ざくっと

Laravel5

5.1
LTS
対応

Hubサイトで学ぶ

イベント・リスナー
キュー・ジョブ
Artisanコマンド

Slack・Trelloを活用

HIROHISA KAWASE

ザクッと **Laravel5**、**Hub** サイトでチュートリアル

Hub サイトを構築しながら、Artisan コマンド、イベント、キューについて学びましょう。

Hirohisa Kawase

This book is for sale at http://leanpub.com/zakutto5-hub-site

This version was published on 2015-12-27

Contents

1. 初めに

購入いただきありがとうございます。

1.1 当書について

Laravel の機能の中でも初心者には理解しづらく、一般に関心も薄い Artisan コマンドとイベントシステム、キューの3コンポーネントについて、チュートリアルを通じて学ぶ電子書籍です。チュートリアルではシンプルな Web サービスの「ハブサイト」を作成します。

ハブサイトとは、IFTTT[1](イフト) やZapier[2](ザピエル) の Web サービスのように、別々の Web サイトサービスを結び付けるサービスです。これらのサービスはアプリからの投稿や、定期的に Web サービスをチェックした結果に基づいて、別のサービスに情報を送ります。様々な Web サービスを結びつけ、便利に活用できるのです。起動側のサービスと受け手側のサービスを 1 対 1 で結びつけます。

両サイトはユーザー操作が簡単で、利用してみれば使い方は習得できます。ハブサービスがどのようなものであるか概念がつかめない方は、一度使用してみてください。

IFTTT は無料ですが、対応サービスの中には十分に機能を利用していなかったり、古い API に対応したままで現状のサービスではうまく動作しなかったりするものがあります。

Zapier は無料で使用できるタスク（2 つのサービスを結びつけたもの）が 5 つしかなかったり、チェック間隔が長かったり、一日当りの実行数の制限が実用にはきつかったりします。ただし現状の Web サービスにきちんと対応しているため有料版を利用すると便利です。

Web サービスはより便利することで利用者を獲得するため、API を利用し他サービスと連携できるように進化しています。たとえばカレンダーサービスの Sunrize カレンダーは、Google、FaceBook、GitHub、Wunderlist など多くの「タスク」や「スケジュール」を扱うサービスと連携できます。しかし存在している多くの Web サービスと連携するのは大変です。全てのサービスに対し様々な連携を提供することは無理でしょう。

こうした状況でハブサイトは、さまざまなサービスを「連携」することに特化し運営されています。ユーザーは起点となるサービスと情報の受け手のサービスをペアで登録します。定期的に起点のサービスを調べ、登録した変化が発生したら、受け手側のサービスへ情報を転送します。

最近では、Yahoo Japan からハードとも連携できるmyThings[3]というサービスもランチされました。本書でも紹介しているチャットサービスの Slack は、日本時間の 2015 年 12 月 16 日に外部との連携サービスを Slack ディレクトリーとして、Web に公開し、その拡張性を広く知らしめようとしています。こうした「連携」を提供するサービスはこれからも段々増えていくでしょう。

今回のチュートリアルは、Web ページを作成しません。その代わりに Laravel の機能を利用し、最終的に次のような形態で Hub サイトを実現します。

[1] https://ifttt.com/
[2] https://zapier.com/
[3] http://mythings.yahoo.co.jp/

- Web サービスをチェックする機能をコマンドで実装する。
- 上記チェックコマンドをスケジューラーで定期的に実行する。
- チェック時、状況をイベントで発行する。
- イベントとそれを処理するリスナーの組み合わせを Laravel のメカニズムで指定する。
- リスナーにより、特定の Web サービスに通知する。
- イベントリスナーをキュー経由で実行してみる。
- Job クラスを使用しチェック機能をキュー経由で実行してみる。

Hub サイトとして機能や設計的なベストではありませんが、Laravel の裏側に用意された機能と仕組みを学ぶためには面白い内容にしています。

これにより、以下の機能に触れてみましょう。

- Artisan CLI
- Artisan コマンドスケジューラー
- イベント（ただし、ブロードキャストは扱いません）
- キュー

今のところ、用語がわからなくても安心してください。この後に学習しましょう。

本書はカナダの自主出版サービスである、Leanpub を利用して出版しています。編集者はいません。著者が自分の知識をそのまま書籍として発行できます。そのため私のレベル通りの文章が、誤記と一緒に発行されてしまいます。間違いに気づかれたら hirokws@gmail.com までお知らせください。

ただし長所もあります。本書のようにセールスが見込まれない（半年で数十冊…）内容の書籍も、発行することが可能です。また、いち早くフレッシュな情報をお伝えできることも魅力です。旧来の紙の出版物と異なり、原稿を書き上げてから出版されるまで数カ月かかるということはありません。

出版後に内容を追加、修正することが簡単にできます。本書ではご期待があれば数章追加するかも知れません。修正は随時行います。内容はバージョン 5.1 向けとさせていただきます。

1.2 必要なスキルと要件

この書籍を読まれる方は、多少なりとも Laravel に慣れたレベルの方を想定しています。

- PHP は一通り OK
- Laravel で作った Web サービスを自力で実働環境へデプロイし、実行できる

「公式ドキュメントを読み、Web 上で適切な情報を見つけ、わからない点はコアの PHP クラスを読んで解決できるレベルの方」には、本書は必要ありません。そこまでのレベルには行き着いていないが、Artisan コマンドやスケジューラー、イベントシステム、キューをどのように活用するのか具体的なイメージを掴みたい学習者向けの内容です。

Hub サイトですから、実際に様々なサービスと連携します。動作させるための VPS などは用意しなくても、インターネットに接続できる開発環境があれば、内容を試すことが可能です。もちろん、以下のような一般的なサーバー上で実行することも可能です。

- サクラやカゴヤ、z.com、AWS、Linode、DigitalOcean などの VPS やクラウドサービス
- もしくは Laravel の動作要件を満たし、毎分の cron 設定が可能な共有サーバー
- もしくはインターネットへ公開しているオンプレミスの Web サーバー

サーバーとして動作させる OS は、Linux ディストリビューションの”Ubuntu” を想定しています。その理由は公式 Vagrant Box である Homestead でも利用されているからです。しかし他の Linux ディストリビューションでも、本書が取り扱う範囲では大差はないかと思います。Unix 系の場合は使用するコマンドやオプションが多少異なることもあります。Windows サーバーでは、大幅に異なるでしょう。

Windows 環境ではファイルパス名で、ディレクトリー区切り文字やドライブ番号を先頭に付けるなどの違いがあります。全てを本書の中で説明できません。環境に合わせて読み替えてください。

1.3 コードについて

掲載したコードは Github から入手可能です。通常の著作物の一部として公開しています。

- 公開先: https://github.com/HiroKws/zakkuto-laravel-hub-site

書籍中のコード中の長い行は Leanpub の電子書籍生成システムにより、複数行へ分割されます。分割された行の最後へ'\'が付加されます。(コード以外の部分で英単語が分割される場合は、行末がハイフン (-) になります。)

コードのフォーマットに特別な意図はありません。チュートリアルを実行する場合は、お好きなようにリフォーマットしてください。

テストの実行は本書の範囲外です。

書籍上で「何をやっているか」が一目で追いやすいように、原則メソッド内にロジックをだらだらと書いています。コードをきれいにまとめ上げてはいません。(コードをどのように書くべきか、構成すべきかは、個人やチームにより差がある部分です。コーディング規則で縛って「…すべき!…すべき!」といじめるつもりはありません。)

また、知識を吸収する学習段階で「べきべき」を持ちだすと、抵抗感により習得が遅れるものです。この書籍の内容を実用にするのであれば、最終的にぜひ皆さんの「…べき」に従いリファクタリングしてください。私も本書を書く気にさせた元々のシステムをこの本のバージョン

で置き換えるつもりですが、その際に私の気に入るように手を加えるつもりです。(たとえば、外部 Composer パッケージを使っている箇所は Service にしていますが、本来はインターフェイスを導入し、交換性を上げるべきでしょう。しかし、インターフェイスによる分離や複雑なデザインパターンの導入は、コードの読みやすさを大きく損ねてしまいます。それは本書の目的とそぐいません。実用としてシステムを構築する場合は、外部パッケージはいつメンテが放棄されるか分かりませんので、交換性を備えておくのは賢い考えです。)

PHP フレームワークの学習時点であるならば、特に IDE を利用することをおすすめします。NetBeans は無料で利用でき、UI が日本語です。PhpStrome は有料ですが、きれいな GUI が気分を盛り立ててくれます。どちらも継承元のクラスを探したり、クラスやメソッド名、PHP コードの補完したり、親切にサポートしています。より深く知りたくなったときに、Laravel の内部を覗くためにとても役立ちます。

1.4 謝辞

図中の図形デザインは、Freepik により作成されたものです。

All icons in this book are designed by Freepik.

1.5 改訂版について

Laravel5.0 のリリースの後、長期メンテナンス版(3年間)のバージョン 5.1LTS が発表されました。

メジャーバージョンは同じで、内部的な動作は 5.0 と大差ないのですが、5.1LTS のドキュメントで説明されなくなった 5.0 の機能もあります。そうした多くの機能はバージョン 5.1LTS でも利用できます。

LTS リリースに合わせドキュメントも大幅に書き直されました。用語と概念はわかりやすいように整理されました。ドキュメントで紹介されている基本的な手法や用語が変更になりました。つまり 5.0 より概念はわかりやすく、かつ機能は使いやすくなっています。

そして何しろ LTS です。長期サポート版です。バグフィックスは2年間、セキュリティー脆弱性の修正は3年間提供されます。安心して利用できます。

そこで、今回は 5.1LTS に合わせ内容を改定しました。前バージョンの本書は zip ファイルで提供します。

2. Laravel の準備

最初に開発環境へ真新しく Laravel をインストールしてください。Web サーバーの設定（ドキュメントルートを public フォルダーへ設定…なんたらかんたら）は当面不要です。

続いて、Laravel の準備を行いましょう。しばらくターミナル（端末、Windows であればコマンドプロンプトやパワーシェルなどのコンソールプログラム）での作業になります。

コンパイル済みクラスファイルの削除

プロジェクトを実働環境で動かしているなら、Laravel が生成するコンパイル済みクラスファイルは動作スピードアップに役立ちます。しかし、開発環境ではエラーがどこで発生しているのかを追いかける邪魔になりますので、削除しておきましょう。以下のコマンドを実行してください。

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

本来コンパイル済みクラスファイルは、.env ファイルの APP_DEBUG=true の指定で、デバッグモードに設定してあれば生成されません。しかし、バージョンが変更になる時に取り扱いが変わったり、バグで生成されることがあります。Laravel に慣れている方ならば、例外発生時のスタックトレースを見れば「ああ、作成されているな」と分かるでしょう。もし、初心者であれば、インストール直後に念のため、このコマンドを叩いておきましょう。

この先、自分でいろいろと工夫していくと、クラスが見つからないという PHP エラーが発生し、解決方法が思い浮かばないことも起きるかもしれません。

その場合、オートロードのために Composer が生成するファイルと、現状のクラス構成が一致していないことが大抵の場合原因です。オートロードのマップファイルを明示的に再生成してみましょう。解決するかもしれません。

```
$ composer dump-autoload
```

日本語化

Laravel では設定ファイルのコメントもドキュメントと考えています。コメントは英語で書かれています。しかし英語が苦手な方もいらっしゃるので翻訳コマンドを用意してあります。日本語の ja 言語ファイルも同時に作成します。

コメントが全部英語でも平気な方は、この手順をスキップしてください。

GitHub の laravel-ja/comja5 リポジトリーで公開しています。comja5 コマンドを提供しています。このコマンドはインストール済みの Laravel に含まれるファイル中の、英文コメントなどを日本語へ文字列置換します。ですから、Laravel プロジェクトを新たにインストールした直後に実行してください。開発が進んでから翻訳実行すると、開発した部分に含まれる英文が意図せず日本語へ置き換えられる可能性があります。

comja5 コマンドはプロジェクトごとに導入することもできますが、composer のグローバルコマンドとしてインストールすると、使い勝手が良いでしょう。

```
$ composer global require laravel-ja/comja5
```

インストールが終わったら、以下のディレクトリーへ実行パスを設定してください。

```
Linux と Mac の場合：
~/.composer/vender/bin

Windows10 の場合：
C:￥Users￥< ユーザー名 >￥AppDate￥Roaming￥Composer￥vendor￥bin
```

これで comja5 コマンドが実行できるようになりました。Laravel のプロジェクトディレクトリーで以下のコマンドを実行すれば、コメントの翻訳し、日本語の言語ファイルを生成します。

```
comja5 -a
```

生成される日本語は UTF8、行末は Unix/Linux 形式です。Windows でも大抵のエディターでは対応しているので通常問題はありません。

.env ファイル

機密性が高い認証情報や、動作環境により設定ファイル中の値を変更したいものは、.env ファイルに記述します。もし.env ファイルが存在していなければ、.env.example ファイルをコピーしてください。本書で提供するコードの ZIP ファイルに含まれている.env.example にはコメントを含めています。それを.env にコピーすると変更箇所がわかりやすくなります。

続いて以下を参考にし、ファイルを編集します。

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=K2NsYNDi938WYvVGgOLZk7Gp7BO9rCjP
(上記の値は各自の環境により異なります)

...
(省略、ここまで変更しない)
...

MAIL_DRIVER=smtp
MAIL_HOST=smtp サービスのドメイン
MAIL_PORT=上記のポート
MAIL_USERNAME=メールのユーザ名
MAIL_PASSWORD=メールのパスワード
```

APP_KEY にランダムな文字列が設定されておらず、代わりに”SomeRandomString” という文字列が設定されていれば、一度エディターを終了させ、次のコマンドで設定してください。

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

メールを送信できるように、SMTP 関連の設定を行っておきましょう。最初の通知機能はメール送信の実装です。なにせメールも重要な「Web サービス」です。後ほど連携に利用します。

例えば Gmail を利用する場合は、以下のようになります。

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME= ユーザ名 @gmail.com
MAIL_PASSWORD= パスワード
```

もしメール送信を実際に行いたくない、できない状況である場合、もしくは設定が面倒であれば、MAIL_DRIVER に log を指定してください。サーバーに送信される代わりに、strage/logs ディレクトリー下のログファイルにメールの内容が保存されます。

データベース設定

今回、システムの情報を保存する目的には直接データベースを使用しません。しかし、Laravel5.0 から新しく導入されたデータベースによるキューを後ほど試してみます。そのためデータベースの準備が必要です。.env ファイルや config/database.php 設定ファイルを変更し、必要な接続情報などを指定してください。

データベースエンジンに SQLite を使用する場合は、database.php に指定したファイルを予め作成しておく必要があります。デフォルトのまま使用するのであれば、database/database.sqlite です。サイズは 0 バイト、つまり空のファイルにしておきます。touch コマンドで簡単に作成できます。

```
$ touch database/database.sqlite
```

このチュートリアルでは、情報をテキストファイルとして保存しています。Laravel のデータベース関連まで本書に含めてしまうと、学習量が増えるためです。Eloquent ORM やクエリービルダーの知識が少なくても学習が進められるように、データベースを利用していません。もちろんデータベースに保存することも可能ですし、実用化するにはデータベースを利用することをおすすめします。

メール設定

メールの設定ファイルも変更しておきましょう。config/mail.php をエディターで開きます。

ほとんどの設定は.env の編集で済ませました。変更する必要があるのは一行だけです。メールアドレスのバリデーションが行われるため、log ドライバーを使用する場合も、メールアドレスとして正しい値を指定してください。

```
...
'from' => ['address' => ' 自分のメールアドレス', 'name' => ' 送信者名'],
...
```

メール送信ができるか、テストしておきましょう。app/Http/routes.php をエディターで開きます。ルート URL に対する定義、`Route::get('/', 'WelcomeController@index');` の一行を次のように変更します。

```
Route::get('/', function () {
    \Mail::raw(' 本日は晴天なり', function ($message) {
        $message->to(' 送信先メールアドレス')
            ->subject(' テスト送信');
    });

    return ' 送信しました。';
});
```

to メソッドには、実際の送信先メールアドレスを指定してください。

`php artisan serve` で PHP サーバーを起動し、'localhost:8000' へブラウザからアクセスし、メールが送信されるか確認しましょう。送信できない場合は、設定値を見直してください。

この手のちょっとした作業に、Artisan コマンドの Laravel Tinker を使用する方法もありますが、環境によりうまく動作しなかったり、途中で実行を止めた場合に文字色を変更してしまったりと、動作に問題があります。(当方の環境では正しく動作させるために readline PHP 拡張が必要でした。)そのため本書では、Tinker を利用していません。もちろん、自分の環境で Tinker が問題なく動作するならば、'\Mail::raw...' 以下の部分を入力することで、送信テストが可能です。

Mail::row() メソッドはビューを使用せず、文字列をそのままメールの本文として送信します。

STMP サーバーに Gmail を指定する場合、Google のセキュリティーチェックが厳しくなったため、Google の API を経由しない接続は原則通信エラーになります。

コード 503 で、以下のような内容が返ってきます。(実際に返ってくる内容は、頻繁に変更されます。多少異なる可能性があります。)

```
535-5.7.8 Username and Password not accepted. Learn more at 535 5.7.8 http://\
support.google.com/mail/bin/answer.py?answer=14257fi8sm14484270pdb.43 - gsmtp
```

さらに、「ログイン試行をブロックしました」というメールも送信先のアドレスへ送信されてきます。(この振る舞いも変更される可能性があります。)そのメールに含まれるURL、https://www.google.com/settings/security/lesssecureappsにアクセスすると、その時点でログインしている Google アカウントの「安全性の低いアプリのアクセス」の設定ができます。「オンにする」を選択すると、メール送信を受け取られるようになります。

もし、ログインしているアカウントとは別の Gmail アカウントをメール送信のアカウントとして利用している場合は、そのアカウントでログインし、(メールの設定ではなく) Google アカウントの設定から「セキュリティ診断」を選んでください。パスワードの再チェックの後に、項目を設定するように促されます。その中の「安全性の低いアプリの無効化」により、STMP など Google API を使用しない接続でもエラーにしないように指定できます。

※ この手の操作や設定方法も、Google ではよく変更になります。今回は半年立たないうちに数回変更になりました。現状と多少の不一致があっても、ご容赦ください。

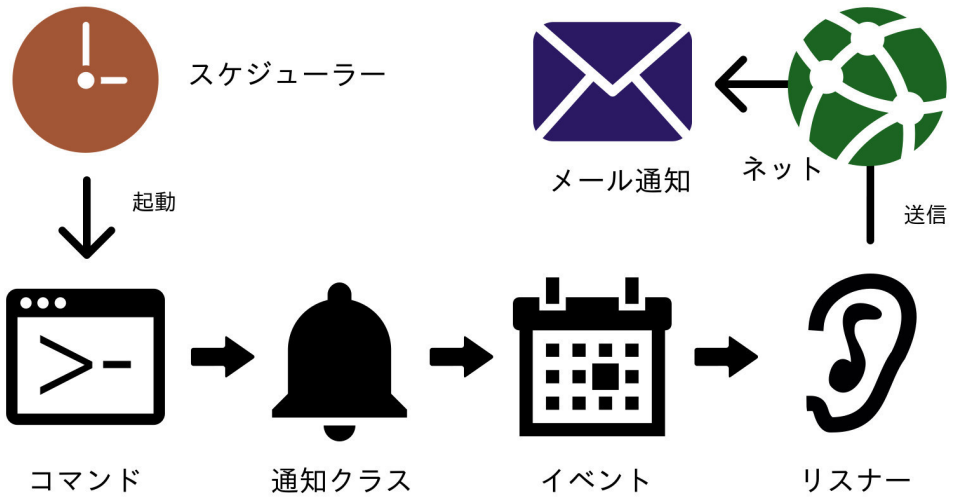
アプリケーションの設定

アプリケーションの設定は config/app.php ファイルで行います。以下の 2 項目を設定してください。(記述していない項目は変更しません。)

```
'timezone' => 'Asia/Tokyo',
'locale' => 'ja',
```

Laravel がコアでも使用し、本書でも活用している日付時間操作ライブラリーの Carbon でも、新しいインスタンスを生成した場合のデフォルトタイムゾーンはここで指定した'timezone' 設定値になります。Laravel が初期処理で、PHP の日付時間関数のデフォルトタイムゾーンをこの設定値で指定するためです。

3. リマインダー



リマインダーの流れ

いよいよ、Hub サイトの作成です。最初はリマインダー機能です。登録しておいた時間にメッセージを送ってくれます。すでに、Reminder イベントも作成していますしね。

今回、起動側は他の Web サービスを利用してイベントを発行しません。Laravel 自身の Aritsan スケジューラーの時間起動機能をそのまま利用しましょう。

送信先は一般的なサービスです。メールで自分のアドレスに通知します。他の人のアドレスに送っちゃ嫌がらせメールです。忘れちゃいけません、メールも立派な Web サービスですよ。面倒な送信プロトコルは Laravel に任せられますし、なにせ私達全員馴染み深いシステムですからね。最初に扱うには、ぴったりな主題です。

3.1 reminder コマンド

以降に作成するコマンドは、起動されると特定の Web サービスにアクセスし、新しいメールが来ているかとか、特定の情報があるかとか、通知すべきものがあるのかチェックします。

今回リマインダーに使用する reminder コマンドは、このようなチェックは一切せず、fire:reminder と同じようにイベントを通知するだけです。指定時間に Artisan スケジューラーから起動されます。機能的には一番単純なコマンドです。app/HubConnections/Commands/Reminder.php です。

```php
<?php

namespace App\HubConnections\Commands;

use App\Console\Commands\BaseCommand;
use App\HubConnections\Notifiers\Reminder as Notifire;

/**
 * リマインダー通知コマンド
 *
 * スケジューラーにより起動され
 * メッセージをリマインダーイベントでディスパッチする。
 */
class Reminder extends BaseCommand
{
    protected $signature = 'hub:reminder '
        .'{message : Reminder message}';

    protected $description = 'Notify a reminder message.';

    /** @var Notifire * */
    private $notifire;

    public function __construct(Notifire $notifire)
    {
        $this->notifire = $notifire;
        parent::__construct();
    }

    public function handle()
    {
        // 必要な場合、ここで引数やオプションのチェックを行う
        // ビジネスロジックは別の責務として分離する
        $this->notifire->run($this->argument('message'));

        // 終了コード
        return 0;
    }
}
```

Artisan コマンドは既にお手の物でしょう。ビジネスロジックの本体部分は handle メソッドでした。注目しましょう。

コンストラクターはサービスコンテナの働きで、タイプヒントされたクラスのインスタンスを自動的に依存解決し、渡してくれるんでしたよね。特に依存の解決方法を指定していなくても、インスタンス可能なクラスであれば、それを生成してくれます。

このコマンドで必要なのは、通知ロジックを実装している Notifire クラスです。具体的には App\HubConnections\Notifiers\Reminder クラスです。それをサービスコンテナによる自動インスタンス注入機能により、受け取っています。

では、この通知ロジッククラスを見てみましょう。

```php
<?php

namespace App\HubConnections\Notifiers;

use App\HubConnections\Events\Reminder as Event;
use Illuminate\Contracts\Events\Dispatcher;

/**
 * リマインダー通知ロジッククラス
 */
class Reminder
{
    /** @var Dispatcher * */
    private $dispacher;

    /** @var Event * */
    private $event;

    /**
     * コンストラクター
     *
     * @param Dispatcher $dispatcher
     * @param Event $event
     */
    public function __construct(Dispatcher $dispatcher, Event $event)
    {
        $this->dispacher = $dispatcher;
        $this->event     = $event;
    }

    /**
     * リマインダー通知ビジネスロジック
     *
     * @param string $message
     */
    public function run($message)
    {
        // イベントにメッセージを設定
        $this->event->message = $message;

        // イベント発行
```

```
        $this->dispacher->fire($this->event);
    }
}
```

コマンドのコンストラクターでタイプヒントを指定し、この通知クラスは自動注入されました。自動注入されるクラスのコンストラクターにタイプヒントが記述されていると、それらも再帰的に自動注入してくれます。そのため、このクラスでもイベントデスパッチャーと Reminder イベントのインスタンスが取得できます。

デスパッチャーをよく見ると、Illuminate\Contracts\Events\Dispatcher です。これ、実はインターフェイスです。

「さすがララベル、インターフェイスまで解決してくれるなんて、魔法みたいだな。」

実のところインターフェイスも解決できます。ですがインターフェイスの場合、「このインターフェイスを解決するときは、このクラスをインスタンス化する」という指定が予め必要です。

Illuminate\Contracts の下には、システムが定義している Contract、つまり「契約[1]」が置かれています。Laravel が提供している契約インターフェイスは起動時の初期処理により、サービスコンテナにその具象クラスが登録されます。それにより、契約としてのインターフェイス名を指定しても、その契約を実装している具象クラスのインスタンスを取得できます。

契約とはプログラム間のやり取りを取り決めたもので、PHP ではインターフェイスで定義します。実際には実装するメソッド名とその引数を呼び出されるクラスと呼び出し元との間で取り決めています。

具象クラスではなく、実装を含まないインターフェイスを指定する利点は、「実装の交換」です。Laravel5 から、ほとんどのコアクラスが契約としてのインターフェイスを継承しています。たとえば今回利用した、Illuminate\Contracts\Events\Dispatcher で取得できるディスパッチャーインスタンスを使ってみて、気に入らない点があるとしましょう。それならば、自分でこの契約を実装し、Laravel のコアの代わりに、自作のイベントバスディスパッチャーを受け取れるようにフレームワークを変更できます。

現状の PHP ではまだ、タイプヒントによる引数の制約しかできませんが、PHP7 からネイティブタイプのタイプヒント指定や戻り値の定義もできるようになるようです。そうなれば、より「契約」としてのインターフェイスも PHP 開発者に広がるでしょう。(実際にはこうした PHP に用意されている型は、自動的にタイプキャストが起きてしまうため、制約としては緩くなります。)

では Illuminate\Contracts\Events\Dispatcher インターフェイスで取得できる実体のクラスとは何でしょう。それは今までイベントを発行する時に利用していた \Event ファサードクラスの実体と同じものです。Illuminate\Events\Dispatcher クラスです。実体クラスを確認するには公式ドキュメントのファサード[2]と契約[3]のページを参照してください。もしくは、契約のインターフェイスと実体クラスを結合している Illuminate\Foundation\Application クラスを読んでください。

Laravel のファサードは静的メソッド記法を提供していますが、実際は静的メソッドではなくインスタンス化して利用する通常のクラスです。Laravel が必要に応じてインスタンス化し、静的

[1] http://readouble.com/laravel/5/0/0/ja/contracts.html
[2] http://readouble.com/laravel/5/1/ja/facades.html
[3] http://readouble.com/laravel/5/1/ja/contracts.html

記法のメソッドに対しマジックメソッドを使用し、static ではない通常のメソッドを呼び出しています。

ですから、\Event::fire() と今回のコード中に存在している $this->dispacher->fire() は、同じメソッドを呼び出しており、「イベントの発行」という同じ動作をします。

3.2 Reminder イベントクラス

イベントクラスです。単純化しています。

```
<?php

namespace App\HubConnections\Events;

/**
 * リマインダーイベント
 */
class Reminder extends HubConnectionBaseEvent
{
    /** @var string * */
    public $message = '';

    /**
     * イベントの文字列変換
     *
     * @return string
     */
    public function __toString()
    {
        return $this->message;
    }
}
```

今までのイベントクラスと大差ありません。app/HubConnections/Events ディレクトリーに保管していることに注意を払ってください。

3.3 MailSender イベントリスナー

イベントリスナーは、App\HubConnections\Listeners\MailSender です。名前が示している通り、メールを送信する役目です。

```php
<?php

namespace App\HubConnections\Listeners;

use App\HubConnections\Events\HubConnectionBaseEvent;

class MailSender
{
    /**
     * 受け取ったイベントをメールで送信する
     */
    public function handle(HubConnectionBaseEvent $event)
    {
        // 実行確認のため、ダミーコードとしてログ出力する
        \Log::info($event);
    }
}
```

今のところ、ダミーです。とりあえずうまく動作しているかチェックするため、イベントをログへ書き込んでいます。イベントが間違いなく到達することを確認してから、内容を実装しましょう。

Log::info メソッドの引数は文字列です。イベントクラスインスタンスを直接渡しています。イベントが持っている情報は、それぞれ異なります。リスナー側でイベントに依存する情報を編集すると、取り扱うイベントごとにロジックが必要になります。そこで、イベントのことはイベントに任せています。リスナーとしてはイベントに「文字列化のためのメソッド」が実装されていれば事が足ります。

文字列化のためのメソッドである__toString が実装されていることを保証したいために、StringizableInterface インターフェイスを impliments し、実装を強要する HubConnectionBaseEvent ベースクラスをタイプヒントし、イベントを受け取っています。もしこの__toString を実装していないクラスインスタンスが渡されると、実行時にエラーになります。

この実行時エラーは「引数に渡された内容が間違っている」ということを表します。誤りがあるまま素通りさせるのではなく、「これ間違っているよ」と叱ってもらうのです。素通りさせてしまうと、他の箇所でエラーになったりするため、原因を探さなくてはなりません。今回は単純なコードですが、複雑なロジックの場合、原因から離れた場所でトラブルになると解決するのが面倒になります。ですから、PHP の仕組みを使い、あらかじめ予防策を取っておきます。

では、EventServideProvider で、イベントとリスナーを結びつけましょう。

```php
<?php

namespace App\Providers;

use App\HubConnections\Events\Reminder;
use App\HubConnections\Listeners\MailSender;
use Illuminate\Contracts\Events\Dispatcher as DispatcherContract;
use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServicePr\
ovider;

class EventServiceProvider extends ServiceProvider
{
    /**
     * アプリケーションのイベントリスナーのマップ
     *
     * @var array
     */
    protected $listen = [
        Reminder::class => [MailSender::class],
    ];

    /**
     * アプリケーションのその他のイベントの登録
     *
     * @param \Illuminate\Contracts\Events\Dispatcher $events
     */
    public function boot(DispatcherContract $events)
    {
        parent::boot($events);
    }
}
```

ここでは、結びつけているだけです。

では、Reminder Artisan コマンドを登録し、実行しましょう。App\Console\Kernel を変更します。

```php
<?php

namespace App\Console;

use App\HubConnections\Commands\Reminder;
use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
```

```
    protected $commands = [
        Reminder::class,
    ];

    protected function schedule(Schedule $schedule)
    {
    }
}
```

これでコマンドが登録できました。実行してください。

```
php artisan hub:reminder テスト実行！
```

適当なメッセージを渡して、ログファイルに書き込まれるか確認してください。
うまく行ったら、MailSender リスナークラスへメール送信のコードを入れましょう。

```
<?php

namespace App\Handlers\Events;

use App\Events\StringizableInterface;
use Illuminate\Contracts\Mail\Mailer;

class MailSender
{
    /** @var Mailer */
    private $mailer;

    public function __construct( Mailer $mailer )
    {
        $this->mailer = $mailer;
    }

    public function handle( StringizableInterface $event )
    {
        $this->mailer->raw( $event, function ($m)
        {
            $m->to( ' 自分のメールアドレス', ' 自分' )
                ->subject( 'Hub サイトのメール通知' );
        } );
    }
}
```

‘自分のメールアドレス’ は自分のメールアドレスに書き換えてください。そうしないとエラーが発生します。(たとえ log メールドライバーを使用していてもです。メールアドレスとして有効な文字列にしてください。)

リスナーの handle メソッドは、イベントを受け取るものと決まっています。そのため、サービスコンテナによる自動依存注入は行われません。コンストラクターの自動注入により、メーラーのインスタンスを受け取っています。

Mail ファサードを使う方法もあります。Mail ファサードを使用すれば、コンストラクターは必要なくなります。$this->mailer->raw を \Mail::raw と短く書けます。

ファサードはインスタンス化のコードが不必要ですし、ファサード名＝使用するコア機能のため、直感的で読みやすいのです。一方の自動依存注入を利用したコードはエディターや IDE の補完などの手助けがフル活用でき便利です。どちらでもお好きな方をご利用ください。

どちらを使用するかの判断の基準は、ユニットテストをどの程度行うのかによります。バリバリテストを書くのであればタイプヒントによる記述が便利です。インスタンスを受け取るメソッドにテスト用の代理（スタブ）オブジェクトを渡せば良いため、そうしたオブジェクトを作成するために自分の好きなテストフレームワークが使用できます。ファサードでも同じように使用できますが、ひと手間必要です。

こうした面から考えても、簡単なシステムやプロトタイプのためにさくっと作成する場合はファサード、本格的な開発であれば Laravel サービスコンテナが提供しているタイプヒントによる自動依存注入を活用する方針をおすすめします。

本書は初心者向けであるため、前半はファサードを使用しコードをすっきりさせ読みやすく、理解しやすくしてきました。この章以降はサービスコンテナの機能を活用した自動依存注入を利用していきます。

3.4 使ってみる

さて、いつまでも自分でコマンドを叩いていたのでは全然便利ではありません。早速スケジューラーに登録しましょう。

```
<?php

namespace App\Console;

use App\HubConnections\Commands\Reminder;
use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    protected $commands = [
        Reminder::class,
    ];

    protected function schedule(Schedule $schedule)
    {
        $schedule
```

```
            ->command('hub:reminder  " 燃えるゴミ、鳥よけカゴ出し"')
            ->dailyAt('08:00')
            ->days(1, 3, 5); // 月水金、配列でも OK
        $schedule
            ->command('hub:reminder  " 川瀬さんの誕生日、プレゼントは現金で OK!Paypal 可!!"')
            ->cron('0 8 4 7 *');
    }
}
```

起動間隔の指定に利用できるメソッドは、Illuminate\Console\Scheduling\Event クラスを確認してみるとよいでしょう。メソッドのコードを読んでみると気がつくことがあります。

内部的には expression プロパティーとして、cron 形式の日時時間が保持されています。(時間部分5つ+何に使用しているか不明1つ、たぶん実行ユーザーを指定するためのもの)cron 指定の 5 つのフィールドをそれぞれ設定できる spliceIntoPosition メソッドが用意されていますが、ほとんどのメソッドで活用されていません。そのため、以下のようなことが起きます。

```
// これは思い通りに動く
$schedule->daily()->weekdays();

// これは weekday が無視される
$schedule->weekdays()->daily();
```

daily メソッドは単に expression プロパティーに、'0 0 * * * *'をセットするだけです。weekdays メソッドは曜日を表す 5 番目のフィールドだけを'1-5' に置き換えます。そのため、指定する順番により意図通りに動かないことになります。

確かにメソッドを使用したほうが意図が読み取りやすくなりますが、そのメソッドが値をただセットするのか、それとも特定のフィールドを置き換えるのかを全部覚えるのはやや大変です。

ところで cron の時間指定は面倒でしたか?ここまでやってきて、さほど難しくはないことが理解できたのではないでしょうか。cron の指定形式に慣れてしまえば、直接 cron メソッドで記述してしまうほうが簡単です。私のスケジュールのゴミかご出しは以下のように書き直せます。

```
$schedule
        ->command( 'reminder " 燃えるゴミ、鳥よけカゴ出し"' )
        ->cron('0 8 * * 1,3,5');
```

これも好みです。皆さんも自分で選択してください。

3.5 拡張

現在、起動側とリスナーは 1 対 1 の関係です。しかし、これからいろいろな条件で起動されたイベントで、メール送信を利用されるかも知れません。

そうすると、メールで指定する subject の内容は、通知する内容に応じたタイトルにできたほうが分かりやすいですよね。では、どうしましょうか?subject ごとにリスナーを作成しましょうか?

それともコマンドで指定できるようにし、イベントに情報を含めましょうか？すると、その情報をどうやってリスナーで取得しましょうか？いろいろと考慮するしなくてはなりません。

実際に拡張をしなくても少し考えてみましょう。設計の練習になります。

同様にメールアドレスの変更をできるようにするには、どんな方法で実現しますか？

もし、このリマインダーを Web サイトから指定できるようにするなら、どう実現しますか？コマンドで追加できるようにするとしたらどうしましょうか？

どう拡張するか少し考えてみるだけでも、勉強になります。