Configuração

Embora as convenções eliminem a necessidade de configurar todo o CakePHP, Você ainda precisará configurar algumas coisas, como suas credenciais de banco de dados por exemplo.

Além disso, há opções de configuração opcionais que permitem trocar valores padrão e implementações com as personalizadas para seu aplicativo.

Configurando sua Aplicação

A configuração é geralmente armazenada em arquivos PHP ou INI, e carregada durante a execução do código de inicialização. O CakePHP vem com um arquivo de configuração por padrão. Mas se necessário, você pode adicionar arquivos de configuração adicionais e carregá-los no código de inicialização do aplicativo. `Cake\Core\Configure` é usado para configuração global, e classes como `Cache` providenciam `config()` métodos para tornar a configuração simples e transparente.

Carregando Arquivos de Configurações Adicionais

Se sua aplicação tiver muitas opções de configuração, pode ser útil dividir a configuração em vários arquivos. Depois de criar cada um dos arquivos no seu **config/** diretório, você pode carregá-los em **bootstrap.php**:

```
use Cake\Core\Configure;
use Cake\Core\Configure\Engine\PhpConfig;

Configure::config('default', new PhpConfig());
Configure::load('app', 'default', false);
Configure::load('other_config', 'default');
```

Você também pode usar arquivos de configuração adicionais para fornecer sobreposições específicas do ambiente. Cada arquivo carregado após **app.php** pode redefinir valores previamente declarados permitindo que você personalize a configuração para ambientes de desenvolvimento ou de homologação.

Configuração Geral

Abaixo está uma descrição das variáveis e como elas afetam seu aplicativo CakePHP.

debug
: Altera a saída de depuração do CakePHP. `false` = Modo Produção. Não é exibido nenhuma mensagem de erro e/ou aviso. `true` = Modo de Desenvolvimento. É exibido todas as mensagens de erros e/ou avisos.

App.namespace
: O namespace em que as classes do aplicativo estão.

Ao alterar o namespace em sua configuração, você também precisará atualizar o arquivo ** composer.json ** para usar esse namespace também. Além disso, crie um novo carregador automático executando `php composer.phar dumpautoload`.

App.baseUrl
: Não comentar esta definição se você **não** planeja usar o mod_rewrite do Apache com o CakePHP. Não se esqueça de remover seus arquivos .htaccess também.

App.base
: O diretório base no qual o aplicativo reside. Se `false` isso será detectado automaticamente. Se não `false`, certifique-se de que sua seqüência de caracteres começa com um / e NÃO termina com um /. Por exemplo, */basedir* deve ser uma App.base válida. Caso contrário, o AuthComponent não funcionará corretamente.

App.encoding
: Defina a codificação que seu aplicativo usa. Essa codificação é usada para gerar o charset no layout e codificar entidades. Ele deve corresponder aos valores de codificação especificados para o seu banco de dados.

App.webroot
: O diretório raiz da aplicação web.

App.wwwRoot
: O diretório raiz dos arquivos da aplicação web.

App.fullBaseUrl
: O nome de domínio totalmente qualificado (incluindo o protocolo) para a raiz do aplicativo. Isso é usado ao gerar URLs absolutos. Por padrão, esse valor é gerado usando a variável $_SERVER. Entretanto, Você deve defini-lo manualmente para otimizar o desempenho ou se você está preocupado com as pessoas manipulando o cabeçalho do `Host`. Em um contexto CLI (do Shell) a *fullBaseUrl* não pode ser lido a partir de $_SERVER, como não há servidor envolvido. Você precisará especificá-lo se precisar gerar URLs de um shell (por exemplo, ao enviar e-mails).

App.imageBaseUrl
: O caminho da web para as imagens públicas na webroot da aplicação. Se você estiver usando um CDN, você deve definir este valor para a localização do CDN.

App.cssBaseUrl
: O caminho da web para os arquivos de estilos em cascata(**.css**) públicos na webroot da aplicação. Se você estiver usando um CDN, você deve definir este valor para a localização do CDN.

App.jsBaseUrl
: O caminho da web para os scripts (em JavaScript) públicos na webroot da aplicação. Se você estiver usando um CDN, você deve definir este valor para a localização do CDN.

App.paths
: Configurar caminhos para recursos não baseados em classe. Suporta as subchaves `plugins`, `templates`, `locales`, que permitem a definição de caminhos para plugins, templates e arquivos de locale respectivamente.

Security.salt
: Uma seqüência aleatória usada em hash. Uma seqüência aleatória usada em hash. Este valor também é usado como o sal HMAC ao fazer criptografia simétrica.

Asset.timestamp

Acrescenta um carimbo de data/hora que é a última hora modificada do arquivo específico no final dos URLs de arquivos de recurso (CSS, JavaScript, Image) ao usar assistentes adequados. Valores válidos:

- (bool) `false` - Não fazer nada (padrão)
- (bool) `true` - Acrescenta o carimbo de data/hora quando depuração é `true`
- (string) 'force' - Sempre anexa o carimbo de data/hora.

Configuração do banco de dados

Consulte Database Configuration para obter informações sobre como configurar suas conexões de banco de dados.

Configuração do Cache

Consulte Caching Configuration para obter informações sobre como configurar o cache no CakePHP.

Configuração de manipulação de erro e exceção

Consulte Error and Exception Configuration para obter informações sobre como configurar manipuladores de erro e exceção.

Configuração de log

Consulte Logging Configuration para obter informações sobre como configurar o log no CakePHP.

Configuração de e-mail

Consulte Email Configuration para obter informações sobre como configurar predefinições de e-mail no CakePHP.

Configuração de sessão

Consulte Session Configuration para obter informações sobre como configurar o tratamento de sessão no CakePHP.

Configuração de roteamento

Consulte Routes Configuration para obter mais informações sobre como configurar o roteamento e criar rotas para seu aplicativo.

Caminhos adicionais de classe

Caminhos de classe adicionais são configurados através dos carregadores automáticos usados pelo aplicativo. Ao usar o `Composer` para gerar o seu arquivo de autoload, você pode fazer o seguinte, para fornecer caminhos alternativos para controladores em seu aplicativo:

```
"autoload": {
    "psr-4": {
        "App\\Controller\\": "/path/to/directory/with/controller/folders",
        "App\": "src"
    }
}
```

O código acima seria configurar caminhos para o namespace `App` e `App\Controller`. A primeira chave será pesquisada e, se esse caminho não contiver a classe/arquivo, a segunda chave será pesquisada. Você também pode mapear um namespace único para vários diretórios com o seguinte código:

```
"autoload": {
    "psr-4": {
        "App\": ["src", "/path/to/directory"]
    }
}
```

Plugin, Modelos de Visualização e Caminhos Locais

Como os plug-ins, os modelos de visualização (Templates) e os caminhos locais (locales) não são classes, eles não podem ter um autoloader configurado. O CakePHP fornece três variáveis de configuração para configurar caminhos adicionais para esses recursos. No **config/app.php** você pode definir estas variáveis

```
return [
    // More configuration
    'App' => [
        'paths' => [
            'plugins' => [
                ROOT . DS . 'plugins' . DS,
                '/path/to/other/plugins/'
            ],
            'templates' => [
                APP . 'Template' . DS,
                APP . 'Template2' . DS
            ],
            'locales' => [
                APP . 'Locale' . DS
            ]
        ]
    ]
];
```

Caminhos devem terminar com um separador de diretório, ou eles não funcionarão corretamente.

Configuração de Inflexão

Consulte Configuração da inflexão para obter mais informações sobre como fazer a configuração de inflexão.

Configurar classe

class `Cake\Core\Configure`

A classe de Configuração do CakePHP pode ser usada para armazenar e recuperar valores específicos do aplicativo ou do tempo de execução. Tenha cuidado, pois essa classe permite que você armazene qualquer coisa nela, para que em seguida, usá-la em qualquer outra parte do seu código: Dando ma certa tentação de quebrar o padrão MVC do CakePHP. O objetivo principal da classe Configurar é manter variáveis centralizadas

que podem ser compartilhadas entre muitos objetos. Lembre-se de tentar viver por “convenção sobre a configuração” e você não vai acabar quebrando a estrutura MVC previamente definida.

Você pode acessar o `Configure` de qualquer lugar de seu aplicativo:

```
Configure::read('debug');
```

Escrevendo dados de configuração

static `Cake\Core\Configure::write`(*$key*, *$value*)

Use `write()` para armazenar dados na configuração do aplicativo:

```
Configure::write('Company.name', 'Pizza, Inc.');
Configure::write('Company.slogan','Pizza for your body and soul');
```

O dot notation usado no parâmetro `$key` pode ser usado para organizar suas configurações em grupos lógicos.

O exemplo acima também pode ser escrito em uma única chamada:

```
Configure::write('Company', [
    'name' => 'Pizza, Inc.',
    'slogan' => 'Pizza for your body and soul'
]);
```

Você pode usar `Configure::write('debug', $bool)` para alternar entre os modos de depuração e produção na mosca. Isso é especialmente útil para interações JSON onde informações de depuração podem causar problemas de análise.

Leitura de dados de configuração

static `Cake\Core\Configure::read`(*$key = null*)

Usado para ler dados de configuração da aplicação. Por padrão o valor de depuração do CakePHP é importante. Se for fornecida uma chave, os dados são retornados. Usando nossos exemplos de write() acima, podemos ler os dados de volta:

```
Configure::read('Company.name');    // Yields: 'Pizza, Inc.'
Configure::read('Company.slogan');  // Yields: 'Pizza for your body
                                    // and soul'

Configure::read('Company');

//Rendimentos:
['name' => 'Pizza, Inc.', 'slogan' => 'Pizza for your body and soul'];
```

Se $key for deixada nula, todos os valores em Configure serão retornados.

static `Cake\Core\Configure::readOrFail`(*$key*)

Lê dados de configuração como `Cake\Core\Configure::read`, mas espera encontrar um par chave/valor. Caso o par solicitado não exista, a `RuntimeException` será lançada:

```
Configure::readOrFail('Company.name');    // Rendimentos: 'Pizza, Inc.'
Configure::readOrFail('Company.geolocation');  // Vai lançar uma exceção

Configure::readOrFail('Company');

// Rendimentos:
['name' => 'Pizza, Inc.', 'slogan' => 'Pizza for your body and soul'];
```

Novo na versão 3.1.7: `Configure::readOrFail()` Foi adicionado na versão 3.1.7

Verificar se os dados de configuração estão definidos

static `Cake\Core\Configure::check`(*$key*)

Usado para verificar se uma chave/caminho existe e tem valor não nulo:

```
$exists = Configure::check('Company.name');
```

Excluindo Dados de Configuração

static `Cake\Core\Configure::delete`(*$key*)

Usado para excluir informações da configuração da aplicação:

```
Configure::delete('Company.name');
```

Leitura e exclusão de dados de configuração

static `Cake\Core\Configure::consume`(*$key*)

Ler e excluir uma chave do Configure. Isso é útil quando você deseja combinar leitura e exclusão de valores em uma única operação.

Lendo e escreveendo arquivos de configuração

static `Cake\Core\Configure::config`(*$name*, *$engine*)

O CakePHP vem com dois mecanismos de arquivos de configuração embutidos. `Cake\Core\Configure\Engine\PhpConfig` é capaz de ler arquivos de configuração do PHP, no mesmo formato que o Configure tem lido historicamente. `Cake\Core\Configure\Engine\IniConfig` é capaz de ler os arquivos de configuração no formato ini(.ini). Consulte a documentação do PHP para obter mais informações sobre os detalhes dos arquivos ini. Para usar um mecanismo de configuração do núcleo, você precisará conectá-lo ao Configure usando `Configure::config()`:

```
use Cake\Core\Configure\Engine\PhpConfig;

// Ler os arquivos de configuração da configuração
```

```
Configure::config('default', new PhpConfig());

// Ler arquivos de configuração de outro diretório.
Configure::config('default', new PhpConfig('/path/to/your/config/files/'));
```

Você pode ter vários mecanismos anexados para Configure, cada um lendo diferentes tipos ou fontes de arquivos de configuração. Você pode interagir com os motores conectados usando alguns outros métodos em Configure. Para verificar quais aliases de motor estão conectados você pode usar `Configure::configured()`:

```
// Obter a matriz de aliases para os motores conectados.
Configure::configured();

// Verificar se um motor específico está ligado.
Configure::configured('default');
```

static `Cake\Core\Configure::drop`(*$name*)

Você também pode remover os motores conectados. `Configure::drop('default')` removeria o alias de mecanismo padrão. Quaisquer tentativas futuras de carregar arquivos de configuração com esse mecanismo falhariam:

```
Configure::drop('default');
```

Carregando arquivos de configurações

static `Cake\Core\Configure::load`(*$key*, *$config = 'default'*, *$merge = true*)

Depois de ter anexado um motor de configuração para o Configure, ficará disponível para poder carregar ficheiros de configuração:

```
// Load my_file.php using the 'default' engine object.
Configure::load('my_file', 'default');
```

Os arquivos de configuração que foram carregados mesclam seus dados com a configuração de tempo de execução existente no Configure. Isso permite que você sobrescreva e adicione novos valores à configuração de tempo de execução existente. Ao definir `$merge` para `true`, os valores nunca substituirão a configuração existente.

Criando ou modificando arquivos de configuração

static `Cake\Core\Configure::dump`(*$key*, *$config = 'default'*, *$keys = []*)

Despeja todos ou alguns dos dados que estão no Configure em um sistema de arquivos ou armazenamento suportado por um motor de configuração. O formato de serialização é decidido pelo mecanismo de configuração anexado como $config. Por exemplo, se o mecanismo 'padrão' é `Cake\Core\Configure\Engine\PhpConfig`, o arquivo gerado será um arquivo de configuração PHP carregável pelo `Cake\Core\Configure\Engine\PhpConfig`

Dado que o motor 'default' é uma instância do PhpConfig. Salve todos os dados em Configure no arquivo *my_config.php*:

```
Configure::dump('my_config', 'default');
```

Salvar somente a configuração de manipulação de erro:

```
Configure::dump('error', 'default', ['Error', 'Exception']);
```

`Configure::dump()` pode ser usado para modificar ou substituir arquivos de configuração que são legíveis com `Configure::load()`

Armazenando Configuração do Tempo de Execução

static `Cake\Core\Configure::store`(*$name*, *$cacheConfig = 'default'*, *$data = null*)

Você também pode armazenar valores de configuração de tempo de execução para uso em uma solicitação futura. Como o configure só lembra valores para a solicitação atual, você precisará armazenar qualquer informação de configuração modificada se você quiser usá-la em solicitações futuras:

```
// Armazena a configuração atual na chave 'user_1234' no cache 'default'.
Configure::store('user_1234', 'default');
```

Os dados de configuração armazenados são mantidos na configuração de cache nomeada. Consulte a documentação Caching para obter mais informações sobre o cache.

Restaurando a Configuração do Tempo de Execução

static `Cake\Core\Configure::restore`(*$name*, *$cacheConfig = 'default'*)

Depois de ter armazenado a configuração de tempo de execução, você provavelmente precisará restaurá-la para que você possa acessá-la novamente. `Configure::restore()` faz exatamente isso:

```
// Restaura a configuração do tempo de execução do cache.
Configure::restore('user_1234', 'default');
```

Ao restaurar informações de configuração, é importante restaurá-lo com a mesma chave e configuração de cache usada para armazená-lo. As informações restauradas são mescladas em cima da configuração de tempo de execução existente.

Criando seus próprios mecanismos de configuração

Como os mecanismos de configuração são uma parte extensível do CakePHP, você pode criar mecanismos de configuração em seu aplicativo e plugins. Os motores de configuração precisam de uma `Cake\Core\Configure\ConfigEngineInterface`. Esta interface define um método de leitura, como o único método necessário. Se você gosta de arquivos XML, você pode criar um motor de XML de configuração simples para sua aplicação:

```
// Em src/Configure/Engine/XmlConfig.php
namespace App\Configure\Engine;

use Cake\Core\Configure\ConfigEngineInterface;
```

```
use Cake\Utility\Xml;

class XmlConfig implements ConfigEngineInterface
{

    public function __construct($path = null)
    {
        if (!$path) {
            $path = CONFIG;
        }
        $this->_path = $path;
    }

    public function read($key)
    {
        $xml = Xml::build($this->_path . $key . '.xml');
        return Xml::toArray($xml);
    }

    public function dump($key, array $data)
    {
        // Code to dump data to file
    }
}
```

No seu **config/bootstrap.php** você poderia anexar este mecanismo e usá-lo:

```
use App\Configure\Engine\XmlConfig;

Configure::config('xml', new XmlConfig());
...

Configure::load('my_xml', 'xml');
```

O método `read()` de um mecanismo de configuração, deve retornar uma matriz das informações de configuração que o recurso chamado `$key` contém.

interface `Cake\Core\Configure\ConfigEngineInterface`

Define a interface usada pelas classes que lêem dados de configuração e armazenam-no em `Configure`

`Cake\Core\Configure\ConfigEngineInterface::read`(*$key*)

Parâmetros:
- **$key** (*string*) – O nome da chave ou identificador a carregar.

Esse método deve carregar/analisar os dados de configuração identificados pelo `$key` e retornar uma matriz de dados no arquivo.

`Cake\Core\Configure\ConfigEngineInterface::dump`(*$key*)

Parâmetros:
- **$key** (*string*) – O identificador para escrever.
- **$data** (*array*) – Os dados para despejo.

Esse método deve despejar/armazenar os dados de configuração fornecidos para uma chave identificada pelo `$key`.

Motores de Configuração Integrados

Arquivos de configuração do PHP

class `Cake\Core\Configure\Engine\PhpConfig`

Permite ler arquivos de configuração que são armazenados como arquivos simples do PHP. Você pode ler arquivos da configuração do aplicativo ou do plugin configs diretórios usando sintaxe plugin. Arquivos *devem* retornar uma matriz. Um exemplo de arquivo de configuração seria semelhante a:

```
return [
    'debug' => 0,
    'Security' => [
        'salt' => 'its-secret'
    ],
    'App' => [
        'namespace' => 'App'
    ]
];
```

Carregue seu arquivo de configuração personalizado inserindo o seguinte em **config/bootstrap.php**:

```
Configure::load('customConfig');
```

Arquivos de configuração Ini

class `Cake\Core\Configure\Engine\IniConfig`

Permite ler arquivos de configuração armazenados como arquivos .ini simples. Os arquivos ini devem ser compatíveis com a função `parse_ini_file()` do php e beneficiar das seguintes melhorias.

- Os valores separados por ponto são expandidos em arrays.
- Valores booleanos como ‘on’ e ‘off’ são convertidos em booleanos.

Um exemplo de arquivo ini seria semelhante a:

```
debug = 0

[Security]
salt = its-secret

[App]
namespace = App
```

O arquivo ini acima, resultaria nos mesmos dados de configuração final do exemplo PHP acima. As estruturas de matriz podem ser criadas através de valores separados por pontos ou por seções. As seções podem conter chaves separadas por pontos para um assentamento mais profundo.

Arquivos de configuração do Json

class `Cake\Core\Configure\Engine\JsonConfig`

Permite ler/descarregar arquivos de configuração armazenados como cadeias codificadas JSON em arquivos .json.

Um exemplo de arquivo JSON seria semelhante a:

```
{
    "debug": false,
    "App": {
        "namespace": "MyApp"
    },
    "Security": {
        "salt": "its-secret"
    }
}
```

Bootstrapping CakePHP

Se você tiver alguma necessidade de configuração adicional, adicione-a ao arquivo **config/bootstrap.php** do seu aplicativo. Este arquivo é incluído antes de cada solicitação, e o comando CLI.

Este arquivo é ideal para várias tarefas de bootstrapping comuns:

- Definir funções de conveniência.
- Declaração de constantes.
- Definição da configuração do cache.
- Definição da configuração de log.
- Carregando inflexões personalizadas.
- Carregando arquivos de configuração.

Pode ser tentador para colocar as funções de formatação lá, a fim de usá-los em seus controladores. Como você verá nas seções Controllers (Controladores) e Views (Visualização) há melhores maneiras de adicionar lógica personalizada à sua aplicação.

Application::bootstrap()

Além do arquivo **config/bootstrap.php** que deve ser usado para configurar preocupações de baixo nível do seu aplicativo, você também pode usar o método `Application::bootstrap()` para carregar/inicializar plugins, E anexar ouvintes de eventos globais:

```
// Em src/Application.php
namespace App;

use Cake\Core\Plugin;
use Cake\Http\BaseApplication;

class Application extends BaseApplication
{
    public function bootstrap()
    {
```

```
        // Chamar o pai para `require_once` config/bootstrap.php
        parent::bootstrap();

        Plugin::load('MyPlugin', ['bootstrap' => true, 'routes' => true]);
    }
}
```

Carregar plugins/eventos em `Application::bootstrap()` torna Controller Integration Testing mais fácil à medida que os eventos e rotas serão re-processados em cada método de teste.

Variáveis de Ambiente

Alguns dos provedores modernos de nuvem, como o Heroku, permitem definir variáveis de ambiente. Ao definir variáveis de ambiente, você pode configurar seu aplicativo CakePHP como um aplicativo 12factor. Seguir as instruções do aplicativo 12factor app instructions é uma boa maneira de criar um app sem estado e facilitar a implantação do seu aplicativo. Isso significa, por exemplo, que, se você precisar alterar seu banco de dados, você precisará modificar uma variável DATABASE_URL na sua configuração de host sem a necessidade de alterá-la em seu código-fonte.

Como você pode ver no seu **app.php**, as seguintes variáveis estão em uso:

- `DEBUG` (`0` ou``1``)
- `APP_ENCODING` (ie UTF-8)
- `APP_DEFAULT_LOCALE` (ie en_US)
- `SECURITY_SALT`
- `CACHE_DEFAULT_URL` (ie `File:///?prefix=myapp_&serialize=true&timeout=3600&path=../tmp/cache/`)
- `CACHE_CAKECORE_URL` (ie `File:///?prefix=myapp_cake_core_&serialize=true&timeout=3600&path=../tmp/cache/persistent/`)
- `CACHE_CAKEMODEL_URL` (ie `File:///?prefix=myapp_cake_model_&serialize=true&timeout=3600&path=../tmp/cache/models/`)
- `EMAIL_TRANSPORT_DEFAULT_URL` (ie `smtp://user:password@hostname:port?tls=null&client=null&timeout=30`)
- `DATABASE_URL` (ie `mysql://user:pass@db/my_app`)
- `DATABASE_TEST_URL` (ie `mysql://user:pass@db/test_my_app`)
- `LOG_DEBUG_URL` (ie `file:///?levels[]=notice&levels[]=info&levels[]=debug&file=debug&path=../logs/`)
- `LOG_ERROR_URL` (ie `file:///?levels[]=warning&levels[]=error&levels[]=critical&levels[]=alert&levels[]=emergency&file=error&path=../logs/`)

Como você pode ver nos exemplos, definimos algumas opções de configuração como DSN. Este é o caso de bancos de dados, logs, transporte de e-mail e configurações de cache.

Se as variáveis de ambiente não estiverem definidas no seu ambiente, o CakePHP usará os valores definidos no **app.php**. Você pode usar a biblioteca php-dotenv library para usar variáveis de ambiente em um desenvolvimento local. Consulte as instruções Leiame da biblioteca para obter mais informações.

Desabilitando tabelas genéricas

Embora a utilização de classes de tabela genéricas - também chamadas auto-tables - quando a criação rápida de novos aplicativos e modelos de cozimento é útil, a classe de tabela genérica pode tornar a depuração mais difícil em alguns cenários.

Você pode verificar se qualquer consulta foi emitida de uma classe de tabela genérica via DebugKit através do painel SQL no DebugKit. Se você ainda tiver problemas para diagnosticar um problema que pode ser causado por tabelas automáticas, você pode lançar uma exceção quando o CakePHP implícitamente usa um `Cake\ORM\Table` genérico em vez de sua classe concreta assim:

```
// No seu bootstrap.php
use Cake\Event\EventManager;
// Prior to 3.6 use Cake\Network\Exception\NotFoundException
use Cake\Http\Exception\InternalErrorException;

$isCakeBakeShellRunning = (PHP_SAPI === 'cli' && isset($argv[1]) && $argv[1] ===
'bake');
if (!$isCakeBakeShellRunning) {
    EventManager::instance()->on('Model.initialize', function($event) {
        $subject = $event->getSubject();
        if (get_class($subject === 'Cake\ORM\Table') {
            $msg = sprintf(
                'Missing table class or incorrect alias when registering table
class for database table %s.',
                $subject->getTable());
            throw new InternalErrorException($msg);
        }
    });
}
```