PHP の数あるフレームワークの中で、かなり高速なフレームワーク。
C 拡張で PHP に組み込まれる形で提供されるため、普通にソース丸見えな他のフレームワークとは異なる。

php-framework-benchmark
https://github.com/kenjis/php-framework-benchmark

インストール（raspbian）

raspbian（wheezy）にインストールしてみる。

Download Phalcon
https://phalconphp.com/ja/download

apt-get でインストールできるのは Debian 6.0（squeeze）用みたいなのでソースからコンパイル。
Phalcon-2.0.8 のコンパイルは PHP-5.3.1 で可能だが、バグとかセキュリティ的に PHP-5.3.11 以降が推奨らしい。
ちなみに wheezy は PHP-5.4.45 で OK。

```
# apt-get install php5-dev php5-mysql gcc libpcre3-dev git make

# git clone --depth=1 git://github.com/phalcon/cphalcon.git
# cd cphalcon/build
# ./install
```

raspbian では /usr/lib/php5/20100525+lfs に拡張が入っていて、phalcon.so もここにできる。
実機が Pi1 だと完了まで 30 分ぐらい。

nginx + php5-fpm 構成なので、php5-fpm のルールに倣って ini を作成。めどいから先にインスコしてた mysql.ini をコピる。

```
# cd /etc/php5/mods-available
# cp mysql.ini phalcon.ini

# vim phalcon.ini
; configuration for php Phalcon module
; priority=20
extension=phalcon.so
```

有効にする。

```
# cd /etc/php5/fpm/conf.d
# ln -s ../../mods-available/phalcon.ini 30-phalcon.ini
# /etc/init.d/php5-fpm restart
```

あとはドキュメントルートの index.php に phpinfo() あたりを書いて出力に phalcon があれば OK。

インストール（XAMPP for Windows）

まず XAMPP の VC と PHP のバージョンを確認。

phpinfo() で最初の PHP バージョンと Zend Extension Build や PHP Extension Build あたりを見る。
例として TS,VC11 とあれば Thread Safe 版の Vusial C でコンパイルされたもの。
VC でコンパイルされたものを実行するには VC の runtime が必要だけど、VC9 は Windows XP まで
VC11 は Windows Vista 以降って認識で良いと思う。

Windows の XAMPP は 32bit 用なので x86 の dll、NTS は Non Thread Safe 版で（NTS が付かないもの）を選ぶこと。
あとは先に調べた PHP のバージョン（5.5 とか 5.6 のあたり）と、VC（9 か 11）を合わせていれば OK。

Download Phalcon for Windows
https://phalconphp.com/ja/download/windows

dll を /xampp/php/ext/php_phalcon.dll に配置し、/xampp/php/php.ini に追記する。

```
extension=php_phalcon.dll
```

チュートリアル

tutorial
https://docs.phalconphp.com/en/latest/reference/tutorial.html

チュートリアルのとおりのディレクトリ構成を作り、ファイルにソースをコピペしていけば動作確認できるのは素晴らしい。

しかし Signup へのアクセスから動かなくなる。

```
http://localhost/signup
```

へアクセスしても SignupController の indexAction が呼ばれない。というか IndexController に fooAction を作っても呼び出せなかった（汗）。

とりあえず index.php に

```
use Phalcon\Mvc\Router;

$di->set('router', function() {
    $router = new Router();
    $router->setUriSource(Router::URI_SOURCE_SERVER_REQUEST_URI);
    $router->handle();
    return $router;
});
```

を追加したら動くようになった。

デフォルトでは Apache での動作を想定して .htaccess で

```
RewriteRule ^(.*)$ index.php?_url=/$1 [QSA,L]
```

としてあり、nginx で同等の設定をしなかったから動かなかったようだ（汗）。

DI\FactoryDefault

Phalcon\DI は空の状態から注入。
Phalcon\DI\FactoryDefault は必要そうなものが最初から注入されている。以下

```
router / Phalcon\Mvc\Router / shared
dispatcher / Phalcon\Mvc\Dispatcher / shared
url / Phalcon\Mvc\Url / shared
modelsManager / Phalcon\Mvc\Model\Manager / shared
modelsMetadata / Phalcon\Mvc\Model\MetaData\Memory / shared
response / Phalcon\Http\Response / shared
cookies / Phalcon\Http\Response\Cookies / shared
request / Phalcon\Http\Request / shared
filter / Phalcon\Filter / shared
escaper / Phalcon\Escaper / shared
security / Phalcon\Security / shared
crypt / Phalcon\Crypt / shared
annotations / Phalcon\Annotations\Adapter\Memory / shared
flash / Phalcon\Flash\Direct / shared
flashSession / Phalcon\Flash\Session / shared
tag / Phalcon\Tag / shared
session / Phalcon\Session\Adapter\Files / shared
sessionBag / Phalcon\Session\Bag
eventsManager / Phalcon\Events\Manager / shared
transactionManager / Phalcon\Mvc\Model\Transaction\Manager / shared
assets / Phalcon\Assets\Manager / shared
```

sessionBag だけ shared じゃない。

```
$di = new Phalcon\DI\FactoryDefault();
$router = $di->getShared('router');
$router->setUriSource(Phalcon\Mvc\Router::URI_SOURCE_SERVER_REQUEST_URI);
```

みたいな使い方。

API リファレンス

Phalcon API Documentation
https://api.phalconphp.com/

オリジナルソース

挙動がわからなければ、ソースを読めば良いじゃない。

cphalcon
https://github.com/phillipmadsen/cphalcon

ハマりどころ

Phalcon 2.0.8 で確認。

キャメルケースの処理

コントローラは foo-bar なら FooBarController となる。
アクションは foo-bar ならルーターではじかれ foo_bar は foo_barAction となる。

アクション名のキャメルケース化
　https://docs.phalconphp.com/ja/latest/reference/dispatching.html#id5
ルーティング
　https://docs.phalconphp.com/ja/latest/reference/routing.html

Page Not Found

Phalcon で Page Not Found しようとすると

```
Router でアンマッチ → Page Not Found
↓
Router でマッチしたが Controller が存在しない → Page Not Found
Router でマッチしたが Action が存在しない → Page Not Found
↓
Router でマッチし Controller::Action は存在するが、テンプレートがない → Page Not Found
```

と何回もチェックしないといけない。

■ルーターの初期設定の挙動

```
http://localhost/foo/hoge/page/1
↓
http://localhost/[controller:]/[action:]/[params:]...
```

コントローラー名は

```
foobar または foo-bar または foo_bar
↓
○ FoobarController
○ FooBarController

だけど
http://localhost/foobar → views/foobar
http://localhost/foo-bar → views/foo-bar
http://localhost/foo_bar → views/foo_bar
```

まぁ PHP 自体がクラス名、メソッド（または関数）名の大文字小文字を区別しないので、挙動としてはハイフンやアンダーバーを除去してコントローラ名とする。これは納得。
view に関しては URL のそのままフォルダ名を探すので、存在しなければ空白ページを返す。これが非常にわかりづらい。

続いてアクション名に関しては

```
foo-bar
↓
デフォルトのコントローラ、アクションが呼ばれる（IndexController、indexAction）

foo_bar
↓
○ foo_barAction
× fooBarAction
```

ハイフンに関しては意味不明。普通トップページは最初に作ってるので表示されてしまいエラーにすらならない。
アンダーバーについては除去されず、そのままアクション名となる。存在しなければ handler の exception となる。

ということで調べた。

ルーティング
https://docs.phalconphp.com/ja/latest/reference/routing.html

```
/:controller   /([a-zA-Z0-9\_\-]+)
/:action       /([a-zA-Z0-9\_]+)
```

これが初期設定で

```
コントローラ名がアンマッチ → デフォルトのコントローラ、アクション
アクション名が　アンマッチ → デフォルトのコントローラ、アクション

コントローラ名がマッチ、クラスが存在しない　　→ handler exception
アクション名が　マッチ、アクションが存在しない → handler exception
```

となるが、ユーザーが web ページのリンクをクリックしているような場合は、製作者がミスしないかぎりアンマッチはない。
しかし、直接 URL を入力してくるユーザーやロボットの総当りなど、普通に考えたらアンマッチだろうが存在しなかろうが Page Not Found が普通なのに、そうはなっていない。

チュートリアルの invo では

```
$di->set('dispatcher' => function () use($id) {
    $eventsManager = new EventsManager;
    :
    $eventsManager->attach('dispatch:beforeException', new NotFoundPlugin);
    :
    return $dispatcher;
});

class NotFoundPlugin extends Plugin
{
    public function beforeException(Event $event, MvcDispatcher $dispatcher, Exception $exception)
    {
        if ($exception instanceof DispatcherException) {
            switch ($exception->getCode()) {
                case Dispatcher::EXCEPTION_HANDLER_NOT_FOUND: // ← コントローラが存在しない
                case Dispatcher::EXCEPTION_ACTION_NOT_FOUND:  // ← アクションが存在しない
                    $dispatcher->forward(array(
                        'controller' => 'errors',
                        'action' => 'show404'
                    ));
                    return false;
            }
        }

        $dispatcher->forward(array(
            'controller' => 'errors',
            'action'     => 'show500'
        ));
        return false;
    }
}
```

というようになっており、ルーターにマッチしない場合はデフォルト（トップページ）を表示す

る。

なのでルーター側は

```
$router = $di->getShared('router');
$router->notFound(array(
    'controller' => 'errors',
    'action' => 'show404',
));
```

とする必要があるが、$router->notFound() すると $router->setDefaults() やルート無し '/' でアクセスされた場合のトップページ表示が死ぬので

```
$router = $di->getShared('router');
$router->add('/', array( // setDefaults() で補完されても match しない為に追加
    'controller' => 'index',
    'action' => 'index',
));
$router->setDefaults(array( // url で省略された際に補完する設定値
    'controller' => 'index',
    'action' => 'index',
));
$router->notFound(array( // router に match しなかった場合
    'controller' => 'errors',
    'action' => 'show404',
));
```

としなければならない。

で、ラスト。invo では

```
http://localhost/index/index  → トップページ
http://localhost/in-dex/index → Unauthorized のページ
```

となる。

いままでの処理で行くと、index と in-dex は同じ IndexController の呼び出しなので Exception は発生しない。
ところが認証で表示されるページは URL から抽出された in-dex のままなので、role に設定された access list の index にマッチしない。
よって Unauthorized のページ表示。

要するに router が controller や action として URL から抽出した部分を整形しないので、ちょっとした違いを吸収できず、下流の処理に影響が出る。なんか微妙に使えない（汗）。

■ View の RenderLevel とイベント view:notFoundView

結論から書くと、view:notFoundView だったら Page Not Found に転送しようとかめどい。

URL からコントローラ、ビューのおさらい。

```
/foobar/baz  → FooberController の bazAction → views/foobar/baz.phtml
/foo-bar/baz → FoobarController の bazAction → views/foo-bar/baz.phtml
```

というようになる。正確には

```
/foo-bar/baz → FooBarController の bazAction → views/foo-bar/baz.phtml
```

かもしれないが、PHP ではクラス名等の大文字小文字を区別しない。

```
/foo-bar/baz  → FooberController の bazAction → views/foobar/baz.phtml
```

だと処理は正常に行われて画面は真っ白になる。

で、本題の RenderLevelってのは初期値で 5 になってて、これは View の初期動作で

```
views/foobar/baz.phtml     → 1 主にコンテンツ部分
views/layouts/foobar.phtml → 3 主にメニューとかナビの部分
views/index.phtml          → 5 HTML の HEAD とかの部分
```

が呼ばれるが、ファイルが無ければ空白を返す。
5 回のループでページソースの中心から外に向けて組み込んで出力する。

で、最初の話に戻って、ページは存在しないんだから Page Not Found にしたいと思い、view:notFoundView イベントに処理を書いたら無限ループ。
パスがない 2、4 は無視されるようですが、baz.phtml のみってこともあるし。

```
$eventsManager->attach('view:notFoundView', function($event, $view) {
    if($view->getCurrentRenderLevel == 1) {
        // view rendering
        $view->start();
        $view->render('errors', 'e404');
        $view->finish();
        echo $view->getContent();
        exit;
    }
});
$view->setEventsManager($eventsManager);
```

とやればテンプレートを使っていない場合の処理を書けるが、コントローラでビューを使わない場合は

```
public function indexAction() {
    $this->view->disable();
    // 処理
}
```

としないといけない。

request

コントローラにて値を取得する場合。

```
$data = $this->request->getPost('data', array('trim', 'striptags'), ' 初期値 ');
```

フォームから送られるデータが多次元配列からなる場合には対応しない。

詳しくは cphalcon から cphalcon/phalcon/http/request.zep で、getPost() と getHelper() 参照のこと。

メソッドと一番目の引数は

- getPost('data') = $_POST['data']
- getQuery('data') = $_GET['data']
- get('data') = $_REQUEST['data']
- getPut('data') = $_PUT['data']

二番目の引数はサニタイズで最初から用意されているものは

- email
- int ・・・ 0-9+-
- float ・・・ 0-9+-.
- alphanum ・・・ a-zA-Z0-9
- string ・・・ HTML タグをエスケープ
- striptags ・・・ HTML タグを除く
- trim
- lower
- upper

array() を使うことにより複数設定可能。

三番目の引数は初期値を設定しておきたい場合。
サニタイズの必要は無いが初期値は設定したいなら

```
$page = $this->request->getQuery('page', null, 1);
```

dispatcher

```
/controller/action/param/param/param...
```

というように URL にパラメータ（param）を設定した場合は

```
$id = $this->dispatcher->getParam(0, 'int', 1);
```

0 は最初の param、1 は次の param、2 は次の次の param... みたいな感じ。