
```php
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Http\Request;

class UserController extends Controller
{

    public function __construct()
    {
        $this->middleware('auth.first');
    }

    public function getIndex()
    {
        $users = User::paginate(5);

        return view('user.index')->with('users', $users);
    }
}
```

Web職人好みの新世代PHPフレームワーク

Laravel リファレンス

Ver.5.1 LTS 対応

新原 雅司、竹澤 有貴、川瀬 裕久、
大村 創太郎、松尾 大［共著］

Webアプリ開発のしやすさで注目度No.1!

Laravelの基本、データベース、フレームワークの活用・拡張、
テスト、実践的なアプリケーション構築など、包括的に解説

インプレス

■ソースコードについて

本書で使用しているソースコードは、以下の URL よりダウンロードできます。

http://www.impressjapan.jp/books/1114101107

はじめに

PHP には数多くのフレームワークが存在しています。かつては Java や Ruby など各種プログラミング言語で人気を博したフレームワークの流れを汲むものが主流でしたが、昨今の PHP の進化や Composer によるパッケージ管理エコシステムの確立で、フレームワーク自らが持つ機能をコンポーネントとして独立させ、コンポーネントの集合としてフレームワークを形成する構成が注目を集めています。また、個別に配布されるコンポーネントはフレームワークと分離して利用でき、必要なコンポーネントを組み合わせて、新たなフレームワークを構築することも可能です。

本書で取り扱う Laravel はまさにこれを体現しており、既存の Symfony を中心に各種のオープンソースコンポーネントやライブラリで構成されています。また、興味深いのは Laravel 自身もコンポーネントとして分離可能である点です。まさにコンポーネント指向である現在の PHP フレームワークのトレンドを牽引しているといえます。

Web アプリケーションの要件は多種多彩ですが、どのアプリケーションでも必要とされる定型処理はフレームワークごとに独自に実装するのではなく、既に実装されている良質なコードを再利用しており、合理的な実装方法がとられています。

Laravel では良質で実績のある既存コンポーネントを簡潔なコードで利用できます。もちろん、内部では複雑でパワフルな実装も行われていますが、一般的な Web アプリケーション開発はシンプルなコードを記述するだけで簡単に実現できます。さらに、より大規模で複雑なアプリケーションを構築するケースでは、コンポーネントの機能をフルに活用することも可能であり、フレームワークの初学者からエキスパートまで幅広い層に受け入れられています。

共著者の川瀬は公式ドキュメントの和訳版を公開したり、その他の執筆陣もそれぞれ個別にブログに展開したり技術イベントに講演者として登壇するなど、様々な情報提供を行っています。また、筆者自身も業務で Laravel を利用した Web アプリケーションをリリースし、追加開発はもちろん継続的な運用を担当するなど、開発現場で実際に Laravel を活用しています。

本書は、執筆陣が様々な普及活動や開発現場で蓄積した知見に基づき、これから Laravel を学ぶ初学者はもちろん、既に活用している開発者にも参考になる構成内容を心掛けました。順を追ってすべてを読破する必要はなく、開発に求められる部分を読み進めることで、Laravel の活用が十分に理解できるはずです。本書が開発者の皆様のお役に立てるならば幸いです。

2015 年冬　執筆陣を代表して
新原 雅司

はじめに …… Ⅲ

Chapter 01

Laravelの概要

1-1 Laravelとは …… 002
1-1-1 Laravelとは …… 002
1-1-2 Laravelの特徴 …… 002
1-1-3 フレームワークによる開発 …… 006
1-1-4 開発情報 …… 007

1-2 環境設定 …… 008
1-2-1 Windows/OS Xにおける開発環境の構築 …… 008
1-2-2 Linuxにおける開発環境の構築 …… 009
1-2-3 Laravel Homestead …… 010
1-2-4 Laravel Homesteadの設定 …… 013
1-2-5 Laravel Homesteadの実行 …… 014
1-2-6 Laravel Homestead環境への接続 …… 016

1-3 Composer …… 017
1-3-1 Composerとは …… 017
1-3-2 ローダーの仕組み …… 018
1-3-3 Composerコマンド …… 021

Chapter 02

Laravelの基本

2-1 はじめてのLaravel …… 026
2-1-1 Laravelのインストール …… 026
2-1-2 ディレクトリ構造 …… 028
2-1-3 アプリケーションの設定 …… 029
2-1-4 Artisanコマンド …… 032

2-2 はじめてのアプリケーション …… 034
2-2-1 アプリケーション構造 …… 034
2-2-2 アプリケーションの準備 …… 038
2-2-3 はじめてのルート定義 …… 040
2-2-4 はじめてのビュー …… 042
2-2-5 はじめてのORM …… 046

2-2-6 はじめてのコントローラ ····· 048
2-2-7 はじめてのフォーム ····· 050

2-3 基本コンポーネント ····· 053

2-3-1 環境設定 ····· 053
2-3-2 リクエスト ····· 056
2-3-3 レスポンス ····· 057
2-3-4 ルーティング ····· 058
2-3-5 ミドルウェア ····· 064
2-3-6 コントローラ ····· 068
2-3-7 バリデーション ····· 069

Chapter 03 データベース

3-1 データベースへの接続設定 ····· 080

3-1-1 サポートしているデータベース ····· 081
3-1-2 接続設定 ····· 082

3-2 マイグレーション ····· 086

3-2-1 マイグレーションの流れ ····· 086
3-2-2 マイグレーションファイルの作成 ····· 087
3-2-3 マイグレーションの実行・ロールバック ····· 089
3-2-4 スキーマビルダ ····· 090
3-2-5 シーダー ····· 095

3-3 DB ファサード ····· 100

3-3-1 DB ファサードを利用したクエリ実行 ····· 100
3-3-2 接続するデータベースの指定 ····· 102
3-3-3 トランザクション ····· 102
3-3-4 その他の便利なメソッド ····· 103

3-4 クエリビルダ ····· 105

3-4-1 テーブルからのデータ取得（基本編） ····· 106
3-4-2 検索条件の指定 ····· 109
3-4-3 JOIN ····· 112
3-4-4 ソート・グルーピング・Limit と Offset ····· 113
3-4-5 UNION クエリ ····· 114
3-4-6 サブクエリ ····· 114
3-4-7 データの挿入 -- insert ····· 115
3-4-8 データの更新 ····· 116
3-4-9 データの削除 ····· 116
3-4-10 悲観的ロック ····· 117
3-4-11 SQL の直接記述 -- raw ····· 117
3-4-12 主なメソッド一覧 ····· 118

3-5 Eloquent ORM ····· 121
3-5-1 モデルの作成 ····· 121
3-5-2 Eloquent モデルの規約およびその変更 ····· 122
3-5-3 データの取得 ····· 124
3-5-4 データの挿入・更新 ····· 126
3-5-5 Mass Assignment ····· 128
3-5-6 データの削除 ····· 130
3-5-7 アクセサとミューテータ ····· 133
3-5-8 日付の扱い ····· 134
3-5-9 シリアライゼーション ····· 134

3-6 リレーション ····· 139
3-6-1 One To One ····· 140
3-6-2 One To Many ····· 142
3-6-3 Many to Many ····· 144
3-6-4 Has Many Through ····· 146
3-6-5 リレーション先のデータ取得 ····· 147
3-6-6 N+1 問題と Eager Loading ····· 148
3-6-7 リレーション先へのデータ更新 ····· 151

Chapter 04 フレームワークの機能

4-1 認証 ····· 156
4-1-1 仕様 ····· 157
4-1-2 認証機能の設定 ····· 159
4-1-3 Auth ファサードによる認証処理 ····· 162

4-2 キャッシュ ····· 168
4-2-1 キャッシュストア ····· 168
4-2-2 キャッシュの利用 ····· 175

4-3 エラーハンドリング ····· 181
4-3-1 エラー表示 ····· 181
4-3-2 エラーのハンドリング ····· 182
4-3-3 HTTP エラーの送出 ····· 185

4-4 ロギング ····· 186
4-4-1 ロギングの設定 ····· 186
4-4-2 ログの出力 ····· 188
4-4-3 ロギングのカスタマイズ ····· 189

4-5 イベント ····· 190
4-5-1 イベントのリスナー ····· 190
4-5-2 イベントの発行 ····· 194
4-5-3 イベント操作メソッド ····· 195
4-5-4 EventServiceProvider によるリスナー管理 ····· 196
4-5-5 イベントクラス ····· 198
4-5-6 イベントのサブスクライブ ····· 202

4-6 ローカリゼーション ····· 205
4-6-1 言語ファイル ····· 205
4-6-2 ロケール設定 ····· 208
4-6-3 メッセージの取得 ····· 209

4-7 メール ····· 211
4-7-1 設定 ····· 211
4-7-2 メールの送信 ····· 213
4-7-3 クラウドメールサービスとの連携 ····· 222

4-8 ページネーション ····· 224
4-8-1 ページネーションデータの取得 ····· 224
4-8-2 Blade によるページネーションの表示 ····· 226
4-8-3 JSON によるページネーション ····· 228
4-8-4 カスタムページネータ ····· 229

4-9 セッション ····· 232
4-9-1 設定 ····· 232
4-9-2 セッションの操作 ····· 236
4-9-3 次のリクエストだけ有効なセッション ····· 241

Chapter 05 フレームワークの拡張

5-1 サービスコンテナ ····· 244
5-1-1 サービスコンテナとは ····· 244
5-1-2 サービスコンテナへのバインド ····· 247
5-1-3 サービスコンテナによる解決 ····· 253
5-1-4 サービスコンテナによる DI ····· 255
5-1-5 その他のメソッド ····· 263

5-2 サービスプロバイダ ····· 264
5-2-1 サービスプロバイダの実装 ····· 264
5-2-2 独自サービスプロバイダの実装（Twilio との連携） ····· 269

5-3 コントラクト ····· 274
5-3-1 コントラクトのメリット ····· 274
5-3-2 コントラクトを利用した実装 ····· 274
5-3-3 コントラクトを実装した独自クラス ····· 278

5-4 ファサード ····· 282
5-4-1 ファサードの仕組み ····· 282
5-4-2 独自ファサードの実装 ····· 286

Chapter 06 テスト

6-1 ユニットテストの基本 ····· 290
6-1-1 PHPUnit によるユニットテスト ····· 290
6-1-2 設定ファイル ····· 291
6-1-3 テストの記述 ····· 291
6-1-4 例外のテスト ····· 295
6-1-5 データプロバイダ ····· 296
6-1-6 テストの依存 ····· 297
6-1-7 protected、private 宣言されたメソッド ····· 298
6-1-8 コードカバレッジ機能 ····· 299
6-1-9 終了時の処理 ····· 301

6-2 モックを利用したテスト ····· 302
6-2-1 Mockery ····· 302
6-2-2 Mockery チュートリアル ····· 302
6-2-3 Mockery リファレンス ····· 306

6-3 モデルファクトリ「Faker」 ····· 313
6-3-1 Faker の利用方法 ····· 313
6-3-2 データベースへの挿入 ····· 315
6-3-3 言語の設定 ····· 316
6-3-4 各種ダミーデータ ····· 317

6-4 各種アプリケーションのユニットテスト ····· 319
6-4-1 ミドルウェアのユニットテスト ····· 319
6-4-2 データベースに依存したクラスのテスト ····· 321
6-4-3 データベースを利用するテスト ····· 322

6-5 ファンクショナルテスト ····· 330
6-5-1 テストクラス ····· 330
6-5-2 テストヘルパーメソッド ····· 330

6-5-3 クロールによる機能テストメソッド …… 333
6-5-4 動作を変更するヘルパーメソッド …… 336
6-5-5 スタブを利用したファンクショナルテスト …… 336
6-5-6 認証を含むファンクショナルテスト …… 338

Chapter 07 実践的なアプリケーション構築

7-1 セキュリティ対策 …… 342
7-1-1 クロスサイトスクリプティング対策 …… 342
7-1-2 SQL インジェクション対策 …… 346
7-1-3 CSRF 対策 …… 349

7-2 コマンドラインアプリケーション開発 …… 352
7-2-1 コマンドラインアプリケーションの作成 …… 352
7-2-2 作成コマンドの登録 …… 353
7-2-3 コマンドオプション・引数の利用 …… 354
7-2-4 対話式コマンドの実装 …… 357
7-2-5 メッセージなどの表示 …… 358
7-2-6 コマンドラインアプリケーションのテスト …… 359

7-3 代表的な拡張パッケージ …… 362
7-3-1 Laravel 5 IDE Helper Generator …… 362
7-3-2 Laravel Debugbar …… 363
7-3-3 Laravel Socialite …… 364
7-3-4 Forms & HTML …… 366
7-3-5 日本語化パッケージ comja5 …… 368
7-3-6 Intervention Image …… 368
7-3-7 JSON Web Token Authentication …… 368
7-3-8 Sentinel …… 369
7-3-9 パッケージの入手先 …… 369
7-3-10 開発時にのみ利用するパッケージ …… 369

Chapter 08 Laravel の実践

8-1 サンプルアプリケーションの概要と設計 …… 372
8-1-1 動作環境 …… 372
8-1-2 実装する基本的な機能 …… 372
8-1-3 テーブル設計 …… 373

8-1-4　実行方法 …… 374
8-1-5　アプリケーション設計 …… 375

8-2　データベースの準備 …… 377

8-2-1　マイグレーションの作成 …… 377
8-2-2　データベースの作成 …… 379

8-3　ユーザー登録の実装 …… 380

8-3-1　ルート …… 380
8-3-2　ユーザー登録フォーム …… 381
8-3-3　ユーザー登録のリファクタリング …… 384
8-3-4　キャプチャ認証によるカスタムバリデート …… 390

8-4　ログインの実装 …… 397

8-4-1　ログイン画面 …… 397
8-4-2　フォームリクエストでのバリデーション …… 397
8-4-3　ログイン実装 …… 398

8-5　認証機能のカスタマイズ …… 400

8-5-1　独自認証ドライバの実装 …… 400
8-5-2　認証コンポーネント …… 402
8-5-3　認証ドライバの追加 …… 403
8-5-4　ドライバ追加のメリット …… 404

8-6　ブログ記事管理機能 …… 405

8-6-1　アクセス制御 …… 405
8-6-2　新規ブログ登録フォームと登録処理 …… 406
8-6-3　記事一覧画面 …… 410
8-6-4　編集フォームと更新処理 …… 414
8-6-5　ブログ記事編集の制御 …… 415
8-6-6　ログインユーザーの表示 …… 420

8-7　ブログ表示・コメント投稿機能 …… 423

8-7-1　ブログ表示 …… 423
8-7-2　コメントの取得と書き込み …… 425

索引 …… 432

著者プロフィール …… 437

Chapter

01

Laravel の概要

常に新しいフレームワークが発表される PHP 界に現れた新星が Laravel です。利便性を第一に考えつつも新しい開発ツールを積極的に統合しています。さらには、コミュニティやカンファレンスを通し開発ノウハウが発信されています。本章では Laravel のバックボーンに触れることで、その方向性を説明し、PHP 開発環境の整備に関しても解説します。

Section
01-01

Laravelとは

何かを学ぶ際には、その対象を知る必要があります。本書で解説するLaravelを学ぶにあたり、そのバックボーンや特徴を把握することで、技術的な知見の理解もスムーズに得られます。本節ではLaravelの概要を説明します。

1-1-1 Laravelとは

「Laravel」（ララベル）は、Taylor Otwell氏（アメリカ・アーカンソー州在住）が開発するPHPフレームワークです。.NETによる開発経験をバックボーンに持つ同氏は、PHPの特徴を活かしてWebアプリを快適に開発できるフレームワークを目指しています。その使いやすさはもちろん、活発なコミュニティ活動のおかげでGoogle Trendsにおける人気度の動向は右肩上がりを示すなど、世界中で人気が急騰しており、日本国内でもユーザーが急増しています。また、オープンソースとしてMITライセンスで配布されているので、無料で自由に利用できます。

Laravelはフルスタック（多機能）なPHPフレームワークで、ルーティング、コントローラ、ビュー、ORMなど基本的な機能を備え、さらに近代的なWebアプリで活用されるジョブキューやWebストレージなども積極的に統合しています。

「幸せな開発者が最高のコードを書く」、これがLaravelのバックボーンとなっている基本的な哲学です。定型的なコードの記述を減らすため、様々な工夫が取り入れられています。ありふれた「雑務的」なプログラミング作業を減らし、「よい動作ロジック」、「シンプルで分かりやすい実装方法」など、より創造的な部分に開発者の能力を投入できます。

1-1-2 Laravelの特徴

Laravelにはたくさんの魅力がありますが、Laravelを使い始めた開発者が賞賛するポイントをいくつか紹介します。

・多種多様なバリデーションルールと容易な拡張性
・簡単に実現できるページネーション（ページ付け）
・柔軟なサービス (DI) コンテナ
・使いやすく使用準備の手間が掛からない ORM
・実行しやすいテスト

上記の特徴は各章で説明していきますが、本項では Laravel フレームワークが持つ魅力のなかでも特に特徴的ともいえる、高い可読性・記述性と開発速度の向上を中心に、バージョン推移とLTS（Long Term Support）、互換性の高い軽量フレームワーク Lumen に関して説明します。

▶高い可読性・記述性

Laravel のファサード（Facade）は、静的メソッド記法によるコアクラスへのアクセスを提供しています。これにより静的メソッド記法の可読・記述性の高さと、交換可能なクラスインスタンスという長所を両立させています。ファサードによるアクセス時に、クラスがインスタンス化されていなければ生成され、呼び出したメソッドが実行されます。

ただし、ファサードを利用する記法が強制されることはありません。別の方法でもコアクラスへのアクセスが可能です。開発手法や嗜好に従い、ベストな形式を選択できます（リスト 1.1）。

リスト 1.1：様々なコアクラスへのアクセス方法

```
// ファサード記法による File クラスへのアクセス
$content = \File::get('sample.txt');

// ヘルパー関数を使用してのアクセス
$content = app('files')->get('sample.txt');
$content = app()->files->get('sample.txt');

// コンテナの自動依存インスタンス注入による取得

// File ファサードの実体クラス
use Illuminate\Filesystem\Filesystem;

class Sample {
    // この Sample クラスがサービスコンテナにより
    // インスタンス化される場合に Filesystem は
    // 自動的にインスタンス化され、$file に渡される。
    public function __construct(Filesystem $file)
```

```
    {
        $this->file = $file;
    }

    public function getFile($fileName)
    {
        $content = $this->file->get('sample.txt');
    }
}
```

また、この他にも簡単で分かりやすいメソッド名の採用など、細かい心遣いが積み重ねられ、可読性と記述性の高いコードを記述できます。

▶開発速度の向上

Laravel を採用する開発者は、「開発速度の向上」を魅力の 1 つにあげます。前述の高い可読性や記述性に加え、定型コーディングの減少や開発チームのスタイルに合わせられる柔軟性など、その使いやすさから開発速度の向上を見込めると判断されています。フレームワークは開発の基礎システムですが、技術者には道具ともいえます。使い勝手の良さは重要な要素です。

さらに Laravel を使った開発により、高揚を感じる開発者も少なくありません。多くの開発者にとって「簡単すぎず、難しすぎない」、「便利だけれど、押し付けがましくない」と感じさせる「程度の良さ」を持っている Laravel であれば、快適に開発できるからです。幸せな開発者は最高のコードを書くだけでなく、効率的に行います。余計な事柄や気分に気が削がれずに、担当する仕事に集中できるからです。

開発速度を上げ、効率を向上できるのは、誰にとってもメリットがあります。日本の Laravel 紹介ページである laravel.jp で紹介されている、Jeff Madsen 氏（http://codebyjeff.com/）のコメントを引用しましょう。

「Laravel はオフィスに夜遅くまで残りたくない人のためのフレームワークです。」

▶バージョン推移と LTS

Laravel はメジャーアップデートごとに大きく発展し続けています。簡単に過去バージョンのリリース経緯を説明します。

・バージョン1および2：学習コストの低さ、軽量さが評価されていました。
・バージョン3：Webアプリに必要な機能が強化され、フルスタックになりました。Laravelが世界的にブレークしたバージョンです。
・バージョン4：大規模開発やテスト性能向上のため、コンテナ機能が強化されたバージョンです。PHPパッケージ管理ツールの「Composer」（P.017）が導入されました。
・バージョン5：Webアプリケーションのレイヤに基づくディレクトリ構造へ刷新されました。タスク自動化の「Elixir」やOAuth認証の「Socialite」パッケージが用意されました。

大きな進化を遂げるメジャーリリースは数年ごとに行われる予定ですが、主に機能追加が行われるマイナーリリースは半年単位です。このサイクルが早いため、長期にわたり開発を継続する必要のあるプロジェクトでは、Laravel採用を躊躇させる面もありました。

しかし、2015年6月9日にリリースされたバージョン5.1はLTS（長期間サポート版）です。LTS版は2年ごとにリリースされる予定です。バグフィックスは2年間、脆弱性の対処は3年間にわたり行われます。これにより、長期的なプロジェクトにも安心して採用できます。本書では、初めてリリースされたLTSであるバージョン5.1を中心に解説します。

▶軽量フレームワークLumen

Laravelはバージョン4以降Composerが導入されたこともあって、積極的に既存のパッケージを採用しています。十分にテストされた多様なコンポーネントを取り入れることで機能を強化していますが、オーバーヘッドの増加により、実行速度が低下する傾向にあります。

そこでシンプルな機能のアプリケーションを素早く動作させるため、新しいフレームワークが開発されています。初期化処理や設定方法を見直し、スピードアップを図った軽量フレームワーク「Lumen」（http://lumen.laravel.com/）です。Laravelとの互換性も高いフレームワークです。

高速に動作する兄弟フレームワーク「Lumen」の登場により、一般的なWebアプリ側ではLaravelを、API側ではLumenを利用するなど、用途に合わせて使い分けることが可能です。

なお、LumenではLaravelと共通のコンポーネントが使用されているため、本書の内容はLumen使用時にも参考になります。

1-1-3 フレームワークによる開発

「フレームワーク」とは骨組みや何かの基礎となる構造を指し示す言葉です。Webアプリケーションのためのフレームワークは、アプリケーション構築に必要な機能が組み合わされ、開発準備が整っている基本的なソフトウェアを指します。また、同様にアプリケーション開発に使用されるプログラムに「ライブラリ」があります。ライブラリは、単一機能を提供するソフトウェアもしくはその集合を示し、システムを構築するための部品です。

フルスタックな現代のPHPフレームワークは通常複数のライブラリで構成されています。両者はともにWeb開発のベストプラクティスやノウハウが詰め込まれたものです。開発者は全知の存在ではありません。フレームワークやライブラリを適切に使用することで、知識や経験を補うことが可能です。ライブラリは、システムを構築する部品として十分にテストされ、品質も管理されています。その機能を自ら作成するよりも簡単に高品質の部品が手に入るわけです。

開発する際に利用するベースシステムによる特徴を見てみましょう（表1.1）。

表1.1：開発ベースによる比較

ベースシステム	必要な知識量	開発自由度	実行効率
CMS系[※1]	低〜中	限定的	低
フレームワーク	やや低〜中	十分	中〜高
ライブラリ	中〜やや高	十分	中〜高
PHPのみ	高	最高	最高

ベースシステムを使わずに、PHPのみでシステムを開発する場合、開発効率や自由度は高いですが、開発者には幅広い知識が求められます。CMS系ソフトウェアをベースとして開発すれば、必要な知識量はさほど多くない代わりに、拡張性が低くなります。機能追加や変更が標準で拡張できる範囲を超えると、要求される知識と作業量が各段に増えます。

フレームワークやライブラリによる開発を始めるには、ある程度の学習が必要とされますが、比較的実行効率が高いシステムを開発できます。新しいサービスを構築するのは、既存サービスとの差別化や優位性が求められることが理由の1つですから、そうした要求に応えられるだけの開発自由度を活用できます。

数多く存在するフレームワークの中でも、Laravelは開発速度や利便性を重視しており、強い規約もないため段階を踏みながら学習することで簡単に習得できます。本書はフレームワークをある程度理解していることを前提としていますが、初学者でもポイントをおさえ効率的に理解できるよう、各節の冒頭で基本概念を解説します。

1-1-4 開発情報

最初に公式サイト（http://laravel.com）のドキュメントに目を通しましょう。左側のナビゲーションに「The Basics」（基礎）と「Architecture Foundations」（構造）として分類されているドキュメントを最初に読むことをお勧めします。以降は必要に応じて目を通しましょう。

・公式サイト：http://laravel.com
　- ドキュメント：http://laravel.com/docs
　- API：http://laravel.com/api

公式サイトはTaylor Otwell氏が管理する英語サイトのみです。英語が苦手な場合は、非公式の日本語ドキュメントもあるので、こちらを参照しましょう。

・日本語翻訳ドキュメント：
　http://readouble.com/laravel

各種の質問はコミュニティで尋ねることをお勧めします（日本語）。

・Facebook Laravel jp：
　https://www.facebook.com/groups/laravel.jp
・Google+ Laravel Japan：
　https://plus.google.com/u/0/communities/118006056115330646882

上記の他にも、「Stackoverflow」（http://ja.stackoverflow.com/）や「Qiita」（https://qiita.com/）などのQ&Aサイトで質問しても回答を得られことが多いようです。

なお、上記のオープンなサイトで質問することが難しい場合、日本語でLaravelに関するチャットが可能な会員制チャットルームも用意されています。larachat.jp@gmail.comにメールを送れば、招待状が返信されます。

・Slack larachat-jp：https://larachat-jp.slack.com

（※1）コンテンツマネージメントシステム（Content Management System）

Section
01-02

環境設定

Laravelを使い開発するには、PHPやWebサーバ、MySQLなどのデータベースが必要です。本項では、XAMPPなどを利用した簡単な開発環境の構築方法や、Vagrantを利用するHomesteadの利用方法を紹介します。なお、本書ではLaravel 5.1を対象にしているため、PHPはバージョン5.5.9以上を利用する必要があります。

1-2-1 Windows/OS Xにおける開発環境の構築

WindowsやOS XでのPHP開発環境の構築には複数の方法がありますが、本項では「XAMPP」による環境構築を紹介します。

▶ XAMPPのインストール

公式サイト「XAMPP Installer and Downloads」（https://www.apachefriends.org/jp/index.html）にアクセスして、インストーラをダウンロードします。インストーラを起動して指示にしたがいインストールします。

インストール完了後は、XAMPPのコントロールパネルでApacheやMySQLを起動します。Apacheの起動後は、「http://localhost」または「http://127.0.0.1」にアクセスできることを確認しましょう。

▶環境変数の設定

続いてコマンドの実行パスにPHPのファイルパスを追加します。Windows環境では、「コントロールパネル」を開き、「システムとセキュリティ」などからたどって表示できる「システムのプロパティ」→「詳細設定」タブの「環境変数」をクリックします。システム環境変数の変数名「Path」を選択し、「編集」ボタンを押してPHPへのパスを追加します。

XAMPPをインストールしたディレクトリのPHP実行パスが「c:¥xampp¥php」であれば、「;」に続いて「c:¥xampp¥php」を追加します。同様にMySQLなどのパスも追加します。

OS X の場合は、ユーザーのホームディレクトリに設置されている .bash_profile をエディタで編集し、環境変数 PATH にパスを追加します（リスト 1.2）。ファイルパスを追加後、コマンドプロンプトで「php -v」を実行して、phpのバージョンが5.5.9以上であることを確認しましょう。

リスト 1.2：~/.bash_profile にパスを追加

```
export PATH=/Applications/XAMPP/bin:$PATH
```

▶ Composer のインストール

Laravel を利用して開発するには「Composer」を導入する必要があります。Windows 環境では、Composer 公式サイト「Windows install」（https://getcomposer.org/doc/00-intro.md#installation-windows）からインストーラをダウンロードしインストールします。

OS X 環境では、下記のコマンドでインストールします。作成された composer.phar（PHP アーカイブ）を実行可能なディレクトリに移動して、「composer」として実行可能にします。

リスト 1.3：Composer のインストール

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /Applications/XAMPP/bin/composer
```

インストールは標準設定のままで構いません。インストール後はターミナルで「composer --version」を実行してバージョン番号が表示されることを確認しましょう。

1-2-2 Linux における開発環境の構築

本項ではUbuntu 14.04を例にインストール手順を紹介します。Ubuntu以外や異なるバージョンの場合、パスやコマンドなどが異なる可能性があるので、注意してください。

Ubuntu 14.04 では「apt-get install php5」コマンドで、php 5.5.9 をインストールできますが、ここではリポジトリを追加して、php 5.6 をインストールします。

下記のコマンドを実行し、software-properties-common をインストールして、リポジトリを追加します。続いて php5 をインストールします（リスト 1.4）。

リスト 1.4：リポジトリを追加

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:ondrej/php5-5.6
$ sudo apt-get update
$ sudo apt-get install -y php5
```

```
$ echo mysql-server mysql-server/root_password password 'rootパスワード' \
| sudo debconf-set-selections
$ echo mysql-server mysql-server/root_password_again password 'rootパスワード' \
| sudo debconf-set-selections
$ sudo apt-get install -y mysql-server
```

PHP と一緒に、必要となる Apache や MySQL などもインストールされます。

▶ Composer のインストール

OS X 環境と同様に、下記のコマンドで Composer をインストールします。その際に作成される composer.phar を実行可能な場所に移動し、ファイル名を変更して「composer」として実行可能にします（リスト 1.5）。「composer --version」コマンドを実行してバージョン表示を確認し、必要に応じてデータベースなどもインストールしましょう。

リスト 1.5：Composer インストールコマンド

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/bin/composer
```

1-2-3 Laravel Homestead

手元の OS に関係なく開発チームに統一した環境を容易に配布できる Vagrant が、PHP による開発でも利用されています。Vagrant は VirtualBox や VMware はもとより、Amazon EC2 や Rackspace などにも対応しています。

Vagrant を利用することで、仮想マシン上に Laravel 開発環境を構築することも可能です。Laravel には簡単に Laravel 開発環境が構築できる、「Laravel Homestead」と呼ばれる公式 VagrantBox が用意されています。

本項では OS X や Ubuntu をホスト OS として、VirtualBox と Vagrant、Homestead を利用する方法を紹介します。

▶ VirtualBox のインストール

「VirtualBox」は、公式サイト「Download VirtualBox」（https://www.virtualbox.org/wiki/Downloads）から、手元の OS に合致するパッケージなどをダウンロードしてインストールします。Vagrant の利用に特別な設定などは特に必要ありません。

▶ Vagrant のインストール

公式サイト「DOWNLOAD VAGRANT」（http://www.vagrantup.com/downloads）から手元の OS に合致するパッケージをダウンロードしてインストールします。

▶ Laravel Homestead に含まれるソフトウェア

Laravel Homestead に含まれるソフトウェアは、次の通りです。なお、PHP 5.6 と HHVM（HipHop Virtual Machine）は、Homestead.yaml で切り替え可能です。

* Ubuntu 14.04
* PHP 5.6
* HHVM
* Nginx
* MySQL
* PostgreSQL
* Node.js（PM2 と Bower、Grunt、Gulp を含む）
* Redis
* Memcached
* Beanstalkd
* Laravel Envoy
* Blackfire Profiler

▶ Laravel Homestead の利用準備

Laravel Homestead の利用には、統合開発環境としてグローバルで環境を共有する方法と、プロジェクトごとに環境を構築する方法の 2 種類があります。それぞれ VirtualBox や Vagrant のインストール後に実行します。本項でのコマンドの実行は Linux のターミナル操作を対象として説明しますが、Windows など各 OS に合わせてコマンドを実行してください。

グローバルで利用する場合

次のコマンドで homestead を box に追加します（リスト 1.6）。

リスト 1.6：Homestead を追加

```
$ vagrant box add laravel/homestead
```

次に Homestead リポジトリをダウンロード、または clone コマンドなどでローカルに設置します。下記に git clone での設置例を示します（リスト 1.7）。

リスト 1.7：git コマンドによる設置例

```
$ git clone https://github.com/laravel/homestead.git Homestead
```

設置後に Homestead に含まれる init.sh を実行します（リスト 1.8）。実行後はホームディレクトリに .homestead ディレクトリが設置され、設定ファイルの Homestead.yaml が作成されます。

リスト 1.8：init.sh を実行

```
$ bash init.sh
```

プロジェクトごとに利用する場合

プロジェクトごとに Homestead を利用する場合は、プロジェクトの composer.json に追記するか（リスト 1.9）、コマンドを実行して開発時にのみ利用するようにします（リスト 1.10）。

リスト 1.9：composer.json への追記例

```
  "require-dev": {
    "fzaninotto/faker": "~1.4",
    "mockery/mockery": "0.9.*",
    "phpunit/phpunit": "~4.0",
    "phpspec/phpspec": "~2.1",
    "laravel/homestead": "*"
  },
```

リスト 1.10：Composer コマンドでインストールする例

```
$ composer require laravel/homestead -dev
```

インストール後は次のコマンドを実行して Homestead.yaml を作成します（リスト 1.11）。

リスト 1.11：各 OS 環境での Homestead.yaml 作成コマンド

```
# OS X や Linux など
$ php vendor/bin/homestead make
# Windows
vendor¥bin¥homestead make
```

1-2-4 Laravel Homestead の設定

Laravel Homestead の動作設定には、Homestead.yaml を利用します。

▶公開鍵の作成と指定

仮想環境への接続で利用する SSH の公開鍵を作成します。作成済みであれば作成の必要はありません。公開鍵は次のコマンドで作成します（リスト 1.12）。

なお、Windows 環境の場合は「PuTTYgen」などを利用してください。

リスト 1.12：公開鍵の作成

```
$ ssh-keygen
```

ホームディレクトリ内の「.ssh」ディレクトリに公開鍵（id_rsa.pub）が作成されます。このファイルパスを Homestead.yaml の authorize で指定します（リスト 1.13）。接続先の仮想環境で、「~/.ssh/authorized_keys」ファイルに公開鍵の内容がコピーされます。

リスト 1.13：公開鍵の指定

```
authorize: ~/.ssh/id_rsa.pub
```

▶共有フォルダの設定

Homestead.html の設定では、folders に Homestead と共有するディレクトリを記述します。プロジェクトごとに利用する場合は、make コマンド実行時に自動で反映されます。なお、マウントに NFS を利用する場合は type で nfs を指定します（リスト 1.14）。利用しない場合は指定しなくて構いません。

リスト 1.14：NFS マウント利用指定

```
folders:
    - map: "/path/to/Reference.Application"
      to: "/home/vagrant/reference-application"
      type: "nfs"
```

▶ vhosts 設定

sites を利用して vhosts などの設定を記述します。Nginx の設定ファイルなどを直接編集する必要はありません。PHP 5.6 ではなく HHVM を利用する場合は、hhvm を true にします（リスト 1.15）。利用しない場合は指定しなくて構いません。

リスト 1.15：HHVM の利用指定

```
sites:
    - map: homestead.app
      to: "/home/vagrant/reference-application/public"
      hhvm: true
```

▶その他の設定の変更など

環境変数を追加したい場合は variables を利用します。また、仮想環境の IP アドレスやメモリなどもそれぞれ変更できます。利用環境に合わせて変更してください。

▶シェルスクリプトなどの追加

Homestead.yaml と同じ階層に after.sh を設置して、実行したいスクリプトなどを記述すると、vagrant 環境実行時に Homestead 環境に反映できます。例えば、PHP や OS のタイムゾーンを日本時間に設定するには、下記の追記で変更できます（リスト 1.16）。

リスト 1.16：after.sh 利用例

```
#!/bin/sh
sudo ln -sf /usr/share/zoneinfo/Japan /etc/localtime
sudo locale-gen ja_JP.UTF-8
sudo /usr/sbin/update-locale LANG=ja_JP.UTF-8

grep '^date.timezone = Asia/Tokyo' /etc/php5/fpm/php.ini
if [ $? -eq 1 ]; then
sudo cat >> /etc/php5/fpm/php.ini << "EOF"
date.timezone = Asia/Tokyo
mbstring.language = Japanese
EOF
fi
```

1-2-5 Laravel Homestead の実行

Laravel Homestead 環境を利用するには、vagrant コマンドを実行します（リスト 1.17）。

リスト 1.17：vagrant コマンドを使った Homestead 環境の起動

```
$ vagrant up
```

開発環境の削除や、プロビジョニングの反映などを行う場合も vagrant のコマンドを実行します。vagrant の主なコマンドは次の通りです。

▶開発環境の実行停止・再開

このコマンドは主に日常の開発で利用します。実行を停止してもデータベースなどのデータは失われません（リスト 1.18）。

リスト 1.18：開発環境の実行を停止

```
$ vagrant halt
```

なお、停止した環境を再開する場合は次のコマンドを実行します（リスト 1.19）。

リスト 1.19：停止した環境を再開

```
$ vagrant resume
$ vagrant up
```

▶ Homestead.yaml などの変更反映

このコマンドで開発環境の実行を停止させて変更を反映して起動します（リスト 1.20）。

リスト 1.20：設定の反映

```
$ vagrant reload
```

▶開発環境の削除

次のコマンドで仮想開発環境を削除します（リスト 1.21）。コマンド実行時はデータベースなどもすべて削除されます。

リスト 1.21：環境の削除

```
$ vagrant destroy
```

この他のコマンドに関しては、Vagrant 公式サイトの「COMMAND-LINE INTERFACE」（http://docs.vagrantup.com/v2/cli/）を参照してください。

1-2-6 Laravel Homestead 環境への接続

▶ Web ブラウザによるアクセス

前述の Homestead.yaml で設定した host へのアクセスには、hosts ファイルへの追記が必要になります。OS X や Linux 環境では、root 権限で「/etc/hosts」に追記し、Windows 環境の場合は、「C:¥Windows¥System32¥drivers¥etc¥hosts」に追記します。hosts には sites で記述したドメインと vagrant の IP アドレスを記述します（リスト 1.22）。

リスト 1.22：hosts 記述例

```
192.168.10.10  homestead.app
```

▶ ssh による接続

Laravel Homestead 環境への SSH 接続は次のコマンドで実行します（リスト 1.23）。

リスト 1.23：Homestead 環境への SSH 接続

```
$ ssh vagrant@127.0.0.1 -p 2222
```

例えば、エイリアスを利用してコマンドを短縮し、SSH 接続を行う場合は次のようになります（リスト 1.24）。alias コマンドは「~/.bash_profile」に記述しておくと便利でしょう。

リスト 1.24：エイリアス利用例

```
$ alias vm="ssh vagrant@127.0.0.1 -p 2222"
$ vm
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
Last login: Sun Aug  2 20:20:56 2015 from 10.0.2.2
vagrant@reference-application:~$
```

Section
01-03

Composer

Composer は PHP パッケージ管理システムとして急速に人気を博し、近代的な PHP の開発エコシステムではなくてはならないものです。PHP フレームワークとしては早期に Composer に対応した Laravel は、自身も Composer を介して提供される安定した PHP パッケージを活用し、多くの機能を実現しています。また、Laravel 専用の拡張パッケージも Composer を通じて入手します。

1-3-1 Composer とは

Composer は PHP のパッケージ管理システムです。単一パッケージの管理にとどまらず、依存関係を調べて必要となるパッケージをすべてインストールします。その利便性から、現在 PHP のパッケージ管理ツールのデファクトスタンダードとなっています。また、既存パッケージの活用や柔軟なクラス配置が可能であることから、Composer の知識は Laravel での開発に必須です。

Composer でインストール可能な公開パッケージはデフォルトリポジトリである「Packagist」（https://packagist.org/）で管理されています。Web サイトで検索語句を入力し、必要なパッケージを検索できます。また、自由にパッケージを公開することも可能です。

▶ composer.json

Composer は composer.json ファイルの内容に基づき指定されたパッケージの依存関係を解決します。依存解決とは指定パッケージに必要なパッケージを再帰的に調べ、追加でインストールすべきパッケージを見つけ出すことです。インストールされたパッケージは vendor ディレクトリ下に設置されます。

Laravel を構成するパッケージの 1 つ、「illuminate/console」（https://packagist.org/packages/illuminate/console）のみをインストールするケースを例にして確認してみましょう（リスト 1.25）

リスト 1.25：composer による依存パッケージの指定

```
{
  "require": {"illuminate/console": "5.0.22"}
}
```

パッケージ取得時に最低限必要な情報は、require セクションのみです。今回指定したパッケージは「illuminate/console」のバージョン 5.0.22 です。

依存関係は以下に示す通りです（リスト 1.26）。直接依存しているのは 3 つのパッケージ、そしてその中の nesbot/carbon は別の 1 パッケージに依存しているため、illuminate/console 自身を含めて合計 5 個のパッケージがインストールされます。

リスト 1.26：illuminate/console パッケージの依存関係

```
illuminate/console:5.0.22
├── illuminate/contracts: 5.0.*
├── symfony/console: 2.6.*
└── nesbot/carbon: ~1.0
    └── symfony/translation: 2.6.*
```

実際にインストールするには、composer update コマンドをターミナルで実行します。実行後に composer info -i を実行すると、下記の通り、5 個のパッケージがインストールされています（リスト 1.27）。

リスト 1.27：インストール済みパッケージの表示

```
$ composer info -i
  illuminate/console   v5.0.22 The Illuminate Console package.
  illuminate/contracts v5.0.0  The Illuminate Contracts package.
  nesbot/carbon        1.18.0  A simple API extension for DateTime.
  symfony/console      v2.6.6  Symfony Console Component
  symfony/translation  v2.6.6  Symfony Translation Component
```

1-3-2 ローダーの仕組み

Composer はパッケージの依存解決とインストールの他に、オートロード機能と呼ばれるクラスファイルの読み込み機能を提供しています。この機能のおかげでソースファイルに include や require 文を記述する必要がありません。

Composerが提供するオートローディングの機能には、下記にあげる3種類があります。

1. PSR-4規約によるクラスのオートロード
2. クラスマップによるクラスのオートロード
3. 指定ファイルの起動時読み込み

PSR-0規約によるオートロードも提供されていますが、より使用しやすいPSR-4が登場しているため、本書では取り扱いません。

▶ PSR-4規約によるオートロード

PSRはPHPライブラリやCMSの開発者グループである、PHP-FIG（http://www.php-fig.org/）により定められている規約です。コーディングスタイルやオートローディングなどの共通化を目的としています。PSR-4規約は、クラスファイルを自動的に読み込み可能にするため、完全修飾クラス名とファイルパスの対応付けを定義しています。

例えば、\Foods\Sushi名前空間をcomposer.jsonの存在するディレクトリからの相対パスNigiriに結び付けているケースを考えましょう（リスト1.28）。

リスト1.28：PSR-4規約によるオートロード定義

```
{
  "autoload": { "psr-4": { "Foods\\Sushi\\": "Nigiri" } }
}
```

上記コード例により、\Foods\Sushi名前空間下のクラスへのアクセス時、そのクラスがロードされていない場合、対応するNigiriディレクトリ下のファイルが読み込まれます（表1.1）。

表1.1：完全修飾クラス名と相対ファイルパスの対応例

完全修飾クラス名	相対ファイルパス
\Foods\Sushi\Ika	Nigiri/Ika.php
\Foods\Sushi\AkaGai	Nigiri/AkaGai.php
\Foods\Sushi\Hikari_Mono\Kohada	Nigiri/Hikari_Mono/Kohada.php

名前空間名やクラス名の大文字･小文字･下線をそのまま、ディレクトリやファイル名に対応し、クラス名に拡張子「php」を付与したものがクラスファイル名です。\Foods\Sushi以降の名前

空間の構造が、Nigiri ディレクトリ下のディレクトリ構造と対応します。

厳密に PSR-4 規約に従えば、完全修飾クラス名の最上位名前空間名、前述のコード例の場合では、\Foods に該当する部分は「ベンダー名として知られる」名前を付けることになっています。というのも、一般公開されるプロジェクトの場合、この名前が他のプロジェクトと一致してしまうと、クラス名衝突の可能性が発生するためです。

▶クラスマップによるオートロード

クラスマップによるオートロードでは、ロード対象の PHP クラスファイルを保存するディレクトリを指定します。対象ディレクトリ下の PHP ソース内で定義されているクラスの完全修飾名と、その PHP ファイルパスを対応付ける「クラスマップ」を元にクラスのオートロードが行われます。

ディレクトリ構造や名前空間に縛られず自由にクラスを配置でき、名前空間を持たないクラスもオートロード可能です（リスト 1.29）。

リスト 1.29：クラスマップによるオートロード定義

```
{
  "autoload": { "classmap": [ "lib", "project/common" ] }
}
```

上記コード例の場合では、lib ディレクトリと project/common ディレクトリ中でクラスを定義している PHP ファイルが、オートロード対象です。「composer dump-autoload」コマンドを実行すると、対応付けのためのクラスマップが生成されます。

対象ディレクトリ下であれば、名前空間とクラス名、ファイルパスも自由に付けられる便利なローディング方法ですが、新たなクラス追加やクラスの名前空間変更、クラス名変更、そして、ディレクトリ名やファイル名を変更した場合には、クラスマップを再生成する必要があります。

▶指定ファイルの起動時読み込み

オードローダーが include される時点で一緒に読み込まれる PHP ファイルを指定できます。下記に示す指定ファイルのロード例では、composer.json が存在するディレクトリを基準として、helper.php と lib/bootstrap.php が読み込まれます（リスト 1.30）。

リスト 1.30：指定ファイルのロード指定

```
{
  "autoload": { "files": [ "helper.php", "lib/bootstrap.php" ] }
}
```

1-3-3 composer コマンド

composer コマンドの詳細は、Composer の公式ドキュメント（https://getcomposer.org/doc/03-cli.md#process-exit-codes）で説明されています。本項では、Laravel の開発時に特に役に立つコマンドとオプションを紹介します。ちなみに、実行の詳細を知りたい場合はコマンドに -vvv オプションを付けて実行すると、デバッグ情報が出力されるためトラブル発生時に解決の手助けになります。

なお、Composer は細かい仕様が頻繁に変更されるため、実際の動作が本書に記載された内容と合致しない可能性もあります。また、上記の公式ドキュメントも更新が遅れがちです。コマンドのヘルプを最新情報として参照してください。

▶ update

update コマンドは composer.json の設定に基づき、パッケージを新規インストール、アップデート、削除します。もっとも使用頻度が高いコマンドです（リスト 1.31）。

リスト 1.31：update コマンド

```
$ composer update
```

デフォルトでは --dev オプションが有効で「開発時」として扱われ、require セクションと require-dev セクションに指定されたパッケージすべてが更新対象となります。require-dev セクションには、テストパッケージなど開発時にのみ必要な依存コンポーネントを指定します。

本番環境で require-dev セクションのパッケージを除外する場合は、--no-dev オプションを付けてください。

▶ install

install コマンドは composer.json の代わりに、composer.lock ファイルの内容に基づきパッケージを構成します（リスト 1.32）。composer.lock が存在しない場合は、composer.json の情報に基づき構成します。

composer.lock には、パッケージ構成が変更された際に変更時の構成内容が保存されます。つまり、composer.lock を保存もしくはソース管理していれば、過去のある時点のパッケージ構成に戻すことが可能です。

リスト 1.32：install コマンド

```
$ composer install
```

前述の update コマンドと同様に、--no-dev オプションを付けると、require-dev セクションで指定したパッケージを除外できます。

▶ require と remove

require コマンドは特定のパッケージを追加、remove コマンドは削除します。composer.json ファイルの内容を変更し update を実行します（リスト 1.33）。

リスト 1.33：require および remove コマンド

```
$ composer require ベンダー 1/ パッケージ 1: バージョン 1 [ ベンダー 2/ パッケージ 2: バージョン 2...]
$ composer remove ベンダー 1/ パッケージ 1 [ ベンダー 2/ パッケージ 2...]
```

両コマンドでは共に、require-dev セクションを追加・削除の対象にする場合は、--dev オプションを付ける必要があります。

依存解決では require-dev セクションを含む「開発時」扱いです。require-dev セクションを対象外にする場合は、--update-no-dev を使用します。

▶ global

COMPOSER_HOME 環境変数が示すディレクトリは通常、ホームディレクトリ下の「.composer」ディレクトリ下の環境であり、そこで install、update、require を実行します。全プロジェクト共通で使用したいライブラリやコマンドをまとめて管理するために用意されています（リスト 1.34）

リスト 1.34：global コマンド

```
$ composer global require ベンダー / パッケージ : バージョン
$ composer global update
$ composer global install
```

global 指定でインストールしたパッケージに含まれるコマンドを利用するには、COMPOSER_HOME 環境変数が示すディレクトリ下に存在する vendor/bin ディレクトリに実行パスを通す必要があります。また、IDE ではコード補完を利用するため、vendor ディレクトリを全プロジェクト共通の PHP ライブラリ（インクルードディレクトリ）として指定する必要もあります。

▶ self-update

self-update は Composer 自身を更新するコマンドです。ユーザー権限では変更できないディレクトリへ Composer をインストールしている場合は sudo を付けて実行する必要があります（リスト 1.35）。

リスト 1.35：self-update コマンド

```
$ composer self-update
```

--rollback オプションを付与して実行するごとに、1 つ前にインストールしたバージョンへ戻すことが可能です。古いバージョンの Composer コマンドは、COMPOSER_HOME 環境変数が示すディレクトリに保存されています。不必要なバージョンであれば削除しましょう。--clean-backups オプションを付けて実行すれば、古いバックアップは削除されます。

▶ create-project

Composer を採用している多くのフレームワークや CMS は、create-project コマンドで基本的なプロジェクト構造を構築できます（リスト 1.36）。バージョンはパッケージ名の後にコロン「:」で区切って指定することも可能です。

リスト 1.36：create-project コマンド

```
$ composer create-project ベンダー／パッケージ［インストールパス］［バージョン］
```

▶ dump-autoload

dump-autoload コマンドは、Composer がサポートしているローディングに関する情報ファイルを生成するコマンドです（リスト 1.37）。

リスト 1.37：dump-autoload コマンド

```
$ composer dump-autoload
```

composer.json 中の autoload セクションの指定を変更した場合、PSR-4 規約による定義を行った場合、クラスマップによるロード対象クラスを追加や変更、削除した場合は、ローディングの管理ファイルを生成し直すため、dump-autoload コマンドを再実行する必要があります。

--optimize または -o オプションを付けて実行すると、その時点で PSR-4 規約のローディング対象となっている全クラスの情報がクラスマップに含まれます。これでロードを素早く実行で

きます。ただし、PSR-4 規約によるローディングよりもクラスマップが優先されるため、PSR-4 規約下の名前空間やクラス名、ファイル名を変更すると、ローディングが正常に動作しない場合があります。開発時は利用せず、プロジェクトの変更が起きない実機環境で活用しましょう。

ちなみに、composer.json の update 終了時に実行されるコマンドとして、「php artisan optimize」コマンドが指定されています。この optimize コマンドはデフォルトで dump-autoload コマンドを --optimize オプション付きで内部実行するため、上述したローディングの不具合が発生する可能性があります。「php artisan optimize --psr」に変更することで、--optimize オプションを抑制可能です。

▶ clear-cache

clear-cache コマンドは Composer のローカルキャッシュを消去します（リスト 1.38）。

リスト 1.38：clear-cache コマンド

```
$ composer clear-cache
```

install や update 系のコマンド実行でパッケージが取得されると、情報と共にパッケージがローカルにキャッシュされます。通常はキャッシュを意識する必要はありません。しかし、ごく稀に依存パッケージが更新されているにも関わらず、update 時にその変更が認識されない状況では、キャッシュをクリアすると正しく反映されるようになるケースがあります。

▶ info と show

info コマンドと show コマンドはいずれも同じ動作であり、現在の PHP 環境とインストール済みの全パッケージの状態を表示します（リスト 1.39）。

リスト 1.39：info および show コマンド

```
$ composer info
$ composer show
```

パッケージ名を付けると、そのパッケージの情報が取得され表示されます。情報取得には時間が掛かるため、複数パッケージの情報を調べる場合は、前述の Web サイト「Packagist」にブラウザでアクセスして、それぞれの情報を閲覧するほうが素早く効率的です。

Chapter

02

Laravel の基本

本章以降では、Laravel による開発に必要となる知識を説明します。本章では実際に Laravel をインストールして、その構造を見ていきます。また、Laravel に必要な最低限の設定と開発に使うコマンドを紹介します。基本的なコンポーネントについて確認しながら、サンプルコードで実装方法をチェックしてみましょう。

Section
02-01

はじめての Laravel

「Laravel」プロジェクトをダウンロードして、インストールする方法を説明しましょう。学習段階では何度かインストールする必要に迫られるはずです。続いて、ファイルの適切な設置場所を把握するため、ディレクトリ構造を確認します。また、Laravel に必要な設定を解説します。また、Laravel に搭載されている代表的なコマンドツールを紹介します。

2-1-1 Laravel のインストール

Laravel のインストールには 2 つの方法が用意されています。専用インストールコマンドを使用する方法と Composer を利用する方法です。通常は専用インストーラを利用することで、素早くプロジェクトディレクトリを作成できます。なお、専用インストーラは Composer コマンドでインストールする必要があります。

▶専用インストーラでのインストール（laravel new コマンド）

Laravel プロジェクトを構築する専用インストーラを使用する方法です。最初に Composer で専用インストーラをインストールします（リスト 2.1）。

リスト 2.1：Laravel インストーラのインストール

```
$ composer global require "laravel/installer=~1.1"
```

続いて、COMPOSER_HOME 環境変数が示すディレクトリとして、通常はホームディレクトリ直下の「~/.composer」にある vendor/bin ディレクトリへ実行パスを通します。Windows 10 の場合は、「C:¥Users¥<ユーザー名>¥AppData¥Roaming¥Composer¥vendor¥bin」です。後はコマンド 1 つで Laravel プロジェクトを作成できます（リスト 2.2）。

リスト 2.2：Laravel インストーラによるプロジェクト作成

```
$ laravel new プロジェクトパス
```

最新の Laravel プロジェクトが ZIP 形式でホストされており、laravel コマンドはこの ZIP ファイルをダウンロードして、指定したプロジェクトパスに展開します。

万が一、OS X や Linux 環境で laravel コマンドが正常に動作しない場合は、一度 vendor ディレクトリを削除した上で、「composer global install」コマンドを実行してください。

なお、バージョン 5.2 のリリース後に、バージョン 5.1 をインストールするオプションが追加される予定です。

▶ Composer でのインストール（composer create-project）

Composer コマンドを使って直接インストールする方法です（リスト 2.3）。

リスト 2.3：Composer によるプロジェクト作成

```
$ composer create-project laravel/laravel:5.1 プロジェクトパス
```

指定したプロジェクトパスにディレクトリが作成され、プロジェクトに必要なファイル一式が準備されます。公式ドキュメントでは「--prefer-dist」の付与が指示されていますが、プロジェクトの composer.json 内で指定されているため、その必要はありません。

▶ Laravel の日本語化

Laravel を開発する Taylor Otwell 氏は英語しか理解できず、同氏が理解できないものは Laravel に取り込まない方針が採択されています。そのため、フレームワークには英語以外の言語ファイルは標準で用意されていません。また、Laravel のコードには英文コメントが豊富に含まれており、コメントもドキュメントの一部であると考えられています。

英語ドキュメントやソース内の英語コメントが苦手な場合は、Laravel プロジェクトに日本語言語ファイルを追加し、設定ファイルなどのコメントを日本語へ翻訳するパッケージ「comja5」（https://github.com/laravel-ja/comja5）をインストールしましょう（リスト 2.4）。

リスト 2.4：日本語化ツール comja5 のインストール

```
// プロジェクトへインストール（プロジェクトのルートディレクトリで実行）
$ composer require laravel-ja/comja5:~1
$ ./vendor/bin/comja5

// グローバルにインストール
$ composer global require laravel-ja/comja5:~1
$ comja5
```

2-1-2 ディレクトリ構造

前項の説明に従って、実際に Laravel をインストールし、ディレクトリ構造を確認しましょう。

Laravel 最初の LTS である 5.1 のディレクトリ構造を下記に示します。なお、Laravel は自由なフレームワークであり、ほとんどのクラスファイルはローディング可能である限り、自由に配置できることに留意しましょう。

```
├── app             ----
|   ├── Console         |
|   ├── Events          |
|   ├── Exceptions      |
|   ├── Facades         | アプリケーション自身のロジックを設置するディレクトリです。
|   ├── Http            | PSR-4 規約下にあり アプリケーションのベンダー名として
|   ├── Jobs            | \App 名前空間下にあります。
|   ├── Listeners       |
|   ├── Policies        |
|   └── Providers   ----
|
├── bootstrap       ----  Laravel フレームワークの起動コードが設置されています。
|   └── cache
|
├── config          ----  設定ファイルを設置するディレクトリです。
|
├── database        ----
|   ├── factories       | データベースに関連するディレクトリです。
|   ├── migrations      |
|   └── seeds       ----
|
├── public          ----  Web サーバのドキュメントルートとして指定します。
|
├── resources       ----
|   ├── assets          | リソースは資源という意味です。
|   ├── lang            | アプリケーションを動作させるために必要なファイルを設置します。
|   └── views       ----
|
├── storage         ----
|   ├── app             | フレームワークが使用するファイルを保存するためのディレクトリです。
|   ├── framework       |
|   └── logs        ----
```

```
|
├── tests          ----  テストを設置するディレクトリです。
|                        phpunit.xml ファイルでこのディレクトリが指定されています。
|
└── vendor         ----  Composer でインストールしたパッケージが配置されています。
```

上述のディレクトリ構造が示すコンポーネントの関連については、次節「2-2 はじめてのアプリケーション」で解説します。

2-1-3 アプリケーションの設定

Laravel は規約よりも設定を重視するフレームワークであり、デフォルト値はほとんどの状況でそのまま利用できるように設定されています。そのため、最小限の設定を変更するだけで開発を開始できます。本項では主要な設定を説明します。

▶ディレクトリパーミッション

storage 下と bootstrap/cache ディレクトリ下は、Web サーバの実行ユーザーによる書き込みを可能にする必要があります。高いセキュリティを確保する必要がない開発環境では、全ユーザーに対して書き込みを許可する指定も可能です（リスト 2.5）。

リスト 2.5：Linux 系システムで全ユーザーに対する書き込み許可を与える

```
$ sudo chmod -R a+w storage/*
$ sudo chmod -R a+w bootstrap/cache
```

もちろん、実環境ではセキュリティに配慮する必要がありますが、求められるセキュリティレベルや OS、デプロイ方法による差異が大きいため、一概には説明できません。本書はあくまでも Laravel の解説に注力するため、実環境の設定には触れません。適切に設定してください。

▶ .env ファイル

プロジェクトルートに存在する .env ファイルは、機密性の高い設定内容や動作環境により切り替える値を保存するためのファイルです。インストール方法によっては存在しないケースがあるので、必要に応じて「.env.example」ファイルを .env としてコピーしてください。

なお、安全のため、Git などのソース管理の対象から外しましょう。フレームワークに含まれ

る .gitignore では標準でソース管理の対象外に設定されています。

ここでは .env の冒頭にある、特に重要な 3 行を確認しておきましょう（リスト 2.6）。

リスト 2.6：.env ファイルの先頭

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=tmGnFP7ovQYlqyts7uG5Cdi6GRVuTiTc # ランダムな英数字 32 文字
```

1 行目の APP_ENV は Laravel の動作環境の名前です。詳細は「2-3-1 環境設定」（P.053）で説明します。開発時は local に設定しておきます。

2 行目の APP_DEBUG はアプリケーションがデバッグモードであるかを設定します。true でデバッグモードとなり、例外発生時にスタックトレースが表示されます。また、artisan optimize コマンド実行時に、コンパイル済みファイルが生成されなくなります。開発段階では true に設定しましょう。逆に本番環境では表示情報が脆弱性に繋がるため、必ず false に設定します。

3 行目の APP_KEY はアプリケーションキーで暗号化に利用されます。ランダムな英数字 32 文字が設定されていない場合は、次のコマンドで設定します（リスト 2.7）。なお、アプリケーションキーは暗号化のキーとして利用されるため、暗号化された値が保存された後に変更してはいけません。復号できなくなります。

リスト 2.7：Artisan コマンドによるアプリケーションキーの生成

```
$ php artisan key:generate
```

.env には、上記の他にもデータベースとメールの認証情報、各種ドライバの設定情報が含まれています。必要に応じて設定しましょう。DB_CONNECTION 項目はクエリビルダや Eloquent ORM で接続名を省略する場合のデフォルト接続名を指定します。値は config/database.php ファイル中の connections 配列でキーとして定義されている接続名（sqlite、mysql など）を指定します。

sqlite 接続で指定されているデータベースファイル（storage/database.sqlite）はインストール時に存在していません。SQLite を使用する場合は、touch コマンドなどで空ファイルを作成し、Web サーバから読み書きできるようにパーミッションを設定しましょう。

なお、あらかじめ用意されている項目以外に機密情報を設置する必要があれば、このファイルに自由に追加できます。「2-3-1 環境設定」（P.053）で詳述します。

▶コンパイル済みコアファイルの削除

Laravel はクラスファイルの読み込みにかかるオーバーヘッドを短くするため、あらかじめ必要なコアファイルを連結して速度を上げる仕組みが用意されています。Composer の update 実行後に自動実行される artisan optimize コマンドが、デバッグモード false 時に生成します。

このファイルが存在するとスタックトレースがこのファイルを示してしまうため、例外が多発する可能性がある開発中は削除しておきましょう。Artisan コマンドで削除できます。

リスト 2.8：コンパイル済みコアファイルの削除

```
// 常に削除
$ php artisan clear-compiled

// デバッグモードが true 時に削除
$ php artisan optimize
```

▶ Whoops の利用

デバッグモードでエラーや例外発生時に表示されるデフォルトページは、スタックトレースしか情報が表示されません。Laravel 4.x まではエラーレポートパッケージ「Whoops」（https://filp.github.io/whoops/）が利用され、リクエストの内容が詳細に表示されていました。

Laravel は例外発生時の処理も簡単に指定できるので、簡単に Whoops を利用できます。開発時の選択肢として、エラーレポートに Whoops を利用する場合は、まず Composer で Whoops をインストールします（リスト 2.9）。

リスト 2.9：Whoops を依存パッケージとしてインストール

```
$ composer require filp/whoops:~1.0
```

続いて app/Exceptions/Handler.php の render メソッド先頭へ、以下に示す if 文を追加します（リスト 2.10）。

リスト 2.10：デバッグモードの例外発生時に Whoops を表示

```
public function render($request, Exception $e)
{
    if (config('app.debug'))
    {
        $whoops = new \Whoops\Run;
        $whoops->pushHandler(new \Whoops\Handler\PrettyPageHandler());
```

```
            return new \Illuminate\Http\Response(
                $whoops->handleException($e),
                $e->getStatusCode(),
                $e->getHeaders()
            );
        }

        return parent::render($request, $e);
    }
```

2-1-4 Artisan コマンド

Artisan コマンドは Laravel のコマンドラインツールです。アプリケーションの一部として開発することもできます。本項では開発に便利な主要コマンドを説明します。なお、list コマンドにより、使用可能な全コマンドを表示できます。

Artisan コマンドでは共通に使用できるオプションがあります。その中の --ansi は ANSI エスケープシーケンスに対応しているターミナルで表示をカラー出力します。-v はエラー発生時にスタックトレースを表示しますので、自作コマンドのデバッグに利用できます。

▶開発に役立つコマンド

clear-compiled

コンパイル済みクラスファイルを削除します。

リスト 2.11：artisan clear compiled コマンド

```
$ php artisan clear-compiled
```

env

現在の動作環境を表示します。

リスト 2.12：artisan env コマンド

```
$ php artisan env
```

serve

PHP の組み込みサーバで Laravel プロジェクトを動作させます。サーバの停止には、CTRL+C などの割り込みキー操作を使います。デフォルトのホスト名は localhost で、--host オプションで変更可能です。また、ポート番号は 8000 がデフォルトで、--port で変更できます。

リスト 2.13：artisan serve コマンド

```
$ php artisan serve [--host= ホスト名 ] [--port= ポート番号 ]
```

tinker

Laravel プロジェクトを REPL（Read-Eval-Print Loop）で動作させます。起動前に読み込むファイルを指定できます。

リスト 2.14：artisan tinker コマンド

```
$ php artisan tinker [include ファイル…]
```

app:name

PSR-4 ローディング規約下の app フォルダに対する名前空間を変更します。デフォルトは App です。これにより、app フォルダ下の PHP ファイル中の上位名前空間や設定など必要なファイルが書き換えられます。

リスト 2.15：artisan app:name コマンド

```
$ php artisan app:name 名前空間名
```

key:generate

ランダムな英数字文字列を生成して、.env の APP_KEY へ設定します。--show オプションを指定した場合、.env は変更せずに新しい文字列を表示します。

リスト 2.16：artisan key:generate コマンド

```
$ php artisan key:generate [--show]
```

route:list

ルート定義の一覧を表示します。--name でルート名、--path で URI により、表示アイテムをフィルタリングできます。

リスト 2.17：artisan route:list コマンド

```
$ php artisan route:list [--name= ルート名 ] [--path= パス ]
```

Section
02-02

はじめてのアプリケーション

本節では、Laravel の全体構造を把握し、基本的なコンポーネントの概念を理解するため、簡単なサンプルアプリケーションを紹介します。Laravel を構成するコンポーネントが、実際の Web アプリケーションでどのように使用されるかサンプルプロジェクトで確認しましょう。なお、本章での「コンポーネント」とは、フレームワークが持つ機能の構成単位を意味します。

2-2-1 アプリケーション構造

GUI を持つ多くのフレームワークは、MVC（Model View Controller）アーキテクチャを採用しています。その目的は GUI に関わるクラスの責務（関心）を、モデル、ビュー、コントローラの 3 つに分担することで、複雑になりがちなプログラムを俯瞰しやすくすることです。

コントローラの責務

- ユーザーからの入力を受け取る
- ビューを選択、生成する

ビューの責務

- ユーザーに対し情報を出力する

モデルの責務

- アプリケーションが表現しようとする機能や概念の実装
- データ構造とそれを操作する全て（処理、検証、保存など）
- データ層へのアクセス（狭義）
- すべてのロジックとデータ（広義）