CODE HAPPY

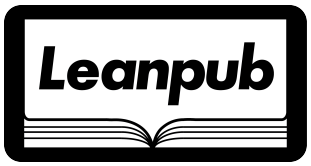
初心者のための、Laravelフレームワークを利用したWebアプリケーション開発

DAYLE REES & HIROHISA KAWASE

Laravel: Code Happy (JP)

初心者のための、Laravel フレームワークによる、Web アプリケーション開発

This version was published on 2012-09-29



This is a Leanpub book, for sale at:

http://leanpub.com/codehappy-jp

Leanpub helps authors to self-publish in-progress ebooks. We call this idea Lean Publishing. To learn more about Lean Publishing, go to: http://leanpub.com/manifesto

To learn more about Leanpub, go to: http://leanpub.com

Tweet This Book!

Please help Dayle Rees and Hirohisa Kawase by spreading the word about this book on Twitter!

The suggested hashtag for this book is #codehappy.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#codehappy

Contents

謝辞 **i**

誤記 **ii**

フィードバック **iii**

前書き **iv**

1 さあ、始めましょう **1**

1.1 方法１仮想ホストを作成する 1

1.2 方法２パブリックフォルダーのシンボリックリンク 2

1.3 話の本線に戻りまして…… 3

2 プロジェクト構成 **4**

2.1 ルートディレクトリーの構成 4

2.2 Ａｐｐｌｉｃａｔｉｏｎディレクトリー構造 5

3 コントローラーの使用 **8**

3.1 コントローラーのルーティング 8

3.2 パラメーターを渡す 10

3.3 ビューを使用する 10

3.4 ＲＥＳＴコントローラー 12

3.5 Ｂａｓｅコントローラー 13

3.6 高度なルーティング 13

4 クロージャーによるルーティング **15**

4.1 クロージャー 15

4.2 リダイレクトと名前付きルート 16

4.3 フィルター 17

4.4 ルートグループ 19

5 リンクとＵＲＬ **20**

5.1 ＵＲＬの取得 20

5.2 リンクの生成 22

CONTENTS

6 フォーム **24**

6.1 フォームを作成する 24
6.2 ラベルを付ける 25
6.3 Ｉｎｐｕｔの生成 26
6.4 ボタンを生成する 27
6.5 セキュアな入力 27
6.6 ＣＳＲＦトークン 27
6.7 フォームマクロ 28

7 入力を扱う **29**

7.1 データーの要求 29
7.2 ファイル 30
7.3 フラッシュデーター 30

8 バリデーション **32**

8.1 バリデーションの準備 32
8.2 エラー 34
8.3 バリデーションルール 34
8.4 カスタムエラーメッセージ 37
8.5 カスタムバリデーションルール 37
8.6 バリデーションクラス 38

9 マイグレーション **40**

9.1 データベースの準備 40
9.2 マイグレーション 41

10 Ｆｌｕｅｎｔクエリービルダー **46**

10.1 クエリー結果の取得 47
10.2 Ｗｈｅｒｅ節 47
10.3 テーブル接続 49
10.4 並び替え 50
10.5 Ｌｉｍｉｔ…何もいらない 50
10.6 結果を飛ばす 50
10.7 集計 50
10.8 式 51

10.9 ＋＋（またはデクリメント） 51
10.10 レコード挿入 51
10.11 レコード更新 52
10.12 レコード削除 52

11 Ｅｌｏｑｕｅｎｔ ＯＲＭ **54**
11.1 Ｅｌｏｑｕｅｎｔモデルを作成し使用する 54
11.2 リレーションシップ 58
11.3 リレーションモデルの挿入 61
11.4 ピボットテーブル 62
11.5 Ｅａｇｅｒローディング 63
11.6 ＳｅｔｔｅｒとＧｅｔｔｅｒ 64

12 イベント **65**
12.1 イベントを起こす 65
12.2 イベントをリッスンする 65
12.3 イベントにパラメーターを渡す 66
12.4 Ｌａｒａｖｅｌのイベント 66
12.5 使用例 67

13 Ｂｌａｄｅテンプレート **68**
13.1 基本 68
13.2 ロジック 68
13.3 Ｂｌａｄｅレイアウト 69

14 認証 **72**
14.1 準備 72
14.2 フォームを用意する 73
14.3 ログインの処理 75
14.4 ルートの保護 77
14.5 カスタマイズ 78

15 ブログチュートリアル **80**

15.1 デザイン 80
15.2 基本的な準備 81
15.3 Ｅｌｏｑｕｅｎｔモデル 82
15.4 ルート 83
15.5 ビュー 84
15.6 コーディング 86
15.7 機能 90

16 ユニットテスト **91**

16.1 インストール 91
16.2 テストの作成 91
16.3 テストの実行 92
16.4 コアをテストする 93

17 キャッシュ **94**

17.1 準備 94
17.2 値をセットする 94
17.3 値を取得する 94
17.4 もっと良い方法 95

18 クラスのオートローダー **96**

18.1 マッピング 96
18.2 ディレクトリーマッピング 97
18.3 名前空間マッピング 97
18.4 アンダースコア・マッピング 98

19 設定 **99**

19.1 新しい設定ファイルを作成する 99
19.2 設定の読み込み 100
19.3 設定をセットする 101

20 ＩｏＣコンテナ **102**

20.1 オブジェクトを登録する 102
20.2 オブジェクトを解決する 103
20.3 シングルトン 103

21 暗号化 **106**

21.1 復元不可能暗号化 106

21.2 復元可能暗号化 107

22 ＡＪＡＸコンテンツ **108**

22.1 テンプレートページ 108

22.2 Ｊａｖａｓｃｒｉｔｐ 109

22.3 データーのポスト 112

22.4 ＪＳＯＮレスポンス 113

22.5 ＡＪＡＸリクエストを見分ける 113

23 アプリケーションのデバッグ **115**

23.1 エラーハンドリング 115

23.2 エラー設定 116

23.3 ログ 117

24 これからも **119**

謝辞

最初にガールフレンドの Emma に感謝したいと思います。私のオタクっぽいベンチャーに我慢してくれるだけでなく、素晴らしいレッサーパンダの写真を本の表紙のために撮ってくれました！愛しているよ、Emma！

Taylor Otwell、君も愛しているよ。けど、普通の男としてだけどね。使うのが本当に楽しくて、コードが詩のように読めるフレームワークを作ってくれました。そのために多くの時間と情熱を注いでくれました。感謝しています。

Eric Barnes、Phill Sparks、Shawn McCool、Jason Lewis、Ian Landsman。フレームワークを支えてくれたサーポート全てに感謝します。そして、可能性が大きいこのプロジェクトの一員として私を迎えてくれていることにも。

両親にも感謝します。私がどんなにオタクになってもサポートし続けてくれました。

私のチュートリアルに興味を持ってくれた、すべてのブログ読者へ。ありがとう！君たちがいなければ、本を書く自信は間違いなく持てませんでした。

誤記

この本から間違いを無くすために細心の注意を行なっていますが、時々目が疲れ読むことが難しくなり、レーダーをすり抜けてしまいます。この本の中で綴り間違いやコードの間違いに気づきましたら、その章と場所を私のメールアドレスme@daylerees.com[1]へお知らせください。

エラーを見つけた場合は修正し、この本の改訂時に発行します。

（翻訳者より：日本語版に関してはhiro.soft@gmail.com[2]へお願いします。コードの間違いに関しては独断で変更できませんが、原作者に報告し、訂正を待ちたいと思います。日本語に関する部分は、私が対処いたします。）

[1] mailto:me@daylerees.com
[2] hiro.soft@gmail.com

フィードバック

この本の内容についてもフィードバックをいただくこともできます。間違いと同様に、me@daylerees.com[3]にメールを送ってください。この本について受け取った全てのメールには、返答しようと思います。

（翻訳者より：英語が苦手は方は、hiro.soft@gmail.com[4]へどうぞ。可能な限り、原作者に伝えます。）

[3] mailto:me@daylerees.com

[4] hiro.soft@gmail.com

前書き

はじめまして、皆さん！Dayle Rees です。みなさんを Laravel の魔法の世界へ通じる旅へお連れいたしましょう！まあ、くだらないことを言っていると、戸惑われていますね。この本は私の最初の本で、どうやらハイソな言葉遣いの調子がおかしくなっているようです。もしあなたが率直に、本当の人間と話すのがお好きであるのでしたら、やり直してみましょう！

ハイ、みなさん。私は Dayle Rees です。私はみなさんに次の質問を自分自身に尋ねるようなことをしでかしています。「どうして、この男が私に Laravel を教られるほど信用できるんだ？全然経験のない著者じゃないか！」

えー、私に足りないものは、熱意で補うつもりです。私は Web アプリケーションの大ファンであり、また時間を節約してくれたり、仕事を簡単にしてくれたりするツールやテクニックのファンでもあります。Laravel は両方の要求を満たしてくれますし、今まで出会ったソフトウェアの中で最も使いやすいものの一つでした。このフレームワークに対する熱意と情熱だけが、私を Laravel のコアチームに入れてくれた開発者と張りあわさせるのです。私達はこのコアチームを「カウンシル（評議会）」と呼ぶのが好きなんです。立派に聞こえるでしょう？

Laravel のコアチームにいるおかげで、新しいアイデアをフレームワークに加えるようなニュースを耳にする特権を持っています。またフレームワークへ貢献することにより、基本的なコードの改善を常に知ることができる立場にいます。こうしたことからこの本をアップデートできますし、フレームワークのリリース毎に新しい機能について書き加えるつもりです。

私自身は話題から外れることを好みませんが、この手の書籍ですと著者は短いパラグラフで書くことが義務付けられているようですので、短く、面白く書こうと思います。私はウェールズの海岸に住んでいます。（アメリカの皆さんにすれば、イングランドの横の国でしょう。）Aberystwyth の大きな公的機関で働いています。私は Laravel フレームワークをより良く改善するためには、時間を惜しみません。そうそう、前にも言いましたが、私は自分が文章の天才だと主張しません。正直なところ、ひどいものです。他のプログラムの書籍のように、ファンシーに書くつもりもありません。あなたのような本当の人間に対し、「話している」ように書くつもりです。あなたも、本当に話し返しているように思えてくることでしょう。私の Laravel に対する情熱が、私の一般的な英語（ひどいものです。私はウェールズの人間ですから。）を補ってくれるように願っています。これ以上、みなさんが私の思いを知る必要はありません。興味を Laravel に切り替えてください。

Laravel は PHP の世界に新しい風を吹き込みました。PHP プログラム言語はひどい関数名で有名であり、一方 PHP 開発者はそれを愛するように教育されてきました。文法も最新の日本の言語で比べるとやや劣ります。そうです、Ruby ですよ！

幸福なことに Laravel はこれを変えてくれました。実際…Laravel の文法は（PHP 自身の文法に基づいているのであっても）とても記述的で綺麗であり、Ruby よりも読みやすいと私は確信しています。コンパクトすぎず、一方で英語の文章のようにとは行きませんが、プログラマーの目からみれば、詩のように読めるでしょう。

> だけど、Dayle さん…

おや。あなたは心をしびれさせてくれる楽しいジュースにお金を払ったほうが良かったんじゃないかと心配し始めましたね。

> Laravel はフレームワークであって、言語ではないですよ！

そのとおりです。私の話を理解してくれていますね。Laravel は言語ではありませんが、そうであっていけないわけでもありません。私達は PHP が好きです。どうか、自分の心に正直になってください。私達は PHP の醜さに付き合い、鍵カッコやセミコロンをタイプするのを楽しんでいるのです。Laravel は単にショートカットを追加し、もしくはより楽になる経験をさせてくれるために、最前面でコードを偽装してくれているのです。

このフレームワークを使う喜びは、フレームワーク全体で一貫している表現的なメソッドによりもたらされると、私は思っています。`Str::upper()`。これの意味がわからないと、私に言ってください。これが意味をなさないと、私に言ってください。

> できませんよ。

そうですね。私も同感です。まあ、Laravel を素晴らしくすることについてであれば、一日中でも私は話せます。さあ、Eloquent を見てみてください。今までで最高の ORM です。これが最初に私をこのフレームワークへ心を向かわせた機能です。

ルーティングにクロージャーを使用することもできますが、望んでいる方にはこれが唯一の Laravel を使う理由でしょう。これも素晴らしいものです！

使用する前にライブラリーをロードする必要は無いことも…そうです。あなたも私の言っていることが正しいと思うことでしょう。これは後ほど実例でもお見せします。

ここで数多くの素晴らしい機能を説明する必要は無いでしょう。一番良い方法は、結局学び始めてもらうことだと思います。もし、心をしびれさせてくれるジュースの代わりに、この本にお金を払ってもらえているのでしたら、あなたのこのフレームワークに対する興味はピークに達していることでしょう。先に進みましょう！

1 さあ、始めましょう

Laravel[1]はTaylor Otwell[2]により書かれた PHP 5.3 ベースの Web アプリケーションフレームワークです。PHP 5.3 の機能を活かすように心がけ、設計されています。5.3 の機能と表現豊かな構文を組み合わせており、人気が上がってきています。

この本で、Laravel のインストールから説明していきますが、皆さん、これが簡単であると賛成してくれるでしょう。

（Laravel に限らず）PHP フレームワークを使用するためには、PHP が動作する Web サーバーが必要です。私はローカルな開発マシン上の Web サーバーにインストールすることをお勧めします。そうすれば、変更があるたびにいちいちアップロードすること無くテストができますからね。

この章はあなたが以下の条件を満たしていることを前提としています：

PHP5.3+ とアパッチベースの Web サーバーの設定ができる
サーバーのファイルシステムに慣れており、ファイルの移動やコピーができる
Apache の設定ファイルにアクセスでき、変更できる

他の Web サーバーを利用している方は、以下で説明している内容をどうやって実現するのか、検索すればネット上にたくさん記事を見つけられるでしょう。

最初にフレームワークのソースコードをコピーしなくてはなりません。Laravel.com[3]へ行き、大きなオレンジ色のダウンロードボタンを押してください。セキュリティーを強化するため、Web ルート以外でパッケージのコンテンツを解凍することをお勧めします。どこに展開したか、心にしっかりと留めておいてください。（もしくは、付箋を見つけて下さい！）

この時点で、フレームワークを思い通りに動かすために、２つの選択肢があります。おすすめは最初の方法です。「実働」環境の様に、フレームワークをインストールすることにより、設定ファイルで様々な設定ができます。しかしながら、２番目の方法は、開発サーバー上で動かす際、大抵のプロジェクトで時間がかかりません。

1.1 方法 1 仮想ホストを作成する

Apache の設定ファイルを新しく作る方法です。一般的な方法は、デフォルトで読み込まれる Apache の config.d フォルダーに” 自分のプロジェクト.conf” を作成します。より詳しく知るには、自分の環境で使用している Web サーバーのドキュメントを読んでください。

新しい設定ファイルに、以下の仮想ホストの設定内容を貼り付けるか、打ち込んでください。

[1] http://laravel.com
[2] https://twitter.com/taylorotwell
[3] http://laravel.com

```
<VirtualHost 127.0.0.2>
        DocumentRoot "/path/to/laravel/project/public"
        ServerName myproject
        <Directory "/path/to/laravel/project/public">
                Options Indexes FollowSymLinks MultiViews
                AllowOverride all
        </Directory>
</VirtualHost>
```

IP アドレスには、現在使用していない値を指定します。（127.0.0.1 を使用するのはよくありません。これはループバックのアドレスです。また、他の目的にも使用しているでしょう。）2 つのパスを Laravel のソースパッケージにある public フォルダーを示すように変更してください。では、Web サーバーを再起動しましょう。

次に、新しいローカル DNS エントリーが仮想ホストのプロジェクトを示すように作成します。最初に hosts ファイルを開きましょう。通常、Windows では、c:\windows\system32\drivers\etc\hosts に存在します。unix ベースの OS では、/etc/hosts になります。

次の一行を追加しましょう。使用している IP アドレスとサーバー名は仮想ホストの設定で指定したものです。

```
127.0.0.2               myproject
```

では、http://myproject へアクセスしてみましょう。ブラウザに、Laravel の Welcome ページが表示されます。

1.2 方法 2 パブリックフォルダーのシンボリックリンク

Unix ベースのシステムでシンボリックリンクに慣れているのでしたら、この方法が明らかにもっと簡単です。

解凍した Laravel のソースコードの中に（どこだったのか覚えていますよね？）、”public” という名前のサブフォルダーが存在しています。このフォルダーは Laravel のブートストラップファイルと公開できるリソースを含んでいます。このフォルダーを Web ルート（多分/var/www/html/でしょう）にシンボリックリンクします。

シンボリックリンクを行うために、お好きな端末で以下のコマンドを叩いてください。パスは必要に応じて変更してください。

```
ln -s /laravel/の/public/folder/までの/パス /web/ルート/サブディレクトリー/までの/パス
```

例えば：

```
ln -s /home/dayle/laravel/myapp/public /var/www/html/myapp
```

注意：直接 **Web** ルートへシンボリックリンクを張ることも可能です。しかし、多くのプロジェクトを実行できるので、サブディレクトリーとしてリンクを張ることをお勧めします。

では、http://localhost/myapp へアクセスしましょう。Laravel の Welcome ページが表示されます。

1.3 話の本線に戻りまして……

この時点で、Laravel の Welcome ページをご覧になっていますね。そうならば……

おめでとうございます！ 真新しい Laravel のプロジェクトが用意でき、コーディングする準備が整いました。

次のチュートリアルでは、Laravel のプロジェクトの構成について学びましょう。重要なファイルとフォルダーについて説明します。

もし、この本で取り扱っている話題に混乱してしまったら、手助けを得るために以下のリンクが役に立つでしょう。もしくは、シンプルにDayleRees.com[4]で新しいコメントを投稿してください。

- Laravel Web サイト[5]
- Laravel ドキュメンテーション[6]
- Laravel API[7]
- Laravel フォーラム[8]

成長し続けているコミュニティーに参加して見ませんか？IRC クライアントで”irc.freenode.net:6667” へ接続し、”#laravel” チャンネルに参加してください！

[4] http://daylerees.com
[5] http://laravel.com
[6] http://laravel.com/docs
[7] http://laravel.com/api
[8] http://forums.laravel.com

2 プロジェクト構成

Laravel のソースパッケージは様々な多くのディレクトリーから構成されています。どのように動作しているのかを深く理解できるよう、プロジェクト構成を確認して行きましょう。様々な Laravel の機能を説明するために、いくつかの新しい言葉を使用するので、混乱するかも知れません。もう、そうなっているかも知れませんが、しばらく我慢してください。これ以降のチュートリアルで詳細に触れていきます。

2.1 ルートディレクトリーの構成

最上位のディレクトリー構成をざっとご覧ください：

```
/application
/bundles
/laravel
/public
/storage
/vendor
/artisan （ファイル）
/paths.php （ファイル）
```

では、それぞれの項目を見ていきます。

/application

ここにあなたが開発するアプリケーションのコードの大半を設置します。ルーティーング、データモデル、ビューも含まれます。最も多くの時間をここで過ごすことになるでしょう！

/bundles

バンドルは Laravel のアプリケーションです。あなたのアプリケーションの独立した部分を設置してもいいですし、リリースされ、ダウンロードされた共通コード[1]を置くこともできます。Artisan により新しいバンドルをインストールすることにより、あなたの要求に合うように、Laravel の機能を拡張できます。

面白いことに、”**/application**” ディレクトリーもバンドルの一つです。デフォルトバンドルとして知られています。これが意味するのは、**/application** で利用できることは、バンドルの中でも利用できということです！

/laravel

ここにはフレームワークのコアファイルが存在します。リクエストにより実行されるファイルです。このディレクトリーに関わることはまず無いでしょうが、クラスやメソッドがどのように動作しているのかを知るために、コードを見てみるのは役に立ちます。もしくは、Laravel の API[2]をチェックするために、調べることもできるでしょう。

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

/public

ここは Web サーバーがアクセスする場所です。Laravel フレームワークとルーティングを起動する、ブートストラップである index.php を含みます。CSS や Javascript、画像ファイルなどのような、アクセスされる資源を置く、公開ディレクトリーでもあります。”laravel” サブディレクトリーはオフラインのドキュメンテーションを正しく表示するために必要なファイルを含んでいます。

/storage

この strage ディレクトリーは、例えば Sessions クラスや Cache クラスのようなドライバーなど、ファイルシステムを利用するサービスが、ファイルを保存するために使用します。このディレクトリーに対し、Web サーバーによる書き込みの許可を与えなくてはなりません。Laravel アプリケーションを構築するために、このディレクトリーに関わる必要はありません。

/vendor

vendor ディレクトリーは Laravel により使用されますが、フレームワーク作成者や協力者により書かれていないコードが置かれます。このフォルダーには、オープンソースのソフトウェアや、Laravel の機能に協力するソフトウェアが置かれます。

/artisan（ファイル）

Artisan は Laravel のコマンドライン・インターフェイスです。コマンドラインによる多くのタスク[3]を実行してくれますし、自分のタスクを作成することもできます！Artisan を実行するためには、以下のようにタイプしてください：

```
php artisan
```

/paths.php（ファイル）

このファイルはフレームワークにより利用され、上記で説明した重要なディレクトリーへのパスを決定します。path() による、こうしたパスへのショートカットを提供するためにも利用されます。このファイルは変更すべきでないでしょう。

2.2 Ａｐｐｌｉｃａｔｉｏｎディレクトリー構造

上記で述べた/application で全ての興味深いことが起きます。ですから、ディレクトリーの構造を見ていきましょう。

[3] http://laravel.com/docs/artisan/commands

```
/config
/controllers
/language
/libraries
/migrations
/models
/tasks
/tests
/views
/bundles.php （ファイル）
/routes.php （ファイル）
/start.php （ファイル）
```

では、それぞれの項目を詳細に確認しましょう。

/config

config フォルダーはフレームワークの振る舞いを変更する多くの設定ファイルが置かれています。フレームワークを動作させるために、インストール時に設定が必要なファイルは、最初からありません。ほとんどの設定ファイルは PHP のキーと値の配列をリターンしています。一部は Laravel のコアクラスの内部動作を自由に変更するために、値とクロージャーのペアであることもあります。

/controllers

Laravel は 2 つのルーティング方法を提供しています。コントローラーを使用するものと、ルートを使用するものです。このフォルダーはアプリケーションの基本ロジックや、データーモデルに関連すること、ビューファイルを読み込むコントローラークラスを含みます。他のフレームワークから移行してきたユーザーが慣れ親しんでいるアイデアを提供しようと、後日コントローラーが付け加えられることもあるでしょう。どんなものが後から思い付きで付け加えられようと、ルートとクロージャーを利用してどんなアクションの実行も、Laravel のパワフルなルーティングシステムにより可能です。

/language

このディレクトリーは Laravel 上に書かれたアプリケーションのローカライズを簡単に実現するため、文字列を含んだ PHP ファイルが設置されます。デフォルトでは、英語のペジネーションとフォームのバリデーションが用意されています。

/libraries

libraries ディレクトリーは、アプリケーションに追加の機能を組み込むために、クラスがひとつだけの PHP ライブラリーを「放り込む」ために使用します。大きなライブラリーは、ここに放り込む代わりに、バンドルを作成することが推奨されています。この libraries フォルダーは start.php により、起動時にオートローダーに追加されます。

/migrations

migrations フォルダーは Laravel により、アプリケーションのすべてのバージョンで同期を保つために使用される、現在のデータベースのスキーマをアップデートしたり、値を設定したりする PHP クラスによって構成されています。マイグレーションファイルは手動で作成してはなりません。ファイル名にはタイムスタンプが組み込まれ、Artisan コマンドラインインターフェイスで、php artisan migrate:make

/models

モデルはプロジェクトのデーターを表現したクラスです。通常これが意味しているのは、データベースの形式か他のデータを統合することを意味しています。Laravel は一般的なデーターベースプラットフォームを操作する 3 つの手法を提供しています。SQL ビルダーを含んだ'Fluent' と名付けられた[4]、SQL クエリーを PHP のメソッドチェーンで作成する手法、Eloquent ORM[5]を使用することで、テーブルを PHP オブジェクトとして表す手法、そして皆さんが今まで使用してきた、プレーンで古い、SQL クエリーそのままの手法です。Fluent と Eloquent は両方共に似たような文法を使用しており、変更しやすくなっています。

models ディレクトリー中のファイルは start.php により自動的にオートロードされます。

/tasks

tasks ディレクトリーにクラスを作成することにより、カスタムタスクを Laravel artisan コマンドラインインターフェイスに追加できます。タスクはクラスとメソッドで表されます。

/tests

tests フォルダーはアプリケーションのユニットテストを保存する場所として提供されています。もし、あなたが PHPUnit を利用しているのでしたら、Laravel artisan PHP コマンドラインインターフェイスを利用することで、全てのテストを一度に実行することができます。

/views

views ディレクトリーは、コントローラーやルートで使用される、HTML テンプレートファイルを設置します。拡張子は.php を使用してください。.blande.php 拡張子を利用すれば、後ほど説明する、Blade テンプレートライブラリーを使用して、パースすることも可能です。

/bundles.php（ファイル）

あるバンドルを有効にするためには、bundles.php の中の配列に、そのバンドルを追加するだけです。キーと値、名前と配列ペアでそのバンドルの追加オプションを定義することもできます。

/routes.php（ファイル）

この routes ファイルは、ルートを Laravel による適切な出力にマッピングするメソッドにより構成されています。この主題は以降の章で徹底的に説明します。また、このファイルはエラーページを含む様々なイベントの宣言や、ビューコンポーサー、ルートフィルターを定義するためにも使われます。

/start.php（ファイル）

start.php は"/application" バンドルのために、オートロードのディレクトリの指定、設定ファイルの読み込み、その他とても便利な機能の準備作業を行います! このファイルへ、自由になんでも付け加えてください。

次の章ではコントローラーを使用したルーティングを説明します。

[4] http://laravel.com/docs/database/fluent
[5] http://laravel.com/docs/eloquent

3 コントローラーの使用

このチュートリアルでは、複雑な別の要素にかかわらずに、Laravel のルーティングシステムの動作をデモするため、シンプルな複数ページの Web サイトを作成しましょう。

前の章で述べた通り、Web リクエストをあなたのコードにルートするために 2 つの選択肢が用意されています。コントローラーとルートです。他のフレームワークから移ってきた人たちにはお馴染みな、コントローラによるルーティングをこの章で学びましょう。

3.1 コントローラーのルーティング

それでは、コントローラーを見てみましょう。

```
// application/controllers/account.php
class Account_Controller extends Base_Controller
{

        public function action_index()
        {
                echo "これはプロフィールページです。";
        }

        public function action_login()
        {
                echo "これはログインフォームです。";
        }

        public function action_logout()
        {
                echo "これはログアウトアクションです。";
        }

}
```

コントローラはあなたの Web サイト、Web アプリケーションの、ある 1 セクションを表す PHP クラスです。メソッドもしくは「アクション」とは個々のページか、HTTP リクエストの最終地点を表します。

上記の例では、Account コントローラーは Web サイトのユーザーセクション、プロフィールページ、ログインページ、ログアウトページを表しています。コントローラー名の最後には `Controller` が、アクション名の先頭には `action` が付くことに、注意してください。コントローラーは `Base_Controller` か、`Controller`、もしくは別のコントローラークラスを拡張しなくてはなりません。

コントローラーは `application/controllers` フォルダーに小文字のファイル名で、コントローラー名と同じ名前で作成します。上記のコントローラーでしたら、以下の場所に保存します。

```
/application/controllers/account.php
```

コントローラーを使用する前に/application/routes.php に登録する必要があります。次の一行を追加してください。

```
// application/routes.php
Route::controller('account');
```

controllers ディレクトリーのサブフォルダーの中にコントローラーを保存する場合、シンプルにディレクトリーをピリオドで区切って表します。こんな風にです：

```
// application/routes.php
Route::controller('in.a.sub.directory.account');
```

もし、コントローラーがバンドルの中に存在するならば、二つ続けたコロンの前にバンドル名を指定して下さい：

```
// application/routes.php
Route::controller('mybundle::account');
```

では、表示してみましょう。

```
http://myproject/account/login
```

「これはログインフォームです」と表示されます。その理由はこのコントローラーが Route クラスの中でマッピングされたからです。最初のセグメント（スラッシュで区切られた）はコントローラーを指定する URL で、２番目のセグメントは（もちろん、スラッシュで区切られています）アクションを指定しています。

簡単に言えば、/account/login が、Account_Controller->action_login() にマップされ、そのメソッドの結果が表示されたわけです。

今度は/accout を代わりに呼んでみましょう。

```
これはプロフィールページです。
```

どうしてこうなったのでしょうか？index アクションは特別なアクションです。これは URL でアクションが指定されない場合に呼び出されます。ですから上記の場合には、次の URL として呼び出されるわけです。

```
/account/index
```

3.2 パラメーターを渡す

このシンプルなルーティングは興味深いですが、シンプルな PHP ウェブサイトができる全てのことを私達に教えてくれてはいません。

もうちょっと動的なルーティングに挑戦しましょう。コントローラーアクションにパラメーターを追加し、URL のセグメントで指定した追加データーを渡してみましょう。welcome アクションをコントローラーに追加しましょう。

```
// application/controllers/account
public function action_welcome($name, $place)
{
        echo "{$place}へようこそ、{$name}さん！";
}
```

今回のアクションパラメーターはメソッドのパラメータです。ですから、上記のコードは理解できるでしょう。/account/welcome/Dayle/Wales のルートで呼び出してください。

```
Wales へようこそ、Dayle さん！
```

データーに対する CRUD アクションを可能にするために、パラメーターはリソースの識別子を渡すために使用できます。もしくはあなたが考えることなんでもです！ご覧のとおり、アクションに対して柔軟な使用ができるようになっています。

> 注意：**URL** でアクションパラメーターに割り付ける値の指定は、オプションにできます。

3.3 ビューを使用する

コントローラーから echo で出力しても結果は表示されますが、エレガントな手法とは言えません。もしあなたが Leravel を学ぶことに興味をお持ちであれば、エレガントな解決法こそ、あなたをここに引き寄せたものでしょう。MVC の精神は、表示に関するレイヤーはログインアプリケーションから切り離し、「ビュー」というパターンの一部分にすることを提案しています。

Laravel では、ビューはこれ以上簡単にできないほど簡単です。単に HTML テンプレートを/application/views/ディレクトリーの下に小文字のファイル名、拡張子を.php で保存します。例えば：

```
/application/views/welcome.php
```

その内容は：

```
<h1>Holla!</h1>
<p>This is the welcome action of the account controller.</p>
```

ここで、Welcome アクションから呼び出したビューのリターン値が必要になりました。Laravel は美しい表現的な方法でこれを行います。御覧ください：

```
// application/controllers/account.php
public function action_welcome($name, $place)
{
        return View::make('welcome');
}
```

私の読者のようにおたくっぽい人たちであれば、このコードは Leravel が、application/views/welcome.php ファイル（ここでは拡張子も必要なく）からビューオブジェクトを作り (make)、welcome アクションの結果としてリターンしているんだろうと気づかれていることでしょう。

それと make() メソッドは、application/views ディレクトリーへビューを探しに言っていることも分かっていると思います。もし、ビューに絶対パスを指定したい場合は、例えば path: /path/to/my/view.php のように、先頭に path: を付けてください。

では、/account/welcome/Dayle/Wales を訪れてみましょう。ビューファイルで定義した Web ページが出迎えてくれるでしょう。

コントローラーの指定と同様に、サブフォルダーとバンドルからでもビューを参照し、使用できることを覚えておいてください。

うーーん。私にはあなたが何を考えているか分かりますよ。この Welcome メッセージは全然動的でないということでしょう？これにとりかかりましょう。アクションパラメーターをビューに渡しましょう。with() メソッドを使用し、アクションの中で Leravel のエレガントなメソッドチェーンを使用できます。やってみましょう！

```
// application/controllers/account.php
public function action_welcome($name, $place)
{
        return View::make('welcome')
                ->with('name', $name)
                ->with('place', $place);
}
```

with() メソッドを利用すれば、どんな値でも（オブジェクトでも）ビューに渡せます。ビューでアクセスするため「ニックネーム」を付けています。この例では、パラメーター名と同じニックネームを使用していますが、好きな名前でかまいません！

では、渡されたデーターをビューで使用しましょう：

```
<h1>いらっしゃいませ！</h1>
<p><?php echo $place; ?>へようこそ、<?php echo $name; ?>さん！</p>
```

これで、前のように動的にアクションが動きました。表示は少し良くなりました。ビジュアルレイヤーと全てのロジックを切り離し、きれいなコードになりました。

複数の `with()` メソッドを使用する代わりに、キーと値のペアの配列を `make()` の第 2 引数として渡すこともできます。これはコードが短くなりますが、同じ結果になります。例をご覧ください：

```
// application/controllers/account.php
public function action_welcome($name, $place)
{
        $data = array(
                'name'  => $name,
                'place' => $place
        );

        return View::make('welcome', $data);
}
```

注意：私はビューに渡す配列として”**$data**” を使うのが好きですが、あなたはお好きな配列名を使用できます！

このあとのチュートリアルで、Blade テンプレート、ビューのネスト、高度なテンプレートのオプションなど、ビューの詳細を説明します。

3.4 ＲＥＳＴコントローラー

REST フルな Web アプリケーションは、個々の HTTP 変数の意味に応じ、適したデーターを使用し、対応することができます。これはアプリケーションにパブリック API を作成する場合に、とても便利です。

Laravel を使えば、REST フルなコントローラーアクションを使用するとこで、個別の HTTP 変数に応じたコントローラーアクションを実現できます。実例をご覧ください。

```
// application/controllers/home.php
class Home_Controller extends Base_Controller
{
    public $restful = true;

    public function get_index()
    {
        //
    }

    public function post_index()
    {
        //
    }

}
```

シンプルに boolean 型の public class 属性で、名前が $restful という変数を作り、true を設定し、action_の代わりに、対応する HTTP メソッドで始まるアクション名を付けます。

一般的な HTTP メソッドは、GET、POST、PUT、DELETE です。

3.5 Ｂａｓｅコントローラー

Base_Controller は編集することができますし、それを拡張して他のコントローラーを作ることで、あなたの作成するコントローラー、デフォルトアクションの index、クラスの値など全てに対し、お好きなようにグローバルな機能を付け加えることもできます。

Base_Controller は/application/controllers/base.php です。

もし、Base_Controller を使いたくないなら、Controller クラスを拡張することもできます。

3.6 高度なルーティング

これで/コントローラー/アクション/パラメーター/パラメーター/... 形式をコントローラーにマッピングできるようになりまた。しかし、この形式だけに限定されたくありませんよね。では一歩抜け出しましょう。前にコントローラーの宣言を routers.php で行いました。以下のコードに置き換えてください。

```
//application/routes.php
Route::get('superwelcome/(:any)/(:any)', 'account@welcome');
```

これが表していているのは、superwelcom/(:any)/(:any) の形式で GET によるリスエスト全部を account コントローラーの welcome アクションに送りなさいという意味です。

(:any) セグメントは、パラメーターのプレースホルダーで、指定された順番に送られます。(:num) を使用すると数字のみに一致します。(:any?) を使えば、オプションのセグメントになります。

それでは、/superwelcome/Dayle/Wales にアクセスしてもらえば、私達の可愛いビューページが表示されるでしょう！

ルートを定義する有利な点は、URL を好きな順番、好きなフォーマットに設定できることです。例えば、このように設定することもできます。

```
//application/routes.php
Route::get('superwelcome/(:any)/(:any)', 'account@welcome');
Route::get('welcome/(:any)/to/(:any)', 'account@welcome');
```

これで 2 つの異なったルートで同じ結果のページが表示されます。

重要なのは routes.php の中で、先に定義したルートが高いプライオリティを持っているということです。次の例を見て下さい：

```
// application/routes.php
Route::get('(:any)/(:any)', 'account@welcome');
Route::get('welcome/(:any)/to/(:any)', 'account@welcome');
```

2 つ目のルートが実行されることはありません。最初のルートの (:any) は 2 番目の welcome に一致してしまいます。これは Laravel を始めた人が大抵間違えるポイントです。しっかりと、ルーティングのプライオリティを確認してください。

次のチュートリアルでは、ルーティングについてもっと深く追求し、コントローラーの代わりにクロージャーを使用するルーティングを紹介します。

4 クロージャーによるルーティング

この章ではコントローラーとアクションの代わりにクロージャーを使用したルティーングを使ってみましょう。まだコントローラーを使用した前の章を読んでないのでしたら、先に読んでおくことをおすすめします。この章の内容は、前の記事の知識をベースとしています。

このフレームワークの URL にクロージャーをマッピングできるため、「クラスのゴミ」を全く作らず、ロジック構成を綺麗に保てます。クロージャーは名前を持たない関数のことで、`function() {}` の形式です。他の変数に代入することもでき、通常の変数として取り扱えます。クロージャーに関する詳しい情報は、PHP の API 記事[1]を御覧ください。

4.1 クロージャー

クロージャーを使用したルーティングを見てみましょう。

```
// application/routes.php
Route::get('/', function()
{
        return View::make('home.index');
});
```

ここでは、Web アプリケーションのルート URL に対する HTTP メソッドの GET リクエストに、クロージャーで単にビューオブジェクトを返しているだけです。Laravel の welcome ページが表示されます。

ルートのスラッシュは、ルートページだけに必要です。他の全てのルートには必要ありません。例えば：

```
// application/routes.php
Route::get('account/profile', function()
{
        return View::make('account.profile');
});
```

REST 対応のルートはメソッド名の通りですが、`Route::any()` を使えば、全ての HTTP メソッドを取り扱えます。全オプションは次のとおりです。

[1] http://php.net/manual/en/functions.anonymous.php

```
// application/routes.php
Route::get();
Route::post();
Route::put();
Route::delete();
Route::any();
```

クロージャーにパラメーターを渡すためには、単に通常のビューの URI に対するプレースホルダーのように追加し、クロージャーにパラメーターを定義します。左から右に指定された順番通りに置き換えられます。例えば：

```
// application/routes.php
Route::get('user/(:any)/task/(:num)', function($username, $task_number)
{
        // $username は (:any) の値に置き換えられる
        // $task_number は (:num) で指定された整数に置き換えられる

        $data = array(
                'username' => $username,
                'task'     => $task_number
        );

        return View::make('tasks.for_user', $data);
});
```

使用出来るプレースホルダーは：

プレースホルダー	説明
(:any)	アルファベットと数字の文字列に一致
(:num)	数字に一致
(:any?)	? をつけるとオプション

4.2 リダイレクトと名前付きルート

使い方を知るまで、名前付きルートは、なんだか馬鹿らしく思えるものですよね？リダイレクトクラスを見てみましょう。これは他のルートへリダイレクトするために使用します。ビューを返す時と同じように使えます。

```
// application/routes.php
Route::get('/', function()
{
        return Redirect::to('account/profile');
});
```

素晴らしい！これ以上簡単にできないほどシンプルです。これだけです！それでは、名前付きルートを見ましょう。

```
Route::get('account/profile', array('as' => 'profile', 'do' => function()
{
        return View::make('account/profile');
}));
```

２番目の引数のクロージャーの代わりに、キーが do の配列で、クロージャーを指定しています。こうして全ての拡張情報をルートに追加します。

as キーは、ルートに与えるニックネームを指定します。これは、ルーティングの名前です。では前のリダイレクトを改善しましょう。

```
Route::get('/', function()
{
        return Redirect::to_route('profile');
});
```

これで、私たちのコードから汚い URI が無くなりました。なんて愛らしいんでしょう。全てのルートを参照するクラスとヘルパーで同じ手法でルートに名前を付けられます。これは本当にあなたのコードを美しく保ち、まるで本のように読みやすくしてくれます。さらに、もし後ほど特定のページの URI を変更しようと決めたのでしたら、全てのリンクとリダイレクトを書き直さないで済むのです！

4.3 フィルター

まあ、まあ。確かに…私はこのチュートリアルではルートについて説明すると言いました。しかし、ここが本当にフィルターについてお話する、良い機会なのです。ルーティングにも関係しています。ですから、見てみましょう。

フィルターはその名の通りのコードやテストのことで、ルーティングの前 (before) や後 (after)、その他フレームワークにより起こされるキーイベントで実行されます。Laravel は 4 つの特殊なルートフィルターがあり、application/routes.php の中にデフォルトとして用意されています。その部分を見てみましょう。

```
Route::filter('before', function()
{
        // アプリケーションに対する全てのリクエストの前に行うこと…
});

Route::filter('after', function($response)
{
        // アプリケーションに対する全てのリクエストの後に行うこと…
});

Route::filter('csrf', function()
{
        if (Request::forged()) return Response::error('500');
});

Route::filter('auth', function()
{
        if (Auth::guest()) return Redirect::to('login');
});
```

最初の 2 つは全てのリクエスト（もしくはルートやアクション）の前後にカプセル化されたクロージャーを実行します。クロージャー内で何を行うのかはお好きにどうぞ。ライブラリーを起動したり、データを「何か」に渡したり、あなたの想像力だけが限界です。これらの特殊なフィルターのおかげで、全てのルートそれぞれに割り付ける必要はありません。

`csrf` は`cross-site-request-forgery`[2]を防ぐために使われ、AJAX による呼び出し結果のルートのセキュリティー強化にも使用できます。

`auth` フィルターは全てのルートに適用され、laravel の認証システムで現在「ログイン」していないユーザーからのアクセスを防ぎます。

`csrf` と `auth` フィルターをあなたのルートに適用するためには、ただ新しい配列要素を 2 番目のパラメーターに指定するだけです。この様にです：

```
Route::get('/', array('as' => 'profile', 'before' => 'auth',
        'do' => function()
        {
                return View::make('account/profile');
        }));
```

キーにはルート実行前に適用する'before' と実行後に適用する'after' を指定できます。複数のフィルターを適用するには名前を縦棒 (|) で区切ります。例えば `auth|csrf` のようにです。

laravel 3.1 以降、フィルターにパターンマッチングで多くの'URI' リクエストに対応できるようになりました。以下のコードが使えます。

```
Route::filter('pattern: admin/*', 'auth');
```

これで"auth" フィルターを admin/で始まる全ての URI ルートに適用できます。

[2] http://en.wikipedia.org/wiki/Cross-site_request_forgery

4.4 ルートグループ

幅広い対象の多くのルートに同じフィルターを適用する場合、ルートグループというオプションを使えば簡単に実現できます。見て行きましょう。

```
Route::group(array('before' => 'auth'), function()
{
	Route::get('panel', function()
	{
		// 何か行う
	});

	Route::get('dashboard', function()
	{
		// 何か行う
	});
});
```

これで panel と dashboard ルートは、auth フィルターにより守られました。

ルーティングはシンプルにもできる一方で、必要に応じて複雑にもできます。多くのルートに共通するルールの重複を防ぐため、ルートグループを使ってください。そしてコードをドライ（Don't Repeat Yourself = 自分で何度も繰り返すな）に保ちましょう。

次の章では、リンクの生成を取り扱います。これである URI のページから、別のページへ移動できるようになりますね。幸運なことに、私達は本のページから、次のページへ移動する方法は既に知っています。

5 リンクとＵＲＬ

もしアプリケーションが１ページしか無ければ、少し退屈ですし、ユーザーがもしページを切り替えるのに、毎回 URI を全部タイプしなくてならなければ、直ぐにイライラすることでしょう。幸いなことに、今日ハイパーリンクがこの状況を救ってくれています。

もしあなたが過去の２０年間を地面の下で過ごしたのでないなら、ハイパーリンクは何であるかご存知でしょう。テクニカルな説明をして、退屈させたくありません。リンクを見る前に、Laravel が URL をどう処理するか取り上げましょう。

5.1 ＵＲＬの取得

最初に問題を取り上げましょう。フレームワークは独自の URL 構造を持っています。いくつかのフレームワークは inex.php を含んでおり、いくつかは含んでいません。別のものは複雑なルートを採用しています。ほとんどのケース、あなたの他の Web サイトでも、相対 URL が使われていますが、長期的な運用で問題を引き起こすことがあります。完全にフル URL を選べば、後日アプリケーションを引越すことを決めた時に、お好みのエディターで検索と置換をフル活用することになるのが分かるでしょう。

Laravel にこうしたハードワークはすべて任せてはいかがですか？Laravel はアプリケーションのフル URL を分かっていますし、URL リライトを使用しているかも判断します。あなたのルートであってもです。きっとあなたのベッドの下の箱についても知っているでしょう…URL クラスでサイト URL を生成し、この情報を生かしてください。

まず、Web サイトのルート URL を見つけましょう。`base()` メソッドがこれに使えます。

```
echo URL::base();
// http://myproject
```

いいですね！これでサイトへのフル URL が、最後に index.php があろうと無かろうと、取得できます。全ては今のセットアップで決まります。現在の URL はどうしましょうか。今現在、アクセスされているルートは取得できるのでしょうか？そうですよね！`current()` メソッドを使うだけです。

```
echo URL::current();
// http://myproject/this/page
```

デフォルトでは、Laravel は URL に付けられたクエリー文字列は取り除きます。クエリー文字列ごと全部含めた、アクセス URL を取得したい場合は `full()` を代わりに使って下さい。

```
echo URL::full();
// http://myproject/this/page?thing=stuff
```

ベース URL と現在の URL は簡単に扱えました。けど、他のルートやページへの URL が取得できたら便利ですよね。それをリンクを作成するのに使えます。

ルートへの URL を生成するには、`to()` メソッドが使えます。取得したいルートの URL を手にできますし、フルパスを指定するより、少しは簡単です。例えば：

```
echo URL::to('my/route');
// http://myproject/my/route
```

このページへ HTTPS プロトコルを使用し、セキュアにリンクを貼りたいのであれば、to_secure() メソッドを代わりに使用してください。

```
echo URL::to_secure('my/route');
// https://myproject/my/route
```

ポストへのルーティングで名前付きルートを学んだことを覚えていますか？もちろん、覚えていることでしょう！もう一回、見てみましょう。

```
Route::get('login', array('as' => 'login', 'do' => function() {
        // コードを書く
}));
```

配列のキーに as を使い、login という名前付きルートを作成しています。これは後で役に立つと言ってました。今こそ、名前付きルートが輝く時です。login という名前のルートへリンクを張りましょう。

```
echo URL::to_route('login');
// http://myproject/login
```

ワオ！とても綺麗で表現的であると、あなたも同意してくれることでしょう。では、ルートに引数を渡したい時は？簡単です。to_route() メソッドの第 2 パラメーターに配列で渡すだけです。こんな感じのログインルートを数秒イメージして下さい…

```
Route::get('my/(:any)/login/(:any)/page')...
```

これはひどいルートです。どうかこのような醜い URL を使わないでください。しかし、この例はポイントを示してくれます。to_route() メソッドへのパラメーターの渡し方を理解できます。Laravel は自動的に、与えられた配列の順番で、URL のプレースフォルダーを置き換え、正しいフル URL を返してくれます。美しい！

```
echo URL::to_route('login', array(5, 7));
```

上のメソッドが生成するのは：

```
http://myproject/my/5/login/7/page
```

素晴らしい！見た通りとてもクリーンです…このように複雑なルートを作成しない限りですが。また、後ほど URI を変更すると決めても、全てのリンクをアップデートする必要はありません。

これでルートはクリーンアップできます。しかし、コントローラーを忘れてはいけません。残して先へ進みたくはないでしょう。幸運なことに、コントローラーアクションにも、ナイスできれいなリンクを生成できます。to_action() メソッドを使うだけです。例えば：

```
echo URL::to_action('dashboard@home');
// http://myproject/dashboard/home
```

アットマーク (@) で分けて、コントローラ名とアクションを渡しているだけです。もし必要であれば、to_action メソッドの第 2 パラメーターとして、追加のパラメータを配列にして渡せます。

アセットはどう扱いましょう。例えば CSS スタイルシートです。ルートのページとは全く違う URL です。URL::to() は使用出来ません。なぜなら、index.php が含まれる可能性があるからです。

代わりに to_asset() メソッドが正しいリンクを生成してくれます。単に、スタイルシートへの相対パスを渡してください。残りの面倒は Laravel が見ます。

```
echo URL::to_asset('css/style.css');
```

この行が生成するのは：

```
http://myproject/css/style.css
```

これらのメソッドは既に便利ですが、Laravel はもっと先に行っています。より短く記述できるようにヘルパーメソッドが用意さており、ビューの中で使用すると素晴らしいのです。一覧をどうぞ。

ヘルパー	メソッド
url()	URL::to()
asset()	URL::to_asset()
route()	URL::to_route()
action()	URL::to_action()

5.2 リンクの生成

これで、サイトのへリンクを取得できました。次の論理的段階は、ハイパーリンクを生成するために、それらを使用することです。今のところ、あなたはこんなふうに使おうと思っているでしょう。

```
<a href="<?php echo URL::to('my/page'); ?>">My Page</a>
```

たしかにこれでも動きます。しかし、やや汚いですね。Laravel の中で、何かが汚いのであれば、そこにはもっと良い方法が用意されています。リンクも例外ではありません。

HTML クラスをリンク生成に利用しましょう。最初に、HTML クラスは何のためにあるかです。これは、すべての種類の HTML タグを生成するためにあります。

```
<?php echo HTML::link('my/page', 'My Page'); ?>
```

だいぶ良くなりました。結果を見てみましょう。

```
<a href="http://myproject/my/page">My Page</a>
```

もし、あなたが SEO 忍者であれば、title アトリビュートなしでリンクを生成するなんて、耐えられないでしょう。追加の配列をシンプルに渡してください。

```
<?php echo HTML::link('my/page', 'My Page', array('title' => 'My page!')); \
?>
```

こうなります：

```
<a href="http://myproject/my/page" title="My page!">My Page</a>
```

Laravel の優れている特徴には、メソッドのネーミングルールがあります。多くの `HTML::link` メソッドは、`URL::to` メソッドと似ています。ですから、覚えやすいのです。HTTPS を利用し、セキュアなリンクを張るには：

```
HTML::link_to_secure('my/page', 'My Page');
// <a href="https://myproject/my/page">My Page</a>
```

さらに `link_to_route` は URL ライブラリでやったのと同様に、名前付きルートのリンクを生成します。

```
HTML::link_to_route('login', 'Login!');
// <a href="http://myproject/login/page">Login!</a>
```

さらに、`link_to_action()` メソッドはコントローラーとアクションのペアに使えます。例えば：

```
HTML::link_to_action('account@login', 'Login!');
// <a href="http://myproject/account/login">Login!</a>
```

Laravel では、メールアドレスから'mailto' リンクを簡単に生成するメソッドも存在しています。見てみましょう。

```
HTML::mailto('me@daylerees.com', 'Mail me!');
// <a href="mailto:me@daylerees.com">Mail me!</a>
```

ナイスでクリーンです！

これで、どうやって URL とリンクを生成するのか理解できたでしょう。あなたのアプリケーションは段々と大きくなり、ルートは膨大となり、この惑星を食いつぶし、やがて宇宙さえ……

アプリケーションはもっと面白くなります！

6 フォーム

どんな Web アプリケーションでも、フォームは重要です。アプリケーションのフローをコントロールし、ユーザーの入力を受け取り、アプリケーションの機能的な結果を決める手助けをしてくれます。そして、私にとって、世界中で一番書きたい主題です。

私にとって（みなさんにとっても）幸運なことに、Laravel のフォームクラスは、一般的なフォーム要素を生成する便利なメソッドが提供されており、フォーム作成に関する多くのハードワークを考慮しています。簡単な Web フォームを一つ作り、フォームクラスを使ってみましょう。

6.1 フォームを作成する

```
// form.php
<?php echo Form::open('my/route'); ?>

        <!-- username 領域 -->
        <?php echo Form::label('username', ' ユーザー名'); ?>
        <?php echo Form::text('username'); ?>

        <!-- password 領域 -->
        <?php echo Form::label('password', ' パスワード'); ?>
        <?php echo Form::password('password'); ?>

        <!-- ログインボタン -->
        <?php echo Form::submit(' ログイン');

<?php echo Form::close(); ?>
```

時間を取り、フォームのソースを眺めてみましょう。こんなに綺麗なフォームは見たことがないでしょう。自分自身に叫んでください。どうぞ…私は待っていますから…

> こんな綺麗なフォームは見たことがないーーー！

そのとおりです。美しいです。私が間違ったことを教えていないことを確信してもらうために、生成されたソースも見て下さい。

```
<form method="POST" action="http://mysite/my/route" accept-charset="UTF-8">

        <!-- username 領域 -->
        <label for="username">ユーザー名</label>
        <input type="text" name="username" id="username">

        <!-- password 領域 -->
        <label for="password">パスワード</label>
        <input type="password" name="password" id="password">

        <!-- ログインボタン -->
        <input type="submit" value=" ログイン">

</form>
```

素晴らしい、うまくいっています！もちろん、うまくいきます。では、ひとつひとつどう動くのかを見て行きましょう。最初の行は `Form::open()` メソッドです。これにより、form open タグが生成されます。

```
<?php echo Form::open('my/route'); ?>
```

最初のパラメーターはフォームの送信先 URI です。２つ目のパラメーターはフォームの送信メソッドです。指定しなければ、Laravel は最も一般的な `POST` で送信します。

３つ目のパラメーターもオプションで、アトリビュート `=>` 値のペアの配列を渡し、`<form>` タグに追加のアトリビュートを付け加えることができます。例えば、ある Javascript にターゲットのフォームを知らせたい時に、('id' => 'myform') と配列で３つ目のパラメーターとして渡せば、id 要素を指定できます。

フォームをセキュアな URI(https) で送信したければ、`open_secure()` メソッドを `open()` の代わりに使って下さい。パラメーターは同じです。

もし、フォームからファイルのアップロードを行いたい時、`multipart/data` を使う必要がありますが、`open()` の代わりに `open_for_files()` を使用してください。これもパラメーターは同じです。

最後に、もしセキュアな URI で、しかもファイルアップロードを行ないたいなら、`open_secure_for_files()` メソッドを使ってください。これもまた同じパラメータです。`open_secure()` と `open_for_files()` を組み合わせた働きをします。

6.2 ラベルを付ける

次の行は `Form::label()` メソッドを使い `<label>` 要素を生成しています。最初のパラメーターは input 要素の名前で `for=""` アトリビュートとして記述される部分です。２番目のパラメーターは `label` 要素のコンテントです。追加の HTML 要素アトリビュートを３つ目のパラメーターとして配列で渡すこともできます。

6.3 Ｉｎｐｕｔの生成

次に input を生成しています。一般的な HTML 要素すべてを生成するために、これらのメソッドは役立ちます。上の例では `text()` と `password()` メソッドはそれぞれ、`<input type="text"…` と `<type="password"…` を生成します。

メソッドの最初のパラメーターは `name` アトリビュート要素の値です。２つ目はオプションで、デフォルト値を設定します。ここでも３つ目はオプションで、HTML アトリビュートの配列を渡せます。もう、パターンが読めましたか？

`textarea()` と `hidden()` フィールドも、同じパラメーターを用います。

チェックボックスは `checkbox()` メソッドを用い、生成します。最初のパラメーターはその要素の名前です。２つ目は値で、３つ目は論理型のオプションで、そのエレメントに最初からチェックを入れるかどうかです。４つ目のオプションパラメーターはアトリビュートで、実際……全ての input 機能は、最後の引数として、オプションでアトリビュートの配列を受け付けます。

```
<?php echo Form::checkbox('admin', 'yes', true,
        array('id' => 'admin-checker')); ?>
```

`radio()` メソッドはラジオボタンです。checkbox() メソッドと同じパラメーターです。

次はドロップダウンです。全てのフォームエレメントの中で一番厄介です。幸福なことに名前と「値 => ラベル」で選択肢の配列、それとオプションでデフォルトの selected をパラメーターで `select()` メソッドに渡してあげるだけです。それでドロップダウンが生成されます。例えば：

```
Form::select('roles', array(
        0 => 'User',
        1 => 'Member',
        2 => 'Editor',
        3 => 'Administrator'
), 2);
```

生成結果は：

```
<select name="roles">
        <option value="0">User</option>
        <option value="1">Member</option>
        <option value="2" selected="selected">Editor</option>
        <option value="3">Administrator</option>
</select>
```

素晴らしい！では、フォームを送信しましょう。

6.4 ボタンを生成する

submit() と button() は両方とも同じパラメーターを取るメソッドです。最初は HTML 要素の値 (value)、２つ目はアトリビュートの配列です。

```
Form::submit('Login');
Form::button('Do other thing!');
```

6.5 セキュアな入力

ドキュメントには書かれていませんが、一般的ではない入力フォームを生成する、追加のメソッドも用意されています。これらの入力は HTML5 の機能として付け加えられたもので、良いブラウザ（IE ではありません）で導入されつつあります。以下にリストに上げ、パラメーターを示します。

```
// 検索フィールドを作成
Form::search($name, $value = null, $attributes = array());

// メールアドレスだけを許すフィールドを作成
Form::email($name, $value = null, $attributes = array());

// 電話番号だけを許すフィールドを作成
Form::telephone($name, $value = null, $attributes = array());

// URL だけを許す
Form::url($name, $value = null, $attributes = array());

// ロッカスイッチによる数値フィールド
Form::number($name, $value = null, $attributes = array());

// 日付ピッカー
Form::date($name, $value = null, $attributes = array());

// ファイルアップロードに使用
Form::file($name, $attributes = array());
```

6.6 ＣＳＲＦトークン

もし組み込まれている CSRF フィルターを AJAX 送信フォームを守るために使いたければ、CSRF トークンを token() メソッドを使うことで、付け加えられます。例えば：

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

6.7 フォームマクロ

Laravel は多くの異なった input メソッドを提供していますが、もうちょっとカスタマイズしたい時には、どうすればいいのでしょう？幸運なことに、Laravel は input の生成コードを作るために'macro()'メソッドを用意しています。

まず input 名、次にクロージャーを `macro()` メソッドに渡し、input 生成コードを定義します。次のコードを見て下さい。

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

これで Form クラスで他の input と同様の方法で、靴のサイズフィールドを生成するメソッドができました。例えば：

```
<?php echo Form::shoe_size(); ?>
```

パラメーターを使いたいなら、単にクロージャーにパラメーターを追加するだけです。自分自身のユニークな input を生成し、楽しんでください！

7 入力を扱う

今やフォームの生成は理解しました。今度はフォームから送られてくる入力をどう扱うかを学びましょう。いつもの通り、Laravel はウルトラクリーンに入力データーを扱う手段を提供しています。$_POST、$_GET、$_FILE のような PHP の配列に関わる必要はありません。正直なところ、それらはひどく醜いですからね。

7.1 データーの要求

代わりに Input クラスを使いましょう。

```
$panda = Input::get('panda');
```

これで Panda が手に入りました！素晴らしい…既に多くを手に入れました。そうですパンダです！素晴らしい…既にたくさん手に入れています。不幸なことにウェールズの橋にある海岸を、レッサーパンダが荒らしています。どこにでもいて、今一匹をフットレストに使っています。エマがどうやってうまいこと表紙の写真を撮ったんだと思っているでしょうが、彼らは中国からやってきたのです！

とにかく、話を元に戻しましょう。Input クラスの get() メソッドについて覚えておく必要がある一つは、$_GET のデーターだけを参照しているのではなく、get() はナイスで記述的にな短いメソッド名ですが、全ての種類のデーターを取得できることです。Input クラスは get() で、$_POST‘も含め、全ての種類のリクエストデーターを手に入れられるようにしています。

要求されたデーターが存在しない場合、Input クラスは NULL を返します。get() メソッドの第 2 パラメーターを指定すれば、その値を代わりに返します。これが意味しているのは、undefined エラーが現れることを心配しなくて済むという事です。とても使いやすいんです！

```
$panda = Input::get('panda', 'Muffin');
```

これでパンダが名前を持っていなければ、マフィン（Muffin）と呼ばれるでしょう。あーー…。

もし、リクエストの配列全部を取得したければ、索引を省略してください。簡単です。

```
$morepandas = Input::get();
```

デフォルトでは、$_FILES 配列の値を get() は含みません。しかしながら、get() の代わりに all() を使用すれば、files も含めて取得できます。

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

もし、post データーが存在しているか調べたいけれど、値を取得する必要がないのであれば、エレガントで非常に記述的な has() メソッドを使ってください。論理型で結果を返してくれます。

```
$do_we_have_a_panda = Input::has('panda');
```

7.2 ファイル

$_FILES 配列の要素にアクセスするには、Input::file() メソッドを呼ぶだけです。例えば：

```
$file = Input::file('spoon');
```

もし単にファイルの属性を知りたいだけでしたら、最初のパラメーターにピリオドを付け、それからアトリビュートキーを付けてください。ファイルサイズを取得したいのであれば：

```
$size = Input::file('spoon.size');
```

> 注目：この文法は全ての多重配列にアクセスするために使えます。単純にピリオドで多重配列のインデックスを区切ってください。

ここでもまた、パラメーターを付けずメソッドを呼び出せば、ファイル全部の配列が取得できます。

```
$files = Input::file();
```

7.3 フラッシュデーター

フラッシュデーターは次のリクエストのために値を保存する便利な手法です。これは何度も表示されるフォームで便利なメソッドです。

全てのリクエストデーターをセッションにさっと送り (flash)、次のリクエストでアクセスできるようにするためには、シンプルに flash() メソッドを使ってください。

```
Input::flash();
```

現在のリクエストデーターの一部だけをフラッシュしたい時は、最初のパラメーターに only を渡し、２つ目のパラメーターとして、フラッシュしたいフィールド名を配列で渡してください。

```
Input::flash('only', array('betty', 'simon'));
```

これで、Betty と Simon を次のリクエストで取得できます。別の指定方法として、使いたくないフィールドの名前をリストにして、除外 (except) オプションを使い方法もあります。例えば：

```
Input::flash('except', array('uncle_bob'));
```

これで、尊大な魂を持ち、我々の国民的動物であるレッサーパンダを嫌っている、ボブおじさんはいなくなりました。

普通、ここで新しいリクエストに向かうために Ridirect::to() メソッドを使うのです。そこから記述的な Input::old() メソッドを使って直前のリクエストでフラッシュされた値を取得してみましょう。

```
$betty = Input::old('betty');
```

ご覧のとおり Betty はテレポートして現れました。あなたはフラッシュデーターはスタートレックのテレポート装置のように思えるでしょう。クラークの体をあるリクエストから、次のリクエストへ移動します。

ここでも、パラメーターを省略することで、フラッシュデーターの全ての配列を取得できます。

```
$people = Input::old();
```

had() メソッドをフラッシュデーターが存在しているかどうか、調べるために使えます。

```
Input::had('uncle_bob');
```

もちろん、私達の嫌いなボブおじさんはいません。

素晴らしいショートカットと記述的なメソッドがなければ、Laravel フレームワークとは言えません。実際に、この例を見てください。

```
return Redirect::to('party')->with_input();
```

この with_input() メソッドは、全てのリクエストデーターをフラッシュしてくれます。flash() メソッドと同様に only と except オプションを受け付けます。

```
return Redirect::to('party')->with_input('only', array('betty', 'simon'));
return Redirect::to('party')->with_input('except', array('uncle_bob'));
```

これでフォームデーターにアクセスできるようになりました。アプリケーションをもっとインターラクティブにできます！

8 バリデーション

バリデーションは多くのアプリケーションで最も重要な部分です。あなたは決してユーザーを信用してはいけません。彼らは、邪悪な Javascript をフォームで不正に使用することで、あなたの数週間をぶち壊す方法を計画しています。

彼らを勝たせてはいけません。私たちの美しいアプリケーションを破壊させてはなりません。ユーザーからの全ての入力をバリデーションしましょう。これで彼らは私達に手出しはできなくなります。

当然のごとく、Laravel は、それにふさわしい名前の `Validation` というライブラリーを持っており、ハードワークを担当してくれます。

8.1 バリデーションの準備

想像上のフォームを創造しましょう。目を閉じ、長くて入力フィールドがたくさんあるフォームをイメージしてください……あれ……どうしたら、もう一度あなたの目を開けられるのでしょう……？

そうです。想像上のフォームを思い、あなたが待つのに飽きて、目を開け、私のもとに戻ってくるまで、じっと待っていましょう。では、入力データをフォームから受け取りましょう。

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

通常は `get()` メソッドのまま使いたくありません。簡単な方法ですが、不必要な余計なデーターまで全部取得してしまいますからね。実際、オープンソースのコラボレーションサイトである github は膨大な課題の犠牲者になっています。チュートリアルでは、`get()` をシンプルに使いますが、あなたのアプリケーションでは必要なフィールドだけを配列に受け取ってください。

たぶん、入力配列はこんな感じでしょう。

```
array(
        'name' => 'John',
        'age'  => 15
)
```

では、アプリケーションにぴったり来るように入力フィールドをバリデーションしましょう。実際にバリデーションする前に、それぞれのフィールドに対しルールを作成する必要があります。validator クラスを使用し、配列の形式でルールを定義します。早速、見てみましょう。

```
$rules = array(
        'name' => 'required|min:3|max:32|alpha',
        'age'  => 'required|integer|min:16'
);
```

いいですね。ルールが用意できました。配列のキーはバリデーションする項目名、配列の値はいくつかのルールを縦棒 (|) で区切ったものです。

この場合、両フィールド共に required ルールが指定されています。ユーザー名は最低 3 文字 (min:3) で、最大 3 2 文字 (max:32)、”alpha” で英文字だけで構成されていることをチェックしています。

age フィールドは、整数値で最低 1 6 以上でなくてはなりません。ご覧のとおり、min ルールはコンテンツに合わせて、判断されます。とても頭がいいですね！

ご心配なく。後ほど、すべてのバリデーションルールを説明します。今は、バリデーションの動きに注目しましょう。では、どうぞ。

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

make() メソッドで validator オブジェクトを作成しました。入力配列を渡し、ルールの配列を渡します。では、バリデーション実行です！

```
if( $v->fails() )
{
        // バリデーション失敗時のコード :(
}
else
{
        // バリデーション成功時のコード！
}
```

見ての通り、fails() メソッドでバリデーション結果を調べます。ture であれば、バリデーション失敗、false であれば成功です。

もし、バリデーションをもっとポジティブな感じにしたければ、passes() メソッドを使ってください。ポジティブな値を返します。

```
if( $v->passes() )
{
        // // バリデーション成功時のコード！
}
else
{
        // バリデーション失敗時のコード :(
}
```

これでポジティブになりました。スパークル・ポニーと一緒に虹の上でダンスが出来ます。

8.2 エラー

もしバリデーションが失敗したなら、例えばユーザーが１６歳未満であったら（すいません、たぶんあなたのスパークルポニーと一緒に遊んでいることでしょう）、何が悪かったのか知りたいと思うでしょう。validator クラスは errors メッセージオブジェクトを提供してくれますので、必要な情報を簡単に見つけることができます。

errors オブジェクトは Input クラスと似たようなメソッドを持っています。では、使ってみましょう。特定のフィールドのエラーの配列を受け取りましょう。

```
$age_errors = $v->errors->get('age');
```

これで、age フィールドに関連するエラー全部の情報を配列で入手できます。

```
array(
        'The age must be at least 16.'
)
```

私自身は、ほとんどの場合、first() メソッドをビューの中で使っています。存在するなら配列の最初のアイテムを返し、もし存在しなければ、null を返します。例えば：

```
<?php echo Form::label('username', ' ユーザー名') ?>
<?php echo $errors->first('username') ?>
<?php echo Form::text('username') ?>
```

これで、もしこのフィールドのバリデーションエラーが起きたなら、表示されます。また、first() メソッドの第２パラメーターに出力フォーマットを指定することも可能です。

```
<?php echo $errors->first('username', '<span class="error">:message</span>'\
) ?>
```

美しい！

もしくは has() でエラーが存在するかどうかチェックできます。all() を使えばすべてのエラーを配列で受け取れます。

8.3 バリデーションルール

バリデーションのルールと目的のリストです。

required

フィールドが存在し、空文字でないことを確実にします。

alpha

文字（アルファベット）で構成されていなくてはなりません。

alpha_num

英数字で構成されていなくてはなりません。username に便利です。

alpha_dash

英数字とダッシュ、下線で構成されていなくてはなりません。URL を保存するときに便利です。

size:5

文字列の場合、５文字である必要があります。数字の場合、値が５である必要があります。

between:5,10

文字列の場合、文字列の長さが５から１０文字である必要があります。数字の場合、値が５から１０文字である必要があります。

min:5

文字列の場合、文字列の長さが５文字以上必要です。数字の場合、値が５以上である必要があります。ファイルの場合、ファイルサイズが５キロバイト以上である必要があります。

max:5

文字列の場合、５文字以下である必要があります。数字の場合、値が５以下である必要があります。ファイルの場合、ファイルサイズが５キロバイト以下である必要があります。

numeric

値は数字でなくてはなりません。

integer

値は整数である必要があります。

in:red,green,blue

リストで指示された値が指定される必要があります。

not_in:pink,purple

指示された値と一致しない必要があります。

confirmed

確認フィールドと値が一致する必要があります。フィールド名は後ろに`_confirmation`がついたものと比較します。

accepted

値は”yes” か 1 でなくてはなりません。チェックボックスのバリデーションに便利です。

same:age

指定されたフィールドと一致する値が指定されなければなりません。

different:age

フィールド値は、指定された値と異なっていなくてはなりません。

match:/[a-z]+/

指定された正規表現と一致しなくてはなりません。

unique:users

これは、私のお気に入りです。validator は `users` データベーステーブルを検索し、そのフィールド名と同じカラムに、入力された値が存在していてはいけません。username やメールアドレスの二重登録を防ぐのに便利です。

他のカラム名を指定したければ、２つのパラメーターとして渡してください。

```
unique:users,nickname
```

`id` を３つ目のパラメーターで渡すことで、unique ルールを無視することができます。

```
unique:users,nickname,5
```

exists:colors

unique と逆の働きをします。データベーステーブルにその値が存在していなくてはなりません。ここでも２番目のパラメーターに他のカラム名を指定できます。

before:1984-12-12

`before` ルールで指定された日付より前の日付を指定する必要があります。

before と after フィルターは strtotime() を比較に利用します。これを利用すると絶妙なトリックが使えます。こんな風に……

```
before:next Thursday
```

残念ながら、この機能を付け加えた一人が私です。ですから、もしこれがあなたにショックを与えたなら、どうか大声私に向かって叫んでください……すいません！

after:1984-12-12

`before` と似ています。日付は `after` ルールで指定された日付より後でなくてはなりません。

email

メールアドレスとして有効でなくてはなりません。

url

URL のフォーマットに一致している必要があります。

active_url

値はアクティブな URL と一致していなくてはなりません。`checkdnsr` を URL がアクティブであるかの確認に使用します。

mimes:png,mp3

値は `$_FILE` の値の MIME タイプで、ファイルの拡張子と一致している必要があります。`config/mimes.php` に MIME タイプを追加することができます。

image

アップロードファイルはイメージでなくてはなりません。

8.4 カスタムエラーメッセージ

デフォルトのエラーメッセージは十分にエラー内容を表していると思いますが、あなたのクライアントは別の考えを持っているかも知れません。要求に応じてエラーメッセージをカスタマイズ方法を理解しましょう。

application/language/en/validation.php を直接編集して、バリデーションエラーメッセージを編集できます…

```
// ...
"after"          => "The :attribute must be a date after :date.",
"alpha"          => "The :attribute may only contain letters.",
"alpha_dash"     => "The :attribute may only contain letters, "
        . "numbers, and dashes.",
// ...
```

Laravel は:attribute マーカーをフィールド名に置き換えます。他のマーカーはルールごとにあり、何に置き換わるかは名前が示している通りです。

グローバルに編集したいのではなく、特定のフォーム専用にメッセージを変更したい場合は、３つ目のパラメーターに配列で Validator::make() メソッドに渡してください。

```
$messages = array(
    'same'    => 'The :attribute and :other must match, fool!',
    'size'    => 'The :attribute must be exactly :size , like duh!'
);

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

いいですね。これでカスタムメッセージが使いこなせます！特別なエラーメッセージを個別のフィールドに割り当てたいのでしたら、キーに「フィールド名_ルール」を指定します。例えば：

```
$messages = array(
    'age_required'    => 'You need to have had at least one birthday!'
);
```

8.5 カスタムバリデーションルール

validator クラスは、あなたのアプリケーションに合うように、ルールを付け加える方法も用意しています。早速、新しいバリデーションルールをどう登録するのか、やってみましょう。

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

新しいバリデーションルール、superdooper は値が文字列 superdooper と一致していることを確実にするために用意します。このカスタムバリデーションルールはマッチする場合は true を、しない場合は false を返します。

$attribute 値はバリデーションしているフィールドの名前で、$value はもちろんその値です。

$parameters 属性はルールのあとのコロンに引き続き、カンマで区切られて指定されるパラメーターを配列にしたものです

新たらしい validator を作成しても、どのエラーメッセージと結びついているのかまだ決まっていないので、いつ・何を間違ったのかを Laravel に知らせるために、メッセージを追加する必要があります。既に説明した方法でエラーメッセージを追加することができます。

```
'superdooper' => 'The :attribute must be superdooper, ok trooper?!',
```

再度説明しますが、追加の 3 つ目のパラメータとして配列で、Validator::make() メソッドに渡すか、もしくはシンプルに application/en/validateion.php に追加するのが安全です。

8.6 バリデーションクラス

もし多くの新しいバリデーションメソッドを追加するか、プロジェクト全体で再利用したいのであれば、自分のバリデーションクラスを作成するのがベストな方法です。バリデーションクラスは Laravel のデーターを拡張し、追加のバリデーションメソッドでオーバーロードします。このクラスは application/libraries ディレクトリーに作成すると、ローディングが簡単でよろしいでしょう。ですが、どこにでも好きな場所に配置し、オートローダー(後ほどの章で説明します）に登録することもできます。クラスを見てみましょう。

```
// application/libraries/validator.php

class Validator extends Laravel\Validator {

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

}
```

ご覧のとおり、Validator クラスは、Laravel\Validator 名前空間コアクラスを拡張し、validate_< ルール名 > というメソッドを持ちます。この方法で Laravel はソースフォルダーのクラスの代わりに、私達が作成したクラスを使用するようになります。

オリジナルのバリデーションメソッドを自分が作成したものに置き換えることもできます。例えば、自分で `validate_size` メソッドを作成すれば、サイズを別の形式で計算することができます。

バリデーションクラスを使用する時、バリデーション言語ファイルにカスタムメッセージを追加することを、私はおすすめします。他のプロジェクトに移行する場合に楽ですし、すべてのメッセージを見つけるために「ソース検索」する必要も起きません。

9 マイグレーション

最初は Fluent クエリービルダーについての説明も、この章に含めようと思っていましたが、それでは余りにも長くなってしまうでしょう。ですから、データベースのセットアップとマイグレーションだけをこのチュートリアルで行います。次のチュートリアルで行う、長いですが素敵な Fluent の説明に、心の準備をしておいてください。

マイグレーションは Laravel のお気に入り機能の一つです。私は SQL を書くことが大嫌いです。この馬鹿げた「言語」を一行も書くことなく、スキーマクラスはテーブルを簡単に作成してくれます。それだけでなく、スキーマコードは絶対的に美しく、小説のように読むことができます。

マイグレーションを以前に使用したことが無い人のために説明すると、ファイルにデーターベースの変更を記述しておく方法であり、現バージョンを認識し、異なったバージョンをインストールしたり、開発するために使われます。バージョンを元に戻す（ロールバック）することも可能です。また、マイグレーションでサンプルデーターをテーブルにセットすることもできます。（シーディングとして知られています。）

9.1 データベースの準備

`application/config/database.php` 設定ファイルを見て下さい。もし、PHP アプリケーションをインストールしたことがあれば、こうしたファイルはおなじみでしょう。自分のデータベースにアクセスするための委任状をお持ちでしょう？馴染みがなければ、今から見て行きましょう。

見てみましたか？いいですね。それではもっと近くで見てみましょう。

`connections` まで矢印キーで画面を下げてください。するといくつかの異なったデータベースタイプに対する複数のオプションが見えてきます。自分が選んだデータベースの接続パラメーターを埋めてください。私は古き良き時代の mySQL を選びます。

```
'mysql' => array(
        'driver'   => 'mysql',
        'host'     => 'localhost',
        'database' => 'codefun',
        'username' => 'root',
        'password' => 'pandaseatbamboo',
        'charset'  => 'utf8',
        'prefix'   => '',
),
```

今度は少し上にスクロールアップして、'default' 配列キーの値をあなたの選択したデータベースにしましょう。私の選んだ mySQL は最初から設定されています。

```
'default' => 'mysql',
```

データベースの設定は整いました。テーブルで遊んでみましょう。マイグレーションへと進みます！

9.2 マイグレーション

まずはマイグレーションファイルを作成するところから開始しましょう。

これから、Laravel のコマンドラインインターフェイスを使用します。Artisan で新しいマイグレーションを作成しますが、Artisan を実行するには PHP のコマンドライン版をインストールする必要があります。（通常これは Web サーバーではインストールされています。）端末を Laravel パッケージのルートで開きます。artisan ファイルが存在する場所です。では、最初の Artisan コマンドを叩きましょう。

```
php artisan migrate:make create_users
```

これにより、Artisan にマイグレートタスクのメソッドを生成させます。マイグレーションに識別のための名前をつけます。私は実行するアクション名に続いてテーブル名をつけるのが好きです。この場合 users テーブルです。

実行結果を確認しましょう。

```
Great! New migration created!
```

素晴らしい！application/migrations フォルダーを覗いてみれば、2012_03_30_220459_create_users.php という新しいファイルが生成されているのが分かります。みなさんのファイルは別の名前でしょう！Artisan は現在の日付と時間を His フォーマットでファイル名に付け加えます。その理由はデートはマイグレーション（と結婚していない人々）にとって大切だからです。システムはどの変更が古いものか知る必要があるのです。

ファイルを開いて、マイグレーションクラスの内容を確認しましょう。

```
class Create_Users {

        /**
         * Make changes to the database.
         *
         * @return void
         */
        public function up()
        {
                //
        }

        /**
         * Revert the changes to the database.
         *
         * @return void
         */
        public function down()
```

```
        {
                //
        }

}
```

ご覧のとおり、マイグレーションは2つのメソッドで構成されています。up() はデータベースに関するすべての変更を受け持ち、down() は全く逆の働きを行います。このようにマイグレーションは行われ、必要に応じてロールバックできるのです。もし up() メソッドのなかでテーブルを生成したら、down() ではテーブルをドロップします。

どうやってデータベースの変更を実行するのでしょうか。間違いなく複雑なひどい SQL クエリーで行うんですよね？ははははは。いいえ。Laravel を使っているのです。もしひどいのなら、それはフレームワークの外の話しです。Laravel の Schema クラスを使ってテーブルを作成する方法を見ていきましょう。

```
Schema::create('users', function($table) {
        // auto incremental id (PK)
        $table->increments('id');

        // varchar 32
        $table->string('username', 32);
        $table->string('email', 255);
        $table->string('password', 64);

        // int
        $table->integer('role');

        // boolean
        $table->boolean('active');

        // created_at | updated_at DATETIME
        $table->timestamps();
});
```

Schema クラスの create() メソッドを呼び出すことで、新しいテーブルが生成されます。パラメータとして生成するテーブル名を指定し、第2パラメータはクロージャーです。クロージャーにもパラメーターを渡していますがお好きな名前をどうぞ。私はコードが読みやすいため、$table を使用しています。

クロージャーの中で、$table パラメーターに対し、上手く名付けられたメソッドを使用することで、いくつもフィールドをお手軽に生成できます。簡単な一覧を見て下さい。

increments()

オートインクリメントの ID を追加する。ほとんどのテーブルにはこれがあることでしょう！

string()

VARCHAR フィールドを追加します。string という名前はぴったりでしょう？

integer()

integer フィールドを追加します。

float()

float フィールドを追加します。

boolean()

boolean フィールドを追加します。

date()

date フィールドを追加します。

timestamp()

timestamp フィールドを追加します。

timestamps()

created_at と updated_at timestamp フィールドを追加します。

text()

text フィールドを追加します。

blob()

blob データーフィールドを追加します。

メソッドに ->nullable() をチェーンすることで、NULL 値を受け付ける属性を設置します。

上記のメソッドの完全なパラメーターは公式ドキュメント[1]に書かれています。

素晴らしい、これでテーブルはできます！up() の中でテーブルの生成を行いましたから、今度は down() メソッドの中で、テーブルをドロップする必要があります。そうすることでスキーマはロールバックの後に元通りになります。ありがたいことに、Schema クラスはこれを行うのにぴったりな別のメソッドを持っています。

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

ええ、これより簡単になんて出来ません。Schema クラスは他のスキーマ操作を行う関数を持っています。フィールドを消去する drop_column()、インデックスを追加する unique()、いくつかの外部キーメソッドです。ここでカバーしたすべてのメソッドは、一連の API として存在しますが、公式ドキュメント[2]によく書かれています。もし新しい項目へ進みたいのでしたら、今までのマイグレーションがどう動くか確認するのはどうでしょうか？

マイグレーションを動かす前に、laravel_migrations テーブルをインストールする必要があります。これにより、Laravel は今まで実行しているマイグレーションをトレースし続けることができます。幸運なことに、Artisan はこのデータベースの変更を行うコマンドを用意しています。やってみましょう。

[1] http://laravel.com/docs/database/migrations
[2] http://laravel.com/docs/database/schema

```
php artisan migrate:install
```

実行結果は…

```
Migration table created successfully.
```

素晴らしい！laravel_migrations テーブルが生成されました。ついに、新しいマイグレーションを実行できるようになりました。今回はシンプルにタスク名だけを使います。では、行いましょう。

```
php artisan migrate
```

```
Migrated: application/2012_03_30_220459_create_users
```

おぉ。データベースを確認すれば、users テーブルが生成されたことがわかるでしょう。待ってください。どうやら間違えたようです！（本当は間違ってません。しかし私はドラマが好きですし、ロールバックする言い訳がほしいのです。）スキーマの変更を戻すために、ロールバックしましょう。

```
php artisan migrate:rollback
```

```
Rolled back: application/2012_03_30_220459_create_users
```

覚えていて欲しいことことは、データベースにサンプルデータを導入したい場合、Fluent クエリービルダーで生成できることです。このガイドでは扱いませんが、以降のチュートリアルで現れます。ですから、待ち遠しいですね。

めちゃくちゃ簡単です！今では皆さんマイグレーションの方法を覚えました。スキーマを組み立て、データベースを構築できます。いつでも好きなときにです！次のチュートリアルは Fluent クエリービルダーを学びましょう。

ここでお約束していた通り、９ドル９９セント以上でこの本を購入して下さった方の名前とメッセージをリストしていきたいと思います。

> IoC コンテナについて話すのを忘れないでください。ユニットテストでどんなに役に立つかという事です。君は Laravel という懸命な選択をしたね。- Jorge Martinez

Jorge さん、ご購入感謝します。IoC コンテナについてはアプリケーション作成の基礎を全てカヴァーした後の章で取り扱います。

> @HelpSpot は Laravel コミュニティーを大きくするサポートを行いましょう。良い仕事を続けてください！- Ian Landsman

ありがとう、Ian さん！私達 Laravel コミュニティーはフレームワークを拡張するためにあなたの行なって下さったことを支持いたします。おかげでこんなに大きくなれました！知らない方のために説明すると、Ian Landsman さんは UserScape 社のトップであり、この会社はフレームワークを拡張できるように、フレームワークの開発者に仕事を与えてくれているのです。UserScape は Laravel のバンドルサイトのスポンサーでもあり、その他多くの支援を行なってくださっています。彼らの製品である素晴らしく柔軟性に富み、入手可能なヘルプデスクソフトウエアである Helpspot[3]をチェックしてみてください。

他にも９ドル９９セント以上で購入された方がいらっしゃるのですが、まだメールを受け取っていません。どうか me@daylerees.com へ送信してください。もし９ドル９９セント以上お支払いいただけば、名前をこの本にのせられることを思い出してください。残念ながら、この出版サイトのセールス・タブを見ても、購入者の名前はわからないのです。

[3] http://www.helpspot.com/

10 Ｆｌｕｅｎｔクエリービルダー

Fluent は Laravel プロジェクトが提供している、もう一つの素晴らしいライブラリーです。SQL 弾を避ける事ができるのですから。まあ、あなたが苦痛を楽しめるというのでしたら、SQL 文を直接書いてもいいんですよ。Fluent の良い所ですか？SQL を使わないことを除いて、プリペアードステートメントを使っているので、SQL インジェクションを完全に防いでくれることです。更に Fluent は……可愛い Fluent は SQL 間にある沢山の方言を超え、提供されるメソッドは様々なデータベースに対して動作します。本格的に始める前に、メソッドチェーンのコンセプトを理解しましょう。次の例を見て下さい。

```
Class::make()->chain()->chain()->chain()->trigger();
```

これはオプションを一度に指定するには便利な方法です。SQL を可愛く表現する方法です。（あとでお目にかけます。）クラスは `make()` によりインスタンス化され、「生成パラメーター」を指定することもあります。`chain()` メソッドはそれぞれ別のオプションを指定します。`trigger()` メソッドは最終的な結果を返します。ちょっと混乱するかも知れませんね。Fluent の実例をごらんください。

```
$users = DB::table('users')->where('username', '=', 'dayle')->get();
```

上の例が実行するのはシンプルな…

```
SELECT * FROM users WHERE username = 'dayle';
```

クエリー結果のデータベースフィールドを表す、オブジェクトの配列を返してきます。

`table()` メソッドでオブジェクトをインスタンス化し、操作するテーブルを指定します。`where()` はクエリーの `WHERE` 節に該当し、メソッドチェーンでつないでいます。`get()` はクエリー結果の全オブジェクトを提供する最終的なメソッドです。

`get()` は結果を配列で返しますので、結果を見るには foreach を利用します。

```
foreach ($users as $user)
{
        echo $user->email;
}
```

上の例でわかるように、結果のオブジェクトの属性として、行のフィールドにアクセスします。ループですべてのユーザーのメールアドレスを出力しています。

10.1 クエリー結果の取得

クエリー結果を取得する、他のメソッドも見ておきましょう。

get()

今使ったばかりですね。すべてのクエリー結果をオブジェクトの配列で返します。

first()

単独のオブジェクトを返します。一番最初にマッチしたクエリー結果です。

find($id)

これはデータベースの id で検索するものです。`where('id', '=', $id)` のお気軽なショートカットで、単独のオブジェクトを返します。

only($fieldname)

クエリーから指定されたフィールドだけを返します。

get(array())

配列で指定されたフィールドだけを返します。

10.2 Ｗｈｅｒｅ節

データベースの結果を獲得するのに必要なメソッドは分かりました。ではどうやって SQL クエリーに検索条件を指定しましょうか？その通り、上の例でも見た通り `where()` メソッドを使います。もう少し詳しく見てみましょう。

```
$users = DB::table('users')->where('username', '=', 'dayle')->get();
```

同じスニペットです。ですがチェーン中、`where()` のところに注目してください。

```
where('username', '=', 'dayle')
```

Laravel が where 節を扱う方法の素晴らしいところは、チェーンが生成される SQL と少し似ていることです。上のメソッドチェーンで、WHERE username = ‘dayle’ を表しています。最初のパラメーターは比較するフィールド、２番目は比較演算子、３番目は比較する値です。ですから、こんな使い方もできます。

```
where('age', '>', '18')
// WHERE age > '18'
// 飲んでください！アメリカに住んでいないのでしたら、ごめんなさい。
```

なに？もっと条件が必要ですか？では、最初に決める必要があるのは、”AND WHERE” が必要なのか、”OR WHERE” が必要なのかですね。”AND” を使いたい時は、シンプルに `where()` メソッドをチェーンとして追加するだけです。例えば：

```
$users = DB::table('users')
        ->where('username', '=', 'dayle')
        ->where('sexyness', '>', 5000)
        ->get();
```

上の例で見た通り、新しい行にチェーンを追加しました。この方法で行が長くなるのを防げるので、読みやすいと思います。このチェーンで次の SQL が生成されます。

```
SELECT * FROM users WHERE username = 'dayle' AND sexyness > 5000;
```

もし OR where 条件が使いたい時は、or_where() メソッドを使えば良いのです。パラメーターは同じです。例えば：

```
$users = DB::table('users')
        ->where('username', '=', 'dayle')
        ->or_where('face', 'LIKE', '%malemodel%')
        ->get();
```

生成されるのは：

```
SELECT * FROM users WHERE username = 'dayle' OR face LIKE '%malemodel%';
```

SQL で何ができるのかをここで全て説明する必要はないでしょう。他に本もたくさん出版されていることですしね。その代わりに、一般的な作業を行うメソッドの名前だけを紹介しましょう。

where_in()、where_not_in()、or_where_in()、or_where_not_in() メソッドは、フィールドと配列で与えられた複数の値を比較するために使います。

where_null()、where_not_null()、or_where_null()、or_where_not_null() メソッドは NULL 値であるか調べるために使用します。

where をネストしたい場合、Laravel は「ネスト Where 節」形式で機能的な手法を提供しています。サンプルコードを御覧ください。

```
$users = DB::table('users')
        ->where('id', '=', 1)
        ->or_where(function($query)
        {
                $query->where('age', '>', 25);
                $query->where('votes' '>', 100);
        })
        ->get();
```

where に含まれたクロージャーを使うことで where 節のネストを行うことができます。この結果、生成される SQL は：

```
SELECT * FROM "users" WHERE "id" = ? OR ("age" > ? AND "votes" > ?)
```

絶妙でしょう？

更にもっと興味深い機能もあります！（いかに私が SQL を嫌っているか分かるでしょう？）動的 where 節で単純な where 節で本当にファンキーなことが出来るのです。チェックしてください。

```
where_size(5)->get();
```

特定のフィールドとの比較をメソッド名に指定できます。Fluent は賢いので、うまく処理してくれます。問題ありません！

AND も OR も理解してくれますよ。チェックしましょう。

```
where_size_and_height(700, 400)->get();
```

叙述的で、クリーンです。Laravel は最高です。

10.3 テーブル接続

Fluent での join を見てみましょう。

```
DB::table('tasks')
        ->join('project', 'tasks.id', '=', 'project.task_id')
        ->get(array('task.name', 'project.name'));
```

join するテーブル名が最初のパラメーターです。ON 節として残りの 3 つのパラメーターが利用され、WHERE と似ています。

フィールドを渡したら、get() メソッドで配列として結果を入手します。

left_join() を同じように使用できます。パラメーターも一緒です。簡単でしょう？

ネスト Where 節を覚えていますか？同じテクニックが ON 節に条件を加えるときに使用できます。ざっと見てみましょう。

```
DB::table('tasks')
        ->join('project', function($join) {
                $join->on('tasks.id', '=', 'project.task_id');
                $join->or_on('tasks.author_id', '=', 'project.author_id');
        })
        ->get(array('task.name', 'project.name'));
```

この場合、join() メソッドの第 2 パラメーターとして無名関数を指定し、on()、or_on()、and_on() メソッドを条件指定に使用します。

10.4 並び替え

並べ替えはとても重要です。PHP を使ってリソースを無駄にしたくないでしょう。確かに…できますよ…たくさんの汚い array_sort() 関数達、多くのループ、けど全然楽しくありません。そうした役目は Fluent に任せて下さい。

```
DB::table('shoes')->order_by('size', 'asc')->get();
```

うむ、靴か？女性は靴が大切だと考えますが、思い浮かぶのは…変だということです。とにかく、フィールド名を渡すのは簡単です。asc で昇順、desc で降順です。更に多くのカラムでソートしたければ、ただ `order_by()` をチェーンしましょう。

10.5 Ｌｉｍｉｔ…何もいらない

特定の数だけ結果を返してもらいたい場合、SQL では間抜けな名前の `LIMIT` を使います。Laravel では、`take()` です。

```
DB::table('shoes')->take(10)->get();
```

これだけで１０足分取得できるのか？これ以上心配するべきでありませんが…明確です。Limit はとてもシンプルです！

10.6 結果を飛ばす

最初の５足は必要ない場合、どうしましょう？革靴だったんですが、私が好きなのはスケート靴で、ご覧のとおり足が大きいものですから…では、飛ばしましょう。

```
DB::table('shoes')->skip(5)->get();
```

これで最初の５つの結果を skip() できました。簡単ですね！

10.7 集計

時々SQL で行うように、基本的な計算のクエリー、AVG, MIN, MAX, SUM,、COUNT を実行すると、いつでも素早く結果が得られるので便利です。Fluent クエリービルダーを使用しても可能です。やってみましょう。

```
$val = DB::table('shoes')->avg('size');
$val = DB::table('shoes')->min('size');
$val = DB::table('shoes')->max('size');
$val = DB::table('shoes')->sum('size');
$val = DB::table('shoes')->count();
```

簡単です！これらのメソッドは結果を直接返すので、`get()` が要らないことを覚えておいてください。`where()` などお好きなもので、条件を絞り込めることもできますよ！

10.8 式

メソッドは指定されたパラメーターを自動的にエスケープし引用符で閉じますが、これを自分で行いたい時はどうしましょうか？気に入らない他の何かの力を借りるのでしょうか？これを行うには DB::raw() メソッドが用意されています。では、例を御覧ください。

```
DB::table('shoes')->update(array('worn' => DB::raw('NOW()')));
```

このクエリーの NOW() はエスケープも引用符で閉じられることもありません。ここには自由があります。しかし、スパイダーマンのセリフを忘れないでください。この力には責任をとってください。

10.9 ＋＋（またはデクリメント）

単に値をインクリメント、もしくはデクリメントしたい時にはどうしましょうか？余りにも簡単です！

```
DB::table('shoes')->increment('size');
DB::table('shoes')->decrement('size');
```

ただ、フィールド名を指定するだけです。これが全てです！

10.10 レコード挿入

遂に、データの保存まで来ました。今までは、データを読み込む方法だけでした。これでもっと面白くなります！まあ、本当に簡単なんです。新しい行を挿入するには、キーと値の配列を insert() メソッドに渡してあげるだけです。

```
DB::table('shoes')->insert(array(
        'color' => 'hot pink',
        'type'  => 'heels',
        'size'  => '12'
));
```

ちょっと待った。新しい行に生成された id が知りたいんですけど。後で取得するの？insert_get_id() を同じパラメーターで使ってください。この様にです。

```
$id = DB::table('shoes')->insert_get_id(array(
        'color' => 'hot pink',
        'type'  => 'heels',
        'size'  => '12'
));
```

これは私が週末に履く予定です。予約しておきましょう…これでかっこいいホットピンクの１２インチのヒールがデータベースの shoes テーブルへ入りました。

10.11 レコード更新

待ってください。もしかしたら私、ヒールって言いました？ホットピンクのスケート靴の間違いでした。元に戻って間違いを修正すれば、テーブルを修正して、誰も私の恥に気が付きません。多分、update() メソッドが使えるでしょう。insert() と同じシンタックスで、配列が使えます。

```
DB::table('shoes')->update(array(
        'type' => 'skate shoes'
));
```

待ってください。レコードを指定してませんね。全レコードをスケートシューズにしたくはありません。素晴らしい Laravel の、逆に悪い点です。'where()' メソッドを使って取っておいた $id をもとに、特定のレコードだけに絞りましょう。チェインを使う時です！

```
DB::table('shoes')
        ->where('id', '=', $id)
        ->update(array(
                'type' => 'skate shoes'
        ));
```

これでよし。誰にも気づかれずに修正できました。

10.12 レコード削除

行を削除するためには（どうか靴で無いものを。靴には罪がありません！）delete() メソッドを where() 節と一緒に使うか、単に削除したい特定の行を示す id を指定するだけです。実際に見てみましょう。

```
DB::table('not_shoes')->where('texture', '=', 'fuzzy')->delete();
```

簡単です。これで毛羽立った (fuzzy) 生地 (texture) が無くなりました。

```
DB::table('shoes')->delete($id);
```

だめ！ホットピンクのスケートシューズはだめです。ああ、明らかに遅すぎました。力と責任のことを心においておいてください。これ以上、靴を大虐殺しないために。

次のチュートリアルでは、靴……Fluent から離れ、Eloquent についてお話ししましょう。Eloquent はデータベースの行をオブジェクトのように扱わせてくれます。データ関係を扱う、明らかにエレガント…または eloquent（表現豊か）な手法です。

Max Schwanekamp 氏のご購入に感謝します。そして Douglas Grubba 氏は次のように言っています。

> Dayle、素晴らしい仕事を続けてください！素晴らしい資料です。ありがとう、@laravel チーム。@taylorotwell さん、あなたは PHP の世界を変えようとしています。

ありがとう Doug さん！Taylor も PHP の世界を変えようとしていると耳にしたら、喜んでくれると確信しています！

11 Ｅｌｏｑｕｅｎｔ　ＯＲＭ

ORM はほんとうに便利なパッケージです。Object Relational Mapper、オブジェクトリレーショナルマッピングの略です。それではこれを（低音のビートを加えて）小さくブレイクダウンしましょう。マッパーの部分は PHP オブジェクトやクラスをデータベーステーブルと行にマッピングすることです。リレーショナルの部分はリレーショナルのセクションで明らかにしましょう。

多くの ORM が存在していますが、Eloquent のように良くて、eloquent（感銘的）なものはありません。Eloquent は Laravel と一緒にパッケージされていますが、単独でも利用できます。この機能を理解するために、マイグレーションのチュートリアルで、データベースのセッテイングを行い、Fluent クエリービルダーのチュートリアルで、メソッドとチェーンに慣れてもらいました。これで基礎ができたわけです。

あなたは Eloquent が Fluent とは全く別物だと思っているかも知れません。しかし、多くのメソッドは共通ですし、唯一の違いといえば Fluent は stdObject を返すのに対し、Eloquent はモデルを結果として返すことです。これはコードを更に綺麗にしてくれます。では Eloquent モデルへ飛び込み、調べてみましょう。

11.1 Ｅｌｏｑｕｅｎｔモデルを作成し使用する

```
// application/models/user.php

class User extends Eloquent
{

}
```

実際これだけです…いや、本当に、私は真面目ですよ。ご覧のとおり、Eloquent はあなたを信用しています。Eloquent は users テーブルを素敵なマイグレーションで生成したことを既にわかっているんです。どんなフィールドがあるのか知らせろと要求することはありません。すべきことはマイグレーションを書き、テーブルを作成することです。ああ、ところで、テーブルに increments('id') を付けましたよね？Eloquent を動かすには、これが必要です。

もう一つ注目すべきことは、モデルのクラスは User という名前で、テーブル名は users であることです。タイプミスではありません。Eloquent は賢いので、英語の複数形を理解します。オブジェクトは単数ですから、User と定義しました、しかしデータベースのテーブルは users で、多くのユーザーで構成されます。ですから、Laravel はオブジェクト名の複数形でテーブルを探せばよいと、わかっているのです。

あなたは、英語だけの複数形を理解するという事実を指摘するかも知れません。別の言語を使っていますか？複数形が上手く働きませんか？でしたら複数形の定義を配列で application/config/strings.php につけ加えるだけです。これで、希望通りに動くようになりました。

もしかしたら、あなたはテーブル名を複数形で名付けなかったかも知れませんね？問題ありません。シンプルにクラス属性の `static $table` に他のテーブル名を指定してください。一例です。

```
class User extends Eloquent
{
        public static $table = 'app_users';
}
```

そうです、巨大なモデルの定義をタイプするなんて、あなたを消耗させるだけです。（嫌味で言っています。）新しいモデルを使ってみましょう。

```
$user = User::find(1);
```

待てよ……これは前に見たことがありますね。Fluent チュートリアルで find() を使っていたのを覚えていますか？これはプライマリキーにより、一件の結果を得るためのものでした。Fluent で使用されている多くのメソッドが Eloquent でも使えます。これは本当に便利で、両方のライブラリーを組み合わせて使用しても、どっちがどっちだったか迷いません。素晴らしい！

Eloqunet に対しては、オブジェクトの名前だけを使用し、static なメソッドが提供されており、`DB::table()` は使いません。

戻り値のオブジェクトも似たようなものです。実際のところあなたは Eloquent オブジェクトを操作しているのですが、しかしこの時点では違いがありません。ユーザー名を取得してみましょう。

```
echo $user->name;
```

前にやったのと同じですね。ナイスでイージーです。

`User` オブジェクトではなく、配列で受け取りたい時には？ただ Eloquent モデルに `to_array()` メソッドをつければ、オブジェクトのインスタンスの代わりに、配列が受け取れます。

```
$user = $user->to_array();
```

もし、特定のフィールドをこの配列に含めたくないのであれば、Eloquent モデルに `static $hidden` 変数を作り、除外するフィールドを配列で指定します。以下のようにです。

```
class User extends Eloquent
{
        public static $hidden = array('nickname');
}
```

複数の結果を受け取りたい時は？全ユーザーレコードを受け取りたい時は `all()` メソッドを使えます。そうしたら、セクシーなユーザーのパーティを開くことができますね。

```
$users = User::all();
```

今受け取った結果は、ユーザーオブジェクトの配列です。それぞれの個人にアクセスするには、ただループで扱えば良いのです。サンプルです。

```
foreach ($users as $user)
{
        add_to_sexy_party($user);
}
```

素敵ですね。いい調子です。何人かの男性は考えているでしょう……チャンスを増やすため、男は参加させない。そうしたら、本当のセクシーパーティになります！

```
$sexy_users = User::where('sex', '=', 'female')->get();
```

どうです、いいですね。見やすさという敷居がだいぶ上がりました。where() をメソッドチェーンして、fluent でやった get() と同じ事をしただけです。何も新しいことはありません。見やすさが向上しました。

どんな結果がクエリーメソッドで返ってくるか全部説明し直すこともできますが、そんな必要はありませんよね。前のチュートリアルに戻り、Fluent という言葉を Eloquent に心の中で置き換えてください。お分かりになったと思います。

その代わり、変更点を見てみましょう。オブジェクト（レコード）の挿入と変更は、本当にやりやすくなってます！

```
$user = new User();

$user->name = 'Captain Sexypants';
$user->mojo = '100%';

$user->save();
```

パーティーにキャプテン・セクシーパンツを招かないでどうしますか。かれのオウムがふんぞり返っていても、どんなアクションも取るつもりはありません。

新しい User オブジェクトを生成するシンプルなやり方を見てもらいました。クラスの属性として存在するフィールドをセットし、オブジェクトを save() するだけで、データベースに変更が保存されます。（ID をセットする必要はありません。Eloquent が面倒みてくれます。）これは一般的にアクティブレコードとして知られるデザインパターンです。アプリケーション中の他のオブジェクトと同じやり方でレコード操作ができます。まるで、データベースと汚い全ての SQL が存在していないかのように！

もし、事前に私たちの友達である「キャプテン・セクシーパンツ」を表現したキーと値の配列が用意されている場合は、コンストラクターにパラメータで渡すか、複数代入メソッドである fill() をデータベースに追加するときに使用できます。これを確認しましょう。

```
$user = new User(array(
        'name' => 'Captain Sexypants',
        'mojo' => '100%'
));

$user->save();
```

もしくは、

```
$arr = array(
        'name' => 'Captain Sexypants',
        'mojo' => '100%'
);

$user = new User();
$user->fill($arr);
$user->save();
```

複数代入を使用する時は、余計な情報を漏らさないように注意してください。セキュリティーリスクになる可能性があります。

では更新はどうするのでしょう？最初に更新したいユーザーを問い合わせること以外、とても似ています。見て下さい。

```
$user = User::where('name', '=', 'Captain Sexypants')->first();
```

ここでは `first()` を使っています。確実に 1 レコードだけを手にしたいからです。配列を使って変更は出来ません。その場合、それぞれのレコードを繰り返しで処理することになりますが、例としては長くなってしまうので必要ないでしょう。

```
$user->mojo = '5%';
$user->save();
```

ええ、彼の魔力 (mojo) を全部使わせたくありません。だって不公平ですからね。いずれにせよ、彼はパーティーにやって来ました。見ての通り、最初に更新したいオブジェクトを見つける必要があること以外、更新も挿入と全く同じやり方です。

マイグレーションのチュートリアルで、updated_at と created_at を作成するために `$table->timestamps()` をしていたことを覚えていますか？Eloquent はあなたのために自動的に更新しています。オブジェクトを挿入したタイムスタンプと、updated_at はセーブするたびに更新されます。これを無効にしたい時には、ただ `$timestamps` クラス属性をモデルに追加し `false` をセットして下さい。

```
class User extends Eloquent
{
        public static $timestamps = false;
}
```

11.2 リレーションシップ

リレーションシップは美しい。決して感傷的になっているわけでありません。テーブル間のリレーションのことです。Eloquent を使うと、美しんです。”JOIN” なんてガラクタはありません。「1 対 1」、「1 対多」、「多対多」をシンプルな Eloquent のモデルにメソッドを付け加えることで定義できます。早速やってみましょう。次のコードから学ぶことができます。

1 対 1

早速、飛び込みましょう…

```
class User extends Eloquent
{
        public function invite()
        {
                return $this->has_one('Invite');
        }
}
```

では先に進め、私たちのセクシーパーティーへの招待のために、invites テーブルと invite モデルを作成しましょう。

新しく作成した public の invite() メソッドを呼び出し、$this->has_one('Invite') を返すことで、一つを持つ関係だと宣言し、ユーザーの招待 (invite) を見つけ出します。

では、はっきりした表現的な文法を使用して、ユーザー（それと、パーティーゲスト）への招待を提供しましょう。見てください。

```
$invite = User::find(1)->invite()->first();
```

ひとつだけ結果をもらいたいので、再度 first() を使用しましょう。->invite()-> をチェーンすることで、これはもちろんリレーションシップメソッド名ですが、そのユーザーに関連する招待 (Invite) オブジェクトを手に入れることができます。

このリレーションシップは 2 つの SQL を実行します。

```
SELECT * FROM "users" WHERE "id" = 1;
SELECT * FROM "invites" WHERE "user_id" = 1;
```

クエリーを見ると、Eloquent は自動的に外部キーとして user_id を探しているのが分かります。そうなると、user_id の整数フィールドを Invites テーブルの中に、マイグレーションで生成しなくてはなりそうです。でも他の外部キーなどを呼び出す場合は？問題ありません！単純に、has_one() メソッドの第 2 パラーメーターとして（他のメソッドでも同様ですが)、新しいフィールド名を指定してください。例えば：

```
class User extends Eloquent
{
        public function invite()
        {
                return $this->has_one('Invite', 'the_invite_id');
        }
}
```

この関係の逆はどうなっているのでしょう。もし、私達が招待状を持っていて、誰がそれを所持しているのか知りたいときは。ここで所属（belongs_to()）メソッドがお手軽に使えます。モデルを見てみましょう。

```
class Invite extends Eloquent
{
        public function user()
        {
                return $this->belongs_to('User');
        }
}
```

belongs_to() を代わりに使用しているだけで、同じような文法でこのテーブルの中にある、外部キーを指定しています。

さあ、これで使えますよ…

```
$user = Invite::find(1)->user()->first();
```

簡単です！

1 対多

関係がある複数のアイテムを手に入れたい時は？多分、あなたはタイトルから、何メソッドか想像していることでしょう？見てみましょう。（何度も同じ事を言っていますよね。これが私のキャッチフレーズになりそうです。）

```
Class User extends Eloquent
{
        public function hats()
        {
                return $this->has_many('Hat');
        }
}
```

またまた、ご覧のとおり、オブジェクト名を先頭を大文字にした文字列で、has_many メソッドに渡しています。

この例では、hats テーブルは user_id 外部キーが必要で、それを使います。

```
$hats = User::find(1)->hats()->get();
```

動的プロパティーを名前に対して使用すれば、同じ結果を短く書くことができるのに注目してください。例えば：

```
$hats = User::find(1)->hats;
```

やっぱり、すごい！！！

前と同様に、第２パラメーターで代わりの外部キーを渡すこともできます。では、先に進みましょう。

多対多

今度は多少複雑になりますが、心配要りません……そのために、私がいるわけです。ウェンディ、手を取って、２つの Eloquent モデルがあるとイメージして。User と Task です。ユーザーは多くの仕事 (tasks) を持っているでしょうし、仕事も多くのユーザーを持っていることでしょう。こうしたタイプの関係には、３つ目のテーブルが必要です。

このテーブルは多くの名前で知られており、ピボットテーブル、ルックアップテーブル、中間テーブル、EI ピボットグランデなどです。

２つのテーブルを結びつけ、多対多の関係を作るために user_id と task_id、２つの整数フィールドを持つ単純なテーブルになります。

テーブル名は関係するテーブルの複数形をアルファベット順につなげたものです。読むと複雑そうですが、見てもらえば単純です。確認してみましょう。

tasks_users

グレート！では、私たちのテーブルの番です。関連付けましょう。

```
class User extends Eloquent
{
        public function tasks()
        {
                return $this->has_many_and_belongs_to('Task');
        }
}
```

これも、同じ書き方です。ちょっとメソッド名が長くなっただけです。

この場合、オプションの第２パラメーターは、違った名前のピボットテーブル名を指定するときに使用します。シンプルですね。

11.3 リレーションモデルの挿入

関係する複数のモデルにレコードを追加する場合、user_id か hat_id 外部キーを自分でセットします。しかし、それは可愛くありません。代わりに、オブジェクトを渡しましょう。

```
$hat = Hat::find(1);
$user = User::find(1);
$user->hats()->insert($hat);
```

ずっと良いですね。個別のオブジェクトを処理するのと似ています。整数の外部キーを直接扱うより綺麗です。

has_many() の多対多リレーションでは、フィールドと値のペアを配列で save() メソッドに渡すことにより、関連するモデルに insert もしくは update されます。例えば：

```
$hat = array(
        'name'  => 'Dennis',
        'style' => 'Fedora'
);

$user = User::find(1);

$user->hats()->save($hat);
```

クリーンです！

新しい関係したアイテムを多対多リレーションで作るときに、insert() メソッドを使えば、Eloquent は新しオブジェクトを作ってくれるだけでなく、ピボットテーブルにも新しいエントリーを作成してくれます。例えば：

```
$hat = array(
        'name'  => 'Dennis',
        'style' => 'Fedora'
);

$user = User::find(1);

$user->hats()->insert($hat);
```

しかし、もしオブジェクトが既に存在していて、ただそれらの間のリレーションだけを作成したい時には？関連するオブジェクトの $id（もしくはオブジェクト）を attach() に渡してあげるだけで、簡単に実現できます。コードを見て下さい。

```
$user->hats()->attach($hat_id);
```

ピボットはあなたのために更新されます！

次のメソッドは私自身も最近気がついたもので、たぶん 3.1 から Eloquent に登場したものだと思います。sync() メソッドで id の配列を渡し、実行すると、指定した配列の中の `id` だけがピボットに残ります。とても使いやすい！これがコードになります。

```
$user->hats()->sync(array(4, 7, 8));
```

11.4 ピボットテーブル

関連するテーブルではなく、ピボットテーブルを直接操作したい時には？Eloquent は `pivot()` メソッドを用意しています。

```
$pivot = $user->hats()->pivot();
```

他のクエリーと同じように、結果はオブジェクトの配列で戻ってきます、しかしそれらはユーザーの hats に関連したものです。

オブジェクトに関連して使用されている特定のフィールドを受け取りたい時には？`pivot` 動的属性で簡単にできますよ。これに関しては公式ドキュメントに素晴らしい例がありますので、パクらせてもらいましょう。なにせ私は汚い人間なんです。心配しないでくださいね。見た目はちょっと変更しています。高校の宿題のようにね。

```
$user = User::find(1);

foreach ($user->hats as $hat)
{
        echo $hat->pivot->created_at;
}
```

いつ帽子が作られたのかを確認するために、ピボットテーブルの created_at フィールドにアクセスしてみましょう。

オッケー、ちょっと帽子にこだわりすぎましたね。実際、私は自分の帽子を最後の一つまで削除したいのです。幸運なことに、自分のユーザーID が 7 だと知っています。ラッキー 7 です。早速、delete() で帽子を全部削除しましょう。

```
User::find(7)->hats()->delete();
```

これで済みました。私の頭も前に進めます。けれど、`delete()` メソッドをどう使うかは価値が高いので、忘れないでください。

11.5 Eagerローディング

Eloquent は eager ローディングでよく議論される N+1 問題を解決する手助けになるオプションを用意しています。なんだか分からないければ、今まで習ったことを利用して、説明しましょう。以下のコードから多くを学べます。

```
$users = User::all();

foreach ($users as $user)
{
        echo $user->hat->size();
}
```

それぞれのループでユーザーの帽子オブジェクトを手に入れ、サイズを見つけるために、Eloquent は別々の SQL を実行します。データーが小さい場合は大きな問題になりません、何百ものレコードになれば、実効速度に大きな影響を与えます。eager ローディングを使用すると、User オブジェクトの取得時に、Eloqunet は SELECT * FROM hats を実行し、必要なデーターを全て予め獲得します。 2 つのクエリーになるため、負荷を大きく軽減できます。だいぶ良くなりました！

Eloquent にリレーションを eager ロードするためのコードを見てみましょう。

```
$users = User::with('hat')->get();
```

これで hat リレーションが eager ロードされました。もし多くのリレーションを eager ロードしたいのなら、ただ `with()` メソッドに配列を渡してあげるだけです。`with()` メソッドはスタティックで、いつもチェーンの先頭に付ける必要があることは、とても重要なことです。

リレーションシップをネストして eager ロードすることさえできます。Hat オブジェクトは `hatstand()` リレーションシップを持っています。それが何かは名前が示しているとおり帽子スタンドです。両方のリレーションシップをこんなふうに eager ロードできます。

```
$users = User::with(array('hat', 'hat.hatstand'))->get();
```

ただ、ネストするオブジェクトの前に、リレーションシップ名を付けるだけです。

では、eager ロードに条件を付け加えたい時は？青い帽子だけ使いたくなることもあるでしょうし。もちろん、Eloquent を使って出来ます！単にリレーションシップ名を配列のキーとして渡すだけです。そしてクロージャーでパラメーターとして、条件を構成します。良い例があります。

```
$users = User::with(array('hat' => function($query) {
        $query->where('color', '=', 'blue');
}))->get();
```

シンプルです！必要に応じていくらでも条件を付け加えてください。

11.6 SetterとGetter

Setter は新しいモデルデーターを確実に同じフォーマットで設定したい時に便利な、小さなメソッドです。コードを見て下さい。

```
public function set_password($password)
{
    $this->set_attribute('hashed_password', Hash::make($password));
}
```

Setter にするには先頭に set_で始まる名前をつけ、受け取るフィールド値の変数を渡すメソッドを作成します。新しい値をフィールドにセットするには set_attribute() を使い、この例では呼び出されると、指定された値をハッシュした値を hashed_password フィールドに設置します。呼び出し側は：

```
$user->password = "secret_panda_ninja";
```

とっても使いやすいですね！

Getter は全く逆で、読んだ値を調整するために使用します。例えば：

```
public function get_panda_name()
{
    return 'Panda'.$this->get_attribute('name');
}
```

では、'Dayle' に使ってみると……

```
echo $user->panda_name;
```

これで"PandaDayle" を受け取る事ができます。

次の章では、Laravel のシンプルなイベントシステムを見てみましょう。

12 イベント

イベントはコードのメインとは別の部分を実行する方法で、アプリケーションを美しいやり方で拡張できます。このような拡張は別のアプリケーションでは「フック」と呼ばれます。あなたが「PHP 導師」であれば「オブザーバー／オブザーバル」デザインパターンと似ていると分かるでしょう。

イベントは多くのリスナーを持ち、発生されると多種のレスポンスを返すことができます。実際のコードで見てもらうのが良いでしょう。Laravel のイベントをよく見てみましょう。

12.1 イベントを起こす

```
$responses = Event::fire('dinner.time');
```

この例は dinner.time という、あなたの食事を知らせるイベントを「トリガー」もしくは「発生 (fire)」させ、すべてのイベントリスナーに通知されます。$responses 変数は、イベントが返すレスポンスで構成されています。詳細は後ほど。

全部ではなく、最初のレスポンスだけを受け取る事もできます。fire() の代わりに first() メソッドを使用するだけです。これで、レスポンスをひとつだけ受け取れます。

```
$response = Event::first('dinner.time');
```

覚えておくべきことは、たとえレスポンスをひとつだけ受け取るにしろ、fire() メソッドを使った時のように発生されたイベントは、すべてのイベントリスナーに通知されるということです。

イベント発生の 3 つ目のオプションは until() メソッドです。このメソッドは最初のレスポンスが「NULL 以外」になるまで、それぞれのリスナーに通知します。ですから、「NULL 以外」が $respons 変数には入ります。

```
$response = Event::until('dinner.time');
```

これでイベントの発行が分かりました。今度はもう一方の側面、どうやって自分自身をリスナーとして登録するかを学びましょう。なにせディナーを逃したくありませんからね！

12.2 イベントをリッスンする

イベントをリッスンするために使うのは…そうです。あなたの想像通りです。listen() メソッドです。文字列で監視したいイベント名を指定し、そのイベントが発行された時の処理をクロージャーで記述します。クロージャーの返却値がレスポンスとして返されます。見てみましょう。

```
Event::listen('dinner.time', function() {
        return ' 食い物ありがとう！';
});
```

$val = Event::first('dinner.time'); をこの時点で実行すれば、$val の値は「食い物ありがとう！」になります。単純ですね。

12.3 イベントにパラメーターを渡す

発生させたイベントに追加の情報を渡すには、値を配列にします。するとリスナーの無名関数に渡ります。こんな感じになります。

```
$responses = Event::fire('dinner.time', array(1, 2, 3));
```

リスナーの無名関数に、値を受け取るパラメーターを加えましょう。ですから：

```
Event::listen('dinner.time', function($one, $two, $three) {
        return ' 食い物ありがとう！';
});
```

とっても簡単です！ディナーと同時に届きそうです。

12.4 Ｌａｒａｖｅｌのイベント

以下が Laravel のコアに含まれるイベントのリストです。通常、あまり魅力的なものではありませんが、一度あなたが Laravel 導師のレベルに達すれば、Laravel フレームワークを思い通りに屈服させる必要を感じるでしょう。その時にはこれらのイベントを使用したくなるでしょう。

```
Event::listen('laravel.log', function($type, $message){});
```

新しく Laravel のログエントリが作成された

```
Event::listen('laravel.query', function($sql, $bindings, $time){});
```

SQL クエリーが実行された

```
Event::listen('laravel.done', function($response){});
```

Laravel を終了する時。Respons は既に送られている。

```
Event::listen('404', function(){});
```

ページが見つからない時

もっとたくさんあります。

> 注意：他のイベントとの名前の重複を避けるために、自分のイベントにはアプリケーション名を先頭につけるのが良いでしょう。Laravel のイベント名が `laravel` で始まっている理由です。

12.5 使用例

ではどういう場面でイベントは利用できるのでしょうか？簡単なタスク管理アプリケーションがあると想像してください。このようなタイプのアプリケーションは明らかにユーザーというオブジェクトを持ち、使用するためにはログインが必要です。ユーザーオブジェクトはシステムに追加できるようにします。Laravel のイベントでユーザー追加処理を考えてみましょう。

```
$new_user = array(.. ユーザーの詳細はここへ ..);
$u = new User($new_user);
$u->save();

Event::fire('myapp.new_user', array($u->id));
```

ご覧のとおり、新しいユーザーの ID 値は `myapp.new_user` イベントに渡され、その結果全てのイベントリスナーは、ユーザーが作成されたことを認識します。

では、アプリケーションに拡張機能を作成することにしましょう。私達の拡張機能は既に存在するアプリケーションのコードに影響を与えてはいけません。これは追加のライブラリー、もしくはバンドルであり、メインアプリケーションの外側にあります。

私達の拡張機能では、ユーザーネームにプレフィックスを使います。（拡張機能には役に立ちます。）ユーザーイベントをリッスンすることで、実現しましょう。

```
Event::listen('myapp.new_user', function($uid) {
        $user = User::find($uid);
        $user->name = 'Myapp_'.$user->name;
        $user->save();
});
```

これでリスナーができました。ユーザーID を受け取ります。これを使うことで、データベースから新しいユーザーを受け取り、変更できます。

これでアプリケーションに機能を追加しても、コアファイルに手を加えずに済む方法を学びました。機能を拡張する必要がありそうな部分に、イベント発行、もうしくは `fire()` メソッドを使うのは、アプリケーション作者の責任です。イベントがなかったら、アプリケーションのソースコードを変更しなくてはならなかったでしょう

13 Ｂｌａｄｅテンプレート

Laravel の Blade テンプレートエンジンはこざっぱりとした文法を PHP に埋め込んでビューにするものです。更に、Laravel に存在する機能を美しく利用できるたくさんの短縮形を含んでいます。Blade テンプレートはデフォルトでキャッシュされますので、とても早いです！

いつものとおり、早速見てみましょう。

13.1 基本

Blade テンプレートを利用するには、ビューの拡張子に.php の代わりに、.blade.php を使うだけのシンプルさです！

PHP フレームワークでビューを使う時、こんなふうにしていると思います……

```
<?php echo $val; ?>
```

PHP ショートタグはこれを少し短くしてくれます。

```
<?=$val?>
```

でも、もっと改善できます。では Blade が同じ echo 文をどう処理するか見てみましょう。

```
{{ $val }}
```

きれいでしょ！括弧内の空白は任意に入れてください。私は入れたほうが見た目が良いと思います。二重のかぎ括弧の中が評価され、echo されます。どんな PHP コードも記述できます。例えば：

```
{{ 5 * time() }}
```

このコードも上手く動作します。これが Blade が行うことのすべてです。`{{ 5 * time() }}` を `<?php echo 5 * time(); ?>` に置き換えてくれます。これが Blade が行うことの全てです。もし Blade を使っていてトラブルが起きたら、これをよく思い出してください。

13.2 ロジック

foreach() ループはどうなっているのでしょう。私はよく使います。if と else も同様にたくさん使います。Blade はこうした条件文すべてを魔法の @ 記号を使って単純にします。見てみましょう。

```
@foreach ($users as $user)
        <div class="user">{{ $user->name }}</div>
@endforeach
```

醜い php タグが一掃されただけでなく、書くのが少しばかり早くなりました！if と elseif はどうでしょう？結果は想像のとおりです。単に <?php を @ に置き換え、?> を省略します。みんな同じです！どうなるかといえば……

```
@if ($user->name == 'Dave')
        <p>Welcome Dave!</p>
@else
        <p>Welcome Guest!</p>
@endif
```

とっても簡単です。Blande で使える他の操作です。そっくりでしょう。

```
@for ($i =0; $i < 100 - 1; $i++)
    Number {{ $i }}<br />
@endfor
```

これはループですね。そして…

```
@forelse ($users as $user)
    {{ $user->name }}
@empty
    <p>ユーザーはありません。</p>
@endforelse
```

最後は少しだけ特別です。forelse() 文は foreach() ループですが、@empty が追加されており、配列が空の場合に出力されます。とても使いやすく、余計な if 文を付け加える必要がありません。

13.3 Ｂｌａｄｅレイアウト

Blade は書くと複雑になったり、ネストしたりするコードに対し、別の手法を提供しています。きれいなシンタックスを使い、フレームワークの中に組み込まれているレイアウトとしては最高のものでしょう。使ってみれば、その美しさを理解できます。Blade ビューによる、主要なテンプレート見てみましょう。（この場合、名前は template.blade.php です。お好きにどうぞ。）

```
<!DOCTYPE HTML>
<html lang="en-GB">
<head>
        <meta charset="UTF-8">
        <title>@yield('title')</title>
</head>
<body>
        <div class="header">
                <ul>
                @section('navigation')
                        <li><a href="#">ホーム</a></li>
                        <li><a href="#">ブログ</a></li>
                @yield_section
                </ul>
        </div>

        @yield('content')

</body>
</html>
```

メインテンプレートで使用されている @yield() メソッドはコンテンツの位置を指定しています。このレイアウトを使用するビューが渡す内容で埋められます。このメソッドにコンテンツ範囲のニックネームを渡してください。

@section() と @yield_section はコンテンツ範囲を定義します。いくつかデフォルトのデーターを定義していますが、後で置き換えることもできます。ビュー(page.blade.php) を見てみましょう。たった今作成したテンプレートを使っています。

```
@layout('template')

@section('title')
Dayle の Web ページ！
@endsection

@section('navigation')
        @parent
        <li><a href="#">About</a></li>
@endsection

@section('content')
        <h1>ようこそ！</h1>
        <p>Dayle の Web ページへようこそ！</p>
@endsection
```

このビューの中で @layout() メソッドを使用し、直前に作成した、使用するテンプレートのビュー名を指定します。@section と @endsection はこのメソッド間の内容で、@yield() で定義されたコンテンツ範囲を置き換えます。

今回の場合、navigation セクションの中に、@parent が content 領域にありますが、Blade はこれをベーステンプレートの内容に置き換えます。

これをリターンすると…

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

ルート／アクションで指定して、私達のページを表示してみましょう。レイアウトテンプレートでラップされて、このように表示されます。

```
<!DOCTYPE HTML>
<html lang="en-GB">
<head>
        <meta charset="UTF-8">
        <title>Dayle の Web ページ！</title>
</head>
<body>
        <div class="header">
                <ul>
                        <li><a href="#">Home</a></li>
                        <li><a href="#">Blog</a></li>
                        <li><a href="#">About</a></li>
                </ul>
        </div>

        <h1>ようこそ！</h1>
        <p>Dayle の Web ページへようこそ！</p>

</body>
</html>
```

素晴らしい！いくらでも好きなだけテンプレートを使えます！シンプルな Blade ビューです！

ところで、Action やルートから @section コンテンツを指定したい時は？シンプルに Secion::inject() メソッドにセクション名、コンテンツの内容を文字列で渡して呼び出してください。ビューに挿入されます。

```
Route::get('/', array('do' => function()
{
        Section::inject('title', 'My Site');

        return View::make('page');
}));
```

これで全てです！もうあなたは Blade できれいなビューを効果的に作成できます。あなたのデザイナーもこれを気に入ってくれるでしょう。

14 認証

多くのアプリケーションが認証を必要としています。もしブログを書くにしても、読者が勝手に新しいトピックを書くことができるのは好ましく思えないでしょう。もしくはデリケートな情報を扱っているのであれば、承認していないユーザーにアクセスしてもらいたくないはずです。

幸運なことに、Laravel は簡単かつ安全で、カスタマイズが可能な認証クラスを持っています。早速、どんなふうに扱うのか見て行きましょう。

14.1 準備

始める前にユーザーの詳細を保存しておく新しいテーブルを作成しましょう。お好きな名前を付けてもかまいませんが、users にしておけば、認証の設定ファイルを変更する必要がありません。では、スキマービルダーを使って、ピッタリのテーブルを作成する方法です。

```
Schema::create('users', function($table) {
        $table->increments('id');
        $table->string('username', 128);
        $table->string('password', 64);
});
```

好きなだけ追加のフィールドを追加してもかまいませんが、このまま先に進みましょう。認証をテストするためのサンプルユーザーも作成しましょう。最初にハッシュクラスについて説明する必要があるでしょう。

Hash クラスでとてもセキュアーな bcrypt アルゴリズムを使用し、パスワードをハッシュすることができます。使い方はシンプルです。例を見て下さい。

```
$pass = Hash::make(' パスワード文字列');
```

上のコードでパスワードに対し、bcrypt ハッシュを生成しています。平文テキストのパスワードをハッシュしてデーターベースに保存することにより、ユーザーのセキュリティーをアップさせます。これは Web アプリケーションの共通なプラクティスですので、検索すれば見つかるでしょう。

もし、ハッシュしたパスワードと値を比較したい時には、check() メソッドが利用できます。例えば：

```
Hash::check('my_pass', $pass);
```

これは論理型の結果を返し、true でマッチング成功、false で失敗です。これで、ハッシュパスワードのやり方を覚えました。サンプルユーザーを作成しましょう。Dexter という名前で作ります。このチュートリアルを書いている時、Dexter というテレビ番組を見ているのがバレているでしょうね。ものを書くにはぴったりなバックグラウンド・ノイズです。コーディングするときにお試しください。本当に良いですよ！Dexter で先に進みましょう。

```
DB::table('users')->insert(array(
        'username' => 'Dexter',
        'password' => Hash::make('knife')
));
```

ではここで、使用したいデフォルトの認証ドライバーを選びましょう。eloquent か fluent です。

fluent ドライバーはデータベースとのやり取りで Fluent を使用し、Auth::user() を呼び出した時にリターンされるオブジェクトはユーザーテーブルのレコードを表したオブジェクトになります。eloquent ドライバーはユーザーを表す Eloquent モデルを代わりに返します。

認証ドライバー、テーブルやオブジェクト名、フィールド名の設定は全て application/config/auth.php で行われています。

```
return array(

        'driver'   => 'eloquent',

        'username' => 'email',

        'model'    => 'User',

        'table'    => 'users',
);
```

driver を fluent クエリードライバーを使用する設定の fluent へ、username 設定項目をメールアドレスでなく、username を認証に使用するように username に変更してください。

```
return array(

        'driver'   => 'fluent',

        'username' => 'username',

        'model'    => 'User',

        'table'    => 'users',
);
```

14.2 フォームを用意する

よろしい！ではログインフォームにとりかかりましょう。可愛いい Blade で作成します！皆さんも好きでしょう？さあ、やりましょう。login.blade.php を作成します。

```
{{ Form::open('login') }}

        <!-- check for login errors flash var -->
        @if (Session::has('login_errors'))
                <span class="error">ユーザー名かパスワードが正しくありません。</span>
        @endif

        <!-- username 領域 -->
        <p>{{ Form::label('username', 'Username') }}</p>
        <p>{{ Form::text('username') }}</p>

        <!-- password 領域 -->
        <p>{{ Form::label('password', 'Password') }}</p>
        <p>{{ Form::password('password') }}</p>

        <!-- submit ボタン -->
        <p>{{ Form::submit(' ログイン') }}</p>

{{ Form::close() }}
```

これはうつく…

しーーー。いくつか前の章が役立っています。私はあなたを洗脳しているわけでありません。あなたが自分でできるようになったのです。確かに美しいフォームです！フォームを表示する、ナイスなルートを設定しましょう。

```
Route::get('login', function() {
        return View::make('login');
});
```

同じルートで POST に変更したものも作りましょう。ログインフォームを送信された時に使うものです。これが同じ URL を使い分ける方法です。

```
Route::post('login', function() {
        return ' ログインフォームが送信されました。';
});
```

では、ルートが正しく動作するか確認しましょう。私は確実に動くことを確信していますが、私の習慣です。'http://localhost/login' にアクセスし、偽のデーターを入力し、フォームを送信すれば、**ログインフォームが送信されました。**が表示されるでしょう。

14.3 ログインの処理

ログインできるようにするには、セッションドライバーを用意する必要があることに注意してください。Larave のログインはセッションに保存されます。application/config/session.php を見てもらえばオプション全てがリストされています。最適なものを選んでください。

素晴らしい！ではログインを試みることができるように処理しましょう。POST のルートで行います。最初はフォームからポストされた入力データーを手に入れましょう。

```
Route::post('login', function() {

        // get POST data
        $userdata = array(
                'username' => Input::get('username'),
                'password' => Input::get('password')
        );

});
```

いいですね！ポストデーターが手に入りました。では使用しましょう。Auth::attempt() メソッドで、username と password でログインできるか調べます。このメソッドの素晴らしい点は、成功したら自動的に「ログイン」セッションを作成してくれることです！Mucho conveniento! （スペイン語を勉強したことはありません。ごめんなさい…）

```
Route::post('login', function() {

        // POST データ入手
        $userdata = array(
                'username' => Input::get('username'),
                'password' => Input::get('password')
        );

        if ( Auth::attempt($userdata) )
        {
                // ログインできたので、ホームへ戻る
                Redirect::to('home');
        }
        else
        {
                // 認証失敗！ログインへ戻る
                    Redirect::to('login')
                        ->with('login_errors', true);
                // どんなエラー通知も渡せます
                // 私はこの方法がお気に入りです :)
        }

});
```

認証が成功し、ログインセッションが作成されたら、ホームルートへリダイレクトされます。完璧です。しかし、先に進む前にテストのためにも、ログアウトルートを作成しておきましょう。

Route::get(‘logout’, function() {

```
        Auth::logout();
        return Redirect::to('login');
});
```

Auth::logout() メソッドを使うことで、ログインセッションは破壊されます。そしてログインページが表示されます。さあ、ログアウトが終わりました。

これでログインの動作を完璧にしました。スーパーシークレットなホームページが作れるようになりました。ログアウトリンクを付け加え、ユーザーがログインするのを迎え入れましょう。

```
//---- ルート ----

Route::get('home', function() {
        return View::make('home');
});


//---- ビュー（home.blade.php） ----

<div class="header">
        ようこそ、{{ Auth::user()->username }} さん！<br />
        {{ HTML::link('logout', 'Logout') }}
</div>

<div class="content">
        <h1>リス情報</h1>
        <p>ここは私達のスーパーレッドリスに関する情報ページです。</p>
        <p>気をつけてください。灰色リスが見張っています。</p>
</div>
```

さあ、これでログインループのテストができます…ログインは、ウェルカムページで歓迎します。ログアウトし、繰り返してください。最低でも８００回やると心が休まります。心配しないで。これは普通の行いです。他には…PHP 導師の 1 時間に支払う方法もあります。

上の例の Auth::user() でお気づきになったでしょうが、このメソッドは現在ログインしているユーザーを表す Fluent データベース結果オブジェクトを返します。id を見つけたり、ウェルカム情報を出力するのにとても便利です。

14.4 ルートの保護

オッケー、これでほとんど終わりました！最初に、小さなバグや、セキュリティ問題に注意をはらいましょう。システムをログアウトし（これを読んでいる機械では無いですよ。サイトの話です。）ホームURL を見て下さい。

オー、なんてこった。灰色リスがログインもしないで私達の攻撃プランを見ています！これを正す必要があります。しかも、システムにログインしないと、ユーザーのユーザー名にアクセスするときにインデックスが定義されていない（もしくは似たような）エラーが発生します。

心を過去に飛ばしてください。解決策はこの本の最初の近くの影に潜んでいます。なに？影がない？影をつけてもらうために、出版社に支払ったというのに……まあ、いいでしょう。話を続けましょう。ルートフィルターを覚えていますか？ルーティング前に実行されるフィルターのコードを使用し、もし何かをリターンしたら、それはルートから戻ったところで使われるでしょう。ワォ、多くの可能性を秘めています。

簡単ですから、あなたはテイラー(Laravel 開発者）がアマチュアのマインドリーダーで、彼はこのチュートリアルを書いていること、そしてログインしていないユーザーからルートを保護する必要があると知っていたと推測するでしょう。彼が Laravel に標準で'auth' フィルターを作った理由はこれでしょう。フィルター自体を見てみましょう。（routes.php の中に見つかります。）

```
Route::filter('auth', function()
{
        if (Auth::guest()) return Redirect::to('login');
});
```

きれいですね！

Auth::guest() メソッドを知っていますか？とても表現的なメソッドで、現在のリクエストがログインしていないユーザーからですと、true を返します。使いやすいでしょう！Auth::check() は、ユーザーがログインしているかという真逆のチェックも出来るのです。私たちは全く同じ事を行うメソッドを知っていますが、しかし目的と一致し、的を射たメソッドの名前や、ソースの中でよりきれいな正しい表現になるものを使いましょう。

見ての通り、ユーザーがログインしていないのなら、Auth フィルターはログインページへのリダイレクトをリターンします。私達のルートに指定されるビューを書き換えてください。ホームルートへ送るために行うべきことは、これだけです。

```
Route::get('home', array('before' => 'auth', 'do' => function() {
        return View::make('home');
}));
```

これで、ホームルートは保護されました。未定義の notice は表示されないでしょうし、認証されていないリス……ユーザーはこれ以上ホームページを見ることができなくなりました。覚えておくべきことは、auth フィルターをログイン URI に適用しないことです。ひどいループを経験することになります。

14.5 カスタマイズ

私はあなたが何を考えているか分かります。Eloquent や Fluent を使いたくない時どうするかですね。これでは制限がありますね。

ねえ、これは Laravel ですよ。これまでに理解しておくべきでしたね！Laravel では `Auth` ドライバーとして知られるカスタムクラスを作成する方法を提供しています。これでパラメーターを変更し、どんな認証システムでもお好きに取り入れることができます。シンプルに新しいクラスを作成してください。私は `application/libraries` に作成するのが好きです。だって自動でロードしてくれますからね！

```
// application/libraries/myauth.php
class Myauth extends Laravel\Auth\Drivers\Driver {

        public function attempt($arguments = array())
        {

        }

        public function retrieve($id)
        {

        }

}
```

あなたが作成する認証ドライバーは `Laravel\Auth\Drivers\Driver` を拡張する必要があります。上記の 2 つのメソッドを作成します。最初のメソッドは `username` と `password` を配列で受け取り、独自の認証を行います。認証に成功したら親の `login()` メソッドを呼び、Laravel に認証が有効であることを知らせます。例えば…

```
public function attempt($arguments = array())
{
        $username = $arguments['username'];
        $password = $arguments['password'];

        $result = my_login_method($username, $password);

        if($result)
        {
                return $this->login($result->id, array_get($arguments, 'remember'));
        }

        return false;
}
```

login メソッドは id（後ほどユーザーを取得するために使われます）と引数配列の'remember' キーの値を受け取ります。認証に失敗したら、いつも false を返します。

retrieve() メソッドは先ほど login() メソッドに渡したのと同じ id を受け取り、現在のユーザーを表すオブジェクトをリターンするのに使用します。例えば…

```
public function retrieve($id)
{
        return get_my_user_object($id);
}
```

素晴らしい！これで認証ドライバーが動かせます。これを使用する前に Auth クラスに登録する必要があります。以下のコードを application/start.php に付け加えてください。

```
Auth::extend('myauth', function() {
        return new Myauth();
});
```

Auth::extend() の最初の引数としてあなたの認証ドライバーの識別子を渡します。２つ目のパラメーターはクロージャーでクラスの新しいインスタンスをリターンするために使用します。

後やるべきことで残っているのは、application/config/auth.php ファイルにこの新しい認証ドライバーを設定することです。

```
'driver' => 'myauth',
```

これで普段通り、認証システムを使えるようになりました。

15 ブログチュートリアル

遂に、最初の完全な Web アプリケーションを作成する時がやって来ました。もし、まだ他のポストを呼んでいないのであれば、ブログを作成するために必要な知識は以下のリストですので、先にお読みください。

- ルート
- マイグレーション
- Eloquent
- Blade テンプレート
- 認証
- フィルター

プランを確認し、実際に何を作ろうとしているのか理解してください。

ああ、そうだ。もし、これから作成するものをご覧になりたいのでしたら、GitHub でこのチュートリアルのソースコード[1]を参照してください。

15.1 デザイン

作成するのは 3 ページのブログです。最初のページはブログポストのリストです。Wordpre… Wordpush ブログのフロントページに似ています。次のページは記事のフルビューで、読みたい記事をクリックした時に表示されるものです。最後のページは投稿を書くためのページです。

通常、ポストの編集と削除ができるようにしますが、あとから付け加えられるものです。ログインページは数に入れていません。それはどんなアプリケーションでも認証に含まれて使用されるからです。

少し、データベースのスキマーについて考えてみましょう。プログポストをテーブルに保存する必要があることは分かっています。更にユーザーもテーブルに保存し、その両方にリレーションを設定する必要があります。マップとフィールドを考えてみましょう。

テーブル：`posts`

Field	Type
id	INTEGER
title	VARCHAR(128)
body	TEXT
author_id	INTEGER
created_at	DATETIME
updated_at	DATETIME

[1] https://github.com/codehappy/blog-tutorial-code

テーブル：users

Field	Type
id	INTEGER
username	VARCHAR(128)
nickname	VARCHAR(128)
password	VARCHAR(64)
created_at	DATETIME
updated_at	DATETIME

よし。これでいいでしょう。Post オブジェクトの author_id フィールドはポストの作者の User オブジェクトを表しています。

では、仕事にとりかかりましょう！

15.2 基本的な準備

まず最初にデータベースとセッションドライバーの準備をしましょう。これまでの学習でどうややるかはわかっていることでしょう。私はデータベースにいつも通り mySQL を使っています。

では、テーブルのマイグレーションを作成しましょう。ポストのマイグレーションに Artisan を使います。

```
php artisan migrate:make create_users
```

デフォルトのユーザーアカウントの設定に加え、今回のスキームを up() に設定します。

```
Schema::create('users', function($table) {
        $table->increments('id');
        $table->string('username', 128);
        $table->string('nickname', 128);
        $table->string('password', 64);
        $table->timestamps();
});

DB::table('users')->insert(array(
        'username' => 'admin',
        'nickname' => 'Admin',
        'password' => Hash::make('password')
));
```

down() も今回の設計に合わせ、変更します。

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

さらに posts のマイグレーションも行いましょう。そう来ると思ったでしょう？

```
php artisan migrate:make create_posts
```

up() のスキーマーです。

```
Schema::create('posts', function($table) {
        $table->increments('id');
        $table->string('title', 128);
        $table->text('body');
        $table->integer('author_id');
        $table->timestamps();
});
```

このテーブルをぶち壊すコードを down() にいれましょう。

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

これらのテーブルを一緒に作成するために、マイグレーションを実行します。

```
php artisan migrate:install
php artisan migrate
```

最後に application/config/auth.php を修正し、認証を変更しましょう。認証ドライバーに fluent を使用し、ログインの識別に username を使うように設定します。

```
return array(

        'driver'   => 'fluent',
        'username' => 'username',
        'model'    => 'User',
        'table'    => 'users',
);
```

さあ、これで Eloquent モデルを作成できる用になりました。

15.3 Eloquent モデル

そうです。あなたは Eloquent モデルをどう作成すれば良いかしっかりと分かっています。ですから、ソースコードをただ眺めてください。

```
class User extends Eloquent
{
        public function posts()
        {
                return $this->has_many('Post');
        }
}

class Post extends Eloquent
{
        public function author()
        {
                return $this->belongs_to('User', 'author_id');
        }
}
```

ナイスでシンプルです。User オブジェクトは複数のポストを持ち (has_many)、Post オブジェクトはユーザーに所属 (belongs_to) しています。

15.4 ルート

これでオッケイ。テーブルができました。モデルもできました。私のワークフローに従って、続いて必要なルートのプレースフォルダーを作成しましょう。

```
Route::get('/', function() {
        // ポストのリストを表示
});

Route::get('view/(:num)', function($post) {
        // 単ページ表示
});

Route::get('admin', function() {
        //  新ポストフォーム表示
});

Route::post('admin', function() {
        // 新ポストフォームの処理
});

Route::get('login', function() {
        // ログインフォーム表示
});

Route::post('login', function() {
```

```
        // ログインフォームの処理
});

Route::get('logout', function() {
        // システムからログアウト
});
```

そうです。これでおしまいです。ご覧の通り、私は GET と POST をフォーム表示と処理に分け、同じ URI で使用しています。

15.5 ビュー

アプリケーションのために Blade レイアウトをラッパーとして作成しましょう。

templates/main.blade.php

```
<!DOCTYPE HTML>
<html lang="en-GB">
<head>
        <meta charset="UTF-8">
        <title>Wordpush</title>
        {{ HTML::style('css/style.css') }}
</head>
<body>
        <div class="header">
                <h1>Wordpush</h1>
                <h2>Code is Limmericks</h2>
        </div>

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

ご覧のとおり、とても単純な HTML5 のテンプレートです。CSS を使用しています。（CSS はカバーしません。ブログを可愛く見せるために使っているだけで……しかし、この本はデザインブックではありません。）yield() で content エリアを取り込んでいます。

ログインフォームにとりかかりましょう。これで作者は新しいプログを書くことができます。このビューは前回のチュートリアルから持ってました。Blade レイアウトに合わせてハックしています。これは良い練習です。再利用できるものはなんでも使い、最後には自分専用の Laravel 開発ツールキットをつくり上げることになります。

pages/login.blade.php

```
@layout('templates.main')

@section('content')
        {{ Form::open('login') }}

                <!-- フラッシュ変数に保存してあるログインエラーをチェック -->
                @if (Session::has('login_errors'))
                        <span class="error">ユーザー名かパスワードが正しくありません。</span>
                @endif

                <!-- username フィールド  -->
                <p>{{ Form::label('username', ' ユーザー名') }}</p>
                <p>{{ Form::text('username') }}</p>

                <!-- password フィールド -->
                <p>{{ Form::label('password', ' パスワード') }}</p>
                <p>{{ Form::password('password') }}</p>

                <!-- 送信ボタン -->
                <p>{{ Form::submit(' ログイン') }}</p>

        {{ Form::close() }}
@endsection
```

この通り、ログインフォームを新しく Blade レイアウトとして作成しました。「ポスト新規作成」フォームを作りましょう。

pages/new.blade.php

```
@layout('templates.main')

@section('content')
        {{ Form::open('admin') }}

                <!-- タイトルフィールド -->
                <p>{{ Form::label('title', ' タイトル') }}</p>
                {{ $errors->first('title', '<p class="error">:message</p>') }}
                <p>{{ Form::text('title', Input::old('title')) }}</p>

                <!-- 本文フィールド -->
                <p>{{ Form::label('body', ' 本文') }}</p>
                {{ $errors->first('body', '<p class="error">:message</p>') }}
                <p>{{ Form::textarea('body', Input::old('body')) }}</p>

                <!-- 送信ボタン -->
                <p>{{ Form::submit(' 作成') }}</p>
```

```
        {{ Form::close() }}
@endsection
```

15.6 コーディング

遂に、手を汚す時が来ました。認証まわりから始めましょう。ログインルートとログインフォームビューを結びつけましょう。

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

そんなに難しくありませんね。今度は POST ルートを通常通り処理しましょう。

```
Route::post('login', function() {

        $userdata = array(
                'username' => Input::get('username'),
                'password' => Input::get('password')
        );

        if ( Auth::attempt($userdata) )
        {
                return Redirect::to('admin');
        }
        else
        {
                return Redirect::to('login')
                        ->with('login_errors', true);
        }
});
```

では、システムにログインしてみましょう。このトピックに理解できないところがあれば、以前のポストを参照してください。何も新しいことは、今回やっていません。

ログインのテストが行えるように、ログアウトのルートを作成しましょう。

```
Route::get('logout', function() {
        Auth::logout();
        return Redirect::to('/');
});
```

小さなプロフィールセクションをメインテンプレートの header に加えましょう。これでシステムのログイン、ログアウトができるようになります。

```
<div class="header">
        @if ( Auth::guest() )
                {{ HTML::link('admin', ' ログイン') }}
        @else
                {{ HTML::link('logout', ' ログアウト') }}
        @endif
        <hr />

        <h1>Wordpush</h1>
        <h2>Code is Limmericks</h2>
</div>
```

今度は admin ルートを auth フィルターに加えましょう。admin だけにルートの保護を施せば良いことに注目してください。これで、人々は私達のブログをブラウジングできますが、書き込むことはできません。

```
Route::get('admin', array('before' => 'auth', 'do' => function() {
        // 新ポストフォームの表示
}));

Route::post('admin', array('before' => 'auth', 'do' => function() {
        // 新ポストフォームの処理
}));
```

admin の GET ルートで、表示する新ポストフォームを付け加えましょう。このビューはログイン済みのユーザーだけが利用できますから、user オブジェクトの ID フィールドで作者を判別できます。

```
Route::get('admin', array('before' => 'auth', 'do' => function() {
        $user = Auth::user();
        return View::make('pages.new')->with('user', $user);
}));
```

現在ログインしているユーザーを取り扱えるようになりました。「新規ポストビュー」に作者を見分けられる情報を追加しましょう。隠しフィールドを使うのはトリックです。

```
{{ Form::open('login') }}

        <!-- 作者 -->
        {{ Form::hidden('author_id', $user->id) }}

        <!-- タイトルフィールド -->
        <p>{{ Form::label('title', 'Title') }}</p>
        <p>{{ Form::text('title') }}</p>

...
```

これで作者を識別できます。「新規ポスト作成」ページも用意できました。これにリンクを作成する必要はありません。隠しておきたいですからね。新ポストを作成したい時は、URL に/admin とタイプしましょう。

では、新ポスト作成の処理にかかりましょう。

```
Route::post('admin', function() {

        //  POST データーとして新しい記事を受け取る
        // これは複数代入を使用するより安全
        $new_post = array(
                'title'     => Input::get('title'),
                'body'      => Input::get('body'),
                'author_id' => Input::get('author_id')
        );

        // 新しいデーターに対するバリデーションルールの設定
        // きっとあなたはもっと良く改善してくれるでしょう
        $rules = array(
                'title'     => 'required|min:3|max:128',
                'body'      => 'required'
        );

        // バリデーション実行
        $v = Validator::make($new_post, $rules);

        if ( $v->fails() )
        {
                // ホームへリダイレクト
                // errors と入力データー、
                // 現在のログインデーターを渡す
                return Redirect::to('admin')
                                ->with('user', Auth::user())
                                ->with_errors($v)
                                ->with_input();
        }

        // 新しいポストを作成
        $post = new Post($new_post);
        $post->save();

        // 新しいポストの表示ページへリダイレクト
        return Redirect::to('view/'.$post->id);

});
```

ブログポストを作成できるようになりました。先へ進みましょう！いくつか記事を書いたら、「全てのポストをリスト」ビューで表示します。

実際に行なってみましょう。

```
Route::get('/', function() {
        // 作者のポストを eager ロードしする
        $posts = Post::with('author')->all();

        return View::make('pages.home')
                ->with('posts', $posts);
});
```

ブログポストをリストするビューも必要ですね。ではどうぞ。

pages/home.blade.php

```
@layout('templates.main')

@section('content')
        @foreach ($posts as $post)
                <div class="post">
                        <h1>{{ HTML::link('view/'.$post->id, $post->title) }}</h1>
                        <p>{{ substr($post->body,0, 120).' [..]' }}</p>
                        <p>{{ HTML::link('view/'.$post->id, 'Read more &rarr;') }}</p>
                </div>
        @endforeach
@endsection
```

最後に、ブログのフルビューが必要です。こうなります。

pages/full.blade.php

```
@layout('templates.main')

@section('content')
        <div class="post">
                <h1>{{ HTML::link('view/'.$post->id, $post->title) }}</h1>
                <p>{{ $post->body }}</p>
                <p>{{ HTML::link('/', '&larr; Back to index.') }}</p>
        </div>
@endsection
```

このビューを見られるようにするために、新しいルートを加えましょう。

```
Route::get('view/(:num)', function($post) {
        $post = Post::find($post);
        return View::make('pages.full')
                ->with('post', $post);
});
```

これで基本機能だけですが、完全に動作するブログが出来ました。他にどこを改善できるかさっと見てみましょう。どうぞ自分で機能を追加してみてください。Laravel のスキルが見についているかのテストになります。

15.7 機能

ページネーション

ポストページにページネーションを追加できます。そうすれば、ポストの数が増えた場合に便利になります。Laravel は Eloquent モデルにページネーションを簡単に追加できるようにしてくれていますから、これは本当に簡単です。

ポストの編集・削除

ポストを編集したり、削除したりする機能を付け加えることも、そう手間はかかりません。これによりブログのメンテナンスができるようになります。

ポスト別名

URL の中の `id` の代わりに、別名を使えるように出来ます。これは SEO に有利になります。保存した時のポストのタイトルから生成できます。それから、ルートのパラメーターとして別名を区別するようにします。

16 ユニットテスト

ユニットテストは Web の開発者にとって、とても便利なツールです。機能を追加したり、コードを変更した際に偶然他の機能に影響を与えてしまうと、テストに失敗します。ある開発者は、たとえ試験制作の環境であっても、すべての要求に答えるため、コードに取り掛かる前にテストを書きます。

Laravel はアプリケーションすべてをテストするために `tests` ディレクトリーを用意しています。PHPUnit のテストケースを実行するためのコマンドライン／インターフェイスである Artisan にヘルパーを付け加えることさえできます。

アプリケーションをテストするだけでなく、バンドルにテストケースを用意することもできます。実際、Laravel はフレームワークのコア機能をテストする目的のバンドルを用意しています。Laravel テストの github リポジトリ[1]で見つかります。

16.1 インストール

Laravel のインストール方法を再度取り上げるわけではありません！

Laravel は PHPunit ソフトウェアを自身のテストに使用しています。Laravel のテスト機能を使用する前に、このソフトウェアをインストールする必要があります。

OS 毎に PHPUnit のインストール方法はバラバラですので、PHP Unit の公式ドキュメントでインストールの指示を見るのが一番良いでしょう。こちらでインスール指示のページ[2]を見つけられます。

16.2 テストの作成

テストケースを見て下さい。ラッキーな事に Laravel はサンプルテストケースを用意しています！

```
// application/tests/example.test.php

class TestExample extends PHPUnit_Framework_TestCase {

        /**
         * Test that a given condition is met.
         *
         * @return void
         */
        public function testSomethingIsTrue()
        {
                $this->assertTrue(true);
        }

}
```

[1] https://github.com/laravel/tests
[2] http://www.phpunit.de/manual/current/ja/installation.html

見ての通り、ファイルに拡張子 test.php を付けて作成しました。クラス名は Test で始まり、PHPUnit_Framework_TestCase を拡張する必要があります。これらは Laravel の制限ではなく、PHPUnit の仕様です。

PHPUnit のテストケースは、いくらでもテストを含むことができ、test で始まる、キャメルケースのアクションとして記述します。テストに成功したか、失敗したかを決めるアサーションはいくつも含むことができます。

アサーションの完全なリストはPHPUnit のウェブサイトにあるドキュメント[3]で見つかります。

16.3 テストの実行

テストは artisan コマンドラインインターフェイスを利用して実行します。単純にすべてのテストケースを実行するには test コマンドを用います。

```
php artisan test
```

結果は：

```
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /home/daylerees/www/laravel/develop/phpunit.xml

.

Time: 0 seconds, Memory: 6.25Mb

OK (1 test, 1 assertion)
```

明るいグリーンで OK が表示されたら、テストを通りました。

もちろん、assertTrue() メソッドで、ture 値をチェックしているのですから、当然成功します。失敗するはずがありません。

では失敗してみましょう。パラーメーターを単に false に変更するだけです。

```
// ...
$this->assertTrue(false);
// ...
```

実行結果は：

[3] http://www.phpunit.de/manual/current/en/writing-tests-for-phpunit.html#writing-tests-for-phpunit.assertions

```
PHPUnit 3.6.10 by Sebastian Bergmann.

Configuration read from /home/daylerees/www/laravel/develop/phpunit.xml

F

Time: 0 seconds, Memory: 6.50Mb

There was 1 failure:

1) TestExample::testSomethingIsTrue
Failed asserting that false is true.

/home/daylerees/www/laravel/develop/application/tests/example.test.php:12
/usr/bin/phpunit:46

FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
```

テストが失敗したことを示す、明るい赤の数行が表示されます。なぜ失敗したかの詳細も表示されます。

もし、バンドルをテストしたければ、test コマンドにパラメータで指定します。例えば：

```
php artisan test mybundle
```

16.4 コアをテストする

Laravel のコアはユニットテスト済みです。自分でテストを実行してみたいのでしたら、以下の手順を行なってください。

テストのバンドルをインストールする

github[4]から tests のバンドルをダウンロードし、bundles/laravel-tests ディレクトリーの中に展開します。

これで、test:core コマンドだけで、コアテスとパッケージを実行できます。

```
php artisan test:core
```

[4] https://github.com/laravel/tests

17 キャッシュ

Laravel にはシンプルなキャッシュクラスが用意されており、必要な物をなんでも簡単に、好きなだけ長くキャッシュしておけます。混乱しますか？では、実例で見て行きましょう。

17.1 準備

キャッシュデーターを保存しておく方法は何通りか用意されており、application/config/-cache.php に使用したい方法のドライバーを指定します。オプションは'file'、'memchched'、'apc'、'redis'、'database' があります。効率を良くしたいのでしたら、'apc' か'memcached' を使用するべきですが、準備が簡単ですから、ここでは'file' を利用します。

```
'driver' => 'file',
```

17.2 値をセットする

キャッシュにデーターを保存するには、put() か forever() メソッドを使います。

put() メソッドは一定期間、値を保存するメソッドです。サンプルコードを見てみましょう。

```
$data = ' 複雑に生成されたデーター';
Cache::put('mydata', $data, 10);
```

$data が単純な文字列ではなく、処理を行うために時間のかかる複雑なアルゴリズムの生成結果であると考えてください。いつもこのデーターを処理したくありません。なので、キャッシュに保存しておきましょう。

メソッドの最初のパラメーターはキーです。キャッシュデーターの識別に使う文字列です。２つ目はデーター自信です。３つ目はキャッシュしておく時間を分で指定します。

forever() メソッドは、頭の２つのパラメーターだけを取ります。永久にキャッシュします。

17.3 値を取得する

キャッシュからアイテムを取り出すためには、キーを指定し、get() メソッドを利用します。例えば：

```
$data = Cache::get('mydata');
```

キャッシュアイテムが存在していなかったり、既に時間切れになっている時、メソッドはデフォルトで NULL を返します。しかしながら、２番目のパラメータに他のデフォルト値を指定することもできます。

```
$data = Cache::get('mydata', ' 代わりのデーター');
```

また、クロージャーを２番目のパラメーターとして指定すれば、関数が返す値をキャッシュデーターが存在しない時の値とします。クロージャーはキーが存在しない場合のみ、実行されます。

17.4 もっと良い方法

has() メソッドをキャッシュアイテムの存在チェックに使用すれば、おなじみのパターンになります。

```
if (! Cache::has('mydata'))
{
        $data = ' 複雑に生成されたデーター';
        Cache::put('mydata', $data, 10);
}

return $data;
```

しかしながら、remember() メソッドを使えば、もっとエレガントに解決できますよ。remember() メソッドはキャッシュに存在していればその値を返し、そうでなければ、指定された時間の間、２番目の値を保存し、その値を返します。例をご覧ください。

```
return Cache::remember('mydata', function () {
        return ' 複雑に生成されたデーター';
}, 10);
```

もちろんクロージャーはキーが存在していないか、期限切れになっていない限り、実行されません。

キャッシュを利用し、楽しんでください！

18 クラスのオートローダー

多くのフレームワークにおいて、どこにファイルを配置するのか、クラスをどうやってロードするのかという知識は、扱いにくい話題になります。しかし、Laravel には、アプリケーションのレイアウトに適用される厳格なルールは存在しません。Laravel のオートローダーはとても賢いライブラリで、色々なネーミングやサブフォルダーの記述を簡素にしています。これは、複雑なライブラリーやパッケージを安心して追加するのに十分な柔軟性も兼ね備えています。用意されている機能を見てみましょう。

18.1 マッピング

マッピングは一番シンプルなクラスのローディング方法です。クラス名とファイルの場所をキーと値のペアの配列としてオートローダーに渡せば、Laravel が残りを処理します。このオートローダーは効率的です。定義されたクラスが使用される時だけロードします。デフォルトでは Autoloder マッピングは start.php ファイルで定義されています。しかしどこからでもこのクラスを使用することができます。でも、start.php ファイルは早い時期にロードされますから、マッピングするファイルとしては良い選択でしょう。マッピングを見てみましょう。

```
// application/start.php

Autoloader::map(array(
        'Cloud' => path('app').'libraries/ff/cloud.php',
        'Tifa'  => path('app').'libraries/ff/tifa.php',
));
```

path('app') はあなたのプロジェクト中にある、application フォルダーの絶対パスを獲得するお手軽なヘルパーメソッドです。path() メソッドで他のフォルダーの絶対パスも手に入ります。次のリストを見て下さい。

メソッド	ディレクトリー
path('app')	application
path('sys')	laravel
path('bundle')	bundles
path('storage')	storage

マッピングの例を見てもらえば、クラス名を配列の索引とし、ファイルとその場所を値としてるのが分かるでしょう。もし、Cloud クラスを使いたいなら…

```
$c = new Cloud();
```

Laravel のオートローダーは、ロードする必要のあるクラスの定義を「見つけ出し ("detect"、PHP マジックメソッド)」続いてソースを `include()` します。

18.2 ディレクトリーマッピング

もしあなたがクラス名を小文字にしたものを名前としたファイルを沢山使っているのでしたら、ひとつひとつを定義するよりは、ディレクトリーを指定したいと思うでしょう。オートローダーの direcories() メソッドを使用できます。この場合、配列の値として場所とファイル名だけを指定する必要があります。例えば：

```
Autoloader::directories(array(
        path('app').'smurfs'
));
```

これで application/smurfs ディレクトリーの中に存在する全てのクラスは、クラス名を小文字にしたものを名前にしている限り、オートロードされます。

18.3 名前空間マッピング

PHP 5.3 では名前空間がサポートされました。完璧では無いですが、PSR-0 規約のファイルローディングも、可能です。PSR-0 では、名前空間はディレクトリー構造とクラス名で識別されます。ですから次の場合…

```
<?php namespace Paladin\Mage;

class Princess
{
        // so pretty
}
```

ファイルの場所は…

```
paladin/mage/princess.php
```

この規約を使用する前に、Laravel に名前空間のルートフォルダーがどこなのかを知らせる必要があります。Autoloader クラスの namespaces() を使用することで、これが指定できます。例えば：

```
Autoloader::namespaces(array(
        'Paladin' => path('libraries').'paladin'
));
```

ご覧のとおり配列をメソッドに渡しています。キーは名前空間のルート名で、この場合 Paladin です。配列の値はこの名前空間に当てはまるルートフォルダーを指定し、この例では allication/libraraies/paladin となります。

18.4 アンダースコア・マッピング

ええと、アンダースコアのマッピングは、実際必要無いですね。キーボードの右端だし。いつも場所を見て、確認し……ごめん、アンダースコア。私は君を正しく使っていないようだが、本当に愛していないんだ。キャメルケースも私には良く思えないんだから！

不満を漏らしてごめんなさい。後で個人的に話しましょう。なぜタイトルがアンダースコア・マッピングかという理由は、名前空間（と Laravel）より以前、PHP の「骨董」的な日々の頃に、よく見られたからです。多くの開発者はアンダースコアをファイルのディレクトリーを表すために採用しました。Laravel は `underscored` メソッドをこのタイプのクラスローディングに対し、用意しています。これも配列のキーでクラスのプレフィックスを表し、値でルートディレクトリーを示します。例えば：

```
Autoloader::underscored(array(
        'Paladin' => path('app').'jobs/pld'
));
```

これで `Paladin_Light_Knight` クラスは、`application/jobs/pld/light/knight.php` からロードされます。

これで Laravel を使って、オートロードするやり方を習得しました。もうソースに `include()` 文を貼り付ける必要は無くなりましたね。

Enrique Lopez に感謝します。彼は 2 か国語でコメントできるほど頭がよく、文字の制限をぶち破ってくれました。Gracias Enrique!

> Laravel を使い、私はどう書くかではなく、何をやりたいかを考えています。プログラミングを楽しんでいます。(Con Laravel pienso qué es lo que quiero y no como escribirlo. Me divierto programando.)

19 設定

Laravel はフレームワークで提供するほとんどの機能を調整できるように、多くの設定ファイルを application/config の下に用意しています。同じように、自分で使う設定ファイルが使えたら良いですよね？今日はラッキーデーです。だって、できるんですから！

19.1 新しい設定ファイルを作成する

Laravel の設定ファイルは application/config とその下のサブディレクトリーの中にある、ただの PHP ファイルで、配列を返します。例えば：

```
// application/config/ourconfig.php

return array(

        'size' => 6,
        'eat'  => true,
);
```

もっとわかりやすくするためにコメントを設定ファイルに使いましょう。私は Laravel のコメントスタイルを気に入っています。例えば：

```
// application/config/ourconfig.php

return array(

        /*
        |------------------------------
        | サイズ
        |------------------------------
        |
        | これは私のもののサイズです。
        |
        */

        'size'        => 6,
);
```

あなたなら、もっとわかりやすくできることでしょう！もう既に皆さん、Laravel の設定ファイルはキーと値のペアで、インデックスはキーを、そして値は…値であると気づいていることでしょう。

セットする値は、どんな値でも、PHP がサポートするオブジェクトでも指定できます。クロージャーでさえ可能です。クロージャーを使用できるので、ユーザーは他のソースから設定を読み込むことが可能となります。例えば：

```
// application/config/ourconfig.php

return array(

        /*
        |------------------------------
        | サイズ
        |------------------------------
        |
        | これは私のもののサイズです。
        |
        */

        'size'          => function() {
                $size = file_get_contents('path/to/file.json');
                $size = json_decode($size);
                return $size;
        },
);
```

これで、size の設定は JSON 形式のファイルから読み込まれます。シンプルですね！

> 翻訳者注：クロージャーをそのまま使用すると現在エラーになります。value() ヘルパーで囲ってください。

19.2 設定の読み込み

get() メソッドを使用し、設定を読み込むことができます。

```
$option = Config::get('ourconfig.size');
```

シンプルに文字列で、ファイルに続き設定名をピリオドで区切って渡しています。これで、値が戻ってきます。もし、設定ファイルがサブディレクトリーにあるのであれば、サブディレクトリーとピリオドを追加してください。例えば：

```
$option = Config::get('ourconfig.sub.directory.size');
```

時々、設定の配列全体を取得できると便利です。そうしたい時は、単純にオプションを付けず、ファイル名だけを指定してください。設定の配列をファイルから読み込んだものが返されます。

```
$option = Config::get('ourconfig');
```

19.3 設定をセットする

設定項目をセットするためには、set() メソッドを使います。最初のパラメーターがファイルとアイテム名で、get() と同じフォーマットです。２つ目のパラメーターがセットする値です。

```
Config::set('ourconfig.size', 7);
```

設定項目が設定ファイルの中に存在している場合、set() を使い、実行時に指定した値で上書きされ、保持されます。しかしながら、設定ファイル自身を書き換えるわけではないことに、注意してください。設定項目が存在しない時、set() メソッドは項目を作成しますが、これもファイルを書き換えるわけではありません。

アプリケーションで、変更可能な設定をできる限り設定ファイルに置くようにしてみてください。そのアプリケーションを移動したり、分割する場合に、設定が随分簡単になりますよ。

20 IoCコンテナ

IoC コンテナは扱いにくい主題で、多くの人々がドキュメントに書かれている記述で混乱しています。しばらくは私もその中の一人でした。慎重に行われた調査と、素晴らしい Laravel のコミュニティー(freenode IRC の #laravel に参加してください）のサポートで、この主題をうまいこと理解できました。このミステリアスな主題をいくらかは照らし出すことができるでしょう。

IoC は制御の逆転 (Inversion of Control) の略です。ここで、完全に説明して事を複雑にしたくないですし、オタクサイドの人達から書かれた多くの記事がネットにもあります。その代わりに、オブジェクトを解決 (resolve) するために、コンテナを「コントロールの逆転」か「Laravel にコントロールを戻す処理」のためのコンテナとして考えてください。

オブジェクトを解決する、これが「コンテナとは何なのか」ということの全てです。もしくはユニットテストで依存性の注入に使用されるものです。（後ほど説明します）あなたは IoC コンテナは、デザインパターンに関連するクラス間の通常の関連を持たせずに、複雑なオブジェクトを解決する、もしくはシングルトンパターンを保つための「ショートカット」であるとシンプルに考えてください。シングルトンについては後ほど。まずはコンテナにオブジェクトを登録するのをご覧下さい。

20.1 オブジェクトを登録する

大きな紫の恐竜を映し出すテレビが教えてくれるように、想像力を使いましょう。「ディスコボール」と言う名前のクラスをイメージしましょう。いたるところで使われまくる、様々なかっこいい目的に使われます。

残念なことに、私たちのディスコボールクラスは使用する前に、多くの設定をしなくてはならないのです。見てみましょう。

```
$db = new Discoball(Discoball::SHINY);
$db->configure_shinyness('max');
$db->spin_speed('8900rpm');
```

ああ！なんて多くの設定でしょう。これは、使用するたびにインタンスを生成し、ディスコボールの準備をしなくてはなりません。IoC コンテナにインスタンスの生成を任せましょう。コードサンプルに飛びつきましょう。

私はこれを start.php に書くのが好きなんですが、あなたはどこでも好きなところにどうぞ。オブジェクトを解決する前である限り、どこでもかまいません。

```
// application/start.php

IoC::register('discoball', function() {

        // 事前にオブジェクトを初期化
        $db = new Discoball(Discoball::SHINY);
        $db->configure_shinyness('max');
        $db->spin_speed('8900rpm');

        // 無名関数の戻り値としてオブジェクトを渡す
        return $db;
});
```

IoC::register() メソッドで、オブジェクトをコントローラーと共に登録します。最初のパラメーターは文字列で、後ほど解決する時に使う者です。私は discoboll という言葉がぴったりくるので、指定しました。２番目のパラメーターはオブジェクトをインスタンス化するのに使います。

クロージャーの中には、見たことのあるディスコボール設定コードがあります。そしてクロージャーで設定済みのディスコボールオブジェクトを返しています。

素晴らしい！オブジェクトが登録されました。これで全て見終わりました…冗談です。登録したオブジェクトをどうやって使用するか、見て行きましょう。

20.2 オブジェクトを解決する

これで、ディスコボールを登録できました。どうやって取り戻すのか見てみましょう。解決するために呼び出してみましょう。

```
$db = IoC::resolve('discoball');
```

これだけです！毎回新しいディスコボールのインスタンスを設定する代わりに、resolve() を呼び出しました。オブジェクトの識別子となる文字列を渡し、最初のセクションで作成した無名関数が IoC コンテナにより実行され、設定の終わったディスコボールオブジェクトのインスタンスが返ってきます。

便利で、多くのコードを省略できます！

あなたは好きなだけ多くのオブジェクトを登録し、解決できます。どうぞ試してください。続いてシングルトンを扱いましょう。

20.3 シングルトン

ディスコボールの解決は便利です。しかしインスタンス化にリソースを大量消費したとしたら、もしくは一度しかインスタンス化できないとしたらどうしましょう。register メソッドは、このような場合には使えません。なぜなら、クロージャーは resolve() が呼び出されるたび、毎回実行されますし、

そのたびに新しいオブジェクトのインスタンスが戻ってくるからです。これこそシングルトンデザインパターンを使う場所です。

シングルトンのデザインパターンは書いているクラスを決まったやり方に巻き込みます。static メソッで呼び出され、それ自身の同じインスタンスを戻すのです。この方法で、クラスは一度だけインスタンス化されます。

シングルトンデザインパターンについて、もっと情報がほしいのなら、手早く Google で検索するか、この主題に関する PHP API を調べてみてください。

シングルトンは便利ですが、特別なクラス構造で使用されます。IoC コンテナは `singleton()` メソッドを持ち、手順をもっとシンプルにしてくれます。そして特別な種類のクラスを要求しません。ディスコボールをシングルトンのインスタンスとして登録してみましょう。

```
// application/start.php

IoC::singleton('discoball', function() {

        // 事前にオブジェクトを初期化 instantiate our object as before
        $db = new Discoball(Discoball::SHINY);
        $db->configure_shinyness('max');
        $db->spin_speed('8900rpm');

        // 無名関数の戻り値としてオブジェクトを返す
        return $db;
});
```

ご覧のとおり、手順はオブジェクトを登録する時とほぼ一緒です。例外は `singleton()` メソッドを使っていることですが、同じパラメーターです。

ディスコボールを解決したい時、最初の `resolve()` の呼び出しで、無名関数は実行され、生成したオブジェクトは保存されます。それ以降の `resolve()` メソッドの呼び出しでは、同じオブジェクトインスタンスを返します。例えば：

```
// 無名関数が実行され、
// インスタンスが戻ってくる
$db = IoC::resolve('discoball');

// 上と同じディスコボールインスタンスが
// 戻ってくる
$another = IoC::resolve('discoball');
```

素晴らしい！もう一つ覚えておく価値があるのは、既にオブジェクトがインスタンス化されている場合、それを `singleton()` の第２パラメーターとして指定すれば、それ以降に `resolve()` を呼び出しても同じインスタンスが返ってくることです。例えば：

```php
$db = new Discoball(Discoball::SHINY);
IoC::singleton('discoball', $db);

// ディスコボールを保持している
$ball = IoC::resolve('discoball');
```

以降の章で、IoC コンテナと、ユニットテストに絡めた依存性の注入について見て行きましょう。

Andrew Smith さん、Julien Tant (AoSix) さん、購入いただきありがとうございます。Julien さんは、次のようにおっしゃっています。

> ハイ、Dayle さん。この本には大変感謝しています。Laravel フレームワークは素晴らしい！もしフランス語で読みたいのなら、http://www.laravel.fr/へどうぞ！

お二人とも、ありがとう。チュートリアルをフランス語に翻訳していただきました！

また、Proger_XP さんにも、購入いただきありがとうございます。チュートリアルをロシア語に翻訳していただき、laravel.ru でお読みいただけます。グッド・ジョブ！

21 暗号化

重要なデーターを守りたい時があるでしょう。Laravel は 2 つの異なったメソッドでこれをお手伝いします。復元不可能と復元可能暗号化です。これらのメソッドを見て行きましょう。

21.1 復元不可能暗号化

復元不可能な暗号化はユーザーのパスワードや取り扱いに注意するデーターを保存するのに最適です。復元不可能とは、暗号化文字列に変換はできるが、頭の痛い複雑な数学的アルゴリズムを使っているため、もとに戻すことは不可能です。

これぞパスワードの保存にはピッタリです！あなたの顧客はパスワードを知られる心配はしなくて良いです。それでも示されたパスワードをハッシュすることで、比較できますし、必要なら変更することもできます。

ハッシュとは細切れにしたり、暗号化したりして、ある文字列を別の文字列に変換することです。

非復元暗号化を使用して、どうやってハッシュするのか見て下さい。

```
$pass = Input::get('password');
```

「ユーザー登録」フォームからパスワードを取得しました。しかしこれは平文です。早速ハッシュしましょう。そうしたら、安全にデータベースに保存できます。

```
$pass = Hash::make($pass);
```

今までにも Laravel の内容をよく表しているメソッドを使ってきましたが、今回も新しいハッシュ (Hash) を作成 (make) します。$pass の値は、bcrypt 暗号化されたパスワードです。きれいですね。

ログインに使うユーザーのパスワードは受け取れました。今度はシステムにログインする前の認証でチェックする必要があります。データベースに保存してあるハッシュ値と値を比較するには、check() メソッドを使います。

```
$pass = Input::get('password');
if ( Hash::check($pass, $user->password) )
{
        // 認証成功
}
```

check() メソッドは 2 つのパラメーターを持ちます。ユーザーから受け取った平文の値と、保存されているハッシュ値です。マッチしていたならば true を返します。

もしデーターを最初のように復元する必要がある時は？復元可能暗号化を使いましょう。

21.2 復元可能暗号化

復元可能暗号化は、暗号化されたデーターをオリジナルの形に戻せます。まるで子供の頃遊んだスパイごっこで使った暗号表のようにです。

Laravel の Crypter クラスは、Mcrypt PHP 拡張で提供されている AES-256 暗号化を使用してしますので、クラスを使用する前に、この PHP 拡張がインストールされているのを確実にしてください！

Crypter クラスはシンプルなメソッド `encrypt()` と `decrypt()` の 2 つを持っています。まず、文字列を暗号化してみましょう。

```
$secret = Crypter::encrypt(' 私は本当にハローキティが好きです');
```

これで、私たちの少し汚い秘密は AES-256 暗号化されました。結果が戻ってきます。これは秘密を復元できなければ、使えません。暗号化したデーターをどうやって復元するか見て下さい。

```
$decrypted_secret = Crypter::decrypt($secret);
```

ご覧のとおり簡単です！シンプルに暗号化された文字列を decrypte() に渡し、復元された結果を受け取っています。

シリアルの箱のおまけでスーパー秘密の解読リングを手に入れた時の感覚をシミュレートするように、Crypter クラスを楽しんで使ってください。

22 ＡＪＡＸコンテンツ

現代的なアプリケーションには、次の HTTP リクエストをじっくりと待つ時間はありません。Web ブラウザの Javascript により、コンテンツを自動的に更新します。また、ページを再読み込みすることなく、情報をポストします。

Laravel の IRC チャンネルで、よく Laravel で Ajax をどう取り扱うのか尋ねられます。しかし、「他の HTTP リクエストと同じように」という単純な答えで、混乱してしまうのです。では、このフレームワークで Ajax リクエストの取り扱いを見て行きましょう。Javascript を通して HTTP リクエストを出す必要がありますが、見苦しくなるので、jQuery Javascript ライブラリーを例に使うことにしました。

22.1 テンプレートページ

最初のリクエストに備えて、ビューかテンプレートを用意しましょう。基本的なところをまとめてしまいます。

```
<!-- application/views/template.php -->
<!DOCTYPE HTML>
<html>
<head>
        <meta charset="UTF-8">
        <title>Ajax ページ</title>
</head>
<body>
        <div id="content">
                <p>当サイトにようこそ！</p>
        </div>
        <a href="#" id="load-content">コンテンツの読み込み</a>
</body>
</html>
```

これで、content の id を持つ <DIV> 要素で定義されたコンテンツエリアができました。ここにコンテンツを読み込むことにします。また、id に load-content を持つリンクも用意しました。これをトリガーに使い、新しいコンテンツを <DIV> に読み込みます。

このビューを表示する前に、読み込むルートを定義する必要があります。ベースルートに定義しましょう。

```
// application/routes.php
Route::get('/', function() {
        return View::make('template');
});
```

さて、これで http://localhost を訪れると、このサイトテンプレートで歓迎されます。しかし、コンテンツの読み込みリンクをクリックしても、Javascript 関係に取り掛かっていないので、何も起きません。しかし、読み込むものがないのなら、Javascript も必要ないでしょう。まず、content の <DIV> に読み込むコンテンツのために、新しいルートとビューを作成しましょう。

最初にビューです…

```
<!-- application/views/content.php -->
<h1>新コンテンツ</h1>
<p>これは AJAX で読み込まれるコンテンツです。</p>
```

それとこのコンテンツのためのルートですが、注意してもらいたのはテンプレートでコンテンツを埋め込むわけではないことです。テンプレートでやってしまえば、AJAX で新しいコンテンツを読みこむのは二度手間になります。

```
// application/routes.php
Route::get('content', function() {
        return View::make('content');
});
```

いいですね。これでメインのテンプレートと、AJAX を通じてコンテンツを読み込むための、第二のルートができました。それでは、Javascript にとりかかりましょう。

22.2 Ｊａｖａｓｃｒｉｔｐ

私は現在 PHP 開発者で、Javascript のスキルはこの章で皆さんを驚かせるほどではありません。こうしたアクションを実現するもっと良い方法を見つけたら、どうぞその方法にチャレンジしてください。（色々なやり方があります）きっと、皆さんのほうがもっと良くできるでしょう！

最初に、script.js という名前の新しいファイルを作成します。それを public/js に設置します。それと最新バージョンの jQuery を jquery.js という名前で、一緒に設置してください。では、メインのテンプレートに HTML::script() メソッドを使い、これらのファイルを参照するように、付け加えましょう。

```
<!-- application/views/template.php -->
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="UTF-8">
    <title>Ajax ページ</title>
</head>
<body>
    <div id="content">
        <p>当サイトへようこそ！</p>
    </div>
    <a href="#" id="load-content">コンテンツの読み込み</a>

    <!-- javascript files -->
    <script type="text/javascript">
        var BASE = "<?php echo URL::base(); ?>";
    </script>
    <?php echo HTML::script('js/jquery.js'); ?>
    <?php echo HTML::script('js/script.js'); ?>
</body>
</html>
```

ご覧の通り、タグを閉じる直前に Javascript ファイルを参照しています。これにより、HTTP リクエストでこれらのファイルを読み込むことで、ページロードをブロックしないようにしています。これは気をつけておくべきグッドプラクティスです。では `public/js/script.js` ファイルを作成し、追加しましょう。

それと、後で URL のリクエストを行うときに必要となる、アプリケーションのベース URL を Javascript に引き渡すために、`BASE` 変数を新しく作成しています。

```
// public/js/script.js
jQuery(document).ready(function($) {
        // Javascritp はページが完全にロード
        // された時点で実行される
});
```

ここで現在のドキュメントを処理するため jQuery() オブジェクトを使用しています。コードを記述するクロージャーと一緒に `ready()` イベントを付け加えています。`document` オブジェクトの準備ができる (`ready()`) まで待つことで、全ての `DOM` オブジェクトがロードを完了し、jQuery ライブラリーもロード済みであることを確実にします。

同様な例としては…

```
// public/js/script.js
$(document).ready(function() {
        // Javascritp はページが完全にロード
        // された時点で実行される
});
```

これでも良いのですが、もし別の Javascript ライブラリーが $ オブジェクトを使用していると後で問題が引き起こされます。私のメソッドは jQuery クラスを使用し、内部のクロージャーに $ を渡すことで、jQuery オブジェクトを再定義しています。これで衝突を避けることができます。

コンテンツ読み込みリンクにクリックイベントを作成開始しましょう。jQuery では多くの方法があります。私は.click() メソッドを使用します。こんなふうに…

```
// public/js/script.js
$(document).ready(function() {

        $('#load-content').click(function(e) {
                e.preventDefault();
        })

});
```

これで click イベントをコールバックのクロージャーと一緒に定義できました。e という event 引数を内部クロージャーに指定することで、e.preventDefault(); メソッドをクリックに対するデフォルトアクションが実行されないように、指定できるようになります。この場合、リンクはハイパーリンクとしては動作しなくなります。これで、新しいコンテンツを入手 (GET) するため他の HTTP リクエストを作成し、#content 領域にロードする必要が起きました。これを行うために jQuery の.get() メソッドを使いましょう。

```
// public/js/script.js
$(document).ready(function() {
        $('#load-content').click(function(e) {
                // リンクでデフォルトのアクションが
                // 起きることを防ぐ
                e.preventDefault();

                // 新しいコンテンツを GET するように試みる
                $.get(BASE+'content', function(data) {
                        $('#content').html(data);
                });
        })
});
```

BASE 属性を事前に設定しておいたのを覚えていますか？これをリクエスト URL を組み立てるために使用しています。そして返される data をキャッチするコールバックメソッドを作成します。GET リクエストのレスポンスを #content 要素に.html() メソッドを使用して挿入します。

これはとてもハードワークだと思いませんか？でも、このハードワークを実行して、見てみましょう。http://localhost でアプリケーションを読み込み、コンテンツ読み込みリンクをクリックしてください。ありがたいことに、うまく動きます！

ここまでご覧になられたとおり、Laravel の AJAX は他のフレームワークや PHP 自身をそのまま利用する時と同じです。ただ最適な方法で、AJAX ルートに対するビューを確実に整形しておくよう心がけましょう。

22.3 データーのポスト

時にはリクエストに応じ、追加で送信する必要もあるでしょう。jQuery を使いどうやるのかデモンストレートするために、新しい POST HTTP リクエストを作成してみましょう。

```
// application/routes.php
Route::post('content', function() {
    // ポストするデーターを取り扱う…
});
```

Route::post() メソッドを get() の代わりに使用し、コンテンツルートを POST リクエストに対応するようにしています。では、Javascript も行いましょう。

```
// 追加データーとともに
// POST リクエストを試みる
$.post(BASE+'content', {
        name: "Dayle",
        age: 27
}, function(data) {
        $('#content').html(data);
});
```

コンテンツルートにデーターをポストするため、POST メソッドを使用しています。Laravel ではフォームで Input クラスを利用して取り扱うのと同じように、データーを受け取ることができます。

```
// application/routes.php
Route::post('content', function() {
    echo Input::get('name');  // Dayle
    echo Input::get('age');   // 27
});
```

とてもシンプルです！

22.4 ＪＳＯＮレスポンス

Javascritp を使用するのでしたら、Javascript と親和性がより高い、文字列ベースの配列表現である JSON 形式をリターンできると便利です。

> JSON レスポンスを返すために、最初に PHP 配列にエンコードし、それから適したヘッダーでクライアントに JSON コンテンツタイプを知らせなくてはなりません。

上記の引用は正しいのですが、Laravel 3.2 に加えられた `Respons::json()` メソッドが同じゴールを達成してくれます。

新しいルートを作成しましょう。

```
// application/routes.php
Route::get('content', function() {

        // JSON にするデーター
    $data = array(
            'name'    => 'Dummy',
            'size'    => 'XL',
            'color'   => 'Blue'
    );

    return Response::json($data);

});
```

JSON エンコードを行なってくれる `json()` メソッドに配列を渡し、最終的にレスポンスオブジェクトをリターンしています。

22.5 ＡＪＡＸリクエストを見分ける

場合により、リクエストが AJAX を使用しているかどうかをルートの中で判断できると便利です。幸運なことに Laravel は AJAX リクエストを見分けるメソッドが用意されています。見てみましょう。

```
// application/routes.php
Route::get('content', function() {

        // リクエストが AJAX からかどうかを
        // チェックする
        if (Request::ajax())
        {
                // AJAX コンテンツを提供する
                return View::make('only_content'); // コンテンツのみ
        }

        // フルコンテンツを提供する
    return View::make('content_with_layout'); // レイアウトと共に

});
```

`Request::ajax()` が `true` を返したら、そのリクエストは AJAX リクエストで、`false` を返したら違います。

Code Happy を購入いただいた、以下の方々に感謝いたします。ありがとうございました。

- William Manley
- Nate Nolting
- Jorijn Schrijvershof
- Caleb Griffin
- Florian Uhlrich
- Stefano Giordano
- Simon Edwards

また、この本について親切なメールを送っていただいた全ての方にも感謝いたします。サポートありがとうございました。

23 アプリケーションのデバッグ

Laravel フレームワークを使い、バグ無しでアプリケーションを書くことは良い経験です。しかしながら私達全員、締め切り間近になるとこれが突然現れることをよく知っていますし、そして怒った上司があなたの後ろに立つことになるのです。

PHP 自身は多くの異なったデバッグ方法を持っています。シンプルな `var_dump()` や `print_r()`、有名な `die()` に上級のデバック機能である PHP xdebug 拡張などです。このような基本的な手法の詳細を取り上げるつもりはありません。なぜなら、これは PHP 言語の基本ではなく、Laravel についての本だからです。代わりに小さなバグを追跡する Laravel の機能について見て行きましょう。

23.1 エラーハンドリング

Laravel はカスタムエラーハンドラーを含んでおり、PHP エラーのデフォルト処理をオーバーライトしています。スタックトレースと、より詳細な情報を提供しています。エラーハンドラーの出力を見てみましょう。

```
Unhandled Exception

Message:

View [page.panda] doesn't exist.
Location:

/Users/daylerees/www/panda/laravel/view.php on line 151
Stack Trace:

#0 /Users/daylerees/www/panda/laravel/view.php(93): Laravel\View->path('pag\
e.panda')
#1 /Users/daylerees/www/panda/laravel/view.php(199): Laravel\View->__constr\
uct(' page.panda ', Array)
#2 /Users/daylerees/www/panda/application/routes.php(35): Laravel\View::mak\
e(' page.panda ')
#3 [internal function]: {closure}()
#4 /Users/daylerees/www/panda/laravel/routing/route.php(163): call_user_fun\
c_array(Object(Closure), Array)
#5 /Users/daylerees/www/panda/laravel/routing/route.php(124): Laravel\Routi\
ng\Route->response()
#6 /Users/daylerees/www/panda/laravel/laravel.php(125): Laravel\Routing\Rou\
te->call()
#7 /Users/daylerees/www/panda/public/index.php(34): require('/Users/daylere\
e...')
#8 {main}
```

これは実行時に存在しないビューをリクエストした時に表示されるエラーです。ご覧の通り、Laravelにより有益なエラーメッセージが表示され、同時にエラーが起きたファイルと行番号も示されます。また、追加でスタックトレースも表示され、ルーティングから、View レイヤーを通り、最初のエラーに行き着くまでのメソッドの呼び出しを知らせています。

ほとんどの場合、スタックトレースは必要ありませんが、例えば経験を積んだ開発者が、複雑なライブラリーを使用し、エラーが起きた場合などに役立つでしょう。

23.2 エラー設定

いつもこのようにエラーを表示したくはありませんよね。特に本番環境でスタックトレースを表示するのは、非常なセキュリティーリスクです。

幸福なことに、そしていつも通り、Laravel ではエラーの表示設定を簡単に変更できます。エラー表示に関する設定ファイルは application/config/error.php です。配列で構成されている設定オプションを一緒に確認してみましょう。

```
'ignore' => array(),
```

ignore 配列にはエラーハンドラーにより無視させるエラーのリストで構成します。指定したエラーはそれ以上表示されることはありませんが、全部ログには残ります。ignore 配列を使用するときは、これを思い出してください。

この配列にエラータイプを設定するためには、PHP エラータイプ定数を追加するか、整数値を指定します。

```
'ignore' => array(E_ERROR);
```

完全な PHP エラータイプ定数のリストは PHP API で見つけられます。しかしながら、いくつか便利なものを紹介しておきましょう。

E_ERROR これは全ての fatal ランタイムエラーに一致します。

E_WARNING これは全ての warning、もしくは fatal タイプでないものに一致します。

E_PARSE これは全てのパース時エラー、もしくはシンタックスエラーに一致します。

E_NOTICE これは全てのランタイム notice に一致します。

E_ALL これは E_Strict エラーを除く、上記のエラー全部に一致します。

```
'detail' => true,
```

detail 設定オプションは詳細エラーレポートのオン・オフを切り替えるのに使用します。true にすると上記で紹介したように、スタックトレースとともに、フルエラーレポートが表示されます。false にすると代わりにデフォルト 500 エラーページが表示されます。

```
'log' => false,
```

log 設定オプションが true の場合、エラーが発生するたび、logger 設定オプションで指定したクロージャーに exception オブジェクトが渡され、実行されます。

```
'logger' => function($exception)
{
        Log::exception($exception);
},
```

デフォルトでは、logger 設定オプションのクロージャーの中で、storage/logs ディレクトリーのログファイルに書き込まれます。しかしながら、クロージャーで提供されているので、非常に柔軟に取り扱うことができ、あなたの考えている通りの方法で、デフォルトを書き換えることも可能です。多分、データベースにログを保存しておきたのではないですか？では、どうぞ！あなたに任せました。

23.3 ログ

エラーを表示し、それをファイルに保存するのは便利ですが、自分のカスタム情報をログしたい時にはどうしましょう？Laravel の Log クラスはアプリケーションからの有益な情報をログする 2 つの異なったメソッドで構成されています。では、確認して行きましょう。

Log::write() メソッドはログファイルに書き出す、ログタイプとメッセージ文字列を引数に取ります。例えば…

```
Log::write('myapp', ' ここに有益な情報を書く');
```

ログファイルには次のように書き込まれます。

```
2012-05-29 19:10:17 MYAPP - ここに有益な情報を書く
```

このエントリーは storage/logs ディレクトリー下に今日の日付を名前とするファイルに、もちろん書き込まれます。例えば 2011-02-05.log です。これはログがローテーションし、ログファイルが巨大になるのを防いでくれます！

また、ログのタイプをマジックメソッで指定もできます。例えば…

```
Log::shoe(' ログの書き出し');
```

これは次のコードと同じです：

```
Log::write('shoe', ' ログの書き出し');
```

便利ですね！

この章で習ったことを全部活用して、アプリケーションの調子が悪い時には、分析してください！

この本を購入して下さった、以下の素晴らしい方々へ、感謝を申し上げます。

- Marc Carson
- David Wosnitza AKA _druuuuuuuu
- Mark van der Waarde from Alsjeblaft!
- Joseph Wynn
- Alexander Karisson
- Victor Petrov
- Josh Kennedy
- Alexandru Bucur

24 これからも

この本には Web アプリケーションをうまく書くために必要なものは全部揃っていると思っています。しかしながら、Laravel の将来の改訂と共にアップデートしていくつもりです。もし、私が詳細に説明できる部分を見つけたら、そうするでしょう！

この本は成長を続ける本です。ですから一度購入したら、将来の全てのアップデートは無料で Leanpub.com からダウンロードしていただけます。

購入いただき、ありがとうございました。

Dayle