Laravel 3

公式ドキュメント

日本語翻訳版

翻訳：川瀬　裕久

Laravel 3 公式ドキュメント日本語版

Laravel 3 の公式ドキュメントを日本語に翻訳したものです。

Hirohisa Kawase

This book is for sale at http://leanpub.com/laravel-3-japanese

This version was published on 2014-02-06

Contents

Ĥ

νĦLaravel 3 ყYٲZP

äɔTˡPDF' Żҿϸưğ TˡΨ

LeanpubGऽЄGZٲʤTǎЄǽʤTGPZĤG 礦åЄʸäƤ IPAZЄƤTZƤऽIPAZYþGZ

Leanpubφ˙Ƥþ 夲 ΦƤƤĲƤȻɔǽऽ¿P

ƤLeanpubGoogle ɒ σ6ĤZˤʤäƤY CZg ˤʤTG 餬 σŪ6ȸäƤɤYю T˙Gɤþ ˤʤ ɵGTGoogle Gɒ Ƥ

ʤ ƤӄֲּʲG

- PDFDZʸ ɻ
- PDFgɻˤʤʤ
- URLɻMarkdownЄ URLåˤŤɻ
- ContentsToCPɒZɻʤ

CƤ6ʤG 礦 2ŦƤȻɒ

ZGȡ˙Pʤʏ˥YٲZσäPDFGɻYٲ

前書き

この書籍について

この電子書籍はLaravel 公式ドキュメント[1]を、個人的に翻訳したものです。公式ドキュメントはオリジナル英語版が Laravel の配布ファイルに含まれており、Laravel インストール後、/docs にアクセスすることで、閲覧することができます。

フレームワークを始め、新しいサービスを日本人が活用するには、日本語のドキュメントが必要です。Laravel の面白さと使いやすさに感動したため、急いで翻訳しました。翻訳したドキュメントは、以下のサイトで公開しております。Web で閲覧したい方は、どうぞ活用ください。もちろん、無料で閲覧していただけます。

- http://laravel.kore1server.com
- http://laravel-ja.phpfogapp.com
- http://laravel-ja.pagodabox.com

一番上のアドレスが、最新版となっております。以降のアドレスはコピーサイトです。他のサイトは最新版のサイトがダウンした場合のバックアップとして用意しました。

また、ローカルサーバーでこの日本語ドキュメントを利用したい方は、http://github.com/HiroKws/Laravel-base-32/zipball/original-cssから、日本語ドキュメントを含んだ、配布 zip をダウンロードしていただけます。内容につきましては、http://kore1server.com/laravel-tutorial/312-laravel-32-development-base-sampleをご覧ください。（メンテナンスの関係上、含めている日本語ドキュメントは常時最新版に保っていません。）

本来、Markdown により記述されており、HTML に変換し Web で表示するために書かれているドキュメントです。書籍としてそぐわない表現もあり、その部分に関しては多少書き換えました。

サポート

この書籍のサポートは Laravel 3 のドキュメントのサポート期間内とさせて頂きます。

現在リリースされている Laravel 3 のバージョンの開発期間中、リリースごとに原文のドキュメントが更新されます。原文の変更に合わせ、最新版を配布します。また、誤記や翻訳の修正、表示の改善は随時行います。

ちなみに、Laravel 4 のリリースは 2013 年 5 月です。

電子版を販売する Leanpub 社の更新通知システムを通じ、メールにて更新をお伝えします。その中のダウンロードリンクから無料で最新版を入手できます。

ダウンロードされる電子書籍のファイル名はいつも同じ名前になります。古いバージョンを保存しておきたい方は、上書きされないように管理してください。旧バージョンのダウンロードはできません。

---

[1] http://www.laravel.com/docs

ライセンス

Ｌａｒａｖｅｌとは

新しい軽量 PHP フレームワークの一つです。後発の利を生かし、様々なフレームワークから機能を取り入れています。コードが読みやすくなるように、設計されています。そのため、メンテナンスがしやすいフレームワークです。Laravel という名前は適当に創りだされた単語であり、特別の意味を持っていません。

Laravel が他のフレームワークとは毛色が異なる多くの特徴を持っているのは、開発者 Taylor Otwell 氏のバックボーンにあるのでしょう。

彼は、PHP による Web 開発の専門家でありませんでした。そのため、多くの PHP フレームワークや SQL ライブラリーにありがちな、ソースの見づらさを当然のものと受け取りませんでした。彼の以前の経歴は Microsoft の.NET solutions に携わっていました。

当初、Laravel は Taylor 氏の遊びで作られたものです。しかし、これを利用して Web アプリを開発した会社が資金援助を行い、結果ここまで発展しました。実に一年での急成長です。

既に 2013 年の2月にカンファレンスがワシントン DC で開かれ、小中規模のスポンサーが11社も付きました。

もし読者の方が、大規模なアプリ開発に携わっており、上流からのきちんとした設計に基づく開発手法を取られ、人員も確保できるのでしたら、他の有名フレームワークをご利用されたほうがよろしいでしょう。Laravel はまだ未熟な部分もあります。

活用できるベストシナリオとしては「アジャイルスタイルの開発方法をとり、数人のグループで新しいアプリを約一ヶ月で開発する」ような場合でしょう。PHP フレームワークでの経験があれば、数日もあれば十分習得でき、コードの読みやすさは、顧客からの要求に対応しやすくなります。複雑なフレームワークの全体を把握するだけで一ヶ月かかることもありません。ちょっとした変更に、自分の書いた難しいソースを読み解くこともありません。（もちろん、いくら Laravel を使用しても、複雑に書いてしまえば、元の木阿弥ですよ。）このベストシナリオは、ここに上げたような条件で Laravel を用い実際に開発を行った方が、彼の記事で Laravel を褒めていた内容を紹介したものです。

Laravel は学習コストが最低で済みます。たやすく習得できるフレームワークです。ですから、ある程度の人数で開発するのだが、共通のフレームワークの経験が見つからず、学習に手間取らないフレームワークを探している場合にピッタリです。

もしくは、多くの Web サイト開発見られるようなアジャイルスタイルの開発手法を取られているチームにも適しているでしょう。コードの読みやすさは抜群です。

更に、数多くの案件を個人で請け負っていられる方にも適しているでしょう。顧客からの修正依頼にも、今までより気軽に答えられるようになるでしょう。

また、新たにフレームワーク自身を勉強したい方にも適しています。最初に大きなフレームワークにとりかかるのでは、全体を把握するだけでも、時間がかかります。余りにも小さなフレーム

ワークでは、使用するメリットを十分に感じられないでしょう。学習しやすく、それなりに機能を備えた Laravel は学習目的にも最適です。

開発するのが「楽しい」と感じさせる不思議なフレームワークです。趣味であれ、仕事であれ、開発の楽しみを感じさせてくれる Laravel を多くの方に使用していただきたいと思います。

Ｌｅａｎｐｕｂ

電子書籍版は Leanpub を利用し販売しています。

国内のサービスを介せず、Leanpub を利用するのは、メンテナンスの理由です。Leanpub で一度購入いただくと、書籍の内容を修正した場合、購入者にメールで通知を行い、新しいバージョンを無料でダウンロードしていただける仕組みになっています。

これにより、多くの時間を費やし、「完全な書物」を用意したため、読者にとって最適な時期に、書籍を手元に届けられない悲劇を回避できます。多くのテクニカルな書物で見られるような、「ちょっと機会を逃した」残念な出版を避ける事ができます。

現在の読者は Web 情報に新しさを、書物には正確さを期待しているとも言えます。Leanpub での出版物は最初は Web 情報と同じく、「新しいが、正確さに欠ける」かも知れません。しかし、アップデートが可能であるということは、出版物の内容の修正・追加が可能であります。読者からのフィードバックを受け付け、直ぐに対応できるということです。その結果、内容を「常に新しく、正確」なものへと育てることができます。

出版ごとにサイトを用意し、正誤表を公開するという、著者と読者にとって手間がかかることを行わなくとも、常に最新版を読者の手元に置いていただけます。

このドキュメントは原文が存在するため、日本語版を勝手に変更はしません。ですが、誤字脱字、翻訳の間違いなどをご指摘いただけば、可能な限り迅速に対処させていただきます。

Leanpub は新しいサービスで、日本語の対応に関してまだ問題を抱えております。ですから、Leanpub の開発者と連絡を取り、少しずつクオリティーも上げていきたいと思います。

また無料開放ではなく、有料での配布にさせていただくのは、私が体調不良のため働けず、サイト維持のためいくらかのお金をご寄付いただかなければならないためです。元気でバリバリ働けるのでしたら、私個人でも、サイト維持は難なく可能ですが、現状それが不可能です。（こうした状況であり、時間が取れるため、翻訳・出版も可能だったので、ネガティブな意味だけでありません。）

値段は、最低 5.55 ドル、希望価格 7.77 ドルにさせていただいております。当初は最低 1.11 ドルに設定していましたが、アップデートの頻度が結構高いため、作業量にそぐわなくなってしまい、値上げしました。この値段はドネーションの意味も込め、幅を持たせております。無理のない範囲で値段を決めていただけます。もし会社等で購入し、多少のドネーションを行なっても良いという場合は、表示される値段をクリックしていただければ、値段を直接ご指定いただけます。

支払いは Paypal およびカードです。今回の Laravel 3 公式ドキュメント日本語翻訳電子書籍版とは関係なく、日本語での情報提供に対しドネーションいただける方は、Paypal アカウント hiro.soft@gmail.com へ寄付をお願いします。また、Laravel 開発者の Taylor Otwell 氏へ寄付されたい方は、taylorotwell@gmail.com の Paypal アカウントへどうぞ。

川瀬　裕久

1 概要

1.1 初めに

Laravel のドキュメントへようこそ。このドキュメントはスタートガイドとして、さらに特徴の紹介としても役立つように書かれています。どこから読んでも学習できますが、以前に学んだ概念をもとに、その後に続くドキュメントは書かれていますので、初めから順番に読むことをお勧めします。

1.2 Ｌａｒａｖｅｌを楽しめるのは誰？

Laravel は柔軟性と読み書きしやすさを重視した、パワフルなフレームワークです。初めて Laravel に触れる方は、人気がある軽量な PHP フレームワークを使用して開発する時と同じ、安らぎを感じるでしょう。もうちょっと経験を積んだユーザーであれば、他のフレームワークではできない方法で、コードをモジュール化できることを評価するでしょう。Laravel の柔軟性は、要求に何度でも応じ、アプリケーションを修正しながら、形作ることを可能にし、表現性はあなたとあなたのチームが開発するコードをシンプルで読みやすくしてくれるでしょう。

1.3 Ｌａｒａｖｅｌはどこが違うの？

Laravel には他のフレームワークと違った特徴を数多く持っています。特に重要な点をいくつか紹介しましょう。

- **バンドル**は Laravel のモジュールパッキングシステムです。Laravel バンドルリポジトリー[1] は、アプリケーションへ簡単に機能を付け加えられるように、予め用意されています。バンドルリポジトリーから bundles ディレクトリーにダウンロードしても良いですし、”Artisan” コマンドラインツールを使い、自動的にインストールすることもできます。
- **Eloquent ORM** は最も進化した PHP アクティブレコードを実装しています。リレーションシップとネストされた eager ローディングで簡単に制約を適用できる能力を使えば、自分のデーターを完全にコントロールでき、アクティブレコードの便利さを十分に体験できるでしょう。Eloquent は Laravel のクエリービルダーである Fluent のメソッドを完全にサポートしています。
- **アプリケーションロジック**を（多くの Web 開発者にはお馴染みの）コントローラーでアプリケーションに実装することもできますし、また Sinatra フレームワークと似たようなシンタックスを使い、ルートの定義に直接記述することもできます。Laravel は小さなサイトから、巨大なエンタープライズアプリケーションまで、必要に応じて全て作成できるだけの柔軟性を開発者に提供する哲学で、設計されています。

[1] http://bundles.laravel.com/

- **リーバスルーティング**で名前付きのルートへリンクを作成できます。リンクを作成するときにルートの名前を使えば、Laravel は自動的に正しい URI を挿入します。これを使うことにより、後ほどルートを変更しても、Laravel がサイト中のリンク全部を適切に更新します。
- **Rest コントローラー**は GET と POST のロジックを分ける一つの手法です。例えばログインにおいて、コントローラーの get_login() アクションでフォームを担当させ、コントローラーの post_login() アクションで、送信されたフォームを受け取り、バリデーションし、エラーメッセージと一緒にログインフォームにリダイレクトさせたり、各ユーザーのダッシュボードにリダイレクトさせたりできます。
- **クラスのオートロード**はオートロードの環境設定を保つ手間を省き、使用していない不必要なコンポーネントをロードしてしまうことを防げます。ライブラリーやモジュールを使いたいのですか?ローディングに悩むことはありません。どうぞ使ってください。後は Laravel が面倒を見ます。
- **ビューコンポーサー**はビューがロードされた時点で実行されるコードブロックです。良い例がブログのサイドナビに見られる、投稿をランダムにリスト表示するものです。コンポーサーは必要のあるブログポストを全てロードするロジックで構成されるでしょう。そうしてビューをロードすれば、表示する準備は全て予め済んでいるわけです。これにより、メソッドのページコンテンツに関連する、ビューのモジュールで使用するデータのロードを、全てのコントローラー側で確実に行わなくてはならない手間を省くことができます。
- **IoC コンテナ** (Inversion of Control) は新しいオブジェクトを生成するメソッドを提供し、随意にインスタンスを生成したり、シングルトンでの使用をできるようにするものです。IoC により、外部ライブラリーの使用準備を行う必要は滅多になくなります。また、きっちりと決まった柔軟性のないファイル構造に係わる必要はなく、IoC を使用したオブジェクトにはコードのどこからでもアクセスできることも意味しています。 。
- **マイグレーション**はデータベーススキーマのバージョンコントロールで、Laravel に直接統合されています。生成も実行も”Artisan” コマンドラインユーティリティーを使用して行えます。他のメンバーがスキーマを変更したら、リポジトリーからコピーをローカル環境に置き、マイグレーションを実行します。すると、あなたのデータベースもアップデートされます!
- **ユニットテスト**は Laravel の大切な一部です。Laravel 自身も何百ものテストにより、新しい変更が予期せず他の部分を壊していないことを確認するために使っています。これは、Laravel が業界で最も安定してるフレームワークであると考えられている理由の一つです。さらに Laravel は皆さんが自分のコードにユニットテストを書くのを簡単にしてくれます。その後で、”Artisan” コマンドユーティリティーを使いテストを実行できます。
- **自動ページネーション**はアプリケーションロジックがページネーションの設定のためにごちゃごちゃになることを防ぎます。現在のページを得て、DB のレコード数を取得し、limit/offset を使用してデーターを SELECT する代わりに、ただ”paginate” を呼び出し、ビューのどこにページリンクを出力するのか Laravel に教えて下さい。Laravel は自動的に残りの面倒を見ます。Laravel のページネーションシステムは簡単に使用でき、簡単に変更できるように設計されています。強調しますが、Laravel がこれらを自動的に処理するからといっても、自分で呼び出したり、システムを設定できないわけではありません。そうしたければ、手動で行えます。

これは他の PHP フレームワークとの違いを示す、わずかな例にすぎません。こうした特徴とその他すべて、このドキュメント全体を通して記述してあります。

1.4 アプリケーション構造

Laravel のディレクトリー構造は他の人気のある PHP フレームワークと似せて設計されています。他のフレームワークで採用されている方法と似ている構造を使うことで、どんなアプリケーションでも、どんなサイズのものでも簡単に作成できます。

Laravel のアーキテクチャがユニークだからといっても、アプリケーションに合わせて、開発者が独自の構造を構築することも可能です。これはコンテントマネージメントシステムのような大きなプロジェクトに有効でしょう。こうした柔軟な構造は Laravel 独自なものです。

このドキュメントを通し、設置するのに最適なデフォルトの位置を指定していきたいと思います。

1.5 Ｌａｒａｖｅｌのコミュニティー

Laravel フォーラム[2]は手助けを得たり、手助けしたり、もしくは他の人が何を言っているかただ眺めたりできる素晴らしい場所です。

我々の多くは毎日 FreeNode の #laravel IRC チャンネルに接続しています。Laravel のフォーラム記事に接続方法が説明されています。[3] この IRC チャンネルにつなぎっぱなしにすることは、Laravel を使用する Web 開発について多くを学ぶ方法です。どうぞ質問をし、他の人の質問に答え、もしくはつないだままにして、他の人の質問と答から学んでください。私達は Laravel を愛していますし、Laravel について話すのも大好きです。ですからよそ者にはならないでください！

1.6 ライセンス情報

Laravel はMIT ライセンス[4]のもとにライセンスされているオープンソースのソフトウェアです。

[2] http://forums.laravel.com

[3] http://forums.laravel.com/viewtopic.php?id=671

[4] http://www.opensource.org/licenses/mit-license.php

2 テンプレート

2.1 基本

多分、あなたのアプリケーションでもほとんどのページに渡って、共通のレイアウトを使用していることでしょう。このレイアウトを手動で、全てのコントローラーアクションに生成するのは、辛いですよね。コントローラーにレイアウトが指定出来れば、開発はもっと楽しくなります。では、これを行なってみましょう。

コントローラーに”ｌａｙｏｕｔ”プロパティを指定する

```
class Base_Controller extends Controller {

        public $layout = 'layouts.common';

}
```

コントローラーのアクションからレイアウトにアクセスする

```
public function action_profile()
{
        $this->layout->nest('content', 'user.profile');
}
```

> **注目:**レイアウトを使う場合、アクションは何もリターンしません。

2.2 セクション

ビューのセクションはネストしたビューからレイアウトにコンテンツを挿入するシンプルな方法を提供します。例えば、多分あなたはレイアウトのヘッダーの中にネストビューが必要としている Javascript を挿入したいとします。これを掘り下げてみましょう。

ビューの中にセクションを生成する

```
<?php Section::start('scripts'); ?>
        <script src="jquery.js"></script>
<?php Section::stop(); ?>
```

セクションの内容をレンダリングする

```
<head>
        <?php echo Section::yield('scripts'); ?>
</head>
```

Ｂｌａｄｅのショートカットを使いセクション操作する

```
@section('scripts')
        <script src="jquery.js"></script>
@endsection

<head>
        @yield('scripts')
</head>
```

2.3 Ｂｌａｄｅテンプレートエンジン

Blade はビューを書くことを至高の喜びにしてくれます。Blade ビューを作成するには、ファイルの拡張子を".blade.php" にするだけです。Blade により、美しく控えめなシンタックスで、PHP コントロール構文やデーターのエコーを書くことができるようになります。例をご覧ください。

Ｂｌａｄｅを使い、変数をエコーする

```
Hello, {{ $name }}.
```

Ｂｌａｄｅを使い、関数の結果をエコーする

```
{{ Asset::styles() }}
```

ビューをレンダーする

@include を使用し、他のビューの中にビューをレンダーすることができます。レンダーされるビューは自動的に、現在のビューの全てのデーターを継承します。

```
<h1>Profile</hi>
@include('user.profile')
```

同様に、**@include** と同じような働きをする **@render** も使用できます。違いはレンダー時に、現在のビューのデーターを継承**しない**ことです。

```
@render('admin.list')
```

Ｂｌａｄｅコメント

```
{{-- これがコメントです --}}

{{--
        これは
        複数行に渡る
        コメント例です。
--}}
```

> **注目：**Blade のコメントは、HTML コメントとは異なり、HTML ソースには出力されません。

Bladeコントロール構文

Ｆｏｒループ：

```
@for ($i = 0; $i <= count($comments); $i++)
        コメントの内容は {{ $comments[$i] }}
@endfor
```

Ｆｏｒｅａｃｈループ：

```
@foreach ($comments as $comment)
        コメントの内容は {{ $comment->body }}.
@endforeach
```

Ｗｈｉｌｅループ：

```
@while ($something)
        まだループ中です！
@endwhile
```

Ｉｆ文：

```
@if ( $message == true )
        メッセージを出力中！
@endif
```

ＩｆＥｌｓｅ文：

```
@if (count($comments) > 0)
        コメントがあります！
@else
        コメントがありません！
@endif
```

Ｅｌｓｅｉｆ文：

```
@if ( $message == 'success' )
        成功した！
@elseif ( $message == 'error' )
        エラーが起きた。
@else
        ここに来るのかな？
@endif
```

ＦｏｒＥｌｓｅ文：

```
@forelse ($posts as $post)
        {{ $post->body }}
@empty
        配列中にはポストはありません！
@endforelse
```

Ｕｎｌｅｓｓ文：

```
@unless(Auth::check())
        Login
@endunless

// 同じ内容…

<?php if ( ! Auth::check()): ?>
        Login
<?php endif; ?>
```

2.4 Ｂｌａｄｅレイアウト

Blade はきれいでエレガントなシンタックスを PHP の一般的なコントロール構文に提供しているだけでなく、ビューのレイアウトに使用できる、美しい手法も用意しています。例えば、あなたのアプリケーションでは、共通のルック・アンド・フィールを提供するために、「マスター」ビューを使っているでしょう。それは多分、こんな感じだと思います：

```
<html>
        <ul class="navigation">
                @section('navigation')
                        <li>Example Item 1</li>
                        <li>Example Item 2</li>
                @endsection
        </ul>

        <div class="content">
                @yield('content')
        </div>
</html>
```

“content” セクションが生成されることに注目してください。このセクションに何かテキストを埋めるなくてはなりません。では、このレイアウトを使用する、別のビューを作成しましょう。

```
@layout('master')

@section('content')
        profileページへようこそ！
@endsection
```

素晴らしい！これで、ルートからシンプルに”profile” ビューをリターンできます。

```
return View::make('profile');
```

porfile ビューはありがたいことに、**@layout** 文により、Laravel は”master” テンプレートを自動的に使用してくれます。

> **重要：** **@layout** はファイルの最初の一行で呼び出す必要があり、先頭にホワイトスペースをつけたり、途中で改行してはいけません。

@ｐａｒｅｎｔで追加する

場合により、セクションのレイアウトを置き換えてしまうよりは、追加したいこともあります。例えば、”master” レイアウトのナビゲーションリストを考えてください。ここに、新しいアイテムを追加してみましょう。こんな風になります：

```
@layout('master')

@section('navigation')
        @parent
        <li>Nav Item 3</li>
@endsection

@section('content')
        profileページへようこそ！
@endsection
```

@parent はレイアウトの *navigation* セクションの内容と置き換わります。これはレイアウトの拡張と継承を実現する美しくてパワフルな手法を提供しています。

3 フォームの作成

> **注意:** フォーム要素に表示されるすべての入力データーは HTML::entities メソッドを通してフィルタリングされます。

3.1 フォームを開く

現在のＵＲＬへＰＯＳＴするフォームを開く

```
echo Form::open();
```

ＵＲＩとリクエスト方法を指定し、フォームを開く

```
echo Form::open('user/profile', 'PUT');
```

ＨＴＴＰＳのＵＲＬへＰＯＳＴするフォームを開く

```
echo Form::open_secure('user/profile');
```

フォームタグに追加のＨＴＭＬ属性を指定する

```
echo Form::open('user/profile', 'POST', array('class' => 'awesome'));
```

ファイルアップロードを受け付けるフォームを開く

```
echo Form::open_for_files('users/profile');
```

ＨＴＴＰＳを使い、ファイルアップロードを受け付けるフォームを開く

```
echo Form::open_secure_for_files('users/profile');
```

フォームを閉じる

```
echo Form::close();
```

3.2 ＣＳＲＦプロテクション

Lravel はクロスサイト・リクエスト・フォージェリからサイトを守る簡単な方法を提供しています。まず、ユーザーのセッションにランダムトークンを設置します。これは自動的に行われますので、何もする必要はありません。次に、フォームに隠し入力フィールドを生成し、ランダムトークンを埋め込みます。

セッションのＣＳＲＦトークンを埋め込む隠しフィールドを生成する

```
echo Form::token();
```

ルートにＣＳＲＦフィルターを追加する

```
Route::post('profile', array('before' => 'csrf', function()
{
        //
}));
```

ＣＳＲＦトークン文字列を取得する

```
$token = Session::token();
```

> Laravel の CSRF プロテクション機能を使用する前に、セッションドライバーを指定する必要があります。

参照:

- ルートフィルター
- クロスサイト・リクエスト・フォージェリ[1]

3.3 ラベル

ラベル要素を生成する

```
echo Form::label('email', 'E-Mail Address');
```

ラベルに追加のＨＴＭＬ要素を指定する

[1] http://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AD%E3%82%B9%E3%82%B5%E3%82%A4%E3%83%88%E3%83%AA%E3%82%AF%E3%82%A8%E3%82%B9%E3%83%88%E3%83%95%E3%82%A9%E3%83%BC%E3%82%B8%E3%82%A7%E3%83%AA

```
echo Form::label('email', 'E-Mail Address', array('class' => 'awesome'));
```

ラベルの表示内容のＨＴＭＬエスケープを行わない

```
echo Form::label('confirm', 'Are you <strong>sure</strong> you want to proceed\
?', null, false);
```

ラベルの表示内容の自動 HTML エスケープを行わないため、4番目の引数にオプションとして `false` を指定することもできます。

> ラベルを生成後に、ラベルと一致する名前で作られる HTML 要素は、その名前と同じ ID も生成されます。

3.4 テキスト、テキストエリア、パスワード、隠しフィールド

テキスト入力要素の生成

```
echo Form::text('username');
```

テキスト入力要素にデフォルト値を指定する

```
echo Form::text('email', 'example@gmail.com');
```

> **注目：** *hidden* と *textarea* メソッドは *text* メソッドと使い方は同じです。一つ覚えるだけで、3つまとめて学べます。

パスワード入力要素を生成する

```
echo Form::password('password');
```

3.5 チェックボックスとラジオボタン

チェックボックス要素を生成する

```
echo Form::checkbox('name', 'value');
```

チェック状態をデフォルトにして生成する

```
echo Form::checkbox('name', 'value', true);
```

> **注目:** *radio* メソッドは *checkbox* と全く同じです。1つで2つ分ですね。

3.6 ファイル入力

ファイル入力要素を生成する

```
echo Form::file('image');
```

3.7 ドロップダウンリスト

配列の要素から、ドロップダウンリストを生成する

```
echo Form::select('size', array('L' => 'Large', 'S' => 'Small'));
```

一つのアイテムをデフォルトに指定し、ドロップダウンリストを生成する

```
echo Form::select('size', array('L' => 'Large', 'S' => 'Small'), 'S');
```

3.8 ボタン

Ｓｕｂｍｉｔボタン要素を生成する

```
echo Form::submit('Click Me!');
```

> **注目:** ボタン要素を生成する必要がある？ならば、*button* メソッドをお試しください。*submit* と使い方は同じです。

3.9 カスタムマクロ

カスタムフォームクラスヘルパー、通称「マクロ」を簡単に定義できます。実例を見て下さい。最初に、マクロを名前と無名関数を指定して、登録します。

フォームマクロを登録する

```
Form::macro('my_field', function()
{
        return '<input type="awesome">';
});
```

次に、名前でそのマクロを呼び出します。

カスタムマクロを呼び出す。

```
echo Form::my_field();
```

4 バリデーション

4.1 基本

ほとんどのインタラクティブな Web アプリケーションは、データーのバリデーションが必要です。例えば、登録フォームでは、パスワードの再確認が必要でしょう。多分、メールアドレスは重複していてはいけません。データーのバリデーションは堅苦しいプロセスです。ありがたいことに、Laravel では、そうではありません。Validator クラスはデーターのバリデーションを簡単にしてくれる素晴らしいヘルパーを用意してくれています。一例を見てみましょう。

バリデーションしたいデーターを配列で獲得

```
$input = Input::all();
```

データーに対するバリデーションルールを定義

```
$rules = array(
        'name'  => 'required|max:50',
        'email' => 'required|email|unique:users',
);
```

Ｖａｌｉｄａｔｏｒインスタンスを作成し、実行する

```
$validation = Validator::make($input, $rules);

if ($validation->fails())
{
        return $validation->errors;
}
```

errors プロパティは、エラーメッセージの取り扱いを簡単にしてくれる、シンプルな message collector クラスです。もちろん、デフォルトのエラーメッセージは全てのバリデーションルールに用意してあります。デフォルトのメッセージは **language/en/validation.php** にあります。

これで、基本的な Validator クラスの使い方に慣れました。データーをバリデーションするのに使用するルールについて、掘り下げて学ぶ用意ができました。

4.2 バリデーションルール

必須項目

存在し、空文字列ではないことをバリデートする属性です。

```
'name' => 'required'
```

あるフィールドが入力済みの場合、同時に入力されていることをバリデートする属性です。

```
'last_name' => 'required_with:first_name'
```

文字種指定

英文字だけで構成されていることをバリデートする属性です。

```
'name' => 'alpha'
```

英文字と数字だけで構成されていることをバリデートする属性です。

```
'username' => 'alpha_num'
```

英数字とダッシュ、下線で構成されていることをバリデートする属性です。

```
'username' => 'alpha_dash'
```

サイズ

与えられた文字数であること、もしくは数字項目の場合はその値であることをバリデートする属性です。

```
'name' => 'size:10'
```

サイズが与えられた範囲内であることをバリデートする属性です。

```
'payment' => 'between:10,50'
```

> **注目:** 最低値と最高値も含まれます。

与えられたサイズ以上であることをバリデートする属性です。

```
'payment' => 'min:10'
```

与えられたサイズ以下であることをバリデートする属性です。

```
'payment' => 'max:50'
```

数字項目

数字であることをバリデートする属性です。

```
'payment' => 'numeric'
```

整数であることをバリデートする属性です。

```
'payment' => 'integer'
```

内包と除外

リストの値の中にあることをバリデートする属性です。

```
'size' => 'in:small,medium,large'
```

リストの値の中に無いことをバリデートする属性です。

```
'language' => 'not_in:cobol,assembler'
```

確認項目

confirmed ルールは *attribute_confirmation* 項目が存在し、その値と一致していることをバリデートする属性です。

確認項目と一致していることをバリデート

```
'password' => 'confirmed'
```

この例で Validator は、*password* 項目が、配列の中の *password_confirmation* 項目と一致していることを、確認します。

受け入れの確認

accepted ルールは項目が *yes* か *1* であることをバリデートします。このルールは「サービスの規約」のようなフォームのチェックボックスのバリデーションに役立ちます。

その項目が受け入れられたかバリデートする

```
'terms' => 'accepted'
```

4.3 他項目との比較

項目値が、他のフィールドの値と同じ事をバリデートする

```
'token1' => 'same:token2'
```

２つの項目の値が異なることをバリデートする

```
'password' => 'different:old_password',
```

正規表現

match ルールは与えられた正規表現と一致することをバリデートします。

正規表現と一致することをバリデートする

```
'username' => 'match:/[a-z]+/';
```

一意と存在

値が与えられたデータベーステーブルで一意であることをバリデートする

```
'email' => 'unique:users'
```

上記の例では、*email* 項目は *users* テーブルで、ユニークであるかチェックされます。その項目名とカラム名が異なっている時にもユニークであることを確かめたいのですか？問題ありません。

ｕｎｉｑｕｅルールでカスタムカラム名を指定する

```
'email' => 'unique:users,email_address'
```

レコードを更新する場合、通常は unique ルールを使用しても、更新するそのレコードに対しては適用を除外したいことはよくあります。例えば、ユーザープロフィールの更新では、メールアドレスの変更は許可されていることでしょう。しかし、*unique* ルールが効いていると、そのユーザーがメールアドレスを変更しなかった場合、*unique* ルールは失敗してしまいます。そのため、更新するユーザーに対しては、このルール適用を飛ばす必要があります。

ＩＤを指定し、ｕｎｉｑｕｅルールを無視するよう強制する

```
'email' => 'unique:users,email_address,10'
```

データベーステーブルに項目の値が存在していることをバリデートする

```
'state' => 'exists:states'
```

ｅｘｉｓｔｓルールにカスタムカラム名を指定する

```
'state' => 'exists:states,abbreviation'
```

日付

指定日付以前であることをバリデートする

```
'birthdate' => 'before:1986-05-28';
```

指定日付以降であることをバリデートする

```
'birthdate' => 'after:1986-05-28';
```

> **注目:** **before** と **after** バリデーションルールは日付の解析に、PHP の関数である **strtotime** を利用しています。

日付が与えられたフォーマットであることをバリデートする

```
'start_date' => 'date_format:H\\:i'),
```

> **注意:** パラメーターのセパレーターt として扱われないように、コロンをバックスラッシュでエスケープすること

日付に対するフォーマットのオプションについてはPHP ドキュメント[1]に記述されています。

メールアドレス

メールアドレスとして正しいかバリデートする

```
'address' => 'email'
```

> **注目:** このルールは PHP 組み込み関数の *filter_var* メソッドを使用しています。

ＵＲＬ

有効なＵＲＬであるかバリデートする

[1] http://php.net/manual/ja/datetime.createfromformat.php#refsect1-datetime.createfromformat-parameters

```
'link' => 'url'
```

アクティブなＵＲＬであるかバリデートする

```
'link' => 'active_url'
```

> **注目：** *active_url* ルールは URL がアクティブであるか判断するために *checkdnsr* を使用しています。

アップロードファイル

mimes ルールはアップロードファイルが指定された MIME タイプであるかバリデートします。このルールは、そのファイルの内容を読み、実際の MIME タイプを決めるために、PHP Fileinfo 拡張を使用しています。*config/mimes.php* の中で定義されている拡張子で、引数で指定されたものは、このルールを通されます

指定されたタイプの一つであることをバリデートする

```
'picture' => 'mimes:jpg,gif'
```

> **注目：**ファイルをバリデートする時は、Input::file() か入力::all() を入力項目収集に使用してください。

ファイルが画像であることをバリデートする

```
'picture' => 'image'
```

ファイルが指定キロバイトより小さいことをバリデートする

```
'picture' => 'image|max:100'
```

配列

配列をバリデートする

```
'categories' => 'array'
```

ちょうど３要素を持つ配列をバリデートする

```
'categories' => 'array|count:3'
```

１から３要素を持つ配列をバリデートする

```
'categories' => 'array|countbetween:1,3'
```

２つ以上の要素を持つ配列をバリデートする

```
'categories' => 'array|countmin:2'
```

多くて２つの要素を持つ配列をバリデートする

```
'categories' => 'array|countmax:2'
```

4.4 エラーメッセージの取得

Laravel では、シンプルなエラー収集クラスを使用し、手軽にエラーメッセージを取り扱えるようになっています Validator のインスタンで *passes* か *fails* メソッドを呼び出した後に、*errors* プロパティーを利用してアクセスできます。メッセージを取得するためにいくつかの関数が用意されています。

一項目にエラーメッセージがあるか確かめる

```
if ($validation->errors->has('email'))
{
        // The e-mail attribute has errors…
}
```

その項目の最初のエラーメッセージを取得する

```
echo $validation->errors->first('email');
```

時には、HTML 要素でラップしたエラーメッセージが必要なこともあるでしょう。大丈夫です。2番目の引数に、:message プレースホルダーを使い、フォーマットを指定してください。

エラーメッセージをフォーマットする

```
echo $validation->errors->first('email', '<p>:message</p>');
```

指定された項目の、すべてのエラーメッセージを取得

```
$messages = $validation->errors->get('email');
```

指定された項目の、すべてのエラーメッセージをフォーマット

```
$messages = $validation->errors->get('email', '<p>:message</p>');
```

全ての項目の、全てのエラーメッセージを取得

```
$messages = $validation->errors->all();
```

全ての項目の、全てのエラーメッセージをフォーマット

```
$messages = $validation->errors->all('<p>:message</p>');
```

4.5 バリデーション実例

一度バリデーションを実行すれば、簡単にビューにそれを表示できます。Laravel では、驚異的なシンプルさで行えます。典型的なシナリオに沿って、行なってみましょう。2つのルートを定義します。

```
Route::get('register', function()
{
        return View::make('user.register');
});

Route::post('register', function()
{
        $rules = array(…);

        $validation = Validator::make(Input::all(), $rules);

        if ($validation->fails())
        {
                return Redirect::to('register')->with_errors($validation);
        }
});
```

素晴らしいですね!2つのシンプルな登録のためのルートができました。一つはフォームを表示処理し、もうひとつはフォームの投稿を処理します。POST ルートでは、入力に対してバリデーションを行なっています。バリデーションが失敗した場合、表示に使えるようにバリデーションエラーをセッションに退避 (flash) させ、登録フォームへリダイレクトします。

しかし、GET ルートで errors とビューを明確に結びつけていないことに注目してください。それでも、エラー変数 ($errors) はビューで使用できます。賢明なことに Laravel は、errors がセッションにあれば、あなたのため、ビューに渡してくれます。errors がセッションに存在していなけ

れば、からのメッセージコンテナがビューに渡されます。あなたはビューの中で、errors 編集を通して、いつもメッセージコンテナが存在すると思っていられます。私たちは、あなたの人生を楽にすることが大好きです。

例えば、メールアドレスのバリデーションに失敗すれば、セッション変数の $errors の中に'email' を見つけることができます。

```
$errors->has('email')
```

Blade を使い、ビューにエラーメッセージを条件付きで付け加えることもできます。

```
{{ $errors->has('email') ? 'Invalid Email Address' : 'Condition is false. Can \
be left blank' }}
```

これは例えば Twitter Bootstrap のようなものを使用しているときに、条件付きでクラスを付け加えたい時に便利に使えます。例えば、メールアドレスのバリデーションに失敗したら、Bootstrap の"error" クラスを *div class="control-group"* 文に付け加えたいことでしょう。

```
<div class="control-group {{ $errors->has('email') ? 'error' : '' }}">
```

バリーションが失敗したら、ビューには *error* クラスが付け加え表示されるでしょう。

```
<div class="control-group error">
```

4.6 カスタムエラーメッセージ

エラーメッセージをデフォルトから変更したいのですか?たぶん、項目名とルールを指定して、カスタムエラーメッセージを使いたい場合さえあるでしょう。どちらにしても、Validator クラスが簡単に実現してくれます。

Ｖａｌｉｄａｔｏｒに渡すカスタムメッセージの配列を作成

```
$messages = array(
        'required' => 'The :attribute field is required.',
);

$validation = Validator::make(Input::get(), $rules, $messages);
```

素晴らしいですね!これで、バリデーションのチェックしに失敗した時、いつでもカスタムメッセージが使用できます。けど、**:attribute** なんたらは、メッセージの中でどうなるんでしょう?あなたが楽になるように、Validator クラスは、**attribute** プレースホルダーを実際の項目の名前に置き換えてくれます!項目名の下線も取り除いてくれます。

エラーメッセージを作成するときには、他にも**:other**、**:size**、**:min**、**:max**、**:values** プレースホルダーも使用できます。

他のバリデーションプレースホルダー

```
$messages = array(
        'same'    => 'The :attribute and :other must match.',
        'size'    => 'The :attribute must be exactly :size.',
        'between' => 'The :attribute must be between :min - :max.',
        'in'      => 'The :attribute must be one of the following types: :values',
);
```

でも、カスタムメッセージが使えると言っても、email 項目に対してしか指定できないのでしょうか？大丈夫です。**項目_ルール**のネーミングルールを使い、メッセージを指定して下さい。

与えられた項目のカスタムメッセージを指定する

```
$messages = array(
        'email_required' => 'We need to know your e-mail address!',
);
```

上記の例のように、要求されたカスタムメッセージは email 項目に使用されますが、他のすべての項目にはデフォルトのメッセージが使用されます。

しかし、たくさんのカスタムエラーメッセージを使用するために、コードの中で指定すれば、扱いにくくめちゃくちゃになるでしょう。ですから、バリデーション言語ファイルの中の **custom** 配列で、カスタムメッセージを指定して下さい。

バリデーション言語ファイルにカスタムエラーメッセージを追加する

```
'custom' => array(
        'email_required' => 'We need to know your e-mail address!',
)
```

4.7 カスタムバリデーションルール

Laravel は多くのパワフルなバリデーションルールを提供しています。しかし、結局自分用に作成する必要が起きるのは、よくあるでしょう。バリデーションルールを作成するには2つのシンプルな方法が用意されています。両方共素晴らしいので、プロジェクトにあった方をお使いください。

カスタムバリデーションルールを登録

```
Validator::register('awesome', function($attribute, $value, $parameters)
{
    return $value == 'awesome';
});
```

この例は、Validator に新しいバリデーションルールを登録しています。ルールは3つの引数を取ります。最初はバリデーションを行う項目名です。2つ目はバリデーションを行う値で、3つ目はルールに指定されるパラメーターです。

あなたのカスタムバリデーションルールを使うには次のように呼び出します。

```
$rules = array(
        'username' => 'required|awesome',
);
```

もちろん、新しいルールのエラーメッセージを定義する必要があります。これは、その場で直ぐに定義する方法と:

```
$messages = array(
        'awesome' => 'The attribute value must be awesome!',
);

$validator = Validator::make(Input::get(), $rules, $messages);
```

もしくは、**language/en/validation.php** の中にあなたのルールに対するエントリーを付け加える方法があります。

```
'awesome' => 'The attribute value must be awesome!',
```

前に述べたように、カスタムルールに引数のリストを指定し、受け取ることができます:

```
// When building your rules array…

$rules = array(
    'username' => 'required|awesome:yes',
);

// In your custom rule…

Validator::register('awesome', function($attribute, $value, $parameters)
{
    return $value == $parameters[0];
});
```

この場合、バリデーションルールの引数は、要素が”yes” だけの配列を受け取ります。

バリデーションルールを作成し保存する、もうひとつの方法は Validator クラス自身を拡張することです。拡張して新しいバージョンの Validator を作成すれば、既に存在する機能を全部使用しつつ、あなたのカスタム機能を追加できます。もし望むのでしたら、デフォルトのメソッドを置き換えることもできます。例を見ていきましょう。

最初に、**LaravelValidator** を拡張し、**application/libraries** に設置します。

カスタムＶａｌｉｄａｔｏｒクラスを定義

```
<?php

class Validator extends Laravel\Validator {}
```

次に、**config/application.php** から Validator の別名 (alias) を削除します。これは必要です。そうしないと2つの”Validator” という名前がコンフリクトを起こしてしまいます。

次に、”awesome” ルールを新しいクラスに付け加えます。

カスタムバリデーションルールを付け加える

```
<?php

class Validator extends Laravel\Validator {

    public function validate_awesome($attribute, $value, $parameters)
    {
        return $value == 'awesome';
    }

}
```

メソッドの名前に、**validate_ルール**命名規則を使っていることに注目してください。”awesome” という名前のルールのメソッドは、”validate_awesome” にしなくてはなりません。これがカスタムルールを登録する時と、Validator クラスを拡張する時の、違いの一つです。Validator クラスはシンプルに true か false をリターンします。これでおしまいです！

自分で作ったバリデーションルールのカスタムメッセージを作成する必要があることも、心に留めておいてください。そうしてもらえるのでしたら、どんなルールを定義してもらってもかまいません！

5 スキーマビルダー

5.1 基本

スキーマビルダーはデータベーステーブルの作成と変更のメソッドを提供します。スラスラ書ける構文で、ベンダー限定の何かにとらわれず、テーブルを操作できます。

参照:

- マイグレーション

5.2 テーブルの作成と削除

Schema クラスはテーブルを作成／修正するために使います。さっそく、例を見てみましょう。

簡単なデータベーステーブルを作成

```
Schema::create('users', function($table)
{
        $table->increments('id');
});
```

このサンプルを確認して行きましょう。スキーマビルダーに **create** メソッドでこれは新しいテーブルで、作成する必要があると伝えます。2つ目の引数で、無名関数を渡し、Table インスタンスを受けます。この Table オブジェクトを利用し、カラムを足したり引いたり、テーブルに索引を付けたり、すらすら書けます。

データベースからテーブルを削除

```
Schema::drop('users');
```

指定したデータベース接続のテーブルを削除

```
Schema::drop('users', 'connection_name');
```

時々、スキーマ操作を行うデータベース接続を指定する必要があるかも知れません。

操作を行う接続を指定

```
Schema::create('users', function($table)
{
        $table->on('connection');
});
```

5.3 カラム追加

Fluent テーブルビルダーのメソッドは、特定のベンダーの SQL を使用せず、カラムを追加できます。まずはメソッドです。見て行きましょう。

コマンド	説明
`$table->increments('id');`	自動増分される ID をテーブルへ
`$table->string('email');`	VARCHAR のカラム
`$table->string('name', 100);`	長さ指定の VARCHAR
`$table->integer('votes');`	INTEGER をテーブルへ
`$table->float('amount');`	FLOAT をテーブルへ
`$table->decimal('amount', 5, 2);`	最大桁数と少数桁を指定し DECIMAL を追加
`$table->boolean('confirmed');`	BOOLEAN をテーブルへ
`$table->date('created_at');`	日付をテーブルへ
`$table->timestamp('added_on');`	TIMESTAMP をテーブルへ
`$table->timestamps();`	**created_at** と **updated_at** を追加
`$table->text('description');`	TEXT をテーブルへ
`$table->blob('data');`	BLOB をテーブルへ
`->nullable()`	NULL 値可能を指定
`->default($value)`	そのカラムのデフォルト値を宣言
`->unsigned()`	整数を符号なしに設定

追記: Laravel の”boolean” タイプはすべてのデータベースシステムで small integer カラムにマップされます。

テーブルの作成とカラム追加例

```
Schema::table('users', function($table)
{
        $table->create();
        $table->increments('id');
        $table->string('username');
        $table->string('email');
        $table->string('phone')->nullable();
        $table->text('about');
        $table->timestamps();
});
```

5.4 カラム削除

データベーステーブルからカラムを削除

```
$table->drop_column('name');
```

データベーステーブルから複数のカラムを削除

```
$table->drop_column(array('name', 'email'));
```

5.5 インデックス追加

スキーマビルダーは多くのタイプのインデックスをサポートしています。インデックスを付け加えるためには2つの方法があります。それぞれのインデックスタイプごとにメソッドがあります。しかしながら、カラムを追加時に索引を定義することもできます。見てみましょう。

インデックス付きでｓｔｒｉｎｇカラムを作成

```
$table->string('email')->unique();
```

もし別の行でインデックスを定義するなら、もっと様々な指定ができます。インデックスメソッドの例をご覧ください。

コマンド	説明
`$table->primary('id');`	プライマリキーを追加
`$table->primary(array('fname', 'lname'));`	複合キーの追加
`$table->unique('email');`	ユニークキーの追加
`$table->fulltext('description');`	フルテキストインデックスの追加
`$table->index('state');`	基本インデックスの追加

5.6 インデックス削除

インデックスを削除するには、名前を指定しなくてはなりません。Laravel はすべてのインデックスに適した名前をつけます。シンプルにテーブル名に続け、インデックスしているカラムの名前、それからインデックスのタイプです。例をご覧ください。

コマンド	説明
`$table->drop_primary('users_id_primary');`	“users” テーブルのプライマリーキーを削除
`$table->drop_unique('users_email_unique');`	“users” テーブルのユニークインデックスを削除
`$table->drop_fulltext('profile_description_fulltext');`	“profile” テーブルから、フルテキストインデックスを削除
`$table->drop_index('geo_state_index');`	“geo” テーブルから、基本インデックスを削除

5.7 外部キー

Schema クラスの記述的なインターフェイスを使用し、テーブルに外部キー束縛を簡単に追加できます。例えば、**posts** テーブルに、**user_id** があり、**users** テーブルの **id** カラムを参照しているとしましょう。カラムに外部キー束縛を付け加える方法です。

```
$table->foreign('user_id')->references('id')->on('users');
```

更に、「削除 (on delete)」と「更新 (on update)」アクションを外部キーに指定できます。

```
$table->foreign('user_id')->references('id')->on('users')->on_delete('restrict\
');
```

```
$table->foreign('user_id')->references('id')->on('users')->on_update('cascade'\
);
```

また、簡単に外部キーを削除することもできます。スキームビルダーにおけるデフォルトの外部キーの名前は、他のインデックスを作成する場合と同じ規則に従っています。サンプルをどうぞ。

```
$table->drop_foreign('posts_user_id_foreign');
```

> **注意:**外部キーで参照されるフィールドは自動増分項目であり、そのため自動的に unsigned integer になります。ですから、外部キーのフィールドは **unsigned()** で作成し、両方共に同じタイプであることを確認してください。さらに、両方のテーブルはエンジンに **InnoDB** をセットしていること、参照されるテーブルは、外部キーのテーブルの**前に**作成することも確実に行なってください。

```
$table->engine = 'InnoDB';

$table->integer('user_id')->unsigned();
```