はじめての PHP フレームワーク

PHP初心者がすぐ現場で活躍するための

CodeIgniter 3 超入門

榊原昌彦
一般社団法人リレーションデザイン研究所

宮川貴子
合同会社 nekomori

フレームワークを使って
「共通の言葉」を手に入れよう！

第１版

目次

目次 2

はじめに 4

1. 開発を加速するPHPフレームワーク 6
   コードをきれいに書くことは将来の自分を救う
   PHPフレームワークをみんなが使う理由

2. CodeIgniterを実行する環境構築 10
   MAMPのインストール
   Composerのインストール
   CodeIgniterのインストール

3. CodeIgniterの「MVCモデル」 24
   View（見た目）を独立させることで分業を可能に
   データベースとやりとりをするModel
   まとめ役のController

4. チュートリアル 「Welcomeページを書き換える」 27
   URL構造を読み解く
   Welcomeページを書き換える

コラム「IDEを使って開発を加速しよう」 33
   IDEのメリット
   オススメIDE

5. チュートリアル「新規ページをつくってみよう」 37
   チュートリアルファイルをダウンロード
   新規ページを作ってみよう

コラム「PHP標準コーディング規約」 45
   PHP標準コーディング規約

6. チュートリアル「HelperとLibraryで機能拡張」 47

Helper
Library
7. チュートリアル「タスクリストを作ってみよう」 53
データベースを用意しよう
データベースの接続設定
Modelを用意してデータ呼び出し
データを新規登録しよう
8. コードリファクタリングをしよう 71
脱・Fat Controller
コードの再利用性を高めよう
9. チュートリアル「リファクタリング」 79
設定をユーザ設定にまとめよう
Viewを分割して使いまわせるように
処理をユーザHelper/ユーザLibraryで共通化
コラム「バージョン管理をしよう」 91
Gitでできること
Gitを利用したサービス
Gitを学ぶためのおすすめ書籍
コラム「もっと生産性を高めるために」 94
テストってどういうもの？
CodeIgniterでテストを書く
品質と安心感を得るために
終わりに　ーコミュニティに参加しよう 97

はじめに

あなたにとって、プログラミングはどんな存在でしょうか。
「職場で指示されたから覚えざる得ないもの」「難しそう」「お金が稼げそう」などいろいろなイメージがあると思います。プログラミングするきっかけはひとそれぞれですし、どういったアプリケーションをつくりだすのかも千差万別です。しかし、ひとつだけ共通しているものがあります。それは、アルゴリズムをつくりだすということです。

- web上から受注する →（アルゴリズム）→ 卸先に発注が自動的に行われる
- 売上伝票を登録する →（アルゴリズム）→ 日計表が自動的に更新される

これらは、作業と結果の間に「アルゴリズム」が存在していて、いろいろな処理を自動化することによって結果をショートカットして得ることができます。ですので、プログラミングできるようになると便利なアプリケーションを作ることができますし、その便利なアプリケーションに対価をもらうことだってできます。

あなたがもしプログラミングをはじめたばかりでしたら、入門書には「Hello World!!」をただ出力したり、延々と四則演算をする仕組みをつくったり、1と10が等しいか等しくないかといった条件分岐をすることであったりが書かれていて、「これでどうやってアプリケーションをつくるところまで辿りつけるか」がなかなか把握できないでいると思います。
本書は、PHPフレームワーク「CodeIgniter」をテーマとして、すでに学んだPHPの基礎でどのようにアプリケーションをつくるとっかかりになることを目的としています。ですので、取り扱うのは、CodeIgniterについてだけではありません。イマドキの開発環境はどうなっているのか、どうやったら開発を加速することができるかという実務に関する部分を広く取り扱っていきます。

個人が部屋を貸し借りすることができるサービス「Airbnb」（ https://www.airbnb.jp/ ）は宿泊産業を変え、オリジナル料理レシピの投稿・検索サービス「クックパッド」（ https://cookpad.com ）は、様々な有名レシピを生み出しました。

　アプリケーションをつくるということは、仕組みをつくることでもあります。他の人がつくった仕組みをつくるだけではなく、仕組みをつくる側にまわって、世の中を変えていきましょう！

1. 開発を加速するPHPフレームワーク

コードをきれいに書くことは将来の自分を救う

最初に「（プログラミング）コードの良し悪し」について考えてみましょう。
誰にも書けないようなコードが良いコードでしょうか。他の人が数十行書くコードを複雑怪奇だけどわずか数行にまとめたら良いコードでしょうか。

そんなことは全くありません。もちろん、凄腕プログラマに分類される人でそういうコードを書く人もいないわけではありませんが、そのコードを「いいコード」と評す人は極小数です。

本書では、いいコードは以下の3点を満たすコードであると考えます。

- 読みやすいコード
- 変更しやすいコード
- セキュリティリスクの小さいコード

コードは他人に読ませるものです。書いた直後はコードを書いた当事者である自分自身はそのコードを一目で理解できますが、半年後、1年後になるとすっかり忘れてしまっています。しかし、その忘れた頃に、開発時には想定しなかった機能変更がやってくることはざらにあり、そういう時は一からコードを読みなおすことになります。
ですので、プログラマは、コメントを書いたり、できるだけ単純化した処理を行ったりして、読みやすく・変更しやすいコードを目指します。

また、セキュリティリスクには細心の注意を払う必要があります。近年、個人情報漏洩で脆弱なシステムの責任をソフトメーカーが負うべきとの判決がでました（東京地方裁判所判決／平成２３年（ワ）第３２０６０号）。サービスを提供する以上、そういったセキュリティリスクをできるだけ排しておくのは、プログラマの義務として一般的に認識されています。

「個人情報を扱っていない」「無料サービスだし」といっても、トラブルが起きる度に、問題が起きる度に連絡がいくのは開発者であるあなたですので、セキュリティリスクをどのように排するかは大きなテーマとなります。

PHPフレームワークをみんなが使う理由

PHPを書くための土台をみんなで共有しよう

「良いコードは良いもの」だといっても、実現するのは容易ではありません。

ファイル名の命名規則の決め方、データベースとの接続方法、セキュリティ対策は何から手をつけるか、保守性をどう高めるかなど考えだすと、きりがありません。

そこで、有志が集まって「PHPを書くための土台」をつくり、多くの人はそれを利用してPHPを書き進めています。この土台を「PHPフレームワーク」といいます。本書で取り扱うCodeIgniterは、ブリティッシュコロンビア工科大学がPHPを用いたアプリケーション開発を加速するために所有・開発をしているPHPを書くための土台です。

フレームワークでは、「フォルダ構成」「ファイル名の命名規則」などが決まっており、そのルールを外れるとエラーがでるようになっています。だからこそ、途中で参画したプロジェクトでも、採用されているフレームワークさえ知っていれば「この処理はここに書いているんだろうな」「この呼び出し先はここか」ということが一目瞭然となり、「読みやすいコード」「変更しやすいコード」の実現に貢献します。

過去の資産がまとめられている

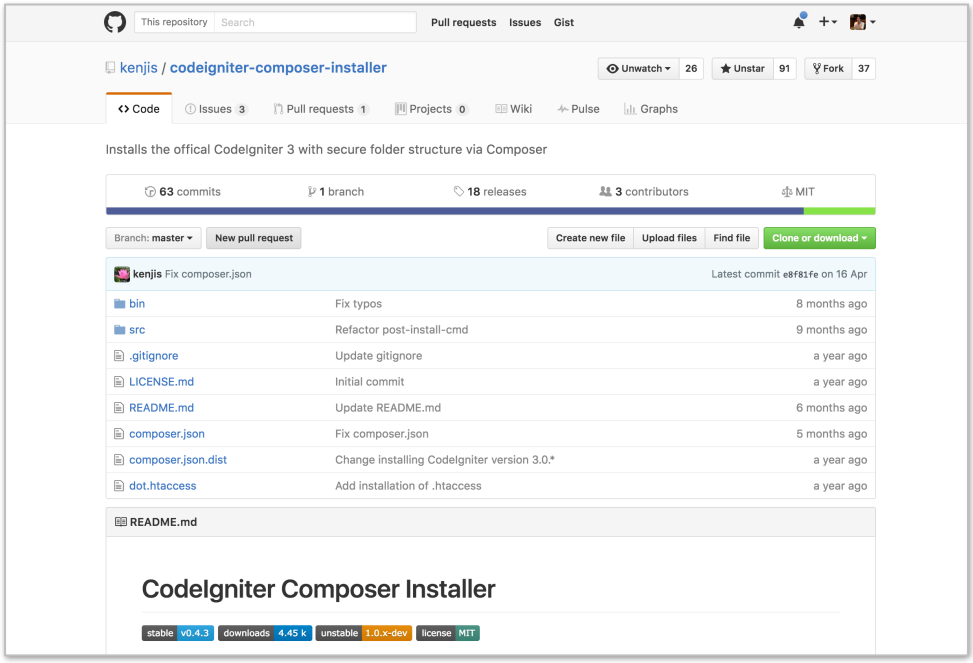
フレームワークには、その開発元と開発に参加しているメンバーの過去の資産がまとめられています。例えば、入力された値が正しいかどうかを判定する「バリデーション」機能がありますが、これを一からPHPで書くと、多くのコードを書かなければなりませんし、そのコードの可読性を高めようとすると至難の業です。

しかし、CodeIgniterには、バリデーションを実現するための機能も搭載されており、必須入力か、数字が入力されたかなどの「バリデーションルール」を一行で書くことができ、それらをまとめて実行するという大変便利な機能があります。PHPで画像を編集する機能、メール送信を簡単にできる機能など様々なものがあり、あなたのアプリケーション開発にとても役立ちます。

オープンな資産も使えるようになる

フレームワークを使うということは「同じ土台をつかってる他の人がいる」ということです。だからこそ、あなたが今からつくろうと思っているアプリケーションの一部の機能はすでに他の人がつくっていて、Webで無料公開してるということもありえます。

例えば、本書で利用しているCodeIgniterインストールの仕組みは、日本CodeIgniterユーザ会のkenjisさんが作成して、以下のURL公開しているものを利用しています。



https://github.com/kenjis/codeigniter-composer-installer

一から仕組みをつくらなくても、他の人がつくったものを使いまわしてその仕組みができるなら、ぜひとも楽したいですよね。フレームワークを使うということは、こうやって他の人の資産を使いやすい環境をつくることでもあります。ですから、現場で活躍しようと思ったら、フレームワークという武器をぜひとも携えてもらえればと思います。

まず、はじめてみよう

PHPフレームワークは、CodeIgniterだけではなく、PHP開発元のZend社がつくってる「ZendFramework」や、大規模の開発で使われる「Symfony」をはじめとして、10

を超えるPHPフレームワークが世の中にはあります。フレームワーク毎に「できるだけ早い処理速度を実現する」「最新のPHP技術を先取りする」「APIを簡単につくれるようにしたフレームワーク」など様々な性格があります。

CodeIgniterは「処理速度がひたすら早い」「初期設定を行わずにとりあえず動かすことができる」「学習コストが少ない」などの特徴があって、フレームワークをまず使い始めてみるというステップに最適です。導入した成果を早いスパンであげることができるので、おすすめのフレームワークです。

選択肢はCodeIgniterただひとつというわけではありません。しかし、「フレームワークで開発する」体験をしてみるために、まずははじめてみましょう。

2. CodeIgniterを実行する環境構築

CodeIgniterを学ぶために、まずはCodeIgniterが動く環境をつくっていきます。CodeIgniterを動かすためには、Apache、PHP、MySQLがインストールされている必要があります。[1]

本書では「MAMP」を利用してローカル環境で動かしていきます。これを使うと、CodeIgniterを実行する環境を一発で作ることができます。

MAMPのインストール

MAMPのダウンロードURL（https://www.mamp.info/en/downloads/）にアクセスします。中段に、自分のOSを選択するボタンがありますので、MacのPCの場合は「Mac OS X」、Windowsの場合は「Windows」を選択します。そして、その下にある「Download」をクリックするとダウンロードがはじまります。

[1] 厳密には、Apache,PHPのみで利用できますのでビルドインサーバーでも起動可能です

Mac OSの場合

1. ダウンロードファイルをクリックするとインストーラーが立ち上がります。「続ける」をクリックします。
2. このインストーラーによって、『アプリケーションフォルダに生成される「MAMP」と「MAMP PRO」というフォルダを削除したり名前の変更をしないように』と案内がでます。「続ける」をクリックします。

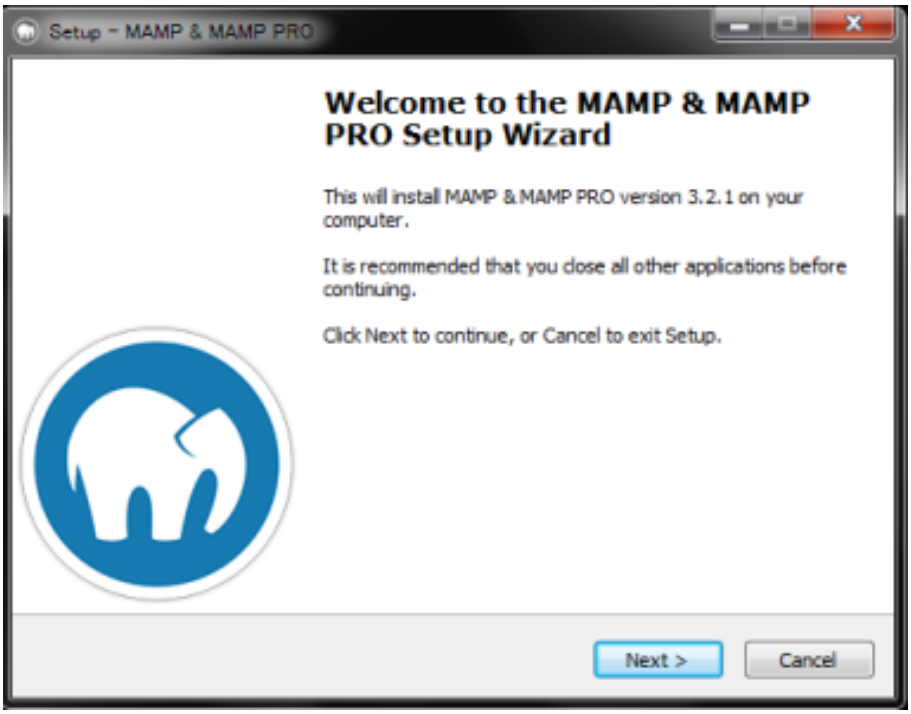
3. 使用許諾契約の画面がでてきます。確認して、「続ける」をクリックします。

4. インストール先の選択を行います。デフォルトの「このコンピューターのすべてのユーザ用にインストール」のまま、「続ける」をクリックします。

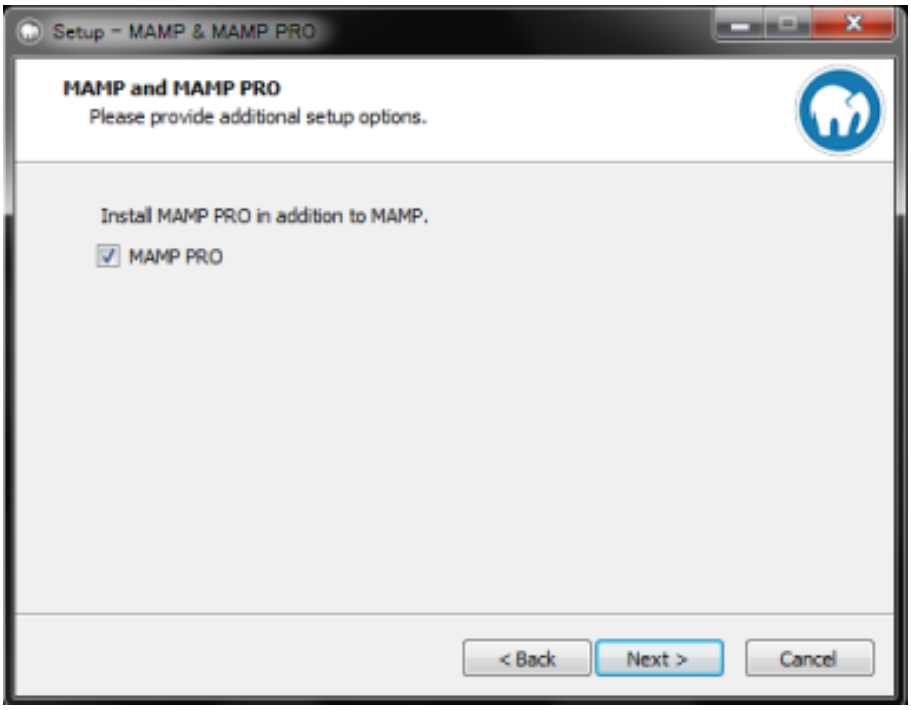
5. “Macintosh HD”に標準インストールを行います。そのまま「インストール」をクリックします。設定によっては、Mac本体がインストールを進めるためにパスワードを要求することがありますが、その場合自分が設定したパスワードを入力します。
6. すると、インストールが開始されます。
7. 「インストールが完了しました」と表示されたらインストールの完了です。「閉じる」をクリックします。

Windowsの場合

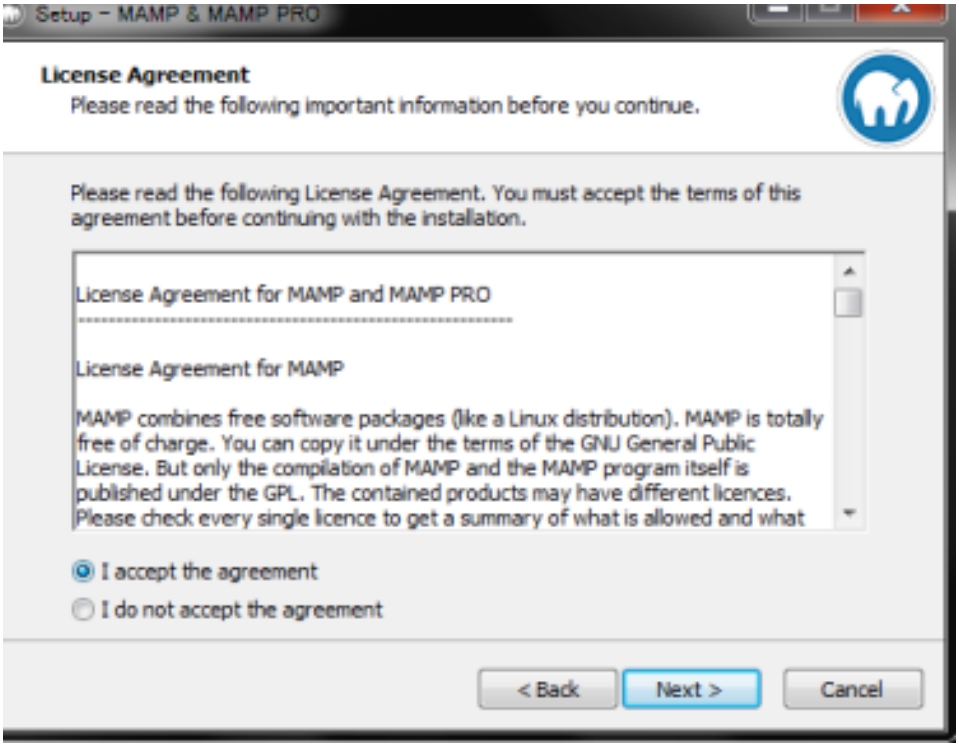
1. ダウンロードファイルをクリックするとインストーラーが立ち上がります。Nextをクリックします。
2. Welcome to the MAMP Setup Wizard：インストーラーが開始します。Nextをクリックします。



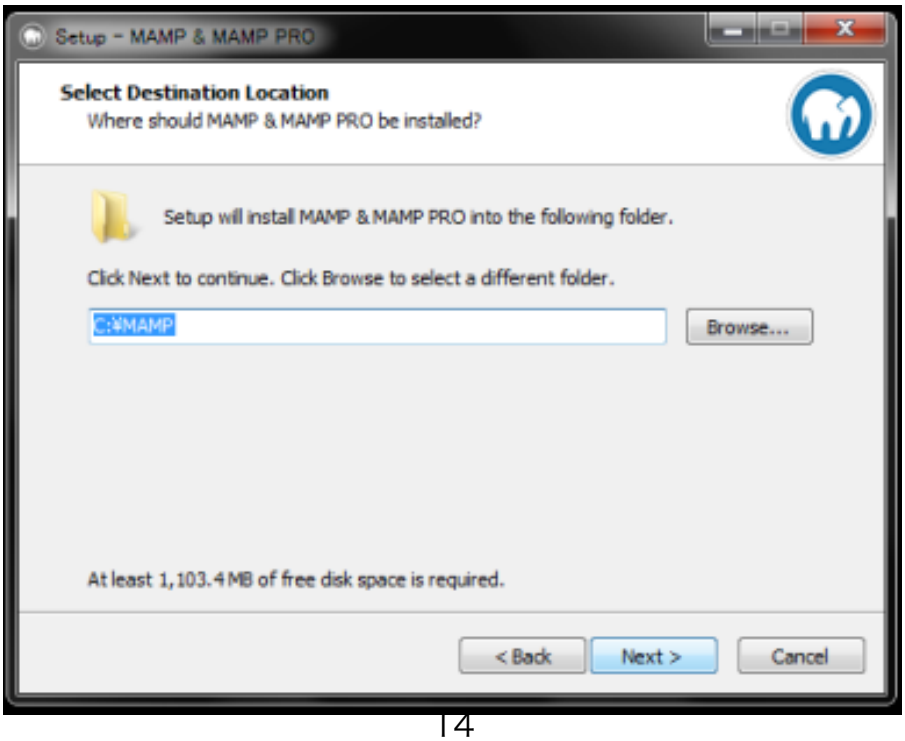
3. MAMP PROはインストールしないのでチェックを外します。



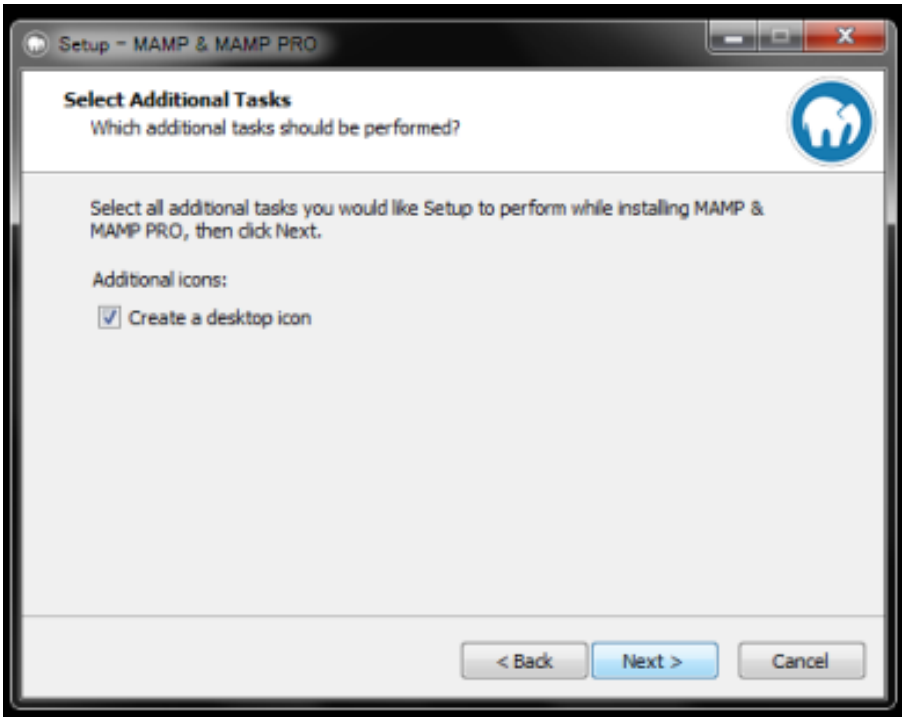
4. 次は利用許諾です。利用許諾を読み、許諾します。
agreementにチェックを入れて許諾します。NEXTをクリックします。



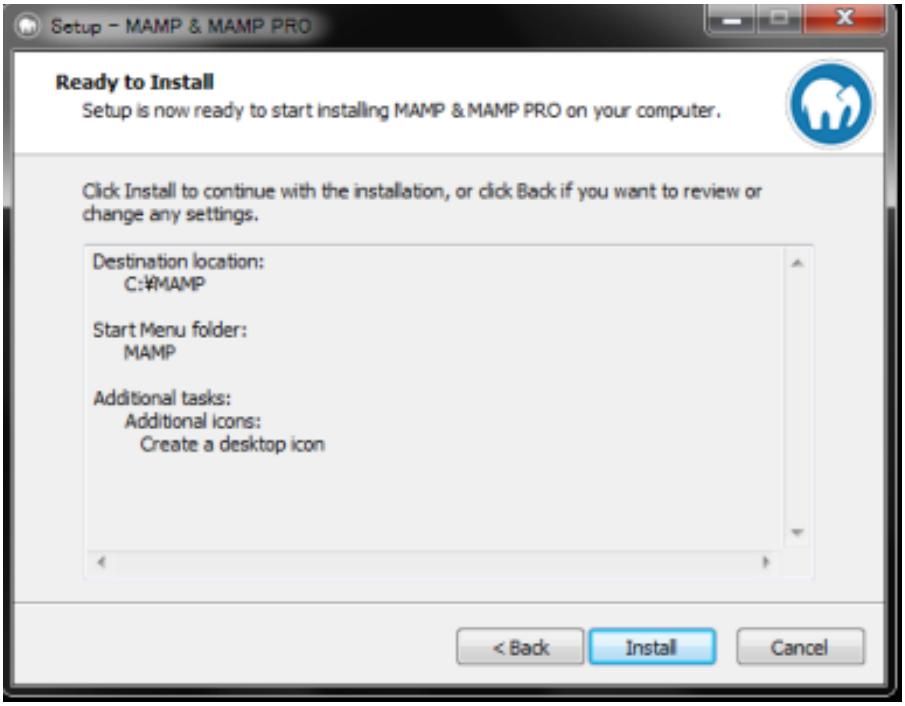
5. インストールするパスを確認
C:¥MAMPにインストールする指定になっていることを確認します。
(修正の必要はありません）NEXTをクリックします。



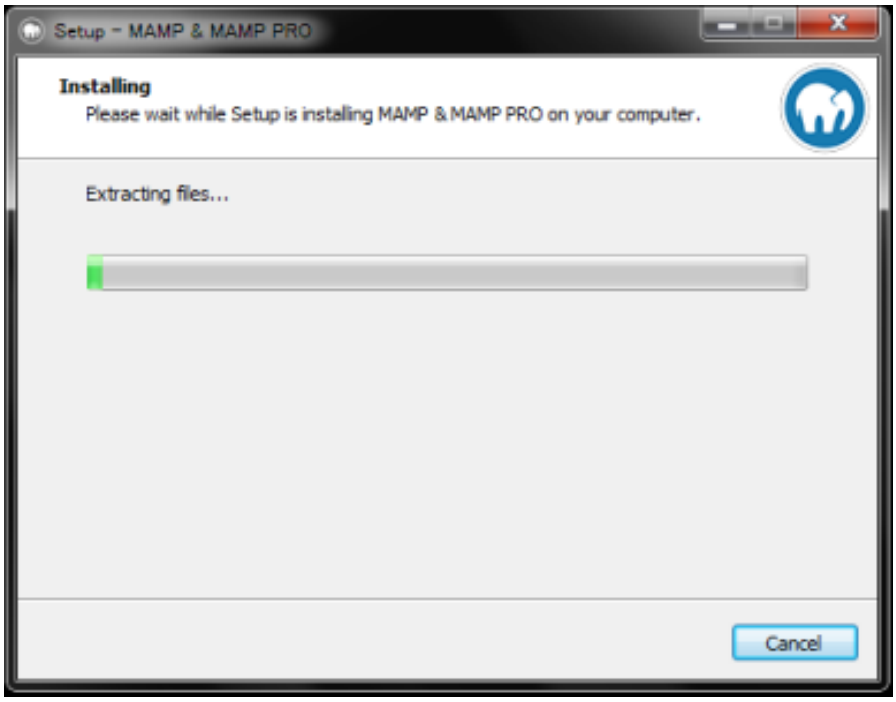
6. デスクトップにアイコンを作成するかどうかの設定をします。チェックを入れるとアイコンが作成されます。NEXTをクリックします。



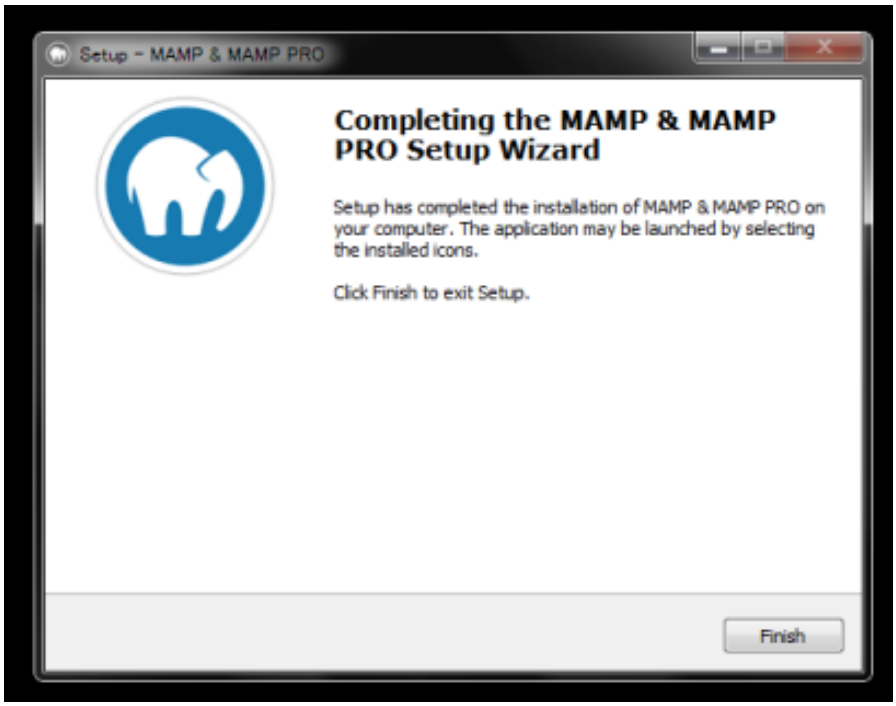
7. ここまで設定した内容を確認します。
問題なければ、NEXTをクリックします。



8. インストールが始まります。インストールが終わるまで待ちましょう。



9. インストールが完了しました。Finishボタンをクリックし、完了します。



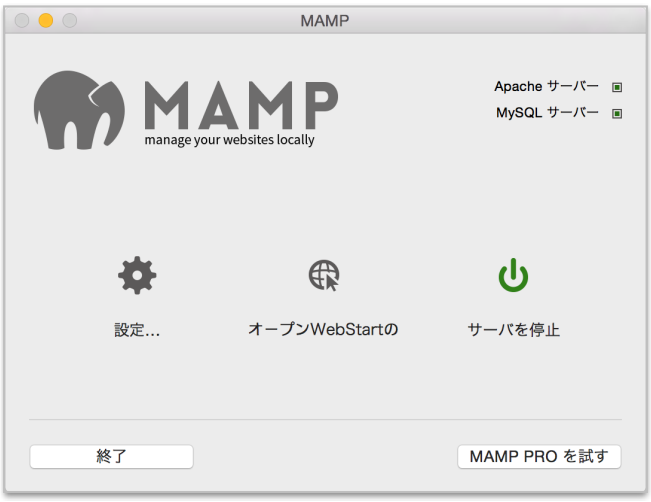
インストールがうまくいかない方へ

うまくMAMPがインストールできないという方は、インストールの流れを詳しいキャプチャ画像つきで以下のスライドで紹介されていますのでご参考ください。Windows版は、「.NET」のインストールを求められることがありますので、そちらについても以下で解説されています。

https://goo.gl/c0dddP

起動確認

インストールしたMAMPを起動してみてください。無事にインストールが完了されていると、MAMPのアプリケーションウインドウが立ち上がります。



このウインドウは、MAMPを利用する間はこのウインドウは立ち上げたままにしておいて下さい。最後にメニュー真ん中の「オープンWebStart」をクリックして、以下のページが起動できたら、インストール成功です

なお、CodeIgniterはPHP5.3系をサポートしておりますが、本書ではPHP5.4以上のPHPを対象とします。

Composerのインストール

次にCodeIgniterをインストールするためのツール「Composer」をインストールします。Composerは、PHPパッケージのバージョン管理と「このパッケージを使うためにはこのパッケージがたりないよ？」という依存管理をすることができる便利なツールで、今ではほとんどの現場でパッケージインストールに利用されるようになりました（2016年現在）。

Mac OSの場合

Mac OSでは、インストーラーを使わずに、ターミナルからインストールします。ターミナルを起動して、以下を入力して下さい。

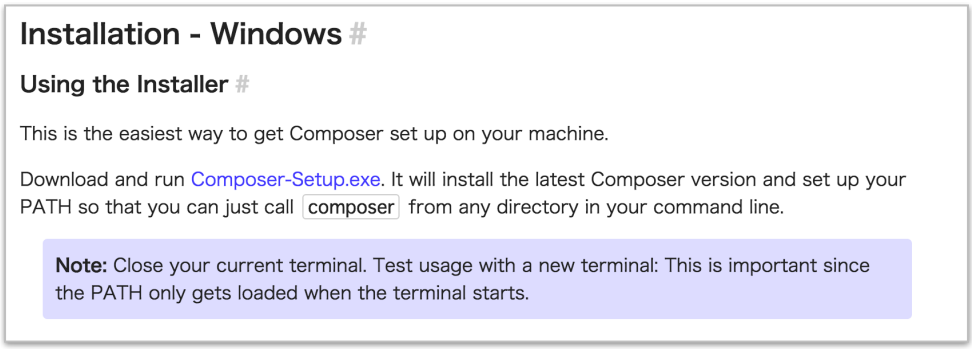
```
$ cd /Applications/MAMP/htdocs
$ php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
$ php composer-setup.php
$ php -r "unlink('composer-setup.php');"
```

一行目に、compose-installerをダウンロードし、2行目、3行目でインストールしております。[2]これでcomposerのインストールは完了です。

[2] 公式では、ダウンロードしたファイルが妥当かどうかをチェックするために、SHA384を用いてデータチェックを行っております。必須ではありませんが、利用する場合は https://getcomposer.org/download/ をご確認ください。

Windowsの場合

Windows版のダウンロードURL（ https://getcomposer.org/doc/00-intro.md#installation-windows ）に移動します。そして、「Composer-Setup.exe」をクリックして、ダウンロードして下さい。



Installation - Windows #

Using the Installer #

This is the easiest way to get Composer set up on your machine.

Download and run Composer-Setup.exe. It will install the latest Composer version and set up your PATH so that you can just call `composer` from any directory in your command line.

Note: Close your current terminal. Test usage with a new terminal: This is important since the PATH only gets loaded when the terminal starts.

1. Composer Setup: Next
2. Select Components: そのままでNext
3. Setting Check: そのままでNext
4. Ready to Install: インストールをクリック
5. Information：そのままでNext
6. Completing Composer Setup：Finish

インストール確認

正しくインストールされてるか確認するために、コマンドプロンプトを起動して「composer」と打って確認しましょう。

```
$ ./composer.phar --version
```

下のような画面が出てくれば正常にインストールされています。

```
  ______
 / ____/___  ____ ___  ____  ____  ________  _____
/ /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/

Composer version 1.2-dev (75074382de9d7b61c041e494aee123477ae01853)
2016-06-25 14:50:18
```

CodeIgniterのインストール

さて、ここまででCodeIgniterを動かす環境づくりは完了しました。それでは、いよいよCodeIgniterをインストールします。
まず、ターミナル（Windowsは「コマンドプロンプト」）を開いて、MAMPの公開フォルダに移動します。Macの場合は

```
$ cd /Applications/MAMP/htdocs
```

と入力。WIndowsの場合は

```
$ cd C:MAMP/htdocs
```

と入力して、インストール場所へと移動します。
そして、次に以下のように入力してEnterを押します。

```
$ ./composer.phar create-project kenjis/codeigniter-composer-installer codeigniter
```

コマンドの意味は『「composer」を使って「create-project」する。projectは「kenjis/codeigniter-composer-installer」からデータをひっぱってきて、「codeigniter」という名前のフォルダにプロジェクトを格納する』です。
しばらくすると、以下の案内が表示されてインストールが完了します。

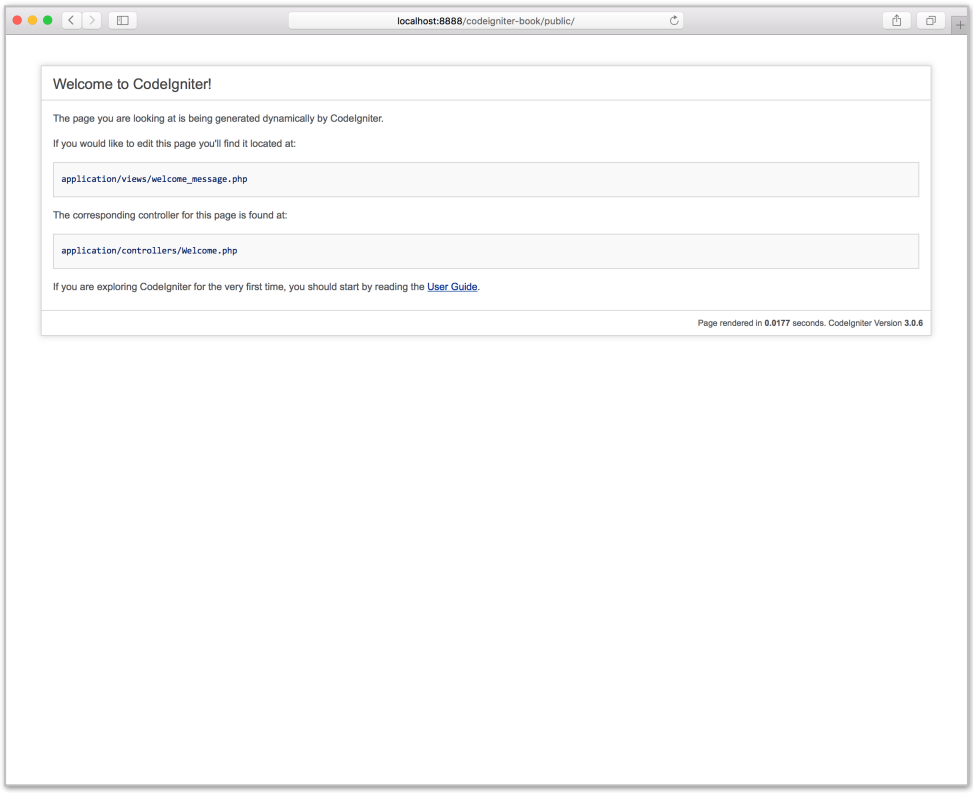
```
=====================================================
`public/.htaccess` was installed. If you don't need it, please remove it.
If you want to install translations for system messages or some third party li-
braries,
$ cd <codeigniter_project_folder>
$ php bin/install.php
Above command will show help message.
See <https://github.com/kenjis/codeigniter-composer-installer> for details
=====================================================
```

起動確認

MAMPの「スタートページを開く（Open start page）」をクリックして遷移したURLを次のように変更して下さい（ポートを変更している時は「8888」の数字を変更ください）。

```
http://localhost:8888/codeigniter/public
```

Welcomeページが表示されたら、CodeIgniterのインストール完了です！



3. CodeIgniterの「MVCモデル」

コードを書き始める前に、CodeIgniterが採用している「MVCモデル」というコードの書き方について説明します。

CodeIgniterでは、処理を大きく3つの役割に分類してコードを書く場所を指定しています。その3つとは、Model（モデル）、View（ビュー）、Controller（コントローラ）で、頭文字をとって「MVCモデル」[3]と言われています。
本章では、それぞれがどういった意味と役割をもっているかを解説します。その気になれば、1つのファイルだけで処理できてしまうところを、分割するのにはちゃんと意味はあるのです。

View（見た目）を独立させることで分業を可能に

Viewが指しているのは、HTML, CSS, JavaScriptなどで書かれているテンプレート（見た目/フロントエンドと呼ばれます）です。これを独立せずに書いてしまうと、デザインを変更する度にアルゴリズムを書いたファイルを更新することになってしまいます。

例えば、ブログシステムを考えてください。仮にテンプレートがメイン処理にかかれているとそれを修正するためにはコードをしっかり読んで、どこからがメイン処理で、どこからがテンプレート部分なのかを理解する必要があります。そして、間違ってメイン処理に手をいれてしまうと、一部の見た目を変えたいだけなのに、そのブログ全体が動かなくなってしまう危険性もあります。

ですので、Viewはアルゴリズムから独立させ、「application/views」というフォルダに配置することが決められています。そして、そのViewを使う時だけ、アルゴリズムからそのファイルを呼び出すのです。
このことは分業にもメリットがあり、デザイナーやフロントエンドエンジニアは、指示されたViewフォルダの中だけで作業を行うことができます。

[3] 本書では、Push-MVCを解説しています

データベースとやりとりをするModel

Modelは、データを保存したり、保存したデータを取り出して扱いやすい形に加工する役割を持っています。データの出入力プログラム（MySQLなどのデータベースなど）はとても専門性が高くなっていくので、これもView同様に独立させることが一般的です。このことによって、データベースとの連携は専門家に任せることが可能になります。

また、別々のページだけど同じデータをデータベースから取得する必要がでてくることはよくあります。この時にModelとして独立させていることによって、同じ処理を複数回書くことなく、1つのModelを共有することができます。これを「コードの再利用性」といいます。
Modelは「application/models」の中に書くことが決められています。

まとめ役のController

Controllerの大きな役割は、「URLを決めること」と「ViewとModelのまとめ役」です。

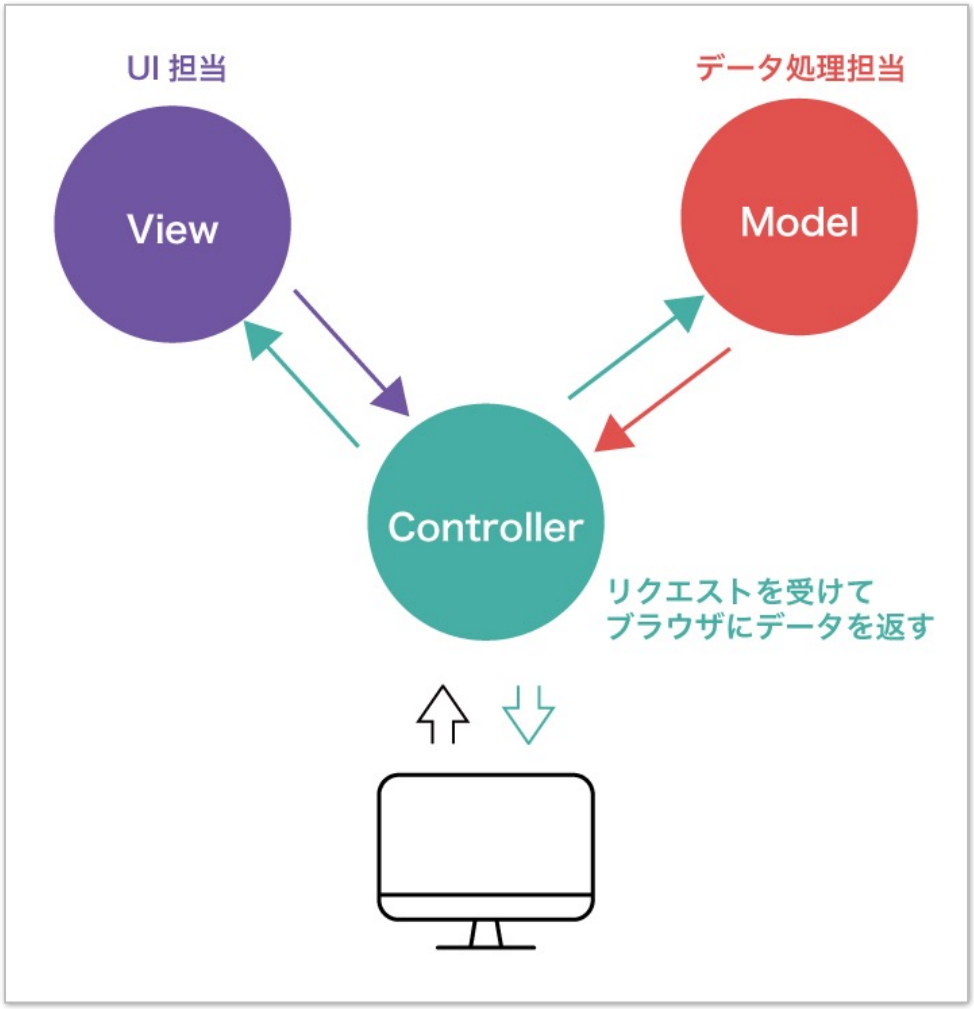
URLを決めるController

CodeIgniterの公開フォルダにあるPHPファイルは「index.php」ひとつだけです。ここに、company.phpとかuser.phpとか、そういった個別ファイルは存在しません。URLをみると階層構造にアクセスしてるように見えますが、それは見せかけです。
例えば、「http://hoge.jp/user/219」があったとすると、これは実は「http://hoge.jp/index.php/user/219」いうURLで、「index.php/user/219」から「user」と「219」という引数を読み込んで、該当するControllerを呼びだしにいっています。
決して、「user」というフォルダに、219というファイルがあるわけではないのです。

CodeIgniterは、特別にルーティングファイルで設定していない場合は、URLからControllerが呼び出され、該当するControllerがなかった場合は「404 page not found」を返す仕組みとなっています。

View/Modelのまとめ役もこなします

指定されたURLにアクセスすると、まずControllerが呼び出されます。そして、Controllerが、どのデータ（Model）を呼び出して、どういった見た目（View）で表示するかを決める役割を持っています。



4. チュートリアル 「Welcomeページを書き換える」

実際にCodeIgniterをさわっていきましょう。CodeIgniterの仕組みに慣れるために、まずは最初に表示されるWelcomeページがどういう仕組みで表示されているかを見ていきます。

URL構造を読み解く

ルーティングを理解する

あなたが表示している「http://localhost:8888/codeigniter/public」は、詳細に表示すると次のURLとなります。

```
http://localhost:8888/codeigniter/public/index.php/welcome/index
```

試しにこちらのURLでもアクセスして下さい。先ほどと同様のWelcomeページが表示されることが確認できます。こうやって、URL構造を省略したり、URLと連動させて指定のControllerを呼び出して表示することを「ルーティング」といいます。
それでは、このURL構造が意味しているものを読み解いていきます。

このURLは4つの意味に分けることができます。

- http://localhost:8888/codeigniter
- /public/index.php
- /welcome
- /index

「http://localhost:8888/codeigniter」はCodeIgniterをインストールしたフォルダを意味します。実際にMAMPの公開フォルダとなる「htdocs」をみると、そこに

「codeigniter」というフォルダがあってそこにCodeIgniterがインストールしているのを確認できると思います。

次に、「/public/index.php」は実際にアクセスしているファイルパスを意味します。CodeIgniterの公開フォルダが「public」となっており、その中にある「index.php」にアクセスしていることを意味します。けれど、「index.php」というファイル名を残しておくのは見かけ上よろしくないということで、「public/.htaccess」によってその文字列を隠しています。

「public/.htaccess」の中身は次の通りで、簡単にいうと「index.php」という文字列を隠すことができるように設定されています。

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d

RewriteRule ^(.*)$ index.php/$1 [L]
```

「/welcome」ですが、これはWelcome.phpというControllerを呼び出すことを意味します。Controllerの置き場所である「application/controllers」の中をみてみると、実際に「Welcome.php」というファイルがあることを確認できます。

これがトップページに表示されるのは「application/config/routes.php」によって「WelcomeというControllerをトップページにしますよ」と設定されているからです。routes.phpの設定はこのようにされています。

```
$route['default_controller'] = 'welcome';
```

最後に、「/index」は、そのWelcome.phpというControllerの中に書かれている「function index()」というメソッドを呼び出しますよ、という意味となります。メソッド名が「index」の時のみ省略していいルールがあり、省略されています。

URLの意味まとめ

ということで、URLの意味は以下の通りになります。

http://localhost:8888/codeigniter	CodeIgniterのインストールフォルダへのアクセス
/public/index.php	実際にアクセスされているファイル
/welcome	Controllerの名前
/index	Controllerのメソッド名

Welcomeページを書き換える

Controllerを書き換える

さて、Welcomeページは、WelcomeというContollerによって生成されていることがわかりました。そして、Controllerのファイル名は、ControllerのClass名を使うことが決まっています

そこで、controllers/Welcome.phpの中身を確認します。

```
class Welcome extends CI_Controller {

	/**
	 * Index Page for this controller.
	 *
	 * Maps to the following URL
	 *		http://example.com/index.php/welcome
	 *	- or -
	 *		http://example.com/index.php/welcome/index
	 *	- or -
	 * Since this controller is set as the default controller in
	 * config/routes.php, it's displayed at http://example.com/
	 *
	 * So any other public methods not prefixed with an underscore will
	 * map to /index.php/welcome/<method_name>
	 * @see http://codeigniter.com/user_guide/general/urls.html
	 */
	public function index()
	{
		$this->load->view('welcome_message');
	}
}
```

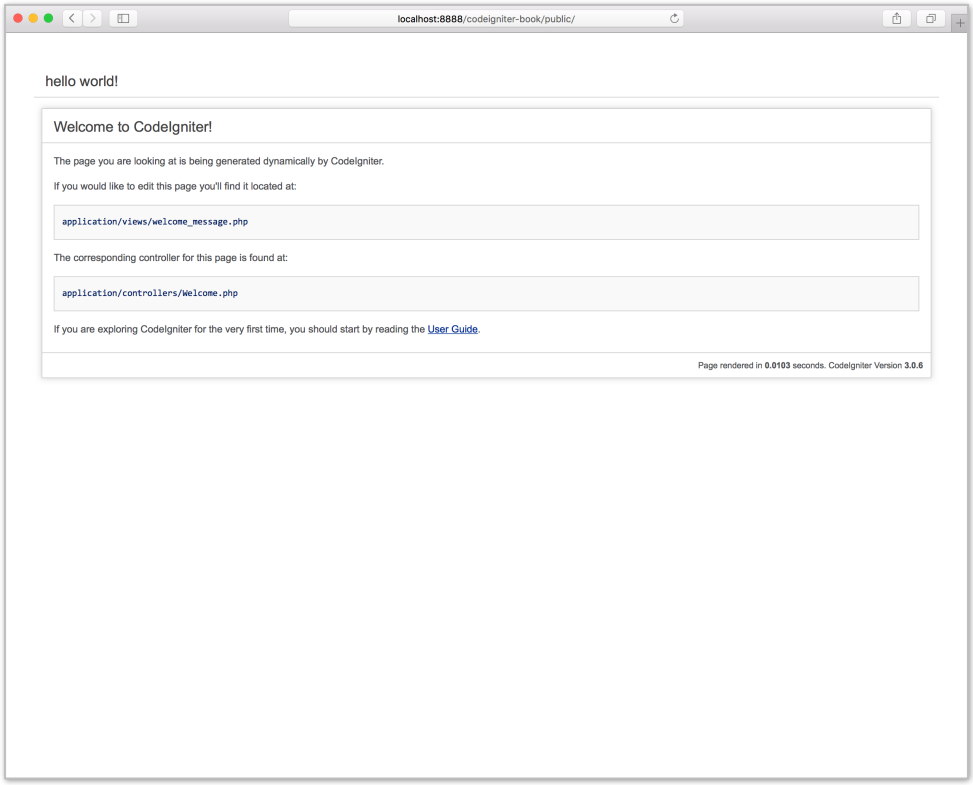
1行目に、「class Welcome」とあります。これは、Controllerの名前を「Welcome」にしますという宣言です。ついで「extends CI_Controller」とありますが、これは「CI_Controller」という機能を継承していることを示しており、Controllerを書く時のルールです。

そして、そのControllerの中で、「public function index()」としてメソッドが定義されています。WelcomeというControllerの中にある、indexというメソッドなのでURLは「welcome/index」となるわけです。

試しに、indexの1行目に以下のように追記してみましょう。「hello world!」という文字列を表示させてみます（変更点は太字）

```
class Welcome extends CI_Controller {
// 中略
    public function index()
    {
✤       echo “<h1>hello world!</h1>”;
        $this->load->view('welcome_message');
    }
```

これでブラウザを更新すると、左上にhello world!の文字が表示されたと思います。



Viewを書き換える

前章で確認した通り、ブラウザに表示させる見た目部分はViewに記述します。CodeIgniterでは、Viewの呼び出しは次のメソッドを使うことが決まっています。

```
$this->load->view()
```

「welcome/index」では「welcome_message」というViewを呼び出しています。

```
$this->load->view('welcome_message')
```

Viewは、「application/views/」というフォルダが格納場所となっており、呼び出しはファイル名ですることが決まっています。実際に確認してみると「welcome_message.php」がありますので、その中身を確認してみます。

試しに、「views/welcome_message.php」の中身を空にして、ブラウザを更新してみてください
ブラウザの画面がhello world!のみの表示となり、「views/welcome_message.php」の内容が消えました。慣れないうちは、「自分が修正しようとしているのは、本当にこのファイルであってるのか」をこまめに確認するために、文字列を出力してみたり、一回全部消して画面に反映されるかを確認することをおすすめします。

さて、これでWelcomeページはどこの何を表示してきてるかがわかりましたね！

コラム「IDEを使って開発を加速しよう」

皆さんは何を使ってコードを書いていますか？？

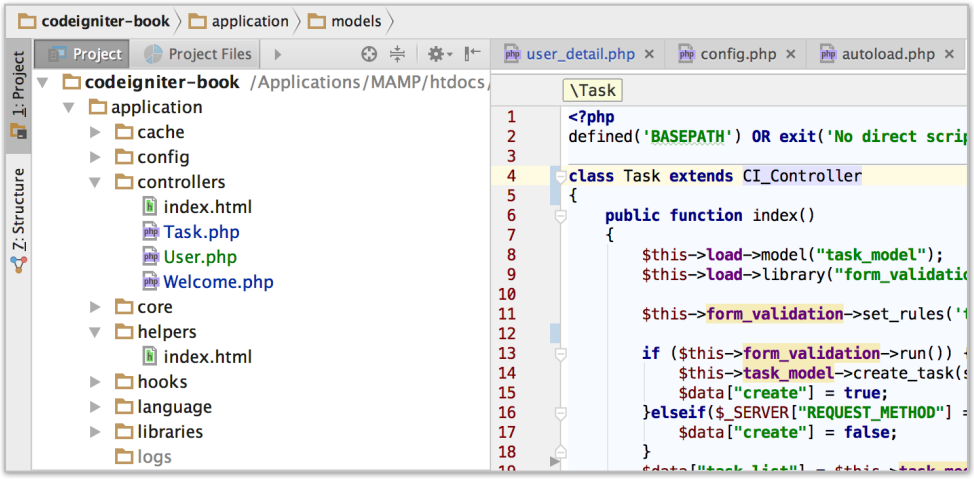
PHP入門書の多くは、MacだとmiエディタWindowsではサクラエディタやTera-Padをコードを書くツールとしてご紹介してることが多いです。実際、これらで不足があるかというと、意外と小規模なWebアプリはつくることが可能ですが、開発速度を加速したいならIDE（総合開発環境）と呼ばれるソフトウェアを使うのがおすすめです。

IDEのメリット

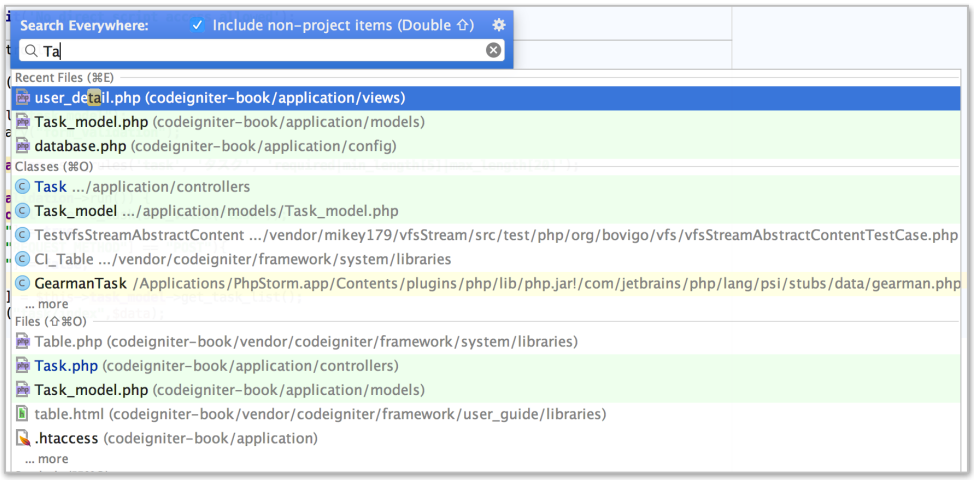
使い始めたらもう離れられない！と巷で評判のIDEですが、コードを書き始めの時はなかなか実感することができません。そこで、コードの書き始めでも、これはわかりやすい！というメリットを2点だけご紹介します。

プロジェクトファイル管理

コードを書いていると、ファイル数は数十、数百へと膨れていきます。これはソースコードを1ファイルあたり数万行とか書かないための止むを得ない選択ではありますが、フォルダを越えて、あっちに書いたコードをこっちで呼び出してということをしていたらそれだけで一大作業です。IDEはプロジェクトファイルをツリーで表示してくれることによって、その作業を効率化します。



また、ファイル検索の機能もすぐれていて、ファイル名がわかっている時はもちろんのこと、どっかのファイルに書いた「Ta」からはじまるClassってどこにあったっけ、といった断片的な情報からも検索をかけることが可能です。



コードの自動整形・エラー検知

テキストエディタを使っていると、if文の中は空白を2つ入れて、その中のループ処理は4つ入れて、とインテントは自分で全て管理する必要がありました。IDEはある程度それを全て自動的に処理してくれます。

また、エラーも自動的に検知してくれます。例えば以下だと、１．同じメソッドが2つあるので、2つ目に赤下線でエラーを表示してくれている、２．arry_codeというのはおそらくarray_codeのタイプミスなので緑下線で誤字の提案をしてくれています。

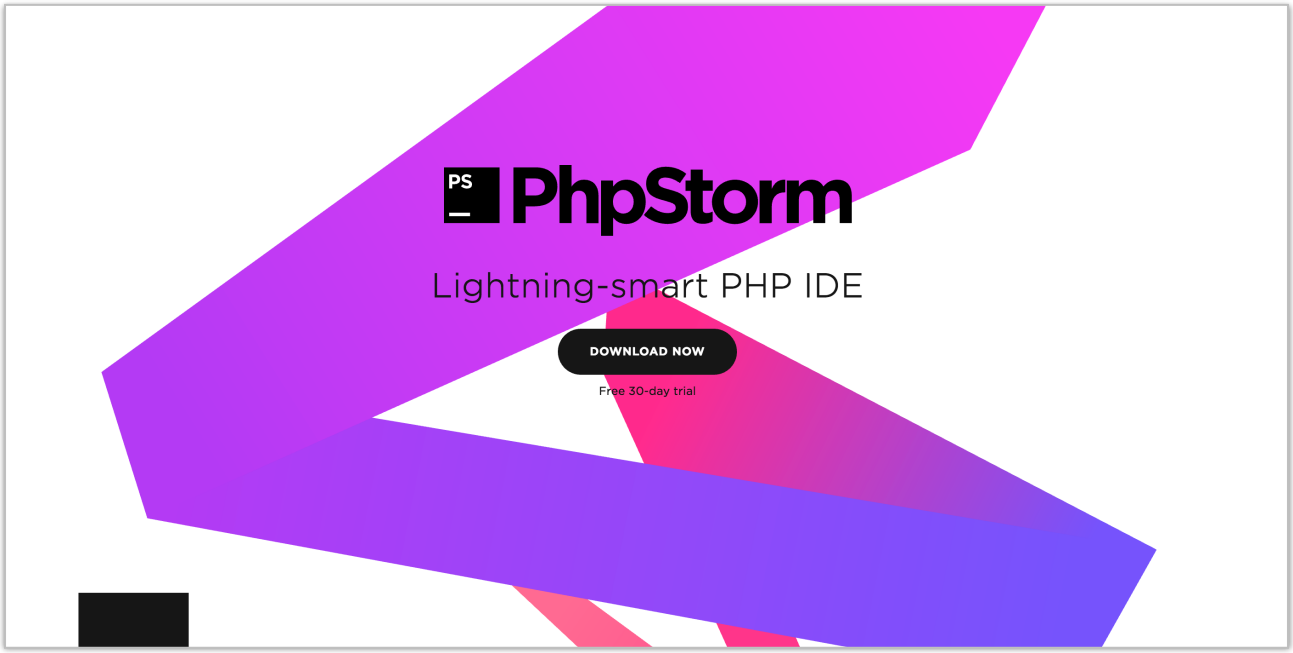
```
public function detail()
{
    $data = [];
    $data["arry_code"] = [];
}

public function detail()
{

}
```

こういったケアレスミスは自動的にIDEが検知してくれますので、コードを書いてて「あれ、なぜか動かなくなった」というときにすばやく気づくことが可能になります。

オススメIDE

IDEといっても、様々な種類があります。その中で、PHPを書くのに便利なIDEを2つご紹介します。

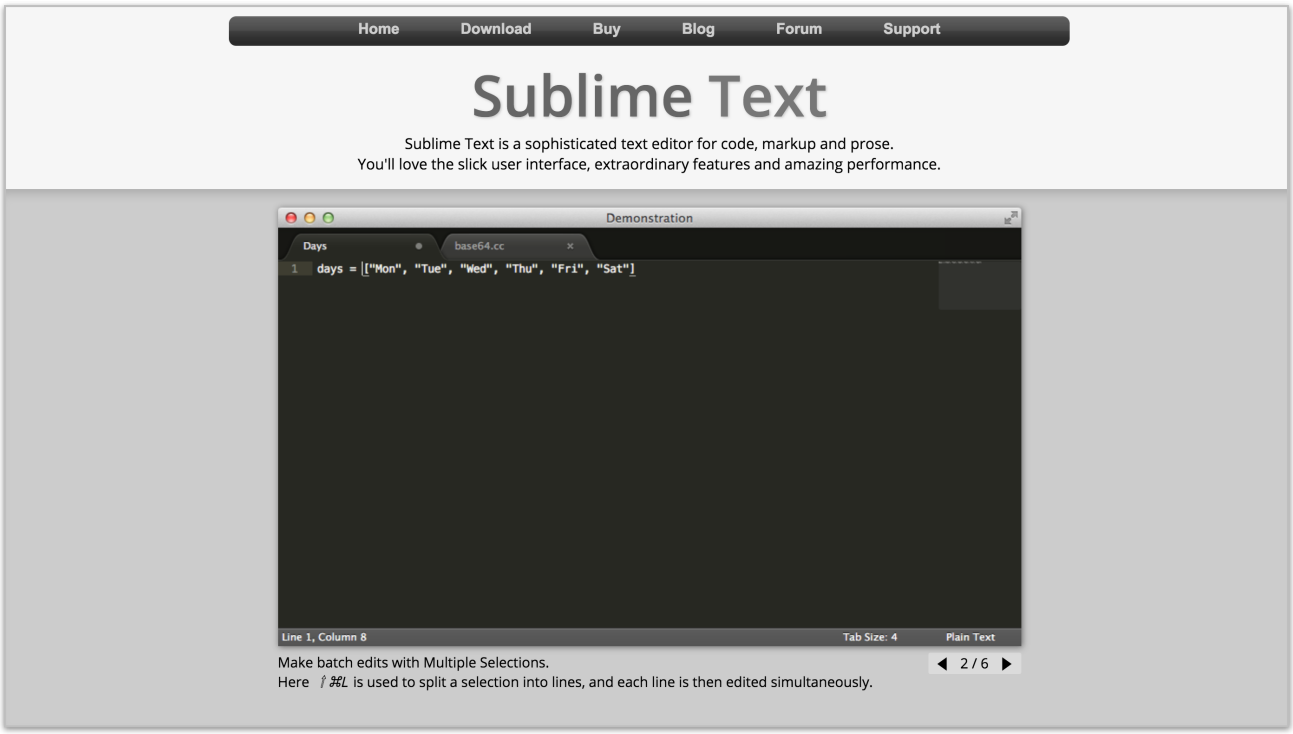
PhpStorm



上記で紹介した開発画面はPhpStormのものです。Jet Brains社が開発しているIDEで、WebStormというIDEにPHP対応機能がついたもの。ですので、HTML,CSS,JavaScriptのサポートも強力。世界中にファンがおり、国内においてもソフトが日本語対応していないにも国内でこれを使ってる現場・制作者多数。ライセンス制をとっており、1年約200ドルと少し高価ですが、自信をもって進められるIDEです。

https://www.jetbrains.com/phpstorm

Sublime Text



国内でも多くのファンがいるSublime Text。「テキストエディタ」を名乗っておりますが、IDEといって遜色がない機能を提供しています。

こちらも有料ソフトなのですが、特徴は「お試し期間に期限がない」ことです（たまに購入を促す画面は表示されるらしい）。また、パッケージのインストールでメニューの日本語化ができますので、そういう面も人気を博してる理由でしょう。
PhpStormと比べると、正解を提示するというよりも、パッケージなどでユーザのカスタマイズ性に任せたIDEとなります。

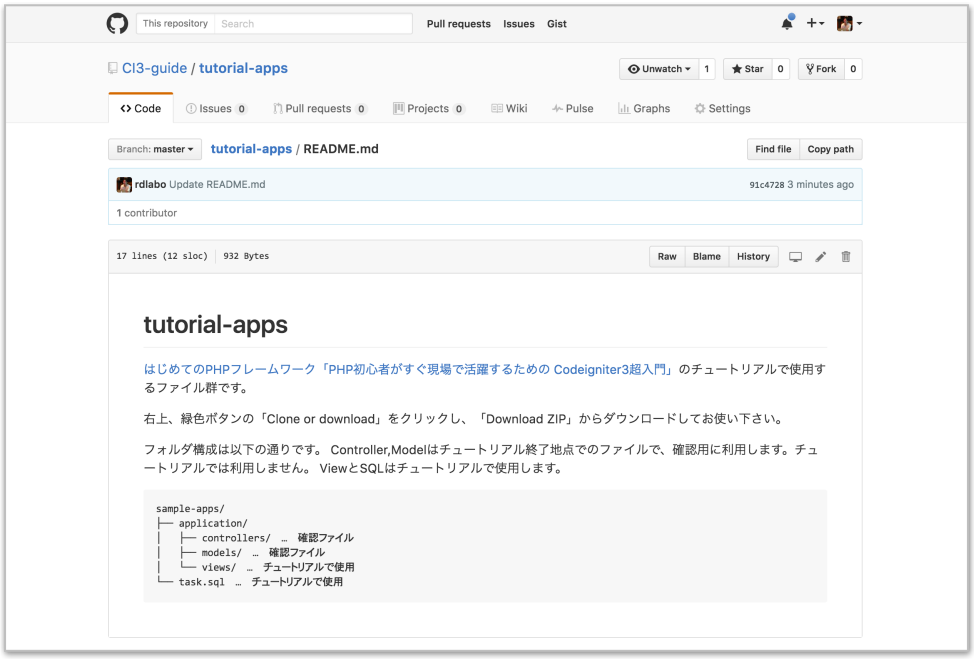
https://www.sublimetext.com/

5. チュートリアル「新規ページをつくってみよう」

Welcome.phpを参考に、オリジナルのページをつくってみましょう。
本書ではCodeIgniterを理解すること以外の作業はしなくていいように、チュートリアル用のファイルを用意していますので、それをダウンロードした上で、チュートリアルを進めます。

チュートリアルファイルをダウンロード

ダウンロードURL（ https://github.com/CI3-guide/tutorial-apps ）にアクセスして下さい。ソースコードを管理・公開できるサービス「GitHub」が表示されますので、「Clone or Download」をクリックして、「Download ZIP」からzipファイルをダウンロードします。



解凍すると以下のフォルダ構成となっているので、「application/views」の中に入っているファイルを先程インストールしたCodeIgniterの「application/views」に移動して下さい。

```
application/
└─── views/
task.sql
```

新規ページを作ってみよう

Controllerの作成

では、まずControllerを作成します。「application/controllers」の中に、User.phpというファイルを作成して下さい。そして、User.phpの中に、以下のコードを書きます。

```
<?php

class User extends CI_Controller
{
}
```

Controllerのファイル名・Class名は1文字目を大文字で、それ以外を小文字で書くルールがあります。ですので、「user.php」であったり、「class user」ではなく、「User.php」「class User」と書くようにしてください。

次にその中にメソッドをつくります。ここでは、profileというメソッドにしましょう。そこに、呼び出されているのを確認するために「welcome ci!」を表示してみます。

```
<?php
class User extends CI_Controller
{
    public function profile()
    {
        echo “<h1>welcome ci!</h1>”;
    }
}
```

ブラウザでこのページにアクセスしてみましょう。URLは「http://localhost:8888/codeigniter/public/user/profile」です。うまくできていれば、“welcome ci!”と表示されたページが作成されたと思います。

また、Controllerのメソッドに引数を用意して、URLから動的に引数を取得することもできます。$_GETでURL引数を利用するのと同じ考え方です。メソッド「profile」に引数を用意して、その引数の値を表示してみましょう。

```
    public function profile($id=null)
    {
✤       echo $id;
    }
```

「$id=null」は、$idが入っていない場合はnullを入れて、入ってる場合はその値を$idとして取り扱うという意味です。

では、試しにidに13を格納するURLにアクセスしてみましょう。以下のURLとなります。

```
http://localhost:8888/codeigniter/public/user/profile/13
```

「13」のところをいろいろ任意で変更して、ブラウザで表示が変わる様子を確認してもらえればと思います。

Viewファイルの呼び出し

次に、同じController「User」の中に新しいメソッドを作ってみましょう。detailメソッドを作成し、冒頭でダウンロードした「views/tutorial/user_detail.php」をviewとして呼び出します。

```
<?php
class User extends CI_Controller
{
    public function profile($id=null)
    {
        echo $id;
    }

    public function detail()
    {
        $this->load->view('tutorial/user_detail');
    }
}
```

「views/tutorial/user_detail.php」を呼び出しています。CodeIgniterでは、Viewの呼び出し時に、ViewファイルがPHPファイルの場合は、拡張子を省略してもいいルールがありますので、「$this->load->view("tutorial/user_detail");」と表記します。これは、「$this->load->view("tutorial/user_detail.php");」と同じものです。

それでは、表示してみましょう。

ControllerからViewへ値の引き渡し

しかし、これではContollerからViewを呼び出してるだけで、動的な処理が行われていません。実務では、ただ呼び出すだけではなく、値の受け渡しも行うことが多々必要があります。そこで、ここでは、氏名、出身都道府県の情報を引き渡してみます。

```
<?php
class User extends CI_Controller
{
    public function profile($id=null)
    {
        echo $id;
    }

    public function detail()
    {
        $data = [
                'name' => '山田太郎',
                'pref' => '東京都'
            ];
         $this->load->view('tutorial/user_detail', $data);
    }
}
```

Viewへの値の受け渡しは、key名をつけた連想配列を、第二引数に置いて行います。このことによって、Viewの中で、key名を変数名とした変数を取り扱うことが可能となります。

「tutorial/user_detail.php」では、予め以下のように値が引き渡された時には表示するようにしていますので、値を受け渡すだけで氏名、都道府県名が表示されます。

```
<tr>
    <th>氏名</th>
    <td><?php if(isset($name)){echo $name;}?></td>
</tr>
<tr>
<th>都道府県</th>
    <td><?php if(isset($pref)){echo $pref;}?></td>
</tr>
```

「name」だと、Controllerで$data[“name”]で受け渡された連想配列が、Viewでは$nameへと入れ替わっています。このようにして、ControllerからViewに動的に変数を受け渡すことが可能となります。

では、実際に「http://localhost:8888/codeigniter/public/user/detail」にアクセスして、どのように表示されるかを確認してみましょう。

氏名と都道府県が追加されます。

以上を応用すると、例えばユーザの詳細ページで「アクセスするURL（引数）に該当するユーザの詳細を表示する」といった動的なページの生成を行うことができるようになります。

コラム「PHP標準コーディング規約」

いざきれいにコードを書こうと思っても、命名規則ってどうしようからはじまり「きれいにコードを書くために決めないといけないこと」が多すぎて戸惑うかと思います。そこで様々な現場では「コーディング規約」がつくられ、それに則ったコーディングを行うのが一般的です。

しかしながら、コーディング規約を0からつくるのはとても手間ですので、現場では「Framework Interoperability Group」というグループがつくっているPHP標準コーディング規約「PSR」を使うことが一般的です。

PHP標準コーディング規約

PHP標準コーディング規約「PSR」は、PSR-1からPSR-7まで定義されています。[4] コードを書く時に最も重要となるPSR-1,PSR-2について軽くご紹介します。

PSR-1

基本的なコーディング規約です。PHPタグは「<?php」か「<?=」かのどちらかを使用しないといけないということからはじまり、以下のようなことを定めています。

- 定数（DEFINE）は全て大文字で書き、区切り文字はアンダースコアで記述する
- プロパティは、大文字からはじめる単語を区切り文字なく接続するパスカルケース（例えば、GetUserData）か、最初の単語以外を大文字からはじめてつなげる、キャメルケース（getUserData）か、単語をアンダーバーで区切るスネークケース（get_user_data）のどれかで統一して書きましょう

PSR-2

コーディングスタイルに関する規約です。例えば、一行あたり絶対的な文字制限はありませんが、120文字も書いたら改行しましょうね、とか、インテントは4つの半角スペースを利用しましょう、や、以下の内容が定められています。

[4] PSR-5,8-17はレビュー、ドラフトプロセスです。またPSR-0は非推奨となりました。

- 複数のメソッドの引数を書く時は、カンマの前にはスペースなし、カンマのあとにはスペースを1つ記述するようにします。
- if文を書く時は、条件文を表記するカッコの前後に半角空白を記述します。また、処理の前後は改行を入れて処理を表記するようにします。

もっと詳しく知るために

PSR-0,PSR-1,PSR-2は日本語化されていますので、ぜひ読んでみて下さい。このルールをチームで共有することによって、コードの個人差が吸収され、読みやすいコードの実現に資することができます。

PSR-0	https://goo.gl/0ZSTnl
PSR-1	https://goo.gl/tNzlN2
PSR-2	https://goo.gl/3pfw0Q

CodeIgniter PHPスタイルガイドってどうなの？

CodeIgniterのユーザガイドでも、PHPスタイルガイド（ https://goo.gl/nXOALa ）を提供しております。

しかしながら、こちらは「CodeIgniter本体の開発」に使われる開発ガイドで、CodeIgniterを利用したアプリ開発を対象したものではなく、前述のPSRと衝突しますので本書では採用いたしません。特に、真偽を「TRUE」「FALSE」と大文字で書くのは近年のPHPの文化にそぐいませんので、注意が必要です。

6. チュートリアル「HelperとLibraryで機能拡張」

CodeIgniterでは、Model,View,Controllerだけではなく、より開発をスピーディにするために「Helper」と「Library」というパッケージ群を備えています。これを使うと、PHPで自作すると数行〜数十行書かないといけない処理を、数行で行うことができます。

本章では、HelperとLibraryの使用方法について確認します。

Helper

Helperは、便利な関数群です。PHPを学ぶ時に「ユーザ定義関数」などはやりませんでしたでしょうか。CodeIgniterでは、よく使われるユーザ定義関数を「Helper」という形でまとめて提供しており、リンクを作成するのを支援する URLヘルパー、テキストの整形処理を行う Textヘルパーなど21種類のHelperが用意されています。

本項では、開発でよく使うURLヘルパーを通して、Helperの使い方を説明します。

URLを処理するURLヘルパー

URLヘルパーは、URLバーに表示されているURLを取り出したり、加工したりすることが簡単にできる機能です。

Helperを使う時、明示的にそのHelperを呼びださなければなりません。ですので、まず、URLヘルパーを呼び出します。

前章のController「User」に追記していきましょう。

```
<?php
class User extends CI_Controller
{
// (中略)

public function url()
{
    $this->load->helper('url');
}
```

「http://localhost:8888/codeigniter/public/user/url」にアクセスできるようになしました。しかし現時点では、URLヘルパーをロードしているだけなので何も表示されません。

では、Helperを具体的に使っていきます。次のように2行追記して、ブラウザを更新して下さい。

```
<?php
class User extends CI_Controller
{
// (中略)

    public function url()
    {
        $this->load->helper('url');
        echo auto_link('googleのurlは、https://google.comです');
        echo '<br>';
        echo site_url();
    }
```

auto_link($str)は、第一引数にテキストを格納し、その中にURLかメールアドレスがあったら、それを自動的に検知して、リンクを貼ってくれます。これを自分でしっかり書こうとすると、いちいち正規表現を書くことになりますので、とても便利です。

また、「site_url()」は、CodeIgniterをインストールしているフォルダのURLを「index.php」までつけて返します。しかしながら、MAMPは一般的なサーバと異なり、「site-url()」ではドメインをとれず「http://[::1]」という表示になってしまうので、MAMPで使う場合はドメインは手動で設定する必要があります。

「application/config/config.php」を開くと、26行目に

```
$config['base_url'] = '';
```

がありますので、ここに「http://localhost:8888/」と入力して下さい。そうすると正常な値が取れるようになります。

CodeIgniterには、21のHelperが用意されています。何があって、どうやって使うかは、以下の公式ドキュメントで紹介されていますのでぜひご確認ください。

http://codeigniter.jp/user_guide/3/helpers/

Library

Libraryは、Helperよりも複雑な処理をパッケージ化しています。簡単にemailを送信したり、htmlのテーブルを作ったり、PHPで処理するのが面倒な画像処理を行ったりすることができます

ここでは、さわりとして、UserAgentクラスを紹介します。

ユーザ情報を簡単に取得できるUserAgentクラス

ユーザは、ブラウザを使ってアクセスしてくる時、利用しているブラウザ情報や端末、どこから訪問してきているかなど、ユーザ毎に様々なデータ（UserAgent）をもってきています。UserAgentクラスは、それを簡単に取得できるLibraryです

それでは早速、使っていきましょう。LibraryもHelperと同じように、最初に呼び出しを行います。Controller「User」に、新しく「agent」というメソッドを追加し、Library を呼び出します。

```
<?php
class User extends CI_Controller
{
// (中略)

    public function agent()
    {
        $this->load->library('user_agent');
```

まず、user_agentクラスでとれるすべての情報を画面に表示してみます。

```
<?php
class User extends CI_Controller
{
// (中略)

    public function agent()
    {
        $this->load->library('user_agent');
        echo $this->agent->agent_string();
    }
```

そして、「http://localhost:8888/codeigniter/public/user/agent」にアクセスすると、あなたのユーザ環境が表示されます。

また、user_agentでは、これらの情報を分割して表示することもできるので試してみます。

```
<?php
class User extends CI_Controller
{
// (中略)

    public function agent()
    {
        $this->load->library('user_agent');
        echo $this->agent->agent_string();
        echo '<br>';
        echo $this->agent->browser();
        echo '<br>';
        echo $this->agent->platform();
    }
```

「$this->agent->browser()」は、閲覧しているブラウザ名だけを返します。「$this->agent->platform()」は、閲覧しているOSを返します。これらを使って、例えば「閲覧ブラウザによって表示を変える」であったり、簡易なアクセス解析をつくるということを簡単に実現できます。

CodeIgniterには、30のLibraryが搭載されています。こちらも、以下の公式ドキュメントで詳細に使い方などが解説されております。

http://codeigniter.jp/user_guide/3/libraries/

7. チュートリアル「タスクリストを作ってみよう」

本章では、タスクリストをつくることを通して、データを取り扱う「Model」と、入力データを精査する「form validationクラス」の使い方を学びます。

本チュートリアル用に、新規でControllerを用意し、Viewを読み込みます。
では、タスクリスト用のControllerを用意しましょう。Controller名はTask、メソッドはindexとします。ですので、Controllerのファイル名は、Task.phpとして用意します。また、viewは、「views/tutorial/task.php」を読み込みます。

```
<?php

class Task extends CI_Controller
{
    public function index()
    {
        $this->load->view('tutorial/task');
    }
}
```

そして、「http://localhost:8888/codeigniter/public/task」にアクセスしてください。上記の画面が表示されます。まだ何もデータを登録していないのでタスクはありません。

データベースを用意しよう

では、ここからデータベースとの連携を進めていきましょう。データベースにも様々な種類がありますが、本書では一般的であるMySQLを採用し、解説を行います。

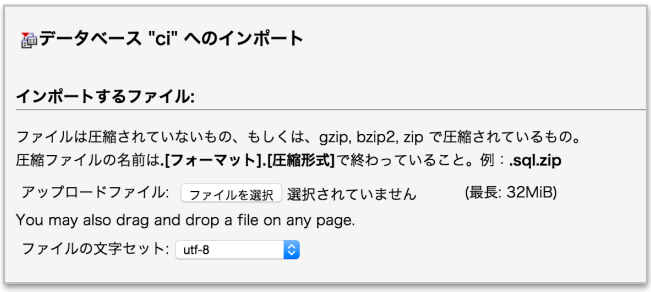
まずはデータベースとテーブルを用意します。

phpMyAdminでデータベースとテーブルを作成

MAMPには、スムーズにMySQLを操作するために、デフォルトでphpMyAdminというMySQLをブラウザから管理できるツールが用意されています。まず、スタートページにアクセスして、上部メニューのツール -> phpMyAdminをクリックし、phpMyAdminに移動します。

まずは、CodeIgniterが使用するデータベースを用意します。左上の「New」をクリックし、「データベースを作成する」に、データベース名「ci」を入力、照合順序「utf8_unicode_ci」を選択して「作成」をクリックするとデータベースが作成されます。

また、そのデータベースにチュートリアル用のテーブルをインポートします。チュートリアル「新規ページをつくってみよう」でダウンロードしたチュートリアル用フォルダに入っている「task.sql」をインポートします。上部メニューの「インポート」をクリックしてインポート画面へ遷移して下さい。



アップロードファイルに、「task.sql」を指定して、下部にある「実行」ボタンをクリックすると、テーブルファイルがインポートされます。構造は次の通りで、デフォルトでテスト用データが3行格納されています。

データベースの接続設定

さて、用意したデータベーステーブルをCodeIgniterで読み込むための設定を行います。データベースの設定は、「application/config/database.php」で行います。

MAMPの前述したdatabase設定を読み込むには、以下の通りです。

```
$db['default'] = array(
    'dsn'   => '',
    'hostname' => 'localhost',        // 接続先
    'username' => 'root',             // ユーザ名
    'password' => 'root',             // パスワード
    'database' => 'ci',               // データベース名
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

変更部分以外は初期設定で大丈夫です。より詳しく知りたい方は、公式ドキュメントのDatabase設定（ https://goo.gl/hSVG3M ）をご覧下さい。

Modelを用意してデータ呼び出し

データをやりとりするModel

冒頭でControllerとViewを用意して、タスクリスト風の画面を表示しましたが、このままではタスクリストと言っても名ばかりで、タスクの登録も表示もできない状態です。

この「データを登録する(Create)」「データを表示する(Read)」であったり、（今回はやりませんが）「更新する（Update）」「削除する（Delete）」が、「データのやりとり」の具体的内容でModelに書くべき内容となります。

では、データベースの操作はPHPからはどのように行ったらいいのでしょうか。MySQLは、データベースの操作をSQL文といわれる構文で行います。例えば、先ほどデータベースに登録したデータを取り出すためのSQL文は次の通りです。

```
SELECT * FROM task
```

しかしながら、この構文を覚えて処理するのはハードルが高いため、CodeIgniterにはSQL文をPHPで生成するための「QueryBuilder」という機能が用意されています。
「QueryBuilder」を使って先ほどのSQL文を書くと、以下の通りです。

```
$this->db->get('task');
```

これを実行すると、先ほどのSQL文が自動的に生成されます。本書では、この機能をつかってデータベースの操作を進めていきます。

データの表示を行おう

データの取得と表示（Read）を行うためのModelを作成します。Modelファイルは「application/models/」に格納するルールがあります。また、Modelファイルは

「◯◯_model.php」という命名ルールにすることを本書では推奨しておりますので、「application/models/Task_model.php」を作成して下さい。

その中に、まず以下のように書きます。

```
<?php

class Task_model extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
        $this->load->database();
    }
}
```

ModelもControllerと同様に、ファイル名・クラス名の先頭を大文字にする必要があります。「parent::__construct()」は、定型句となります。[5]2行目の「$this->load->database();」はデータベースクラスを呼び出してきています。

そして、このModelの中にデータベースからデータを取り出すためのメソッドを書いていきます。

[5] システムのためのログ関数を呼び出しています。

```
<?php

class Task_model extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
        $this->load->database();
    }

✣     public function lists()
✣     {
✣         $per_page = 10;         // 表示数
✣         $query = $this->db->get('task', $per_page);
✣         if ($query->num_rows() > 0){
✣             return $query->result_array();
✣         }else{
✣             return [];
✣         }
✣     }
}
```

「$query = $this->db->get('task', $per_page)」で、$queryにテーブル「task」の中身を$per_pageに設定されている件数、格納しています。$queryには、成功したか失敗したか、何行返ってきたのか、などの様々な情報が入っていますので、まず「$query->num_rows()」で1行以上の結果があったかを判定して、1行以上あった時は「$query->result_array()」で返り値を返します。[6]

0行だった場合は、空の配列を返しています。[7]

ControllerでModelを呼びだそう！

では、ControllerからModelを読み込んで、先ほど作成したメソッドを実行しましょう。Controller「Task」に、次の2行を追記します。

---

[6] 0行より値が上だった場合と条件付けておりますが、num_rows()は自然数を返しますので実質は「1行以上」という意味になります

[7] 0行で「$query->result_array()」を返した場合も空の配列が返されますので、この条件分岐は省略可能です

```
<?php

class Task extends CI_Controller
{
    public function index()
    {
✤           $this->load->model('task_model');
✤           $data['task_list'] = $this->task_model->lists();
            $this->load->view('tutorial/task' , $data);
    }
}
```

1行目の「$this->load->model("task_model")」で、Task_modelを読み込みます。そうすると、Task_modelは、$this->task_modelに格納され、Controllerで呼び出すことが可能となります。そして、「$this->task_model->lists()」で先ほどつくったメソッド「lists」を呼び出して、返り値を$data["task_list"]に格納。その後、その値をviewに引き渡しています。

「http://localhost:8888/codeigniter/task」をリロードしてみてください。次の画面がでると成功です。

データを新規登録しよう

続いて、タスクの追加を行うことができるようにしましょう。Viewに、入力フォームがありますが、これが機能するように設定します。

まずは、データをバリデーションしよう

フォームをつかって送信されたデータを登録したり、メールの本文にして送信するといった処理は、アプリケーションでは当たり前のように行われます。

しかしながら、そのデータは「ユーザが入力」するものであり、データ自体が正しい保証はありません。必須入力なのに入力されていなかったり、半角数字を入力する欄に全角数字が入っていたりと、予期しないデータが入ってくることは多々あります。

そこで、フォームから送信されたデータの妥当性をチェックする「バリデーション」が必要となってきます。このことによって、アプリケーションを保全し、セキュリティリスクを大きく下げます。

CodeIgniterでは、フォームバリデーションのために「form_validationライブラリ」を提供しているので、これを使ってデータのバリデーションを行います。

まず、「form_validationライブラリ」の呼び出しを行います。

```
<?php

class Task extends CI_Controller
{
	public function index()
	{
		$this->load->model('task_model');
		$this->load->library('form_validation');

		$data["task_list"] = $this->task_model->lists();
		$this->load->view('tutorial/task', $data)
	}
}
```

「form_validationライブラリ」はバリデーションするルール（どれが必須入力で、どれは半角数字じゃないといけないかの設定）を記述し、それが終わってから妥当性チェックを行います。

ViewのFormからは、以下のコードでデータが送られてきます。つまり、$_POST[“task”] が送信されるので、そのデータをバリデーションする必要があります。

```
<form method="POST">
	<div class="form-group">
		<label for="exampleInputEmail1">新規タスク</label>
		<input type="text" name="task" class="form-control" placeholder="タスク名"
required>
	</div>
	<button type="submit" class="btn btn-primary btn-block">タスク追加</button>
</form>
```

ここでは、データは以下のルールのものでなければならないとします。

データ名	バリデーションルール
task	必須入力・5文字以上・20文字以下

　このルールに沿って、Controllerにバリデーションルールを追記していきます。バリデーションが挙動するのは、POSTが送信される時だけですので、「$this->input->post()」でPOST処理の有無を確認して[8]、バリデーション処理を行います。
ルールは、「$this->form_validation->set_rules」で書いていきます。

```
class Task extends CI_Controller
{
    public function index()
    {
        $this->load->model('task_model');
        $this->load->library('form_validation');

        if($this->input->post()){
            $this->form_validation->set_rules('task', 'タスク', 'required|
min_length[5]|max_length[20]');
        }

        $data["task_list"] = $this->task_model->lists();
        $this->load->view('tutorial/task', $data);
    }
}
```

　「set_rule」は、引数を3つ必要とします。第一引数はデータの名前（name属性の値）、第二引数はその名前の表示名です。エラーメッセージなどに利用されます。
　そして、第三引数が、フォームのバリデーションルールです。バリデーションルールは公式ドキュメントを参考ください。

[8] if ('POST' === $this->input->method('TRUE')) で、送信メソッドを確認する方法もあります

ルールリファレンス

The following is a list of all the native rules that are available to use:

ルール	Parameter	説明	例
required	No	空き要素の場合はFALSEを返す	
matches	Yes	formの要素が一致しない時はFALSEを返す	matches[form_item]
regex_match	Yes	Returns FALSE if the form element does not match the regular expression.	regex_match[/regex/]
differs	Yes	Returns FALSE if the form element does not differ from the one in the parameter.	differs[form_item]
is_unique	Yes	Returns FALSE if the form element is not unique to the table and field name in the parameter. Note: This rule requires Query Builder to be enabled in order to work.	is_unique[table.field]
min_length	Yes	指定する文字数より少ない場合はFALSEを返します。	min_length[3]
max_length	Yes	指定する文字数を超えた場合はFALSEを返します。	max_length[12]
exact_length	Yes	指定する文字数と一致しない場合はFALSEを返します。	exact_length[8]
greater_than	Yes	指定した値よりも（数字的に）小さいか、数字でない時にFALSEを 返します。	greater_than[8]
greater_than_equal_to	Yes	指定した値よりも（数字的に）等しいもしくは小さいか、数字でない時にFALSEを 返します。	greater_than_equal_to[8]
less_than	Yes	指定した値よりも（数字的に）大きいか、数字でない時にFALSEを 返します。	less_than[8]

https://goo.gl/JZABh7

そして、「$this->form_validation->run()」でこのバリデーションを実行します。データに問題がなければTRUE、問題に問題があればFALSEを返しますので、if文の中に入れて使います。

```
class Task extends CI_Controller
{
    public function index()
    {
        $this->load->model("task_model");
        $this->load->library("form_validation");

        if($this->input->post()){
            $this->form_validation->set_rules('task', 'タスク', 'required|min_length[5]|
max_length[20]');

✤           if ($this->form_validation->run()) {
✤               $data['create'] = true;
✤            }else{
✤               $data['create'] = false;
✤            }
        }

        $data["task_list"] = $this->task_model->lists();
        $this->load->view("tutorial/task",$data);
    }
}
```

では、実際にformからデータを送信してみましょう。5〜20文字のタスク名が送信されると、「タスクを追加しました」が表示され、間違っていると何も表示されていないことを確認して下さい。

現時点では、まだタスクを追加するModelを書いていないので「タスクを追加しました」と表示されますが、実際に追加はされません。

データの登録を行おう

前項で、送信されたデータの妥当性のチェックができましたので、データ登録用のModelメソッドを用意して、ControllerからModelにデータを引き渡しましょう。

まず、先ほど作った「Task_model」に、データを登録するメソッド「create」を作成します。

「create」では、登録するデータを第一引数で持つものとします。またこれらのデータを格納するのは、MySQLのテーブル「task」にあるカルム「task_name」となります。

「Query Builder」では、データの登録を行う時は、連想配列にカルム名をkeyとして持ったデータを格納し、それをinsertメソッドに第一引数をテーブル名、第二引数を連想配列として受け渡し、実行します。コードは次の通りです。

```
<?php

class Task_model extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
    }

    public function lists()
    {
        $per_page = 10;         // 1ページあたりの表示数
        $query = $this->db->get('task', $per_page);
        if ($query->num_rows() > 0){
            return $query->result_array();
        }else{
           return [];
        }
    }

    public function create($task)
    {
        $data = ['task_name' => $task];
        $this->db->insert('task', $data);
    }
}
```

このメソッドをControllerで呼び出すと、テーブルへの登録が行われます。早速、呼び出してみましょう。

```
class Task extends CI_Controller
{
    public function index()
    {
        $this->load->model("task_model");
        $this->load->library("form_validation");

        if($this->input->post()){
            $this->form_validation->set_rules('task', 'タスク', 'required|min_length[5]|
max_length[20]');

            if ($this->form_validation->run()) {
                $this->task_model->create($this->input->post('task'));
                $data["create"] = true;
            }else {
                $data["create"] = false;
            }

        }

        $data["task_list"] = $this->task_model->lists();
        $this->load->view("tutorial/task",$data);
    }
}
```

「$this->task_model->create」で、先ほどつくったメソッドを呼び出します。引数は、$this->input->post('task')で受け取ったものを引き渡します。

これで、先ほどの同じようにformからタスク登録をしてみましょう。うまく動いていれば、正しい値を入力して送信する度に、下のタスクリストにタスクが増えていきます。

localhost:8888/CI3-guide-sample-apps/public/task
CodeIgniter超入門
チュートリアル「タスクリストを作ってみよう」
新規タスク
タスク名
タスク追加
タスクを追加しました。
タスク
登録日
「タスクリストを作ってみよう」を終わらせる
166日前
CodeIgniter本のチュートリアルを全部終わらせる
146日前
MySQLについて学んで見る。「SQLアンチパターン」は入門向けじゃないもののおすすめです。
146日前
追加したタスクですよ
0日前

8. コードリファクタリングをしよう

コードは動きさえすればいいかというと、「1. 開発を加速するPHPフレームワーク」の章で確認したように「いいコード」を書かなければなりません。
しかしながら、いきなりいいコードを書くことは難しいので、「一度コードを書いて、それを修正していいコードにする」という作業を一般的に行います。この作業を「コードリファクタリング」といいます。

プログラムが行う処理は同じですが、より読みやすく、より使い回しのきくコードへと修正を入れていきます。本章では、この考え方について学んでいきます。

脱・Fat Controller

読みにくく、使い回しのきかないコードの代表格が、「Fat Controller」です。「コントローラーが太っている」状態で、Controllerに処理が集中しすぎ、ModelやViewなどが大して機能していないものを指します。Controllerの1methodが100行を超えるようなことになれば、Fat Controllerになっている可能性が高いです。

Fat Controllerを修正していくために効果的な方法を4つご紹介します。

1. データ処理はModelに任せよう

データ処理は、Controllerではなく、Modelで行いましょう。例えば、以下のような処理を行うModelとControllerがあるとします。

```
// Model

class Sample_model extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
        $this->load->database();
    }

    public function get_users()
    {
        $query = $this->db->get('users');
        if ($query->num_rows() > 0){
            return $query->result_array();
        }else{
            return [];
        }
    }

    public function get_record($user_id)
    {
        $this->db->where('user_id', $user_id);   //「user_id」が一致するレコードを抽出
        $query = $this->db->get('record');
        if ($query->num_rows() > 0){
            return $query->row_array();
        }else{
            return [];
        }
    }
}
```

```
// Controller

class Sample extends  CI_Controller
{
    function users()
    {
        $users = $this->user_model->get_users();
        $user = [];
        foreach($users as $key => $tmp_user){
            $record = $this->sample_model->get_record($tmp_user['user_id']);
            $tmp_user['recode'] = $recode;
            $user[] = $tmp_user;
        }
    }
}
```

Modelの「users」メソッドでユーザー一覧を取得し、Controllerでユーザー一覧をループで回して、ループ内でデータを結合しています。

もちろんこれでも問題なく動くのですが、そもそもの話、Modelはデータベースとの通信を担当するのではなく「データ処理」を担当しますので、本来は以下のように書くべきです。

```
// Model
class Sample_model extends CI_Model
{
// 中略

    public function get_user()
    {
        $query = $this->db->get('users');
        $users = $query->result_array();

        if ($query->num_rows() > 0){
            $user = [];
            foreach($users as $tmp_user){
                $this->db->where('name', $tmp_user['user_id']);
                $query = $this->db->get('record');
                $tmp_user['recode'] = $query->row_array();
                $user[] = $tmp_user;
            }
        }else{
            $user = [];
        }
        return $user;
    }
}
```

```
// Controller
class Sample extends  CI_Controller
{
    public function users()
    {
        $this->load->model('user_model');
        $user = $this->user_model->get_user();
    }
}
```

こうすると、Controllerがすっきりしますね！

２．Controllerにhtmlは書かない

Controllerに次のような処理も書きがちです。どうせ1回foreachをまわすなら、そこでテーブルの作成までやってみたいと思うかもしれません。

```
// Controller
class Sample extends  CI_Controller
{
    public function users()
    {
        $this->load->model('user_model');
        $users = $this->sample_model->get_users();
        $table = '';
        $i = 0;
        foreach($users as $tmp_user){
            $i++;
            $table .= '<tr><td>'.$i.'</td><td>'.$tmp_user['name'].'</td></tr>';
        }
        $this->load->view('sample',['table'=>$table]);
    }
}
```

けれど、これだと結果的にViewをさわる人は列の入れ替えなどのテーブル構造の変更であったり、もしくはタグの入れ替えなどをしようと思ってもできない状態となってしまいます。本来のViewの役割をControllerが奪ってしまっている状態です。

ですので「ControllerにhtmlはＷ書かない」とルールを決めておく方が大切です。

３．自作Library、自作Helperにコードを移動

先ほどLibrary、Helperを紹介しましたが、これらは自分専用のものをつくることができます。それぞれ、「application/libraries」「application/helpers」が格納場所になります。

共通する処理であったり、またControllerの可読性を悪くしている複雑な処理は、そちらへ移動してしまうと読みやすいコードになる場合が多いです。次のチュートリアルでつくってみますので、具体的な手順はそちらで確認していきます。

４．設定ファイルを活用

また、消費税率など汎用的に使い回す設定をconfigファイルに格納することもできます。「application/config」の中に設定ファイルとして保存します。こちらも、次のチュートリアルで具体的な手順を確認します。

コードの再利用性を高めよう

１．命名規則を決めよう

その時の気分であったり、メンバー毎の発想でModelやLibraryの名前をくると、あとで管理が大変になります。Model名ひとつとっても、テーブル名をつけるか、機能・役割名をつけるか、単語の間にはアンダーバー（_）をつけるか単語の最初の文字を大文字にしてつなげて書くかといった違いがあります。

そこで、本書では、以下の命名規則でファイル名、クラス名をつけることを進めています。Modelファイルは、「機能_model.php」です。なので、ユーザ情報を取り扱うModelだと、「User_model.php」となり、クラス名は「User_model」となります。

また、Libraryファイルは「機能_lib.php」を命名規則としております。成績管理に関するコードを集めたLibraryファイルだと「Record_lib.php」となります。

ファイル名・クラス名ももちろんのことながら、メソッド名も「lists」「create」といったように、何をするメソッドなのかがわかるように気をつけながら名前をつけていきましょう。

２．処理を共通化しよう

例えば、以下のように5件の結果と、10件の結果を取得するModelをつくるとします。

```
//Model Class
(中略)
public function get_user_5()
{
    $query = $this->db->get('users',5); // 5件だけを取得
    return $query->result_array();
}

public function get_user_10()
{
    $query = $this->db->get('users',10); // 10件だけを取得
    return $query->result_array();
}
```

こうすると、ほとんど同じコードを2つ書くことになります。15件取得する必要ができたら、更にもうひとつメソッドを作ったりしたら大変手間ですよね。ですので、こういう場合、件数を変数化して$numに格納するようにします。

```
//Model Class
(中略)
public function get_user($num)
{
    $query = $this->db->get('users' , $num);
    return $query->result_array();
}
```

すると、Controllerで呼び出す時、引数を変えるだけで異なる結果を得ることができます。これは、Modelだけではなく、Library、Helperを作成する時も同じ考え方ですので、ぜひ身につけましょう。

コードリファクタリングは「絶対」のルールはありませんが、ルールをチームで共有することが大切です。

9. チュートリアル「リファクタリング」

それでは、実際にコードリファクタリングに挑戦してみましょう。
コードリファクタリングには、「鳥の目」と「虫の目」の使い分けが重要です。「鳥の目」によって、ファイル間にまたがってコードを見直し、似たコードを書いてる部分を共通化します。また、「虫の目」で、ControllerがFat Controllerになっているか否か、どうしたら読みやすいコードになるのかというリファクタリングに取り組みます。

設定をユーザ設定にまとめよう

- 消費税率
- 1ページあたりに表示するリストの数
- 公式のメールアドレス

など、開発を進めていくと、webアプリに「設定項目」が生まれます。最初は書き散らしてしまうのですが、あとで変更が起きると修正箇所が分散しているので、修正漏れが生まれがちです。そこで、設定を1箇所にまとめる「ユーザ設定」があります。「application/config/」以下に、PHPファイルを置いて「$config」に配列で格納することによって、どこからでもアクセスできるようになります。
実際に、先程つくったタスクリストの1ページあたりの表示件数を変更してみましょう。

まず、設定ファイルを「application/config/per.php」としてつくります。内容は以下の通りです。

```
<?php

$config['per_page'] = 10;
```

次に、この設定を、「application/model/Task_model.php」で呼び出します。最初に 「application/config/per.php」をロードして、そのあと設定名で呼び出します。

```
<?php

class Task_model extends CI_Model
{
    public function __construct()
    {
        parent::__construct();
        $this->load->database();
✤ [9]
    }


    public function lists()
    {
        // 1ページあたりの表示数
✤       $per_page =  $this->config->item('per_page');
        $query = $this->db->get('task', $per_page);
        if ($query->num_rows() > 0){
            return $query->result_array();
        }else{
           return [];
        }
    }
}
```

PHPのグローバル変数で設定を用意することも可能ですが、CodeIgniterのユーザ設定を用いると、開発環境と本番環境で設定ファイルを変更する、といったことも可能となりますので、こちらがおすすめです。

常に呼び出すものはautoloadしよう

ModelをControllerで使う時、

[9] 公式ドキュメントでは、 $this->config->load();となっておりますが、挙動は同じです。

```
$this->load->model("呼び出すmodel");
```

と、当該Controllerで「これから使うよ」宣言が必要であることをご紹介しました。けれど、Modelは、どこか一箇所だけで使うことは稀で、基本的にはController間を横断して使います。
ですので、一箇所一箇所でloadするのは非効率である場合があります。

そこで、CodeIgniterには「いつも呼び出すもの」をまとめてloadする機能があります。それがautoload機能で、「config/autoload.php」に書くことによって全ての箇所で自動的にloadされます。

実際に試してみましょう。「application/controllers/Task.php」のタスクリストの

```
$this->load->model('task_model');
```

を削除して、「application/config/autoload.php」の135行目にtask_modelを追加します。

```
$autoload['model'] = array('task_model');
```

先程のタスクリストを表示しても問題なく表示できるのを確認できるはずです。

こうすることで、どこでも「task_model」をloadせずに使うことができるようになります。user_modelなど、多くのControllerで使いまわすものはここに登録することが多いです。

これは、Modelにかぎらず、Library、Helper、Configなど、View以外の$this->loadで呼び出すものは全てautoloadに登録することができます。

ちなみに、筆者は、autoload.phpには以下のパッケージはデフォルトで入れて開発しています。ご参考下さい。

```
$autoload['libraries'] = array('database','session', 'form_validation');
$autoload['helper'] = array('url');
```

ただ、使うものをすべてautoloadに登録すると、loadするものが多くなりすぎて、CodeIgniterの挙動が遅くなることがあるので注意が必要です。

Viewを分割して使いまわせるように

Headerの分離

Viewを見直してみましょう。Viewフォルダの中には次の2つのファイルが格納されています。

- tutorial/user_detail.php
- tutorial/task.php

この2つのファイルをよくみてみると、<body>以前が同じHTMLで書かれていることに気づきます。この部分を「header」と名付けます。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>CodeIgniter超入門</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
alpha.4/css/bootstrap.min.css" crossorigin="anonymous">
  <style>
    header {
      text-align:center;
      padding:2rem 0;
    }
    form{
      border:1px solid  #c0c0c0;
      padding:2rem;
      margin-bottom:2rem;
    }
  </style>
</head>
```

今は何らかの変更があっても変更するファイルは2つだけですが、Viewファイルの数が10,20と膨れていくと、一箇所変更するのにいくつものファイルを変更しなければならなくなります。これは、前章で確認した「コードの再利用性」に反しますので、リファクタリングをかけます。

まず、「header」を別ファイルに保存します。「application/views」の中に、「common/」というフォルダをつくって、その中に「header.php」を用意します。

```
<?php /* application/views/common/header.php */ ?>

<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>CodeIgniter超入門</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
alpha.4/css/bootstrap.min.css" crossorigin="anonymous">
  <style>
    header {
      text-align:center;
      padding:2rem 0;
    }
    form{
      border:1px solid  #c0c0c0;
      padding:2rem;
      margin-bottom:2rem;
    }
  </style>
</head>
```

続いて、「user_detail.php」と「task.php」で「header」をheader.phpを呼び出すコードに置き換えます。

```
<?php  $this->load->view('common/header'); ?>
```

これで、headerに修正いれる時は、複数箇所に手をいれなくても、一箇所のみの修正で対応できるようになりました。

CSSを外部ファイル化する

しかし、header.phpをみてみますと、CSSがHTMLファイルに直書きされています。まだ、CSSが2要素しかないのですっきりして見えますが、これが10要素、100要素になった時……ということで、更にCSSを切り出します。

まず、CSSファイルを外部に保存します。style.cssとしましょう。CodeIgniterの公開フォルダはapplication/publicですので、publicの中に保存します。

```
/* application/public/style.css */

header {
    text-align:center;
    padding:2rem 0;
}
form{
    border:1px solid  #c0c0c0;
    padding:2rem;
    margin-bottom:2rem;
}
```

公開フォルダに置いていますので、直接style.cssにアクセスすることも可能です。試しに、http://localhost:8888/codeigniter/public/style.css にアクセスしてみて下さい。無事、style.cssの中身が表示されたら成功です。

続いて、header.phpから、style.cssを呼び出すようにします。<style>から</style>までを以下に置き換えましょう。

```
<link type='text/css' href=‘http://localhost:8888/codeigniter/public/style.css'/>
```

しかしながら、これだとMAMPではなく、サーバにあげた時などにドメインを毎回書き換える必要があり、手間です。そこで、URLヘルパーを使ってドメインを挿入するよう

にします。URLヘルパーをロードしていない場合はエラーがでますので、autoloadの設定をするか、当該Controller内でロードするようにしておいて下さい。

```
<link rel="stylesheet" href="<?php echo base_url('style.css');?>">
```

これで、CSSを外部ファイルに置くことができました。ちなみに、この「ファイルを公開フォルダに置いて、テンプレートから呼び出す」テクニックは、画像表示であったり、JavaScriptのコードの外部化の時に役立ちます。

処理をユーザHelper/ユーザLibraryで共通化

ここまでで、loadを共通化し、設定を共通化し、Viewのパーツを共通化してきました。最後に、PHPでの動的処理を共通化してみます。

CodeIgniterでは、同じ処理を共通化するための仕組みとして、ユーザHelperとユーザLibarayの機能がありますので、それぞれを紹介していきます。

自作関数を登録できるユーザHelper

ユーザHelperでは、自作関数を登録することができます。

ユーザタスクリストで、データベースから取り出した登録日を「○日前」と表示するためにPHPで日付加工していますが、これをHelperに移動します。ユーザHelperの置き場所は「application/helper」ですので、ここに「interval_helper.php」を設置してコードを移動します。

```
// application/helper/interval_helper.php

if(! function_exists(‘calc’)){
    function calc($created_at)
    {
        $today = new DateTime();
        $created_day = new DateTime($created_at);
        $interval = $today->diff($created_day);
        return $interval->format('%a日前');
    }
}
```

第一引数に日付を入れたら、計算結果が返ってくるHelperができあがりました。「if(! function_exists(‘calc’))」は、すでに「calc」という関数があった場合は、先にある方を使うという設定です。

これをControllerでloadして、Viewで呼び出して利用します。

まずはControllerでloadします。modelとlibraryは前述したとおりautoloadに移行しているので、Task.phpのindex()からはなくなっております。

```
// application/controllers/Task.php

    public function index()
    {
        $this->load->helper("interval_helper");
```

そして、次にViewで当該部分をこのhelperのメソッドcalc()に置き換えます。

```
<?php
if(isset($task_list) and count($task_list) > 0){
    foreach($task_list as $item)
    {
?>
    <tr>
        <td><?php echo html_escape($item['task_name']); ?></td>
         <td>
            <?php echo calc($item[‘created_at']); ?>
           </td>
    </tr>
<?php
        }
    }
?>
```

書き換える前と比べると随分すっきりとしたと思います。本項ではHelperをControllerでloadしておりますが、もちろんautoloadに追加することも可能です。

ユーザLibraryを作って処理を引き渡そう

ユーザLibraryは、自作のClassを登録します。Helperは手続き型関数でしか表現できませんが、ユーザLibraryはオブジェクト指向でコードを共有することができます。本書のサンプルでは、ユーザLibraryを必要とするコードはありませんが、練習のために先程のユーザHelperをユーザLibraryに移してみます。

ユーザLibraryは「application/libraries」にファイルを置くことが決まっていますので、ここに「Inteval_lib.php」を設置します。先程のHelperの中身を少しアレンジして設置します。なお、ユーザLibraryはController,Modelと同様にファイル名、クラス名は大文字ではじめますのでご注意ください。

```
// application/libraries/Interval_lib.php

class Interval_lib
{
    public function calc($created_at)
    {
        $today = new DateTime();
        $created_day = new DateTime($created_at);
        $interval = $today->diff($created_day);
        return $interval->format('%a日前');    }
}
```

class名「Interval_lib」の中に、メソッド「calc」を用意しました。これを先程のTask.phpの自作Helperと置き換えます。なお、load以下で取り回す時には、小文字で表記します。

```
// application/controllers/Task.php

    public function index()
    {
        $this->load->library('interval_lib');
```

また、ViewでHelperで日付操作を行っていた部分をユーザLibraryで日付操作を行うように置き換えます。

```
<?php
if(isset($task_list) and count($task_list) > 0){
    foreach($task_list as $item)
    {
?>
    <tr>
        <td><?php echo html_escape($item['task_name']); ?></td>
         <td>
            <?php echo $this->interval_lib->calc($item['created_at']); ?>
           </td>
    </tr>
<?php
        }
    }
?>
```

これでユーザLibraryを使うことができました。

もちろん、Controller内で利用することが可能ですので、コードリファクタリングする時にぜひお考え下さい。

コラム「バージョン管理をしよう」

welcome.php
welcome2.php
welcome.php_bk
welcome.php_new
welcome.php_old
welcome.php_20150707

……と、念のためなんて理由でコピーして保存したファイルが増えてしまい、うっかり消すこともできないなんて経験ありませんか？誰がなんのために作ったのかわからないファイル。うっかり消せず、なのに上書き保存されて修正したはずのものがなくなった！とかなどなど、ファイル管理を手作業で行うのは煩雑ですよね。

けれど、バージョン管理ソフトを利用すると、この問題は一気に解決されます。バージョン管理ソフトにはいくつか種類があります。その中でもGitは、CodeIgniter本体のソースコード管理に利用されているのをはじめ、多くの現場で導入されておりますので、ぜひプロジェクトに試してもらえればと思います。

Gitでできること

いきなり「バージョン管理」といわれてもぱっとわからないと思うので、Gitを使ったバージョン管理フローを簡単にご紹介します。

1. プロジェクトを登録

最初に、プロジェクトの親フォルダをGitに登録します（レポジトリ）。そうすると、そのフォルダ内の変更をGitが検知できるようになります。

2. プロジェクトの変更を登録

作業を開始すると、当然プロジェクト内のファイルが変更されていきます。そのきりのいいところで「前回登録してから、この作業まで」を登録する作業をします（コミット）。例えば「Aのモジュールドラフト時点」や「コードレビュー完了時点」などで登録するこ

とが多いです。誰が何時にその登録作業を行ったのかが記録されますので、作業進捗管理にも用いることができます。

3. トラブルがあったら、前の登録状況に戻す

プロジェクトの変更を登録した地点は、全て「その地点のファイル一式」を記録しています。ですので、いろいろ変更加えて動かなくなったり、重要なソースコードを消してしまった時などは、きちんと動いた時のファイル一式に戻す（リセット）機能を使います。

4. リリースは、Gitからサーバにファイル同期

Gitを使った作業フローでは、サーバにFTPでファイルアップロードすることはありません。サーバをFTPで操作していたら、足りないファイルがあったり、どれかのファイルを変更して補足できなくなったりするので、Gitからサーバにファイルを同期します（プル）

Gitを利用したサービス

Gitはローカルマシンだけでも運用することは可能ですが、それだと複数人での同時作業やサーバへのファイル同期ができないので、ネット上のGitサーバを利用することが一般的です。しかしながら、わざわざGitサーバを独自で立てるのは手間がかかるので、「Gitホスティングサービス」といわれるサービスがよく使われます。「Gitホスティングサービス」はいくつもありますが、以下の2つが代表的です。

GitHub

一度は聞いたことがあるのではないでしょうか。現在、世界のほとんどのオープンライセンスのソフトウェア開発はGitHub上で行われており、世界中のコミッター（開発者）がGitHub上でソースコードの開発を行っています。CodeIgniterもそのひとつですし、WordPressもGitHub上での開発が行われております。

GitHubはオープンレポジトリ（誰でもソースコードにアクセスできる）だと無料で使えますが、クローズレポジトリ（関係者しかアクセスできない）だと有料です。

- https://github.com/

Bitbucket

GitHubがオープンな開発の場だとすれば、クローズな開発にはBitbucketが用いられることが多いです。Bitbucketは5人までの開発メンバーでしたら、クローズレポジトリを無料で使えるためです。プロジェクト管理機能やSNS機能などはGitHubに一歩劣りますが、Gitをまずはプロジェクトで使ってみようと思うならBitbucketはおすすめです。

- https://bitbucket.org/

Gitを学ぶためのおすすめ書籍

GitHubの教科書

http://www.amazon.co.jp/dp/B00QPSXY1I/
GitとGitHubの入門書。GitHubを利用することを前提に書かれている本です。

Gitが、おもしろいほどわかる基礎の使い方

http://www.amazon.co.jp/dp/4844365010/
BitbucketとSourceTreeを例題にGitを学ぶ良書です。

コラム「もっと生産性を高めるために」

コードが期待通りに動くかをチェックするためには、ブラウザからそのControllerにアクセスして、目視確認するのも一つですし、ちゃんと結果が返ってくるかどうかをSQLを実行して確認するというのも手です。

しかし、このやり方は多くのエンジニアが不安を抱えています。「先月書いたあの処理ちゃんと動いてたっけ」「誰も変更していないよな」「ちゃんと動くかどうかわからないから、えらい人に見せる前に全部実行してみよう」と二重、三重に確認を重ね、その確認だけに多くの時間を要しています（そしてよく確認忘れした箇所に限ってエラーを起こすという……）

しかし、「テストコード」を書きながらコーディングしていくと、テストを実行しただけで全てのコードが実行され、エラーがでるか、期待した値がでてくるかどうか、が自動的にわかりますので、確認作業の時間を大きく短縮することができます。

テストってどういうもの？

PHPのテストは一般的に、PHPUnitというテストフレームワークを用いられて実施されます。PHPで書いたコードを、PHPUnitから実行して、期待している値と実際の値が一致するかを確認します。そして、もしも一致しなかったら、そこのコードが間違っていますよ、とエラーを表示します。

例えば、先程の日付計算の関数があります。

```
function calc($created_at)
{
    $today = new DateTime();
    $created = new DateTime($created_at);
    $interval = $today->diff($created);
    return $interval;
}
```

これは、指定した日にち（$created_at）と本日を比較して何日差があるかを返す関数ですので、このコードをテストする場合は例えば以下のようにします。

```
function calc_test()
{
    $yesterday = new DateTime('yesterday');
    $diff = calc($yesterday->format('Y-m-d H:i:s'));
    $this->assertEquals(1, $diff);
}
```

calcの第一引数に昨日の日付を入れて、今日との比較を行います。この場合、期待する値は1日違いなので、「1」となります。そこで、assertEqualsで、返ってきた値が1であることを確認します。

上記の例は、コード量も少ないですし、多重呼び出しも行っていませんが、ひとつのControllerでいくつものModelを呼び出し、データベースの設計変更を途中で行い、としていると、変更の影響のなかったはずなのになぜかエラーがでる、値が変わっている、ということが起こりえます。
そこで、このようにテストコードを書いておき、常に期待する値が出力されるように監視を行うのです。

CodeIgniterでテストを書く

CodeIgniterでは、独自でテストを書くためのLibraryが用意されています。

- ユニットテストクラス（https://goo.gl/oqgwTN）

しかしながら、テストの範囲が限られるため、一般的につかわれているPHPUnitをCodeIgniterで使おうという流れが一般的になっております。また、CodeIgniterでPHPUnitを実行するためのパッケージも公開されています。

- ci-phpunit-test（https://github.com/kenjis/ci-phpunit-test）

品質と安心感を得るために

テストは、実際のコードがテストの通りに動いていることを担保します。

テスト通りに動いていなければ、テストはエラーとなり想定通りに動いていないことをわかります。書いたコードが少ない間は1つ１つ手作業で実行し確認することができますが、コーディングが進むにつれコードは増えていきます。それらの依存関係を覚えておいてチェックすることは難しいですが、テストコードが用意されていれば、テストを実行するだけで確認は一瞬です。

CodeIgniterを使い始めたばかりだと、まだテストコードを書かなければならないほどの規模を取り扱ってる方は少数でしょう。ですが、今後より生産を高め、開発を進めていこうと思うと、テストコードを書くという技術的選択は有効な打ち手となりますので、概念だけは覚えておいていただければと思います。

終わりに　―コミュニティに参加しよう

CodeIgniterのインストールからMVCを使った実装、コードリファクタリングまでみてきましたがいかがでしたでしょう。「広く浅く」をテーマに執筆しましたので、本書だけでは物足りなく、随所に散りばめられたURL、書籍情報、キーワードで更に学習を進めていくことになろうかと思います。

プログラミングの世界は、年々新たな技術が生まれ、廃れ、気づいたらメジャーだった技術が時代遅れになり、ということを繰り返しています。どのような技術も書籍一冊で完結することはなく、複数にまたがった知見が必要ですし、もっというと書籍執筆中に新たな技術が生まれるので書籍の情報は「一周遅れ」であることすらあります。

ですので、これから実際にコードを書き、アプリをつくっていこうという方は、書籍の世界に閉じこもらずに、積極的に最新情報にアクセスしていただければと思います。

とはいえ、星の数ほどWebサイトがあり、技術ブログが執筆され、玉石混交の世界ですので、最新情報に触れるために、コミュニティに参加することをオススメします。

Twitterに参加しよう

エンジニア業界においてはTwitterは重要なツールです。多くのエンジニアは、コードを書きながらTwitterクライアントを立ち上げて「うごかねー！」「このバグはここを直せばいいんだ！」「おー、動いた！」みたいなツイートを黙々としています。業務中に。

その一方で、自分が一押しのフレームワークを検索キーワードに入れてwatchしつづけて、「動かない」「わからない」というツイートがあったらアドバイスするというユーザも大勢おりますので、「何か困った時はまずTwitterに書く」というのは有効な手法です。

また、そこでコミュニケーションを進めていると、「とりあえず一緒にコード書いてみようか」ということもあったりしますので、ぜひご利用下さい。

勉強会に参加しよう

不定期ではありますが、エンジニア業界では有志で勉強会を開催して、相互に情報交換をしています。前で喋りたい人が喋って、聞きたい人が聞いて、という形ですが、ほとんどの勉強会は新規参加者歓迎ですので、一度参加してみてはいかがでしょうか。

CodeIgniterの場合は、以下のURLで開催予定の勉強会の案内をしておりますし、また大体イベント主催者がツイートしますので、Twitterの検索欄に「CodeIgniter」といれると近くのイベントを知ることができます。

https://goo.gl/OLKU20

ひとりで辛い戦いをするのはやめよう

「1行のコードの間違いに気づくのに、1週間かかった」ということは誰もが通る道です。しかしながら、それをひとりでやり通すのは精神的に辛いことが多く、途中で挫折する原因は大抵「ひとりでやっていた」ことです。

ですので、自宅で引きこもって書籍とにらめっこして作業を進めるのではなく、ぜひTwitterにいろいろ書きなぐってみたり、勉強会にいったりして仲間をつくり、仲間とともに開発を進めてもらえればと思います。

はじめてのPHPフレームワーク

PHP初心者がすぐ現場で活躍するための
CodeIgniter3超入門

2016年10月31日（予定）

執筆	榊原昌彦（一般社団法人リレーションデザイン研究所 代表理事）
テクニカルチェック	宮川貴子（合同会社ねこもり 代表社員）
レビュアー	@hamaco 様　@noldor 様　@nanndemoiikara 様　@ayato_tw 様　@kenji_s 様　46-96 様

本書に掲載されている写真、イラストレーション及び記事の無断転載、使用を禁止します。