

# FuelPHP

## Advent Calendar 2013

**FuelPHP Advent  
Calendar 2013 参加有志**

**25のトピックで  
FuelPHPを極める**

# **FuelPHP Advent Calendar 2013**

**FuelPHP Advent Calendar 2013 参加有志 著**

2013-12-27 版 達人出版会 発行

# はじめに

---

一昨年、去年に引き続き、今年も FuelPHP Advent Calendar を無事に完了することができました。

このイベントは、技術系アドベントカレンダーの一環で、アドベント（待降節）の期間中、参加者が順に 1 日 1 つずつ、FuelPHP に関する記事をブログに書いていくというものです。12 月 1 日から 12 月 25 日までの期間に参加有志により実施されたものです。本書は、この FuelPHP Advent Calendar 2013 の記事を編集し電子書籍としてまとめたものです。

FuelPHP は、日本で人気のあるオープンソース（MIT ライセンス）の PHP フレームワークです。この 1 年で FuelPHP で構築された多くのサイトがリリースされており、日本での実務での利用実績も飛躍的に増え定着した感があります。日本語の情報も非常に増え、CakePHP には及びませんが、他の有名フレームワークに勝るとも劣らない状況だと思います。FuelPHP についてまだよくご存じない方は、以下の公式サイトやまとめ Wiki をご覧ください。

- FuelPHP 公式サイト ... <http://fuelphp.com/>
- FuelPHP まとめ Wiki ... <http://wiki.fuelphp1st.com/wiki/>

去年に引き続き IT 系の電子書籍出版社として定評のある達人出版会のご協力により、本書をタイムリーに出版できることになりました。本書が、IT 系出版社から刊行された FuelPHP に関する 5 冊目の日本語の書籍になります。

本書の内容は、開発環境の構築からレンタルサーバや PaaS での使い方、ユニットテストの書き方、Twig の使い方、RESTful API の作成方法、Composer、MongoDB、Fluentd、ユーザ認証に関するものなど非常に多岐に渡ります。FuelPHP に関する最新の情報を、PC、タブレット、電子書籍リーダー、スマートフォンなどお好きなデバイスでどうぞお楽しみください！

また、本書の読者が来年は FuelPHP Advent Calendar に参加していただけたらと思います。

2013 年 12 月 25 日   FuelPHP Advent Calendar 2013 参加者一同

# 目次

はじめに	i
Day 1 FuelPHP を phar 化してポータブルに！	1
Day 2 FuelPHP の開発環境を 20 分で構築する (Vagrant 編)	8
Day 3 FuelPHP のデータベースマイグレーションを Pagoda Box で使うときの注意	12
Day 4 FuelPHP のエラーハンドリングがなんか今ひとつ物足りなかったのになんとかしてみた話	16
Day 5 FuelPHP で Twig Extension	19
Day 6 JavaScript 側に PHP 変数を簡単にまるごと渡す方法	22
Day 7 FuelPHP 開発でローカルと Web で構造が変わっても対応できる小技	27
Day 8 FuelPHP を更に使ってみて使えるなと思った拡張 ValidationRule の書き方と Core 拡張の小技	30
Day 9 AspectMock で FuelPHP のアプリを 100 %テスト可能にする	35
Day 10 イベント機能を使ってアプリケーションをカスタマイズする	42
Day 11 FuelPHP をもっと Composer で使う	46
Day 12 FuelPHP で ChatWork パッケージを使ってみる	51
Day 13 FuelPHP (Twitter Bootstrap 3) で jQuery のプラグインの DataTables を使う	54
Day 14 Request_Curl にまつわるエトセトラ	58
Day 15 続・Cloudn PaaS で FuelPHP を動かしてみた	61
Day 16 FuelPHP の module を使いこなす	67
Day 17 レンタルサーバー XREA/CORESERVER で FuelPHP を使う (実践編)	70
Day 18 FuelPHP と MongoDB と TraceKit で JavaScript のエラー情報を収集してみる	76
Day 19 FuelPHP 5 分で API を実装するチュートリアル (スクリーンキャストあり)	80
Day 20 FuelPHP と Fluentd を連携させてみる	85
Day 21 FuelPHP を Rocketeer で自動デプロイしてみる。マイグレーションと PHPUnit も実行してみる。	87
Day 22 FuelPHP が OAuth 対応になったので facebook ログインを試してみる	97
Day 23 Heroku (PaaS) で FuelPHP 環境 (PHP 5.3 + MySQL + Apache) を構築する	105

<b>Day 24</b>	<b>本当は怖い FuelPHP 1.6 までの Rest コントローラ</b>	<b>112</b>
<b>Day 25</b>	<b>Auth と他モデルにリレーションをつける</b>	<b>116</b>

## FuelPHP を phar 化してポータブルに！

FuelPHP Advent Calendar 2013 の 1 日目の参加記事です。

初めましての方もご存知の方も、よろしくお願いします。

@sharkpp です。

さて、昨年の 12 月 1 日はアドベント（待降節）ではありませんでしたが、安心してください、今年は 12 月 1 日からアドベントは始まります。

とりあえず、初日なので軽い内容でいきたいと思います。

内容は、FuelPHP を phar（PHP Archive）で 1 ファイルにしてウェブサーバーで動かしてみよう、です。

子ネタをやりつつ phar の紹介も兼ねています。

環境としては、

- PHP 5.3 以上
- FuelPHP 1.7
- Apache on CentOS or Windows

を想定しています。

### phar（PHP Archive）ってご存知ですか？

まず、大前提。

PHP マニュアル（<http://php.net/manual/ja/intro.phar.php>）によると、

phar 拡張モジュールは、PHP アプリケーション全体をひとつの"phar"（PHP Archive）ファイルにまとめてしまい、配布やインストールを容易にするためのものです。

となっています。

実際に使われている例としては、

- [composer.phar](#) ... パッケージ管理ツール
- [goutte.phar](#) ... スクレイピングライブラリ
- [guzzle.phar](#) ... HTTP クライアントライブラリ
- [pyrus.phar](#) ... PEAR2

などがあります。

例として挙げた中でも composer は FuelPHP を使っている方であれば

```
$ php composer.phar update
```

と、このような形で触ったことがあると思います。

## FuelPHP をインストール

Phar クラスの中でも、今回は Phar::webPhar (<http://php.net/manual/ja/phar.webphar.php>) を使います。

まずは、FuelPHP を適当なフォルダに配置します。詳しい手順は FuelPHP ドキュメント (<http://fuelphp.jp/docs/1.7/>) に書かれているので参考にしてください。ここでは、~/fuelphp-1.7 に配置されるものとします。

```
$ curl get.fuelphp.com/oil | sh
$ cd ~
$ oil create fuelphp-1.7
```

もしくは、

```
$ wget http://fuelphp.com/files/download/25 -O fuelphp.zip
$ unzip fuelphp.zip
```

とすることで、git がインストールされていない場合は fuelphp.com からダウンロードして展開ができます。  
次に

```
$ cd fuelphp-1.7
$ php composer.phar self-update
$ php composer.phar update
```

として、composer 自身のアップデートとパッケージを更新します。

これで、Apache などのウェブサーバー上に公開すると Welcome 画面が表示されるはずです。

## FuelPHP を Phar で 1 ファイルにまとめる

まず、そのままでは 1 ファイルにまとめても動かないのでいくつかソースを変更する必要があります。

残念なことに core の中も変更する必要がありました。

インストール直後のページを表示できるようにするために変更するファイルは

- public/index.php
- fuel/app/config/config.php
- fuel/app/config/asset.php ※ fuel/core/config/asset.php からコピー
- fuel/core/bootstrap.php
- fuel/core/classes/file/area.php

の 5 個のファイルです。

実際のアプリケーションの場合は先に挙げたファイル以外にも変更が必要になると思います。

変更のポイントは、

- phar 内からの realpath が常に空文字で返ってくるのでダミー関数に置き換え
- Windows であっても パスの区切りは '/' とする
- パスに含まれる親ディレクトリへの移動などを削除し正規化
- ログやキャッシュの保存先が .phar 外を示すようにする

と、主に、ファイルパスに関する物が主となります。

まず、public/index.php の変更部分です。

パスを正規化する canonicalizePath 関数と realpath 関数のダミーとして realpat\_関数を定義しています。

```
error_reporting(-1);
ini_set('display_errors', 1);

+function canonicalizePath($path) {
+    $path = 0===strpos($path, 'phar://')?'phar://'.preg_replace('!//!', '/', substr($path,7))
+                                   :preg_replace('!//!', '/', $path);
+    do {
+        $tmp = $path;
+        $path = preg_replace('!/[^\w+\.\.\/!]', '/', $tmp);
+    } while ($tmp != $path);
+    return rtrim($path, '/');
+}
+
+function realpat_($path) {
+    return canonicalizePath(str_replace(array('/', '\\'), '/', $path));
+}
```

あとは、realpath 関数の代わりに realpat\_関数を使うようにし、パスの区切りも '/' に変更しています。

```
-define('DOCROOT', __DIR__.DIRECTORY_SEPARATOR);
+define('DOCROOT', realpat_(__DIR__.'/'));
```

```
-define('APPPATH', realpath(__DIR__.'/../fuel/app/').DIRECTORY_SEPARATOR);
+define('APPPATH', realpat_(__DIR__.'/../fuel/app/').'/');
```

```
-define('PKGPATH', realpath(__DIR__.'/../fuel/packages/').DIRECTORY_SEPARATOR);
+define('PKGPATH', realpat_(__DIR__.'/../fuel/packages/').'/');
```

```
-define('COREPATH', realpath(__DIR__.'/../fuel/core/').DIRECTORY_SEPARATOR);
+define('COREPATH', realpat_(__DIR__.'/../fuel/core/').'/');
```

fuel/app/config/config.php の変更部分です。.phar 内には書き込めないで.phar と同じ場所の writable ディレクトリを示すように変更しています。

保存先は公開ディレクトリ外を示すべきなので、さらに 1 つ上などに示すようにするのが本来は良いでしょう。

```
-    // 'cache_dir'          => APPPATH.'cache/',
+    'cache_dir'            => canonicalizePath(str_replace('phar://', '', APPPATH).'../../writable/cache/'),
```



```
- // 'log_path'          => APPPATH.'logs/',
+ 'log_path'            => canonicalizePath(str_replace('phar://', '', APPPATH).'../../../writable/logs/'),
```

fuel/app/config/asset.php の変更部分です。fuel/core/config/asset.php をコピーして使うのでそのファイルとの比較になります。一部、三項演算子を使っていますが、phar でまとめない場合にもそのまま動くようにとの苦肉の策です。

```
- 'paths' => array('assets/'),
+ 'paths' => array(DOCROOT . 'assets/'),
```

```
- 'url' => Config::get('base_url'),
+ 'url' => Config::get('base_url').(0===strpos(__DIR__, 'phar://')?'index.phar/':''),
```

```
- 'add_mtime' => true,
+ 'add_mtime' => false,
```

fuel/core/bootstrap.php の変更部分です。パスの区切りの変更と関数の置き換えです。

```
-define('DS', DIRECTORY_SEPARATOR);
+define('DS', '/');

-defined('VENDORPATH') or define('VENDORPATH', realpath(COREPATH.'../'.DS.'vendor').DS);
+defined('VENDORPATH') or define('VENDORPATH', realpat_(COREPATH.'../'.DS.'vendor').DS);
```

最後、fuel/core/classes/file/area.php の変更部分です。

```
- {
+     $this->basedir = realpath($this->basedir) ?: $this->basedir;
+     $this->basedir = realpat_($this->basedir) ?: $this->basedir;
+ }
```

```
- {
+     $pathinfo['dirname'] = realpath($pathinfo['dirname']);
+     $pathinfo['dirname'] = realpat_($pathinfo['dirname']);
+ }
else
{
    // attempt to get the realpath(), otherwise just use path with any double dots taken out when
    basedir is set (for security)
-     $pathinfo['dirname'] = ( ! empty($this->basedir) ? realpath($this->basedir.DS.$pathinfo['dirname'])
: realpath($pathinfo['dirname']) )
+     $pathinfo['dirname'] = ( ! empty($this->basedir) ? realpat_($this->basedir.DS.$pathinfo['dirname'])
: realpat_($pathinfo['dirname']) )
```

```
?: ( ! empty($this->basedir) ? $this->basedir.DS.str_replace('..', '', $pathinfo['dirname'])
: $pathinfo['dirname']);
```

1 つ 1 つ編集するのが大変であれば Gist の <https://gist.github.com/sharkpp/7716098> に差分をアップしたので

```
$ cd fuelphp-1.7
$ wget -q https://gist.github.com/sharkpp/7716098/raw -O - | patch -u -p0
```

とすることで変更を適用することができます。

次は、phar の生成スクリプトです。

```
<?php
/*
 * Copyright (c) 2013 sharkpp
 * This software is released under the MIT License.
 * http://opensource.org/licenses/mit-license.php
 */

// 確実に削除
@unlink('index.phar');
// phar 書庫作成のためクラスを生成
$phar = new Phar(__DIR__ . '/index.phar', 0, 'index.phar');
// fuelphp1.7 ディレクトリ丸ごと固める
$phar->buildFromDirectory(__DIR__ . '/fuelphp-1.7/');
// gzip で圧縮
// $phar->compressFiles(Phar::GZ); // ※ css などがうまく取り出せない
// 起動スタブを設定
$phar->setStub(<<<'EOD'
<?php
    function phar_rewrites($path) {
        if (0 === strpos($path, '/assets/'))
            return '/public' . $path; // assets だけはパスを変更
        return '/public/index.php' . $path; // あとはすべて index に渡す
    }
    Phar::interceptFileFuncs();
    Phar::webPhar('index.phar', 'public/index.php', '', array(), 'phar_rewrites');
    __HALT_COMPILER(); ?>
EOD
);
```

FuelPHP をインストールした fuelphp-1.7 ディレクトリの上にファイルを保存してください。

こちらも Gist の <https://gist.github.com/sharkpp/7716423> にアップしてあるので、

```
$ cd ~
$ wget -q https://gist.github.com/sharkpp/7716423/raw/mkphar.php
```

として、ローカルに保存できます。

準備ができたら

```
$ php mkphar.php
```

と入力して、index.phar を作成すると、70MB ぐらいのファイルが出来上がります。

ドキュメントや.git などが含まれているので巨大になってしまいました。

ちなみに、**Phar** クラスでアーカイブを作成するには設定を変える必要があるかもしれません。

具体的には、php.ini の Phar セクション内で phar.readonly = Off と設定されている必要があります。

## ブラウザで確認

ここまでできたら index.phar をウェブサーバーの公開フォルダに置きましょう。

と、その前に、AddType で.phar を php で実行できるように.htaccess を設置しましょう。

```
Options +FollowSymLinks
DirectoryIndex index.phar
AddType application/x-httpd-php .phar

<IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    <IfModule mod_fcgid.c>
        RewriteRule ^(.*)$ index.phar?/$1 [QSA,L]
    </IfModule>
    <IfModule !mod_fcgid.c>
        <IfModule mod_php5.c>
            RewriteRule ^(.*)$ index.phar/$1 [L]
        </IfModule>
        <IfModule !mod_php5.c>
            RewriteRule ^(.*)$ index.phar?/$1 [QSA,L]
        </IfModule>
    </IfModule>
</IfModule>
```

こちらも例によって Gist の <https://gist.github.com/sharkpp/7718075> にアップしてあるので、

```
$ wget -q https://gist.github.com/sharkpp/7718075/raw/.htaccess
```

で取得できます。

例えば、ローカルホストでウェブサーバーを動かしていてドキュメントルートに先の.htaccess と共に置いたのであれば、

```
http://127.0.0.1/
```

にブラウザでアクセスすると Welcome 画面が表示されます。

```
http://127.0.0.1/hello
```

にアクセスすると hello と表示されます。

```
http://127.0.0.1/xxxx
```

エラーページも表示できます。

## まとめ

お遊びのつもりで手を出してみたら、かなり時間をかけないとうまくいかなかったりで当てが外れてちょっとションボリ。

実際問題として core の修正が必要となるので実用性となると皆無だと思います。

ただ、1 ファイルでウェブサーバーにアプリが公開できるのは、うまく作れば面白いことが出来るのではないかと期待が持てそうな機能でした。

### @sharkpp

へっぽこプログラマ。手羽先が有名な所に在住な PHP と C++ のプログラマです。

Twitter: [@sharkpp](https://twitter.com/sharkpp)

Blog: <http://www.sharkpp.net/>

# FuelPHP の開発環境を 20 分で構築する (Vagrant 編)

FuelPHP Advent Calendar 2013 の 2 日目です。

従来は、FuelPHP の開発環境を構築する場合、XAMPP や MAMP を使う方法が一般的でした。この方法は簡単に手許の PC に開発環境を構築でき便利なのですが、開発環境と本番環境の PHP のバージョンが異なったり、開発環境は Windows や Mac だが本番環境は Linux であったりと、ほとんどの場合、本番環境と開発環境が異なるという問題がありました。

PHP のポータビリティはかなり高いので、多くの場合、実際には問題は生じませんが、ファイル名の太文字小文字の違いやパーミッション、PHP のバグなど、本番環境だけで問題が発生するということも可能性としてはあります。

この問題を解消するには、開発環境と本番環境をできる限り一致させることが望ましいです。そこで、本番と同じような仮想マシンを作成し、そこで開発をすれば開発環境と本番環境をほとんど一致させることが可能になります。

本日は、VirtualBox と Vagrant を使って、FuelPHP のためのそのような開発環境を簡単に作成する方法を解説します。

なお、この方法のデメリットは、仮想マシンを維持するためのリソースが余計に必要となることです。OS からまるごとインストールするわけですから、ハードディスクの容量もその分多く必要になりますし、仮想マシン実行のオーバーヘッドがありますので、実効速度もネイティブで動作している Web サーバ／PHP より、多くの場合、遅くなるでしょう。

## VirtualBox と Vagrant のインストール

フリーな仮想化ソフトウェアである VirtualBox を、以下からダウンロードしインストールします（執筆時の動作確認バージョンは VirtualBox 4.3.2）。

- <https://www.virtualbox.org/wiki/Downloads>

VirtualBox の仮想イメージを操作するツールである Vagrant を、以下からダウンロードしインストールします（執筆時の動作確認バージョンは Vagrant 1.3.5）。

- <http://downloads.vagrantup.com/>

これで準備は完了です。

## 仮想マシンの作成

FuelPHP のプロジェクト用のフォルダを作成し、その中に Vagrant 用のファイルを配置します。

```
$ mkdir fuelphp
$ cd fuelphp/
$ git clone git@github.com:kenjis/vagrant-fuelphp-centos6.git
$ cd vagrant-fuelphp-centos6/
$ git submodule update --init --recursive
```

仮想マシンを構築します。

```
$ vagrant up
```

初回は CentOS6 の仮想イメージをダウンロードするため、かなり時間がかかりますので気長に待ちます（このダウンロード時間を含めると 20 分で開発環境を構築するのは無理です）。

これで、仮想マシンが作成され、FuelPHP の開発に必要なサーバなどもインストール設定されます。

FuelPHP がインストールされていない場合は、`oil create` コマンドでインストールされます。

これで、<http://localhost:8000/> にアクセスすれば、おなじみの FuelPHP の Welcome ページが表示されます。

## ディレクトリ構成

ホスト（手許の PC）側

```
fuelphp/ (FuelPHP プロジェクトのトップ)
├── docs
├── fuel
├── public
└── vagrant-fuelphp-centos6
```

ゲスト（仮想マシン）側

```
/mnt/fuelphp/
├── docs
├── fuel
├── public
└── vagrant-fuelphp-centos6
```

仮想マシンから、ホスト側の `fuelphp` フォルダを共有しているので、ホスト側から好きなエディタでソースを変更すれば、仮想マシンに自動的に反映されます。

## テストの実行

`vagrant-fuelphp-centos6` フォルダから、`vagrant ssh` コマンドで仮想マシンに SSH で接続できます（Windows を除く）。

```
$ cd fuelphp/vagrant-fuelphp-centos6/
$ vagrant ssh
Last login: Mon Dec  2 01:11:37 2013 from 10.0.2.2
Welcome to your Vagrant-built virtual machine.
```

ホームディレクトリにシンボリックリンクが張ってあるので、`/mnt/fuelphp` フォルダには `~/fuelphp` でアクセスできます。  
`oil` コマンドと `phpunit` もインストール済みなのですぐに実行できます。

```
[vagrant@localhost ~]$ cd fuelphp/
[vagrant@localhost fuelphp]$ oil test --group=Core
Tests Running...This may take a few moments.
PHPUnit 3.7.28 by Sebastian Bergmann.
```

```
Configuration read from /mnt/fuelphp/fuel/core/phpunit.xml

..... 63 / 361 ( 17%)
..... 126 / 361 ( 34%)
..... 189 / 361 ( 52%)
..... 252 / 361 ( 69%)
..... 315 / 361 ( 87%)
.....

Time: 7.92 seconds, Memory: 19.25Mb

OK (361 tests, 413 assertions)
```

## サーバ環境

vagrant-fuelphp-centos6 で作成される仮想マシンのサーバ環境は以下のようになっています。

- メモリ 480MB
- HDD 200GB
- OS CentOS 6.4 (64bit)
- Apache 2.2.15-29.el6.centos.x86\_64
- MySQL 5.1.69-1.el6\_4.x86\_64
- PHP 5.4.21-2.ius.el6.x86\_64
- phpMyAdmin 3.5.8.2-1.el6.noarch
- PHPUnit 3.7.28

ホスト側のポート 8000 が仮想マシンのポート 80 に転送されるようになっています。仮想マシンに直接アクセスする場合は、<http://192.168.33.33/>にアクセスします。

ホストの FuelPHP のプロジェクトのフォルダが仮想マシンの /mnt/fuelphp にマウントされるようになっています。

MySQL データベースは、fuel\_dev と fuel\_test が作成されており、root のパスワードは root です。

また、<http://localhost:8000/phpmyadmin/>から、phpMyAdmin にアクセスできます。

## 仮想マシンの起動と停止

仮想マシンの停止は、vagrant-fuelphp-centos6 フォルダに移動して、

```
$ vagrant halt
```

とします。vagrant suspend コマンドを実行すれば、仮想マシンをシャットダウンせずに状態を保存したまま停止できます。  
仮想マシンの起動は、

```
$ vagrant up
```

仮想マシンを破棄するには、

```
$ vagrant destroy
```

とします。

### その他の Vagrant 環境

今回の開発環境は CentOS 6.4 を使ったものですが、Ubuntu 12.04 (precise64) の場合は、以下の vagrant-fuelphp が公開されています。

- <https://github.com/iturgeon/vagrant-fuelphp>
- [vagrant-fuelphp を使って FuelPHP の開発環境を構築する](#)

また、PHP の開発環境として以下にまとめがあり、参考になります。

- [PHP の開発に使える Vagrantfile のまとめ](#)

### 関連

- [Vagrant を使って FuelPHP の開発用の CentOS 6.4 を構築する](#)
- [Vagrant で VirtualBox の仮想マシンのウィンドウを表示させる](#)

#### kenjis

FuelPHP まとめ Wiki 管理人。「PHP5 技術者認定上級試験」認定者。

Twitter: [@kenji\\_s](#)

Blog: <http://blog.a-way-out.net/>



# FuelPHP のデータベースマイグレーションを Pagoda Box で使うときの注意

FuelPHP Advent Calendar 2013 の 3 日目です。

今日は、FuelPHP のデータベースマイグレーションを Pagoda Box で使う時にハマるポイントについて書きたいと思います。テーマがニッチ過ぎて、同じようなことで困っている人が他にもいるのか少し心配ではありますが、気にせず行きたいと思います。

## Pagoda Box とは

まず、Pagoda Box ってなんや？ というはなしを。

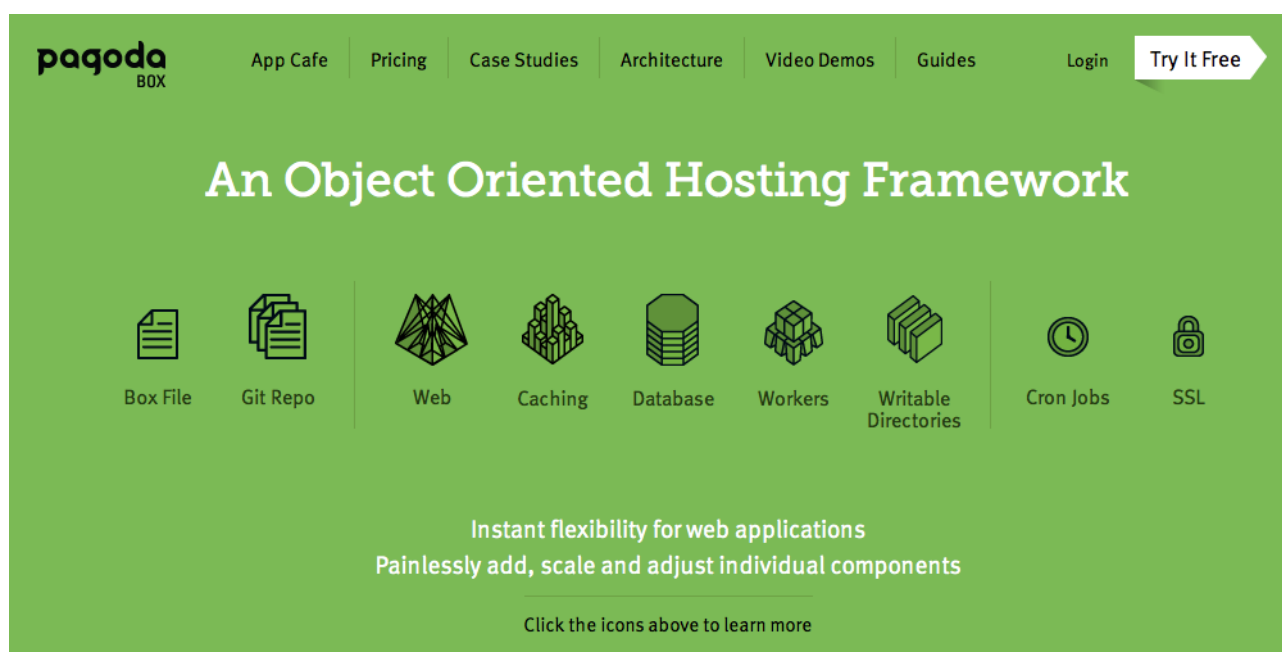


図 3.1 Pagoda Box (<http://pagodabox.com>)

Pagoda Box は、LAMP 環境を構築できる PHP 専門の PaaS です。

PHP が使える PaaS はいくつかありますが、Pagoda Box は PHP に特化した PaaS なので、PHP エンジニアにはとてもなじみやすく、僕は結構気に入ってよく使っています。ちなみに、FuelPHP の公式サイトも Pagoda Box 上で動いているようです。

なお、Pagoda Box については、この間発売されたこの本に書きました。みなさん、よければ読んでみてください。



図 3.2 『PHP エンジニア養成読本』

『PHP エンジニア養成読本（現場で役立つイマドキ開発ノウハウ満載!）（Software Design plus）』

技術評論社

（著）新原 雅司、原田 康生、小山 哲志、田中 久輝、保科 一成、大村 創太郎、増永 玲

（編集）PHP エンジニア養成読本編集部

## FuelPHP データベースマイグレーションのおさらい

続いて、FuelPHP でのデータベースマイグレーションの手順を簡単におさらいしておきましょう。

公式サイトの説明は、<http://fuelphp.com/docs/general/migrations.html> です。

### oil generate model

\$ oil generate model xxx で Model クラスと、マイグレーション用のクラスが一気に作られます。僕は好みで、--crud を使うことが多いです（というか crud しか使いません）。

```
% php oil generate model post id:int name:varchar message:text created_at:datetime --crud --mysql-timestamp
Creating model: /Users/omoon/Documents/www/speak_on_fuelphp/fuel/app/classes/model/post.php
Creating migration: /Users/omoon/Documents/www/speak_on_fuelphp/fuel/app/migrations/001_create_posts.php
```

で、つくられる Model クラス。

```
// fuel/app/classes/model/post.php
class Model_Post extends \Model_Crud
{
    protected static $_properties = array(
        'id',
        'name',
        'message',
        'created_at'
    );
    protected static $_mysql_timestamp = true;
    protected static $_table_name = 'posts';
}
```

つくられるマイグレーションクラス。

```
// fuel/app/migrations/001_create_posts.php
namespace Fuel\Migrations;
class Create_posts
{
    public function up()
    {
        \DBUtil::create_table('posts', array(
            'id' => array('constraint' => 11, 'type' => 'int'),
            'name' => array('constraint' => 255, 'type' => 'varchar'),
            'message' => array('type' => 'text'),
            'created_at' => array('type' => 'datetime'),

        ), array('id'));
    }
    public function down()
    {
        \DBUtil::drop_table('posts');
    }
}
```

## oil r migrate

で、\$ oil r migrate で該当テーブルがデータベースに作成されます。この際、

- データベース上に migration というテーブル
- fuel/app/config/FUEL\_ENV/migrations.php というファイル (FUEL\_ENV は環境)

が作成され、マイグレーションの状況を管理するしくみになっています。

## 何が問題なのか

このしくみを使って、Pagoda Box へアプリケーションをデプロイしたタイミングでデータベースの初期化まで一気にやってしまいたいのですが、以下の問題がありうまくいかないということがわかりました。

- \$ oil r migration 実行時に fuel/config/production/migrations.php ファイルが生成されるので、fuel/config/production/を writable にする必要がある
- Pagoda Box では、writable ディレクトリは、ソースコードレポジトリとは別に、ネットワーク上にマウントされる
- そのため、同じディレクトリにあるデータベースへの接続ファイル fuel/config/production/db.php が消える
- データベースへの接続ができなくなる

## 解決方法

- fuel/app/config/pagoda/db.php に別ファイルで Pagoda Box 設定ファイルを用意しておく
- before\_deploy フックで、fuel/app/config/pagoda/db.php ファイルを fuel/app/config/production/配下へコピーする

という作戦で解決です。

参考までに Boxfile 例。

```
global:
  env:
    - FUEL_ENV: production
db1:
```

```
type: mysql
name: speak
web1:
  shared_writable_dirs:
    - /fuel/app/cache
    - /fuel/app/logs
    - /fuel/app/tmp
    - /fuel/app/config/production # <- migrations.php が作られるため writable に
document_root: public
php_version: 5.4.14
php_date_timezone: "Asia/Tokyo"
php_extensions:
  - pdo_mysql
  - zip
after_build:
  - "curl -s http://getcomposer.org/installer | php"
  - "php composer.phar install"
before_deploy:
  # production とは別のディレクトリにおいておいた db.php をコピー
  - "cp fuel/app/config/pagoda/db.php fuel/app/config/production/db.php"
  - "php oil r migrate"
```

fuel/app/config/pagoda.db.php はこうなります。

```
return array(
    'default' => array(
        'connection' => array(
            'dsn' => 'mysql:host=' . $_SERVER['DB1_HOST'] . ';port=' . $_SERVER['DB1_PORT'] . ';dbname=' .
$_SERVER['DB1_NAME'],
            'username' => $_SERVER['DB1_USER'],
            'password' => $_SERVER['DB1_PASS'],
        ),
    ),
);
```

これで、\$ git push pagoda で一気にアプリケーションのデプロイからデータベースの初期化までできるようになります。  
それでは、また。

### omoon

40 代、縄文系。大阪を拠点に PHP などを使って仕事をしています。Kansai PHP Users Group スタッフ。

Twitter: [@omoon](#)

Blog: <http://blog.omoon.org/>

# FuelPHP のエラーハンドリングがなんか今ひとつ物足りなかったの でなんとかしてみた話

FuelPHP Advent Calendar 2013 です。もう 2013 年ですね。早いですね。そうこうしているうちに 2014 年になります。なんとも恐ろしい。

思えば今年も FuelPHP でした。もうこの子しか愛せなさすぎて辛い。ポリアモリーを自称する割にはこういう所は変に一途だったりするのです（あとはまあ眼鏡とか時計とかカバンとか）。

そういえば去年は何書いたかなあ…と思って FuelPHP Advent Calendar 2012 を見に行ったら、自分の担当のリンクだけ「お探しのページは見つかりません」。というわけで本日は 404 のお話です。

FuelPHP のエラーハンドリングは何かと複雑です。便利機能が却って便利じゃなかったりとか、公式ドキュメントが index.php 書き換えたらイイヨ!!とか、もうなんかしまったかめっちゃかな状況ですが、ざっと以下のような流れになっているみたいです。

- リクエストの処理中にコントローラレベルで catch されない例外が発生!!
- HttpNotFoundException 系列のみ index.php にてキャッチ
- [index.php] config/route に \_404\_ 設定アレば、そちらで処理
- [index.php] 無かったらそのまま例外横流し
- 今度はエラーハンドラ (Error::exception\_handler) がキャッチ
- [エラーハンドラ] 例外オブジェクトが handle メソッド持ってたならそちらで処理
- [エラーハンドラ] handle が無くて本番環境なら、errors/production をレンダする。
- [エラーハンドラ] handle が無くて本番でも無ければ、errors/php\_fatal\_error をレンダする。

errors/production とか errors/php\_fatal\_error は core に入ってる方の View。同名ファイル作れば app 側で上書きできる。production ってのはあの「Oops!」ってやつで、php\_fatal\_error ってのはあの開発中に便利なバックトレースとかつけてくれる例外画面。

んで、ドキュメントを見るとなやら HttpNotFoundException って例外を投げるとエラー画面を描画してくれる、とかある。これは core に入ってる HttpNotFoundException って例外クラスに例の handle メソッドが実装されているからそうなるわけで、描画される View は views/404.php になる。

## やりたいこと

CodeIgniter のエラーハンドリングでもそうなんだけど、実際アプリケーションの開発では 1 システム 1 エラー画面というわけには行かない。

PC 版のリクエストなら PC 版の 404 を出したいし、SP 版のリクエストなら SP 版の 404 を出したい。もっと言うなら API のリクエストでは JSON 形式の 404 メッセージを送りたい（はず）。

CodeIgniter 使ってた時にはエラーの View 内で条件分岐して、リクエストの種別 (PC/SP) 毎にそれぞれ表示する View 書き換えたりしてたので、それを応用しながら上手いこと出来ないかなあとか考えてたら以下のような形になりました。

```
/*
 * Copyright (c) 2013 mkkn.info
 * This software is released under the MIT License.
```

```

* http://opensource.org/licenses/mit-license.php
*/

class HttpNotFoundException extends \Fuel\Core\HttpNotFoundException
{
    public function response()
    {
        Fuel::$profiling = false;
        // デフォルト 404 出力の定義
        //$response = Response::forge(View::forge('404'), 404); // デフォルト
        $response = Request::forge("top/404")->execute()->response(); // デフォルト

        $req = Request::forge();
        $req->action = "404";
        try {
            $response = $req->execute()->response();
        } catch (Exception $e) {
            // 何もしない
        }
        $response->set_status(404);
        return $response;
    }
}

```

各リクエストの URL 形式からコントローラの検出まで可能であればそのコントローラに `action_404` が存在しないかをチェックし、あればそれを出力する、なければデフォルトの 404 出力を返すというものです。

`index.php` はいじらずにすみませんが、`core` の `HttpException` を上書きするので `bootstrap` に以下のような記述が必要になります。

```

Autoloader::add_classes(array(
    // Add classes you want to override here
    // Example: 'View' => APPPATH.'classes/view.php',
    'HttpException' => APPPATH.'classes/httpnotfoundexception.php' // 上記クラスを記述した場所
));

```

クラス名が `_` の全くない長い名前なので律儀に FuelPHP の命名規則に従う必要はないです。上記 `bootstrap` の記述でしっかりパス指定さえ行えばどこに置いてもオートロードします。

## 解説

```
Fuel::$profiling = false;
```

これを入れておかないとプロファイラが先に出力されて、HTML がおかしくなる。端的に言うと文字が化ける。

```

// デフォルト 404 出力の定義
//$response = Response::forge(View::forge('404'), 404); // デフォルト
$response = Request::forge("top/404")->execute()->response(); // デフォルト

```

コメントアウトしてある上の行は `View` ファイルを直接指定するタイプのデフォルト 404 指定。コアの `HttpException` の挙動に近い形。その下の行は `Uri` で直接指定するタイプ。`config/router.php` の `_404_` の書き方に近い形。それぞれ好きな方をご利用ください。

```
$req = Request::forge();
$req->action = "404";
try {
    $response = $req->execute()->response();
} catch (Exception $e) {
    // 何もしない
}
```

例外の処理内部でリクエストを再生成して、action だけ書き換えてからリトライする形。try catch しとかないと多重ループする気がする。もともと action\_404 に対してのダメ元リトライなので catch してもとくにすることはない。

## あとがき

Controller ごとにエラーハンドリングしたいなーって思いは結構前からあったのですが、そこまでしっかりした開発を FuelPHP でやる機会もしばらく無く、ようやくこの機会に着手する事が出来ました。

正直 views/404.php が着地点なんだから、そこから Request::active() なり Request::main() なりで Controller のインスタンス引っ張ってきて action\_404 のコールをトライしたら終わりじゃね？ くらいの軽い気持ちだったのですが、よくよく処理を追いかけてみると、バッチリ reset\_request なるメソッドが張り巡らされており、404.php はおろか、HttpException から Controller のインスタンスは取得できませんでした。まさかコントローラを再生成するはめにはなるとは…

FuelPHP 的なコントローラは基本的に生成コスト低め、のはずなので問題無いとは思いますが、ファットなコントローラで生成コスト高め（特に before がごちゃごちゃしてる…）の時には、処理負担的にあまりオススメできませんが、参考になれば幸いです。

**@mkkn\_info**

mikakane です。

Twitter: [@mkkn\\_info](#)

Blog: <http://mkkn.hatenablog.jp/>

## FuelPHP で Twig Extension

4 日目の@mkkn\_info さんの「Fuelphp のエラーハンドリングがなんか今ひとつ物足りなかったのでなんとかしてみた話 - どうにもならない日々@mkkn」に引き続き、FuelPHP Advent Calendar 2013 の 5 日目です。

ここ数年はアドベントカレンダーの時にしか技術的な内容を書いていない気がするのが恐ろしいところですが、気にせずいきましょう。

### FuelPHP の Parser パッケージ

FuelPHP は、基本的にはビューに生の PHP スクリプトを使うことになっていますが、標準バンドルされている Parser パッケージを用いることで、様々なテンプレートエンジンを用いることができます。現在サポートされているエンジンは以下の通り。

- Twig
- Mustache
- Markdown
- Dwoo
- Jade
- Haml
- MitHaml
- Smarty
- PHPTAL

このうち自分では Twig を愛用しています。何か機能を追加するにも簡単にできるところが良いですね。

### Parser パッケージが標準で用意してくれる FuelPHP 向け Extension

Parser パッケージで Twig を使用すると、Uri、Config、Form、Input、Html、Asset などの便利そうな Fuel core のメソッドを、あらかじめ Twig Extension としてロードしてくれます。これを行っているのは

```
fuel/packages/parser/classes/twig/fuel/extension.php
```

にある Parser\Twig\_Fuel\_Extension クラスで、これ自体も標準的な Twig Extension です。

これのおかげで、例えば

```
Asset::js('hoge-hoge.js');
```

を呼びたい場所では

```
{{ asset_js('hoge-hoge.js') }}
```



と書くことができるわけです。

## アプリ独自の Twig Extension を使う

とはいえ、ただ単に Twig を使って HTML テンプレートを書くだけではなく、アプリケーション独自の Twig Extension をがし登録して使いこなしてこそ Twig の便利さが際立つというもの。早速やってみましょう。

独自の Twig Extension を登録するには、まず Twig\_Extension クラスを継承したクラスを作ります。クラス名は他とぶつからなければ何でも良いですが、ここでは Hoge アプリ向けに Hoge\_Twig\_Extension という名前にすることにしましょう。FuelPHP のファイル名規則に則り以下の場所に作ります。

```
fuel/app/classes/hoge/twig/extension.php
```

中身はこんな感じ。

```
<?php
/*
 * Copyright (c) 2013 KOYAMA Tetsuji
 * This software is released under the MIT License.
 * http://opensource.org/licenses/mit-license.php
 */

class Hoge_Twig_Extension extends Twig_Extension
{
    /**
     * Gets the name of the extension.
     *
     * @return string
     */
    public function getName()
    {
        return 'hoge';
    }

    /**
     * Sets up all of the functions this extension makes available.
     *
     * @return array
     */
    public function getFunctions()
    {
        return array(
            new Twig_SimpleFunction('swap_empty', array($this, 'swapEmpty')),
        );
    }

    /**
     * Sets up all of the filters this extension makes available.
     *
     * @return array
     */
    public function getFilters()
    {
        return array(
            new Twig_SimpleFilter('json', 'json_encode'),
        );
    }

    public function swapEmpty($value)
    {
        return empty($value)? '-' : $value;
    }
}
```

ここでは Twig の関数とフィルターを 1 つずつ登録しています。

### swap\_empty 関数

もし引数が empty() で真だったら「-」を出力、そうでなければそのまま。

### json フィルター

引数を PHP の json\_encode に渡した結果を出力。

テンプレート上では以下のように使います。

```
{# 変数の設定、本来は PHP 側から渡される #}
{% set foo = 0 %}
{% set bar = {'fuga': 'hoge', 'move': 'puge'} %}

{{ swap_empty(foo) }}
{{ bar|json }}
```

ただファイルを置いただけでは Parser パッケージはその Extension の存在を知らないので、Parser の config を通して教えてやります。FuelPHP の Config は大変に賢くて、追加・変更したい部分だけ app 以下に書けば良いので、

```
fuel/app/config/parser.php
```

に以下の内容を記述します。

```
<?php
return array(
    'View_Twig' => array(
        'extensions' => array(
            'Hoge_Twig_Extension',
        ),
    ),
);
```

Twig は非常に柔軟性の高いテンプレートエンジンで、上記で紹介した関数・フィルターの他にも

- グローバル変数
- タグ
- オペレーター
- テスト

などを独自に拡張できます。詳しく知りたい方は、[Extending Twig](#) を読むとよいでしょう。

明日のアドベントカレンダーも引き続き私です w

### @koyhoge

2014 年からはフリーになる予定は未定の PHP 大好きっこです、テヘ☆

Twitter: [@koyhoge](#)

Blog: <http://d.hatena.ne.jp/koyhoge/>

## JavaScript 側に PHP 変数を簡単にまろごと渡す方法

ハイ、昨日のオレに引き続き [FuelPHP Advent Calendar 2013](#) の 6 日目です。  
今回の内容もまた Twig 絡みです。実は昨日の記事は、本日の記事の前準備になっていたのです。

### JavaScript 側に PHP のオブジェクトを渡したい

最近の Web アプリは UI のインタラクションが凝っていて、ブラウザ側の JavaScript で色んな制御をすることも当たり前になってきました。jQuery や様々な jQuery プラグインを駆使して、ユーザに分かりやすく使いやすいサービスを提供することは、もはやウェブエンジニアとしては持っていて当然のスキルになっています。

そのような UI を作っている際に、JavaScript 側に動作パラメータの初期値を渡すのに値を 1 つ 1 つテンプレート記法で埋め込むのが面倒だったので、一発で渡せる Twig Extension を作ったので紹介します。

### data\_bind 関数

アイデアとしては、見えない HTML 要素を作成してそのテキストに値を JSON 化して突っ込むという、まあ普通に思いつきそうなものです。でもこれがやってみると思ったより便利で。

extension 本体はこんなコードです。

```
<?php
/*
 * Copyright (c) 2013 KOYAMA Tetsuji
 * This software is released under the MIT License.
 * http://opensource.org/licenses/mit-license.php
 */

class Hoge_Twig_Extension extends Twig_Extension
{
    ...略...

    public function getFunctions()
    {
        return array(
            new Twig_SimpleFunction('data_bind', array($this, 'dataBind')),
        );
    }

    public function dataBind($name, $val, $exclude = null)
    {
        if (is_object($val) && is_callable(array($val, 'to_array'))) {
            $val = $val->to_array();
        }
        if (! empty($exclude)) {
            if (is_string($exclude)) {
                $exclude = array($exclude);
            }
            foreach ($exclude as $key) {
                unset($val[$key]);
            }
        }
    }
}
```

```
        $fmt = '<div id="data-%s" class="hide">%s</div>';
        return sprintf($fmt, $name, json_encode($val));
    }
}
```

div 要素を不可視にするために、ここでは hide という class を指定していますが、これは CSS で

```
.hide {
    display: none;
}
```

的なものがあることを前提にしています。Bootstrap には含まれてますね。もちろん直接 style を書いてしまってもよいでしょう。

コントローラ側のアクションメソッドで以下のようにテンプレートに値を渡して、

```
function action_xxx()
{
    ...略...
    // $userinfo は情報が入った Object または連想配列
    $this->template->user = $userinfo;
}
```

テンプレート側ではこう記述します。

```
{{ data_bind('user', user) }}
```

JavaScript 側でその値を使用するには、例えば jQuery だったら

```
var user = $.parseJSON($('#data-user').text());
```

と書くと、user 変数に PHP で渡した値が入ります。

## パラメータの解説

data\_bind 関数は 3 つのパラメータを持ちます。

### \$name: 名前

HTML 上で展開される名前です。'data-名前'がその要素の id になります。

### \$val: 変数

展開する変数です。Twig の変数になります。

## \$exclude: 排除するキー（省略可能）

変数を全部 JS 側に渡すのが楽とはいっても、ユーザ側に公開したくない内部プロパティが含まれているかもしれません。そういう場合には、第 3 引数にそのプロパティ名を渡すことであらかじめ削除した上で展開することができます。

単なる文字列として指定することもできますし、

```
{{ data_bind('user', user, 'password') }}
```

配列にして複数指定することもできます。

```
{{ data_bind('user', user, ['password', 'rank']) }}
```

ということでお手軽 tips でした。明日のアドベントカレンダーは@LandscapeSketch さんです。

## 【補足】FuelPHP の View の自動エスケープについて

上記のエントリについて、PHP の json\_encode() 関数は標準ではエスケープ処理は行わないので XSS 脆弱性があるのではないか、という指摘をいただきました。

### json\_encode() のエスケープオプション

確かに PHP のマニュアルには、各種文字にエスケープ対応するオプションが存在します。

PHP: json\_encode - Manual ([http://php.net/json\\_encode](http://php.net/json_encode))

この場合で言えば

```
return sprintf($fmt, $name, json_encode($val));
```

を以下のようにエスケープオプションを追加するべきということです。

```
return sprintf($fmt, $name, json_encode($val, JSON_HEX_TAG | JSON_HEX_APOS | JSON_HEX_QUOT | JSON_HEX_AMP));
```

ただこのコードを書いた時に自動的にエスケープ処理がかかることを確認していたので、どこでそれが行われるかは深く調べずに、json\_encode のオプションを省いたという経緯がありました。

### 自動エスケープはFuel\Core\View の機能だった

その後 [fuelphp.jp グループ](#)で@kenji\_s さんに指摘されて、Parser パッケージの標準設定で'auto\_encode'が true になっているおかげでテンプレートに渡される変数が自動でエスケープされていた事がわかりました。

```
fuel/packages/parser/config/parser.php
```

の以下の部分ですね。

```
<?php
...略...
    'View_Twig' => array(
        'auto_encode' => true,
    ...略...
```

この `auto_encode` 設定は、`\Fuel\Core\View` のコンストラクタに `$auto_filter` として渡され、結果的に `\Fuel\Core\Security::clean()` が呼び出されます。つまり Twig Extension に渡される際にはすでにエスケープ済みになっていたわけですね。

PHP 変数を JSON にして JavaScript に渡す仕組みは、別に FuelPHP でなくても使用できますので、その場合は XSS に注意して `json_encode` にオプションを随時追加して下さい。

### JSON の埋め込み方の問題

他にも fuelphp.jp グループでは [@takayuki\\_h](#) さんより、HTML 要素にテキストとして JSON を書き出すよりは、要素の `data-option` 属性として埋め込んだ方が良いのではないかと指摘を受けました。

```
<div class="hidden">{"fuga":"hoge"}</div>
```

ではなく

```
<div class="hidden" data-option='{"fuga":"hoge"}'></div>
```

とせよということですね。ふむ一、これはちょっと試してみたいと思います。

### その他の反応への返事

はてブより。

```
id:teppei $name も json_encode() もエスケープが足りません。危険。
http://b.hatena.ne.jp/teppei/20131207#bookmark-172246146
```

`json_encode` については上記に書いたとおり。`$name` はテンプレートに直接記述されるので、そこに外部からの変数が渡される事態は、コード全体を見直したほうが良いレベルだと思うのですがどうでしょう？

```
id:thujikun JSON 形式のコードを JS の変数に直接代入する方が楽な気が。。。ひとつグローバル変数使うことにはなるけども。
http://b.hatena.ne.jp/thujikun/20131208#bookmark-172246146
```

JavaScript にテンプレートエンジンを通して変数展開を埋め込む方が、自分的にはあり得ないです。HTML に埋め込み JS を

直接記述することは現在は全くやっていません。

id:fakechan PHP のレガシーっぷりに驚きを隠せない。というか、こういう場合は REST API を作って「js 側から」Ajax でアクセスすればいいのでは。Ajax のロードが終わるまでは「ロード中...」とかかぶせて。

<http://b.hatena.ne.jp/fakechan/20131208#bookmark-172246146>

いやこれと PHP のレガシーは関係ないでしょ。PHP ディスりたい病にかかっているようですね。何でも REST で Ajax すれば良いやというのは、JS 側の処理を無駄に複雑にするだけではありませんか？

**@koyhoge**

このエントリは思っていたよりも反響がありました。JS と PHP を組み合わせてうまく処理するニーズは、まだまだ沢山埋もれているように思います。

Twitter: [@koyhoge](https://twitter.com/koyhoge)

Blog: <http://d.hatena.ne.jp/koyhoge/>

## FuelPHP 開発でローカルと Web で構造が変わっても対応できる小技

こんにちはタケ (@LandscapeSketch) です。

今日は「FuelPHP 開発でローカルと Web で構造が変わっても対応できる小技」として、ローカルと Web 上でディレクトリ構造を変えていても便利に開発できる小技を紹介します。

### 開発環境と Web 上の構造

現在、ローカルでは Windows、XAMPP、NetBeans の組み合わせで開発しています。  
ローカルでは 1 つのプロジェクトディレクトリに全てを入れ管理を行います。

#### XAMPP開発中

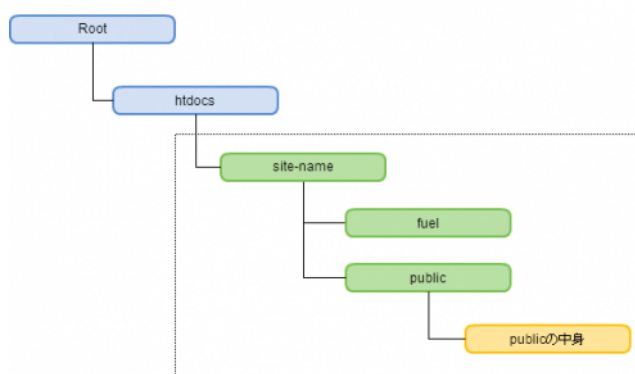


図 7.1 ローカル開発環境

Web 上では安全のために fuelphp/public の中身のみを公開ディレクトリに入れ、fuelphp/fuel は非公開エリアに移動しようと思いました。

#### オンライン

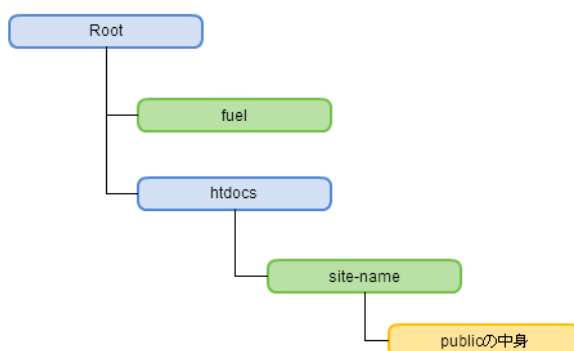


図 7.2 オンライン公開状態

しかし NetBeans ではプロジェクトディレクトリ以下に入っているファイルしか管理できません。サイト名のメインディレク



トリがあり、その下に fuel、public 両方が入っている必要があります（図 7.1 の点線部分）。

Web と同じ構造にすると NetBeans で管理できなくなってしまうのです。

となるとデプロイするたびに設定を書き換えてアップロードする必要があります。手間がかかり、もし書き忘れた場合はエラー画面となってしまいます。

## index.php を柔軟にする

いろいろ試した結果、public/index.php を書き換えることで良い感じになりました。

public/index.php を開くと FuelPHP で使用するディレクトリパスの設定部分があります。

通常では

```
define('APPPATH', realpath(DIR . '/../fuel/app/').DIRECTORY_SEPARATOR);
```

のように ../fuel/app/ の部分が固定されています。

まずこの部分を変数に置き換えます。../fuel を \$fuel\_dir に定義しています。

```
define('APPPATH', realpath(DIR . $fuel_dir . '/app/').DIRECTORY_SEPARATOR);
```

## .htaccess に仕掛け

次にそれぞれの環境の .htaccess に fuel の環境設定を書き加えます。

ローカル

```
SetEnv FUEL_ENV development
```

Web

```
SetEnv FUEL_ENV production
```

## 再度 index.php

再度 public/index.php に以下の構文を加えました。

```
if ($_SERVER['FUEL_ENV'] == 'production') {
    $fuel_dir = '/../../fuel';
    ini_set('display_errors', 0);
} elseif ($_SERVER['FUEL_ENV'] == 'development') {
    $fuel_dir = '/../fuel';
    ini_set('display_errors', 1);
}
```

\$fuel\_dir に環境ごとの fuel ディレクトリ位置を代入しています。その後は各グローバル変数に代入されていきます。また同時に

```
iniset('display_errors', 0);
```

などエラー表示設定を書いておくことで、公開した時に内部エラーが丸見えにならないようにもできました。

## まとめ

ごく簡単な Tips でしたがいかがでしたでしょうか？

ただこれは私の考えた方法で、もっと簡潔で安全な方法もあるかと思います。もし「こんなふうを書くともっと簡単だよ！」という方法がありましたら、ぜひコメントしてください！

あすは@sa2yasu さんの「FuelPHP を更に使ってみて使えるなと思った CustomValidationRule の書き方と Core 拡張の小技」です。お楽しみに～

(文中のソースコードは非常に短いですが、念のためすべて **MIT ライセンス**といたします。)

### @LandscapeSketch

FuelPHP をベースに Web サービスやサイトを作っています。ブログではときどき FuelPHP の Tips など公開中。

Twitter: [@LandscapeSketch](#)

Blog: <http://worktoolsmith.com/>

# FuelPHP を更に使ってみて使えるなと思った拡張 ValidationRule の書き方と Core 拡張の小技巧

タイトル長っ w

FuelPHP Advent Calendar 2013 の 8 日目です。

CMS 系のサイト構築は大体どのフレームワークでも問題ないのですが、基幹業務チックな機能要件があると、様々なモデルの組み合わせで入力値の検証をしたりだとか、項目間において依存性のあるチェックだとかが頻発します。

PHP で基幹システム作らない方が・・・というご意見もごもっともで、そのうちもっとナウい言語にシフトしていきたいと思います。

本題に戻ると FuelPHP はその辺が結構柔軟で、やりようによってはいくらでも対応できるので、1 年半ぐらい実プロジェクトで使ってきましたが困ることは特になかったです。

強いていえば、ORM で related して rows\_limit するとあばばばってなりますが、クエリビルダと ORM の勘所も掴んだので特に問題ありません。

## 1. おさらい

公式ドキュメント（日本語訳）(<http://fuelphp.jp/docs/1.7/classes/validation/validation.html>)

デフォルトの入力検証含め、こちらに書いてある内容でほとんどの入力値の検証は可能です。

## 2. 複数の任意のパラメータを用いて検証したい場合

複数のポストされたパラメータを用いた検証を行いたい場合、拡張 Validation を書きますが、公式サンプルのパラメータの渡し方がなんとも・・・

公式のサンプル

```
// app/classes/myrules.php
class MyRules
{
    // note this is a static method
    public static function _validation_unique($val, $options)
    {
        list($table, $field) = explode('.', $options); // デリミタエエ・・・

        $result = DB::select("LOWER (\\"$field\\")")
            ->where($field, '=', Str::lower($val))
            ->from($table)->execute();

        return ! ($result->count() > 0);
    }

    // note this is a non-static method
    public function _validation_is_upper($val)
    {
        return $val === strtoupper($val);
    }
}
```

```

}

// and call it like:
$val = Validation::forge();

// Note the difference between static and non-static validation rules:

// Add it statically, will only be able to use static methods
$val->add_callable('MyRules');

// Adds it non-static, will be able to use both static and non-static methods
$val->add_callable(new MyRules());

$val->add('username', 'Your username', array(), array('trim', 'strip_tags', 'required', 'is_upper'))
    ->add_rule('unique', 'users.username');

```

以下のようにして書くと、わざわざパラメータをデリミタで区切らなくても引数として渡すことができます。

一部抜粋&修正

```

// app/classes/myrules.php
class MyRules
{
    // note this is a static method
    public static function _validation_unique($val, $table, $field) // 引数で渡ってくるよ
    {
        $result = DB::select("LOWER (\\"$field\\")")
            ->where($field, '=', Str::lower($val))
            ->from($table)->execute();

        return ! ($result->count() > 0);
    }
    ...略...
}

...略...
$val->add('username', 'Your username', array(), array('trim', 'strip_tags', 'required', 'is_upper'))
    ->add_rule('unique', 'users', 'username'); // デリミタつけなくても渡せます

```

### 3. 独自バリデーションはバリデーションクラス作らなきゃダメですか？

答えはノー。再利用性のあるもの (汎用的な独自バリデーション) はクラスにしておいた方が後々使い回しがききますが、例えばログイン検証なんかは再利用しませんよね？

バリデーションクラスを作らずに独自ルールを定義するには Closure を使います。

Closure で書くルール

```

/*
 * Copyright (c) 2013 Yasuyuki Sasaki
 * This software is released under the MIT License.
 * http://opensource.org/licenses/mit-license.php
 */

public static function login_validate()
{
    $val = Validation::forge();
    $password = Input::post('password');

    $val->add('email', 'Email')
        ->add_rule('required')

```

```

->add_rule( // login_validate
    function($email) use ($password) {
        if (! $email || ! $password) {
            return true;
        }
        $user = Model_User::query()->
            where('email', $email)->
            where('password', $password)->
            where('del_flg', 0)->get_one();

        if ($user) {
            return true;
        } else {
            Validation::active()->set_message('closure', 'ログインに失敗しました。');
            return false;
        }
    });
$val->add_field('password', 'パスワード', 'required');
return $val;
}

```

## 検証ルールに Closure を使う場合の落とし穴

Closure で複数のルールを定義すると Validation::active()->set\_message で定義したメッセージがおかしなことになります。以下のサンプルを見てください。

わざと検証に失敗する2つのルールを Closure で定義します。

ルールを Closure で複数書く

```

$val = Validation::forge();

$val->add('test1', 'test1')->add_rule(
    function($val) {
        Validation::active()->set_message('closure', 'closure message 1');
        return false;
    }
);
$val->add('test2', 'test2')->add_rule(
    function($val) {
        Validation::active()->set_message('closure', 'closure message 2');
        return false;
    }
);

```

この検証を実行すると以下のようなエラーが出力されます。

```

closure message 2
closure message 2

```

後に定義した方のメッセージが先に定義した Closure のメッセージを上書きしてしまうんですね。

よく公式ドキュメントを読むとそれらしいことが書いてあるのですが、こういう場合は Closure のルールに名前をつけてあげるといいらしいです。(匿名関数に名前つけるってどういうこっちゃと思うかもしれませんが・・・)

Closure に名前をつける

```

$val = Validation::forge();

```

```

$val->add('test1', 'test1')->add_rule(['closure1' =>
    function($val) {
        Validation::active()->set_message('closure1', 'closure message 1');
        return false;
    }
]);
$val->add('test2', 'test2')->add_rule(['closure2' =>
    function($val) {
        Validation::active()->set_message('closure2', 'closure message 2');
        return false;
    }
]);

```

見てわかるとおり Closure のルールをキーと Closure の連想配列で add\_rule に渡してあげます。

こうすると FuelPHP の中で「この Closure のルールで検証に失敗したらこのメッセージを表示する」というような関連づけが行われるようになります。

## 4. Core 拡張の小技巧

ついでの小ネタ。

拡張バリデーションを定義する際、デフォルトのルールにないけど汎用的な検証を行うというケースがあると思います。

sample

```

$val = Validation::forge();
$val->add_callable(new Validate_Common()); // 汎用的な独自ルール

```

数カ所から呼ばれるならこれでもよいかもしれませんが、さすが、これが数十とかになると汎用ルールの add\_callable 書くの面倒くさくなりませんか？ DRY (Don't repeat yourself) しましょ。

fuel/app/classes/core/validation.php

```

/*
 * Copyright (c) 2013 Yasuyuki Sasaki
 * This software is released under the MIT License.
 * http://opensource.org/licenses/mit-license.php
 */

class Validation extends \Fuel\Core\Validation
{
    protected function __construct($fieldset)
    {
        parent::__construct($fieldset);
        $this->add_callable(new Validate_Common());
    }
}

```

fuel/app/bootstrap.php のオートローダ部分

```

Autoloader::add_classes(array(
    'Validation'      => __DIR__.'/classes/core/validation.php',
    'View'            => __DIR__.'/classes/core/view.php',
));

```

これで add\_callable しなくてもバリデーション定義に汎用ルールが使えるようになります。

同じ要領で汎用的な View の表示処理やデータ整形なども View クラスを拡張すると DRY できるようになります。

fuel/app/core/view.php

```
class View extends \Fuel\Core\View
{
    public function hogeFormat($val)
    {
        // なんかも整形する処理
    }
}
```

そんなこんなで「もうバリデーションなんて怖くないぞ」という感じにまとまりました。

## 最後に

コミュニティも盛り上がり、各地で実績が増え、弊社でも FuelPHP が定着化してきました。

私自身 PHP 歴は2年弱、FuelPHP は1年半と PHP を初めて間もなく FuelPHP と出会いました。

元々他言語においてもフレームワークを追っかけるのが好きだったので、PHP でも Zend、Symfony、CakePHP、CodeIgniter、Kohana、どれにしようかな神様の言う通りせっ (ry 状態だったのですが、FuelPHP 選んでよかったです (笑)

**@sa2yasu**

趣味がストリートダンスのロールキャベツ系男子ベチぱー。青森県青森市、宮城県石巻市を中心に何かしら活動してます。

Twitter: [@sa2yasu](https://twitter.com/sa2yasu)

Blog: <http://qiita.com/ya-sasaki@github/items/e238f86cabce3acfbe53>

# AspectMock で FuelPHP のアプリを 100 %テスト可能にする

FuelPHP Advent Calendar 2013 の 9 日目です。

今日は、いま話題の AspectMock を FuelPHP で使い、FuelPHP のアプリを 100 %テストできないかというお話です。

ちなみに、本当に 100% テストできるかどうかはまだ定かではありません。あと、カバー率を 100% にすること自体は目的ではないので、あまり気にする必要はないと思います。

"Testability" should not be used as argument deciding what design pattern is right to use and what is not.

訳：どのデザインパターンが適切か否かという論拠に、「テスト可能性」が使われるべきではない。

これが、AspectMock からの主張です。

なお、この記事の前提環境は、以下のとおりです。

- FuelPHP 1.7.1
- AspectMock master ブランチ cc2be6945a705e65a2a4a12df7e35de82d0129f7 (2013-09-09)

## 準備

AspectMock と必要なライブラリを Composer で追加します。

```
diff --git a/composer.json b/composer.json
index e1b21ea..006ac5b 100644
--- a/composer.json
+++ b/composer.json
@@ -16,10 +16,14 @@
     "forum": "http://fuelphp.com/forums"
   },
   "require": {
-     "php": ">=5.3.3",
+     "php": ">=5.4",
+     "monolog/monolog": "1.5.*",
+     "fuelphp/upload": "2.0.1"
   },
+  "require-dev": {
+    "codeception/aspect-mock": "*",
+    "symfony/finder": "*"
+  },
   "suggest": {
     "mustache/mustache": "Allow Mustache templating with the Parser package",
     "smarty/smarty": "Allow Smarty templating with the Parser package",
```

インストールします。



```
$ php composer.phar update
```

### FuelPHP 1.7.1 の変更

テスト実行の場合、AspectMock を使うように FuelPHP を変更します。

```
diff --git a/fuel/app/bootstrap.php b/fuel/app/bootstrap.php
index a6213d5..b491688 100644
--- a/fuel/app/bootstrap.php
+++ b/fuel/app/bootstrap.php
@@ -1,9 +1,5 @@
 <?php

-// Load in the Autoloader
-require COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php';
-class_alias('Fuel\\Core\\Autoloader', 'Autoloader');
-
 // Bootstrap the framework DO NOT edit this
require COREPATH.'bootstrap.php';
```

```
diff --git a/oil b/oil
index 62033d6..4a21f80 100644
--- a/oil
+++ b/oil
@@ -48,6 +48,10 @@ define('COREPATH', realpath(__DIR__.'../fuel/core/').DIRECTORY_SEPARATOR);
 defined('FUEL_START_TIME') or define('FUEL_START_TIME', microtime(true));
 defined('FUEL_START_MEM') or define('FUEL_START_MEM', memory_get_usage());

+// Load in the Autoloader
+require COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php';
+class_alias('Fuel\\Core\\Autoloader', 'Autoloader');
+
 // Boot the app
require APPPATH.'bootstrap.php';
```

```
diff --git a/public/index.php b/public/index.php
index e01d3a4..9cb90d3 100644
--- a/public/index.php
+++ b/public/index.php
@@ -40,6 +40,10 @@ define('COREPATH', realpath(__DIR__.'../fuel/core/').DIRECTORY_SEPARATOR);
 defined('FUEL_START_TIME') or define('FUEL_START_TIME', microtime(true));
 defined('FUEL_START_MEM') or define('FUEL_START_MEM', memory_get_usage());

+// Load in the Autoloader
+require COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php';
+class_alias('Fuel\\Core\\Autoloader', 'Autoloader');
+
 // Boot the app
require APPPATH.'bootstrap.php';
```

AspectMock がロードされるようにし、AspectMock の設定をします (\$kernel->init() の部分)。ここが正しくないと AspectMock が正常に動作しません。

```

diff --git a/bootstrap_phpunit.php b/bootstrap_phpunit.php
index 3b5b851..2605850 100644
--- a/bootstrap_phpunit.php
+++ b/bootstrap_phpunit.php
@@ -32,6 +32,34 @@ unset($app_path, $core_path, $package_path, $_SERVER['app_path'], $_SERVER['core
    defined('FUEL_START_TIME') or define('FUEL_START_TIME', microtime(true));
    defined('FUEL_START_MEM') or define('FUEL_START_MEM', memory_get_usage());

+/**
+ * Load the Composer autoloader if present
+ */
+defined('VENDORPATH') or define('VENDORPATH', realpath(COREPATH.'../vendor').DS);
+if ( ! is_file(VENDORPATH.'autoload.php'))
+{
+    die('Composer is not installed. Please run "php composer.phar update" in the root to install Composer');
+}
+require VENDORPATH.'autoload.php';
+
+// Add AspectMock
+$kernel = \AspectMock\Kernel::getInstance();
+$kernel->init([
+    'debug' => true,
+    'appDir'   => __DIR__ . '/../',
+    'includePaths' => [
+        __DIR__.'../app', __DIR__.'../core', __DIR__.'../packages',
+    ],
+    'excludePaths' => [
+        __DIR__.'../app/tests', __DIR__.'../core/tests',
+    ],
+]);
+
+// Load in the Autoloader
+//require COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php';
+$kernel->loadFile(COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php'); // path to your autoloader
+class_alias('Fuel\Core\Autoloader', 'Autoloader');
+
+// Boot the app
+require_once APPPATH.'bootstrap.php';

```

AspectMock が動作するように、fuel/core/phpunit.xml を fuel/app/phpunit.xml にコピーし、backupGlobals を false に変更します。

```

--- fuel/core/phpunit.xml    2013-12-02 19:56:52.847375706 +0900
+++ fuel/app/phpunit.xml    2013-12-02 19:56:38.910935143 +0900
@@ -1,6 +1,6 @@
 <?xml version="1.0" encoding="UTF-8"?>

-<phpunit colors="true" stopOnFailure="false" bootstrap="../../core/bootstrap_phpunit.php">
+<phpunit colors="true" stopOnFailure="false" bootstrap="../../core/bootstrap_phpunit.php" backupGlobals="false">
   <php>
     <server name="doc_root" value="../../"/>
     <server name="app_path" value="fuel/app"/>

```

## FuelPHP 1.8/develop (1.7.2 以降) の変更

1.8/develop ブランチでは、上記の変更が取り込まれていますので、fuel/core の bootstrap\_phpunit.php を変更し、

```
diff --git a/bootstrap_phpunit.php b/bootstrap_phpunit.php
index 50b9c88..20678e7 100644
--- a/bootstrap_phpunit.php
+++ b/bootstrap_phpunit.php
@@ -40,8 +40,22 @@ if ( ! is_file(VENDORPATH.'autoload.php'))
 }
 require VENDORPATH.'autoload.php';

+// Add AspectMock
+$kernel = \AspectMock\Kernel::getInstance();
+$kernel->init([
+    'debug' => true,
+    'appDir' => __DIR__ . '/../',
+    'includePaths' => [
+        __DIR__ . '/../app', __DIR__ . '/../core', __DIR__ . '/../packages',
+    ],
+    'excludePaths' => [
+        __DIR__ . '/../app/tests', __DIR__ . '/../core/tests',
+    ],
+]);
+
+// Load in the Fuel autoloader
-require COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php';
+//require COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php';
+$kernel->loadFile(COREPATH.'classes'.DIRECTORY_SEPARATOR.'autoloader.php'); // path to your autoloader
class_alias('Fuel\\Core\\Autoloader', 'Autoloader');

// Boot the app
```

fuel/app/phpunit.xml を作成するだけで ok です。

```
--- fuel/core/phpunit.xml    2013-12-02 19:56:52.847375706 +0900
+++ fuel/app/phpunit.xml    2013-12-02 19:56:38.910935143 +0900
@@ -1,6 +1,6 @@
<?xml version="1.0" encoding="UTF-8"?>

-<phpunit colors="true" stopOnFailure="false" bootstrap="../../core/bootstrap_phpunit.php">
+<phpunit colors="true" stopOnFailure="false" bootstrap="../../core/bootstrap_phpunit.php" backupGlobals="false">
  <php>
    <server name="doc_root" value="../../"/>
    <server name="app_path" value="fuel/app"/>
```

## テストの書き方

コントローラから `Response::redirect()` でリダイレクトする場合のテストを作成してみましょう。

`Response::redirect()` は安全のため内部で `exit()` しているため、そのままではテストがそこで終了してしまい、テストできません。これをテストダブルで置き換えて、テスト可能にします。

まず、以下のようなコントローラを作成します。

```
<?php

class Controller_Test extends Controller
{
    public function action_redirect()
    {
        return Response::redirect('welcome/404', 'location', 404);
    }
}
```

```
}

```

続いてテストを作成しましょう。

```
<?php

// AspectMock の Test クラスを test としてインポート
use AspectMock\Test as test;

/**
 * Tests for Controller_Test
 *
 * @group App
 * @group Controller
 */
class Test_Controller_Test extends TestCase
{
    protected function tearDown()
    {
        test::clean(); // 登録したテストダブルをすべて削除
    }

    public function test_redirect()
    {
        // Response::redirect() を単に true を返すテストダブルに置き換え
        $req = test::double('Fuel\Core\Response', ['redirect' => true]);

        // 'test/redirect' へのリクエストを生成
        $response = Request::forge('test/redirect')
            ->set_method('GET')->execute()->response();

        // Response::redirect() が以下の引数で実行されたことを確認
        $req->verifyInvoked('redirect', ['welcome/404', 'location', 404]);
    }
}
```

テストを実行します。

```
$ oil test --group=App
Tests Running...This may take a few moments.
PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /mnt/fuelphp/fuel/app/phpunit.xml

.

Time: 8.08 seconds, Memory: 48.50Mb

OK (1 test, 0 assertions)
```

通りました。「0 assertions」というのがちょっと変ですが、PHPUnit の検証メソッドを使っていないため、いたしかたありません。

試しに、verifyInvoked() での第 3 引数の指定を 404 から 405 に変更してみます。

```
$req->verifyInvoked('redirect', ['welcome/404', 'location', 405]);
```

```
$ oil test --group=App
Tests Running...This may take a few moments.
PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /mnt/fuelphp/fuel/app/phpunit.xml

.

Time: 8.08 seconds, Memory: 48.50Mb

OK (1 test, 0 assertions)
[vagrant@localhost fuelphp]$ oil test --group=App
Tests Running...This may take a few moments.
PHPUnit 3.7.28 by Sebastian Bergmann.

Configuration read from /mnt/fuelphp/fuel/app/phpunit.xml

F

Time: 7.72 seconds, Memory: 48.50Mb

There was 1 failure:

1) Test_Controller_Test::test_redirect
Expected Fuel\Core\Response::redirect('welcome/404','location',405) to be invoked but it never occur.

/mnt/fuelphp/fuel/vendor/codeception/aspect-mock/src/AspectMock/Proxy/Verifier.php:73
/mnt/fuelphp/fuel/app/tests/controller/test.php:28

FAILURES!
Tests: 1, Assertions: 0, Failures: 1.
```

正しく失敗しました。

このように、AspectMock を使うと静的メソッドをテストダブルに置き換えたり、メソッドを動的に再定義して、簡単にテストすることができます。AspectMock の主張どおり、テスト可能にするためだけに DI を使う必要はなくなります。

ただし、AspectMock は「Stability: alpha」となっており、まだ発展途上のツールのようなので期待したとおりに動作しない可能性はあります。

## 参考

- <https://github.com/Codeception/AspectMock>
- [Nothing is Untestable: AspectMock in Action](#)
- [Understanding AspectMock](#)

## おまけ

この FuelPHP Advent Calendar は今年で 3 年目ですが、一昨年、去年の分は、電子書籍化されて IT 系の有名出版社より出版されています。

- 『FuelPHP Advent Calendar 2011』【電子書籍】 技術評論社
- 『FuelPHP Advent Calendar 2012』【電子書籍】 達人出版会

いずれも無料でダウンロードできますので、まだ、読んでいない FuelPHP ユーザの方は、読んでみるといろいろな発見があると思いますよ。

**kenjis**

FuelPHP まとめ Wiki 管理人。「PHP5 技術者認定上級試験」認定者。

Twitter: [@kenji\\_s](#)

Blog: <http://blog.a-way-out.net/>

## イベント機能を使ってアプリケーションをカスタマイズする

FuelPHP advent Calendar 2013 の 10 日目です。

今日は、「イベント機能を使ってアプリケーションをカスタマイズする」です。

### イベントとは何か

イベントとは、FuelPHP コアを書き換えることなく、独自の処理を差し込むことができる仕組みです。

- FuelPHP の処理の途中に、トリガー (ここを通ったときに、追加処理を実行する場所) がいくつか用意されている
- トリガー名を指定して、独自の処理を追加する
- 独自の処理が追加されたトリガーのところで、独自の処理が実行される

という仕組みです。WordPress を使っている方であれば、「WordPress のアクションフックに似ている」というかもしれません。

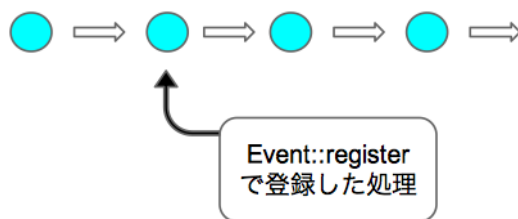


図 10.1 アプリケーションの処理の流れ：青丸がトリガー

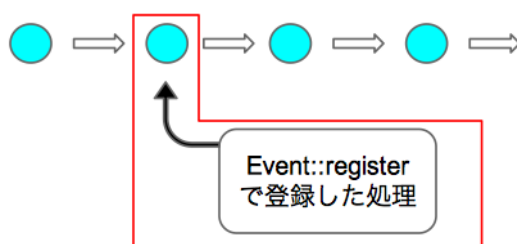


図 10.2 イベントを実行：処理途中の各トリガーの所では、登録されているイベントがあれば実行する

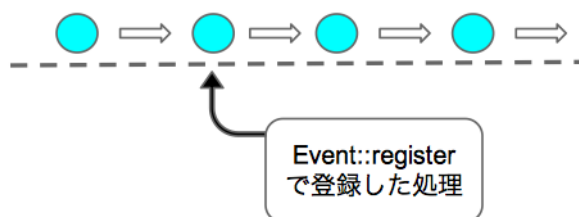


図 10.3 コードは点線部分で分離されている：アプリケーション自体のコードに手を入れずに、機能追加／変更する

## 独自の処理を追加する

2通りの方法が用意されています。

1. app/config/event.php に追加する方法
2. Event::register を使う方法

1. は、app/config に event.php に書く方法です。詳細は [Event クラス \(FuelPHP 1.7\)](#) をごらんください。  
2. は、register メソッドを使う方法です。今回はこちらを使います。Event クラス (FuelPHP 1.7) では、user\_login に Class::method 処理を追加する例が掲載されています。

```
Event::register('user_login', 'Class::method');
```

bootstrap.php に書けば、トリガーにフックできます。

## トリガーにフックしてみる

では、実際にトリガーにフックしてみましょう。ここでは、[Novius OS](#) にフックする例を紹介します。Novius OS は、FuelPHP ベースの CMS で、jQuery UI、Wijmo を使った使い易いインターフェースが特徴です。[Novius OS Chiba2 のイベント一覧](#)にある admin.loginFail を使い、ログイン失敗をログに記録します。

```
/**
 * Copyright (c) 2013 Fumito MIZUNO
 * License: MIT
 * http://opensource.org/licenses/mit-license.php
 */

Event::register('admin.loginFail', 'warning_on_loginfail');

function warning_on_loginfail()
{
    $message = 'Login Fail.';
    $message .= ' Email: ' . Input::post('email');
    $message .= ' Password: ' . Input::post('password');
    $message .= ' IP: ' . Input::ip();
    Log::warning($message);
}
```

Novius OS は、メールアドレスとパスワードでユーザー認証するので、ログイン失敗時にそれらを記録します。これらは FuelPHP の Input クラスを利用します。引数無しで Input::post() を実行すると \$\_POST を全部取ってきて配列で返すので、丸ごと記録したければ Format::forge(Input::post())->to\_serialized() としてもいいでしょう。またアクセス元の IP アドレスも記録し、ログ記録には FuelPHP の Log クラスを利用します。ログ記録の内部処理は [Monolog](#) ライブラリ (MIT ライセンス) が行っています。

ログインに失敗すると、ログファイルに

```
WARNING - 2013-12-09 09:15:23 --> Login Fail. Email: email@example.com Password: p@ssw0rd IP: 127.0.0.1
```

のように記録されます。



## デフォルトで用意されているイベント

**Event クラス (FuelPHP 1.7)** によると、デフォルトで用意されているイベントは、`app_created`、`request_created`、`request_started`、`controller_started`、`controller_finished`、`response_created`、`request_finished`、`shutdown` です。これらのトリガーに処理を追加することができます。

## アプリケーションにトリガーを用意する

コアに用意されているトリガーを使うだけではなく、アプリケーションにもトリガーを用意することができます。図 10.1 の、青丸を作る作業です。コード中の適当な箇所に、`Event::trigger(トリガー名)` と書けば OK です。

Novius OS のログイン部分のコード ([novius-os/framework/classes/controller/admin/login.ctrl.php](http://novius-os/framework/classes/controller/admin/login.ctrl.php)) を見てみると、ログイン成功時に `admin.loginSuccess` イベント、ログイン失敗時に `admin.loginFail` イベント、が用意されています。

```
/**
 * NOVIUS OS - Web OS for digital communication
 *
 * @copyright 2011 Novius
 * @license GNU Affero General Public License v3 or (at your option) any later version
 * http://www.gnu.org/licenses/agpl-3.0.html
 * @link http://www.novius-os.org
 */

namespace Nos;

class Controller_Admin_Login extends Controller
{
    ...略...
    protected function post_login()
    {
        if (\Nos\Auth::login($_POST['email'], $_POST['password'], (bool) \Input::post('remember_me', false))) {
            if (\Event::has_events('user_login')) {
                \Log::deprecated('Event "user_login" is deprecated, use "admin.loginSuccess" instead.',
                    'Chiba.2');
                \Event::trigger('user_login');
            }
            \Event::trigger('admin.loginSuccess');
            return true;
        }
        \Event::trigger('admin.loginFail');
        return __('These details won't get you in. Are you sure you've typed the correct email address and password? Please try again.');
```

## どういうときに役に立つのか

アプリケーションを全て自分で作っている場合は、わざわざイベントトリガーを用意するメリットはあまりないかもしれません。トリガーを使わずに書いても良いでしょう。トリガーが威力を発揮するのは、パッケージやアプリケーションをオープンソースで公開する場合だと思います。

Novius OS のログイン失敗時のログを取りたい場合、上述の `post_login` メソッドを直接書き換えるというやり方でも、ログ機能を追加することはできます。でもこの方法だと、元のアプリケーション (Novius OS) のアップデート時に書き直すことになります。

では、イベントを活用した場合はどうでしょう。Novius OS ではトリガーが用意されています。カスタマイズしたい人は、トリガーにフックするコードを自作し、`bootstrap.php` に記述します。この方法だと、アプリケーションのコードに手を入れる

ことなく、処理を追加することができます。自分で追加した部分は元のアプリケーションのコードから分離でき、アップデートが楽になります。

もちろん、イベントを使ってカスタマイズするには、トリガーが設定されていなければなりません。なので、パッケージやアプリケーションを公開して、他の人にも使ってもらいたい、という場合には、イベントトリガーを設定しておくといいでしょう。

## まとめ

イベントを使うことで、フレームワークやアプリケーションのコードに手を入れることなく、処理を追加することができます。オープンソースで公開するアプリケーションには、イベントのトリガーを用意しておく、カスタマイズしやすくなります。

**@ounziw**

Novius OS のコア貢献者であり、日本語対訳ファイルの作成も行っている。

Twitter: [@ounziw](https://twitter.com/ounziw)

Blog: <http://ounziw.com/>

## FuelPHP をもっと Composer で使う

こんにちは、chatii です。FuelPHP Advent Calendar 2013 11 日目を担当させていただきます。  
本日は chatii が Composer で FuelPHP を入れた話を披露したいと思います。

### Composer を採用した FuelPHP への不満

さて、みなさん Composer への理解は進みましたか？ chatii はほぼ日曜プログラマー的レガシー的野良 PHPer なわけですし、フレームワークおいしい (^^) → Composer なにそれうまいの？ という状態で FuelPHP 1.6 を迎えました。『PHP エンジニア養成読本』のおかげで今やそれなりにモダンな PHPer になれたと思います。

さて、Composer のドキュメントを読みますと

Composer is a tool for dependency management in PHP.

Composer は PHP の依存性管理ツールです、とっております。そこで我々が FuelPHP を見てみると、そうですね、えらく中途半端な感があります。ちょうどこの頃、こちらのスライドを拝見しまして。

- FuelPHP を composer に本気に対応させた時の話 from Keishi Hosoba

そうですね、今年の Advent Calendar にも参加されている @hosopy さんのスライドですね。

言いたいことは全部言われてしまっていますが、要約すると、「なんで全部 Composer で入ってくれないの？」と。利用者の立場から言うとですね、それ以外の立場ってあんまないですけど、「おいらのこの新しいプロジェクトは、FuelPHP に依存してるんですよ！ その辺 Composer 先生にはきっちりやっていただきたい！」

ということで。細羽さんのスライドを参考にしつつ、git submodule どころか、oil も使わないオレオレな composer.json を書いてみました。

### 脱 git submodule & oil

まず、git submodule は全て composer.json に加えることができます。

- fuel/core
- fuel/auth
- fuel/email
- fuel/oil
- fuel/orm
- fuel/parser

うん、でもですね、これだけじゃ動かないんで。fuel/fuel ってのがいますよね…。諸々のディレクトリ群と、config.php などの設定ファイル。これらを、【必要最低限】用意します。あ、そうそう、oil コマンドも必要なので入手しないと。

何しているかって、ただ mkdir して curl でダウンロードしてくるだけです。なんともスマートさのかけらも無い。

```

{
    "name": "chatii/fuelphp_setup",
    "type": "metapackage",
    "description": "FuelPHP Installer (not Oil)",
    "keywords": ["framework"],
    "homepage": "http://chatii.net",
    "license": "MIT",
    "authors": [
        {
            "name": "chatii",
            "email": "contact@chatii.net"
        }
    ],
    "support": {
        "source": "https://github.com/chatii/fuelphp_setup",
        "email": "contact@chatii.net"
    },
    "require": {
        "php": ">=5.3.3",
        "monolog/monolog": "1.5.*",
        "fuelphp/upload": "2.0.1",
        "fuel/core": "1.7",
        "fuel/auth": "1.7",
        "fuel/email": "1.7",
        "fuel/oil": "1.7",
        "fuel/orm": "1.7",
        "fuel/parser": "1.7"
    },
    "repositories": [
        {
            "type": "package",
            "package": {
                "name": "fuel/core",
                "type": "fuel-package",
                "version": "1.7",
                "require": {
                    "composer/installers": "*"
                },
                "source": {
                    "url": "https://github.com/fuel/core.git",
                    "type": "git",
                    "reference": "1.7/master"
                }
            }
        },
        {
            "type": "package",
            "package": {
                "name": "fuel/auth",
                "type": "fuel-package",
                "version": "1.7",
                "require": {
                    "composer/installers": "*"
                },
                "source": {
                    "url": "https://github.com/fuel/auth.git",
                    "type": "git",
                    "reference": "1.7/master"
                }
            }
        },
        {
            "type": "package",
            "package": {
                "name": "fuel/email",
                "type": "fuel-package",
                "version": "1.7",
                "require": {

```

```

        "composer/installers": "*"
    },
    "source": {
        "url": "https://github.com/fuel/email.git",
        "type": "git",
        "reference": "1.7/master"
    }
},
{
    "type": "package",
    "package": {
        "name": "fuel/oil",
        "type": "fuel-package",
        "version": "1.7",
        "require": {
            "composer/installers": "*"
        },
        "source": {
            "url": "https://github.com/fuel/oil.git",
            "type": "git",
            "reference": "1.7/master"
        }
    }
},
{
    "type": "package",
    "package": {
        "name": "fuel/orm",
        "type": "fuel-package",
        "version": "1.7",
        "require": {
            "composer/installers": "*"
        },
        "source": {
            "url": "https://github.com/fuel/orm.git",
            "type": "git",
            "reference": "1.7/master"
        }
    }
},
{
    "type": "package",
    "package": {
        "name": "fuel/parser",
        "type": "fuel-package",
        "version": "1.7",
        "require": {
            "composer/installers": "*"
        },
        "source": {
            "url": "https://github.com/fuel/parser.git",
            "type": "git",
            "reference": "1.7/master"
        }
    }
}
],
"extra": {
    "installer-paths": {
        "fuel/core/": ["fuel/core"],
        "fuel/packages/auth": ["fuel/auth"],
        "fuel/packages/email": ["fuel/email"],
        "fuel/packages/oil": ["fuel/oil"],
        "fuel/packages/orm": ["fuel/orm"],
        "fuel/packages/parser": ["fuel/parser"]
    }
},
"suggest": {

```

```

    "mustache/mustache": "Allow Mustache templating with the Parser package",
    "smarty/smarty": "Allow Smarty templating with the Parser package",
    "twig/twig": "Allow Twig templating with the Parser package",
    "mthaml/mthaml": "Allow Haml templating with Twig supports with the Parser package"
},
"config": {
    "vendor-dir": "fuel/vendor"
},
"scripts": {
    "post-install-cmd": [
        "mkdir -p public",
        "mkdir -p fuel/app/cache fuel/app/logs fuel/app/tmp",
        "mkdir -p fuel/app/classes/controller fuel/app/classes/model fuel/app/classes/view",
        "mkdir -p fuel/app/migrations fuel/app/modules fuel/app/tasks fuel/app/lang",
        "mkdir -p fuel/app/tests fuel/app/vendor fuel/app/views",
        "mkdir -p fuel/app/config/development fuel/app/config/production fuel/app/config/staging fuel/app/c
onfig/test",
        "if [ ! -e public/index.php ]; then curl https://raw.github.com/fuel/fuel/1.7/master/public/index.p
hp -o public/index.php; fi",
        "if [ ! -e public/.htaccess ]; then curl https://raw.github.com/fuel/fuel/1.7/master/public/.htacce
ss -o public/.htaccess; fi",
        "if [ ! -e fuel/app/bootstrap.php ]; then curl https://raw.github.com/fuel/fuel/1.7/master/fuel/app
/bootstrap.php -o fuel/app/bootstrap.php; fi",
        "if [ ! -e fuel/app/config/config.php ]; then curl https://raw.github.com/fuel/fuel/1.7/master/fuel
/app/config/config.php -o fuel/app/config/config.php; fi",
        "if [ ! -e fuel/app/config/db.php ]; then curl https://raw.github.com/fuel/fuel/1.7/master/fuel/app
/config/db.php -o fuel/app/config/db.php; fi",
        "if [ ! -e fuel/app/config/routes.php ]; then curl https://raw.github.com/fuel/fuel/1.7/master/fuel
/app/config/routes.php -o fuel/app/config/routes.php; fi",
        "if [ ! -e fuel/app/config/development/db.php ]; then curl https://raw.github.com/fuel/fuel/1.7/mas
ter/fuel/app/config/development/db.php -o fuel/app/config/development/db.php; fi",
        "if [ ! -e fuel/app/config/production/db.php ]; then curl https://raw.github.com/fuel/fuel/1.7/mast
er/fuel/app/config/production/db.php -o fuel/app/config/production/db.php; fi",
        "if [ ! -e fuel/app/config/staging/db.php ]; then curl https://raw.github.com/fuel/fuel/1.7/master/
fuel/app/config/staging/db.php -o fuel/app/config/staging/db.php; fi",
        "if [ ! -e fuel/app/config/test/db.php ]; then curl https://raw.github.com/fuel/fuel/1.7/master/fue
l/app/config/test/db.php -o fuel/app/config/test/db.php; fi",
        "curl https://raw.github.com/fuel/fuel/1.7/master/oil -o oil",
        "php oil r install"
    ]
},
"minimum-stability": "dev"
}

```

ださいですね。

使い方は簡単。簡単に README を書いています。

[https://github.com/chatii/fuelphp\\_setup](https://github.com/chatii/fuelphp_setup)

- `git clone git@github.com:chatii/fuelphp_setup.git` して
- `cd` で移って
- `rm -rf ./git/` で chatii/fuelphp\_setup の情報消して
- `git init` して
- `curl -sS https://getcomposer.org/installer | php` で Composer.phar 手に入れて
- `php composer.phar install` で FuelPHP をインストール

## 解消したかった手間

- oil でインストールすると、FuelPHP のリポジトリがめんどくさい
- core も含めてバージョン管理したい
- Vagrant と連携させてさくっと開発に着手したい

お、Vagrant? と思われた方、実は Vagrantfile と chef-solo のレシピを用意しました。

- [chatii/fuelphp\\_vagrant](#)

こちらも簡単ですが README を用意していますので参照してください。Vagrant についての説明は省きますが、簡単に使い方を。

この Vagrantfile のあるディレクトリに、chatii/fuelphp\_setup クローンしてあげてください。クローン時に名付けたディレクトリ名を、Vagrantfile の src\_dir に指定してあげます。また、ステージング環境を指定する場合は fuel\_env に指定してください。デフォルトでは development を指定しています。

あまり Vagrant、Chef についても詳しくありませんが、個人的には FuelPHP の開発開始までの手間がかなり削減されました。

無駄な Welcome も無いため、そのまま oil generate を使ってモデルなりコントローラーなり作ってもいいですし、自分で手打ちしてもいいですし。

## 終わりに

FuelPHP も 1.x 系が現行の 1.7 で開発は終わり、ということで、今回用意した composer.json もどうせ今だけしか使えない OR 使わないんじゃないかなあ、と思ってます。が、自分が作った物が誰かの目に触れる、役に立つことになれば、それはとても嬉しいので公開しました。

FuelPHP 2.0 では Composer への対応ってどの程度なんでしょうか?<sup>\*1</sup>情報を追っていないため、今回作ったものはすぐに用無しになるかもしれませんが、そのほうがいいと思ってます。

最後に。@hosopy さん、スライド勝手に拝借しました。この場を借りて、お礼申し上げます。ありがとうございました！

**chatii**

Twitter: [@chatii](#)

Blog: <http://chatii.net/>

<sup>\*1</sup> 【編注】 FuelPHP 2.0 からはすべてが Composer からインストールできるようになる予定です。ただし、2.0 の最初のリリース目標は 2014 年春とされており、現在はまだ Composer 経由でインストールできるようにはなっていません。

## FuelPHP で ChatWork パッケージを使ってみる

FuelPHP Advent Calendar 2013 12 日目です。@madmamor が担当します。

今日は、FuelPHP の ChatWork パッケージを紹介します。

ChatWork の API は、昨年 (2013 年 11 月) 末にプレビュー版として公開されました。

- <http://blog-ja.chatwork.com/2013/11/api-preview.html>

そこで早速、FuelPHP のパッケージとして実装してみました。

- <https://github.com/mp-php/fuel-packages-chatwork>

ChatWork パッケージのライセンスは MIT ライセンスです。

- <http://opensource.org/licenses/MIT>

では、準備と使い方の説明です。

### 1. ChatWork の API トークンを発行する

現在、ChatWork の API はプレビュー版なので、利用の申請が必要です。

- <http://blog-ja.chatwork.com/2013/11/api-preview.html>

の"お申し込み方法"の通りに、API の利用申請をします。後日、利用開始のメールが届きます。

利用開始のメールが届いたら、API トークンを発行します。

- <http://developer.chatwork.com/ja/authenticate.html>

の、"API トークンの取得"の通りです。発行された API トークンは後で使いますので、控えておいて下さい。

### 2. FuelPHP と ChatWork パッケージのインストール

FuelPHP のインストールを済ませて、トップページが閲覧可能な状態にします。以下の手順は FuelPHP1.7.1 で確認していますが、composer 対応以降のバージョンであれば問題無いと思います。

次に、composer.json の"require"に以下を追記します。

```
"mp-php/fuel-packages-chatwork": "dev-master"
```

次に、composer update します。

```
$ php composer.phar update
```



FuelPHP と ChatWork パッケージのインストールは以上です。尚、ChatWork の API を使用する関係で、curl と OpenSSL を有効にしておいて下さい。

### 3. ChatWork パッケージの設定

fuel/packages/chatwork/config/chatwork.php を fuel/app/config/ にコピーして、ChatWork の API トークンを設定します。

```
<?php

return array(
    'api_token' => 'ここに ChatWork の API トークンを設定します。',
);
```

次に、ChatWork パッケージを有効にします。方法は 2 つあります。

各所で ChatWork パッケージを使う場合は fuel/app/config/config.php の "always\_load.packages" に "chatwork" を追記します。

```
'always_load' => array(
    ...略...
    'packages' => array(
        ...略...
        'chatwork',
        ...略...
```

局所的に ChatWork パッケージを使う場合はその場所（や、そのクラスのコンストラクタ等）でロードします。

```
Package::load('chatwork');
```

これで、全ての準備が完了です。

### 4. ChatWork パッケージを使ってみる

自分の情報を取得してみます。

[http://developer.chatwork.com/ja/endpoint\\_me.html#GET-me](http://developer.chatwork.com/ja/endpoint_me.html#GET-me)

```
$response = Chatwork::get('/me');
Debug::dump($response);
```

指定したルームに投稿してみます。

[http://developer.chatwork.com/ja/endpoint\\_rooms.html#POST-rooms-room\\_id-messages](http://developer.chatwork.com/ja/endpoint_rooms.html#POST-rooms-room_id-messages)

```
$response = Chatwork::post('/rooms/[ルーム ID]/messages', array('body' => '内容'));
Debug::dump($response);
```

ルーム ID は、URL に含まれる "rid" に続く数値です。"rid0123456789" であれば、ルーム ID は "0123456789" になります。

どちらも簡単ですね。

今回は Chatwork::get() メソッドと Chatwork::post() メソッドを紹介しましたが、Chatwork::put() メソッドと Chatwork::delete() メソッドも用意してあります。

## 5. その他の API を使ってみる

現在、API は大きく分けると

- /me
- /rooms
- /my
- /contacts

の 4 種類で、それぞれ、更に細かく API が用意されています。

- <http://developer.chatwork.com/ja/endpoints.html>

どれを使うにしても

- HTTP メソッドは何か
- 必須パラメータは何か、任意パラメータにどんなパラメータがあるか

を確認すれば簡単に使えるので、ぜひ試してみてください。

最後に、現在の ChatWork API の認証はトークン式のみなので、自分（トークンの持主）が使う前提になっています。パッと思いつく使い方としては、GitHub の Hook から特定のルームにメッセージを投稿稿する。というような、通知の自動投稿でしょうか。タスクを管理する何か。も面白そうですね。

尚、今後、OAuth 2.0 式の認証も提供予定とのことですよ。

以上、ChatWork パッケージの紹介でした。

**@madmamor**

Developer & Bassist on TAMACENTER OUTSiDERS.

Twitter: [@madmamor](#)

Blog: <http://madroom-project.blogspot.jp/>

# FuelPHP (Twitter Bootstrap 3) で jQuery のプラグインの DataTables を使う

---

FuelPHP Advent Calendar 2013 の 13 日目です。

それでは今日のお題ですが DataTables です。

Web アプリを作るとき、業務系の管理画面を作るとき、FuelPHP の軽量さと作りやすさは強力です。私もここ最近の Web アプリケーションは FuelPHP で作ることが大半ですね。その中で特に管理画面を作るときは Twitter Bootstrap は非常に便利です。皆様も日頃お世話になっていますよね？

また今日ご紹介する DataTables も強力な jQuery のプラグインです。DataTables は HTML のテーブルタグを読み込み、ソートや検索を始め、多くの機能を提供します。この 2 つを組み合わせることで表出力が簡単で便利になるのでご紹介します。

- DataTables 公式サイト <http://datatables.net/>
- 日本語まとめ Wiki
- Twitter Bootstrap で DataTables を使う

ただ、Twitter Bootstrap も FuelPHP 1.7 で 2 から 3 に変わりました。Twitter Bootstrap 3 は 2 と互換性のない変更が多く、過去の資産が使いまわせません。

『はじめての Bootstrap (I・O BOOKS)』 槇俊明 著、工学社

※ Twitter Bootstrap 3 用の書籍で非常にわかりやすかったのでオススメです

DataTables も例に漏れず上記の方法では無理です。Twitter Bootstrap 3 の UI に合わせるのには公式サイト以外の方法が必要です。とは言ってもすでに 3 用のプラグインが作成されています。そこで DataTables の簡単な使い方と Twitter Bootstrap 3 の UI に合わせる方法をご紹介します。

まずは最新のソース・ファイルを FuelPHP に配置します。

- GitHub DataTables <https://github.com/DataTables/DataTables>

本体

- <https://github.com/DataTables/DataTables/blob/master/media/js/jquery.dataTables.js>

Bootstrap 用 CSS

- <https://github.com/DataTables/Plugins/blob/master/integration/bootstrap/3/dataTables.bootstrap.css>

Bootstrap 用 JS

- <https://github.com/DataTables/Plugins/blob/master/integration/bootstrap/3/dataTables.bootstrap.js>

Bootstrap 用 image

- <https://github.com/DataTables/Plugins/tree/master/integration/bootstrap/images>

これらを public/asset/のそれぞれに配置します。

ここで注意があります。FuelPHP の画像を置く場所はデフォルトは img です。ですが CSS のファイルパスは images になっています。これを下記のとおり変更します。

```
table.dataTable thead .sorting { background: url('../images/sort_both.png') no-repeat center right; }
table.dataTable thead .sorting_asc { background: url('../images/sort_asc.png') no-repeat center right; }
table.dataTable thead .sorting_desc { background: url('../images/sort_desc.png') no-repeat center right; }

//変更

table.dataTable thead .sorting { background: url('../img/sort_both.png') no-repeat center right; }
table.dataTable thead .sorting_asc { background: url('../img/sort_asc.png') no-repeat center right; }
table.dataTable thead .sorting_desc { background: url('../img/sort_desc.png') no-repeat center right; }
```

これで asset/img に配置してもソート用の画像が読み込まれます。

あとは Table に ID を降って

```
var Table = $('#table_id').dataTable();
```

とすれば自動的に読み込んで整形してくれます。これでよく要望に上がるソート・ページャー・サーチを一度に対応することが出来ます。

更により Bootstrap っぽくするには

```
/**
 * Copyright (c) 2013 @soudai1025
 * License: MIT
 * http://opensource.org/licenses/mit-license.php
 */

var Table = $('#table_id').dataTable({
  // 日本語対応したい場合は GitHub の plugin から該当のファイルをダウンロードしてください
  oLanguage: {
    sUrl: "/assets/js/plugins/i18n/Japanese.lang"
  },
  // Table 生成後に実行されます
  fnInitComplete: function() {
    $('#table_id').each(function() {
      var datatable = $(this);
      // サーチの input タグをより BootstrapUI よりに
      var search_input = datatable
        .closest('.dataTables_wrapper')
        .find('div[id$=_filter] input');
      search_input.attr('placeholder', 'Search')
        .addClass('form-control input-sm');
      // 表示列数の表示をより BootstrapUI よりに
      var length_sel = datatable
        .closest('.dataTables_wrapper')
        .find('div[id$=_length] select');
      length_sel.addClass('form-control input-sm')
        .change(function() {
          oFC.fnRedrawLayout();
        });
      oFC.fnRedrawLayout();
    });
  }
});
```

```
    }
  });
```

とするといい感じになります。

他にも DataTables には多様な追加機能が plugin として用意されています。公式サイトは情報が古かったりするので GitHub のサンプルの index.html も参考にしてみてください。

さて FuelPHP の良いところ言えば REST Controller ですよね。

また DataTables はデータを JSON で受け取ることが出来ます。大きい data でも Ajax で JSON を受けることができるので肥大した Table でも対応することが出来ます。今日は 13 日の金曜日ですし JSON を使った例もご紹介します。

簡単な例ですと

```
/**
 * Copyright (c) 2013 @soudai1025
 * License: MIT
 * http://opensource.org/licenses/mit-license.php
 */

var Table = $('#table_id').dataTable({
  // 読み込み中の表示
  bProcessing: true,
  // data を全て読み込む前に表示を始める
  bDeferRender: true,
  // 読み込む JSON の URL (REST Controller の URL)
  sAjaxSource: "/api/dataTables.json",
  // 配列名 (デフォルトは aadata)
  sAjaxDataProp: "setJson",
  // 表示列と json の項目をマッピングします
  aoColumns: [
    { mData: "id", sDefaultContent: "" },
    { mData: "name", sDefaultContent: "" },
  ],
  oLanguage: {
    sUrl: "/assets/js/plugins/i18n/Japanese.lang"
  },
  fnInitComplete: function() {
    $('#table_id').each(function() {
      var datatable = $(this);
      // サーチの input タグをより BootstrapUI よりに
      var search_input = datatable
        .closest('.dataTables_wrapper')
        .find('div[id$=_filter] input');
      search_input.attr('placeholder', 'Search')
        .addClass('form-control input-sm');
      // 表示列数の表示をより BootstrapUI よりに
      var length_sel = datatable
        .closest('.dataTables_wrapper')
        .find('div[id$=_length] select');
      length_sel.addClass('form-control input-sm')
        .change(function() {
          oFC.fnRedrawLayout();
        });
      oFC.fnRedrawLayout();
    });
  }
});
```

として REST Controller は

```
public function action_dataTables() {
    $aadata['setJson'][] = ['id'=>1, 'name'=>'hage'];
    $aadata['setJson'][] = ['id'=>1, 'name'=>'fuga'];
    return $this->response($aadata);
}
```

といった感じでいつもどおり配列を渡して JSON を生成するだけです。

HTML は次の通り。

```
<table id="table_id" class="table table-bordered table-hover table-striped">
  <thead>
    <tr>
      <th>id</th>
      <th>name</th>
    </tr>
  </thead>
  <tbody>
    </tbody>
</table>
```

とすると tbody の内容を JSON を元に自動生成してくれます。

もちろん作りこみ次第で例の最初に全て読み込むのではなく、API と View が連携して適時やりとりすることが出来ます。この API 側が非常に簡単につくれるのは FuelPHP のメリットです。

以上のように FuelPHP と DataTables の相性は抜群です。是非、公私でアプリ作成時に試してみてください！

**@soudai1025**

Twitter: [@soudai1025](https://twitter.com/soudai1025)

Blog: <http://soudai1025.blogspot.jp/>

## Request\_Curl にまつわるエトセトラ

FuelPHP Advent Calendar 2013 14 日目。

@sharkpp です。

2 回目の FuelPHP Advent Calendar 2013 登場となります。

### Request\_Curl 使っていますか？

さて、Request\_Curl 使ってますか？

えっ？ Guzzle のが便利だからそっち使ってるですって？

まあ、そう言わずに Request\_Curl は標準で含まれているので使ってみませんか？

簡単な使い方

```
$url = 'http://www.example.net/';
$curl = \Request::forge($url, 'curl');
$curl->execute();
$response = $curl->response();
echo $response->body;
```

ね！ 簡単でしょ？

### GET / POST 時のパラメータ指定

GET / POST 時のパラメータの指定は通常であれば、

```
$param['user'] = 'john';
$param['data'] = 'test';
$curl->set_params($param);
```

で問題ありません。

が、`http://www.example.net/?user=john&user=smith` のように同じキーが複数存在する場合は先の方法ではうまくいきません。そもそも、そんな指定はありえない？ いえいえ、実際にこのような指定をするアプリケーションがありました。

そんな場合は、

```
// Copyright (c) 2013 sharkpp
// This function is released under the MIT License.
// http://opensource.org/licenses/mit-license.php

function build_query($data) {
    array_walk($data, function (&$value, $key) {
        is_array($value) ? $value = array($value);
        $value = array_map(function($value){ return urlencode($value); }, $value);
    });
}
```

```
        $value = implode('&'.$key.'=', $value);
        $value = $key.'='.$value;
    });
    return implode('&', $data);
}
```

のようなクエリ文字列の構築関数を使って

```
$param['user'] = array('john', 'smith');
$curl->set_params(build_query($param));
```

とすれば OK です。

実はドキュメントに書かれていないですが、`Request_Curl::set_params()` の引数に文字列を渡すとクエリ文字列としてそのまま使ってくれます。

## Cookie はおいしい？

Cookie の与え方も簡単です。

```
// Copyright (c) 2013 sharkpp
// This function is released under the MIT License.
// http://opensource.org/licenses/mit-license.php

protected static function build_cookie($data) {
    if (is_array($data)) {
        $cookie = '';
        foreach ($data as $key => $value) {
            $cookie[] = $key.'='.urlencode($value);
        }
        if (count($cookie) > 0) {
            return trim(implode('; ', $cookie));
        }
    }
    return false;
}
```

こんな関数を用意して

```
$cookie['hoge'] = 'test';
$cookie['fuga'] = '1234';
$curl->set_option(CURLOPT_COOKIE, build_cookie($cookie));
```

とすれば OK です。

## PHP さんですか？ いえいえ IE11 です

User Agent 略して UA の偽装もちろんできます。

```
$UA = 'Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko';
```



```
$header['User-Agent'] = $UA;  
foreach ($header as $key => $value) { $curl->set_header($key, $value); }
```

ちなみに、ヘッダの複数指定は出来ないようなので foreach で連想配列を処理して登録しています。一回ずつ Request\_Curl::set\_header() を呼び出してもいいですが、foreach の方が見やすい気がします。

## SSL が検証できない？ 大丈夫だ、問題ない

まったくもって大丈夫じゃないですが、、そんな時もあります。

HTTPS なサーバーに対してアクセスする場合に、どうにもエラーが出てうまくいかない場合があります。

本来は無効にすべきではないのですが、SSL の証明書の検証を無効にすることも出来ます。

```
$curl->set_option(CURLOPT_SSL_VERIFYPEER, false);
```

本来は

```
$curl->set_option(CURLOPT_CAINFO, 'path/to/cacert.pem');
```

のような感じで検証用のファイルを指定するようですがうまくいきませんでした。

## あれ？

あれ？ ちょっと Request\_Curl::set\_option() がいっぱい出てくるのだけど…

あ、気が付きましたか。名前の通りと言ったところではあるのですが、curl\_\* のラッパーになっているため、**PHP: curl\_setopt - Manual** 辺りを見ながら Request\_Curl::set\_option() に引数を与えてあげれば色々な事が出来ます。

クラス内部で色々やっているのですべてのオプションが確実に指定できるとは限らないですがある程度は自由に出来るようです。

ということで、Request\_Curl クラスの紹介でした。

### @sharkpp

へっぽこプログラマ。手羽先が有名な所に在住な PHP と C++ のプログラマです。

Twitter: [@sharkpp](#)

Blog: <http://www.sharkpp.net/>

## 続・Cloudn PaaS で FuelPHP を動かしてみた

この記事は [FuelPHP Advent Calendar 2013](#) の 15 日目です。

本日の記事を担当する [@Tukimikage](#) です。この記事は以前公開した「[Cloud\(n\)PaaS で FuelPHP を動かしてみた](#)」をベースに中身をブラッシュアップしたものとなります。尚、本記事のコードや各種コマンドは Mac OS X 10.9 で動作確認しています。

### 1. Cloudn PaaS とは

NTT コミュニケーションズが [Cloudn](#) のサービス・メニューの 1 つとして提供する [Platform as a Service](#) です。最小構成であれば 1 インスタンスあたり月額上限 525 円で利用可能です。

### 2. デプロイ環境の準備

前回の記事では VMC (VMware Cloud CLI) を利用していましたが、今回は 2013/09/26 に公開された **Cloudn PaaS UDN** (コマンドラインツール) を利用します。UDN は ruby が入っていれば gem を利用してインストール可能です。

```
$ sudo gem install udn
Password:
Fetching: json_pure-1.6.8.gem (100%)
Successfully installed json_pure-1.6.8
Fetching: rubyzip-0.9.9.gem (100%)
Successfully installed rubyzip-0.9.9
:
:
Installing ri documentation for rb-readline-0.4.2
Parsing documentation for udn-0.3.23.3
Installing ri documentation for udn-0.3.23.3
10 gems installed
```

インストールが完了したらログインしましょう。ログインに必要なメールアドレスとパスワードは Cloudn PaaS の公式操作マニュアルをご覧ください。

```
$ udn login
Attempting login to [http://api.cloudnpaas.com]
Email:
Password:
Successfully logged into [http://api.cloudnpaas.com]
```

### 3. FuelPHP の開発環境準備

今回の開発環境準備には、[@chatii0079](#) さんが作成された `fuelphp_setup` を利用します。`fuelphp_setup` については、[FuelPHP Advent Calendar 2013 11 日目「FuelPHP をもっと Composer で使う」](#)をご覧ください。

Cloudn PaaS は 2013/12/15 現在 PHP v5.3 に対応していますが、PHP のフレームワークには全く対応していません。(個

人的にはかなり残念ですが…)

FuelPHP を利用する場合は Document Root の変更が必要になります。ユーザ設置の.htaccess が動作するため、FuelPHP プロジェクトのディレクトリトップに以下の通りファイルを作成します。

```
RewriteEngine On
RewriteBase /
RewriteRule ^(.*)$ /public/index.php [L]
```

@chatii0079 さんが GitHub で公開されている [fuelphp\\_setup](#) を Fork して、.htaccess ファイルを追加した [fuelphp\\_setup\\_for\\_cloudn\\_paas](#) を作成させていただきました。@chatii0079 さん、ありがとうございます！

GitHub からのプロジェクト作成方法は、[fuelphp\\_setup](#) の手順と同様です。

```
$ git clone https://github.com/Y-NAKA/fuelphp_setup_for_cloudn_paas.git project_name
$ cd project_name
$ rm -rf ./.git/
$ git init
$ curl -sS https://getcomposer.org/installer | php
$ php composer.phar install
```

## 4. テストアプリ作成

今回は動作確認のためにテスト用クラスを生成します（ディレクトリ構成はダミーです）。

```
$ oil g controller main index
Creating view: /project_name/fuel/app/views/template.php
Creating view: /project_name/fuel/app/views/main/index.php
Creating controller: /project_name/fuel/app/classes/controller/main.php
```

テスト用クラスを読み込むためにルーティング設定を変更します。[fuelphp\\_setup](#) を利用して環境構築した場合でも、Welcome コントローラーが存在する前提のルーティングになっているため、必要ないところは削除し root のルーティング先を書き換えます。

ファイル名：/project\_name/fuel/app/config/routes.php

```
<?php
return array(
    '_root_' => 'main/index', // The default route
);
```

oil で生成されたテンプレートファイルには bootstrap.css をインポートする記述があるため、public ディレクトリに必要なサブディレクトリとファイルを設置します。

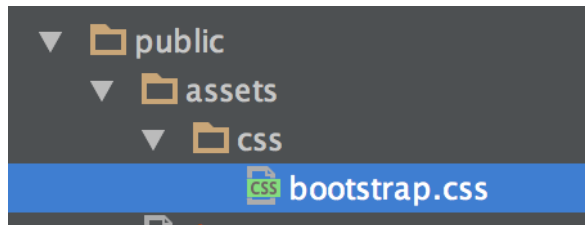


図 15.1 bootstrap.css を設置

## 5. デプロイ

では、早速デプロイしていきます。以下の例では **testapp1** という名前でアプリケーションを作成しています。

ポイントは **Detected a Standalone Application, is this correct?** という質問に **No** で答えて、次の言語／FW 選択で **9: PHP** を選ぶことです。Deployed URL やインスタンスに割り当てる性能要件などはアプリの要件に合わせて適宜変更してください。

```
$ cd project_name
$ udn push testapp1
Would you like to deploy from the current directory? [Yn]:
Detected a Standalone Application, is this correct? [Yn]: n
1: Rails
2: Spring
3: Grails
4: Lift
5: JavaWeb
6: Standalone
7: Sinatra
8: Node
9: PHP
10: WSGI
11: Django
12: Rack
13: Play
Select Application Type: 9
Selected PHP Application
Application Deployed URL [testapp1.cloudnpaas.com]:
Memory reservation (128M, 256M, 512M, 1G, 2G) [128M]:
How many instances? [1]:
Bind existing services to 'testapp1'? [yN]:
Create services to bind to 'testapp1'? [yN]:
Would you like to save this configuration? [yN]:
Creating Application: OK
Uploading Application:
  Checking for available resources: OK
  Processing resources: OK
  Packing application: OK
  Uploading (8M): OK
Push Status: OK
Staging Application 'testapp1': OK
Starting Application 'testapp1': OK
```

## 6. 動作確認

デプロイ後はブラウザから動作確認を行ってください。

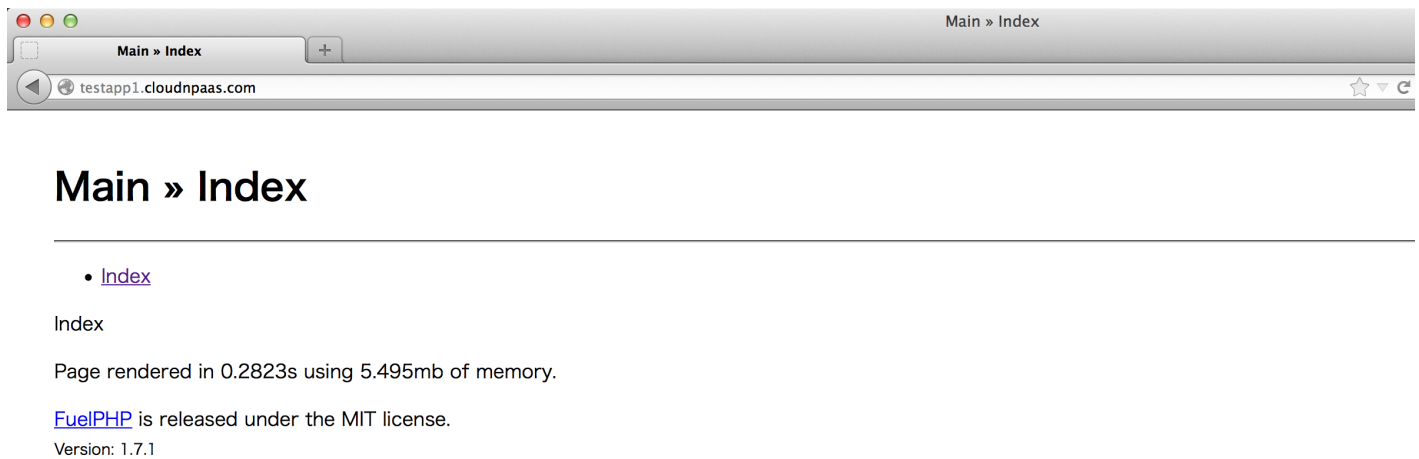


図 15.2 動作確認

余談ですが、Cloudn PaaS 含め、Cloudn の様々な機能をコントロールできる Web の管理 UI は、Firefox が推奨ブラウザとなっています。Chrome など他のブラウザで正常に動かない機能があるので注意してください。

## 7. アプリケーションのアクセス制限

開発中のアプリや一般公開する必要があるアプリは、IP アドレスでアクセス制限をかけることができます。  
アクセス制限設定（IP アドレスは CIDR 記法で記載し、カンマで区切ることで複数指定可能）

```
$ udn env-add testapp1 ALLOW_CIDR_WHITELIST="*, *, *, */*"
Adding Environment Variable [ALLOW_CIDR_WHITELIST=*, *, *, */*]: OK
Stopping Application 'testapp1': OK
Staging Application 'testapp1': OK
Starting Application 'testapp1': OK
```

アクセス制限状況確認

```
$ udn env testapp1
ALLOW_CIDR_WHITELIST *, *, *, */*
```

アクセス制限解除（一度登録した内容を変更する場合は一度解除した上で新規設定）

```
$ udn env-del testapp1 ALLOW_CIDR_WHITELIST
Deleting Environment Variable [ALLOW_CIDR_WHITELIST]: OK
Stopping Application 'testapp1': OK
Staging Application 'testapp1': OK
Starting Application 'testapp1': OK
```

アクセス制限中は以下のように表示されます。

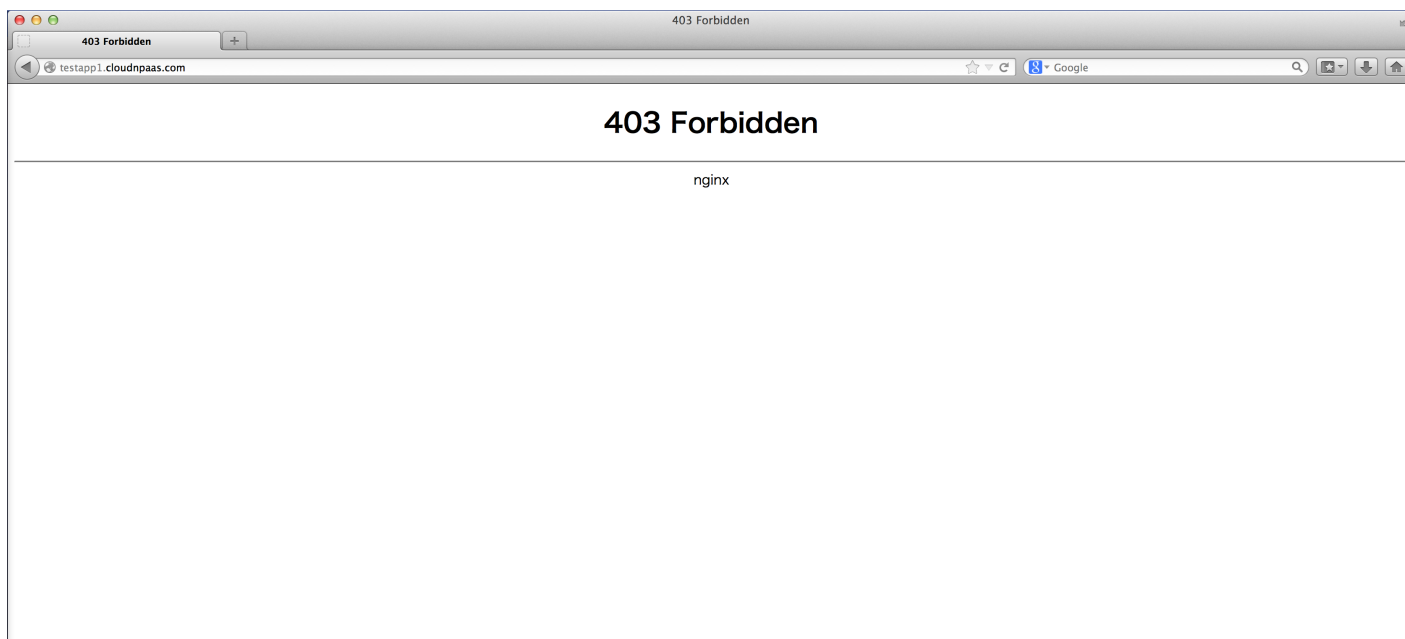


図 15.3 アクセス制限

### アクセス制限に関する注意事項

.htaccess にて IP アドレス制限や Basic 認証を行うこともできますが、Cloudn PaaS のサーバは、ユーザアプリが動作するアプリサーバと、インターネットからのアクセスを受け付けるプロキシサーバという 2 段構えになっているようです。そのため、アプリサーバ上の .htaccess で単純にアクセス制限すると正常に動かなくなります。

## 8. アプリケーションの更新

アプリの更新は、更新するファイルが格納されているディレクトリにて、以下の通りコマンドを実行します。ファイルの更新とアプリの再起動を行ってくれます。

```
$ cd project_name
$ udn update testapp1
Uploading Application:
  Checking for available resources: OK
  Processing resources: OK
  Packing application: OK
  Uploading (92K): OK
Push Status: OK
Stopping Application 'testapp1': OK
Staging Application 'testapp1': OK
Starting Application 'testapp1': OK
```

## 9. おわりに

今回は Cloudn PaaS にて FuelPHP を動かす方法を、前回記事にした部分からのアップデートを交えて解説しました。

FuelPHP の開発環境自体も、[@chatii0079](#) さんの [fuelphp\\_setup](#) を利用することでかなり簡単に整えることができます。大規模場システムを作る場合は役不足な感じがある PaaS ですが、簡単な Web ツールを動かしたい、手早く動作検証してみたいなど、利用シーンによってはかなり便利に使えるのではないのでしょうか。

FuelPHP Cloudn に関しては今後も記事を書いていきたいと思いますので、ご期待ください。

**Yusuke NAKA**

Web デベロッパー、サーバーエンジニア、技術コミュニティ運営・支援などをやっています。

Twitter: [@Tukimikage](#)

Blog: <http://think-sv.net/blog/>

## FuelPHP の module を使いこなす

こんにちは、**hosopy** です。

**FuelPHP Advent Calendar 2013** 16 日目を担当させていただきます。

本日は、「FuelPHP の module を使いこなす」のお話をしたいと思います。

### FuelPHP の module のおさらい

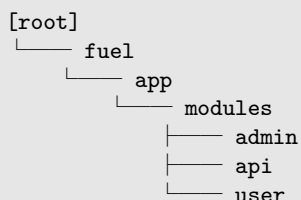
FuelPHP の module については、ここでは割愛させていただきます。「module とは何だっけ？」な方は、下記のサイトでざっくりとおさらいしておきましょう。

- <http://fuelphp.com/docs/general/modules.html>
- <http://d.hatena.ne.jp/dix3/20111212/1323660316>

### テーマ

それでは本題。「使いこなす」と風呂敷を広げてしまいましたが、内容としては「module の階層化」になります。

まず、以下のような FuelPHP のモジュール構成を考えます。



このようなモジュール構成をとり始めると、以下のような URL の構成を実現したくなったりします。

- /admin/controller/action/params
  - admin モジュールの機能が動作する
- /api/controller/action/params
  - api モジュールの機能が動作する
- /user/controller/action/params
  - user モジュールの機能が動作する
- /admin/user/controller/action/params
  - user モジュールの機能が、admin モジュールのコンテキストで動作する
- /api/user/controller/action/params
  - user モジュールの機能が、api モジュールのコンテキストで動作する

つまり、admin、api という基本モジュールの中に、user モジュールをアドオンしていくというイメージです。



「いや、自分は/user/admin になってもいいわ」という方はここで終了ですw

## サンプルコード

GitHub にサンプルコードを置きました。

[https://github.com/hosopy/fuel\\_module\\_sample](https://github.com/hosopy/fuel_module_sample)

README.md の再掲になりますが、開発サーバを動作させて URL を叩くと、以下の様な挙動を示すと思います。

- /admin
  - admin モジュールの root/index アクションが呼ばれます
  - これは、fuel/app/modules/admin/config/routes.php の \_root\_ 定義によるものです
- /admin/user
  - user モジュールの admin/index アクションが呼ばれます。
- /admin/user/analytics
  - user モジュールの admin/analytics/index アクションが呼ばれます。
- /api
  - api モジュールの root/index アクションが呼ばれます
  - これは、fuel/app/modules/api/config/routes.php の \_root\_ 定義によるものです
- /api/user
  - user モジュールの api/index アクションが呼ばれます。
- /api/user/analytics
  - user モジュールの api/analytics/index アクションが呼ばれます。
- /user
  - user モジュールの root/index アクションが呼ばれます
  - これは、fuel/app/modules/user/config/routes.php の \_root\_ 定義によるものです

## ポイント

fuel/app/modules/admin/classes/controller/admin.php の処理が全てです。

/admin/\*\*/\* なる URL へのリクエストを、まずは Controller\_Admin で受け取り、適切なモジュールへ HMVC でリクエストを転送しています。

fuel/app/modules/admin/classes/controller/admin.php

```
<?php
namespace \Admin;

class Controller_Admin extends \Controller_Hybrid
{
    /**
     * /admin/module/controller/action/params への HTTP リクエストを、
     * /module/admin/controller/action/params への HMVC リクエストとして転送する
     *
     * @Override
     */
    public function router($resource, $arguments)
    {
        if (\Module::loaded($resource))
        {
            return \Request::forge($resource.'/admin/'.join('/', $arguments))->execute()->response;
        }
    }
}
```

```
        else
        {
            return parent::router($resource, $arguments);
        }
    }
}
```

api モジュールでも、全く同様のことを行っています。

## 何が嬉しいか？

### コードがスッキリする

モジュール間の連携処理を app 内でゴニョゴニョ実装する手もありますが、こちらのほうがソースの構成がスッキリすると思います。

## admin モジュールと user モジュールの疎結合

user モジュールは、admin モジュールの提供する機能を存分に利用してコーディングを行う事ができますが、admin モジュールからは、user モジュールを前提としたコーディングはしない方がよいでしょう。

この点については、設定ファイルや PHP の interface を活用して、モジュール間の実装インターフェースの規約が必要になると思います。

実際、自分の仕事では、admin モジュールの管理画面テンプレートへ組み込む View の受け渡し方法として、適切な interface を実装した ViewModel を利用するなど、実装上の疎結合を保っています。

ただし、やり過ぎると分かりにくくなるのでホドホドに…

## 最後に

ここまでお読み頂き、ありがとうございます。少しでも皆様のお役に立つことが出来れば幸いです。

@hosopy

Twitter: @hosopy

Blog: <http://qiita.com/hosopy/items/0428be74f1c3868c55ba>

## レンタルサーバー XREA/CORESERVER で FuelPHP を使う（実践編）

この記事は [FuelPHP Advent Calendar 2013](#) の 17 日目の記事として公開します。

先々月に「[レンタルサーバー XREA/CORESERVER で FuelPHP を動かす](#)」という記事を書きました。今回はその続きとして、もう少し実践的な内容をお届けします。前回の記事と併せてお読みください。

本稿の内容は s110.coreserver.jp 上の PHP 5.4.21 で FuelPHP 1.7.1 を使って動作確認をしてあります。以降、ユーザー名を hogefuga、サーバーを s1024.coreserver.jp と仮定して説明します。みなさんのアカウント情報に適宜読み換えながら試してみてください。

### ローカルの開発環境でデータベースを使う

XREA/CORESERVER では MySQL と PostgreSQL が利用可能ですが、セキュリティの理由からか、同一サーバーからのアクセス以外は弾かれるようで、ローカルの開発環境からサーバー上のデータベースを読み書きすることはできません。そこで開発用のデータベースをローカルに立て、ローカル環境ではローカルデータベースを、サーバー上の運用環境ではサーバーのデータベースを参照するようにしましょう。

まず、サーバー上の環境を production（運用環境）に設定しましょう。FuelProject/public/.htaccess の先頭、「AddHandler application/x-httpd-php54cgi .php」の上あたりに、次の行を追加します。

```
SetEnv FUEL_ENV production
```

.htaccess はローカルとサーバーでは記述が異なるので、不用意な上書きに注意してください。

これでローカル環境は development、サーバー環境は production と別々の FUEL\_ENV が設定されました。では、それぞれの環境用に db.php を作ります。

私は PostgreSQL が好きなので、基本的にサーバーにも PostgreSQL でデータベースを作成しています。ここでは PostgreSQL の例を挙げますが、MySQL でも同様に設定できますので、参考にしてください。

まず、fuel/app/config/db.php に共通設定を書きます。

```
<?php
return [
    'default' => [
        'type' => 'pdo',
        'identifier' => '',
        'table_prefix' => '',
        'charset' => 'utf8',
        'enable_cache' => true,
        'profiling' => false,
    ],
];
```

続いてサーバー側の設定を `fuel/app/config/production/db.php` に記述しましょう。ホスト名は `s1024.coreserver.jp` とせず、`localhost` にします。

```
<?php

return [
    'default' => [
        'connection' => [
            'dsn'          => 'pgsql:host=localhost;dbname=hogefuga',
            'username'     => 'hogefuga',
            'password'     => 'your_password',
            'persistent'   => false,
            'compress'     => false,
        ],
    ],
];
```

同様に、ローカル環境の設定を `fuel/app/config/development/db.php` に記述します。

```
<?php

return [
    'default' => [
        'connection' => [
            'dsn'          => 'pgsql:host=localhost;dbname=hogefuga_local',
            'username'     => 'hogefuga_local',
            'password'     => 'your_local_password',
            'persistent'   => false,
            'compress'     => false,
        ],
    ],
];
```

ローカルに立てたデータベースの DB 名とユーザー名、パスワードをサーバーのものと同じにすれば設定を分けずに済みますが、ローカル環境で複数のデータベースを切り替えて開発できるようにしておくほうがよいでしょう。

## タスクと Email パッケージを利用してログファイルの肥大化を防ぐ

`fuel/app/logs` に日々蓄積されるログファイルを、FuelPHP を使って定期的に掃除させましょう。

次のような仕様を考えます。

- 日付が変わったら、前日のログファイルをメールで送信する。
- 1 週間前のログファイルは削除する。その際、カラになったディレクトリも削除する。

cron ジョブによるバッチ処理を毎晩走らせましょう。FuelPHP ではタスクという機能でバッチ処理が実現できます。

適当なプロジェクトを作成し、`fuel/app/tasks/logsender.php` というファイルを作成します。プロジェクトの設置場所は `~/php/logsender` とします。

```
<?php
/*
 * Copyright (c) 2013 SUNOHARA, Hiroyasu
 * This software is released under the MIT License.
 * http://opensource.org/licenses/mit-license.php
 */
```

```

/**
 * FuelPHP の前日分ログファイルをメールで送信し、1 週間前のログファイルを削除するタスク。
 */

namespace Fuel\Tasks;

\Package::load('email');

/**
 * ログのメール送信と古いログの削除を行うクラス。
 */
class LogSender
{
    /**
     * バッチ本体
     */
    public static function run()
    {
        // メールヘッダー情報
        $address_from = 'hogefuga@s1024.coreserve.jp';
        $name_from = 'FuelPHP Task';
        $address_to = 'hogefuga@s1024.coreserver.jp';
        $name_to = '俺';

        $yesterday = date('Y/m/d', time() - 86400);
        $one_week_ago = date('Y/m/d', time() - 86400 * 7);

        $log_dir = '/virtual/hogefuga/php/logsender/fuel/app/logs/';
        $yesterday_log_name = $log_dir . $yesterday . '.php';

        // メール送信
        $subject = "FuelPHP Log - $yesterday.php";

        $email = \Email::forge();
        $email->from($address_from, $name_from);
        $email->to($address_to, $name_to);
        $email->subject($subject);
        if (file_exists($yesterday_log_name)) {
            $email->body('');
            $email->attach($yesterday_log_name);
        } else {
            $email->body('昨日のログファイルはありません。');
        }
        try {
            $email->send();
        } catch (Exception $ex) {

        }

        // 1 週間前のログを削除
        $log_to_delete = $log_dir . $one_week_ago . '.php';
        if (file_exists($log_to_delete)) {
            if (! unlink($log_to_delete)) {
                echo 'ファイル削除に失敗' . PHP_EOL;
            }
            static::remove_directory(dirname($log_to_delete));
            static::remove_directory(dirname(dirname($log_to_delete)));
        }
    }

    /**
     * ディレクトリを削除する。ディレクトリ内にファイルがある時は何もしない。
     * @param string $dir ディレクトリ名
     */
    private static function remove_directory($dir)
    {
        $files = scandir($dir);
        if (count($files) === 2) {

```

```

        rmdir($dir);
    }
}
}

```

このタスクをコマンドラインから実行する際、以下のコマンドではエラーが返ってきます。

```
php ~/php/logsender/oil r logsender
```

本稿執筆時点では XREA（一部のサーバーを除く）と CORESERVER は PHP のバージョンが 5.2 なので、FuelPHP は動かないのでした。PHP をフルパスで指定しましょう。

ここでサーバーにログインし<sup>\*1</sup>、ホームディレクトリで `ls /usr/local/bin/php*` を実行してみてください。以下のような表示になると思います。

```

hogefuga@si024:~> ls /usr/local/bin/php*
/usr/local/bin/php          /usr/local/bin/php-5.2.4cli  /usr/local/bin/php-5.4.19cli  /usr/local/bin/php55-conf
/usr/local/bin/php4         /usr/local/bin/php-5.2.5    /usr/local/bin/php-5.4.21    /usr/local/bin/php55size
/usr/local/bin/php-4.4.7    /usr/local/bin/php-5.2.5cli /usr/local/bin/php-5.4.21cli /usr/local/bin/php5cli
/usr/local/bin/php-4.4.7cli /usr/local/bin/php52cli     /usr/local/bin/php-5.4.5     /usr/local/bin/php5-confi
/usr/local/bin/php-4.4.8    /usr/local/bin/php53        /usr/local/bin/php-5.4.5cli  /usr/local/bin/php5ize
/usr/local/bin/php-4.4.8cli /usr/local/bin/php-5.3.15    /usr/local/bin/php-5.4.7     /usr/local/bin/php6
/usr/local/bin/php4cli      /usr/local/bin/php-5.3.15cli /usr/local/bin/php-5.4.7cli  /usr/local/bin/php6cli
/usr/local/bin/php4-config  /usr/local/bin/php-5.3.17    /usr/local/bin/php54cli     /usr/local/bin/php6-confi
/usr/local/bin/php4ize      /usr/local/bin/php-5.3.17cli /usr/local/bin/php54-config  /usr/local/bin/php6ize
/usr/local/bin/php5         /usr/local/bin/php-5.3.27    /usr/local/bin/php54ize     /usr/local/bin/php_back
/usr/local/bin/php52        /usr/local/bin/php-5.3.27cli /usr/local/bin/php55        /usr/local/bin/php-cgi
/usr/local/bin/php-5.2.2    /usr/local/bin/php53cli     /usr/local/bin/php-5.5.3     /usr/local/bin/php-config
/usr/local/bin/php-5.2.2cli /usr/local/bin/php53-config  /usr/local/bin/php-5.5.3cli  /usr/local/bin/phpize
/usr/local/bin/php-5.2.3    /usr/local/bin/php53ize     /usr/local/bin/php-5.5.5     /usr/local/bin/phpize
/usr/local/bin/php-5.2.3cli /usr/local/bin/php54        /usr/local/bin/php-5.5.5cli  /usr/local/bin/php5cli
/usr/local/bin/php-5.2.4    /usr/local/bin/php-5.4.19    /usr/local/bin/php55cli
hogefuga@si024:~>

```

FuelPHP の実行には PHP 5.3 以上であればいいので、好きなバージョンの PHP を指定しましょう。

```
/usr/local/bin/php-5.5.5cli ~/php/logsender/oil r logsender
```

これを実行するシェルスクリプトを適当な場所に設置します。ここでは `~/cron_logsender.sh` として置くことにします。改行コードを LF にするのを忘れないようにしてください。

```

#!/bin/sh
/usr/local/bin/php-5.5.5cli ~/php/logsender/oil r logsender
exit

```

これを cron ジョブで起動するように設定します。CORESERVER のコントロールパネルにログインして、左メニューの「CRON ジョブ」をクリックし、「CRON ジョブの編集」画面を開きます。

<sup>\*1</sup> 「[レンタルサーバー XREA/CORESERVER で FuelPHP を動かす](#)」の「3. シンボリックリンクの作成」を参照してください。



図 17.1 CORESERVER コントロールパネル メニュー

CRONジョブの編集(hogefuga)					
設定0					
分	時	日	月	曜日	
30	0	*	*	*	
/virtual/hogefuga/cron_logsender.sh >/dev/null 2>&1					
設定1					
分	時	日	月	曜日	
/virtual/hogefuga/					
設定2					

図 17.2 CORESERVER コントロールパネル CRON ジョブ設定

空いている設定欄に、以下のように入力します。

- ・ 分 → 30
- ・ 時 → 0
- ・ 日 → \*
- ・ 月 → \*
- ・ 曜日 → \*

その下にある「/virtual/hogefuga/」に続くテキストボックスには、

```
cron_logsender.sh > /dev/null 2>&1
```

と入力します。これで、毎日 0:30 に先ほどのタスクが実行されます。

メール送信時にエラーが発生したときの処理などは、みなさんで追加してみてください。

## 【12/18 追記】

添付ファイルのサイズの制限について、Fumito Mizuno さんから質問をいただきました。公式発表はないようなので、実際にどのくらいまで添付で送れるか、簡単なコードを書いて調べてみました。

1MB から 1MB 刻みで添付ファイルのサイズを増やして送信したところ、こんなエラーメッセージが出ました。

```
Fatal error: Allowed memory size of 94371840 bytes exhausted (tried to allocate 28329953 bytes) in /virtual/hogefuga/php/logsender/fuel/packages/email/classes/email/driver.php on line 965
```

```
Fatal Error - Allowed memory size of 94371840 bytes exhausted (tried to allocate 28329953 bytes) in PKG-PATH/email/classes/email/driver.php on line 965
```

19MB の添付ファイルがついたメールまでは届きましたが、20MB の添付ファイルつきのメールは届きませんでした。サーバーによって差はあると思いますが、ログが 20MB 以上になると厳しいかもしれません。

本当はログファイルを ZIP 圧縮してから添付したいのですが、XREA/CORESERVER では ZipArchive クラスが使えないので、非圧縮のまま添付しています。

## 終わりに

この記事を書いたとき、メール送信部分は素の PHP で、ログの内容をメール本文として `mb_send_mail()` で送信するコードでした。書き終えた後、「FuelPHP のアドベントカレンダーなのに FuelPHP 成分が少ないぞ」と思い直し、使ったこともない Email パッケージで送信するコードを書き始めたところ、ものの 5 分程度で添付ファイルつきのメールを送信するコードが動いてしまい、ますます FuelPHP が気に入ってしまいました。

**@suno88**

Delphi と PHP をこよなく愛する長野のプログラマー。2013 年に FuelPHP と出会って人生が変わりつつあります。

Twitter: [@suno88](#)

Blog: <http://d.hatena.ne.jp/suno88/>



## FuelPHP と MongoDB と TraceKit で JavaScript のエラー情報を収集してみる

FuelPHP Advent Calendar 2013 18 日目です。@madmamor が担当します。

今日は、FuelPHP と MongoDB と TraceKit を使って、JavaScript のエラー情報を監視、収集する方法を紹介します。

TraceKit は JavaScript のエラーを簡単に監視できる、MIT ライセンスな JavaScript ライブラリです。

- <https://github.com/occ/TraceKit>

また、FuelPHP では MongoDB を簡単に扱えるので、それらを組み合わせることで、JavaScript のエラーを容易に収集できるのでは。と思いつき、試してみました。

記事内のソースは、WTFPL ライセンスとします。

- <http://www.wtfpl.net/txt/copying/>

以下、手順です。

### 1. 下準備

MongoDB と PECL モジュールのインストールを済ませておいて下さい。

MongoDB

- <http://www.mongodb.org/>

PECL :: Package :: mongo

- <http://pecl.php.net/package/mongo>

PHP から MongoDB が使用可能かは、phpinfo() で確認できます。

#### **mongo**

MongoDB Support	enabled
Version	1.4.5
SSL Support	enabled
Streams Support	enabled

FuelPHP のインストールも済ませておいて下さい。トップページが見れる状態です。尚、当記事では v1.7.1 で確認しています。

### 2. TraceKit のインストール

<https://github.com/occ/TraceKit> の tracekit.js をダウンロードして、public/assets/js に置きます。

### 3. FuelPHP の設定

config/db.php に MongoDB 用の設定を追加します。以下、例です。

```
'mongo' => array(
    'tracekit' => array(
        'hostname' => 'localhost',
        'port' => '27017',
        'database' => 'tracekit',
        'username' => 'YOUR_USERNAME',
        'password' => 'YOUR_PASSWORD',
    ),
),
```

### 4. コントローラの作成

app/classes/controller/tracekit.php を作成します。

```
<?php

/**
 * TraceKit が送信するエラー情報を MongoDB に Insert するコントローラ
 *
 * @author Mamoru Otsuka http://madroom-project.blogspot.jp/
 * @copyright 2013 Mamoru Otsuka
 * @license WTFPL License http://www.wtfpl.net/txt/copying/
 */
class Controller_Tracekit extends Controller
{

    /**
     * Ajax で POST されたエラー情報を MongoDB に Insert する
     */
    public function post_errors()
    {
        if (Input::is_ajax() and Security::check_token())
        {
            $input = Input::post();
            unset($input[Config::get('security.csrf_token_key')]);

            $mongodb = Mongo_Db::instance('tracekit');
            $mongodb->insert('errors', $input);
        }
    }
}
```

### 5. view の修正

app/views/welcome/index.php を修正します。

```
<!DOCTYPE html>
<html>
```



- message ... エラーメッセージです。普段、コンソールに出るヤツです。
- url ... エラーが発生した URL です。
- stack.line ... エラーが発生した行、のように見えますが、ズれています。
- stack.func ... JavaScript 側の送信関数名です。
- stack.context ... 発生した行と、前後 5 行ずつのソースです。
- useragent ... ユーザエージェントです。

注意: stack.context に minify された JavaScript が含まれると、相当な量になってしまいます。

## 7. まとめ

FuelPHP と MongoDB と TraceKit を組み合わせると、JavaScript のエラーを簡単に保存できました。ブラウザで発生するエラーも、こういった方法で把握して、改善していきたいものです。

**@madmamor**

Developer & Bassist on TAMACENTER OUTSiDERS.

Twitter: [@madmamor](https://twitter.com/madmamor)

Blog: <http://madroom-project.blogspot.jp/>

# FuelPHP 5 分で API を実装するチュートリアル（スクリーンキャストあり）

---

FuelPHP Advent Calendar 2013 の 19 日目です。

API を作る機会が増えていますよね。スマホアプリから叩いたり、JavaScript のフレームワークから叩いたり。あなたも私も、いきなり誰かに「私、API が叩きたいの♥」と言われるかも知れません。ということで、いつそうなってもいいように、FuelPHP で API を実装する流れをおさらいしておきましょう。

FuelPHP には `Controller_Rest` というものが用意されていて、API を作るのがとても簡単なんですよー、とはよく言われるところですが、実際にどれくらい簡単なのかやってみました。

- Building an api on FuelPHP in 5 minutes <http://www.youtube.com/watch?v=Uin3hh0ldgM>

結果は、FuelPHP のインストールも含めて、4 分 44 秒で実装できました。

以下、詳しく説明します。

## 前提条件

今回は、下記環境での実装例となります。

- PHP 5.4 (Built-in web server、Short array syntax を使っています)
- FuelPHP 1.7.1
- データベースに MySQL を利用
- MySQL の host は localhost、port は 3306、ユーザ、パスワードともに root、データベース名は `fuel_dev` (FuelPHP インストール時の development 環境でのデフォルト設定です)

では、はじめましょう。

## データベースの準備

あらかじめ、`fuel_dev` という名前のデータベースを作っておきます。

```
$ echo 'create database fuel_dev' | mysql -u root -proot
```

## FuelPHP のインストール

oil コマンドでさくっと行きましょう。参考) <http://fuelphp.com/docs/>

```
$ curl get.fuelphp.com/oil | sh
$ cd Sites/
$ oil create api
```

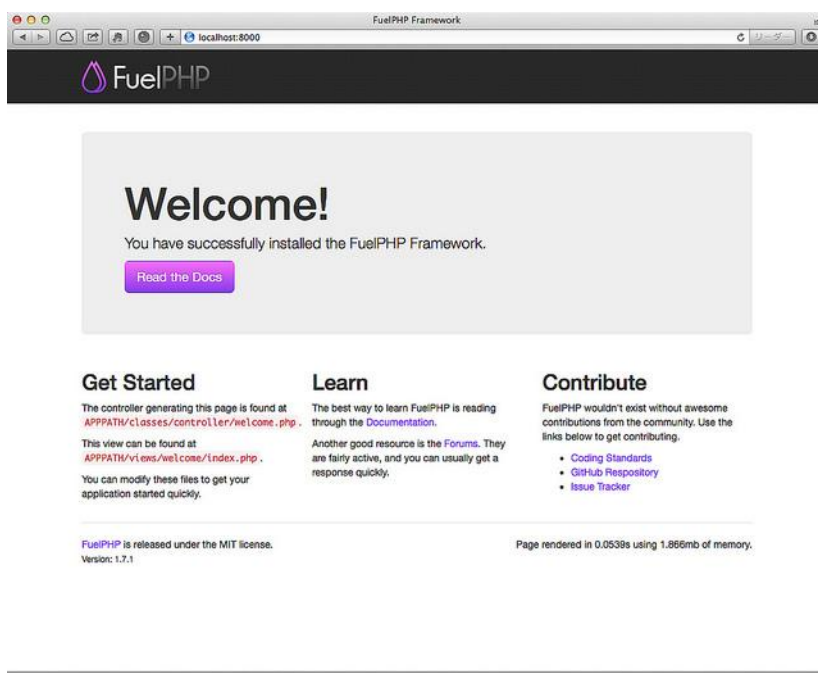
できたら、そのディレクトリへ移動。

```
$ cd api
```

一応念のため、動くか確認しておきましょう。ビルトインサーバ使います。

```
$ php -S localhost:8000 -t public
```

はいこんにちは。



## 何を作るか

今回は、簡単なメモみたいなのを作りましょう。メモを記録したり、取得したりするだけのもの。

- <http://localhost:8000/memo/write>（POST メソッドで）
- <http://localhost:8000/memo/read>（GET メソッドで、JSON を返す）

という2つの URL を用意して、それぞれ、メモの投稿と取得をさせたいと思います。

では、進みましょう。

## oil generate で model 作成

oil generate でモデル作ります。

```
$ php oil generate model memo memo:varchar created_at:timestamp updated_at:timestamp --mysql-timestamp --crud --created-at --updated-at
```

僕の好みで、--crud --mysql-timestamp --created-at --updated-at オプションをつけてます。

```
Creating model: /Users/omoon/Documents/api/fuel/app/classes/model/memo.php
Creating migration: /Users/omoon/Documents/api/fuel/app/migrations/001_create_memos.php
```

とメッセージが表示されて、model と同時にマイグレーションファイルも作成されていることが分かります。  
忘れないうちに、そのまま `oil r migrate` して、テーブルを作成してしまいましょう。

```
$ php oil r migrate
Performed migrations for app:default:
001_create_memos
```

## route.php の編集

続いて、`fuel/app/config/route.php` を下記のように編集します。

```
// fuel/app/config/route.php
return array(
    '_root_' => 'welcome/index', // The default route
    '_404_'  => 'welcome/404',   // The main 404 route

    'memo/read' => 'memo/read',
    'memo/write' => 'memo/write',
);
```

これで、

- `http://localhost:8000/memo/write`
- `http://localhost:8000/memo/read`

この2つの URL へアクセスできるようになります。

## oil generate で controller 作成

上記で設定したルーティングに合わせてコントローラを作りましょう。

FuelPHP では、あらかじめ用意されている **Controller\_Rest** を継承して作ります。ここがキモですね。

ここも `oil generate` でさくっと行きたいところですが、ざっと見た感じでは、`Controller_Rest` 用のオプションなどがあまり用意されていないようでしたので、ある程度まで作ってちょこっと修正する方針で行きます。

```
$ php oil generate controller memo write read --extends=Controller_Rest
```

これで、`fuel/app/classes/controller/memo.php` に下記のようなファイルができあがりますので、

```
class Controller_Memo extends Controller_Rest
{

    public function action_read()
    {
        $data["subnav"] = array('read'=> 'active' );
        $this->template->title = 'Memo » Read';
        $this->template->content = View::forge('memo/read', $data);
    }

    public function action_write()
    {
        $data["subnav"] = array('write'=> 'active' );
        $this->template->title = 'Memo » Write';
        $this->template->content = View::forge('memo/write', $data);
    }

}
```

このように編集します。

```
class Controller_Memo extends Controller_Rest
{

    protected $format = 'json';

    public function get_read()
    {
        $memos = Model_Memo::find_all();
        $this->response($memos);
    }

    public function post_write()
    {
        $memo = Input::post('memo');
        Model_Memo::forge(['memo' => $memo])->save();
    }

}
```

ポイントは、

- `protected $format = 'json';` でデフォルトフォーマットを JSON に
- `public function get_read()` で `read` を GET メソッドで受け付けるように
- `public function post_write()` で `write` を POST メソッドで受け付けるように

各メソッドでは、今回は単純に `Model_Memo::find_all()` で取得、`Model_Memo::forge()->save()` で保存を行っています。

さ、これでできたはずですよ。

## 動作確認

curl コマンドでちゃんと動くか確認します。

POST で書き込み。

```
$ curl http://localhost:8000/memo/write -d "memo=test1"
```



```
$ curl http://localhost:8000/memo/write -d "memo=test2"
$ curl http://localhost:8000/memo/write -d "memo=test3"
```

GET で取得。

```
$ curl http://localhost:8000/memo/read
[{"id": "1", "memo": "test1", "created_at": "2013-12-19 19:48:08", "updated_at": "2013-12-19 19:48:08"}, {"id": "2", "memo": "test2", "created_at": "2013-12-19 19:48:12", "updated_at": "2013-12-19 19:48:12"}, {"id": "3", "memo": "test3", "created_at": "2013-12-19 19:48:15", "updated_at": "2013-12-19 19:48:15"}]
```

無事、JSON が返却されました。

## 最後に

噂通り、FuelPHP での API 実装はとてもお手軽でした。みなさんも一度やってみておくとかの時にアワアワなくていいのではないのでしょうか。

ところで、ゴリゴリ記事を書いて、必死でスクリーンキャストまで撮って、公開ボタンを押そうとしたまさにその時、@kenji\_s さんが、過去に同様のことをなさっていることを発見してしまいました。

- [FuelPHP を使って 5 分で Web API を作成する](#)

決してパクったわけではありません（信じてー）が、もうちょっと過去記事などをしっかり見ておくべきでした。。大変失礼しました。

ただ（言い訳みたいですが）、@kenji\_s さんのバージョンは 1.4 で今回は 1.7.1、また、実装している API や手法も少し異なりますので、実装方法の異なるバージョンとして見ていただければ幸いです。汗。。

では、また。

### omoon

40 代、縄文系。大阪を拠点に PHP などを使って仕事をしています。Kansai PHP Users Group スタッフ。

Twitter: [@omoon](#)

Blog: <http://blog.omoon.org/>

## FuelPHP と Fluentd を連携させてみる

FuelPHP AdventCalendar 2013 の 20 日目です。今回は Fluentd + FuelPHP について書きたいと思います。

まず、「Fluentd って何？」と思った方は <http://fluentd.org/> を参照してください。今回の目標は Fluentd と Observer を連携して、データを送れるようにします。

手順は次のとおりです。

1. ライブラリの取得
2. packages の準備
3. Observer の設定

### 1. ライブラリの取得

Fluentd 用の PHP の Logger は <https://github.com/fluent/fluent-logger-php> にあります。

これを fuel/packages/fluentd/vendor/以下に入れます。

### 2. packages の準備

fuel/packages/fluentd/以下に bootstrap.php を作成します。

fuel/packages/fluentd/bootstrap.php

```
<?php
require_once __DIR__.'./vendor/Fluent/Autoloader.php';

Autoloader::add_classes(array(
    'Fluentd\Observer_Td' => __DIR__.'./classes/observer/td.php',
));
```

この Autoloader に合わせて fuel/packages/fluentd/classes/に必要なファイルを記述します。

fuel/packages/fluentd/classes/observer/td.php

```
<?php
class Observer_Td extends \Orm\Observer {
    public function after_save(\Orm\Model $obj) {
        $save_data = array();
        foreach (array_keys($obj->properties()) as $p) {
            $save_data[$p] = $obj->{$p};
        }

        $instance = new \Fluent\Logger\FluentLogger('localhost', '24800', array(), null);
        $instance->post('tag_name_for_fluentd', $save_data);
    }
}
```

### 3. Observer の設定

Observer を設定して実際に model のデータを Fluentd に渡します。

```
<?php
class Model_Model extends \Orm\Model {
    protected static $_observers = array(
        'Fluentd\Observer_Td' => array(
            'events' => array('after_save')
        )
    );
}
```

これで model で save メソッドが実行された時に save されたデータを Fluentd に渡すことが出来ます。  
これと同様 (特に 1、2 の部分) にして FuelPHP のデバッグログやエラーログを Fluentd に渡すことも出来ます。

#### Altsencturely

東京の PHP 大好きおじさん 最近は Fluentd を使ってログ収集 + 分析基盤ごによごによ

Twitter: [@Altsencturely](#)

Blog: <http://takashi-kun.hatenablog.com/>

# FuelPHP を Rocketeer で自動デプロイしてみる。マイグレーションと PHPUnit も実行してみる。

FuelPHP Advent Calendar 2013 21 日目です。@madmamor が担当します。

今日は、PHP 製デプロイツール「Rocketeer」を使って、FuelPHP をコマンド一つでデプロイしてみます。デプロイする最中に、PHPUnit やマイグレーションも実行してみます。

- Rocketeer 公式ドキュメント：<http://rocketeer.autopergamene.eu/>

ライセンスファイルへのリンクが切れてしまっていますが、MIT ライセンスと書かれています。

- Rocketeer の GitHub：<https://github.com/Anahkiasen/rocketeer>

今回の内容は、MacOS X Mavericks（以下、ローカル）と Vagrant で起動している Ubuntu 13.10（以下、リモート）な環境で確認しています。ローカルは普段通りの開発を行う場所で、そこからコマンドを実行して、リモートにデプロイするイメージです。FuelPHP は 1.7.1 を使いましたが、Composer 対応以降のバージョンであれば、あまり関係は無いはずです。

記事内のソースのライセンスについては、Rocketeer が生成するファイルは Rocketeer のライセンスに準じます。私が作成したファイルは、ソースにも書きますが、WTFPL ライセンスにします。

- <http://www.wtfpl.net/txt/copying/>

## 1. 下準備 (ローカル)

php.ini で以下の設定をします。

```
phar.readonly = Off
```

これをしないと、後述の rocketeer.phar が自身の内部を更新する関係か、以下の警告が大量に出ました。

```
failed to open stream: phar error: write operations disabled by the php.ini setting phar.readonly
```

併せて、FuelPHP プロジェクトを作成して、Git リポジトリへコミットしておいて下さい。この Git リポジトリはリモート側からアクセスできる必要があります。アクセスには、ユーザ名とパスワード、あるいはユーザ名と鍵ファイル (と鍵のパスワード) による認証が使えます。

注意：リモートで鍵の設定時 ~/.ssh/config に以下が無いとエラーになる可能性が有ります。あるいは、一度手動で clone して、ホストの登録を済ませておきましょう。

```
StrictHostKeyChecking no
```

尚、この記事を作成するにあたって作成した FuelPHP プロジェクトのサンプルを公開してあります。

- <https://github.com/mp-php/fuelphp-advent-calendar-2013-rocketeer-sample>

## 2. 下準備 (リモート)

Git、PHP と mcrypt extension、Composer をインストールしておきます。更に、以下の symlink を貼っておきます。このリンク先は、今現在は存在しませんが、それで構いません。

```
$ sudo ln -s /home/vagrant/www/fuel-rocketeer-sample/current/public /var/www/fuel-rocketeer-sample
```

## 3. Rocketeer のインストールと設定ファイルの準備

ローカルで、以下をダウンロードして、プロジェクトルートに配置します。ダウンロード方法は何でも構いません。

- <http://rocketeer.autopergamene.eu/versions/rocketeer.phar>

以下のコマンドでコマンド一覧やヘルプが表示できれば Rocketeer のインストールは完了です。

```
$ php rocketeer.phar # コマンド一覧
$ php rocketeer.phar -h # ヘルプ
```

次に、設定ファイルを準備します。以下のコマンドを実行して下さい。設問は、とりあえず全て未入力で Enter で良いです。

```
$ php rocketeer.phar ignite
```

以下のファイルが生成されたはずです。

- rocketeer/config.php ... 主にリモートの接続情報を設定する
- rocketeer/hooks.php ... 主にデプロイ時等の before/after のタスクを設定する（今回は使いません）
- rocketeer/paths.php ... php や composer 等のコマンドのパスを設定する
- rocketeer/remote.php ... リモートのデプロイ先に関する色々な設定をする
- rocketeer/scm.php ... Git リポジトリの設定をする（SVN も使えるみたいです）
- rocketeer/stages.php ... 同一サーバに複数ステージ（staging や production）がある場合に使う？（今回は使いません）

注意: rocketeer.phar は自身の内部にキャッシュ的に接続設定を保存するようです。以降の設定が正しく反映されない場合、以下のコマンドを実行してみてください。

```
$ php rocketeer.phar flush
```

また、その性質上、rocketeer.phar をパブリックなリポジトリにコミットするのはリスクが有るかもしれません。 .gitignore で除外してしまうのも有りかと思います。

## 4. リモートの接続情報を設定して確認してみる

rocketeer/config.php を修正します。以下は例なので、適切に書き換えて下さい（以降、同様です）。

```
'connections' => array(
    'production' => array(
        'host'      => '192.168.33.10',
        'username'  => 'vagrant',
        'password'  => '',
        'key'       => '/Users/mamor/.vagrant.d/insecure_private_key',
        'keyphrase' => '',
    ),
),
```

SSH のポートを 22 以外にしている場合は "xxx.yyy.com:2222" のように指定してあげれば OK です。  
早速、正しく設定できたか確認してみましょう。

```
$ php rocketeer.phar check

No repository is set for the repository, please provide one :
No username is set for the repository, please provide one :
No password is set for the repository, please provide one :
Checking presence of git
Checking PHP version
Checking presence of Composer
Checking presence of mcrypt extension
Your server is ready to deploy
Execution time: 0.8238s
```

正しく接続できて、git コマンド、PHP バージョン、composer コマンド、mcrypt extension のチェックが行われました。必要な PHP バージョンは、すみません、確認していません。が、Rocketeer の composer.json には "php": ">=5.3.0" と書かれています。ちなみに手元は 5.5 です。

## 5. デプロイの設定をしてデプロイしてみる

rocketeer/remote.php を修正します。

```
$ git diff rocketeer/remote.php
diff --git a/rocketeer/remote.php b/rocketeer/remote.php
index a21279b..51424c4 100644
--- a/rocketeer/remote.php
+++ b/rocketeer/remote.php
@@ -11,12 +11,12 @@
     ),

    // The root directory where your applications will be deployed
-    'root_directory' => '/home/www/',
+    'root_directory' => '/home/vagrant/www/',

    // The name of the application to deploy
    // This will create a folder of the same name in the root directory
    // configured above, so be careful about the characters used
```

```

-     'application_name' => '',
+     'application_name' => 'fuel-rocketeer-sample',

    // The number of releases to keep at all times
    'keep_releases' => 4,
@@ -25,23 +25,24 @@
    // Use this to list folders that need to keep their state, like
    // user uploaded data, file-based databases, etc.
    'shared' => array(
-         '{path.storage}/logs',
-         '{path.storage}/sessions',
+         'fuel/app/cache',
+         'fuel/app/logs',
+         'fuel/app/tmp',
    ),

    'permissions' => array(

        // The permissions to CHMOD folders to
        // Change to null to leave the folders untouched
-         'permissions' => 755,
+         'permissions' => 777,

        // The folders and files to set as web writable
        // You can pass paths in brackets, so {path.public} will return
        // the correct path to the public folder
        'files' => array(
-             'app/database/production.sqlite',
-             '{path.storage}',
-             '{path.public}',
+             'fuel/app/cache',
+             'fuel/app/logs',
+             'fuel/app/tmp',
        ),

        // The web server user and group to CHOWN folders to

```

"root\_directory"の下に"application\_name"な名前のディレクトリが作成され、その中にデプロイされます。この例だと"/home/vagrant/www/fuel-rocketeer-sample"になりますね。

"shared"では、デプロイをまたいで共有したいディレクトリやファイルを設定します。大抵の場合、ログディレクトリやキャッシュディレクトリ等、.gitignore に書かれているものになると思います。裏を返せば、例えばデプロイ毎にキャッシュをクリアしたければ、あえて共有しなれば OK です。尚、共有は symlink によって実現されます。

注意：ディレクトリを共有する場合、ディレクトリそのものがリポジトリに含まれている必要があります。.gitkeep や、以下のような.gitignore ファイルをそのディレクトリに入れるなどしておいて下さい。

```

*
!.gitignore

```

"permissions"は、指定したディレクトリやファイルを、指定したパーミッションに変更します。今回の例では 777 を指定していますが、適切な値を設定するようにお願いします。

次に rocketeer/scm.php を修正します。

```

'repository' => 'https://github.com/mp-php/fuelphp-advent-calendar-2013-rocketeer-sample.git',

```

"repository"に、Git のリポジトリ URL を設定します。今回の例は GitHub 上のリポジトリなので、username と password

は空のまま構いません。また、ファイル内のコメントにも書かれているように、既に鍵認証の設定がされている場合も、空で構いません。

以上で基本的な設定が済んだので、お待ちかねのデプロイを実行してみましょう。

```
$ php rocketeer.phar deploy

No username is set for the repository, please provide one :
No password is set for the repository, please provide one :
Cloning repository in "/home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831"
Initializing submodules if any
Installing Composer dependencies
Setting permissions for /home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831/fuel/app/cache
Setting permissions for /home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831/fuel/app/logs
Setting permissions for /home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831/fuel/app/tmp
Sharing file /home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831/fuel/app/cache
Sharing file /home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831/fuel/app/logs
Sharing file /home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831/fuel/app/tmp
Successfully deployed release 20131220204831
No releases to prune from the server
Execution time: 61.1617s
```

Git リポジトリの clone (submodules があればそれも) が行われ、composer install が行われ、パーミッション変更が行われ、共有が行われました。

ブラウザから [http://\[ドメイン\]/fuel-rocketeer-sample/](http://[ドメイン]/fuel-rocketeer-sample/) にアクセスして、おなじみのトップ画面が表示されればデプロイ成功です。

ざっとディレクトリ構造を見てみましょう。

```
$ ll /home/vagrant/www/fuel-rocketeer-sample/
total 20
drwxrwxr-x 4 vagrant vagrant 4096 Dec 20 11:49 ./
drwxrwxr-x 3 vagrant vagrant 4096 Dec 20 11:48 ../
lrwxrwxrwx 1 vagrant vagrant   63 Dec 20 11:49 current -> /home/vagrant/www/fuel-rocketeer-sample/releases/20131220204831/
drwxrwxr-x 3 vagrant vagrant 4096 Dec 20 11:48 releases/
drwxrwxr-x 3 vagrant vagrant 4096 Dec 20 11:49 shared/
```

"current"ディレクトリが、先ほどデプロイしたディレクトリへ symlink されています。

```
$ ll /home/vagrant/www/fuel-rocketeer-sample/current/fuel/app/
total 56
drwxrwxr-x 11 vagrant vagrant 4096 Dec 20 11:49 ./
drwxrwxr-x  6 vagrant vagrant 4096 Dec 20 11:49 ../
-rw-rw-r--  1 vagrant vagrant  718 Dec 20 11:48 bootstrap.php
lrwxrwxrwx  1 vagrant vagrant   61 Dec 20 11:49 cache -> /home/vagrant/www/fuel-rocketeer-sample/shared/fuel/app/cache
drwxrwxr-x  5 vagrant vagrant 4096 Dec 20 11:48 classes/
drwxrwxr-x  6 vagrant vagrant 4096 Dec 20 11:48 config/
drwxrwxr-x  3 vagrant vagrant 4096 Dec 20 11:48 lang/
lrwxrwxrwx  1 vagrant vagrant   60 Dec 20 11:49 logs -> /home/vagrant/www/fuel-rocketeer-sample/shared/fuel/app/logs
drwxrwxr-x  2 vagrant vagrant 4096 Dec 20 11:48 migrations/
drwxrwxr-x  2 vagrant vagrant 4096 Dec 20 11:48 modules/
drwxrwxr-x  2 vagrant vagrant 4096 Dec 20 11:48 tasks/
drwxrwxr-x  5 vagrant vagrant 4096 Dec 20 11:48 tests/
lrwxrwxrwx  1 vagrant vagrant   59 Dec 20 11:49 tmp -> /home/vagrant/www/fuel-rocketeer-sample/shared/fuel/app/tmp
drwxrwxr-x  2 vagrant vagrant 4096 Dec 20 11:48 vendor/
drwxrwxr-x  3 vagrant vagrant 4096 Dec 20 11:48 views/
```

共有設定したディレクトリが、"shared"ディレクトリ下に symlink されています。パーミッションも、設定したとおりに変更



されています。

以上が、Rocketeer による基本的なデプロイ方法です。

## 6. マイグレーションも実行してみる

だいぶ長くなってきましたが、続いてマイグレーションの実行です。簡単なマイグレーションファイルを作成してコミットしておきます。

```
$ php oil generate migration create_users name:text email:string password:string
```

リモート側で DB や DB ユーザの作成、それに対する FuelPHP の config の db.php の設定も済ませておいて下さい。

次に、2 ファイルを新規作成します。

まず、rocketeer/tasks/Migrate.php を新規作成します。今回はサンプルなので、名前空間はつけていません。尚、"Rocketeer\Traits\Task"を継承しますが、このクラスは abstract class であってトレイトではないようです。

```
<?php

/**
 * Migrate class
 *
 * @author Mamoru Otsuka http://madroom-project.blogspot.jp/
 * @copyright 2013 Mamoru Otsuka
 * @license WTFPL License http://www.wtfpl.net/txt/copying/
 */
class Migrate extends Rocketeer\Traits\Task
{
    /**
     * @inheritDoc
     */
    protected $description = 'Migrates the database';

    /**
     * @inheritDoc
     */
    public function execute()
    {
        // 実行時にメッセージとして表示されます
        $this->command->info($this->description);

        // current ディレクトリ内でコマンドを実行します
        $output = $this->runForCurrentRelease('php oil r migrate');

        // 第一引数は失敗時のメッセージです
        // 第二引数は失敗時の詳細です
        // 第三引数は成功時のメッセージです
        return $this->checkStatus('Migrate failed', $output, 'Migrate successfully');
    }
}
```

rocketeer/tasks.php を新規作成します。

```
<?php
/**
 * rocketeer/tasks.php
 *
 * @author Mamoru Otsuka http://madroom-project.blogspot.jp/
```

```
* @copyright 2013 Mamoru Otsuka
* @license   WTFPL License http://www.wtfpl.net/txt/copying/
*/

// Migrate クラスをカスタムタスクとして登録します
// $ php rocketeer.phar migrate で個別に実行できます
require_once __DIR__ . '/tasks/Migrate.php';
Rocketeer\Facades\Rocketeer::add('Migrate');

// deploy 後に自動実行されます
// 自動実行したくない場合は書かないで下さい
Rocketeer\Facades\Rocketeer::after('deploy', 'Migrate');
```

Git リポジトリにコミット（Push）したら、デプロイを実行してみます。

```
$ php rocketeer.phar deploy
...略...
Migrates the database
Migrate successfully
Removing 1 release from the server
Execution time: 34.8705s
```

マイグレーションも実行できました。以下のコマンドでマイグレーションのみを個別に実行することもできます。

```
$ php rocketeer.phar migrate
```

注意：deploy の after タスクは、既に symlink が貼替えられていることに注意して下さい。尚、マイグレーションを行う場合は、別途、何らかの方法でメンテナンスモードに切り替えるタスクを作成する必要があるかと思います（もちろん、手作業でも良いですが）。また、後述の PHPUnit 失敗時の挙動も気になるところで、マイグレーションを自動化するのはそれなりのリスクが伴いそうです。が、今回はとりあえず自動化して進めます。

after（や before）タスクにはクラスの他に、インラインによるコマンド設定や、クロージャの設定もできるみたいです（まだやったことはありません）。クラスにすると"\$this->runForCurrentRelease('コマンド')のように、便利なメソッドでパス周りの調整が簡単になるので、迷ったらクラスで良いのかなと思います。クラスだと、前述のように個別で実行もできますね。

## 7. PHPUnit も実行してみる

そろそろ最後です。ソースを clone して、PHPUnit を実行して、全てのテストが成功したらデプロイ続行、1 つでも失敗したらデプロイ中止（symlink の貼替えを行わない）できたら良いですね。

composer.json に以下を追記します。

```
$ git diff composer.json
diff --git a/composer.json b/composer.json
index e1b21ea..5ef630e 100644
--- a/composer.json
+++ b/composer.json
@@ -20,6 +20,9 @@
     "monolog/monolog": "1.5.*",
     "fuelphp/upload": "2.0.1"
 },
+ "require-dev": {
```

```
+     "phpunit/phpunit": "3.*"
+ },
+ "suggest": {
+     "mustache/mustache": "Allow Mustache templating with the Parser package",
+     "smarty/smarty": "Allow Smarty templating with the Parser package",
```

プロジェクト直下に phpunit.xml を用意します。fuel/core/phpunit.xml をコピーして、パス周りを整えただけです。

```
<?xml version="1.0" encoding="UTF-8"?>

<phpunit colors="true" stopOnFailure="false" bootstrap="fuel/core/bootstrap_phpunit.php">
  <php>
    <server name="doc_root" value="."/>
    <server name="app_path" value="fuel/app"/>
    <server name="core_path" value="fuel/core"/>
    <server name="package_path" value="fuel/packages"/>
    <server name="vendor_path" value="fuel/vendor"/>
    <server name="FUEL_ENV" value="test"/>
  </php>
  <testsuites>
    <testsuite name="core">
      <directory suffix=".php">fuel/core/tests</directory>
    </testsuite>
    <testsuite name="packages">
      <directory suffix=".php">fuel/packages/*/tests</directory>
    </testsuite>
    <testsuite name="app">
      <directory suffix=".php">fuel/app/tests</directory>
    </testsuite>
  </testsuites>
</phpunit>
```

rocketeer/paths.php を修正します。

```
$ git diff rocketeer/paths.php
diff --git a/rocketeer/paths.php b/rocketeer/paths.php
index f366b41..2b4d6d4 100644
--- a/rocketeer/paths.php
+++ b/rocketeer/paths.php
@@ -19,4 +19,6 @@
     // Path to the Artisan CLI
     'artisan' => '',

+    // Path to PHPUnit
+    'phpunit' => 'fuel/vendor/bin/phpunit',
+  );
```

Rocketeer は/usr/local/bin 等のグローバルな場所の phpunit、あるいはプロジェクト直下の vendor/bin/phpunit は勝手に見つけてくれます。FuelPHP の場合は fuel/vendor/bin/phpunit になるので、この設定が必要です（この設定がない場合は、対話式でパスの入力が可能ですが）。

-t オプションをつけてデプロイを実行してみます。

```
$ php rocketeer.phar deploy -t
...略...
Running tests...
Tests passed successfully
```

```
...略...
Execution time: 194.1824s
```

テストが行われました。単体でも実行できます。

```
$ php rocketeer.phar test
...略...
Testing the application
Running tests...
...略...
[vagrant@192.168.33.11] (production) Time: 9.23 seconds, Memory: 23.25Mb
[vagrant@192.168.33.11] (production) OK (361 tests, 413 assertions)
Tests passed successfully
Execution time: 9.6588s
```

注意：テストの実行を after タスクで行うこともできますが、その時には既に symlink が貼替わってしまっています。-t オプションを使うようにしましょう。

最後に、必ず失敗するテストを作成して、どうなるかも確認してみます（ソースは割愛します）。

```
$ php rocketeer.phar deploy -t

No username is set for the repository, please provide one :
No password is set for the repository, please provide one :
Cloning repository in "/home/vagrant/www/fuel-rocketeer-sample/releases/20131220215555"
Initializing submodules if any
Installing Composer dependencies
Running tests...
Tests failed
PHPUnit 3.8-g5fb30aa by Sebastian Bergmann.

Configuration read from /home/vagrant/www/fuel-rocketeer-sample/releases/20131220215555/phpunit.xml

The Xdebug extension is not loaded. No code coverage will be generated.

..... 63 / 362 ( 17%)
..... 126 / 362 ( 34%)
..... 189 / 362 ( 52%)
..... 252 / 362 ( 69%)
..... 315 / 362 ( 87%)
.....F

Time: 8.6 seconds, Memory: 23.25Mb

There was 1 failure:

1) Test_Example::test_fail

/home/vagrant/www/fuel-rocketeer-sample/releases/20131220215555/fuel/app/tests/example.php:14

FAILURES!
Tests: 362, Assertions: 413, Failures: 1.
Tests failed
Rolling back to release 20131220215158
Migrates the database
Migrate successfully
Execution time: 196.3031s
```

デプロイが中断されました。symlink は以前のままです。中断されたデプロイのディレクトリはゴミとして残りますが、

rocketeer/remote.php の "keep\_releases" により、そのうち掃除されると思いますので、あまり気にしなくても良いかなと思います。マイグレーションが実行されてしまうのは想定外だったので、この点は今後の課題にします…

## 8. まとめ

（全ての機能を把握できているわけではありませんし、公式ドキュメントに記載されているプラグイン機能も気になるところですが）Rocketeer を使って、FuelPHP をコマンド 1 つで、PHPUnit やマイグレーションの実行を含めてデプロイできました。

デプロイツールは Ruby 製の Capistrano が有名ですが、Rocketeer は PHP 製ということもあり、Composer や PHPUnit の扱いを標準でサポートしてくれていて助かります。今日現在、Rocketeer の使い方に関する日本語の情報はかなり少ない（というか無いかもしれません。あったらすみません）ので、今後、盛り上がってくれると良いな—と思います。

Chef と Vagrant のおかげで、こういったサーバが絡む実験もやりやすくなったので、ぜひ試してみてください。

以上です。お疲れ様でした。

**@madmamor**

Developer & Bassist on TAMACENTER OUTSiDERS.

Twitter: [@madmamor](https://twitter.com/madmamor)

Blog: <http://madroom-project.blogspot.jp/>

# FuelPHP が OAuth 対応になったので facebook ログインを試してみる

FuelPHP Advent Calendar2013 参加記事です。

昨年のアドベントカレンダーでは主に Phil Sturgeon 氏がメンテされていた NinjAuth という OAuth 認証パッケージを使用して簡単にログイン認証が行えるパッケージを作成したのですが、FuelPHP 1.6.1 から NinjAuth に代わり OAuth に対応したパッケージが標準で入るようになったので、今年はそちらでログイン連携を行う最小限の方法を紹介します。ざっくりとした作業量比較としては、インストール作業自体は NinjAuth より楽に行えますが、コントローラ、ビューのサンプルなどは付属していないので、そのあたりはまるっと書く必要があるので手間ある感じです。

とりあえず動かすところまでなので、バリデーションの処理やコントローラでのハンドリングを実際のアプリケーションで作りこむと良い感じになると思われます。

環境：FuelPHP 1.7.1、Composer はインストール済み、DB はセットアップ済みの想定

それでは各ステップごとにいってみましょう。

## 各 config の設定

fuel/app/config/config.php に always\_load に auth と orm を追加します。

fuel/app/config/config.php

```
'always_load' => array(

    /**
     * These packages are loaded on Fuel's startup.
     * You can specify them in the following manner:
     *
     * array('auth'); // This will assume the packages are in PKGPATH
     *
     * // Use this format to specify the path to the package explicitly
     * array(
     *     array('auth' => PKGPATH.'auth/')
     * );
     */
    'packages' => array(
        'auth',
        'orm',
    ),
),
```

packages/auth/config から opauth.php をコピーしてきて app/config 以下に配置し、opauth.php の Strategy を追加します。今回は Facebook のみ。

fuel/app/config/opauth.php

```
'Strategy' => [  
    'Facebook' => [  
        'app_id' => 'xxxxx',  
        'app_secret' => 'xxxxxx',  
    ],  
]
```

## composer で必要なパッケージをインストール

composer.json の require に以下を追加します。

composer.json

```
"oauth/oauth": "0.4.*",  
"oauth/facebook": "dev-master",
```

プロジェクトのルートディレクトリで

```
composer update
```

を実行すると必要な oauth 本体と Facebook ストラテジがインストールされます。

## マイグレーションを実行して必要なテーブルを作成

```
php oil r migrate --packages=auth
```

を実行すると、下記のテーブルが作成されます。

```
+-----+  
| migration      |  
| users          |  
| users_clients  |  
| users_providers|  
| users_scopes   |  
| users_sessions |  
| users_sessionscopes |  
+-----+
```

## コントローラの作成

auth 用のコントローラを作成します。コントローラ名に特に縛りはありませんが、今回は Controller\_Auth という名前で作成します。

FuelPHP の公式ドキュメントにある [Using Auth in your application](#) のサンプルをコピペしつつ、動くように調整してみます。

fuel/app/classes/controller/auth.php

```
<?php
/**
 * 認証コントローラーサンプル
 *
 * @author      egmc
 * @copyright   2013 egmc
 * @license     MIT License
 *
 * The Original Code (Part of the FuelPHP Documentation) distributed under the MIT License
 * @copyright   2010 - 2013 Fuel Development Team
 * @link        http://fuelphp.jp/docs/1.7/packages/auth/examples/oauth.html
 * @link        http://fuelphp.com
 */

use Fuel\Core\Controller;
use Fuel\Core\Log;

class Controller_Auth extends Controller {

    public function action_oauth($provider = null)
    {
        // bail out if we don't have an OAuth provider to call
        if ($provider === null)
        {
            Log::error(__('login-no-provider-specified'));
            \Response::redirect_back();
        }

        // load OAuth, it will load the provider strategy and redirect to the provider
        \Auth_Oauth::forge();
    }

    public function action_logout()
    {
        // remove the remember-me cookie, we logged-out on purpose
        \Auth::dont_remember_me();

        // logout
        \Auth::logout();

        // and go back to where you came from (or the application
        // homepage if no previous page can be determined)
        \Response::redirect_back();
    }

    public function action_callback()
    {
        // OAuth can throw all kinds of nasty bits, so be prepared
        try
        {
            // get the OAuth object
            $oauth = \Auth_Oauth::forge(false);

            // and process the callback
            $status = $oauth->login_or_register();

            // fetch the provider name from the oauth response so we can display a message
            $provider = $oauth->get('auth.provider', '?');

            // deal with the result of the callback process
            switch ($status)
            {
                // a local user was logged-in, the provider has been linked to this user
                case 'linked':
                    // inform the user the link was successfully made
                    // and set the redirect url for this status

```



```

        $url = '/';
        break;

        // the provider was known and linked, the linked account as logged-in
    case 'logged_in':
        // inform the user the login using the provider was succesful
        // and set the redirect url for this status

        $url = '/';
        break;

        // we don't know this provider login, ask the user to create a local account first
    case 'register':
        // inform the user the login using the provider was succesful,
        // but we need a local account to continue
        // and set the redirect url for this status
        $url = 'auth/register';
        break;

        // we didn't know this provider login, but enough info was returned to auto-register the user
    case 'registered':
        // inform the user the login using the provider was succesful, and we created a local account
        // and set the redirect url for this status
        $url = '/';
        break;

    default:
        throw new \FuelException(
            'Auth_Opauth::login_or_register() has come up with a result '
            .'that we dont know how to handle.'
        );
    }

    // redirect to the url set
    \Response::redirect($url);
}

// deal with Opauth exceptions
catch (\OpauthException $e)
{
    Log::error($e->getMessage());
    \Response::redirect_back();
}

// catch a user cancelling the authentication attempt (some providers allow that)
catch (\OpauthCancelException $e)
{
    // you should probably do something a bit more clean here...
    exit('It looks like you canceled your authorisation.'
        .\Html::anchor('users/oath/'.$provider, 'Click here')
        .' to try again.');
```

```

}

public function action_register()
{
    // create the registration fieldset
    $form = \Fieldset::forge('registerform');

    // add a csrf token to prevent CSRF attacks
    $form->form()->add_csrf();

    // and populate the form with the model properties
    $form->add_model('Model\\Auth_User');

    // add the fullname field, it's a profile property, not a user property
    $form->add_after(

```

```

        'fullname',
        __('login.form.fullname'),
        array(),
        array(),
        'username'
    )->add_rule('required');

// add a password confirmation field
$form->add_after(
    'confirm',
    __('login.form.confirm'),
    array('type' => 'password'), array(), 'password'
)->add_rule('required');

// make sure the password is required
$form->field('password')->add_rule('required');

// and new users are not allowed to select the group they're in (duh!)
$form->disable('group_id');

// since it's not on the form, make sure validation doesn't trip on its absence
$form->field('group_id')->delete_rule('required')->delete_rule('is_numeric');

// fetch the oauth provider from the session (if present)
$provider = \Session::get('auth-strategy.authentication.provider', false);

// if we have provider information, create the login fieldset too
if ($provider)
{
    // disable the username, it was passed to us by the OAuth strategy
    $form->field('username')->set_attribute('readonly', true);

    // create an additional login form so we can link providers to existing accounts
    $login = \Fieldset::forge('loginform');
    $login->form()->add_csrf();
    $login->add_model('Model\Auth_User');

    // we only need username and password
    $login->disable('group_id')->disable('email');

    // since they're not on the form, make sure validation doesn't trip on their absence
    $login->field('group_id')->delete_rule('required')->delete_rule('is_numeric');
    $login->field('email')->delete_rule('required')->delete_rule('valid_email');
}

// was the registration form posted?
if (\Input::method() == 'POST')
{
    // was the login form posted?
    if ($provider and \Input::post('login'))
    {
        // check the credentials.
        if (\Auth::instance()->login(\Input::param('username'), \Input::param('password')))
        {
            // get the current logged-in user's id
            list($userid) = \Auth::instance()->get_user_id();

            // so we can link it to the provider manually
            $this->link_provider($userid);

            // logged in, go back where we came from,
            // or the the user dashboard if we don't know
            \Response::redirect_back('dashboard');
        }
        else
        {
            // login failed, show an error message
            Log::error(__('login.failure'));
        }
    }
}

```

```

    }

    // was the registration form posted?
    elseif (\Input::post('register'))
    {
        // validate the input
        $form->validation()->run();

        // if validated, create the user
        if ( ! $form->validation()->error() )
        {
            try
            {
                // call Auth to create this user
                $created = \Auth::create_user(
                    $form->validated('username'),
                    $form->validated('password'),
                    $form->validated('email'),
                    \Config::get('application.user.default_group', 1),
                    array(
                        'fullname' => $form->validated('fullname'),
                    )
                );

                // if a user was created successfully
                if ($created)
                {
                    // inform the user

                    // link new user
                    $this->link_provider($created);

                    // and go back to the previous page, or show the
                    // application dashboard if we don't have any
                    \Response::redirect_back('/');
                }
                else
                {
                    // oops, creating a new user failed?
                    Log::error(__('login.account-creation-failed'));
                }
            }

            // catch exceptions from the create_user() call
            catch (\SimpleUserUpdateException $e)
            {
                // duplicate email address
                if ($e->getCode() == 2)
                {
                    Log::error(__('login.email-already-exists'));
                }

                // duplicate username
                elseif ($e->getCode() == 3)
                {
                    Log::error(__('login.username-already-exists'));
                }

                // this can't happen, but you'll never know...
                else
                {
                    Log::error($e->getMessage());
                }
            }
        }
    }

    // validation failed, repopulate the form from the posted data
    $form->repopulate();

```

```

    }
    else
    {
        // get the auth-strategy data from the session (created by the callback)
        $user_hash = \Session::get('auth-strategy.user', array());

        // populate the registration form with the data from the provider callback
        $form->populate(array(
            'username' => \Arr::get($user_hash, 'nickname'),
            'fullname' => \Arr::get($user_hash, 'name'),
            'email' => \Arr::get($user_hash, 'email'),
        ));
    }
    $form->add('register', '', array('type'=>'hidden', 'value' => '1'));
    $form->add('submit', '', array('type'=>'submit', 'value' => 'submit'));

    // pass the fieldset to the form, and display the new user registration view
    return \View::forge('login/registration')->set('form', $form->build(), false)
        ->set('login', isset($login) ? $login : null, false);
}

protected function link_provider($userid)
{
    // do we have an auth strategy to match?
    if ($authentication = \Session::get('auth-strategy.authentication', array()))
    {
        // don't forget to pass false, we need an object instance, not a strategy call
        $opauth = \Auth_Opauth::forge(false);

        // call Opauth to link the provider login with the local user
        $insert_id = $opauth->link_provider(array(
            'parent_id' => $userid,
            'provider' => $authentication['provider'],
            'uid' => $authentication['uid'],
            'access_token' => $authentication['access_token'],
            'secret' => $authentication['secret'],
            'refresh_token' => $authentication['refresh_token'],
            'expires' => $authentication['expires'],
            'created_at' => time(),
        ));
    }
}
}
}

```

action\_oauth が起点となるアクションとなり、ここから各サービスへリダイレクトされます。

コールバックは config で特に指定していない場合、同コントローラの action\_callback アクションに帰ってきますので、action\_callback 内で処理を判定します。

未登録の場合は auth/register に飛ばしてユーザー登録を行わせ、登録が完了した時点で link\_provider でユーザーの紐付けを行うようになっています。

## ユーザー登録用ビューの作成

今回は fieldset を使っているので、単純に echo してとりあえずフォームを出してみましょう。

fuel/app/views/login/registration.php

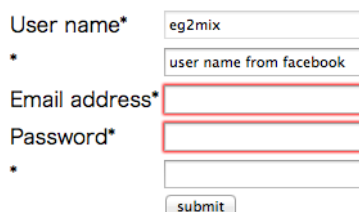
```

<?php
echo $form;

```

## 確認してみる

コントローラ、ビューまで作成して/auth/oauth/facebook にアクセスすると facebook 認証が行われ、未登録であれば register に飛ばされます。

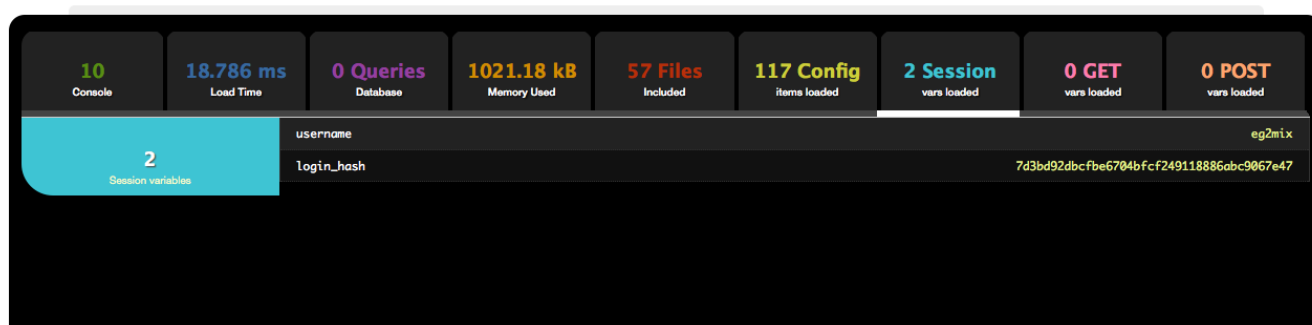


The registration form contains the following fields and elements:

- User name\***: Input field with the value "eg2mix".
- \***: Input field with the value "user name from facebook".
- Email address\***: Input field.
- Password\***: Input field.
- \***: Input field.
- submit**: Submit button.

図 22.1 登録フォーム

登録を完了し、再度同じ URL からアクセスするとログインとなり、セッションにログイン情報が格納されます。



10 Console	18.786 ms Load Time	0 Queries Database	1021.18 kB Memory Used	57 Files Included	117 Config Items loaded	2 Session vars loaded	0 GET vars loaded	0 POST vars loaded
2 Session variables		username eg2mix						
		login_hash 7d3bd92dbcfbef04bfcf249118886abc9067e47						

図 22.2 セッション情報

なお、各 OAuth プロバイダから取得したトークンなどの情報は users\_providers テーブルに格納されていますので、こちらを利用して facebook への登録や、twitter への投稿などを行う処理を個別に実装出来ます。

### @egmc

主に PHP エンジニア。CAMPFIRE、他個人的にもクロアプリなど。

Twitter: [@egmc](#)

Blog: <http://dasalog.eg2mix.com/>

# Heroku (PaaS) で FuelPHP 環境 (PHP 5.3 + MySQL + Apache) を構築する

## Heroku へ FuelPHP 環境を構築する手順をメモ

FuelPHP Advent Calendar 2013 の 23 日目です。

以前まではサービスをリリースするときにはレンタルサーバを借りてサービスをデプロイすることが一般的でしたが、最近は PaaS と呼ばれるアプリケーションの動作環境をプラットフォーム上で一式提供されている形態を使う事が増えてきています。有名どころでは Amazon の AWS や Microsoft の Windows Azure などがあります。

PaaS サービスのそれぞれの違いはスペック・料金・機能などであり、レンタルサーバと比較した場合の PaaS のメリットはスケールアップ、アウトが簡単に行えることです。

サービスのユーザ数が大幅にふえてサーバに負荷がかかるようになった場合や、予想してた以上にユーザ数が伸びなかった場合などに役立ちます。

個人的には Heroku は Rails のアプリをテスト的に公開したくなったときなどに使ったりしています。もともと Heroku 自体は Ruby の環境用としてスタートしていて現在は Java、node.js、Ruby、Pythonなどをサポートしています。

現在 Heroku では PHP は非サポートとなっていますが PHP も動作します。

Buildpack は <https://github.com/winglian/heroku-buildpack-php> を使います。

### ■ Note :

下記サイトに記載されている Buildpack を使用してサーバは nginx を使用するつもりでしたが、nginx の config ファイルの設定を FuelPHP のプロジェクトに合わせると、なぜかうまくページが表示されず原因不明だったので今回は見送ることにします。すみません。

<http://tkyk.name/blog/2012/11/28/php-on-heroku/>

- Heroku コマンド参考ページ : <http://d.hatena.ne.jp/xyk/20101102>

## 1. Heroku アカウントを取得

公式サイトより「login」押下して「signup」よりアカウントを取得します。

- Heroku 公式サイト : <https://www.heroku.com/>

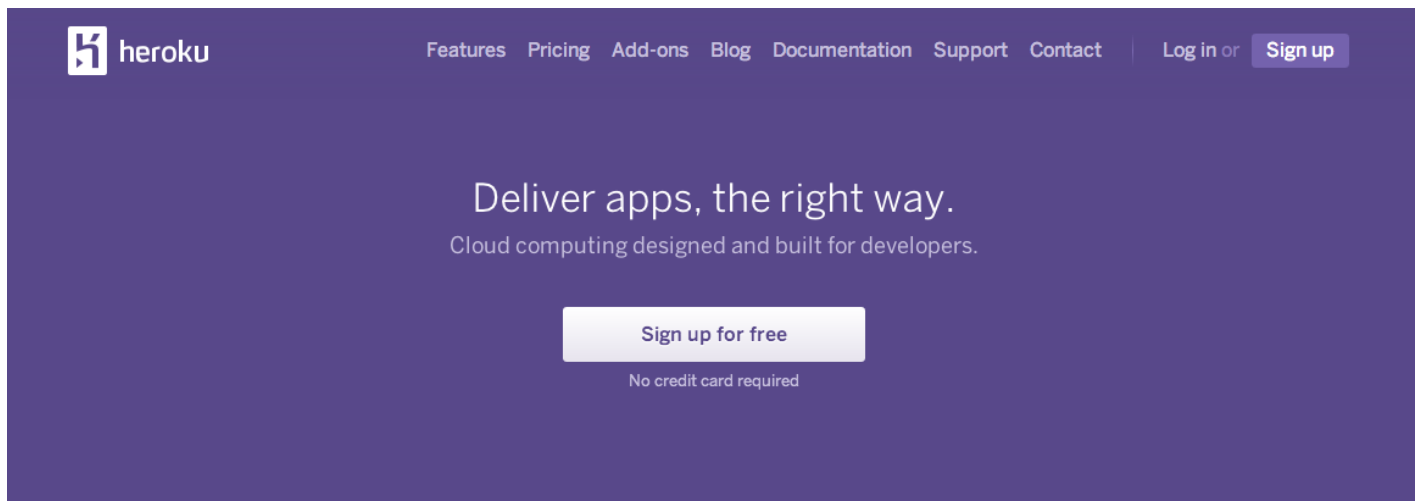


図 23.1 Heroku

## 2. Heroku Toolbelt (ターミナルから Heroku を操作するツール) をインストール

下記サイトより Heroku Toolbelt を環境に合わせてインストールします。

- <https://toolbelt.heroku.com/>

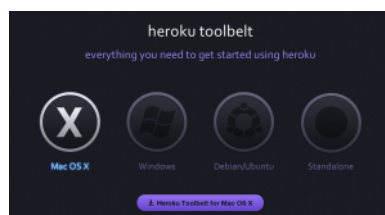


図 23.2 Heroku Toolbelt

## 3. SSH 公開鍵の設定

### (1) ターミナルを立ち上げて login コマンドを実行

```
$ heroku login
Enter your Heroku credentials.
Email: [1 で作成したアカウントを入力]
Password (typing will be hidden): [1 で作成したアカウントのパスワードを入力]
Authentication successful.
```

「Authentication successful.」でアカウント認証成功、「Authentication failed.」は失敗です。

### (2) 公開鍵をジェネレート

```
Could not find an existing public key.
Would you like to generate one? [Yn] [Y を指定してエンター]
Generating new SSH public key.
```

```
Uploading SSH public key /Users/(PC ユーザー名)/.ssh/id_rsa.pub
Authentication successful.
```

「Authentication successful.」で成功です。

初期ログイン時に公開鍵を生成しなかった場合は手動で「ssh-keygen」で作成し Heroku に公開鍵を設定する必要があります。

ssh-keygen については「ssh-keygen コマンドで秘密鍵・公開鍵生成 (<http://to-developer.com/blog/?p=563>)」を参照してください。

### (3) Heroku に公開鍵を設定

```
$ heroku keys:add
```

Heroku の GUI 画面から設定も可能です。

## 4. FuelPHP プロジェクトを生成

### (1) アプリケーションを作成

```
$ oil create [アプリ名を入力]
```

### (2) index.php の作成

Heroku は root ディレクトリに index.php がないと動作しないため、今のところ空の index.php ファイルを生成します。

```
$ touch index.php
```

### (3) .htaccess を作成 (FuelPHP のディレクトリ構成に合わせてリダイレクト処理を入れる)

FuelPHP ディレクトリ構成の public/以下に index.php のアクセスをリダイレクトします。

参考サイト：<http://blog.livedoor.jp/erscape/archives/6937126.html>

```
$ vim .htaccess
```

.htaccess の設定情報

```
RewriteEngine on
RewriteBase /
RewriteRule ^(.+)-info\.php$ $1-info.php [L]
RewriteCond %{SCRIPT_FILENAME} !~/app/www/public/
RewriteRule ^(.*)$ public/$1 [L]
```



#### (4) 不要ファイル削除 (サブモジュールなどは add できないので削除)

```
$ rm -rf .git .gitmodules
$ rm *.md
$ rm -rf docs
// 下記も不要なため削除
$ rm -fr fuel/core/
$ rm -fr fuel/packages/auth/
$ rm -fr fuel/packages/email/
$ rm -fr fuel/packages/oil/
$ rm -fr fuel/packages/orm/
$ rm -fr fuel/packages/parser/
```

#### サブモジュールを add する方法 (git submodule add コマンド)

(例) opauth サブモジュールの場合

```
$ git submodule add git://github.com/andreoav/fuel-opauth.git fuel/packages/opauth
```

### 5. ローカルリポジトリにコミット

```
$ cd [アプリ名を入力]
$ git init
$ git commit -am "initial commit"
```

### 6. buildpack を Heroku へインストール

```
$ heroku create --buildpack https://github.com/winglian/heroku-buildpack-php [アプリ名を入力]
```

アプリ名はここで入れなくてもデフォルトの名前が付けられます。GUI 画面などから確認・変更が可能です。

### 7. Heroku のリポジトリへ反映

```
$ git push heroku master
```

アプリが複数存在する場合、Heroku のリモートリポジトリが違い push できない場合があるので都度確認が必要です。

#### 1. リモートリポジトリ heroku の設定

```
$ git remote add heroku [リモートリポジトリ]
```

## 2. リモートリポジトリ確認

```
$ git remote show
```

## 3. リモートリポジトリ削除

```
$ git remote rm [リモートリポジトリ]
```

## 8. 動作確認

```
$ heroku open
```

Fuel の Welcome 画面がでたら成功！

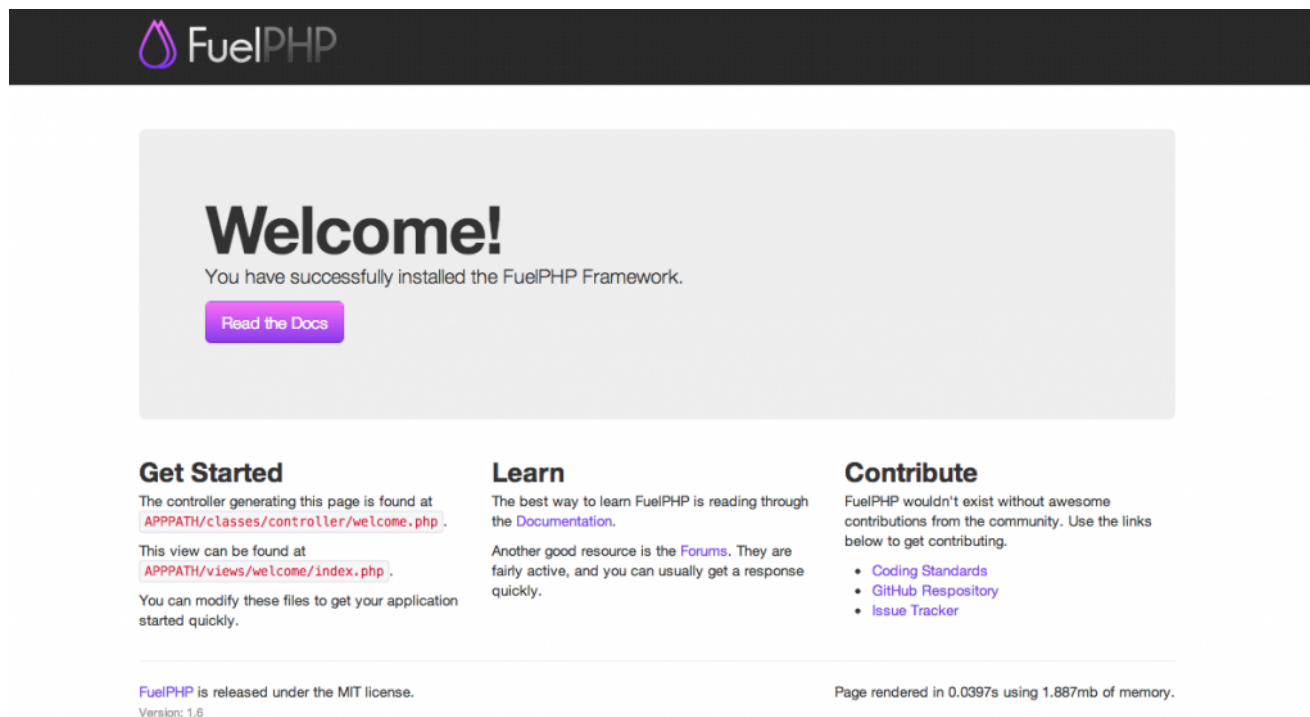


図 23.3 Welcome 画面

## 9. MySQL アドオンを入れる

無料版のアドオンを入れる場合も公式サイトからログインを行いクレジットカードの登録が必要です。ただ無料版の場合は料金が引かれるなど初期費用なども基本ないようです。

MySQL のアドオンは <https://addons.heroku.com/> から search ボックスに「mysql」と検索すると 2013/12 時点で 4 つ見つかりました。

- Adminium Full fledged admin interface without touching your app code heroku addons:add adminium
- Amazon RDS Hook your app up to Amazon' s RDS heroku addons:add amazon\_rds

- ClearDB MySQL Database The high speed, 100% uptime database for your MySQL powered applications. heroku addons:add cleardb
- Xeround Cloud Database  $\alpha$  lpha Scalable, highly available, zero-management cloud database for MySQL heroku addons:add xeround

今回は「ClearDB」を使用します。

- 参考サイト：<http://www.ownway.info/Ruby/index.php?heroku%2Fhow%2Fmanagement%2Fdatabase%2Fcleardb>

### (1) 公式サイトよりクレジットカード登録を行う

### (2) アドオンをインストール

```
$ heroku addons:add cleardb:ignite
Adding cleardb:ignite on tranquil-cliffs-2547... done, v6 (free)
Use `heroku addons:docs cleardb:ignite` to view documentation.
```

インストール完了です。

### (3) 接続情報を確認

```
$ heroku config
=== tranquil-cliffs-2547 Config Vars
BUILDPACK_URL:      https://github.com/winglian/heroku-buildpack-php
CLEARDB_DATABASE_URL: mysql://[ユーザ名]:[パスワード]@[ホスト名]/[DB 名]?reconnect=true
```

CLEARDB\_DATABASE\_URL に接続文字列が表示されます。

```
CLEARDB_DATABASE_URL: mysql://[ユーザ名]:[パスワード]@[ホスト名]/[DB 名]?reconnect=true
```

### (4) 接続文字列は？

```
mysql --host=[ホスト名] --user=[ユーザ名] --password=[パスワード] [DB 名]
```

## 10. FuelPHP プロジェクトから MySQL へ環境変数で接続

```
$cleardb = parse_url(getenv('CLEARDB_DATABASE_URL'));
$conn = new PDO(
    sprintf("mysql:dbname=%s;host=%s", substr($cleardb['path'], 1), $cleardb['host']),
    $cleardb['user'],
    $cleardb['pass']
);
```

ここまでで **PHP + MySQL + Apache** で **FuelPHP** が動作する環境の構築完了です。

### まとめ

Heroku にやや癖があり使いにくいと思ってしまった事もありましたが、慣れるとここまで行うのに 15 分程でできるとおもいます。一度環境を作ってしまえばあとの更新作業はすぐにできますね。他の PaaS はあまり使った事ありませんが、動作確認程度の使用でしたら Heroku でいいと思います。また Heroku から公式に php がサポートされれば、もっとアドオン等増えてくるのではないのでしょうか。

明日はクリスマスイブですね。いいことありますように！

**@mycb750**

Twitter: [@mycb750](https://twitter.com/mycb750)

Blog: <http://to-developer.com/blog/>

## 本当は怖い FuelPHP 1.6 までの Rest コントローラ

FuelPHP Advent Calendar 2013 の 24 日目です。

FuelPHP 1.6 までには、Rest コントローラを使うと XSS 脆弱性を作り込みやすい隠れ機能（アンドキュメントな機能）がありました。具体的には、公式ドキュメントのサンプルコードをそのまま実行すれば、XSS が可能でした。

ただし、FuelPHP 1.7 ではバグのためエラーが発生し XSS は成立せず、1.7.1 で修正されました。

なお、この問題を重視しているのは世界中で私 1 人かも知れませんが、実際に影響があるユーザは少ないのかも知れません。本家は[セキュリティ勧告](#)は出しませんでした（[Changelog](#) に仕様変更の記述はあります）し、[fuelphp.jp Google グループ](#)で告知しましたが、特に反応はありませんでした。

ただし、まだ知らないユーザの方もいるかも知れませんが、今後の参考のためにも、どのような問題があったのかまとめておきます。

### Rest コントローラとは？

Rest コントローラとは、RESTfuel な API を簡単に作成するためのコントローラです。以下は[公式ドキュメント](#)からのサンプルコードです。

```
class Controller_Test extends Controller_Rest
{
    public function get_list()
    {
        return $this->response(array(
            'foo' => Input::get('foo'),
            'baz' => array(
                1, 50, 219
            ),
            'empty' => null
        ));
    }
}
```

<http://localhost/fuel/>以下が FuelPHP だとして、ブラウザから、<http://localhost/fuel/test/list.json> にアクセスすれば、以下のよう JSON の結果が返ります。

```
{"foo":null,"baz":[1,50,219],"empty":null}
```

<http://localhost/fuel/test/list.xml> にアクセスすれば、以下のように XML の結果が返ります。

```
<?xml version="1.0" encoding="utf-8"?>
<xml><foo/><baz><item>1</item><item>50</item><item>219</item></baz><empty/></xml>
```

このようにアクセスされる URL などにより動的に出力を変更する機能を持っています。

## XSS 脆弱性の解説

それでは、`http://localhost/fuel/test/list.html` にアクセスしたらどうでしょうか？ 結果は以下のようになりました。

```
array(3) {
  ["foo"]=>
  string(0) ""
  ["baz"]=>
  array(3) {
    [0]=>
    int(1)
    [1]=>
    int(50)
    [2]=>
    int(219)
  }
  ["empty"]=>
  string(0) ""
}
```

何故か、`var_dump()` した結果が表示されました。

ということで、Firefox から以下の URL にアクセスすれば、めでたく警告ダイアログが表示されます。

- `http://localhost/fuel/test/list.html?foo=%3Cscript%3Ealert%28document.cookie%29%3C/script%3E`

それなら、エスケープすればいいのでは？ と普通は考えます。

```
'foo' => Input::get('foo'),
```

上記の `Input::get('foo')` は `$_GET['foo']` を返す FuelPHP のメソッドです。これを以下のようにエスケープして渡せばいいと考えるでしょう。 `e()` は、FuelPHP での `htmlentities()` 関数みたいなものです。

```
'foo' => e(Input::get('foo')),
```

でもダメです。実際に試してみると、結果は変わりません。

## どこに問題があったのか？

Fuel\Core\Response クラスの `__toString()` メソッドに問題がありました。

```
/**
 * Returns the body as a string.
 *
 * @return string
 */
public function __toString()
{
    // special treatment for array's
    if (is_array($this->body))
```

```

{
    // this var_dump() is here intentionally !
    ob_start();
    var_dump($this->body);
    $this->body = html_entity_decode(ob_get_clean());
}

return (string) $this->body;
}

```

配列の場合は、`var_dump()` して、しかも `html_entity_decode()` して返されています。ここが問題の所在です。Rest コントローラから配列を返す場合は、ここで `var_dump()` されるというわけです。

さきほどのサンプルコードでは、Rest コントローラで配列を作成しているため、配列であることがすぐにわかりますが、ORM などからも結果が配列で返ることもありますので注意してください。

今となっては何故こんなコードがあったのか正確にはわかりませんが、デバッグ用のためのもののように思われます。デバッグ用コードがデフォルトで有効になっており、それが脆弱性につながるという典型的なパターンのように思われます。

## 対策

FuelPHP 1.7.1 (=現在の 1.7/master ブランチ) にアップデートすれば問題は修正されています。

`http://localhost/fuel/test/list.html?foo=%3Cscript%3Ealert%28document.cookie%29%3C/script%3E` にアクセスしても本番環境以外では、以下のようにメッセージと JSON での結果が表示され、

The requested REST method returned an array:

```
{ "foo": "\u003Cscript\u003Ealert(document.cookie)\u003C\/script\u003E", "baz": [ 1, 50, 219 ], "empty": null }
```

本番環境では 406 Not Acceptable が返ります。

アップデートが無理な場合は、Rest コントローラで出力フォーマットを json や xml (html 以外) に固定すればいいでしょう。

```
protected $format = 'json';
```

こう指定することで、URL の拡張子から出力フォーマットを動的に決定することはなくなり、出力フォーマットが固定されます。

というか、FuelPHP のドキュメントには

結果のフォーマットを REST コントローラ内でハードコードすることはバッドプラクティスであることに注意してください。

と書いてあるんですが、私自身はなんでバッドプラクティスなのかよくわかりません。なので、必ず、フォーマットは固定します。バッドプラクティスである理由がわかる方がいましたら、是非、お教え願いたいです。

ああ、それから、JSON を返す Web API などは、そもそもブラウザからの直接のアクセスは禁止した方がいいでしょうね。JSON を返す Web API の作り方は、『[PHP 逆引きレシピ 第2版](#)』で解説されていますので、興味のある方はご覧ください (宣伝)。あの徳丸先生も

この第 2 版は本当に素晴らしい。PHP セキュリティの最新動向をよく把握して、具体的なレシピに落とし込んでいます。すべての PHP 開発者にお勧めします。(…略…) 冒頭に書いたように、この第 2 版は素晴らしいです。「もうペチパーは緩いなんて言わせない」と叫びたいほどのインパクトがあります。

<http://blog.tokumaru.org/2013/12/php12sql.html>

と絶賛されてますので、一家に一冊あって損はないと思います (宣伝)。

あと、FuelPHP 1.7.1 から Rest コントローラからの JSON 出力もそうですが、Format クラスの JSON 出力でのエスケープ (`json_encode()` 関数の第 2 引数のオプション指定) がより安全なものに変更されています。1.7 以前はオプション指定はありませんでした。

## 最後に

どんなフレームワークも完璧ではないと考えた方が現実的です。今まで、脆弱性が修正されたことがないメジャーなフレームワークというのはないと思います。そのため、フレームワークのソースを読み、実装が本当に安全か確認することが有効です。

FuelPHP は規模的にもまだソースが読めるっぽい分量だと思いますし、読みやすいと思いますので、みなさん、がんがんソースを読んで、バグなどがあれば修正する Pull Request を本家に送るなり、とりあえず、Google グループで相談してみるとよいと思います。

ただし、セキュリティ上の問題を発見した場合は、いきなり Pull Request を送ったり、Google グループに投稿するなど、いきなり内容を公開してはいけません。これは、脆弱性情報がいきなり公開されても、即座には対応できないため、ユーザがより危険な状態に置かれてしまうからです。脆弱性情報はその対策とともに公開されるべきものです。

本家へセキュリティ上の問題を報告する場合は、<http://fuelphp.com/contact> のコンタクトフォームからしてください。

### kenjis

FuelPHP まとめ Wiki 管理人。「PHP5 技術者認定上級試験」認定者。

Twitter: [@kenji\\_s](https://twitter.com/kenji_s)

Blog: <http://blog.a-way-out.net/>



## Auth と他モデルにリレーションをつける

### 要旨

User テーブルに対してリレーションを付けたい。自分で作った User モデルであれば `/project_name/fuel/app/class/model/` に有るファイルを編集すればいい。Auth パッケージを使いテーブルを作った場合 `packages` に有るファイルを編集してリレーションをつけて使っていた。 `packages` 内を直接編集するのは気持ち悪いし Auth パッケージを submodule にしてアップデートすることが出来無くなる。解決するには package を新しく作る。そこに作ったファイルにリレーションを記述すればよい。

`packages` を拡張して使う記事は複数あるがリレーションを書いて拡張する方針の物は検索しても見つからなかった。そこでここでは Auth を拡張してリレーションを付け加える方法を書く。私は PHP を使い始めて間もないので、この方針が正しいのか判別がつかない。Advent Calender に投稿することでよりよい方法についてご助言をいただけたらと考えています。

### 下準備

FuelPHP は oil で作ります。 `project_name` フォルダが作られます。Windows のひとは Download して使って下さい。

```
# project_name フォルダを作って、中に FuelPHP をダウンロードする
oil create project_name
```

### MySQL の準備

データベース (`auth_test`) と、管理ユーザーを作ります。

```
# MySQL にログインします
mysql -u root -p
# データベースを作る
create database auth_test;
# 管理ユーザーに権限を与える
# ユーザー名は dbuser, パスワードは lab1lab1
grant all on auth_test.* to dbuser@localhost identified by 'lab1lab1';
# 一旦ログアウトする
exit;
# dbuser でログインする
mysql -u dbuser -p auth_test
# テーブルがないことを確認する
show tables;
```

### 設定ファイルの編集

- [Changeset](#)

`/project_name/fuel/app/config/development/db.php` を編集します。データベースの名前と、使うユーザーとパスワードを入力します。

```
return array(
    'default' => array(
        'connection' => array(
            'dsn' => 'mysql:host=localhost;dbname=auth_test',
            'username' => 'dbuser',
            'password' => 'lab1lab1',
        ),
    ),
);
```

/project\_name/fuel/app/config/config.php を編集します。always load に ORM と Auth を読み込むよう設定します。あとから MyAuth も読み込むよう設定をします。

```
'always_load' => array(
    'packages' => array(
        'orm',
        'auth',
        'myauth',
    ),
),
```

### Auth の設定ファイルをコピーする

- [Changeset](#)

/project\_name/fuel/packages/auth/config/ から auth.php と ormauth.php を /project\_name/fuel/app/config/ にコピーします。

```
$ cp ./fuel/packages/auth/config/auth.php ./fuel/app/config/
$ cp ./fuel/packages/auth/config/ormauth.php ./fuel/app/config/
```

auth.php には driver と salt を設定します。

```
return array(
    'driver' => 'Ormauth',
    'verify_multiple_logins' => false,
    'salt' => 'ZomBie$22@OPS',
    'iterations' => 10000,
);
```

ormauth.php にも同じように login\_hash\_salt を設定します。

```
'login_hash_salt' => 'ZomBie$22@OPS',
```

## ORM を編集する

### 記事テーブルを作る

- [Changeset](#)

Blog のような物を想定しているので、記事を保存するテーブルを作ります。テーブル名は `articles`。 `id`、 `title` (タイトル)、 `comment` (コメント)、 `user_id` (どのユーザーの記事か) の 4 つのデータをもたせます。

```
# migrate 用のファイルをつくる (このコマンドだけではデータベースにテーブルは作られていない)
oil generate model article title:varchar[255] comment:text user_id:int
# これを実行するとテーブルができる
oil refine migrate:up
```

### リレーションを付ける

- [Changeset](#)

`articles` テーブルは `User` に属しています。 `Users` テーブルは複数の `articles` を持ちます。

```
protected static $_belong_to =
  array('user' => array(
    'model_to' => '\MyAuth\Model\Auth_User',
    'key_from' => 'user_id',
    'key_to' => 'id',
    'cascade_save' => false,
    'cascade_delete' => true,
  ));
```

### User テーブルを作る

- [Changeset](#)

`Auth` パッケージにある `migrate` ファイルからテーブルを作ります。

```
# テーブルを作ってもらいます。
oil refine migrate:current --packages=auth
```

### リレーションを付ける

`User` テーブルは複数 (`has_many`) の `article` を持ちます。これは後で書きます。

## packages を書き換える

### ディレクトリ構成

- [Changeset](#)

`/project_name/fuel/packages/`以下に次のようにファイルを作ります。

```
./myauth/
|-- bootstrap.php
`-- classes
   |-- model
   |-- auth
   `-- user.php
```

## ファイル

/auth/model/auth/user.php のファイルを/myauth/model/auth/user.php へコピーします。auth/user.php では、function や \$\_table\_name の変数を消しておきます。そして、継承元を\Orm\Model から\Auth\Model\Auth\_User にかえます。

```
namespace MyAuth\Model;

class Auth_User extends \Auth\Model\Auth_User
{
    protected static $_has_many = array(
        'articles' => array(
            'key_to' => 'user_id',
            'model_to' => 'Model_Article',
            'key_from' => 'id',
            'cascade_save' => false,
            'cascade_delete' => true,
        ),
    );
}
```

## Controller を書き換える

Controller を作り込むのは面倒なので/project\_name/fuel/app/classes/controller/welcome.php を編集します。action\_test、action\_create、action\_login の 3 つのメソッドを作ります。

### Login User を作る。

/project\_name/上 で、oil console を実行し、ユーザー名を samui、パスワードを password、メールアドレスを ss.to13@gmail.com で登録します。

```
Auth::create_user('samui','password','ss.to13@gmail.com');
```

ユーザー ID は 2 または 3 をもつユーザーを作ることができます。

## ログインページを作る

- [Changeset](#)

先程つくったユーザーで勝手にログインするメソッドを作ります。本来は Form を作ってユーザー入力させるべき部分です。今会は面倒なので作りません。

```
public function action_login()
{
    // Auth のインスタンスを作る
    $auth = Auth::instance();
    // Auth でログインできたら、
    if ($auth->login('samui', 'password'))
    {
        // welcome/view に移動する
        Response::redirect('welcome/view');
    }
}
```

### 記事を作る

- [Changeset](#)

article に与えるデータを作るメソッドを書き加えます。

```
public function action_create()
{
    // Auth でログイン出来てなかったら、
    if(! Auth::check())
    {
        // ログインさせる
        Response::redirect('welcome/login');
    }
    // ユーザー ID で、モデルからデータを取得
    $user = \Model\Auth_User::find(Auth::get_user_id()[1]);
    // $user = \MyAuth\Model\Auth_User::find(Auth::get_user_id()[1]);
    $regist = \Model_Article::forge(
        array(
            'title' => 'FirstBlog',
            'comment' => 'Comment!Comment!Comment!',
            'user_id' => Auth::get_user_id()[1],
            'created_at' => time(),
        )
    );
    $user->articles[] = $regist;
    $regist->save();
    $regist->user = $user;
    $user->save();
    Response::redirect('welcome/view');
}
```

### 記事を表示する

- [Changeset](#)

正しくパッケージが読み込まれているか確認します。

```
public function action_view()
{
    // ユーザーデータ取得、
    $user = \Model\Auth_User::find(Auth::get_user_id()[1]);
    // ユーザーデータのインスタンスの名前を表示
    echo get_class($user); // MyAuth\Model\Auth_User
    // ユーザーからブログ記事を取得
    if (count($user->articles) > 0)
        echo get_class(($user->articles[0]));
}
```

### 動作確認

localhost/project\_name/public/index.php/welcome/login にアクセスします。すぐに localhost/project\_name/public/index.php/welcome/view に飛されます。localhost/project\_name/public/index.php/welcome/create に移動します。articles が 1 つ追加されます。

インスタンスの名前が作ったものと一致すれば成功です。

## まとめ

Auth パッケージに僕が最初に不満に思ったことを改善する方法をまとめました。

Auth パッケージを書き換えることで User テーブルとのリレーションを作ると、Auth パッケージの不具合が更新されると対処することが出来ません。Packages の拡張を行うことで解決できることは FuelPHP 界では自明なことのようには扱われていません。しかし、「auth リレーション」の検索ワードではこの解決を見付けることが出来ません。実際にサービスを作ってみると User テーブルと他のテーブルとの関係は直に付けたくなるものです。そこで Auth パッケージを使う所から一歩すんで初学者（フレームワークを使ったことがない人）が躓きそうであることについてまとめておきました。

## 参考文献

packages の拡張については

- [パッケージ - 概要 - FuelPHP ドキュメント](#)
- [FuelPHP でパッケージを拡張する方法について](#)
- [\[FuelPHP\] ネームスペースを上書きしてパッケージのクラスを書き換える](#)

Auth については

- [\[すごい広島\]FuelPHP で認証機能を実装してみる!](#)

独自に作ったユーザーテーブルと他のテーブルにリレーションを付ける

- [モデル間のリレーションをためしてみた \(FuelPHP\)](#)

今回つくった Source

- [AuthExtend](#)

### **samui\_**

学生です！ Web Service を FuelPHP で作りました。AdventCalender に記事を書いて宣伝しようと考えてました。が、致命的なバグが出たので間に合わず…

Twitter: [@samui\\_](#)

Blog: <http://samui13.hatenablog.com/>

## FuelPHP Advent Calendar 2013

---

2013 年 12 月 27 日 v0.9.0 版発行

著 者 FuelPHP Advent Calendar 2013 参加有志

発行所 達人出版会

---