endforeach; ?>
</table>
```

Você deve ter notado o uso de um objeto chamado `$this->Html`, uma instância da classe `Cake\View\Helper\HtmlHelper` do CakePHP. O CakePHP vem com um conjunto de view helpers que simplificam tarefas como gerar links e formulários. Você pode aprender como usá-los em *Helpers (Facilitadores)*, mas aqui é importante notar que o método `link()` irá gerar um link HTML com o referido título (primeiro parâmetro) e URL (segundo parâmetro).

Quando se especifíca URLs no CakePHP, é recomendado o uso do formato de array. Isto será melhor explicado posteriormente na seção Rotas. Usando o formato de array para URLs, você toma vantagem das capacidades de roteamento reverso do CakePHP. Você também pode especificar URLs relativas a base da aplicação com o formato `/controller/action/param1/param2` ou usar *named routes*.

Neste ponto, você pode visitar http://www.example.com/articles/index no seu navegador. Você deve ver sua view corretamente formatada listando os artigos.

Se você clicar no link do título de um artigo listado, provavelmente será informado pelo CakePHP que a action ainda não foi definida, então vamos criá-la no `ArticlesController` agora:

```
// src/Controller/ArticlesController.php

namespace App\Controller;

class ArticlesController extends AppController
{

    public function index()
    {
         $this->set('articles', $this->Articles->find('all'));
    }

    public function view($id = null)
    {
        $article = $this->Articles->get($id);
        $this->set(compact('article'));
    }
}
```

O uso do `set()` deve parecer familiar. Repare que você está usando `get()` ao invés de `find('all')` porquê nós queremos a informação de apenas um artigo.

Repare que nossa action recebe um parâmetro: o ID do artigo que gostariamos de visualizar. Esse parâmetro é entregue para a action através da URL solicitada. Se o usuário requisitar `/articles/view/3`, então o valor '3' é passado como `$id` para a action.

Ao usar a função `get()`, fazemos também algumas verificações para garantir que o usuário realmente está acessando um registro existente , se não ou se o `$id` for indefinido, a função irá lançar uma `NotFoundException`.

Agora vamos criar a view para nossa action em **src/Template/Articles/view.ctp**

```
<!-- File: src/Template/Articles/view.ctp -->

<h1><?= h($article->title) ?></h1>
<p><?= h($article->body) ?></p>
<p><small>Criado: <?= $article->created->format(DATE_RFC850) ?></small></p>
```

Verifique se está tudo funcionando acessando os links em `/articles/index` ou manualmente solicite a visualização de um artigo acessando `articles/view/{id}`. Lembre-se de substituir `{id}` por um 'id' de um artigo.

Adicionando artigos

Primeiro, comece criando a action `add()` no `ArticlesController`:

```
// src/Controller/ArticlesController.php

namespace App\Controller;

use App\Controller\AppController;
```

```php
class ArticlesController extends AppController
{

    public function initialize()
    {
        parent::initialize();

        $this->loadComponent('Flash'); // Inclui o FlashComponent
    }

    public function index()
    {
        $this->set('articles', $this->Articles->find('all'));
    }

    public function view($id)
    {
        $article = $this->Articles->get($id);
        $this->set(compact('article'));
    }

    public function add()
    {
        $article = $this->Articles->newEntity();
        if ($this->request->is('post')) {
            $article = $this->Articles->patchEntity($article, $this->request->data);
            if ($this->Articles->save($article)) {
                $this->Flash->success(__('Seu artigo foi salvo.'));
                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('Não é possível adicionar o seu artigo.'));
        }
        $this->set('article', $article);
    }
}
```

Nota: Você precisa incluir o *Flash* component em qualquer controller que vá usá-lo. Se necessário, inclua no `AppController` e assim o `FlashComponent` estará disponível para todos os controllers da aplicação.

A action `add()` checa se o método HTTP da solicitação foi POST, e então tenta salvar os dados utilizando o model Articles. Se por alguma razão ele não salvar, apenas renderiza a view. Isto nos dá a chance de exibir erros de validação ou outros alertas.

Cada requisição do CakePHP instancia um objeto `Request` que é acessível usando `$this->request`. O objeto contém informações úteis sobre a requisição que foi recebida e pode ser usado para controlar o fluxo de sua aplicação. Nesse caso, nós usamos o método `Cake\Network\Request::is()` para checar se a requisição é do tipo HTTP POST.

Quando se usa um formulário para postar dados, essa informação fica disponível em `$this->request->data`. Você pode usar as funções `pr()` ou `debug()` caso queira verificar esses dados.

Usamos os métodos `success()` e `error()` do `FlashComponent` para definir uma mensagem que será armazenada numa variável de sessão. Esses métodos são gerados usando os recursos de métodos mágicos[6] do PHP. Mensagens flash serão exibidas na página após um redirecionamento. No layout nós temos `<?= $this->Flash->render() ?>` que exibe a mensagem e limpa a variável de sessão. A função do controller `Cake\Controller\Controller::redirect` redireciona para qualquer outra URL. O parâmetro `['action' => 'index']` corresponde a URL /articles, isto é, a action `index()` do `ArticlesController`. Você pode consultar a função `Cake\Routing\Router::url()` na API[7] e checar os formatos a partir dos quais você pode montar uma URL.

Chamar o método `save()` vai checar erros de validação e abortar o processo caso os encontre. Nós vamos abordar como esses erros são tratados nas sessões a seguir.

Validando artigos

O CakePHP torna mais prática e menos monótona a validação de dados de formulário.

Para tirar proveito dos recursos de validação, você vai precisar usar o *Form* helper em suas views. O `Cake\View\Helper\FormHelper` está disponível por padrão em todas as views pelo uso do `$this->Form`.

Segue a view correspondente a action add:

```
<!-- File: src/Template/Articles/add.ctp -->

<h1>Add Article</h1>
<?php
    echo $this->Form->create($article);
    echo $this->Form->input('title');
    echo $this->Form->input('body', ['rows' => '3']);
    echo $this->Form->button(__('Salvar artigo'));
    echo $this->Form->end();
?>
```

Nós usamos o `FormHelper` para gerar a tag de abertura HTML de um formulário. Segue o HTML gerado por `$this->Form->create()`:

```
<form method="post" action="/articles/add">
```

Se `create()` é chamado sem parâmetros fornecidos, assume-se a construção de um formulário que submete dados via POST para a action `add()` (ou `edit()` no caso de um `id` estar incluído nos dados do formulário).

O método `$this->Form->input()` é usado para criar elementos do formulário do mesmo nome. O primeiro parâmetro diz ao CakePHP qual é o campo correspondente, e o segundo parâmetro permite que você especifique um vasto array de opções, nesse, o número de linhas para o textarea. `input()` vai gerar diferentes elementos de formulários baseados no tipo de campo especificado no model.

O `$this->Form->end()` fecha o formulário, entregando também elementos ocultos caso a prevenção contra CSRF/Form Tampering esteja habilitada.

[6] http://php.net/manual/en/language.oop5.overloading.php#object.call
[7] http://api.cakephp.org

Agora vamos voltar e atualizar nossa view **src/Template/Articles/index.ctp** para incluir um novo link. Antes do `<table>`, adicione a seguinte linha:

```
<?= $this->Html->link('Adicionar artigo', ['action' => 'add']) ?>
```

Você deve estar se perguntando: como eu digo ao CakePHP meus critérios de validação? Regras de validação são definidas no model. Vamos fazer alguns ajustes no nosso model:

```
// src/Model/Table/ArticlesTable.php

namespace App\Model\Table;

use Cake\ORM\Table;
use Cake\Validation\Validator;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
    }

    public function validationDefault(Validator $validator)
    {
        $validator
            ->notEmpty('title')
            ->notEmpty('body');

        return $validator;
    }
}
```

O método `validationDefault()` diz ao CakePHP como validar seus dados quando o método `save()` for solicitado. Aqui, estamos especificando que tanto o campo body quanto title não devem estar vazios. O CakePHP possui muitos recursos de validação e um bom número de regras pré-determinadas (número de cartões, endereços de email, etc), além de flexibilidade para adicionar regras de validação customizadas. Para mais informações sobre configuração de validações, visite a documentação em *Validação*.

Agora que suas regras de validação estão definidas, tente adicionar um artigo sem definir o campo title e body para ver como a validação funciona. Desde que tenhamos usado o método `Cake\View\Helper\FormHelper::input()` do `FormHelper` para criar nossos elementos, nossas mensagens de alerta da validação serão exibidas automaticamente.

Editando artigos

Edição, aí vamos nós! Você já é um profissional do CakePHP agora, então possivelmente detectou um padrão... Cria-se a action e então a view. Aqui segue a action `edit()` que deverá ser inserida no `ArticlesController`:

```
// src/Controller/ArticlesController.php

public function edit($id = null)
```

```
{
    $article = $this->Articles->get($id);
    if ($this->request->is(['post', 'put'])) {
        $this->Articles->patchEntity($article, $this->request->data);
        if ($this->Articles->save($article)) {
            $this->Flash->success(__('Seu artigo foi atualizado.'));
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('Seu artigo não pôde ser atualizado.'));
    }

    $this->set('article', $article);
}
```

Essa action primeiramente certifica-se que o registro apontado existe. Se o parâmetro `$id` não foi passado ou se o registro é inexistente, uma `NotFoundException` é lançada pelo `ErrorHandler` do CakePHP.

Em seguida, a action verifica se a requisição é POST ou PUT e caso seja, os dados são usados para atualizar a entidade de artigo em questão ao usar o método `patchEntity()`. Então finalmente usamos o `ArticlesTable` para salvar a entidade.

Segue a view correspondente a action edit:

```
<!-- File: src/Template/Articles/edit.ctp -->

<h1>Edit Article</h1>
<?php
    echo $this->Form->create($article);
    echo $this->Form->input('title');
    echo $this->Form->input('body', ['rows' => '3']);
    echo $this->Form->button(__('Salvar artigo'));
    echo $this->Form->end();
?>
```

Essa view retorna o formulário de edição com os dados populados, juntamente com qualquer mensagem de erro proveniente de validações.

O CakePHP irá determinar se o `save()` vai inserir ou atualizar um registro baseado nos dados da entidade.

Você pode atualizar sua view index com os links para editar artigos:

```
<!-- File: src/Template/Articles/index.ctp  (edit links added) -->

<h1>Blog articles</h1>
<p><?= $this->Html->link("Adicionar artigo", ['action' => 'add']) ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Título</th>
        <th>Criado</th>
        <th>Ações</th>
    </tr>

<!-- Aqui é onde iremos iterar nosso objeto de solicitação $articles, exibindo informações
```

```
<?php foreach ($articles as $article): ?>
    <tr>
        <td><?= $article->id ?></td>
        <td>
            <?= $this->Html->link($article->title, ['action' => 'view', $article->id]) ?>
        </td>
        <td>
            <?= $article->created->format(DATE_RFC850) ?>
        </td>
        <td>
            <?= $this->Html->link('Editar', ['action' => 'edit', $article->id]) ?>
        </td>
    </tr>
<?php endforeach; ?>

</table>
```

Deletando artigos

A seguir, vamos criar uma forma de deletar artigos. Comece com uma action `delete()` no `ArticlesController`:

```
// src/Controller/ArticlesController.php

public function delete($id)
{
    $this->request->allowMethod(['post', 'delete']);

    $article = $this->Articles->get($id);
    if ($this->Articles->delete($article)) {
        $this->Flash->success(__('O artigo com id: {0} foi deletado.', h($id)));
        return $this->redirect(['action' => 'index']);
    }
}
```

Essa lógica deleta o artigo especificado pelo `$id` e usa `$this->Flash->success()` para exibir uma mensagem de confirmação após o redirecionamento para `/articles`. Tentar excluir um registro usando uma requisição GET, fará com que o `allowMethod()` lance uma exceção. Exceções são capturadas pelo gerenciador de exceções do CakePHP e uma página de erro é exibida. Existem muitos *Exceptions* embutidos que podem indicar variados erros HTTP que sua aplicação possa precisar.

Por estarmos executando apenas lógica e redirecionando, essa action não tem uma view. Vamos atualizar nossa view index com links para excluir artigos:

```
<!-- File: src/Template/Articles/index.ctp (delete links added) -->

<h1>Blog articles</h1>
<p><?= $this->Html->link('Adicionar artigo', ['action' => 'add']) ?></p>
<table>
    <tr>
        <th>Id</th>
```

```
        <th>Título</th>
        <th>Criado</th>
        <th>Ações</th>
    </tr>

    <!-- Aqui é onde iremos iterar nosso objeto de solicitação $articles, exibindo informac

    <?php foreach ($articles as $article): ?>
    <tr>
        <td><?= $article->id ?></td>
        <td>
            <?= $this->Html->link($article->title, ['action' => 'view', $article->id]) ?>
        </td>
        <td>
            <?= $article->created->format(DATE_RFC850) ?>
        </td>
        <td>
            <?= $this->Form->postLink(
                'Deletar',
                ['action' => 'delete', $article->id],
                ['confirm' => 'Tem certeza?'])
            ?>
            <?= $this->Html->link('Edit', ['action' => 'edit', $article->id]) ?>
        </td>
    </tr>
    <?php endforeach; ?>

</table>
```

Usar `View\Helper\FormHelper::postLink()` vai criar um link que usa JavaScript para criar uma requisição POST afim de deletar um artigo.

> **Aviso:** Permitir que registros sejam deletados usando requisições GET é perigoso, pois rastreadores na web podem acidentalmente deletar todo o seu conteúdo.

Nota: Esse código da view também usa o `FormHelper` para confirmar a action através de JavaScript.

Rotas

Para muitos o roteamento padrão do CakePHP funciona bem o suficiente. Desenvolvedores que consideram facilidade de uso e SEO irão apreciar a forma como o CakePHP mapeia determinadas URLs para actions específicas. Vamos realizar uma pequena mudança nas rotas neste tutorial.

Para mais informações sobre técnicas avançadas de roteamento, visite *Connecting Routes*.

Por padrão, o CakePHP responde a uma requisição pela raíz do seu site usando o `PagesController`, ao renderizar uma view chamada **home.ctp**. Alternativamente, nós vamos substituir esse comportamento pelo `ArticlesController` ao criar uma regra de roteamento.

A configuração de rotas do CakePHP pode ser encontrada em **config/routes.php**. Você deve comentar ou

remover a linha que define o roteamento padrão:

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Essa linha conecta a URL '/' com a página padrão do CakePHP. Nós queremos que ela conecte-se ao nosso próprio controller, então a substitua por esta:

```
$routes->connect('/', ['controller' => 'Articles', 'action' => 'index']);
```

Isso irá conectar requisições por '/' a action `index()` do nosso `ArticlesController`

Nota: O CakePHP aproveita-se do uso de roteamento reverso. Se com a rota anterior definida você gerar um link com a seguinte estrutura de array: `['controller' => 'Articles', 'action' => 'index']`, a URL resultante será '/'. Portanto, é uma boa ideia sempre usar arrays para URLs, pois assim suas rotas definem o endereço gerado e certificam-se que os links apontem sempre para o mesmo lugar.

Conclusão

Simples, não é? Tenha em mente que esse foi um tutorial básico. O CakePHP tem *muito* mais recursos a oferecer. Não abordamos outros tópicos aqui para manter a simplicidade. Use o restante do manual como um guia para criar aplicações mais ricas.

Agora que você criou uma aplicação básica no CakePHP, você pode continuar no *Tutorial - Criando um Blog - Parte 3*, ou começar seu próprio projeto. Você também pode folhear os `/topics` ou a *API <http://api.cakephp.org/3.0>* para aprender mais sobre o CakePHP.

Se você precisar de ajuda, há muitas formas de conseguir, por favor, visite a página *Onde Conseguir Ajuda* e bem-vindo(a) ao CakePHP!

Leitura complementar

Existem tópicos comuns que as pessoas que estão estudando o CakePHP normalmente visitam a seguir:

1. *Layouts*: Customizando o layout da aplicação
2. *Elements*: Inclusão e reutilização de elementos na view
3. *Geração de código com o Bake*: Gerando código CRUD
4. ***Tutorial - Criando um Blog - Autenticação e Autorização*: Tutorial de** autorização e autenticação

Tutorial - Criando um Blog - Parte 3

Criar uma arvore de Categoria

Vamos continuar o nosso aplicativo de blog e imaginar que queremos categorizar os nossos artigos. Queremos que as categorias sejam ordenadas, e para isso, vamos usar o comportamento de árvore para nos ajudar a organizar as categorias.

Mas primeiro, precisamos modificar nossas tabelas.

Migração de Plugin

Nós vamos usar o plugin de migrações para criar uma tabela em nosso banco de dados. Se você tem a tabela articles no seu banco de dados, apague. Agora abra o arquivo composer.json do seu aplicativo. Normalmente, você veria que o plugin de migração já está requisitando. Se não, addicione atráves da execução:

```
composer require cakephp/migrations:~1.0
```

O plugin de migração agora está na pasta de sua aplicação. Também, adicionar `Plugin::load('Migrations');` para o arquivo bootstrap.php do seu aplicativo.

Uma vez que o plugin está carregado, execute o seguinte comando para criar um arquivo de migração:

```
bin/cake bake migration CreateArticles title:string body:text category_id:integer created m
```

Um arquivo de migração será gerado na pasta /config/Migrations com o seguinte:

```
<?php

use Migrations\AbstractMigration;

class CreateArticlesTable extends AbstractMigration
{
    public function change()
    {
        $table = $this->table('articles');
        $table->addColumn('title', 'string', [
            'default' => null,
            'limit' => 255,
            'null' => false,
        ]);
        $table->addColumn('body', 'text', [
            'default' => null,
            'null' => false,
        ]);
        $table->addColumn('category_id', 'integer', [
            'default' => null,
            'limit' => 11,
            'null' => false,
        ]);
        $table->addColumn('created', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->addColumn('modified', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->create();
```

```
    }
}
```

Executar outro comando para criar uma tabela de categorias. Se você precisar especificar um comprimento de campo, você pode fazê-lo dentro de colchetes no tipo de campo, ou seja:

```
bin/cake bake migration CreateCategories parent_id:integer lft:integer[10] rght:integer[10]
```

Isso irá gerar o seguinte arquivo no config/Migrations:

```
<?php

use Migrations\AbstractMigration;

class CreateCategoriesTable extends AbstractMigration
{
    public function change()
    {
        $table = $this->table('categories');
        $table->addColumn('parent_id', 'integer', [
            'default' => null,
            'limit' => 11,
            'null' => false,
        ]);
        $table->addColumn('lft', 'integer', [
            'default' => null,
            'limit' => 10,
            'null' => false,
        ]);
        $table->addColumn('rght', 'integer', [
            'default' => null,
            'limit' => 10,
            'null' => false,
        ]);
        $table->addColumn('name', 'string', [
            'default' => null,
            'limit' => 100,
            'null' => false,
        ]);
        $table->addColumn('description', 'string', [
            'default' => null,
            'limit' => 255,
            'null' => false,
        ]);
        $table->addColumn('created', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->addColumn('modified', 'datetime', [
            'default' => null,
            'null' => false,
        ]);
        $table->create();
    }
```

```
}
```

Agora que os arquivos de migração estão criadas, você pode editá-los antes de criar suas tabelas. Precisamos mudar o 'null' => false para o campo parent_id com `'null' => true` porque uma categoria de nível superior tem null no parent_id

Execute o seguinte comando para criar suas tabelas:

```
bin/cake migrations migrate
```

Modificando as Tabelas

Com nossas tabelas configuradas, agora podemos nos concentrar em categorizar os nossos artigos.

Supomos que você já tem os arquivos (Tabelas, controladores e modelos dos artigos) da parte 2. Então vamos adicionar as referências a categorias.

Precisamos associar os artigos e categorias juntos nas tabelas. Abra o arquivo src/Model/Table/ArticlesTable.php e adicione o seguinte:

```
// src/Model/Table/ArticlesTable.php
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
        // Just add the belongsTo relation with CategoriesTable
        $this->belongsTo('Categories', [
            'foreignKey' => 'category_id',
        ]);
    }
}
```

Gerar código esqueleto por categorias

Crie todos os arquivos pelo comando bake:

```
bin/cake bake model Categories
bin/cake bake controller Categories
bin/cake bake template Categories
```

A ferramenta bake criou todos os seus arquivos em um piscar de olhos. Você pode fazer uma leitura rápida se quiser familiarizar como o CakePHP funciona.

Nota: Se você estiver no Windows lembre-se de usar em vez de /.

Você vai precisar editar o seguinte em **src/Template/Categories/add.ctp** e **src/Template/Categories/edit.ctp**:

```
echo $this->Form->input('parent_id', [
    'options' => $parentCategories,
    'empty' => 'No parent category'
]);
```

Anexar árvore de compartamento para CategoriesTable

O *TreeBehavior* ajuda você a gerenciar as estruturas de árvore hierárquica na tabela do banco de dados. Usa a lógica MPTT para gerenciar os dados. Estruturas de árvore MPTT são otimizados para lê, o que muitas vezes torna uma boa opção para aplicações pesadas, como ler blogs.

Se você abrir o arquivo src/Model/Table/CategoriesTable.php, você verá que o TreeBehavior foi anexado a sua CategoriesTable no método initialize(). Bake acrescenta esse comportamento para todas as tabelas que contêm lft e colunas rght:

```
$this->addBehavior('Tree');
```

Com o TreeBehavior anexado você vai ser capaz de acessar alguns recursos como a reordenação das categorias. Vamos ver isso em um momento.

Mas, por agora, você tem que remover as seguintes entradas em seus Categorias de adicionar e editar arquivos de modelo:

```
echo $this->Form->input('lft');
echo $this->Form->input('rght');
```

Além disso, você deve desabilitar ou remover o requirePresence do validador, tanto para a `lft` e `rght` nas colunas em seu modelo CategoriesTable:

```
public function validationDefault(Validator $validator)
{
    $validator
        ->add('id', 'valid', ['rule' => 'numeric'])
        ->allowEmpty('id', 'create');

    $validator
        ->add('lft', 'valid', ['rule' => 'numeric'])
    //    ->requirePresence('lft', 'create')
        ->notEmpty('lft');

    $validator
        ->add('rght', 'valid', ['rule' => 'numeric'])
    //    ->requirePresence('rght', 'create')
        ->notEmpty('rght');
}
```

Esses campos são automaticamente gerenciados pelo TreeBehavior quando uma categoria é salvo.

Usando seu navegador, adicione algumas novas categorias usando os `/yoursite/categories/add` ação do controlador.

Reordenar categorias com TreeBahavior

Em seu arquivo de modelo de índices de categorias, você pode listar as categorias e reordená-los.

Vamos modificar o método de índice em sua CategoriesController.php e adicionar moveUp() e moveDown() para ser capaz de reordenar as categorias na árvore:

```
class CategoriesController extends AppController
{
    public function index()
    {
        $categories = $this->Categories->find()
            ->order(['lft' => 'ASC']);
        $this->set(compact('categories'));
        $this->set('_serialize', ['categories']);
    }

    public function moveUp($id = null)
    {
        $this->request->allowMethod(['post', 'put']);
        $category = $this->Categories->get($id);
        if ($this->Categories->moveUp($category)) {
            $this->Flash->success('The category has been moved Up.');
        } else {
            $this->Flash->error('The category could not be moved up. Please, try again.');
        }
        return $this->redirect($this->referer(['action' => 'index']));
    }

    public function moveDown($id = null)
    {
        $this->request->allowMethod(['post', 'put']);
        $category = $this->Categories->get($id);
        if ($this->Categories->moveDown($category)) {
            $this->Flash->success('The category has been moved down.');
        } else {
            $this->Flash->error('The category could not be moved down. Please, try again.')
        }
        return $this->redirect($this->referer(['action' => 'index']));
    }
}
```

Em src/Template/Categories/index.ctp substituir o conteúdo existente com:

```
<div class="actions large-2 medium-3 columns">
    <h3><?= __('Actions') ?></h3>
    <ul class="side-nav">
        <li><?= $this->Html->link(__('New Category'), ['action' => 'add']) ?></li>
    </ul>
</div>
<div class="categories index large-10 medium-9 columns">
    <table cellpadding="0" cellspacing="0">
    <thead>
        <tr>
```

```
            <th>Id</th>
            <th>Parent Id</th>
            <th>Lft</th>
            <th>Rght</th>
            <th>Name</th>
            <th>Description</th>
            <th>Created</th>
            <th class="actions"><?= __('Actions') ?></th>
        </tr>
    </thead>
    <tbody>
    <?php foreach ($categories as $category): ?>
        <tr>
            <td><?= $category->id ?></td>
            <td><?= $category->parent_id ?></td>
            <td><?= $category->lft ?></td>
            <td><?= $category->rght ?></td>
            <td><?= h($category->name) ?></td>
            <td><?= h($category->description) ?></td>
            <td><?= h($category->created) ?></td>
            <td class="actions">
                <?= $this->Html->link(__('View'), ['action' => 'view', $category->id]) ?>
                <?= $this->Html->link(__('Edit'), ['action' => 'edit', $category->id]) ?>
                <?= $this->Form->postLink(__('Delete'), ['action' => 'delete', $category->
                <?= $this->Form->postLink(__('Move down'), ['action' => 'moveDown', $categ
                <?= $this->Form->postLink(__('Move up'), ['action' => 'moveUp', $category->
            </td>
        </tr>
    <?php endforeach; ?>
    </tbody>
    </table>
</div>
```

Modificando o ArticlesController

Em nossa ArticlesController, vamos obter a lista de todas as categorias. Isto irá permitir-nos para escolher uma categoria para um artigo ao criar ou editar ele:

```
// src/Controller/ArticlesController.php
namespace App\Controller;

use Cake\Network\Exception\NotFoundException;

class ArticlesController extends AppController
{
    // ...

    public function add()
    {
        $article = $this->Articles->newEntity();
        if ($this->request->is('post')) {
            $article = $this->Articles->patchEntity($article, $this->request->data);
```

```
            if ($this->Articles->save($article)) {
                $this->Flash->success(__('Your article has been saved.'));
                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('Unable to add your article.'));
        }
        $this->set('article', $article);

        // Just added the categories list to be able to choose
        // one category for an article
        $categories = $this->Articles->Categories->find('treeList');
        $this->set(compact('categories'));
    }
}
```

Modificando os artigos Templates

O artigo adicionado deveria se parecer como isto:

```
<!-- File: src/Template/Articles/add.ctp -->

<h1>Add Article</h1>
<?php
echo $this->Form->create($article);
// just added the categories input
echo $this->Form->input('category_id');
echo $this->Form->input('title');
echo $this->Form->input('body', ['rows' => '3']);
echo $this->Form->button(__('Save Article'));
echo $this->Form->end();
```

Quando você vai para o endereço `/yoursite/articles/add` você deve ver uma lista de categorias para escolher.

Tutorial - Criando um Blog - Autenticação e Autorização

Continuando com o exemplo de *Tutorial - Criando um Blog - Parte 1*, imagine que queríamos garantir o acesso a certas URLs, com base no usuário logado. Temos também uma outra exigência: permitir que o nosso blog para tenha vários autores que podem criar, editar e excluir seus próprios artigos, e bloquear para que outros autores não façam alterações nos artigos que não lhes pertencem.

Criando todo o código relacionado ao Usuário

Primeiro, vamos criar uma nova tabela no banco de dados do blog para armazenar dados de nossos usuários:

```
CREATE TABLE users (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
```

```
    password VARCHAR(255),
    role VARCHAR(20),
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);
```

Respeitado as convenções do CakePHP para nomear tabelas, mas também aproveitando de outras convenção: Usando as colunas `username` e `password` da tabela de usuários, CakePHP será capaz de configurar automaticamente a maioria das coisas para nós, na implementação do login do usuário.

O próximo passo é criar a nossa classe UsersTable, responsável por encontrar, salvar e validar os dados do usuário:

```
// src/Model/Table/UsersTable.php
namespace App\Model\Table;

use Cake\ORM\Table;
use Cake\Validation\Validator;

class UsersTable extends Table
{

    public function validationDefault(Validator $validator)
    {
        return $validator
            ->notEmpty('username', 'Usuário é necessário')
            ->notEmpty('password', 'Senha é necessária')
            ->notEmpty('role', 'Função é necessária')
            ->add('role', 'inList', [
                'rule' => ['inList', ['admin', 'author']],
                'message' => 'Por favor informe uma função válida'
            ]);
    }

}
```

Vamos também criar o nosso UsersController. O conteúdo a seguir corresponde a partes de uma classe UsersController básica gerado atráves do utilitário de geração de código `bake` fornecido com CakePHP:

```
// src/Controller/UsersController.php

namespace App\Controller;

use App\Controller\AppController;
use Cake\Event\Event;

class UsersController extends AppController
{

    public function beforeFilter(Event $event)
    {
        parent::beforeFilter($event);
        $this->Auth->allow('add');
    }
```

```
     public function index()
     {
         $this->set('users', $this->Users->find('all'));
    }

    public function view($id)
    {
        $user = $this->Users->get($id);
        $this->set(compact('user'));
    }

    public function add()
    {
        $user = $this->Users->newEntity();
        if ($this->request->is('post')) {
            $user = $this->Users->patchEntity($user, $this->request->data);
            if ($this->Users->save($user)) {
                $this->Flash->success(__('O usuário foi salvo.'));
                return $this->redirect(['action' => 'add']);
            }
            $this->Flash->error(__('Não é possível adicionar o usuário.'));
        }
        $this->set('user', $user);
    }

}
```

Da mesma maneira que criamos as `views` para os nossos artigos usando a ferramenta de geração de código, podemos implementar as `views` do usuário. Para o propósito deste tutorial, vamos mostrar apenas o add.ctp:

```
<!-- src/Template/Users/add.ctp -->

<div class="users form">
<?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Add User') ?></legend>
        <?= $this->Form->input('username') ?>
        <?= $this->Form->input('password') ?>
        <?= $this->Form->input('role', [
            'options' => ['admin' => 'Admin', 'author' => 'Author']
        ]) ?>
   </fieldset>
<?= $this->Form->button(__('Submit')); ?>
<?= $this->Form->end() ?>
</div>
```

Autenticação (Login e Logout)

Agora estamos prontos para adicionar a nossa camada de autenticação. Em CakePHP isso é tratado pelo `Cake\Controller\Component\AuthComponent`, uma classe responsável por exigir o `login` para

determinadas ações, a manipulação de `login` e `logout` de usuário, e também permite as ações para que estão autorizados.

Para adicionar este componente em sua aplicação abra o arquivos **src/Controller/AppController.php** e adicione as seguintes linha:

```
// src/Controller/AppController.php

namespace App\Controller;

use Cake\Controller\Controller;
use Cake\Event\Event;

class AppController extends Controller
{
    //...

    public function initialize()
    {
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'loginRedirect' => [
                'controller' => 'Articles',
                'action' => 'index'
            ],
            'logoutRedirect' => [
                'controller' => 'Pages',
                'action' => 'display',
                'home'
            ]
        ]);
    }

    public function beforeFilter(Event $event)
    {
        $this->Auth->allow(['index', 'view', 'display']);
    }
    //...
}
```

Não há muito para ser configurado, como usamos as convenções para a tabela de usuários. Nós apenas configuramos as URLs que serão carregados após o `login` e `logout`, estás ações são realizadas no nosso caso para os `/articles/` e `/` respectivamente.

O que fizemos na função `beforeFilter()` foi dizer ao `AuthComponent` para não exigir `login` em todos `index()` e `view()`, em cada controlador. Queremos que os nossos visitantes sejam capaz de ler e listar as entradas sem registrar-se no site.

Agora, precisamos ser capaz de registrar novos usuários, salvar seu `username` e `password`, e mais importante, o hash da senha para que ele não seja armazenado como texto simples no nosso banco de dados. Vamos dizer ao `AuthComponet` para permitir que usuários deslogados acessem a função add e execute as ações de `login` e `logout`:

```
// src/Controller/UsersController.php

public function beforeFilter(Event $event)
{
    parent::beforeFilter($event);
    // Permitir aos usuários se registrarem e efetuar logout.
    // Você não deve adicionar a ação de "login" a lista de permissões.
    // Isto pode causar problemas com o funcionamento normal do AuthComponent.
    $this->Auth->allow(['add', 'logout']);
}

public function login()
{
    if ($this->request->is('post')) {
        $user = $this->Auth->identify();
        if ($user) {
            $this->Auth->setUser($user);
            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Flash->error(__('Usuário ou senha ínvalido, tente novamente'));
    }
}

public function logout()
{
    return $this->redirect($this->Auth->logout());
}
```

O hashing da senha ainda não está feito, precisamos de uma classe a fim de manipular sua geração. Crie o arquivo **src/Model/Entity/User.php** e adicione a seguinte trecho:

```
// src/Model/Entity/User.php
namespace App\Model\Entity;

use Cake\Auth\DefaultPasswordHasher;
use Cake\ORM\Entity;

class User extends Entity
{

    // Gera conjunto de todos os campos exceto o com a chave primária.
    protected $_accessible = [
        '*' => true,
        'id' => false
    ];

    // ...

    protected function _setPassword($password)
    {
        return (new DefaultPasswordHasher)->hash($password);
    }
```

```
    // ...
}
```

Agora, a senha criptografada usando a classe `DefaultPasswordHasher`. Está faltando apenas o arquivo para exibição da tela de login. Abra o arquivo **src/Template/Users/login.ctp** e adicione as seguintes linhas:

```
<!-- File: src/Template/Users/login.ctp -->

<div class="users form">
<?= $this->Flash->render('auth') ?>
<?= $this->Form->create() ?>
    <fieldset>
        <legend><?= __('Por favor informe seu usuário e senha') ?></legend>
        <?= $this->Form->input('username') ?>
        <?= $this->Form->input('password') ?>
    </fieldset>
<?= $this->Form->button(__('Login')); ?>
<?= $this->Form->end() ?>
</div>
```

Agora você pode registrar um novo usuário, acessando a URL `/users/add` e faça login com o usuário recém-criado, indo para a URL `/users/login`. Além disso, tente acessar qualquer outro URL que não tenha sido explicitamente permitido, como `/articles/add`, você vai ver que o aplicativo redireciona automaticamente para a página de login.

E é isso! Parece simples demais para ser verdade. Vamos voltar um pouco para explicar o que aconteceu. A função `beforeFilter()` está falando para o AuthComponent não solicitar um login para a ação `add()` em adição as ações `index()` e `view()` que foram prontamente autorizadas na função `beforeFilter()` do AppController.

A ação `login()` chama a função `$this->Auth->identify()` da AuthComponent, que funciona sem qualquer outra configuração porque estamos seguindo convenções, como mencionado anteriormente. Ou seja, ter uma tabela de usuários com um `username` e uma coluna de `password`, e usamos um form para postar os dados do usuário para o controller. Esta função retorna se o login foi bem sucedido ou não, e caso ela retorne sucesso, então nós redirecionamos o usuário para a URL que configuramos quando adicionamos o AuthComponent em nossa aplicação.

O logout funciona quando acessamos a URL `/users/logout` que irá redirecionar o usuário para a url configurada em logoutUrl. Essa url é acionada quando a função `AuthComponent::logout()`.

Autorização (quem tem permissão para acessar o que)

Como afirmado anteriormente, nós estamos convertendo esse blog em uma ferramenta multi usuário de autoria, e para fazer isso, precisamos modificar a tabela de artigos um pouco para adicionar a referência à tabela de Usuários:

```
ALTER TABLE articles ADD COLUMN user_id INT(11);
```

Além disso, uma pequena mudança no ArticlesController é necessário para armazenar o usuário conectado no momento como uma referência para o artigo criado:

```
// src/Controller/ArticlesController.php

public function add()
{
    $article = $this->Articles->newEntity();
    if ($this->request->is('post')) {
        $article = $this->Articles->patchEntity($article, $this->request->data);
        // Adicione esta linha
        $article->user_id = $this->Auth->user('id');
        // Você também pode fazer o seguinte
        //$newData = ['user_id' => $this->Auth->user('id')];
        //$article = $this->Articles->patchEntity($article, $newData);
        if ($this->Articles->save($article)) {
            $this->Flash->success(__('Seu artigo foi salvo.'));
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('Não foi possível adicionar seu artigo.'));
    }
    $this->set('article', $article);
}
```

A função `user()` fornecida pelo componente retorna qualquer coluna do usuário logado no momento. Nós usamos esse metódo para adicionar a informação dentro de request data para que ela seja salva.

Vamos garantir que nossa app evite que alguns autores editem ou apaguem posts de outros. Uma regra básica para nossa aplicação é que usuários admin possam acessar qualquer url, enquanto usuários normais (o papel author) podem somente acessar as actions permitidas. Abra novamente a classe AppController e adicione um pouco mais de opções para as configurações do Auth:

```
// src/Controller/AppController.php

public function initialize()
{
    $this->loadComponent('Flash');
    $this->loadComponent('Auth', [
        'authorize' => ['Controller'], // Adicione está linha
        'loginRedirect' => [
            'controller' => 'Articles',
            'action' => 'index'
        ],
        'logoutRedirect' => [
            'controller' => 'Pages',
            'action' => 'display',
            'home'
        ]
    ]);
}

public function isAuthorized($user)
{
    // Admin pode acessar todas as actions
    if (isset($user['role']) && $user['role'] === 'admin') {
        return true;
```

```
    }

    // Bloqueia acesso por padrão
    return false;
}
```

Acabamos de criar um mecanismo de autorização muito simples. Nesse caso os usuários com papel `admin` poderão acessar qualquer url no site quando estiverem logados, mas o restante dos usuários (author) não podem acessar qualquer coisa diferente dos usuários não logados.

Isso não é exatamente o que nós queremos, por isso precisamos corrigir nosso metódo `isAuthorized()` para fornecer mais regras. Mas ao invés de fazer isso no AppController, vamos delegar a cada controller para suprir essas regras extras. As regras que adicionaremos para o `add` de ArticlesController deve permitir ao autores criarem os posts mas evitar a edição de posts que não sejam deles. Abra o arquivo **src/Controller/ArticlesController.php** e adicione o seguinte conteúdo:

```
// src/Controller/ArticlesController.php

public function isAuthorized($user)
{
    // Todos os usuários registrados podem adicionar artigos
    if ($this->request->action === 'add') {
        return true;
    }

    // Apenas o proprietário do artigo pode editar e excluí
    if (in_array($this->request->action, ['edit', 'delete'])) {
        $articleId = (int)$this->request->params['pass'][0];
        if ($this->Articles->isOwnedBy($articleId, $user['id'])) {
            return true;
        }
    }

    return parent::isAuthorized($user);
}
```

Estamos sobrescrevendo a chamada `isAuthorized()` ``do AppController e internamente verificando na classe pai se o usuário está autorizado. Caso não esteja, então apenas permitem acessar a action ``add, e condicionalmente action `edit` e `delete`. Uma última coisa não foi implementada. Para dizer ou não se o usuário está autorizado a editar o artigo, nós estamos chamando uma função `isOwnedBy()` na tabela artigos. Vamos, então, implementar essa função:

```
// src/Model/Table/ArticlesTable.php

public function isOwnedBy($articleId, $userId)
{
    return $this->exists(['id' => $articleId, 'user_id' => $userId]);
}
```

Isso conclui então nossa autorização simples e nosso tutorial de autorização. Para garantir o UsersController você pode seguir as mesmas técnicas que usamos para ArticlesController, você também pode ser mais criativo e codificar algumas coisas mais gerais no AppController para suas próprias regras baseadas em

papéis.

Se precisar de mais controle, nós sugerimos que leia o guia completo do Auth *Authentication* seção onde você encontrará mais sobre a configuração do componente, criação de classes de Autorização customizadas, e muito mais.

Sugerimos as seguintes leituras

1. *Geração de código com o Bake* Generating basic CRUD code
2. *Authentication*: User registration and login

CAPÍTULO 5

Contribuindo

Existem várias maneiras de contribuir com o CakePHP. As seções abaixo irão abordar estas formas de contribuição:

Documentação

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Tickets

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Código

[1] https://github.com/cakephp/docs
[2] https://github.com/cakephp/docs

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[3] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Padrões de código

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[4] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Guia de retrocompatibilidade

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[5] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[3] https://github.com/cakephp/docs
[4] https://github.com/cakephp/docs
[5] https://github.com/cakephp/docs

CAPÍTULO 6

Instalação

O CakePHP é rápido e fácil de instalar. Os requisitos mínimos são um servidor web e uma cópia do CakePHP, só isso! Apesar deste manual focar principalmente na configuração do Apache (porquê ele é o mais simples de instalar e configurar), o CakePHP vai ser executado em uma série de servidores web como nginx, LightHTTPD, ou Microsoft IIS.

Requisitos

- HTTP Server. Por exemplo: Apache. De preferência com mod_rewrite ativo, mas não é obrigatório.
- PHP 5.5.9 ou superior.
- extensão mbstring
- extensão intl

Nota: Tanto no XAMPP quanto no WAMP, as extensões mcrypt e mbstring são setadas por padrão.

Se você estiver usando o XAMPP, já tem a extensão intl inclusa, mas é preciso descomentar a linha `extension=php_intl.dll` no arquivo `php.ini` e então, reiniciar o servidor através do painel de controle do XAMPP.

Caso você esteja usando o WAMP, a extensão intl está "ativa" por padrão, mas não está funcional. Para fazê-la funcionar, você deve ir à pasta do php (que por padrão é) `C:\wamp\bin\php\php{version}`, copiar todos os arquivos que se pareçam com `icu***.dll` e colá-los no diretório "bin" do apache `C:\wamp\bin\apache\apache{version}\bin`. Reiniciando todos os serviços a extensão já deve ficar ok.

Apesar de um mecanismo de banco de dados não ser exigido, nós imaginamos que a maioria das aplicações irá utilizar um. O CakePHP suporta uma variedade de mecanismos de armazenamento de banco de dados:

- MySQL (5.1.10 ou superior)
- PostgreSQL

- Microsoft SQL Server (2008 ou superior)
- SQLite 3

Nota: Todos os drivers inclusos internamente requerem PDO. Você deve assegurar-se que possui a extensão PDO correta instalada.

Instalando o CakePHP

O CakePHP utiliza Composer[1], uma ferramenta de gerenciamento de dependências para PHP 5.3+, como o método suportado oficial para instalação.

Primeiramente, você precisará baixar e instalar o Composer se não o fez anteriormente. Se você tem cURL instalada, é tão fácil quanto executar o seguinte:

```
curl -s https://getcomposer.org/installer | php
```

Ou, você pode baixar `composer.phar` do Site oficial do Composer[2].

Para sistemas Windows, você pode baixar o instalador aqui[3]. Mais instruções para o instalador Windows do Composer podem ser encontradas dentro do LEIA-ME aqui[4].

Agora que você baixou e instalou o Composer, você pode receber uma nova aplicação CakePHP executando:

```
php composer.phar create-project --prefer-dist cakephp/app [app_name]
```

Ou se o Composer estiver instalado globalmente:

```
composer create-project --prefer-dist cakephp/app [app_name]
```

Uma vez que o Composer terminar de baixar o esqueleto da aplicação e o núcleo da biblioteca CakePHP, você deve ter uma aplicação funcional instalada via Composer. Esteja certo de manter os arquivos composer.json e composer.lock com o restante do seu código fonte.

You can now visit the path to where you installed your CakePHP application and see the setup traffic lights.

Mantendo sincronização com as últimas alterações no CakePHP

Se você quer se manter atualizado com as últimas mudanças no CakePHP, você pode adicionar o seguinte ao `composer.json` de sua aplicação:

```
"require": {
    "cakephp/cakephp": "dev-master"
}
```

[1] http://getcomposer.org
[2] https://getcomposer.org/download/
[3] https://github.com/composer/windows-setup/releases/
[4] https://github.com/composer/windows-setup

Onde <branch> é o nome do branch que você segue. Toda vez que você executar `php composer.phar update` você receberá as últimas atualizações do branch escolhido.

Permissões

O CakePHP utiliza o diretório `tmp` para diversas operações. Descrição de models, views armazenadas em cache e informações de sessão são apenas alguns exemplos. O diretório `logs` é utilizado para escrever arquivos de log pelo mecanismo padrão `FileLog`.

Como tal, certifique-se que os diretórios `logs`, `tmp` e todos os seus sub-diretórios em sua instalação CakePHP são graváveis pelo usuário relacionado ao servidor web. O processo de instalação do Composer faz `tmp` e seus sub-diretórios globalmente graváveis para obter as coisas funcionando rapidamente, mas você pode atualizar as permissões para melhor segurança e mantê-los graváveis apenas para o usuário relacionado ao servidor web.

Um problema comum é que os diretórios e sub-diretórios de `logs` e `tmp` devem ser graváveis tanto pelo servidor quanto pelo usuário da linha de comando. Em um sistema UNIX, se seu usuário relacionado ao servidor web é diferente do seu usuário da linha de comando, você pode executar somente uma vez os seguintes comandos a partir do diretório da sua aplicação para assegurar que as permissões serão configuradas corretamente:

```
HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' | grep -v root | h
setfacl -R -m u:${HTTPDUSER}:rwx tmp
setfacl -R -d -m u:${HTTPDUSER}:rwx tmp
setfacl -R -m u:${HTTPDUSER}:rwx logs
setfacl -R -d -m u:${HTTPDUSER}:rwx logs
```

Servidor de Desenvolvimento

Uma instalação de desenvolvimento é o método mais rápido de configurar o CakePHP. Neste exemplo, nós vamos utilizar o console CakePHP para executar o servidor integrado do PHP que vai tornar sua aplicação disponível em `http://host:port`. A partir do diretório da aplicação, execute:

```
bin/cake server
```

Por padrão, sem nenhum argumento fornecido, isso vai disponibilizar a sua aplicação em `http://localhost:8765/`.

Se você tem algo conflitante com `localhost` ou porta `8765`, você pode dizer ao console CakePHP para executar o servidor web em um host e/ou porta específica utilizando os seguintes argumentos:

```
bin/cake server -H 192.168.13.37 -p 5673
```

Isto irá disponibilizar sua aplicação em `http://192.168.13.37:5673/`.

É isso aí! Sua aplicação CakePHP está instalada e funcionando sem ter que configurar um servidor web.

Aviso: O servidor de desenvolvimento *nunca* deve ser usado em um ambiente de produção. Destina-se apenas como um servidor de desenvolvimento básico.

Se você preferir usar um servidor web real, você deve ser capaz de mover a instalação do CakePHP (incluindo os arquivos ocultos) para dentro do diretório raiz do seu servidor web. Você deve, então, ser capaz de apontar seu navegador para o diretório que você moveu os arquivos para dentro e ver a aplicação em ação.

Produção

Uma instalação de produção é uma forma mais flexível de configurar o CakePHP. Usar este método permite total domínio para agir como uma aplicação CakePHP singular. Este exemplo o ajudará a instalar o CakePHP em qualquer lugar em seu sistema de arquivos e torná-lo disponível em http://www.example.com. Note que esta instalação pode exigir os direitos de alterar o `DocumentRoot` em servidores web Apache.

Depois de instalar a aplicação usando um dos métodos acima no diretório de sua escolha - vamos supor que você escolheu /cake_install - sua configuração de produção será parecida com esta no sistema de arquivos:

```
/cake_install/
    bin/
    config/
    logs/
    plugins/
    src/
    tests/
    tmp/
    vendor/
    webroot/ (esse diretório é definido como DocumentRoot)
    .gitignore
    .htaccess
    .travis.yml
    composer.json
    index.php
    phpunit.xml.dist
    README.md
```

Desenvolvedores utilizando Apache devem definir a diretiva `DocumentRoot` pelo domínio para:

```
DocumentRoot /cake_install/webroot
```

Se o seu servidor web está configurado corretamente, agora você deve encontrar sua aplicação CakePHP acessível em http://www.example.com.

Aquecendo

Tudo bem, vamos ver o CakePHP em ação. Dependendo de qual configuração você usou, você deve apontar seu navegador para http://example.com/ ou http://localhost:8765/. Nesse ponto, você será apresentado à página home padrão do CakePHP e uma mensagem que diz a você o estado da sua conexão atual com o banco de dados.

Parabéns! Você está pronto para *create your first CakePHP application*.

Reescrita de URL

Apache

Apesar do CakePHP ser construído para trabalhar com mod_rewrite fora da caixa, e normalmente o faz, nos atentamos que aluns usuários lutam para conseguir fazer tudo funcionar bem em seus sistemas.

Aqui estão algumas coisas que você poderia tentar para conseguir tudo rodando corretamente. Primeiramente observe seu httpd.conf. (Tenha certeza que você está editando o httpd.conf do sistema ao invés de um usuário, ou site específico.)

Esses arquivos podem variar entre diferentes distribuições e versões do Apache. Você também pode pesquisar em http://wiki.apache.org/httpd/DistrosDefaultLayout para maiores informações.

1. Tenha certeza que a sobreescrita do .htaccess está permitida e que AllowOverride está definido para All no correto DocumentRoot. Você deve ver algo similar a:

```
# Cada diretório ao qual o Apache tenha acesso pode ser configurado com respeito
# a quais serviços e recursos estão permitidos e/ou desabilitados neste
# diretório (e seus sub-diretórios).
#
# Primeiro, nós configuramos o "default" para ser um conjunto bem restrito de
# recursos.
<Directory />
    Options FollowSymLinks
    AllowOverride All
#    Order deny,allow
#    Deny from all
</Directory>
```

2. Certifique-se que o mod_rewrite está sendo carregado corretamente. Você deve ver algo como:

```
LoadModule rewrite_module libexec/apache2/mod_rewrite.so
```

   Em muitos sistemas estará comentado por padrão, então você pode apenas remover os símbolos #.

   Depois de fazer as mudanças, reinicie o Apache para certificar-se que as configurações estão ativas.

   Verifique se os seus arquivos .htaccess estão realmente nos diretórios corretos. Alguns sistemas operacionais tratam arquivos iniciados com '.' como ocultos e portanto, não os copia.

3. Certifique-se de sua cópia do CakePHP vir da seção de downloads do site ou do nosso repositório Git, e que foi descompactado corretamente, verificando os arquivos .htaccess.

   O diretório app do CakePHP (será copiado para o diretório mais alto de sua aplicação através do bake):

```
<IfModule mod_rewrite.c>
   RewriteEngine on
   RewriteRule    ^$    webroot/    [L]
```

```
    RewriteRule    (.*) webroot/$1    [L]
</IfModule>
```

O diretório webroot do CakePHP (será copiado para a raíz de sua aplicação através do bake):

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Se o seu site CakePHP ainda possuir problemas com mod_rewrite, você pode tentar modificar as configurações para Virtual Hosts. No Ubuntu, edita o arquivo /etc/apache2/sites-available/default (a localização depende da distribuição). Nesse arquivo, certifique-se que `AllowOverride None` seja modificado para `AllowOverride All`, então você terá:

```
<Directory />
    Options FollowSymLinks
    AllowOverride All
</Directory>
<Directory /var/www>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride All
    Order Allow,Deny
    Allow from all
</Directory>
```

No Mac OSX, outra solução é usar a ferramenta virtualhostx[5] para fazer um Virtual Host apontar para o seu diretório.

Para muitos serviços de hospedagem (GoDaddy, land1), seu servidor web é na verdade oferecido a partir de um diretório de usuário que já utiliza mod_rewrite. Se você está instalando o CakePHP em um diretório de usuário (http://example.com/~username/cakephp/), ou qualquer outra estrutura URL que já utilize mod_rewrite, você precisará adicionar declarações RewriteBase para os arquivos .htaccess que o CakePHP utiliza. (.htaccess, webroot/.htaccess).

Isso pode ser adicionado na mesma seção com a diretiva RewriteEngine, por exemplo, seu arquivo webroot/.htaccess ficaria como:

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /path/to/app
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Os detalhes dessas mudanças vão depender da sua configuração, e podem incluir coisas adicionais que não estão relacionadas ao CakePHP. Por favor, busque pela documentação online do Apache para mais informações.

4. (Opcional) Para melhorar a configuração de produção, você deve prevenir conteúdos inváidos de serem analisados pelo CakePHP. Modifique seu webroot/.htaccess para algo como:

[5] http://clickontyler.com/virtualhostx/

```
    <IfModule mod_rewrite.c>
        RewriteEngine On
        RewriteBase /path/to/app/
        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteCond %{REQUEST_URI} !^/(webroot/)?(img|css|js)/(.*)$
        RewriteRule ^ index.php [L]
    </IfModule>
```

Isto irá simplesmente prevenir conteúdo incorreto de ser enviado para o index.php e então exibir sua página de erro 404 do servidor web.

Adicionalmente você pode criar uma página HTML de erro 404 correspondente, ou utilizar a padrão do CakePHP ao adicionar uma diretiva `ErrorDocument`:

```
    ErrorDocument 404 /404-not-found
```

nginx

nginx não utiliza arquivos .htaccess como o Apache, então é necessário criar as reescritas de URL na configuração de sites disponíveis. Dependendo da sua configuração, você precisará modificar isso, mas pelo menos, você vai precisar do PHP rodando como uma instância FastCGI:

```
server {
    listen   80;
    server_name www.example.com;
    rewrite ^(.*) http://example.com$1 permanent;
}

server {
    listen   80;
    server_name example.com;

    # root directive should be global
    root   /var/www/example.com/public/webroot/;
    index  index.php;

    access_log /var/www/example.com/log/access.log;
    error_log /var/www/example.com/log/error.log;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        include /etc/nginx/fastcgi_params;
        fastcgi_pass    127.0.0.1:9000;
        fastcgi_index   index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }
}
```

IIS7 (Windows hosts)

IIS7 não suporta nativamente arquivos .htaccess. Mesmo existindo add-ons que adicionam esse suporte, você também pode importar as regras .htaccess no IIS para utilizar as reescritas nativas do CakePHP. Para isso, siga os seguintes passos:

1. Utilize o Microsoft's Web Platform Installer[6] para instalar o Rewrite Module 2.0[7] ou baixe-o diretamente (32-bit[8] / 64-bit[9]).
2. Crie um novo arquivo chamado web.config em seu diretório raiz do CakePHP.
3. Utilize o Notepad ou qualquer editor seguro XML para copiar o seguinte código em seu novo arquivo web.config:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <rewrite>
            <rules>
                <rule name="Exclude direct access to webroot/*"
                  stopProcessing="true">
                    <match url="^webroot/(.*)$" ignoreCase="false" />
                    <action type="None" />
                </rule>
                <rule name="Rewrite routed access to assets(img, css, files, js, favic
                  stopProcessing="true">
                    <match url="^(img|css|files|js|favicon.ico)(.*)$" />
                    <action type="Rewrite" url="webroot/{R:1}{R:2}"
                      appendQueryString="false" />
                </rule>
                <rule name="Rewrite requested file/folder to index.php"
                  stopProcessing="true">
                    <match url="^(.*)$" ignoreCase="false" />
                    <action type="Rewrite" url="index.php"
                      appendQueryString="true" />
                </rule>
            </rules>
        </rewrite>
    </system.webServer>
</configuration>
```

Uma vez que o arquivo web.config é criado com as regras amigáveis de reescrita do IIS, os links, CSS, JavaScript, e roteamento do CakePHP agora devem funcionar corretamente.

Não posso utilizar Reescrita de URL

Se você não quer ou não pode ter mod_rewrite (ou algum outro módulo compatível) funcionando no seu servidor, você precisará utilizar as URLs amigáveis natívas do CakePHP. No **config/app.php**, descomente

[6] http://www.microsoft.com/web/downloads/platform.aspx
[7] http://www.iis.net/downloads/microsoft/url-rewrite
[8] http://www.microsoft.com/en-us/download/details.aspx?id=5747
[9] http://www.microsoft.com/en-us/download/details.aspx?id=7435

a linha que se parece como:

```
'App' => [
    // ...
    // 'baseUrl' => env('SCRIPT_NAME'),
]
```

Também remova esses arquivos .htaccess:

```
/.htaccess
webroot/.htaccess
```

Isso fará suas URLs parecem como www.example.com/index.php/controllername/actionname/param ao invés de www.example.com/controllername/actionname/param.

CAPÍTULO 7

Configuração

Additional Class Paths

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 8

Roteamento

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Connecting Routes

Using Named Routes

Filtros do Dispatcher

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs
[2] https://github.com/cakephp/docs

CAPÍTULO 9

Objetos de requisição e resposta

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Request

[1] https://github.com/cakephp/docs

CAPÍTULO 10

Controllers (Controladores)

class `Cake\Controller\`**`Controller`**

Os controllers (controladores) correspondem ao 'C' no padrão MVC. Após o roteamento ter sido aplicado e o controller correto encontrado, a ação do controller é chamada. Seu controller deve lidar com a interpretação dos dados de uma requisição, certificando-se que os models corretos são chamados e a resposta ou view esperada seja exibida. Os controllers podem ser vistos como intermediários entre a camada Model e View. Você vai querer manter seus controllers magros e seus Models gordos. Isso lhe ajudará a reutilizar seu código e testá-los mais facilmente.

Mais comumente, controllers são usados para gerenciar a lógica de um único model. Por exemplo, se você está construindo um site para uma padaria online, você pode ter um `RecipesController` e um `IngredientsController` gerenciando suas receitas e seus ingredientes. No CakePHP, controllers são nomeados de acordo com o model que manipulam. É também absolutamente possível ter controllers que usam mais de um model.

Os controllers da sua aplicação são classes que estendem a classe `AppController`, a qual por sua vez estende a classe do core `Controller`. A classe `AppController` pode ser definida em **src/Controller/AppController.php** e deve conter métodos que são compartilhados entre todos os controllers de sua aplicação.

Os controllers fornecem uma série de métodos que lidam com requisições. Estas são chamados de *actions*. Por padrão, todos os métodos públicos em um controller são uma action e acessíveis por uma URL. Uma action é responsável por interpretar a requisição e criar a resposta. Normalmente as respostas são na forma de uma view renderizada, mas também existem outras formas de criar respostas.

O App Controller

Como mencionado anteriormente, a classe `AppController` é a mãe de todos os outros controllers da sua aplicação. A própria `AppController` é estendida da classe `Cake\Controller\Controller` incluída no CakePHP. Assim sendo, `AppController` é definida em **src/Controller/AppController.php** como a seguir:

```
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
}
```

Os atributos e métodos criados em seu `AppController` vão estar disponíveis para todos os controllers que a extendam. Components (sobre os quais você irá aprender mais tarde) são a melhor alternativa para códigos usados por muitos (mas não necessariamente em todos) controllers.

Você pode usar seu `AppController` para carregar components que serão usados em cada controller de sua aplicação. O CakePHP oferece um método `initialize()` que é invocado ao final do construtor do controller para esse tipo de uso:

```
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{

    public function initialize()
    {
        // Sempre habilite o CSRF component.
        $this->loadComponent('Csrf');
    }

}
```

Em adição ao método `initialize()`, a antiga propriedade `$components` também vai permitir você declarar quais components devem ser carregados. Enquanto heranças objeto-orientadas normais são enquadradas, os components e helpers usados por um controller são especialmente tratados. Nestes casos, os valores de propriedade do `AppController` são mesclados com arrays de classes controller filhas. Os valores na classe filha irão sempre sobre-escrever aqueles na `AppController`.

Fluxo de requisições

Quando uma requisição é feita para uma aplicação CakePHP, a classe `Cake\Routing\Router` e a classe `Cake\Routing\Dispatcher` usam *Connecting Routes* para encontrar e criar a instância correta do controller. Os dados da requisição são encapsulados em um objeto de requisição. O CakePHP coloca todas as informações importantes de uma requisição na propriedade `$this->request`. Veja a seção *Request* para mais informações sobre o objeto de requisição do CakePHP.

Métodos (actions) de controllers

Actions de controllers são responsáveis por converter os parâmetros de requisição em uma resposta para o navegador/usuário que fez a requisição. O CakePHP usa convenções para automatizar este processo e remove alguns códigos clichês que você teria que escrever de qualquer forma.

Por convenção, o CakePHP renderiza uma view com uma versão flexionada do nome da action. Retornando ao nosso exemplo da padaria online, nosso `RecipesController` poderia abrigar as actions `view()`, `share()` e `search()`. O controller seria encontrado em **src/Controller/RecipesController.php** contendo:

```
// src/Controller/RecipesController.php

class RecipesController extends AppController
{
    function view($id)
    {
        // A lógica da action vai aqui.
    }

    function share($customerId, $recipeId)
    {
        // A lógica da action vai aqui.
    }

    function search($query)
    {
        // A lógica da action vai aqui.
    }
}
```

Os arquivos de template para estas actions seriam **src/Template/Recipes/view.ctp**, **src/Template/Recipes/share.ctp** e **src/Template/Recipes/search.ctp**. A nomenclatura convencional para arquivos view é a versão lowercased (minúscula) e underscored (sem sublinhado) do nome da action.

Actions dos controllers geralmente usam `Controller::set()` para criar um contexto que a `View` usa para renderizar a camada view. Devido às convenções que o CakePHP usa, você não precisa criar e renderizar as views manualmente. Ao invés, uma vez que uma action de controller é completada, o CakePHP irá manipular a renderização e devolver a view.

Se por alguma razão você quiser pular o comportamento padrão, você pode retornar um objeto `Cake\Network\Response` a partir da action com a resposta definida.

Quando você usa métodos do controller com `Routing\RequestActionTrait::requestAction()` você irá tipicamente retornar uma instância `Response`. Se você tem métodos de controller que são usados para requisições web normais + requestAction, você deve checar o tipo de requisição antes de retornar:

```
// src/Controller/RecipesController.php

class RecipesController extends AppController
{
    public function popular()
    {
```

```
        $popular = $this->Recipes->find('popular');
        if ($this->request->is('requested')) {
            $this->response->body(json_encode($popular));
            return $this->response;
        }
        $this->set('popular', $popular);
    }
}
```

A action do controller acima é um exemplo de como um método pode ser usado com `Routing\RequestActionTrait::requestAction()` e requisições normais.

Para que você possa utilizar um controller de forma eficiente em sua própria aplicação, nós iremos cobrir alguns dos atributos e métodos oferecidos pelo controller do core do CakePHP.

Interagindo com views

Os controllers interagem com as views de diversas maneiras. Primeiro eles são capazes de passar dados para as views usando `Controller::set()`. Você também pode decidir no seu controller qual arquivo view deve ser renderizado através do controller.

Definindo variáveis para a view

`Cake\Controller\Controller::`**`set`**(*string $var*, *mixed $value*)

O método `Controller::set()` é a principal maneira de enviar dados do seu controller para a sua view. Após ter usado o método `Controller::set()`, a variável pode ser acessada em sua view:

```
// Primeiro você passa os dados do controller:

$this->set('color', 'pink');

// Então, na view, você pode utilizar os dados:
?>

Você selecionou a cobertura <?php echo $color; ?> para o bolo.
```

O método `Controller::set()` também aceita um array associativo como primeiro parâmetro. Isto pode oferecer uma forma rápida para atribuir uma série de informações para a view:

```
$data = [
    'color' => 'pink',
    'type' => 'sugar',
    'base_price' => 23.95
];

// Faça $color, $type, e $base_price
// disponíveis na view:

$this->set($data);
```

Renderizando uma view

`Cake\Controller\Controller::render`(*string $view*, *string $layout*)

O método `Controller::render()` é chamado automaticamente no fim de cada ação requisitada de um controller. Este método executa toda a lógica da view (usando os dados que você passou usando o método `Controller::set()`), coloca a view em `View::$layout`, e serve de volta para o usuário final.

O arquivo view usado pelo método `Controller::render()` é determinado por convenção. Se a action `search()` do controller `RecipesController` é requisitada, o arquivo view encontrado em **src/Template/Recipes/search.ctp** será renderizado:

```
namespace App\Controller;

class RecipesController extends AppController
{
// ...
    public function search()
    {
        // Render the view in src/Template/Recipes/search.ctp
        $this->render();
    }
// ...
}
```

Embora o CakePHP irá chamar o método `Controller::render()` automaticamente (ao menos que você altere o atributo `$this->autoRender` para `false`) após cada action, você pode usá-lo para especificar um arquivo view alternativo especificando o nome do arquivo view como primeiro parâmetro do método `Controller::render()`.

Se o parâmetro `$view` começar com '/', é assumido ser um arquivo view ou elemento relativo ao diretório `/src/Template`. Isto permite a renderização direta de elementos, muito útil em chamadas AJAX:

```
// Renderiza o elemento em src/Template/Element/ajaxreturn.ctp
$this->render('/Element/ajaxreturn');
```

O segundo parâmetro `$layout` do `Controller::render()` permite que você especifique o layout pelo qual a view é renderizada.

Renderizando uma view específica

Em seu controller você pode querer renderizar uma view diferente do que a convencional. Você pode fazer isso chamando o método `Controller::render()` diretamente. Uma vez chamado o método `Controller::render()`, o CakePHP não tentará renderizar novamente a view:

```
namespace App\Controller;

class PostsController extends AppController
{
    public function my_action()
    {
        $this->render('custom_file');
```

```
    }
}
```

Isto renderizaria o arquivo **src/Template/Posts/custom_file.ctp** ao invés de **src/Template/Posts/my_action.ctp**

Você também pode renderizar views de plugins utilizando a seguinte sintaxe: `$this->render('PluginName.PluginController/custom_file')`. Por exemplo:

```
namespace App\Controller;

class PostsController extends AppController
{
    public function my_action()
    {
        $this->render('Users.UserDetails/custom_file');
    }
}
```

Isto renderizaria **plugins/Users/src/Template/UserDetails/custom_file.ctp**

Redirecionando para outras páginas

`Cake\Controller\Controller::redirect` (*string|array $url*, *integer $status*)

O método de controle de fluxo que você vai usar na majoritariamente é `Controller::redirect()`. Este método recebe seu primeiro parâmetro na forma de uma URL relativa do CakePHP. Quando um usuário executar um pedido com êxito, você pode querer redirecioná-lo para uma tela de recepção.

```
public function place_order()
{
    // Logic for finalizing order goes here
    if ($success) {
        return $this->redirect(
            ['controller' => 'Orders', 'action' => 'thanks']
        );
    }
    return $this->redirect(
        ['controller' => 'Orders', 'action' => 'confirm']
    );
}
```

Este método irá retornar a instância da resposta com cabeçalhos apropriados definidos. Você deve retornar a instância da resposta da sua action para prevenir renderização de view e deixar o dispatcher controlar o redirecionamento corrente.

Você também pode usar uma URL relativa ou absoluta como o parâmetro $url:

```
return $this->redirect('/orders/thanks');
return $this->redirect('http://www.example.com');
```

Você também pode passar dados para a action:

```
return $this->redirect(['action' => 'edit', $id]);
```

O segundo parâmetro passado no `Controller::redirect()` permite a você definir um código de status HTTP para acompanhar o redirecionamento. Você pode querer usar o código 301 (movido permanentemente) ou 303 (veja outro), dependendo da natureza do redirecionamento.

Se você precisa redirecionar o usuário de volta para a página que fez a requisição, você pode usar:

```
$this->redirect($this->referer());
```

Um exemplo usando seqüências de consulta e hash pareceria com:

```
return $this->redirect([
    'controller' => 'Orders',
    'action' => 'confirm',
    '?' => [
        'product' => 'pizza',
        'quantity' => 5
    ],
    '#' => 'top'
]);
```

A URL gerada seria:

```
http://www.example.com/orders/confirm?product=pizza&quantity=5#top
```

Redirecionando para outra action no mesmo Controller

`Cake\Controller\Controller::`**`setAction`**(*$action*, *$args...*)

Se você precisar redirecionar a atual action para uma diferente no *mesmo* controller, você pode usar `Controller::setAction()` para atualizar o objeto da requisição, modificar o template da view que será renderizado e redirecionar a execução para a action especificada:

```
// De uma action delete, você pode renderizar uma página
// de índice atualizada.
$this->setAction('index');
```

Carregando models adicionais

`Cake\Controller\Controller::`**`loadModel`**(*string $modelClass*, *string $type*)

O método `loadModel` vem a calhar quando você precisa usar um model que não é padrão do controller ou o seu model não está associado com este.:

```
// Em um método do controller.
$this->loadModel('Articles');
$recentArticles = $this->Articles->find('all', [
    'limit' => 5,
    'order' => 'Articles.created DESC'
]);
```

Se você está usando um provedor de tabelas que não os da ORM nativa você pode ligar este sistema de tabelas aos controllers do CakePHP conectando seus métodos de factory:

```
// Em um método do controller.
$this->modelFactory(
    'ElasticIndex',
    ['ElasticIndexes', 'factory']
);
```

Depois de registrar uma tabela factory, você pode usar o `loadModel` para carregar instâncias:

```
// Em um método do controller
$this->loadModel('Locations', 'ElasticIndex');
```

Nota: O TableRegistry da ORM nativa é conectado por padrão como o provedor de 'Tabelas'.

Paginando um model

`Cake\Controller\Controller::paginate()`

Este método é usado para fazer a paginação dos resultados retornados por seus models. Você pode especificar o tamanho da página (quantos resultados serão retornados), as condições de busca e outros parâmetros. Veja a seção *pagination* para mais detalhes sobre como usar o método `paginate()`

O atributo paginate lhe oferece uma forma fácil de customizar como `paginate()` se comporta:

```
class ArticlesController extends AppController
{
    public $paginate = [
        'Articles' => [
            'conditions' => ['published' => 1]
        ]
    ];
}
```

Configurando components para carregar

`Cake\Controller\Controller::loadComponent` (*$name*, *$config* = [ ])

Em seu método `initialize()` do controller você pode definir qualquer component que quiser carregado, e qualquer configuração de dados para eles:

```
public function intialize()
{
    parent::initialize();
    $this->loadComponent('Csrf');
    $this->loadComponent('Comments', Configure::read('Comments'));
}
```

property Cake\Controller\Controller::$**components**

A propriedade `$components` em seus controllers permitem a você configurar components. Components configurados e suas dependências serão criados pelo CakePHP para você. Leia a seção *Configuring Components* para mais informações. Como mencionado anteriormente, a propriedade `$components` será mesclada com a propriedade definida em cada classe parente do seu controller.

Configurando helpers para carregar

property Cake\Controller\Controller::$**helpers**

Vamos observar como dizer ao controller do CakePHP que você planeja usar classes MVC adicionais:

```
class RecipesController extends AppController
{
    public $helpers = ['Form'];
}
```

Cada uma dessas variáveis são mescladas com seus valores herdados, portanto não é necessário (por exemplo) redeclarar o `FormHelper`, ou qualquer coisa declarada em seu `AppController`.

Ciclo de vida de callbacks em uma requisição

Os controllers do CakePHP vêm equipados com callbacks que você pode usar para inserir lógicas em torno do ciclo de vida de uma requisição:

Cake\Controller\Controller::**beforeFilter**(*Event $event*)
: Este método é executado antes de cada ação dos controllers. É um ótimo lugar para verificar se há uma sessão ativa ou inspecionar as permissões de um usuário.

  Nota: O método beforeFilter() será chamado para ações perdidas.

Cake\Controller\Controller::**beforeRender**(*Event $event*)
: Chamada após a lógica da action de um controller, mas antes da view ser renderizada. Esse callback não é usado frequentemente, mas pode ser necessário se você estiver chamando `Controller\Controller::render()` manualmente antes do final de uma determinada action.

Cake\Controller\Controller::**afterFilter**()
: Chamada após cada ação dos controllers, e após a completa renderização da view. Este é o último método executado do controller.

Em adição ao ciclo de vida dos callbacks do controller, *Components (Componentes)* também oferece um conjunto de callbacks similares.

Lembre de chamar os callbacks do `AppController` em conjunto com os callbacks dos controllers para melhores resultados:

```
public function beforeFilter(Event $event)
{
    parent::beforeFilter($event);
}
```

Mais sobre controllers

O Pages Controller

CakePHP é distribuído com o controller **PagesController.php**. Esse controller é simples, seu uso é opcional e normalmente direcionado a prover páginas estáticas. A homepage que você vê logo depois de instalar o CakePHP utiliza esse controller e o arquivo da view fica em **src/Template/Pages/home.ctp**. Se você criar o arquivo **src/Template/Pages/about.ctp**, você poderá acessá-lo em **http://example.com/pages/about**. Fique a vontade para alterar esse controller para atender suas necessacidades ou mesmo excluí-lo.

Quando você cria sua aplicação pelo Composer, o `PagesController` vai ser criado na pasta **src/Controller/**.

Components (Componentes)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Configuring Components

Authentication

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

[1] https://github.com/cakephp/docs
[2] https://github.com/cakephp/docs

CookieComponent

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[3] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Cross Site Request Forgery

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[4] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Flash

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[5] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Security

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[6] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[3] https://github.com/cakephp/docs
[4] https://github.com/cakephp/docs
[5] https://github.com/cakephp/docs
[6] https://github.com/cakephp/docs

Pagination

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[7] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Request Handling

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[8] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[7] https://github.com/cakephp/docs
[8] https://github.com/cakephp/docs

CAPÍTULO 11

Views (Visualização)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Using View Blocks

Layouts

Elements

More About Views

View Cells (Células de Visualização)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs
[2] https://github.com/cakephp/docs

Temas

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[3] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Views JSON & XML

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[4] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Helpers (Facilitadores)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[5] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Flash

class `Cake\View\Helper\`**`FlashHelper`**(*View $view*, *array $config* =[])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[6] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[3] https://github.com/cakephp/docs
[4] https://github.com/cakephp/docs
[5] https://github.com/cakephp/docs
[6] https://github.com/cakephp/docs

Form

class Cake\View\Helper**FormHelper** (*View $view*, *array $config* =[ ])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[7] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Html

class Cake\View\Helper**HtmlHelper** (*View $view*, *array $config* =[ ])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[8] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Number

class Cake\View\Helper**NumberHelper** (*View $view*, *array $config* =[ ])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[9] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Paginator

class Cake\View\Helper**PaginatorHelper** (*View $view*, *array $config* =[ ])

[7] https://github.com/cakephp/docs
[8] https://github.com/cakephp/docs
[9] https://github.com/cakephp/docs

PaginatorHelper Templates

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[10] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

RSS

class `Cake\View\Helper\`**`RssHelper`**(*View $view*, *array $config* =[ ])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[11] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Session

class `Cake\View\Helper\`**`SessionHelper`**(*View $view*, *array $config* =[ ])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[12] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Text

class `Cake\View\Helper\`**`TextHelper`**(*View $view*, *array $config* =[ ])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[13] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

[10] https://github.com/cakephp/docs
[11] https://github.com/cakephp/docs
[12] https://github.com/cakephp/docs
[13] https://github.com/cakephp/docs

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Time

class `Cake\View\Helper\`**`TimeHelper`**(*View $view*, *array $config* =[])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[14] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Url

class `Cake\View\Helper\`**`UrlHelper`**(*View $view*, *array $config* =[])

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[15] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

[14] https://github.com/cakephp/docs
[15] https://github.com/cakephp/docs

CAPÍTULO 12

Models (Modelos)

Models (Modelos) são as classes que servem como camada de negócio na sua aplicação. Isso significa que eles devem ser responsáveis pela gestão de quase tudo o que acontece em relação a seus dados, sua validade, interações e evolução do fluxo de trabalho de informação no domínio do trabalho.

No CakePHP seu modelo de domínio da aplicação é dividido em 2 tipos de objetos principais. Os primeiros são **repositories (repositórios)** ou **table objects (objetos de tabela)**. Estes objetos fornecem acesso a coleções de dados. Eles permitem a você salvar novos registros, modificar/deletar os que já existem, definir relacionamentos, e executar operações em massa. O segundo tipo de objetos são as **entities (entidades)**. Entities representam registros individuais e permitem a você definir comportamento em nível de linha/registro e funcionalidades.

O ORM (MOR - Mapeamento Objeto-Relacional) nativo do CakePHP especializa-se em banco de dados relacionais, mas pode ser extendido para suportar fontes de dados alternativas.

O ORM do Cakephp toma emprestadas ideias e conceitos dos padrões ActiveRecord e Datamapper. Isso permite criar uma implementação híbrida que combina aspectos de ambos padrões para criar uma ORM rápida e simples de utilizar.

Antes de nós começarmos explorando o ORM, tenha certeza que você *configure your database connections*.

Nota: Se você é familiarizado com versões anteriores do CakePHP, você deveria ler o *Guia de atualização para o novo ORM* para esclarecer diferenças importantes entre o CakePHP 3.0 e suas versões antigas.

Exemplo rápido

Para começar você não precisa escrever código. Se você seguiu as convenções do CakePHP para suas tabelas de banco de dados, você pode simplesmente começar a usar o ORM. Por exemplo, se quiséssemos carregar alguns dados da nossa tabela `articles` poderíamos fazer:

```
use Cake\ORM\TableRegistry;
$articles = TableRegistry::get('Articles');
$query = $articles->find();
```

```
foreach ($query as $row) {
    echo $row->title;
}
```

Nota-se que nós não temos que criar qualquer código ou definir qualquer configuração. As convenções do CakePHP nos permitem pular alguns códigos clichê, e permitir que o framework insera classes básicas enquanto sua aplicação não criou uma classe concreta. Se quiséssemos customizar nossa classe ArticlesTable adicionando algumas associações ou definir alguns métodos adicionais, deveriamos acrescentar o seguinte a **src/Model/Table/ArticlesTable.php** após a tag de abertura `<?php`:

```
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{

}
```

Classes de tabela usam a versão CamelCased do nome da tabela com o sufixo `Table` como o nome da classe. Uma vez que sua classe fora criada, você recebe uma referência para esta utilizando o `ORM\TableRegistry` como antes:

```
use Cake\ORM\TableRegistry;
// Agora $articles é uma instância de nossa classe ArticlesTable.
$articles = TableRegistry::get('Articles');
```

Agora que temos uma classe de tabela concreta, nós provavelmente vamos querer usar uma classe de entidade concreta. As classes de entidade permitem definir métodos de acesso, métodos mutantes, definir lógica personalizada para os registros individuais e muito mais. Vamos começar adicionando o seguinte para **src/Model/Entity/Article.php** após a tag de abertura `<?php`:

```
namespace App\Model\Entity;

use Cake\ORM\Entity;

class Article extends Entity
{

}
```

Entidades usam a versão singular CamelCase do nome da tabela como seu nome de classe por padrão. Agora que nós criamos nossa classe de entidade, quando carregarmos entidades do nosso banco de dados, nós iremos receber instâncias da nossa nova classe Article:

```
use Cake\ORM\TableRegistry;

// Agora uma instância de ArticlesTable.
$articles = TableRegistry::get('Articles');
$query = $articles->find();

foreach ($query as $row) {
    // Cada linha é agora uma instância de nossa classe Article.
```

```
    echo $row->title;
}
```

CakePHP utiliza convenções de nomenclatura para ligar as classes de tabela e entidade juntas. Se você precisar customizar qual entidade uma tabela utiliza, você pode usar o método `entityClass()` para definir nomes de classe específicos.

Veja os capítulos em *Objetos de tabela* e *Entities (Entidades)* para mais informações sobre como usar objetos de tabela e entidades em sua aplicação.

Mais informação

O básico sobre banco de dados

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Construtor de queries

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Objetos de tabela

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[3] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

[1] https://github.com/cakephp/docs
[2] https://github.com/cakephp/docs
[3] https://github.com/cakephp/docs

Entities (Entidades)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[4] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Requisitando dados e conjuntos de resultados

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[5] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Validando dados

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[6] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

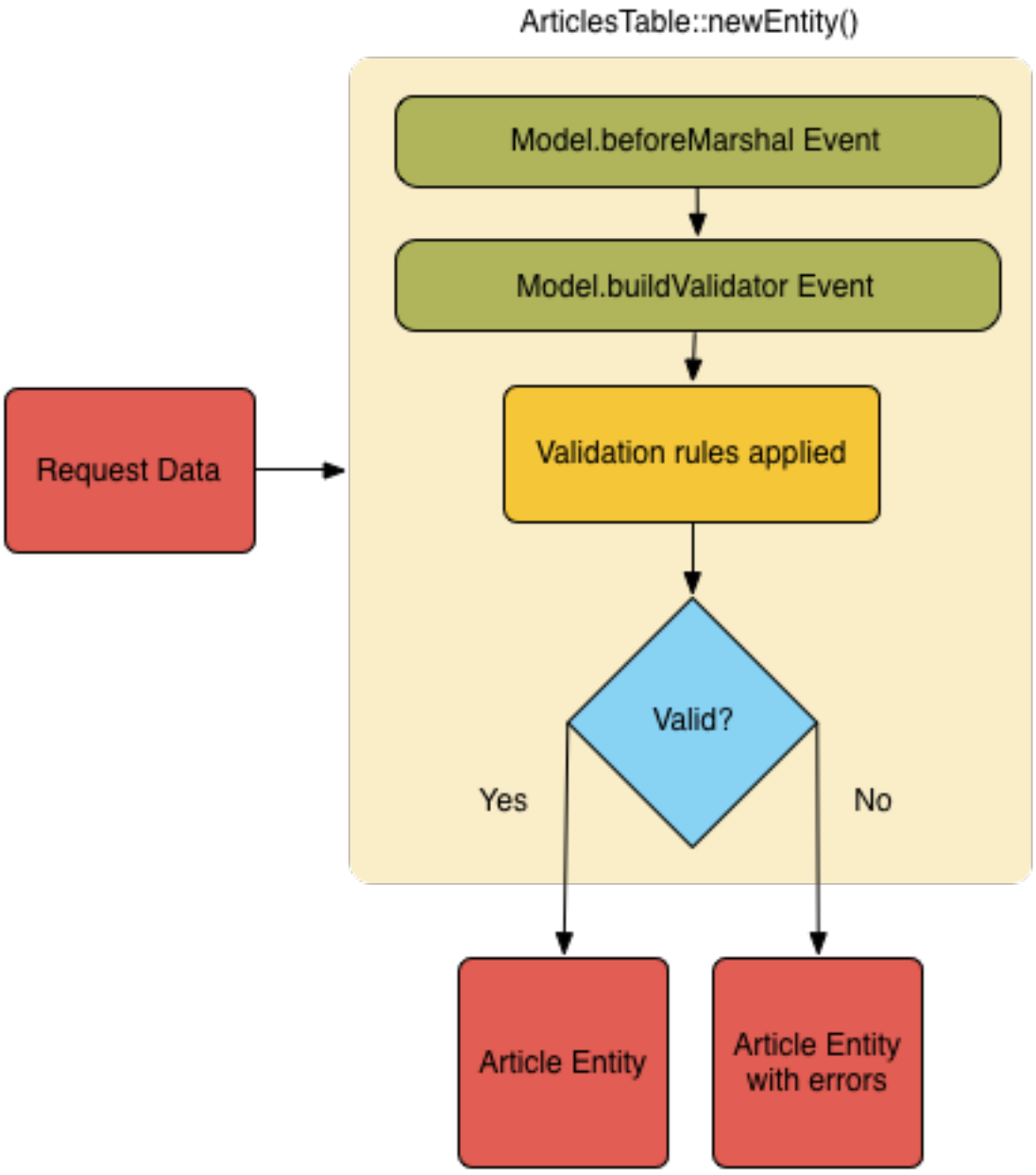
Salvando dados

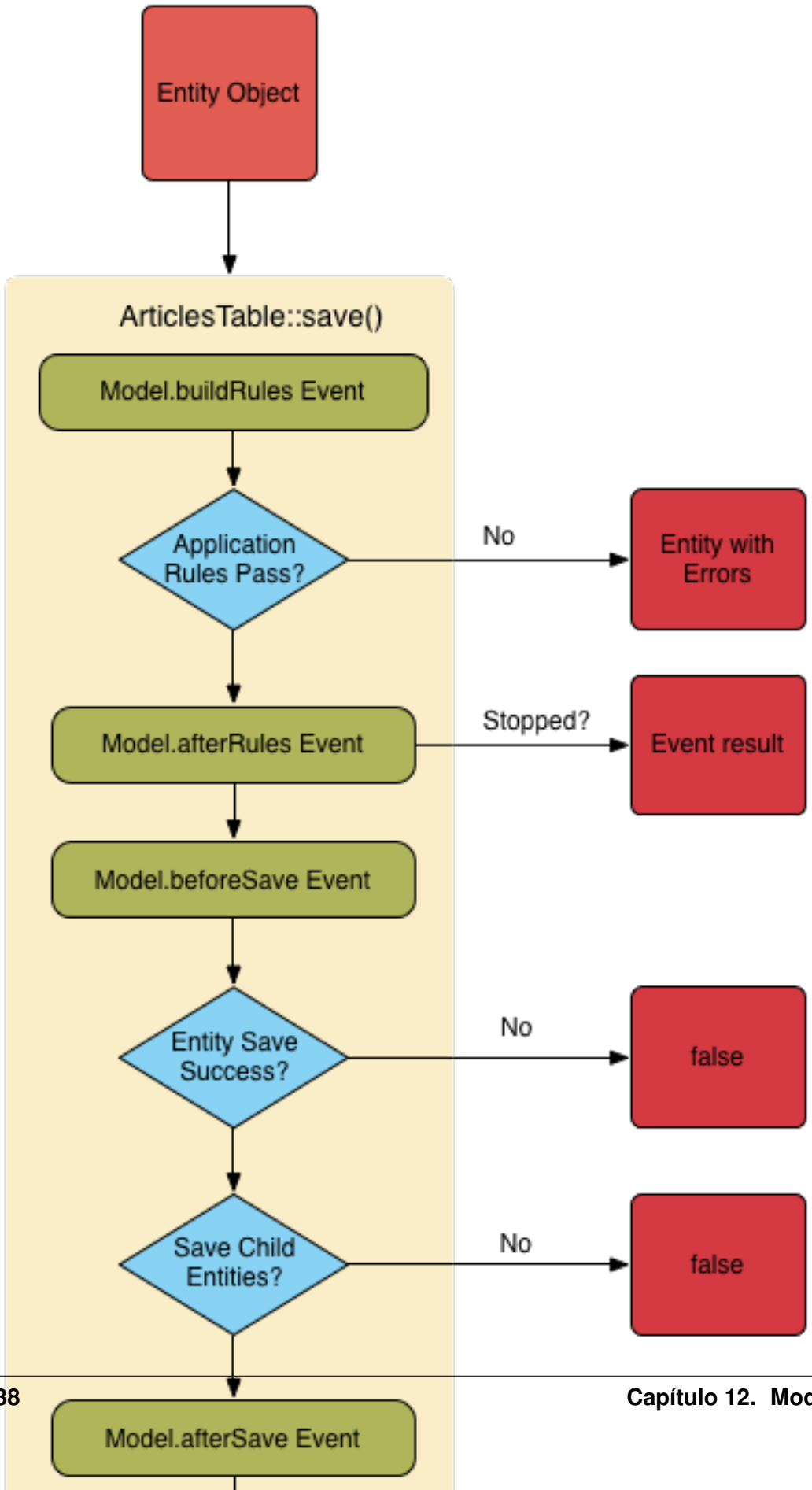
Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[7] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[4] https://github.com/cakephp/docs
[5] https://github.com/cakephp/docs
[6] https://github.com/cakephp/docs
[7] https://github.com/cakephp/docs


ArticlesTable::newEntity()
Model.beforeMarshal Event
Model.buildValidator Event
Validation rules applied
Request Data
Valid?
Yes
No
Article Entity
Article Entity with errors


Entity Object
ArticlesTable::save()
Model.buildRules Event
Application Rules Pass?
No
Entity with Errors
Model.afterRules Event
Stopped?
Event result
Model.beforeSave Event
Entity Save Success?
No
false
Save Child Entities?
No
false
Model.afterSave Event

Deletando dados

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[8] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Associações - Conectando tabelas

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[9] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Behaviors (Comportamentos)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[10] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

CounterCache

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[11] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[8] https://github.com/cakephp/docs
[9] https://github.com/cakephp/docs
[10] https://github.com/cakephp/docs
[11] https://github.com/cakephp/docs

Timestamp

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[12] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Translate

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[13] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Tree

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[14] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Schema

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[15] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[12] https://github.com/cakephp/docs
[13] https://github.com/cakephp/docs
[14] https://github.com/cakephp/docs
[15] https://github.com/cakephp/docs

ORM Cache Shell

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[16] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[16] https://github.com/cakephp/docs

CAPÍTULO 13

Authentication

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 14

Bake Console

O bake console do CakePHP é outro empenho para você ter o CakePHP configurado e funcionando rápido. O bake console pode criar qualquer ingrediente básico do CakePHP: models, behaviors, views, helpers, components, test cases, fixtures e plugins. E nós não estamos apenas falando de classes esqueleto: O Bake pode criar uma aplicação totalmente funcional em questão de minutos. De fato, o Bake é um passo natural a se dar uma vez que a aplicação tem seu alicerce construído.

Instalação

Antes de tentar usar ou extender o bake, tenha certeza que ele está instalado em sua aplicação. O bake é distribuído como um plugin que você pode instalar com o Composer:

```
composer require --dev cakephp/bake:~1.0
```

Isto irá instalar o bake como uma dependência de desenvolvimento, sendo assim, não instalado quando em um ambiente de produção. As seções a seguir cobrem o uso do bake com mais detalhes:

Geração de código com o Bake

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

[1] https://github.com/cakephp/docs

Extendendo o Bake

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[2] https://github.com/cakephp/docs

CAPÍTULO 15

Caching

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 16

Console e Shells

O CakePHP não oferece um framework apenas para desenvolvimento web, mas também um framework para criação de aplicações de console. Estas aplicações são ideais para manipular variadas tarefas em segundo plano como manutenção e complementação de trabalho fora do ciclo requisição-resposta. As aplicações de console do CakePHP permitem a vocë reutilizar suas classes de aplicação a partir da linha de comando.

O CakePHP traz consigo algumas aplicações de console nativas. Algumas dessas aplicações são utilizadas em conjunto com outros recursos do CakePHP (como i18n), e outros de uso geral para aceleração de trabalho.

O Console do CakePHP

Esta seção provê uma introdução à linha de comando do CakePHP. Ferramentas de console são ideais para uso em cron jobs, ou utilitários baseados em linha de comando que não precisam ser acessíveis por um navegador web.

O PHP provê um cliente CLI que faz interface com o seu sistema de arquivos e aplicações de forma muito mais suave. O console do CakePHP provê um framework para criar scripts shell. O console utiliza uma configuração tipo dispatcher para carregar uma shell ou tarefa, e prover seus parâmetros.

Nota: Uma linha de comando (CLI) constutuída a partir do PHP deve estar disponível no sistema se você planeja utilizr o Console.

Antes de entrar em detalhes, vamos ter certeza de que você pode executar o console do CakePHP. Primeiro, você vai precisar executar um sistema shell. Os exemplos apresentados nesta seção serão em bash, mas o Console do CakePHP é compatível com o Windows também. Este exemplo assume que o usuário está conectado em um prompt do bash e está atualmente na raiz de uma aplicação CakePHP.

Aplicações CakePHP possuem um diretório *Console‘‘* que contém todas as shells e tarefas para uma aplicação. Ele também vem com um executável:

```
$ cd /path/to/app
$ bin/cake
```

Executar o Console sem argumentos produz esta mensagem de ajuda:

```
Welcome to CakePHP v3.0.0 Console
---------------------------------------------------------------
App : App
Path: /Users/markstory/Sites/cakephp-app/src/
---------------------------------------------------------------
Current Paths:

 -app: src
 -root: /Users/markstory/Sites/cakephp-app
 -core: /Users/markstory/Sites/cakephp-app/vendor/cakephp/cakephp

Changing Paths:

Your working path should be the same as your application path. To change your path use the
Example: -app relative/path/to/myapp or -app /absolute/path/to/myapp

Available Shells:

[Bake] bake

[Migrations] migrations

[CORE] i18n, orm_cache, plugin, server

[app] behavior_time, console, orm

To run an app or core command, type cake shell_name [args]
To run a plugin command, type cake Plugin.shell_name [args]
To get help on a specific command, type cake shell_name --help
```

A primeira informação impressa refere-se a caminhos. Isso é útil se você estiver executando o console a partir de diferentes partes do sistema de arquivos.

Criando uma Shell

Vamos criar uma shell para utilizar no Console. Para este exemplo, criaremos uma simples Hello World (Olá Mundo) shell. No diretório **src/Shell** de sua aplicação crie **HelloShell.php**. Coloque o seguinte código dentro do arquivo recem criado:

```
namespace App\Shell;

use Cake\Console\Shell;

class HelloShell extends Shell
{
    public function main()
    {
        $this->out('Hello world.');
```

```
    }
}
```

As convenções para as classes de shell são de que o nome da classe deve corresponder ao nome do arquivo, com o sufixo de Shell. No nosso shell criamos um método `main()`. Este método é chamado quando um shell é chamado sem comandos adicionais. Vamos adicionar alguns comandos daqui a pouco, mas por agora vamos executar a nossa shell. A partir do diretório da aplicação, execute:

```
bin/cake hello
```

Você deve ver a seguinte saída:

```
Welcome to CakePHP Console
---------------------------------------------------------------
App : app
Path: /Users/markstory/Sites/cake_dev/src/
---------------------------------------------------------------
Hello world.
```

Como mencionado antes, o método `main()` em shells é um método especial chamado sempre que não há outros comandos ou argumentos dados para uma shell. Por nosso método principal não ser muito interessante, vamos adicionar outro comando que faz algo:

```
namespace App\Shell;

use Cake\Console\Shell;

class HelloShell extends Shell
{
    public function main()
    {
        $this->out('Hello world.');
    }

    public function heyThere($name = 'Anonymous')
    {
        $this->out('Hey there ' . $name);
    }
}
```

Depois de salvar o arquivo, você deve ser capaz de executar o seguinte comando e ver o seu nome impresso:

```
bin/cake hello hey_there your-name
```

Qualquer método público não prefixado por um _ é permitido para ser chamado a partir da linha de comando. Como você pode ver, os métodos invocados a partir da linha de comando são transformados do argumento prefixado para a forma correta do nome camel-cased (camelizada) na classe.

No nosso método `heyThere()` podemos ver que os argumentos posicionais são providos para nossa função `heyThere()`. Argumentos posicionais também estão disponíveis na propriedade `args`. Você pode acessar switches ou opções em aplicações shell, estando disponíveis em `$this->params`, mas nós iremos cobrir isso daqui a pouco.

Quando utilizando um método `main()` você não estará liberado para utilizar argumentos posicionais. Isso

se deve ao primeiro argumento posicional ou opção ser interpretado(a) como o nome do comando. Se você quer utilizar argumentos, você deve usar métodos diferentes de `main()`.

Usando Models em suas shells

Você frequentemente precisará acessar a camada lógica de negócios em seus utilitários shell; O CakePHP faz essa tarefa super fácil. Você pode carregar models em shells assim como faz em um controller utilizando `loadModel()`. Os models carregados são definidos como propriedades anexas à sua shell:

```
namespace App\Shell;

use Cake\Console\Shell;

class UserShell extends Shell
{

    public function initialize()
    {
        parent::initialize();
        $this->loadModel('Users');
    }

    public function show()
    {
        if (empty($this->args[0])) {
            return $this->error('Por favor, indique um nome de usuário.');
        }
        $user = $this->Users->findByUsername($this->args[0])->first();
        $this->out(print_r($user, true));
    }
}
```

A shell acima, irá preencher um user pelo seu username e exibir a informação armazenada no banco de dados.

Tasks de Shell

Haverão momentos construindo aplicações mais avançadas de console que você vai querer compor funcionalidades em classes reutilizáveis que podem ser compartilhadas através de muitas shells. Tasks permitem que você extraia comandos em classes. Por exemplo, o `bake` é feito quase que completamente de tasks. Você define tasks para uma shell usando a propriedade `$tasks`:

```
class UserShell extends Shell
{
    public $tasks = ['Template'];
}
```

Você pode utilizar tasks de plugins utilizando o padrão *plugin syntax*. Tasks são armazenadas sob `Shell/Task/` em arquivos nomeados depois de suas classes. Então se nós estivéssemos criando uma nova task 'FileGenerator', você deveria criar **src/Shell/Task/FileGeneratorTask.php**.

Cada task deve ao menos implementar um método `main()`. O ShellDispatcher, vai chamar esse método quando a task é invocada. Uma classe task se parece com:

```
namespace App\Shell\Task;

use Cake\Console\Shell;

class FileGeneratorTask extends Shell
{
    public function main()
    {
    }
}
```

Uma shell também pode prover acesso a suas tasks como propriedades, que fazem tasks serem ótimas para criar punhados de funcionalidade reutilizáveis similares a *Components (Componentes)*:

```
// Localizado em src/Shell/SeaShell.php
class SeaShell extends Shell
{
    // Localizado em src/Shell/Task/SoundTask.php
    public $tasks = ['Sound'];

    public function main()
    {
        $this->Sound->main();
    }
}
```

Você também pode acessar tasks diretamente da linha de comando:

```
$ cake sea sound
```

Nota: Para acessar tasks diretamente através da linha de comando, a task **deve** ser incluída na propriedade da classe shell `$tasks`. Portanto, esteja ciente que um método chamado "sound" na classe SeaShell deve sobrescrever a habilidade de acessar a funcionalidade na task Sound, especificada no array $tasks.

Carregando Tasks em tempo-real com TaskRegistry

Você pode carregar arquivos em tempo-real utilizando o Task registry object. Você pode carregar tasks que não foram declaradas no $tasks dessa forma:

```
$project = $this->Tasks->load('Project');
```

Carregará e retornará uma instância ProjectTask. Você pode carregar tasks de plugins usando:

```
$progressBar = $this->Tasks->load('ProgressBar.ProgressBar');
```

Invocando outras Shells a partir da sua Shell

Cake\Console**dispatchShell**(*$args*)

Existem ainda muitos casos onde você vai querer invocar uma shell a partir de outra. `Shell::dispatchShell()` lhe dá a habilidade de chamar outras shells ao providenciar o `argv` para a sub shell. Você pode providenciar argumentos e opções tanto como variáveis ou como strings:

```
// Como uma string
$this->dispatchShell('schema create Blog --plugin Blog');

// Como um array
$this->dispatchShell('schema', 'create', 'Blog', '--plugin', 'Blog');
```

O conteúdo acima mostra como você pode chamar a shell schema para criar o schema de um plugin de dentro da shell do próprio.

Recenendo Input de usuários

Cake\Console**in**(*$question*, *$choices* = *null*, *$defaut* = *null*)

Quando construir aplicações interativas pelo console você irá precisar receber inputs dos usuários. CakePHP oferece uma forma fácil de fazer isso:

```
// Receber qualquer texto dos usuários.
$color = $this->in('What color do you like?');

// Receber uma escolha dos usuários.
$selection = $this->in('Red or Green?', ['R', 'G'], 'R');
```

A validação de seleção é insensitiva a maiúsculas / minúsculas.

Criando Arquivos

Cake\Console**createFile**(*$path*, *$contents*)

Muitas aplicações Shell auxiliam tarefas de desenvolvimento e implementação. Criar arquivos é frequentemente importante nestes casos de uso. O CakePHP oferece uma forma fácil de criar um arquivo em um determinado diretório:

```
$this->createFile('bower.json', $stuff);
```

Se a Shell for interativa, um alerta vai ser gerado, e o usuário questionado se ele quer sobreescrever o arquivo caso já exista. Se a propriedade de interação da shell for `false`, nenhuma questão será disparada e o arquivo será simplesmente sobreescrito.

Saída de dados do Console

A classe `Shell` oferece alguns métodos para direcionar conteúdo:

```
// Escreve para stdout
$this->out('Normal message');

// Escreve para stderr
$this->err('Error message');

// Escreve para stderr e para o processo
$this->error('Fatal error');
```

A Shell também inclui métodos para limpar a saída de dados, criando linhas em branco, ou desenhando uma linha de traços:

```
// Exibe 2 linhas novas
$this->out($this->nl(2));

// Limpa a tela do usuário
$this->clear();

// Desenha uma linha horizontal
$this->hr();
```

Por último, você pode atualizar a linha atual de texto na tela usando `_io->overwrite()`:

```
$this->out('Counting down');
$this->out('10', 0);
for ($i = 9; $i > 0; $i--) {
    sleep(1);
    $this->_io->overwrite($i, 0, 2);
}
```

É importante lembrar, que você não pode sobreescrever texto uma vez que uma nova linha tenha sido exibida.

Console Output Levels

Shells frequentemente precisam de diferentes níveis de verbosidade. Quando executadas como cron jobs, muitas saídas são desnecessárias. E há ocasiões que você não estará interessado em tudo que uma shell tenha a dizer. Você pode usar os níveis de saída para sinalizar saídas apropriadamente. O usuário da shell, pode então decidir qual nível de detalhe ele está interessado ao sinalizar o chamado da shell. `Cake\Console\Shell::out()` suporta 3 tipos de saída por padrão.

- QUIET - Apenas informação absolutamente importante deve ser sinalizada.
- NORMAL - O nível padrão, e uso normal.
- VERBOSE - Sinalize mensagens que podem ser irritantes em demasia para uso diário, mas informativas para depuração como VERBOSE.

Você pode sinalizar a saíde da seguinte forma:

```
// Deve aparecer em todos os níveis.
$this->out('Quiet message', 1, Shell::QUIET);
$this->quiet('Quiet message');

// Não deve aparecer quando a saída quiet estiver alternado.
$this->out('normal message', 1, Shell::NORMAL);
$this->out('loud message', 1, Shell::VERBOSE);
$this->verbose('Verbose output');

// Deve aparecer somente quando a saíde verbose estiver habilitada.
$this->out('extra message', 1, Shell::VERBOSE);
$this->verbose('Verbose output');
```

Você pode controlar o nível de saída das shells, ao usar as opções `--quiet` e `--verbose`. Estas opções são adicionadas por padrão, e permitem a você controlar consistentemente níveis de saída dentro das suas shells do CakePHP.

Estilizando a saída de dados

Estilizar a saída de dados é feito ao incluir tags - como no HTML - em sua saída. O ConsoleOutput irá substituir estas tags com a seqüência correta de código ansi. Hão diversos estilos nativos, e você pode criar mais. Os nativos são:

- `error` Mensagens de erro. Texto sublinhado vermelho.
- `warning` Mensagens de alerta. Texto amarelo.
- `info` Mensagens informativas. Texto ciano.
- `comment` Texto adicional. Texto azul.
- `question` Texto que é uma questão, adicionado automaticamente pela shell.

Você pode criar estilos adicionais usando `$this->stdout->styles()`. Para declarar um novo estilo de saíde você pode fazer:

```
$this->_io->styles('flashy', ['text' => 'magenta', 'blink' => true]);
```

Isso deve então permití-lo usar uma `<flashy>` tag na saída de sua shell, e se as cores ansi estiverem habilitadas, o seguinte pode ser renderizado como texto magenta piscante `$this->out('<flashy>Whoooa</flashy> Something went wrong');`. Quando definir estilos você pode usar as seguintes cores para os atributos `text` e `background`:

- black
- red
- green
- yellow
- blue
- magenta

- cyan
- white

Você também pode usar as seguintes opções através de valores boleanos, defini-los com valor positivo os habilita.

- bold
- underline
- blink
- reverse

Adicionar um estilo o torna disponível para todas as instâncias do ConsoleOutput, então você não tem que redeclarar estilos para os objetos stdout e stderr respectivamente.

Desabilitando a colorização

Mesmo que a colorização seja incrível, haverão ocasiões que você quererá desabilitá-la, ou forçá-la:

```
$this->_io->outputAs(ConsoleOutput::RAW);
```

O citado irá colocar o objeto de saída em modo raw. Em modo raw, nenhum estilo é aplicado. Existem três modos que você pode usar.

- `ConsoleOutput::RAW` - Saída raw, nenhum estilo ou formatação serão aplicados. Este é um modo indicado se você estiver exibindo XML ou, quiser depurar porquê seu estilo não está funcionando.
- `ConsoleOutput::PLAIN` - Saída de texto simples, tags conhecidas de estilo serão removidas da saída.
- `ConsoleOutput::COLOR` - Saída onde a cor é removida.

Por padrão em sistemas *nix objetos ConsoleOutput padronizam-se a a saída de cores. Em sistemas Windows, a saída simples é padrão a não ser que a variável de ambiente `ANSICON` esteja presente.

Opções de configuração e Geração de ajuda

class `Cake\Console\`**`ConsoleOptionParser`**

`ConsoleOptionParser` oferece uma opção de CLI e analisador de argumentos.

OptionParsers permitem a você completar dois objetivos ao mesmo tempo. Primeiro, eles permitem definir opções e argumentos para os seus comandos. Isso permite separar validação básica de dados e seus comandos do console. Segundo, permite prover documentação, que é usada para gerar arquivos de ajuda bem formatados.

O console framework no CakePHP recebe as opções do seu interpetador shell ao chamar `$this->getOptionParser()`. Sobreescrever esse método permite configurar o OptionParser para definir as entradas aguardadas da sua shell. Você também pode configurar interpetadores de subcomandos,

que permitem ter diferentes interpretadores para subcomandos e tarefas. O ConsoleOptionParser implementa uma interface fluida e inclui métodos para facilmente definir múltiplas opções/argumentos de uma vez:

```
public function getOptionParser()
{
    $parser = parent::getOptionParser();
    // Configure parser
    return $parser;
}
```

Configurando um interpretador de opção com a interface fluida

Todos os métodos que configuram um interpretador de opções podem ser encadeados, permitindo definir um interpretador de opções completo em uma série de chamadas de métodos:

```
public function getOptionParser()
{
    $parser = parent::getOptionParser();
    $parser->addArgument('type', [
        'help' => 'Either a full path or type of class.'
    ])->addArgument('className', [
        'help' => 'A CakePHP core class name (e.g: Component, HtmlHelper).'
    ])->addOption('method', [
        'short' => 'm',
        'help' => __('The specific method you want help on.')
    ])->description(__('Lookup doc block comments for classes in CakePHP.'));
    return $parser;
}
```

Os métodos que permitem encadeamento são:

- description()
- epilog()
- command()
- addArgument()
- addArguments()
- addOption()
- addOptions()
- addSubcommand()
- addSubcommands()

`Cake\Console\ConsoleOptionParser::description`(*$text = null*)

Recebe ou define a descrição para o interpretador de opções. A descrição é exibida acima da informação do argumento e da opção. Ao instanciar tanto em array como em string, você pode definir o valor da descrição. Instanciar sem argumentos vai retornar o valor atual:

```
// Define múltiplas linhas de uma vez
$parser->description(['line one', 'line two']);

// Lê o valor atual
$parser->description();
```

Cake\Console\ConsoleOptionParser::**epilog**(*$text = null*)

Recebe ou define o epílogo para o interpretador de opções. O epílogo é exibido depois da informação do argumento e da opção. Ao instanciar tanto em array como em string, você pode definir o valor do epílogo. Instanciar sem argumentos vai retornar o valor atual:

```
// Define múltiplas linhas de uma vez
$parser->epilog(['line one', 'line two']);

// Lê o valor atual
$parser->epilog();
```

Adicionando argumentos

Cake\Console\ConsoleOptionParser::**addArgument**(*$name*, *$params* = [])

Argumentos posicionais são frequentemente usados em ferramentas de linha de comando, e `ConsoleOptionParser` permite definir argumentos bem como torná-los requiríveis. Você pode adicionar argumentos um por vez com `$parser->addArgument();` ou múltiplos de uma vez com `$parser->addArguments();`:

```
$parser->addArgument('model', ['help' => 'The model to bake']);
```

Você pode usar as seguintes opções ao criar um argumento:

- `help` O texto de ajuda a ser exibido para este argumento.
- `required` Se esse parâmetro é requisito.
- `index` O índice do argumento, se deixado indefinido, o argumento será colocado no final dos argumentos. Se você definir o mesmo índice duas vezes, a primeira opção será sobreescrita.
- `choices` Um array de opções válidas para esse argumento. Se deixado vazio, todos os valores são válidos. Uma exceção será lançada quando parse() encontrar um valor inválido.

Argumentos que forem definidos como requisito lançarão uma exceção quando interpretarem o comando se eles forem omitidos. Então você não tem que lidar com isso em sua shell.

Cake\Console\ConsoleOptionParser::**addArguments**(*array $args*)

Se você tem um array com múltiplos argumentos você pode usar `$parser->addArguments()` para adicioná-los de uma vez.:

```
$parser->addArguments([
    'node' => ['help' => 'The node to create', 'required' => true],
    'parent' => ['help' => 'The parent node', 'required' => true]
]);
```

Assim como todos os métodos de construção no ConsoleOptionParser, addArguments pode ser usado como parte de um fluido método encadeado.

Validando argumentos

Ao criar argumentos posicionais, você pode usar a marcação `required` para indicar que um argumento deve estar presente quando uma shell é chamada. Adicionalmente você pode usar o `choices` para forçar um argumento a ser de uma lista de escolhas válidas:

```
$parser->addArgument('type', [
    'help' => 'The type of node to interact with.',
    'required' => true,
    'choices' => ['aro', 'aco']
]);
```

O código acima irá criar um argumento que é requisitado e tem validação no input. Se o argumento está tanto indefinodo, ou possui um valor incorreto, uma exceção será lançada e a shell parará.

Adicionando opções

`Cake\Console\ConsoleOptionParser::addOption`(*$name*, *$options* =[ ])

Opções são frequentemente usadas em ferramentas CLI. `ConsoleOptionParser` suporta a criação de opções com verbose e aliases curtas, suprindo padrões e criando ativadores boleanos. Opções são criadas tanto com `$parser->addOption()` ou `$parser->addOptions()`.:

```
$parser->addOption('connection', [
    'short' => 'c',
    'help' => 'connection',
    'default' => 'default',
]);
```

O código citado permite a você usar tanto `cake myshell --connection=other`, `cake myshell --connection other`, ou `cake myshell -c other` quando invocando a shell. Você também criar ativadores boleanos. Estes ativadores não consumem valores, e suas presenças apenas os habilitam nos parâmetros interpretados.:

```
$parser->addOption('no-commit', ['boolean' => true]);
```

Com essa opção, ao chamar uma shell como `cake myshell --no-commit something` o parâmetro no-commit deve ter um valor de `true`, e *something* deve ser tratado como um argumento posicional. As opções nativas `--help`, `--verbose`, e `--quiet` usam essa funcionalidade.

Ao criar opções você pode usar os seguintes argumentos para definir o seu comportamento:

- `short` - A variação de letra única para essa opção, deixe indefinido para none.
- `help` - Texto de ajuda para essa opção. Usado ao gerar ajuda para a opção.
- `default` - O valor padrão para essa opção. Se não estiver definido o valor padrão será `true`.
- `boolean` - A opção não usa valor, é apenas um ativador boleano. Por padrão `false`.

- `choices` - Um array de escolhas válidas para essa opção. Se deixado vazio, todos os valores são considerados válidos. Uma exceção será lançada quando parse() encontrar um valor inválido.

`Cake\Console\ConsoleOptionParser::addOptions` (*array $options*)

Se você tem um array com múltiplas opções, você pode usar `$parser->addOptions()` para adicioná-las de uma vez.:

```
$parser->addOptions([
    'node' => ['short' => 'n', 'help' => 'The node to create'],
    'parent' => ['short' => 'p', 'help' => 'The parent node']
]);
```

Assim como com todos os métodos construtores, no ConsoleOptionParser, addOptions pode ser usado como parte de um método fluente encadeado.

Validando opções

Opções podem ser fornecidas com um conjunto de escolhas bem como argumentos posicionais podem ser. Quando uma opção define escolhas, essas são as únicas opções válidas para uma opção. Todos os outros valores irão gerar um `InvalidArgumentException`:

```
$parser->addOption('accept', [
    'help' => 'What version to accept.',
    'choices' => ['working', 'theirs', 'mine']
]);
```

Usando opções boleanas

As opções podem ser definidas como opções boleanas, que são úteis quando você precisa criar algumas opções de marcação. Como opções com padrões, opções boleanas sempre irão incluir -se nos parâmetros analisados. Quando as marcações estão presentes elas são definidas para `true`, quando elas estão ausentes, são definidas como `false`:

```
$parser->addOption('verbose', [
    'help' => 'Enable verbose output.',
    'boolean' => true
]);
```

A opção seguinte resultaria em `$this->params['verbose']` sempre estando disponível. Isso permite a você omitir verificações `empty()` ou `isset()` em marcações boleanas:

```
if ($this->params['verbose']) {
    // Do something.
}
```

Desde que as opções boleanas estejam sempre definidas como `true` ou `false`, você pode omitir métodos de verificação adicionais.

Adicionando subcomandos

`Cake\Console\ConsoleOptionParser::addSubcommand`(*$name*, *$options* =[])

Aplicativos de console são muitas vezes feitas de subcomandos, e esses subcomandos podem exigir a análise de opções especiais e terem a sua própria ajuda. Um perfeito exemplo disso é `bake`. Bake é feita de muitas tarefas separadas e todas têm a sua própria ajuda e opções. `ConsoleOptionParser` permite definir subcomandos e fornecer comandos analisadores de opção específica, de modo que a shell sabe como analisar os comandos para as suas funções:

```
$parser->addSubcommand('model', [
    'help' => 'Bake a model',
    'parser' => $this->Model->getOptionParser()
]);
```

A descrição acima é um exemplo de como você poderia fornecer ajuda e um especializado interpretador de opção para a tarefa de uma shell. Ao chamar a tarefa de `getOptionParser()` não temos de duplicar a geração do interpretador de opção, ou misturar preocupações no nosso shell. Adicionar subcomandos desta forma tem duas vantagens. Primeiro, ele permite que o seu shell documente facilmente seus subcomandos na ajuda gerada. Ele também dá fácil acesso ao subcomando help. Com o subcomando acima criado você poderia chamar `cake myshell --help` e ver a lista de subcomandos, e também executar o `cake myshell model --help` para exibir a ajuda apenas o modelo de tarefa.

Nota: Uma vez que seu Shell define subcomandos, todos os subcomandos deve ser explicitamente definidos.

Ao definir um subcomando, você pode usar as seguintes opções:

- `help` - Texto de ajuda para o subcomando.
- `parser` - Um ConsoleOptionParser para o subcomando. Isso permite que você crie métodos analisadores de opção específios. Quando a ajuda é gerada por um subcomando, se um analisador está presente ele vai ser usado. Você também pode fornecer o analisador como uma matriz que seja compatível com `Cake\Console\ConsoleOptionParser::buildFromArray()`

Adicionar subcomandos pode ser feito como parte de uma cadeia de métodos fluente.

Construir uma ConsoleOptionParser de uma matriz

`Cake\Console\ConsoleOptionParser::buildFromArray`(*$spec*)

Como mencionado anteriormente, ao criar interpretadores de opção de subcomando, você pode definir a especificação interpretadora como uma matriz para esse método. Isso pode ajudar fazer analisadores mais facilmente, já que tudo é um array:

```
$parser->addSubcommand('check', [
    'help' => __('Check the permissions between an ACO and ARO.'),
    'parser' => [
        'description' => [
            __("Use this command to grant ACL permissions. Once executed, the "),
            __("ARO specified (and its children, if any) will have ALLOW access "),
```

```
            __("to the specified ACO action (and the ACO's children, if any).")
        ],
        'arguments' => [
            'aro' => ['help' => __('ARO to check.'), 'required' => true],
            'aco' => ['help' => __('ACO to check.'), 'required' => true],
            'action' => ['help' => __('Action to check')]
        ]
    ]
]);
```

Dentro da especificação do interpretador, você pode definir as chaves para `arguments`, `options`, `description` e `epilog`. Você não pode definir `subcommands` dentro de um construtor estilo array. Os valores para os argumentos e opções, devem seguir o formato que `Cake\Console\ConsoleOptionParser::addArguments()` e `Cake\Console\ConsoleOptionParser::addOptions()` usam. Você também pode usar buildFromArray por conta própria, para construir um interpretador de opção:

```
public function getOptionParser()
{
    return ConsoleOptionParser::buildFromArray([
        'description' => [
            __("Use this command to grant ACL permissions. Once executed, the "),
            __("ARO specified (and its children, if any) will have ALLOW access "),
            __("to the specified ACO action (and the ACO's children, if any).")
        ],
        'arguments' => [
            'aro' => ['help' => __('ARO to check.'), 'required' => true],
            'aco' => ['help' => __('ACO to check.'), 'required' => true],
            'action' => ['help' => __('Action to check')]
        ]
    ]);
}
```

Recebendo ajuda das Shells

Com a adição de ConsoleOptionParser receber ajuda de shells é feito de uma forma consistente e uniforme. Ao usar a opção `--help` ou `-h` você pode visualizar a ajuda para qualquer núcleo shell, e qualquer shell que implementa um ConsoleOptionParser:

```
cake bake --help
cake bake -h
```

Ambos devem gerar a ajuda para o bake. Se o shell suporta subcomandos você pode obter ajuda para estes de uma forma semelhante:

```
cake bake model --help
cake bake model -h
```

Isso deve fornecer a você a ajuda específica para a tarefa bake dos models.

Recebendo ajuda como XML

Quando a construção de ferramentas automatizadas ou ferramentas de desenvolvimento que necessitam interagir com shells do CakePHP, é bom ter ajuda disponível em uma máquina capaz interpretar formatos. O ConsoleOptionParser pode fornecer ajuda em xml, definindo um argumento adicional:

```
cake bake --help xml
cake bake -h xml
```

O trecho acima deve retornar um documento XML com a ajuda gerada, opções, argumentos e subcomando para o shell selecionado. Um documento XML de amostra seria algo como:

```
<?xml version="1.0"?>
<shell>
    <command>bake fixture</command>
    <description>Generate fixtures for use with the test suite. You can use
        `bake fixture all` to bake all fixtures.</description>
    <epilog>
        Omitting all arguments and options will enter into an interactive
        mode.
    </epilog>
    <subcommands/>
    <options>
        <option name="--help" short="-h" boolean="1">
            <default/>
            <choices/>
        </option>
        <option name="--verbose" short="-v" boolean="1">
            <default/>
            <choices/>
        </option>
        <option name="--quiet" short="-q" boolean="1">
            <default/>
            <choices/>
        </option>
        <option name="--count" short="-n" boolean="">
            <default>10</default>
            <choices/>
        </option>
        <option name="--connection" short="-c" boolean="">
            <default>default</default>
            <choices/>
        </option>
        <option name="--plugin" short="-p" boolean="">
            <default/>
            <choices/>
        </option>
        <option name="--records" short="-r" boolean="1">
            <default/>
            <choices/>
        </option>
    </options>
    <arguments>
        <argument name="name" help="Name of the fixture to bake.
```

```
            Can use Plugin.name to bake plugin fixtures." required="">
            <choices/>
        </argument>
    </arguments>
</shell>
```

Roteamento em Shells / CLI

Na interface de linha de comando (CLI), especificamente suas shells e tarefas, `env('HTTP_HOST')` e outras variáveis de ambiente webbrowser específica, não estão definidas.

Se você gerar relatórios ou enviar e-mails que fazem uso de `Router::url()`, estes conterão a máquina padrão `http://localhost/` e resultando assim em URLs inválidas. Neste caso, você precisa especificar o domínio manualmente. Você pode fazer isso usando o valor de configuração `App.fullBaseURL` no seu bootstrap ou na sua configuração, por exemplo.

Para enviar e-mails, você deve fornecer a classe CakeEmail com o host que você deseja enviar o e-mail:

```
$Email = new CakeEmail();
$Email->domain('www.example.org');
```

Iste afirma que os IDs de mensagens geradas são válidos e adequados para o domínio a partir do qual os e-mails são enviados.

Métodos enganchados

`Cake\Console\ConsoleOptionParser::initialize()`
: Inicializa a Shell para atua como construtor de subclasses e permite configuração de tarefas antes de desenvolver a execução.

`Cake\Console\ConsoleOptionParser::startup()`
: Inicia-se a Shell e exibe a mensagem de boas-vindas. Permite a verificação e configuração antes de comandar ou da execução principal.

  Substitua este método se você quiser remover as informações de boas-vindas, ou outra forma modificar o fluxo de pré-comando.

Mais tópicos

Shell Helpers

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

[1] https://github.com/cakephp/docs

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Console Interativo (REPL)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Executando Shells como Cron Jobs

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[3] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

I18N Shell

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[4] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Completion Shell

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[5] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

[2] https://github.com/cakephp/docs
[3] https://github.com/cakephp/docs
[4] https://github.com/cakephp/docs
[5] https://github.com/cakephp/docs

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Plugin Shell

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[6] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Routes Shell

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[7] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Upgrade Shell

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[8] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Server Shell

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[9] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

[6] https://github.com/cakephp/docs
[7] https://github.com/cakephp/docs
[8] https://github.com/cakephp/docs
[9] https://github.com/cakephp/docs

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

CAPÍTULO 17

Debugging

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 18

Implantação

Uma vez que sua aplicação está completa, ou mesmo antes quando você quiser colocá-la no ar. Existem algumas poucas coisas que você deve fazer quando colocar em produção uma aplicação CakePHP.

Atualizar config/app.php

Atualizar o arquivo **core.php**, especificamente o valor do `debug` é de extrema importância. Tornar o debug igual a `false` desabilita muitos recursos do processo de desenvolvimento que nunca devem ser expostos ao mundo. Desabilitar o debug, altera as seguintes coisas:

- Mensagens de depuração criadas com `pr()` e `debug()` serão desabilitadas.
- O cache interno do CakePHP será descartado após 999 dias ao invés de ser a cada 10 segundos como em desenvolvimento.
- Views de erros serão menos informativas, retornando mensagens de erros genéricas.
- Erros do PHP não serão mostrados.
- O rastreamento de stack traces (conjunto de exceções) será desabilitado.

Além dos itens citados acima, muitos plugins e extensões usam o valor do `debug` para modificarem seus comportamentos.

Por exemplo, você pode setar uma variável de ambiente em sua configuração do Apache:

```
SetEnv CAKEPHP_DEBUG 1
```

E então você pode definir o level de debug dinamicamente no **config/app.php**:

```
$debug = (bool)getenv('CAKEPHP_DEBUG');

return [
    'debug' => $debug,
    .....
];
```

Checar a segurança

Se você está jogando sua aplicação na selva, é uma boa idéia certificar-se que ela não possui vulnerabilidades óbvias:

- Certifique-se de utilizar o *Cross Site Request Forgery*.
- Você pode querer habilitar o *Security*. Isso pode prevenir diversos tipos de adulteração de formulários e reduzir a possibilidade de overdose de requisições.
- Certifique-se que seus models possuem as regras *Validação* de validação habilitadas.
- Verifique se apenas o seu diretório `webroot` é visível publicamente, e que seus segredos (como seu app salt, e qualquer chave de segurança) são privados e únicos também.

Definir a raiz do documento

Definir a raiz do documento da sua aplicação corretamente é um passo importante para manter seu código protegido e sua aplicação mais segura. As aplicações desenvolvidas com o CakePHP devem ter a raiz apontando para o diretório `webroot`. Isto torna a aplicação e os arquivos de configurações inacessíveis via URL. Configurar a raiz do documento depende de cada servidor web. Veja a *Reescrita de URL* para informações sobre servidores web específicos.

De qualquer forma você vai querer definir o host/domínio virtual para o `webroot/`. Isso remove a possibilidade de arquivos fora do diretório raiz serem executados.

Aprimorar a performance de sua aplicação

O carregamento de classes pode alocar facilmente o tempo de processamento de sua aplicação. A fim de evitar esse problema, é recomendado que você execute este comando em seu servidor de produção uma vez que a aplicação esteja implantada:

```
php composer.phar dumpautoload -o
```

Sabendo que manipulação de referências estáticas, como imagens, JavaScript e arquivos CSS, plugins, através do `Dispatcher` é incrivelmente ineficiente, é fortemente recomendado referenciá-los simbolicamente para produção. Por exemplo:

```
ln -s Plugin/YourPlugin/webroot/css/yourplugin.css webroot/css/yourplugin.css
```

CAPÍTULO 19

Email

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 20

Erros & Exceções

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 21

Sistema de eventos

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 22

Internacionalização e Localização

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 23

Logging

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 24

Formulários sem models

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 25

Pagination

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 26

Plugins

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 27

REST

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 28

Segurança

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Segurança

Choosing a Specific Crypto Implementation

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Cross Site Request Forgery

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[3] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

[1] https://github.com/cakephp/docs
[2] https://github.com/cakephp/docs
[3] https://github.com/cakephp/docs

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Security

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[4] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[4] https://github.com/cakephp/docs

CAPÍTULO 29

Sessions

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 30

Testing

O CakePHP vem com suporte interno para testes e integração para o PHPUnit[1]. Em adição aos recursos oferecidos pelo PHPUnit, o CakePHP oferece alguns recursos adicionais para fazer testes mais facilmente. Esta seção abordará a instalação do PHPUnit, começando com testes unitários e como você pode usar as extensões que o CakePHP oferece.

Instalando o PHPUnit

O CakePHP usa o PHPUnit como framework de teste básico. O PHPUnit é um padrão para testes unitários em PHP. Ele oferece um profundo e poderoso conjunto de recursos para você ter certeza que o seu código faz o que você acha que ele faz. O PHPUnit pode ser instalado usando o PHAR package[2] ou Composer[3].

Instalando o PHPUnit com Composer

Para instalar o PHPUnit com Composer:

```
$ php composer.phar require --dev phpunit/phpunit
```

Isto adicionará a dependência para a seção `require-dev` do seu `composer.json`, e depois instalará o PHPUnit com qualquer outra dependência.

Agora você executa o PHPUnit usando:

```
$ vendor/bin/phpunit
```

Usando o arquivo PHAR

Depois de ter baixado o arquivo **phpunit.phar** , você pode usar ele para executar seus testes:

[1] http://phpunit.de
[2] http://phpunit.de/#download
[3] http://getcomposer.org

```
php phpunit.phar
```

Dica: Como conveniência você pode deixar phpunit.phar disponivel globalmente em sistemas Unix ou Linux com os comandos:

```
chmod +x phpunit.phar
sudo mv phpunit.phar /usr/local/bin/phpunit
phpunit --version
```

Por favor, consulte a documentação do PHPUnit para instruções sobre como instalar globalmente o PHPUnit PHAR em sistemas Windows[4].

Configuração do banco de dados test

Lembre-se de ter o debug abilitado em seu arquivo **config/app.php** antes de executar qualquer teste. Antes de executar quaisquer testes você deve adicionar um datasourse `test` para o arquivo **config/app.php**. Esta configuração é usada pelo CakePHP para fixar tabelas e dados:

```
'Datasources' => [
    'test' => [
        'datasource' => 'Cake\Database\Driver\Mysql',
        'persistent' => false,
        'host' => 'dbhost',
        'username' => 'dblogin',
        'password' => 'dbpassword',
        'database' => 'test_database'
    ],
],
```

Controller Integration Testing

Nota: É uma boa ideia usar bancos de dados diferentes para o banco de testes e para o banco de desenvolvimento. Isto evitara erros mais tarde.

[4] http://phpunit.de/manual/current/en/installation.html#installation.phar.windows

CAPÍTULO 31

Validação

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 32

App Class

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 33

Collections (Coleções)

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 34

Arquivos & Pastas

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 35

Hash

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 36

Http Client

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 37

Inflector

class Cake\Utility**Inflector**

A classe `Inflector` recebe uma string e a manipula afim de suportar variações de palavras como pluralizações ou CamelCase e normalmente é acessada estaticamente. Exemplo: `Inflector::pluralize('example')` retorna "examples".

Você pode testar as inflexões em inflector.cakephp.org[1].

Criando as formas singulares e plurais

static Cake\Utility\Inflector::**singularize**(*$singular*)

static Cake\Utility\Inflector::**pluralize**(*$singular*)

Tanto `pluralize()` quanto `singularize()` funcionam para a maioria dos substantivos do Inglês. Caso seja necessário o suporte para outras línguas, você pode usar *Configuração da inflexão* para personalizar as regras usadas:

```
// Apples
echo Inflector::pluralize('Apple');
```

Nota: `pluralize()` pode não funcionar corretamente nos casos onde um substantivo já esteja em sua forma plural.

```
// Person
echo Inflector::singularize('People');
```

Nota: `singularize()` pode não funcionar corretamente nos casos onde um substantivo já esteja em sua forma singular.

[1] http://inflector.cakephp.org/

Criando as formas CamelCase e nome_sublinhado

static Cake\Utility\Inflector::**camelize**(*$underscored*)

static Cake\Utility\Inflector::**underscore**(*$camelCase*)

Estes métodos são úteis para a criação de nomes de classe ou de propriedades:

```
// ApplePie
Inflector::camelize('Apple_pie')

// apple_pie
Inflector::underscore('ApplePie');
```

É importante ressaltar que `underscore()` irá converter apenas palavras formatadas em CamelCase. Palavras com espaços serão convertidas para caixa baixa, mas não serão separadas por sublinhado.

Criando formas legíveis para humanos

static Cake\Utility\Inflector::**humanize**(*$underscored*)

Este método é útil para converter da forma sublinhada para o "Formato Título" para a leitura humana:

```
// Apple Pie
Inflector::humanize('apple_pie');
```

Criando formatos para nomes de tabelas e classes

static Cake\Utility\Inflector::**classify**(*$underscored*)

static Cake\Utility\Inflector::**dasherize**(*$dashed*)

static Cake\Utility\Inflector::**tableize**(*$camelCase*)

Ao gerar o código ou usar as convenções do CakePHP, você pode precisar inferir os nomes das tabelas ou classes:

```
// UserProfileSettings
Inflector::classify('user_profile_settings');

// user-profile-setting
Inflector::dasherize('UserProfileSetting');

// user_profile_settings
Inflector::tableize('UserProfileSetting');
```

Criando nomes de variáveis

static Cake\Utility\Inflector::**variable**(*$underscored*)

Nomes de variáveis geralmente são úteis em tarefas de meta-programação que involvem a geração de código ou rotinas baseadas em convenções:

```
// applePie
Inflector::variable('apple_pie');
```

Criando strings de URL seguras

static `Cake\Utility\Inflector::slug`(*$word*, *$replacement* = '-')

`slug()` converte caracteres especiais em suas versões normais e converte os caracteres não encontrados e espaços em traços. O método `slug()` espera que a codificação seja UTF-8:

```
// apple-puree
Inflector::slug('apple purée');
```

Nota: `Inflector::slug()` foi depreciado desde a versão 3.2.7. Procure usar `Text::slug()` de agora em diante.

Configuração da inflexão

As convenções de nomes do CakePHP podem ser bem confortáveis. Você pode nomear sua tabela no banco de dados como `big_boxes`, seu modelo como `BigBoxes`, seu controlador como `BigBoxesController` e tudo funcionará automaticamente. O CakePHP entrelaça todos estes conceitos através da inflexão das palavras em suas formas singulares e plurais.

Porém ocasionalmente (especialmente para os nossos amigos não Anglófonos) podem encontrar situações onde o infletor do CakePHP (a classe que pluraliza, singulariza, transforma em CamelCase e em nome_sublinhado) não funciona como você gostaria. Caso o CakePHP não reconheça seu "quaisquer" ou "lápis", você pode ensiná-lo a entender seus casos especiais.

Carregando inflexões personalizadas

static `Cake\Utility\Inflector::rules`(*$type*, *$rules*, *$reset* = *false*)

Define novas inflexões e transliterações para o `Inflector` usar. Geralmente este método deve ser chamado no seu **config/bootstrap.php**:

```
Inflector::rules('singular', ['/^(bil)er$/i' => '\1', '/^(inflec|contribu)tors$/i' => '\1ta
Inflector::rules('uninflected', ['singulars']);
Inflector::rules('irregular', ['phylum' => 'phyla']); // The key is singular form, value is
```

As regras ditadas por este método serão agregadas aos conjuntos de inflexão definidos em `Cake/Utility/Inflector`, onde elas terão prioridade sobre as regras já declaradas por padrão. Você pode usar `Inflector::reset()` para limpar todas as regras e retornar o `Inflector` para seu estado original.

CAPÍTULO 38

Número

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 39

Objetos de Registro

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 40

Texto

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 41

Tempo

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 42

Xml

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 43

Constantes e Funções

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 44

Debug Kit

DebugKit é um plugin suportado pelo time principal que oferece uma barra de ferramentas para ajudar a fazer a depuração de aplicações do CakePHP mais facilmente.

Instalação

Por padrão o DebugKit é instalado com com o esqueleto padrão da aplicação. Se você o removeu e gostaria de reinstalá-lo, você pode fazê-lo ao executar o seguinte comando a partir do diretório raiz da aplicação (onde o arquivo composer.json está localizado):

```
php composer.phar require --dev cakephp/debug_kit "~3.0"
```

Then, you need to enable the plugin by executing the following line:

```
bin/cake plugin load DebugKit
```

Armazenamento do DebugKit

Por padrão, o DebugKit usa um pequeno banco de dados SQLite no diretório `/tmp` de sua aplicação para armazenar os dados do painel. Se você quizesse que o DebugKit armazenasse seus dados em outro lugar, você deve definir uma conexão `debug_kit`.

Configuração doe banco de dados

Por padrão, o DebugKit armazenará os dados do painel em um banco de dados SQLite no diretório `/tmp` de sua aplicação. Se você não puder instalar a extensão do PHP pdo_sqlite, você pode configurar o DebugKit para usar um banco de dados diferente ao definir uma conexão `debug_kit` em seu arquivo **config/app.php**.

Uso da barra de ferramentas

A Barra de Ferramentas DebugKit é composta por vários painéis, que são mostrados ao clicar no ícone do CakePHP no canto inferior direito da janela do seu navegador. Uma vez que a barra de ferramentas é aberta, você deve ver uma série de botões. Cada um desses botões expande-se em um painel de informações relacionadas.

Cada painel permite que você olhar para um aspecto diferente da sua aplicação:

- **Cache** Exibe o uso de cache durante uma solicitação e limpa caches.
- **Environment** Exibe variáveis de ambiente relacionadas com PHP + CakePHP.
- **History** Exibe uma lista de requisições anteriores, e permite que você carregue e veja dados da barra de ferramentas a partir de solicitações anteriores.
- **Include** Exibe os arquivos inclusos divididos por grupo.
- **Log** Exibe todas as entradas feitas nos arquivos de log este pedido.
- **Request** Exibe informações sobre a requisição atual, GET, POST, informações sobre a rota atual do Cake e Cookies.
- **Session** Exibe a informação atual da sessão.
- **Sql Logs** Exibe logs SQL para cada conexão com o banco de dados.
- **Timer** Exibe qualquer temporizador que fora definido durante a requisição com `DebugKit\DebugTimer`, e memória utilizada coletada com `DebugKit\DebugMemory`.
- **Variables** Exibe variáveis de View definidas em um Controller.

Tipicamente, um painel manipula a recolha e apresentação de um único tipo de informações como logs ou informações de requisições. Você pode optar por visualizar painéis padrão da barra de ferramentas ou adicionar seus próprios painéis personalizados.

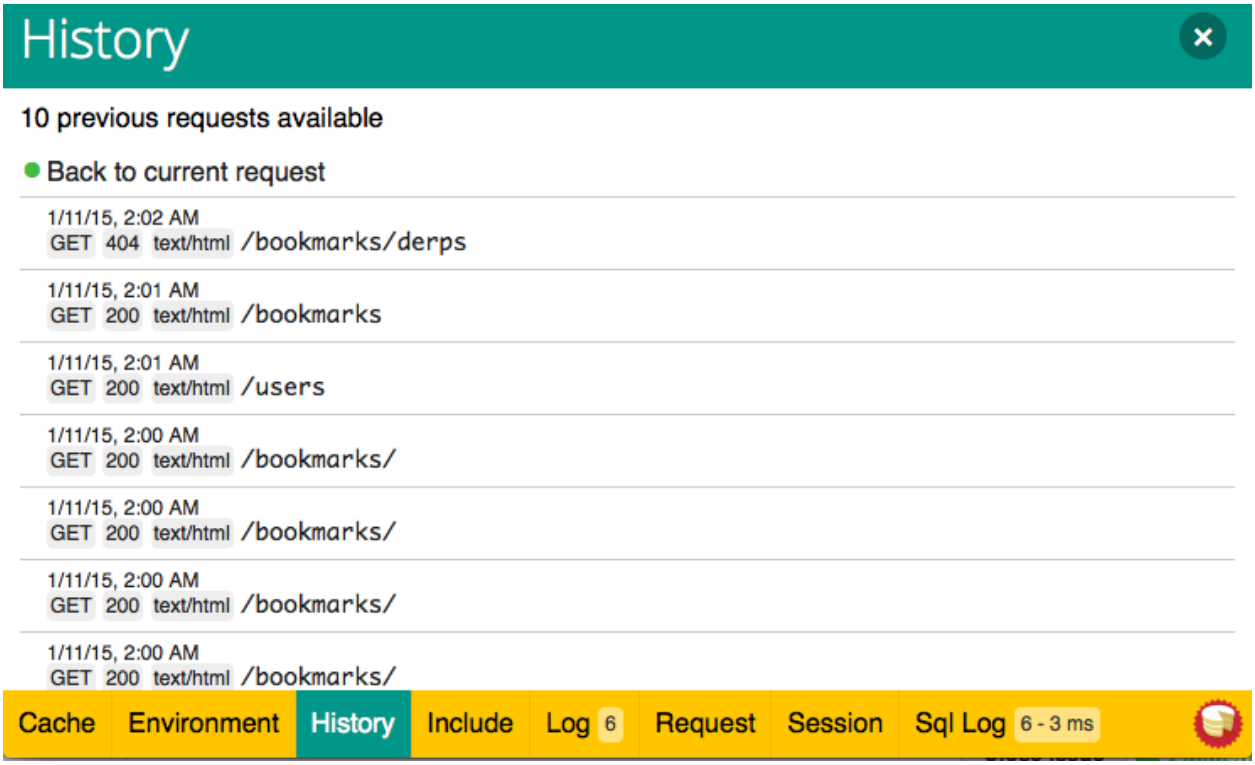
Usando o painel History

O painel History é uma das mais frequentemente confundidas características do DebugKit. Ele oferece uma forma de visualizar os dados da barra de ferramentas de requisições anteriores, incluindo erros e redirecionamentos.

Como você pode ver, o painel contém uma lista de requisições. Na esquerda você pode ver um ponto marcando a requisição ativa. Clicar em quaisquer dados de requisição vai carregar os dados do painel para aquela requisição. Quando os dados são carregados, os títulos do painel vão sofrer uma transição para indicar que dados alternativos foram carregados.

Desenvolvendo seus próprios painéis

Você pode criar seus próprios painéis customizados para o DebugKit para ajudar na depuração de suas aplicações.



Criando uma Panel Class

Panel Classes precisam ser colocadas no diretório **src/Panel**. O nome do arquivo deve combinar com o nome da classe, então a classe `MyCustomPanel` deveria ter o nome de arquivo **src/Panel/MyCustomPanel.php**:

```
namespace App\Panel;

use DebugKit\DebugPanel;

/**
 * My Custom Panel
 */
class MyCustomPanel extends DebugPanel
{
    ...
}
```

Perceba que painéis customizados são necessários para extender a classe `DebugPanel`.

Callbacks

Por padrão objetos do painel possuem dois callbacks, permitindo-lhes acoplar-se na requisição atual. Painéis inscrevem-se aos eventos `Controller.initialize` e `Controller.shutdown`. Se o seu painel precisa inscrever-se a eventos adicionais, você pode usar o método `implementedEvents` para definir todos os eventos que o seu painel possa estar interessado.

Você deveria estudar os painéis nativos para absorver alguns exemplos de como construir painéis.

Elementos do painel

Cada painel deve ter um elemento view que renderiza o conteúdo do mesmo. O nome do elemento deve ser sublinhado e flexionado a partir do nome da classe. Por exemplo `SessionPanel` possui um elemento nomeado **session_panel.ctp**, e SqllogPanel possui um elemento nomeado **sqllog_panel.ctp**. Estes elementos devem estar localizados na raiz do seu diretório **src/Template/Element**.

Títulos personalizados e Elementos

Os painéis devem pegar o seu título e nome do elemento por convenção. No entanto, se você precisa escolher um nome de elemento personalizado ou título, você pode definir métodos para customizar o comportamento do seu painel:

- `title()` - Configure o título que é exibido na barra de ferramentas.
- `elementName()` - Configure qual elemento deve ser utilizada para um determinado painel.

Painéis em outros plugins

Painéis disponibilizados por *Plugins* funcionam quase que totalmente como outros plugins, com uma pequena diferença: Você deve definir `public $plugin` para ser o nome do diretório do plugin, com isso os elementos do painel poderão ser encontrados no momento de renderização:

```
namespace MyPlugin\Panel;

use DebugKit\DebugPanel;

class MyCustomPanel extends DebugPanel
{
    public $plugin = 'MyPlugin';
        ...
}
```

Para usar um plugin ou painel da aplicação, atualize a configuração do DebugKit de sua aplicação para incluir o painel:

```
Configure::write(
    'DebugKit.panels',
    array_merge(Configure::read('DebugKit.panels'), ['MyCustomPanel'])
);
```

O código acima deve carregar todos os painéis padrão tanto como os outros painéis customizados do `MyPlugin`.

CAPÍTULO 45

Migrations

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

[1] https://github.com/cakephp/docs

CAPÍTULO 46

Apêndices

Os apêndices contêm informações sobre os novos recursos introduzidos em cada versão e a forma de executar a migração entre versões.

Guia de Migração para a versão 3.x

3.x Migration Guide

Migration guides contain information regarding the new features introduced in each version and the migration path between versions.

3.2 Migration Guide

3.2 Guia de migração

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[1] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

3.1 Migration Guide

3.1 Guia de migração

---

[1] https://github.com/cakephp/docs

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[2] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

3.0 Migration Guide

3.0 - Guia de migração

Esta página resume as alterações do CakePHP 2.x e irá auxiliar na migração do seu projeto para a versão 3.0, e também será uma referência para atualizá-lo quanto às principais mudanças do branch 2.x. Certifique-se de ler também as outras páginas nesse guia para conhecer todas as novas funcionalidades e mudanças na API.

Requerimentos

- O CakePHP 3.x suporta o PHP 5.4.16 e acima.
- O CakePHP 3.x precisa da extensão mbstring.
- O CakePHP 3.x precisa da extensão intl.

Aviso: O CakePHP 3.0 não irá funcionar se você não atender aos requisitos acima.

Ferramenta de atualização Enquanto este documento cobre todas as alterações e melhorias feitas no CakePHP 3.0, nós também criamos uma aplicação de console para ajudar você a completar mais facilmente algumas das alterações mecânicas que consomem tempo. Você pode pegar a ferramenta de atualização no github[3].

Layout do diretório da aplicação O Layout do diretório da aplicação mudou e agora segue o PSR-4[4]. Você deve usar o projeto do esqueleto da aplicação[5] como um ponto de referência quando atualizar sua aplicação.

O CakePHP deve ser instalado via Composer Como o CakePHP não pode mais ser instalado facilmente via PEAR, ou em um diretório compartilhado, essas opções não são mais suportadas. Ao invés disso, você deve usar o Composer[6] para instalar o CakePHP em sua aplicação.

[2] https://github.com/cakephp/docs
[3] https://github.com/cakephp/upgrade
[4] http://www.php-fig.org/psr/psr-4/
[5] https://github.com/cakephp/app
[6] http://getcomposer.org

Namespaces Todas as classes do core do CakePHP agora usam namespaces e seguem as especificações de autoload (auto-carregamento) do PSR-4. Por exemplo **src/Cache/Cache.php** tem o namespace `Cake\Cache\Cache`. Constantes globais e métodos de helpers como `__()` e `debug()` não usam namespaces por questões de conveniência.

Constantes removidas As seguintes constantes obsoletas foram removidas:

- `IMAGES`
- `CSS`
- `JS`
- `IMAGES_URL`
- `JS_URL`
- `CSS_URL`
- `DEFAULT_LANGUAGE`

Configuração As configurações no CakePHP 3.0 estão significativamente diferentes que nas versões anteriores. Você deve ler a documentação *Configuração* para ver como a configuração é feita.

Você não pode mais usar o `App::build()` para configurar caminhos adicionais de classes. Ao invés disso, você deve mapear caminhos adicionais usando o autoloader da sua aplicação. Veja a seção *Caminhos de Classes Adicionais* para mais informações.

Três novas variáveis de configuração fornecem o caminho de configuração para plugins, views e arquivos de localização. Você pode adicionar vários caminhos em `App.paths.templates`, `App.paths.plugins`, `App.paths.locales` para configurar múltiplos caminhos para templates, plugins e arquivos de localização respectivamente.

A chave de configuração `www_root` mudou para `wwwRoot` devido a consistência. Por favor, ajuste seu arquivo de configuração `app.php` assim como qualquer uso de `Configure::read('App.wwwRoot')`.

Novo ORM O CakePHP 3.0 possui um novo ORM que foi refeito do zero. O novo ORM é significativamente diferente e incompatível com o anterior. Migrar para o novo ORM necessita de alterações extensas em qualquer aplicação que esteja sendo atualizada. Veja a nova documentação *Models (Modelos)* para informações de como usar o novo ORM.

Básico

- O `LogError()` foi removido, ele não tinha vantagens e era raramente ou mesmo, nunca usado.
- As seguintes funções globais foram removidas: `config()`, `cache()`, `clearCache()`, `convertSlashes()`, `am()`, `fileExistsInPath()`, `sortByKey()`.

Debug

- A função `Configure::write('debug', $bool)` não suporta mais 0/1/2. Um boleano simples é usado para mudar o modo de debug para ligado ou desligado.

Especificações/Configurações de objetos

- Os objetos usados no CakePHP agora tem um sistema consistente de armazenamento/recuperação de configuração-de-instância. Os códigos que anteriormente acessavam, por exemplo `$object->settings`, devem ser atualizados para usar `$object->config()` alternativamente.

Cache

- `Memcache` foi removido, use `Cake\Cache\Cache\Engine\Memcached` alternativamente.
- Cache engines são carregados sob demanda no primeiro uso.
- `Cake\Cache\Cache::engine()` foi adicionado.
- `Cake\Cache\Cache::enabled()` foi adicionado. Substituindo a opção de configuração `Cache.disable`.
- `Cake\Cache\Cache::enable()` foi adicionado.
- `Cake\Cache\Cache::disable()` foi adicionado.
- Configuração de cache agora é imutável. Se você precisa alterar a configuração, será necessário desfazer-se da configuração e recriá-la. Isso previne problemas de sincronização com as opções de configuração.
- `Cache::set()` foi removido. É recomendado criar múltiplas configurações de cache para substituir ajustes de configuração em tempo de execução anteriormente possíveis com `Cache::set()`.
- Todas as subclasses `CacheEngine` agora implementam um método `config()`.
- `Cake\Cache\Cache::readMany()`, `Cake\Cache\Cache::deleteMany()`, e `Cake\Cache\Cache::writeMany()` foram adicionados.

Todos os métodos `Cake\Cache\Cache\CacheEngine` agora são responsáveis por manipular o prefixo chave configurado. O `Cake\Cache\CacheEngine::write()` não mais permite definir a duração na escrita, a duração é captada pela configuração de tempo de execução do mecanismo de cache. Chamar um método cache com uma chuva vazia irá lançar uma `InvalidArgumentException` ao invés de retornar `false`.

Core

App

- `App::pluginPath()` foi removido. Use `CakePlugin::path()` alternativamente.
- `App::build()` foi removido.

- `App::location()` foi removido.
- `App::paths()` foi removido.
- `App::load()` foi removido.
- `App::objects()` foi removido.
- `App::RESET` foi removido.
- `App::APPEND` foi removido.
- `App::PREPEND` foi removido.
- `App::REGISTER` foi removido.

Plugin

- O `Cake\Core\Plugin::load()` não configura a carga automática a menos que você defina a opção `autoload` como `true`.
- Quanto estiver carregando plugins você não pode mais fornecer um `callable`.
- Quanto estiver carregando plugins você não pode mais fornecer um array de arquivos de configuração para carregar.

Configure

- O `Cake\Configure\PhpReader` foi renomeado para `Cake\Core\Configure\EnginePhpConfig`
- O `Cake\Configure\IniReader` foi renomeado para `Cake\Core\Configure\EngineIniConfig`
- O `Cake\Configure\ConfigReaderInterface` foi renomeado para `Cake\Core\Configure\ConfigEngineInterface`
- O `Cake\Core\Configure::consume()` foi adicionado.
- O `Cake\Core\Configure::load()` agora espera o nome de arquivo sem o sufixo de extensão como isso pode ser derivado do mecanismo. Ex.: para usar o PhpConfig use `app` para carregar `app.php`.
- Definir uma variável `$config` no arquivo PHP config está obsoleto. `Cake\Core\Configure\EnginePhpConfig` agora espera que o arquivo de configuração retorne um array.
- Um novo mecanismo de configuração `Cake\Core\Configure\EngineJsonConfig` foi adicionado.

Object A classe `Object` foi removida. Ela anteriormente continha um monte de métodos que eram utilizados em vários locais no framework. O mais útil destes métodos foi extraido como um `trait`. Você pode usar o `Cake\Log\LogTrait` para acessar o método `log()`. O `Cake\Routing\RequestActionTrait` fornece o método `requestAction()`.

Console O executável `cake` foi movido do diretório `app/Console` para o diretório `bin` dentro do esqueleto da aplicação. Você pode agora invocar o console do CakePHP com `bin/cake`.

TaskCollection Substituído Essa classe foi renomeada para `Cake\Console\TaskRegistry`. Veja a seção em *Objetos de Registro* para mais informações sobre funcionalidades fornecidas pela nova classe. Você pode usar o `cake upgrade rename_collections` para ajuda ao atualizar seu código. Tarefas não tem mais acesso a callbacks, como nunca houve nenhum callback para se usar.

Shell

- O `Shell::__construct()` foi alterado. Ele agora usa uma instância de `Cake\Console\ConsoleIo`.
- O `Shell::param()` foi adicionado como um acesso conveniente aos parâmetros.

Adicionalmente todos os métodos shell serão transformados em camel case quando invocados. Por exemplo, se você tem um método `hello_world()` dentro de um shell e chama ele com `bin/cake my_shell hello_world`, você terá que renomear o método para `helloWorld`. Não há necessidade de mudanças no modo que você chama os métodos/comandos.

ConsoleOptionParser

- O `ConsoleOptionParser::merge()` foi adicionado para mesclar os parsers.

ConsoleInputArgument

- O `ConsoleInputArgument::isEqualTo()` foi adicionado para comparar dois argumentos.

Shell / Tarefa Os Shells e Tarefas foram movidas de `Console/Command` e `Console/Command/Task` para `Shell` e `Shell/Task`, respectivamente.

ApiShell Removido O ApiShell foi removido pois ele não fornecia nenhum beneficio além do próprio arquivo fonte e da documentação/API[7] online.

SchemaShell Removido O SchemaShell foi removido como ele nunca foi uma implementação completa de migração de banco de dados e surgiram ferramentas melhores como o Phinx[8]. Ele foi substituído pelo CakePHP Migrations Plugin[9] que funciona como um empacotamento entre o CakePHP e o Phinx[10].

[7] http://api.cakephp.org/
[8] https://phinx.org/
[9] https://github.com/cakephp/migrations
[10] https://phinx.org/

ExtractTask

- O `bin/cake i18n extract` não inclui mais mensagens de validação sem tradução. Se você quiser mensagens de validação traduzidas você deve encapsula-las com chamadas *__()* como qualquer outro conteúdo.

BakeShell / TemplateTask

- O Bake não faz mais parte do fonte do núcleo e é suplantado pelo CakePHP Bake Plugin[11]
- Os templates do Bake foram movidos para **src/Template/Bake**.
- A sintaxe dos templates do Bake agora usam tags estilo erb (`<% %>`) para denotar lógica de template, permitindo código php ser tratado como texto plano.
- O comando `bake view` foi renomeado para `bake template`.

Eventos O método `getEventManager()`, foi removido de todos os objetos que continham. Um método `eventManager()` é agora fornecido pelo `EventManagerTrait`. O `EventManagerTrait` contém a lógica de instanciação e manutenção de uma referência para um gerenciador local de eventos.

O subsistema `Event` teve um monte de funcionalidades opcionais removidas. Quando despachar eventos você não poderá mais usar as seguintes opções:

- `passParams` Essa opção está agora ativada sempre implicitamente. Você não pode desliga-la.
- `break` Essa opção foi removida. Você deve agora parar os eventos.
- `breakOn` Essa opção foi removida. Você deve agora parar os eventos.

Log

- As configurações do Log agora não imutáveis. Se você precisa alterar a configuração você deve primeiro derrubar a configuração e então recria-la. Isso previne problemas de sincronização com opções de configuração.
- Os mecanismos de Log agora são carregados tardiamente após a primeira escrita nos logs.
- O `Cake\Log\Log::engine()` foi adicionado.
- Os seguintes métodos foram removidos de `Cake\Log\Log` :: `defaultLevels()`, `enabled()`, `enable()`, `disable()`.
- Você não pode mais criar níveis personalizados usando `Log::levels()`.
- Quando configurar os loggers você deve usar `'levels'` ao invés de `'types'`.
- Você não pode mais especificar níveis personalizados de log. Você deve usar o conjunto padrão de níveis de log. Você deve usar escopos de log para criar arquivos de log personalizados ou manipulações específicas para diferentes seções de sua aplicação. Usando um nível de log não padrão irá lançar uma exceção.

[11] https://github.com/cakephp/bake

- O `Cake\Log\LogTrait` foi adicionado. Você pode usar este trait em suas classes para adicionar o método `log()`.
- O escopo de log passado para `Cake\Log\Log::write()` é agora encaminhado para o método `write()` dos mecanismos de log de maneira a fornecer um melhor contexto para os mecanismos.
- Os mecanismos de Log agora são necessários para implementar `Psr\Log\LogInterface` invés do próprio `LogInterface` do Cake. Em geral, se você herdou o `Cake\Log\Engine\BaseEngine` você só precisa renomear o método `write()` para `log()`.
- O `Cake\Log\Engine\FileLog` agora grava arquivos em `ROOT/logs` no lugar de `ROOT/tmp/logs`.

Roteamento

Parâmetros Nomeados Os parâmetros nomeados foram removidos no 3.0. Os parâmetros nomeados foram adicionados no 1.2.0 como uma versão 'bonita' de parâmetros de requisição. Enquanto o benefício visual é discutível, os problemas criados pelos parâmetros nomeados não são.

Os parâmetros nomeados necessitam manipulação especial no CakePHP assim como em qualquer biblioteca PHP ou JavaScript que necessite interagir com eles, os parâmetros nomeados não são implementados ou entendidos por qualquer biblioteca *exceto* o CakePHP. A complexidade adicionada e o código necessário para dar suporte aos parâmetros nomeados não justificam a sua existência, e eles foram removidos. No lugar deles, você deve agora usar o padrão de parâmetros de requisição (querystring) ou argumentos passados configurados nas rotas. Por padrão o `Router` irá tratar qualquer parâmetro adicional ao `Router::url()` como argumentos de requisição.

Como muitas aplicações ainda precisarão analisar URLs contendo parâmetros nomeados, o `Cake\Routing\Router::parseNamedParams()` foi adicionado para permitir compatibilidade com URLs existentes.

RequestActionTrait

- O `Cake\Routing\RequestActionTrait::requestAction()` teve algumas de suas opções extras alteradas:
  - o `options[url]` é agora `options[query]`.
  - o `options[data]` é agora `options[post]`.
  - os parâmetros nomeados não são mais suportados.

Roteador

- Os parâmetros nomeados foram removidos, veja acima para mais informações.
- A opção `full_base` foi substituída com a opção `_full`.
- A opção `ext` foi substituída com a opção `_ext`.
- As opções `_scheme`, `_port`, `_host`, `_base`, `_full`, `_ext` foram adicionadas.

- As URLs em strings não são mais modificados pela adição de plugin/controller/nomes de prefixo.
- A manipulação da rota padrão de `fallback` foi removida. Se nenhuma rota combinar com o conjunto de parâmetros, o `/` será retornado.
- As classes de rota são responsáveis por *toda* geração de URLs incluindo parâmetros de requisição (query string). Isso faz com que as rotas sejam muito mais poderosas e flexíveis.
- Parâmetros persistentes foram removidos. Eles foram substituídos pelo `Cake\Routing\Router::urlFilter()` que permite um jeito mais flexível para mudar URLs sendo roteadas reversamente.
- O `Router::parseExtensions()` foi removido. Use o `Cake\Routing\Router::extensions()` no lugar. Esse método **deve** ser chamado antes das rotas serem conectadas. Ele não irá modificar rotas existentes.
- O `Router::setExtensions()` foi removido. Use o `Cake\Routing\Router::extensions()` no lugar.
- O `Router::resourceMap()` foi removido.
- A opção `[method]` foi renomeada para `_method`.
- A habilidade de combinar cabeçalhos arbitrários com parâmetros no estilo `[]` foi removida. Se você precisar combinar/analisar em condições arbitrárias considere usar classes personalizadas de roteamento.
- O `Router::promote()` foi removido.
- O `Router::parse()` irá agora lançar uma exceção quando uma URL não puder ser atendida por nenhuma rota.
- O `Router::url()` agora irá lançar uma exceção quando nenhuma rota combinar com um conjunto de parâmetros.
- Os escopos de rotas foram adicionados. Escopos de rotas permitem você manter seu arquivo de rotas limpo e dar dicas de rotas em como otimizar análise e reversão de rotas de URL.

Route

- O `CakeRoute` foi renomeado para `Route`.
- A assinatura de `match()` mudou para `match($url, $context = [])`. Veja `Cake\Routing\Route::match()` para mais informações sobre a nova assinatura.

Configuração de Filtros do Despachante Mudaram Os filtros do despachante não são mais adicionados em sua aplicação usando o `Configure`. Você deve agora anexa-los com `Cake\Routing\DispatcherFactory`. Isso significa que sua aplicação usava `Dispatcher.filters`, você deve usar agora o método `Cake\Routing\DispatcherFactory::add()`.

Além das mudanças de configuração, os filtros do despachante tiveram algumas convenções atualizadas e novas funcionalidades. Veja a documentação em *Filtros do Dispatcher* para mais informações.

FilterAssetFilter

- Os itens de plugins e temas manipulados pelo AssetFilter não são mais lidos via `include`, ao invés disso eles são tratados como arquivos de texto plano. Isso corrige um número de problemas com bibliotecas javascript como TinyMCE e ambientes com short_tags ativadas.
- O suporte para a configuração `Asset.filter` e ganchos foram removidos. Essa funcionalidade pode ser facilmente substituída com um plugin ou filtro de despachante.

Rede

Requisição

- O `CakeRequest` foi renomeada para `Cake\Network\Request`.
- O `Cake\Network\Request::port()` foi adicionado.
- O `Cake\Network\Request::scheme()` foi adicionado.
- O `Cake\Network\Request::cookie()` foi adicionado.
- O `Cake\Network\Request::$trustProxy` foi adicionado. Isso torna mais fácil colocar aplicações CakePHP atrás de balanceadores de carga.
- O `Cake\Network\Request::$data` não é mais mesclado com a chave de dados prefixada, pois esse prefixo foi removido.
- O `Cake\Network\Request::env()` foi adicionado.
- O `Cake\Network\Request::acceptLanguage()` mudou de um método estático para não-estático.
- O detector de requisição para dispositivos móveis foi removido do núcleo. Agora o app template adiciona detectores para dispositivos móveis usando a biblioteca `MobileDetect`.
- O método `onlyAllow()` foi renomeado para `allowMethod()` e não aceita mais "argumentos var". Todos os nomes de métodos precisam ser passados como primeiro argumento, seja como string ou como array de strings.

Resposta

- O mapeamento do mimetype `text/plain` para extensão `csv` foi removido. Como consequência o `Cake\Controller\Component\RequestHandlerComponent` não define a extensão para `csv` se o cabeçalho `Accept` tiver o mimetype `text/plain` que era um problema comum quando recebia uma requisição XHR do jQuery.

Sessões A classe de sessão não é mais estática, agora a sessão (session) pode ser acessada através do objeto de requisição (request). Veja a documentação em *Sessions* para ver como usar o objeto de sessão.

- O `Cake\Network\Session` e classes de sessão relacionadas foram movidas para o namespace `Cake\Network`.
- O `SessionHandlerInterface` foi removido em favor ao fornecido pelo próprio PHP.

- A propriedade `Session::$requestCountdown` foi removida.
- O funcionalidade de sessão `checkAgent` foi removida. Ela causava um monte de bugs quando quadros do chrome e o flash player estavam envolvidos.
- A convenção de nome para a tabela de sessão no banco de dados agora é `sessions` ao invés de `cake_sessions`.
- O cookie de tempo limite da sessão é atualizado automaticamente em conjunto com o tempo limite dos dados de sessão.
- O caminho padrão para o cookie de sessão agora é o caminho base da aplicação, ao invés de "/". Além disso, uma nova variável de configuração `Session.cookiePath` foi adicionada para facilitar a personalização do caminho para os cookies.
- Um novo método conveniente `Cake\Network\Session::consume()` foi adicionado para permitir a leitura e exclusão de dados de sessão em um único passo.
- O valor padrão do argumento `$renew` de `Cake\Network\Session::clear()` mudou de `true` para `false`.

Network\Http

- O `HttpSocket` agora é `Cake\Network\Http\Client`.
- O HttpClient foi reescrito do zero. Ele tem uma API mais simples/fácil de usar, suporta novos sistemas de autenticação como OAuth, e uploads de arquivos. Ele usa as API de stream do PHP de modo que não há requerimentp para o cURL. Veja a documentação *Http Client* para mais informações.

Network\Email

- O `Cake\Network\Email\Email::config()` agora é usado para definir perfis de configuração. Isso substitui as classes `EmailConfig` nas versões anteriores.
- O `Cake\Network\Email\Email::profile()` substitui o `config()` como modo de modificar opções de configuração por instância.
- O `Cake\Network\Email\Email::drop()` foi adicionado para permitir a remoção de configurações de email.
- O `Cake\Network\Email\Email::configTransport()` foi adicionado para permitir a definição de configurações de transporte. Essa mudança retira as opções de transporte dos perfis de entrega e permite a você reusar facilmente os transportes através de perfis de e-mails.
- O `Cake\Network\Email\Email::dropTransport()` foi adicionado para permitir a remoção de configurações de transporte.

Controller

Controller

- As propriedades `$helpers` e `$components` agora estão mescladas com **todas** classes pai, não apenas a `AppController` e o plugin de AppController. As propriedades são mescladas de modo diferente agora também. No lugar de todas as configurações em todas as classes serem mescladas juntas, as configurações definidas nas classes filho serão usadas. Isso quer dizer que se você tem alguma configurações definida no seu AppController, e alguma configuração definida em uma a subclasse, apenas a configuração na subclasse será usada.
- O `Controller::httpCodes()` foi removido, use o `Cake\Network\Response::httpCodes()` no lugar.
- O `Controller::disableCache()` foi removido, use o `Cake\Network\Response::disableCache()` no lugar.
- O `Controller::flash()` foi removido. Esse método era raramente usado em aplicações reais e não tinha mais propósito algum.
- O `Controller::validate()` e `Controller::validationErrors()` foram removidos. Eles eram restos dos dias do 1.x onde as preocupações com os models + controllers eram muito mais entrelaçados.
- O `Controller::loadModel()` agora carrega uma tabela de objetos.
- A propriedade `Controller::$scaffold` foi removida. O scaffolding dinâmico foi removido do núcleo do CakePHP. Um plugin de scaffolding melhorado, chamado CRUD, pode ser encontrado em: https://github.com/FriendsOfCake/crud
- A propriedade `Controller::$ext` foi removida. Você deve agora estender e sobrescrever a propriedade `View::$_ext` se você deseja usar uma extensão de arquivo de visão não padrão.
- A propriedade `Controller::$methods` foi removida. Você deve usar o `Controller::isAction()` para determinar quando ou não um nome de método é uma ação. Essa mudança foi feita para permitir personalizações mais fáceis do que vai contar ou não como uma ação.
- A propriedade `Controller::$Components` foi removida e substituída pelo `_components`. Se você precisar carregar componentes em tempo de execução você deve usar o `$this->loadComponent()` em seu controller.
- A assinatura do `Cake\Controller\Controller::redirect()` mudou para `Controller::redirect(string|array $url, int $status = null)`. O terceiro argumento `$exit` foi removido. O método não pode mais enviar resposta e sair do script, no lugar ele retorna uma instância de `Response` com os cabeçalhos apropriados definidos.
- As propriedades mágicas `base`, `webroot`, `here`, `data`, `action`, e `params` foram removidas. Você deve acessar todas essas propriedades em `$this->request` no lugar.
- Métodos de controlar prefixados com sublinhado como `_someMethod()` não são mais tratados como métodos privados. Use as palavras chaves de visibilidade apropriadas no lugar. Somente métodos públicos podem ser usados como ação de controllers.

Scaffold Removido O scaffolding dinâmico no CakePHP foi removido do núcleo do CakePHP. Ele não era usado com frequência, e não era voltado para uso em produção. Um plugin melhorado de scaffolding, chamado CRUD, pode ser encontrado em: https://github.com/FriendsOfCake/crud

ComponentCollection Substituído Essa classe foi renomeada para `Cake\Controller\ComponentRegistry`. Veja a seção em *Objetos de Registro* para mais informações sobre as funcionalidades fornecidas pela nova classe. Você pode usar o `cake upgrade rename_collections` para ajudar você a atualizar o seu código.

Components

- A propriedade `_Collection` é agora `_registry`. Ela contém uma instância do `Cake\Controller\ComponentRegistry` agora.
- Todos components devem agora usar o método `config()` para obter/definir configurações.
- A configuração padrão para components deve ser definido na propriedade `$_defaultConfig`. Essa propriedade é automaticamente mesclada com qualquer configuração fornecida pelo construtor.
- Opções de configuração não são mais definidas como propriedades públicas.
- O método `Component::initialize()` não é mais um `event listener` (ouvinte de eventos). Ao invés disso, ele é um gancho pós-construtor como o `Table::initialize()` e `Controller::initialize()`. O novo método `Component::beforeFilter()` é ligado ao mesmo evento que o `Component::initialize()` costumava ser. O método de inicialização deve ter a seguinte assinatura `initialize(array $config)`.

Controller\Components

CookieComponent

- Ele usa o `Cake\Network\Request::cookie()` para ler os dados de cookies, isso facilita os testes, e permite o ControllerTestCase definir os cookies.
- Os Cookies encriptados pelas versões anteriores do CakePHP usando o método `cipher()`, agora não podem ser lidos, pois o `Security::cipher()` foi removido. Você precisará reencriptar os cookies com o método `rijndael()` ou `aes()` antes de atualizar.
- O `CookieComponent::type()` foi removido e substituído com dados de configuração acessados através de `config()`.
- O `write()` não aceita mais os parâmetros `encryption` ou `expires`. Os dois agora são gerenciados através de dados de configuração. Veja *CookieComponent* para mais informações.
- O caminho padrão para os cookies agora é o caminho base da aplicação, ao invés de "/".

AuthComponent

- O `Default` é agora o hasher de senhas padrão usado pelas classes de autenticação. Ele usa exclusivamente o algoritmo de hash bcrypt. Se você desejar continuar usando o hash SHA1 usado no 2.x, use `'passwordHasher' => 'Weak'` nas configurações de seu autenticador.
- O novo `FallbackPasswordHasher` foi adicionado para ajudar os usuários migrar senhas antigas de um algoritmo para o outro. Veja a documentação do AuthComponent para mais informações.
- A classe `BlowfishAuthenticate` foi removida. Apenas use `FormAuthenticate`.
- A classe `BlowfishPasswordHasher` foi removida. Use o `DefaultPasswordHasher` no lugar.
- O método `loggedIn()` foi removido. Use o `user()` no lugar.
- As opções de configuração não são mais definidas como propriedades públicas.
- Os métodos `allow()` e `deny()` não aceitam mais "var args". Todos os nomes de métodos precisam ser passados como primeiro argumento, seja como string ou array de strings.
- O método `login()` foi removido e substituído por `setUser()`. Para logar um usuário agora você deve chamar `identify()` que retorna as informações do usuário caso identificado com sucesso e então usar `setUser()` para salvar as informações na sessão de maneira persistente entre as requisições.
- O `BaseAuthenticate::_password()` foi removido. Use a classe `PasswordHasher` no lugar.
- O `BaseAuthenticate::logout()` foi removido.
- O `AuthComponent` agora dispara dois eventos `Auth.afterIdentify` e `Auth.logout` após um usuário ser identificado e antes de um usuário ser deslogado respectivamente. Você pode definir funções de callback para esses eventos retornando um array mapeado no método `implementedEvents()` de sua classe de autenticação.

Classes relacionadas a ACL foram movidas para um plugin separado. Hashers de senha, fornecedores de Autenticação e Autorização foram movidos para o namespace `\Cake\Auth`. Você DEVE mover seus fornecedores e hashers para o namespace `App\Auth` também.

RequestHandlerComponent

- Os seguintes métodos foram removidos do componente RequestHandler: `isAjax()`, `isFlash()`, `isSSL()`, `isPut()`, `isPost()`, `isGet()`, `isDelete()`. Use o método `Cake\Network\Request::is()` no lugar com o argumento relevante.
- O `RequestHandler::setContent()` foi removido, use `Cake\Network\Response::type()` no lugar.
- O `RequestHandler::getReferer()` foi removido, use `Cake\Network\Request::referer()` no lugar.
- O `RequestHandler::getClientIP()` foi removido, use `Cake\Network\Request::clientIp()` no lugar.
- O `RequestHandler::getAjaxVersion()` foi removido.

- O `RequestHandler::mapType()` foi removido, use `Cake\Network\Response::mapType()` no lugar.
- As opções de configuração não são mais definidas como propriedades públicas.

SecurityComponent

- Os seguintes métodos e as propriedades relacionadas foram removidas do componente Security: `requirePost()`, `requireGet()`, `requirePut()`, `requireDelete()`. Use o `Cake\Network\Request::allowMethod()` no lugar.
- `SecurityComponent::$disabledFields()` foi removido, use o `SecurityComponent::$unlockedFields()`.
- As funções relacionadas ao CSRF no SecurityComponent foram extraídas e movidas em separado no CsrfComponent. Isso permite que você use a proteção CSRF facilmente sem ter que usar prevenção de adulteração de formulários.
- As opções de configuração não são mais definidas como propriedades públicas.
- Os métodos `requireAuth()` e `requireSecure()` não aceitam mais "var args". Todos os nomes de métodos precisam ser passados como primeiro argumento, seja como string ou array de strings.

SessionComponent

- O `SessionComponent::setFlash()` está obsoleto. Você deve usar o *Flash* no lugar.

Error ExceptionRenderers personalizados agora espera-se que retornem ou um objeto `Cake\Network\Response` ou uma string quando renderizando erros. Isso significa que qualquer método que manipule exceções específicas devem retornar uma resposta ou valor de string.

Model A camada de model do 2.x foi completamente reescrita e substituída. Você deve revisar o *Guia de atualização para o novo ORM* para saber como usar o novo ORM.

- A classe `Model` foi removida.
- A classe `BehaviorCollection` foi removida.
- A classe `DboSource` foi removida.
- A classe `Datasource` foi removida.
- As várias classes de fonte de dados foram removidas.

ConnectionManager

- O ConnectionManager (gerenciador de conexão) foi movido para o namespace `Cake\Datasource`.
- O ConnectionManager teve os seguintes métodos removidos:

  - `sourceList`
  - `getSourceName`
  - `loadDataSource`
  - `enumConnectionObjects`
- O `Database\ConnectionManager::config()` foi adicionado e é agora o único jeito de configurar conexões.
- O `Database\ConnectionManager::get()` foi adicionado. Ele substitui o `getDataSource()`.
- O `Database\ConnectionManager::configured()` foi adicionado. Ele junto com `config()` substitui o `sourceList()` e `enumConnectionObjects()` com uma API mais padrão e consistente.
- O `ConnectionManager::create()` foi removido. Ele pode ser substituído por `config($name, $config)` e `get($name)`.

Behaviors

- Os métodos de comportamentos (behaviors) prefixados com sublinhado como `_someMethod()` não são mais tratados como métodos privados. Use as palavras chaves de visibilidade.

TreeBehavior O TreeBehavior foi completamente reescrito para usar o novo ORM. Embora ele funcione do mesmo modo que no 2.x, alguns métodos foram renomeados ou removidos:

- `TreeBehavior::children()` é agora uma busca personalizada `find('children')`.
- `TreeBehavior::generateTreeList()` é agora uma busca personalizada `find('treeList')`.
- `TreeBehavior::getParentNode()` foi removido.
- `TreeBehavior::getPath()` é agora uma busca personalizada `find('path')`.
- `TreeBehavior::reorder()` foi removido.
- `TreeBehavior::verify()` foi removido.

Suíte de Testes

Casos de Teste

- O `_normalizePath()` foi adicionado para permitir testes de comparação de caminhos para executar em todos os sistemas operacionais, independente de sua configuração (`\` no Windows vs `/` no UNIX, por exemplo).

Os seguintes métodos de asserção foram removidos já que eles estavam há muito obsoletos e foram substituídos pelo seu equivalente no PHPUnit:

- `assertEqual()` é substituído por `assertEquals()`

- `assertNotEqual()` é substituído por `assertNotEquals()`
- `assertIdentical()` é substituído por `assertSame()`
- `assertNotIdentical()` é substituído por `assertNotSame()`
- `assertPattern()` é substituído por `assertRegExp()`
- `assertNoPattern()` é substituído por `assertNotRegExp()`
- `assertReference()` é substituído por `assertSame()`
- `assertIsA()` é substituído por `assertInstanceOf()`

Note que alguns métodos tiveram a ordem dos argumentos trocada, ex. `assertEqual($is, $expected)` deve ser agora `assertEquals($expected, $is)`.

Os seguintes métodos de asserção estão obsoletos e serão removidos no futuro:

- `assertWithinMargin()` é substituído por `assertWithinRange()`
- `assertTags()` é substituído por `assertHtml()`

Em ambas as substituições dos métodos também mudaram a ordem dos argumentos para manter a consistência na API com `$expected` como primeiro argumento.

Os seguintes métodos de asserção foram adicionados:

- `assertNotWithinRange()` em contrapartida ao `assertWithinRange()`

View

Temas são agora Plugins Básicos Ter os temas e plugins de modo a criar components modulares da aplicação se provou limitado e confuso. No CakePHP 3.0, temas não residem mais **dentro** da aplicação. Ao invés disso, eles são plugins independentes. Isso resolveu alguns problemas com temas:

- Você não podia colocar temas *nos* plugins.
- Temas não podiam fornecer helpers (helpers), ou classes de visão personalizadas.

Esses dois problemas foram resolvidos ao converter os temas em plugins.

Pasta das views renomeada As pastas contendo os arquivos de views agora ficam em **src/Template** no lugar de **src/View**. Isso foi feito para separar os arquivos de visão dos arquivos contendo classes php. (ex. helpers, Classes de visão).

As seguintes pastas de Visão foram renomeadas para evitar colisão de nomes com nomes de controllers:

- `Layouts` agora é `Layout`
- `Elements` agora é `Element`
- `Errors` agora é `Error`
- `Emails` agora é `Email` (o mesmo para `Email` dentro de `Layout`)

Coleção de Helpers Substituída Essa classe foi renomeada para `Cake\View\HelperRegistry`. Veja a seção em *Objetos de Registro* para mais informações sobre as funcionalidades fornecidas pela nova classe. Você pode usar o `cake upgrade rename_collections` para ajudar você a atualizar seu código.

Classe View

- A chave `plugin` foi removida do argumento `$options` de `Cake\View\View::element()`. Especifique o nome do elemento como `AlgumPlugin.nome_do_elemento` no lugar.
- O `View::getVar()` foi removido, use o `Cake\View\View::get()` no lugar.
- O `View::$ext` foi removido e no lugar uma propriedade protegida `View::$_ext` foi adicionada.
- O `View::addScript()` foi removido. Use o *Using View Blocks* no lugar.
- As propriedades mágicas `base`, `webroot`, `here`, `data`, `action`, e `params` foram removidas. Ao invés disso, você deve acessar todas essas propriedades no `$this->request`.
- O `View::start()` não se liga mais a um bloco existente. Ao invés disso ele irá sobrescrever o conteúdo do bloco quando o `end()` for chamado. Se você precisa combinar o conteúdo de um bloco você deverá buscar o conteúdo do bloco quando chamar o start uma segunda vez, ou usar o modo de captura de `append()`.
- O `View::prepend()` não tem mais um modo de captura.
- O `View::startIfEmpty()` foi removido. Agora que o start() sempre sobrescreve, o startIfEmpty não tem mais propósito.
- A propriedade `View::$Helpers` foi removida e substituída com `_helpers`. Se você precisar carregar helpers em tempo de execução você deve usar o `$this->addHelper()` em seus arquivos de visão.
- O `View` agora irá lançar `Cake\View\Exception\MissingTemplateException` quando templates estiverem faltando, ao invés de `MissingViewException`.

ViewBlock

- O `ViewBlock::append()` foi removido, use o `Cake\ViewViewBlock::concat()` no lugar. Entretanto o `View::append()` ainda existe.

JsonView

- Agora os dados JSON terão as entidades HTML codificadas por padrão. Isso previne possíveis problemas de XSS quando o conteúdo de visão JSON está encapsulado em arquivos HTML.
- O `Cake\View\JsonView` agora suporta a variável de visão `_jsonOptions`. Isso permite a você configurar as opções de máscara de bits usadas ao gerar JSON.

XmlView

- A `Cake\View\XmlView` agora suporta a variável de visão `_xmlOptions`. Isso permite a você configurar as opções usadas quando gerar XML.

View\Helper

- A propriedade `$settings` é agora chamada `$_config` e deve ser acessada através do método `config()`.
- As opções de configuração não são mais definidas como propriedades públicas.
- O `Helper::clean()` foi removido. Ele nunca foi robusto o suficiente para prevenir completamente XSS. Ao invés disso você deve escapar o conteúdo com `h` ou ou usar uma biblioteca dedicada como o htmlPurifier.
- O `Helper::output()` foi removido. Esse método estava obsoleto no 2.x.
- Os métodos `Helper::webroot()`, `Helper::url()`, `Helper::assetUrl()`, `Helper::assetTimestamp()` foram movidos para o novo ajudante `Cake\View\Helper\UrlHelper`. O `Helper::url()` está agora disponível como `Cake\View\Helper\UrlHelper::build()`.
- Os Assessores Mágicos a propriedades obsoletas foram removidos. A seguinte propriedade agora deve ser acessada a partir do objeto de requisição:
  - base
  - here
  - webroot
  - data
  - action
  - params

Helpers A classe Helper teve os seguintes métodos removidos:

- `Helper::setEntity()`
- `Helper::entity()`
- `Helper::model()`
- `Helper::field()`
- `Helper::value()`
- `Helper::_name()`
- `Helper::_initInputField()`
- `Helper::_selectedArray()`

Esses métodos eram partes usadas apenas pelo FormHelper, e parte de uma funcionalidade de persistência de campos que se mostrou problemática com o tempo. O FormHelper não precisa mais destes métodos e a complexidades que eles provêm não é mais necessária.

Os seguintes métodos foram removidos:

- `Helper::_parseAttributes()`
- `Helper::_formatAttribute()`

Esses métodos podem agora ser encontrados na classe `StringTemplate` que os helpers usam com frequência. Veja o `StringTemplateTrait` para um jeito fácil de integrar os templates de string em seus próprios helpers.

FormHelper O FormHelper foi completamente reescrito para o 3.0. Ele teve algumas grandes mudanças:

- O FormHelper trabalha junto com o novo ORM. Mas também possui um sistema extensível para integrar com outros ORMs e fontes de dados.
- O FormHelper possui um sistema de widgets extensível que permite a você criar novos widgets de entrada personalizados e expandir facilmente aqueles inclusos no framework.
- Os Templates de String são a fundação deste ajudante. Ao invés de encher de arrays por toda parte, a maioria do HTML que o FormHelper gera pode ser personalizado em um lugar central usando conjuntos de templates.

Além dessas grandes mudanças, foram feitas algumas mudanças menores que causaram rompendo algumas coisas da versão anterior. Essas mudanças devem simplificar o HTML que o FormHelper gera e reduzir os problemas que as pessoas tinham no passado:

- O prefixo `data[` foi removido de todas as entradas geradas. O prefixo não tem mais propósito.
- Os vários métodos de entradas independentes, como `text()`, `select()` e outros, não geram mais atributos id.
- A opção `inputDefaults` foi removida de `create()`.
- As opções `default` e `onsubmit` do `create()` foram removidas. No lugar você deve usar JavaScript event binding ou definir todos os códigos js necessários para o `onsubmit`.
- O `end()` não gerará mais botões. Você deve criar botões com `button()` ou `submit()`.
- O `FormHelper::tagIsInvalid()` foi removido. Use `isFieldError()` no lugar.
- O `FormHelper::inputDefaults()` foi removido. Você pode usar `templates()` para definir/expandir os templates que o FormHelper usa.
- As opções `wrap` e `class` foram removidas do método `error()`.
- A opção `showParents` foi removida do select().
- As opções `div`, `before`, `after`, `between` e `errorMessage` foram removidas do `input()`. Você pode usar templates para atualizar o HTML envoltório. A opção `templates` permite você sobrescrever os templates carregados para uma entrada.

- As opções `separator`, `between`, e `legend` foram removidas do `radio()`. Você pode usar templates para mudar o HTML envoltório agora.
- O parâmetro `format24Hours` foi removido de `hour()`. Ele foi substituído pela opção `format`.
- Os parâmetros `minYear` e `maxYear` foram removidos do `year()`. Ambos podem ser fornecidos como opções.
- Os parâmetros `dateFormat` e `timeFormat` foram removidos do `datetime()`. Você pode usar o template para definir a ordem que as entradas devem ser exibidas.
- O `submit()` teve as opções `div`, `before` e `after` removidas. Você pode personalizar o template `submitContainer` para modificar esse conteúdo.
- O método `inputs()` não aceita mais `legend` e `fieldset` no parâmetro `$fields`, você deve usar o parâmetro `$options`. Ele também exige que o parâmetro `$fields` seja um array. O parâmetro `$blacklist` foi removido, a funcionalidade foi substituída pela especificação de `'field' => false` no parâmetro `$fields`.
- O parâmetro `inline` foi removido do método postLink(). Você deve usar a opção `block` no lugar. Definindo `block => true` irá emular o comportamento anterior.
- O parâmetro `timeFormat` para `hour()`, `time()` e `dateTime()` agora é 24 por padrão, em cumprimento ao ISO 8601.
- O argumento `$confirmMessage` de `Cake\View\Helper\FormHelper::postLink()` foi removido. Você deve usar agora a chave `confirm` no `$options` para especificar a mensagem.
- As entradas do tipo Checkbox e radio são agora renderizadas *dentro* de elementos do tipo label por padrão. Isso ajuda a aumentar a compatibilidade com bibliotecas CSS populares como Bootstrap[12] e Foundation[13].
- As tags de template agora são todas camelBacked (primeira letra minúscula e inicio de novas palavras em maiúsculo). As tags pré-3.0 `formstart`, `formend`, `hiddenblock` e `inputsubmit` são agora `formStart`, `formEnd`, `hiddenBlock` e `inputSubmit`. Certifique-se de altera-las se elas estiverem personalizando sua aplicação.

É recomendado que você revise a documentação *Form* para mais detalhes sobre como usar o FormHelper no 3.0.

HtmlHelper

- O `HtmlHelper::useTag()` foi removido, use `tag()` no lugar.
- O `HtmlHelper::loadConfig()` foi removido. As tags podem ser personalizadas usando `templates()` ou as configurações de `templates`.
- O segundo parâmetro `$options` para `HtmlHelper::css()` agora sempre irá exigir um array.
- O primeiro parâmetro `$data` para `HtmlHelper::style()` agora sempre irá exigir um array.

[12] http://getbootstrap.com/
[13] http://foundation.zurb.com/

- O parâmetro `inline` foi removido dos métodos meta(), css(), script() e scriptBlock(). Ao invés disso, você deve usar a opção `block`. Definindo `block => true` irá emular o comportamento anterior.
- O `HtmlHelper::meta()` agora exige que o `$type` seja uma string. Opções adicionais podem ser passadas como `$options`.
- O `HtmlHelper::nestedList()` agora exige que o `$options` seja um array. O quarto argumento para o tipo tag foi removido e incluido no array `$options`.
- O argumento `$confirmMessage` de `Cake\View\Helper\HtmlHelper::link()` foi removido. Você deve usar agora a chave `confirm` no `$options` para especificar a menssagem.

PaginatorHelper

- O `link()` foi removido. Ele não era mais usado internamente pelo ajudante. Ele era pouco usado em códigos de usuários e não se encaixava mais nos objetivos do ajudante.
- O `next()` não tem mais as opções 'class' ou 'tag'. Ele não tem mais argumentos desabilitados. Ao invés disso são usados templates.
- O `prev()` não tem mais as opções 'class' ou 'tag'. Ele não tem mais argumentos desabilitados. Ao invés disso são usados templates.
- O `first()` não tem mais as opções 'after', 'ellipsis', 'separator', 'class' ou 'tag'.
- O `last()` não tem mais as opções 'after', 'ellipsis', 'separator', 'class' ou 'tag'.
- O `numbers()` não tem mais as opções 'separator', 'tag', 'currentTag', 'currentClass', 'class', 'tag' e 'ellipsis'. Essas opções são agora facilitadas pelos templates. Ele também exige que agora o parâmetro `$options` seja um array.
- O espaço reservado de estilo `%page%` foi removido de `Cake\View\Helper\PaginatorHelper::counter()`. Use o espaço reservado de estilo `{{page}}` no lugar.
- O `url()` foi renomeada para `generateUrl()` para evitar colisão de declaração de método com `Helper::url()`.

Por padrão todos os links e textos inativos são encapsulados em elementos `<li>`. Isso ajuda a fazer o CSS mais fácil de escrever, e aumenta a compatibilidade com frameworks de CSS populares.

Ao invés de várias opções em cada método, você deve usar a funcionalidade de templates. Veja a documentação *PaginatorHelper Templates* para informações de como se usar templates.

TimeHelper

- `TimeHelper::__set()`, `TimeHelper::__get()`, e `TimeHelper::__isset()` foram removidos. Eles eram métodos mágicos para atributos obsoletos.
- O `TimeHelper::serverOffset()` foi removido. Ele provia práticas incorretas de operações com tempo.
- O `TimeHelper::niceShort()` foi removido.

NumberHelper

- O `NumberHelper::format()` agora exige que `$options` seja um array.

SessionHelper

- O `SessionHelper` está obsoleto. Você pode usar `$this->request->session()` diretamente, e a funcionalidade de mensagens flash foi movida para *Flash*.

JsHelper

- O `JsHelper` e todos motores associados foram removidos. Ele podia gerar somente um subconjunto muito pequeno de códigos JavaScript para biblioteca selecionada e consequentemente tentar gerar todo código JavaScript usando apenas o ajudante se tornava um impedimento com frequência. É recomendado usar diretamente sua biblioteca JavaScript preferida.

CacheHelper Removido O CacheHelper foi removido. A funcionalidade de cache que ele fornecia não era padrão, limitada e incompatível com layouts não-HTML e views de dados. Essas limitações significavam que uma reconstrução completa era necessária. O ESI (Edge Side Includes) se tornou uma maneira padronizada para implementar a funcionalidade que o CacheHelper costumava fornecer. Entretanto, implementando Edge Side Includes[14] em PHP tem várias limitações e casos. Ao invés de construir uma solução ruim, é recomendado que os desenvolvedores que precisem de cache de resposta completa use o Varnish[15] ou Squid[16] no lugar.

I18n O subsistema de internacionalização foi completamente reescrito. Em geral, você pode esperar o mesmo comportamento que nas versões anteriores, especialmente se você está usando a família de funções `__()`.

Internamente, a classe `I18n` usa `Aura\Intl`, e métodos apropriados são expostos para dar acesso a funções específicas da biblioteca. Por esta razão a maior parte dos métodos dentro de `I18n` foram removidos ou renomeados.

Devido ao uso do `ext/intl`, a classe L10n foi removida completamente. Ela fornecia dados incompletos e desatualizados em comparação com os dados disponíveis na classe `Locale` do PHP.

O idioma padrão da aplicação não será mais alterado automaticamente pelos idiomas aceitos pelo navegador nem por ter o valor `Config.language` definido na sessão do navegador. Você pode, entretanto, usar um filtro no despachante para trocar o idioma automaticamente a partir do cabeçalho `Accept-Language` enviado pelo navegador:

```
// No config/bootstrap.php
DispatcherFactory::addFilter('LocaleSelector');
```

Não há nenhum substituto incluso para selecionar automaticamente o idioma a partir de um valor configurado na sessão do usuário.

[14] http://en.wikipedia.org/wiki/Edge_Side_Includes
[15] http://varnish-cache.org
[16] http://squid-cache.org

A função padrão para formatação de mensagens traduzidas não é mais a `sprintf`, mas a mais avançada e funcional classe `MessageFormatter`. Em geral você pode reescrever os espaços reservados nas mensagens como segue:

```
// Antes:
__('Hoje é um dia %s na %s', 'Ensolarado', 'Espanha');

// Depois:
__('Hoje é um dia {0} na {1}', 'Ensolarado', 'Espanha');
```

Você pode evitar ter de reescrever suas mensagens usando o antigo formatador `sprintf`:

```
I18n::defaultFormatter('sprintf');
```

Adicionalmente, o valor `Config.language` foi removido e ele não pode mais ser usado para controlar o idioma atual da aplicação. Ao invés disso, você pode usar a classe `I18n`:

```
// Antes
Configure::write('Config.language', 'fr_FR');

// Agora
I18n::locale('en_US');
```

- Os métodos abaixo foram movidos:
  - De `Cake\I18n\Multibyte::utf8()` para `Cake\Utility\Text::utf8()`
  - De `Cake\I18n\Multibyte::ascii()` para `Cake\Utility\Text::ascii()`
  - De `Cake\I18n\Multibyte::checkMultibyte()` para `Cake\Utility\Text::isMultibyte()`
- Como agora o CakePHP requer a extensão mbstring, a classe `Multibyte` foi removida.
- As mensagens de erro por todo o CakePHP não passam mais através das funções de internacionalização. Isso foi feito para simplificar o núcleo do CakePHP e reduzir a sobrecarga. As mensagens apresentadas aos desenvolvedores são raramente, isso quando, são de fato traduzidas - de modo que essa sobrecarga adicional trás pouco beneficio.

Localização

- Agora o construtor de `Cake\I18n\L10n` recebe uma instância de `Cake\Network\Request` como argumento.

Testes

- O `TestShell` foi removido. O CakePHP, o esqueleto da aplicação e novos plugins "cozinhados", todos usam o `phpunit` para rodar os testes.
- O webrunner (webroot/test.php) foi removido. A adoção do CLI aumentou grandemente desde o release inicial do 2.x. Adicionalmente, os CLI de execução oferecem integração superior com IDE's e outras ferramentas automáticas.

Se você sentir necessidade de um jeito de executar os testes a partir de um navegador, você deve verificar o VisualPHPUnit[17]. Ele oferece muitas funcionalidades adicionais que o antigo webrunner.

- O `ControllerTestCase` está obsoleto e será removido no CakePHP 3.0.0. Ao invés disso, você deve usar a nova funcionalidade *Controller Integration Testing*.
- As Fixtures devem agora ser referenciadas usando sua forma no plural:

```
// No lugar de
$fixtures = ['app.artigo'];

// Você deve usar
$fixtures = ['app.artigos'];
```

Utilitários

Classe Set Removida A classe Set foi removida, agora você deve usar a classe Hash no lugar dela.

Pastas & Arquivos As classes de pastas e arquivos foram renomeadas:

- O `Cake\Utility\File` foi renomeado para `Cake\Filesystem\File`
- O `Cake\Utility\Folder` foi renomeado para `Cake\Filesystem\Folder`

Inflexão

- O valor padrão para o argumento `$replacement` do `Cake\Utility\Inflector::slug()` foi alterado do sublinhado (_) para o traço (-). Usando traços para separar palavras nas URLs é a escolha popular e também recomendada pelo Google.
- As transliterações para `Cake\Utility\Inflector::slug()` foram alteradas. Se você usa transliterações personalizadas você terá que atualizar seu código. No lugar de expressões regulares, as transliterações usam simples substituições de string. Isso rendeu melhorias de performance significativas:

```
// No lugar de
Inflector::rules('transliteration', [
    '/ä|æ/' => 'ae',
    '/å/' => 'aa'
]);

// Você deve usar
Inflector::rules('transliteration', [
    'ä' => 'ae',
    'æ' => 'ae',
    'å' => 'aa'
]);
```

[17] https://github.com/NSinopoli/VisualPHPUnit

- Os conjuntos distintos de regras de não-inflexões e irregulares para pluralização e singularização foram removidos. No lugar agora temos uma lista comum para cada. Quando usar `Cake\Utility\Inflector::rules()` com o tipo 'singular' e 'plural' você não poderá mais usar chaves como 'uninflected' e 'irregular' no array de argumentos `$rules`.

  Você pode adicionar / sobrescrever a lista de regras de não-inflexionados e irregulares usando `Cake\Utility\Inflector::rules()` com valores 'uninflected' e 'irregular' para o argumento `$type`.

Sanitize

- A classe `Sanitize` foi removida.

Segurança

- O `Security::cipher()` foi removido. Ele era inseguro e promovia práticas ruins de criptografia. Você deve usar o `Security::encrypt()` no lugar.
- O valor de configuração `Security.cipherSeed` não é mais necessário. Com a remoção de `Security::cipher()` ele não tem utilidade.
- A retrocompatibilidade do `Cake\Utility\Security::rijndael()` para valores encriptados antes do CakePHP 2.3.1 foi removido. Você deve reencriptar os valores usando `Security::encrypt()` e uma versão recente do CakePHP 2.x antes de migrar.
- A habilidade para gerar um hash do tipo blowfish foi removido. Você não pode mais usar o tipo "blowfish" em `Security::hash()`. Deve ser usado apenas o *password_hash()* do PHP e *password_verify()* para gerar e verificar hashes blowfish. A compabilidade da biblioteca ircmaxell/password-compat[18] que é instalado junto com o CakePHP fornece essas funções para versões de PHP menor que 5.5.
- O OpenSSL é usado agora no lugar do mcrypt ao encriptar/desencriptar dados. Esse alteração fornece uma melhor performance e deixa o CakePHP a prova de futuros abandonos de suporte das distribuições ao mcrypt.
- O `Security::rijndael()` está obsoleto e apenas disponível quando se usa o mcrypt.

Aviso: Dados encriptados com Security::encrypt() em versões anteriores não são compatíveis com a implementação openssl. Você deve *definir a implementação como mcrypt* quando fizer atualização.

Data e Hora

- O `CakeTime` foi renomeado para `Cake\I18n\Time`.
- O `CakeTime::serverOffset()` foi removido. Ele provia práticas incorretas de operações com tempo.
- O `CakeTime::niceShort()` foi removido.
- O `CakeTime::convert()` foi removido.

[18] https://packagist.org/packages/ircmaxell/password-compat

- O `CakeTime::convertSpecifiers()` foi removido.
- O `CakeTime::dayAsSql()` foi removido.
- O `CakeTime::daysAsSql()` foi removido.
- O `CakeTime::fromString()` foi removido.
- O `CakeTime::gmt()` foi removido.
- O `CakeTime::toATOM()` foi renomeado para `toAtomString`.
- O `CakeTime::toRSS()` foi renomeado para `toRssString`.
- O `CakeTime::toUnix()` foi renomeado para `toUnixString`.
- O `CakeTime::wasYesterday()` foi renomeado para `isYesterday` para combinar com o resto da renomeação de métodos.
- O `CakeTime::format()` não usa mais o formato do `sprintf`, ao invés disso você deve usar o formato `i18nFormat`.
- O `Time::timeAgoInWords()` agora exige que o `$options` seja um array.

A classe Time não é mais uma coleção de métodos estáticos, ela estende o `DateTime` para herdar todos seus métodos e adicionar funções de formatação baseado em localização com ajuda da extensão `intl`.

Em geral, expressões assim:

```
CakeTime::aMethod($date);
```

Podem ser migradas reescrevendo para:

```
(new Time($date))->aMethod();
```

Números A biblioteca Number foi reescrita para usar internamente a classe `NumberFormatter`.

- O `CakeNumber` foi renomeada para `Cake\I18n\Number`.
- O `Number::format()` agora exige que o `$options` seja um array.
- O `Number::addFormat()` foi removido.
- O `Number::fromReadableSize()` foi movido para `Cake\Utility\Text::parseFileSize()`.

Validação

- A faixa de valores para `Validation::range()` agora é inclusiva se `$lower` e `$upper` forem fornecidos.
- O `Validation::ssn()` foi removido.

Xml

- O `Xml::build()` agora exige que o `$options` seja um array.

- O `Xml::build()` não aceita mais uma URL. Se você precisar criar um documento XML a partir de uma URL, use o

Guia de atualização para o novo ORM

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[19] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

Informações Gerais

Processo de desenvolvimento no CakePHP

Aqui tentamos explicar o processo utilizado no desenvolvimento com o framework CakePHP. Nós dependemos fortemente da interação por tickets e no canal do IRC. O IRC é o melhor lugar para encontrar membros do time de desenvolvimento[20] e discutir idéias, o ultimo código e fazer comentários gerais. Se algo mais formal tem que ser proposto ou exite um problema com uma versão, o sistema de tickets é o melhor lugar para compartilhar seus pensamentos.

Nós atualmente mantemos 4 versões do CakePHP.

- **versões tageadas** : Versões tageadas são destinadas para produção onde uma estabilidade maior é mais importante do que funcionalidades. Questões sobre versões tageadas serão resolvidas no branch relacionado e serão parte do próximo release.
- **branch principal** : Esses branches são onde todas as correções são fundidas. Versões estáveis são rotuladas apartir desses branches. `master` é o principal branch para a versão atual. `2.x` é o branch de manutenção para a versão 2.x. Se você está usando versões estáveis e precisa de correções que não chegaram em uma versão tageada olhe aqui.
- **desenvolvimento** : O branch de desenvolvimento contém sempre as ultimas correções e funcionalidades. Eles são nomeados pela versão a qual se destinam, ex: *3.next*. Uma vez que estas braches estão estáveis elas são fundidas na branch principal da versão.
- **branches de funcionalidades** : Branches de funcionalidade contém trabalhos que estão sendo desenvolvidos ou possivelmente instáveis e são recomendadas apenas para usuários avançados interessados e dispostos a contribuir com a comunidade. Branches de funcionalidade são nomeadas pela seguinte convenção *versão-funcionalidade*. Um exemplo seria *3.3-router* Que conteria novas funcionalidades para o Router na 3.3

Esperamos que isso te ajudará a entender que versão é correta pra você. Uma vez que escolhida a versão você pode se sentir compelido a reportar um erro ou fazer comentários gerais no código.

[19] https://github.com/cakephp/docs
[20] https://github.com/cakephp?tab=members

- Se você está usando uma versão estável ou de manutenção, por favor envie tickets ou discuta conosco no IRC.
- Se você está usando uma branch de desenvolvimento ou funcionalidade, o primeiro lugar para ir é o IRC. Se você tem um comentário e não consegue entrar no IRC depois de um ou dois dias, envie um ticket.

Se você encontrar um problema, a melhor resposta é escrever um teste. O melhor conselho que podemos oferecer em escrever testes é olhar nos que estão no núcleo do projeto.

E sempre, se você tiver alguma questão ou cometários, nos visite no #cakephp no irc.freenode.net

Glossário

Nota: A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no Github[21] ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

routing array An array of attributes that are passed to `Router::url()`. They typically look like:

```
['controller' => 'Posts', 'action' => 'view', 5]
```

HTML attributes An array of key => values that are composed into HTML attributes. For example:

```
// Given
['class' => 'my-class', 'target' => '_blank']

// Would generate
class="my-class" target="_blank"
```

If an option can be minimized or accepts its name as the value, then `true` can be used:

```
// Given
['checked' => true]

// Would generate
checked="checked"
```

plugin syntax Plugin syntax refers to the dot separated class name indicating classes are part of a plugin:

```
// The plugin is "DebugKit", and the class name is "Toolbar".
'DebugKit.Toolbar'

// The plugin is "AcmeCorp/Tools", and the class name is "Toolbar".
'AcmeCorp/Tools.Toolbar'
```

dot notation Dot notation defines an array path, by separating nested levels with `.` For example:

[21] https://github.com/cakephp/docs

```
Cache.default.engine
```

Would point to the following value:

```
[
    'Cache' => [
        'default' => [
            'engine' => 'File'
        ]
    ]
]
```

CSRF Cross Site Request Forgery. Prevents replay attacks, double submissions and forged requests from other domains.

CDN Content Delivery Network. A 3rd party vendor you can pay to help distribute your content to data centers around the world. This helps put your static assets closer to geographically distributed users.

routes.php A file in `config` directory that contains routing configuration. This file is included before each request is processed. It should connect all the routes your application needs so requests can be routed to the correct controller + action.

DRY Don't repeat yourself. Is a principle of software development aimed at reducing repetition of information of all kinds. In CakePHP DRY is used to allow you to code things once and re-use them across your application.

PaaS Platform as a Service. Platform as a Service providers will provide cloud based hosting, database and caching resources. Some popular providers include Heroku, EngineYard and PagodaBox

DSN Data Source Name. A connection string format that is formed like a URI. CakePHP supports DSN's for Cache, Database, Log and Email connections.

PHP Namespace Index

C

Índice

I

L

N

P

R

S

T

U

V