Hackで作る
マイクロフレームワーク

yuuki takezawa (ytake)

PHPerKaigi 2018

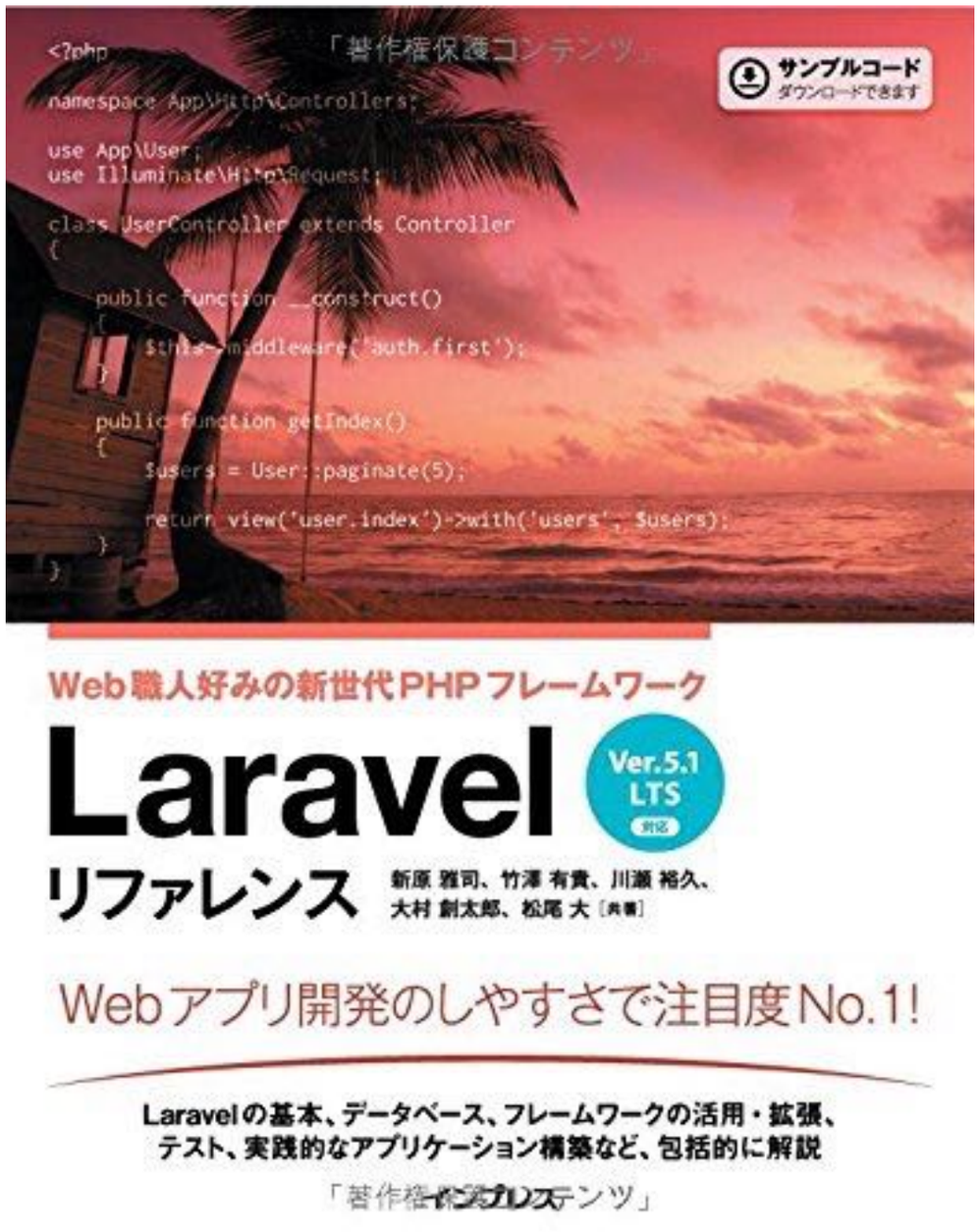
Profile

- 竹澤 有貴 / ytake
- 株式会社アイスタイル
- **PHP**, **Hack**, **Go**, **Scala**
- **Apache Hadoop**, **Apache Spark**, **Apache Kafka**

- twitter https://twitter.com/ex_takezawa
- facebook https://www.facebook.com/yuuki.takezawa
- github https://github.com/ytake

author


Software Design plus
モダンな開発を実現するPHPフレームワーク！
ガッチリ！最新技術
Laravel
エキスパート
速習!!
養成読本
Laravel Expert
[Web職人の開発効率をグングン上げる！]
注目のPHPフレームワーク
Laravelを理解するためのキーワード
▶IoCコンテナ ▶ファサード
▶サービスプロバイダ ▶Eloquent
◎実践! REST APIアプリ
◎Laravel 5新機能詳解
◎MVCモデルの基礎
◎ブームまでのストーリー


「著作権保護コンテンツ」
サンプルコード
ダウンロードできます
Web職人好みの新世代PHPフレームワーク
Laravel
Ver.5.1
LTS
対応
リファレンス
新原 雅司、竹澤 有貴、川瀬 裕久、
大村 創太郎、松尾 大［共著］
Webアプリ開発のしやすさで注目度No.1!
Laravelの基本、データベース、フレームワークの活用・拡張、
テスト、実践的なアプリケーション構築など、包括的に解説
「著作権保護コンテンツ」


@cosme

ZEND FRAMEWORK

cassandra

php

goa

presto

Scala

MySQL®

Microsoft® SQL Server®

hadoop

はなすこと

- **開発するための知識**

  .hhconfig、decl / strict、hhi

- **強力なライブラリ**

  hhvm/hhvm-autoload、hhvm/hack-router

- **PHPライブラリとの互換を保つ開発**

<?hh

開発するための知識

<?hh Hack？

- PHPのコードは基本的にそのまま実行可能
- 厳格なType Checker
- Async、XHP、Generics、Collections
- 3.24 LTS (2018/03/10 現在)
- HHVMは、PHPとHack両方をサポートするのではなく、Hackを対象としていく

*Forget PHP! Facebook's HHVM engine switches to Hack instead
https://www.infoworld.com/article/3226489/web-development/forget-php-facebooks-hhvm-engine-switches-to-hack-instead.html 参照

<?hh Type checker

- モードは3つ Partial / Strict / Decl

- デフォルトではPartial

<?hh Type Check: Partial

- PHPの型宣言 strictと同程度
- 必要以上に型チェックはしない
- 主にPHPのコードを Hackとして実行する
  コード移植中のファイル等で利用
- 参照渡し 利用可能

<?hh Type Check: Decl

- <?hh // decl
- 型チェックはしない
- 他のコードチェック時には参照される

- New Hack code should never be written in decl mode

<?hh Type Check: Strict

- <?hh // strict
- PHP依存がなく、100% Hackで実装する場合に選択
- 厳格な型チェックを行うモードのため、

  コードレビュー時の型宣言についての議論はなし
- 想定外の型変換などは行われないため、

  レビューはクラス設計・アーキテクチャなどに

  フォーカスできる
- Type Checkerに教えるためのコードに

<?hh Type Check: Strict

- 最も厳格なモードのため、

  PHPライブラリを使うコードでは型宣言エラー発生

  => hhiファイルを作成(後述)

- __invokeの扱いがPHPと異なるため型エラー発生

  => strictで__invokeで処理するミドルウェア(非PSR-15)

  を取り入れる場合は注意

- Hackで実装するのであればStrictを基準に、

  一部だけPartialがベスト

<?hh callableに気をつける

- PHPでは callableで型宣言されているものはClosure、

  または__invokeメソッドを持つクラスであれば良い

- Hackではcallableではなく、無名関数の引数も型宣言

  __invokeはcallable扱いではない

  -> PHPの __invoke を実装するミドルウェア系を

  そのままstrictで移植はできない

- inst_meth という選択肢

```
class Example {
  public function foo<T>(T $t): T {
    return $t;
  }
}
```

Must be a constant string.

```
inst_meth(new Example(), ‘foo’);
```

<?hh

hhi

<?hh // strict

- TypeScript, flowと同様な型定義ファイル
- 単体では実行できない
- 型定義ファイルを作ることで、型宣言を行い、
  厳格モードで実行することが可能

- 使いたいPHPのコードも補完してほしい！！
  という方は作成してpackagistで公開してください！

packagistにあるhhiファイルたち

- hack-psr/psr7-http-message-hhi
- 91carriage/phpunit-hhi
- ytake/psr-http-handlers-hhi
- ytake/psr-container-hhi
- libreworks/psr3-log-hhi

- Hack対応ライブラリ開発者が使うもの程度しかありません
- nikic/fast-route にも含まれています

PSR For Hack

- https://github.com/facebookexperimental/hack-http-request-response-interfaces

- **This project aims to create standard request and response interfaces for Hack, using PSR-7 as a starting point.**

<?hh

hhi

PSR-11 Container Interface : Example

```
<?hh // strict

namespace Psr\Container;

interface ContainerInterface
{
    public function get(string $id): mixed;

    public function has(string $id): bool;
}
```

戻りの型宣言を追加しただけ

<?hh

.hhconfig

<?hh .hhconfig

- Hackで実行環境に設置するファイル
- 実は様々な設定を記述できる

- PHP利用を想定しない(PHP混在不可)

  assume_php = false(default: true)
- Type Checker 一部無視

  ignored_paths = ["vendor/hhvm/hhast/.+"]

<?hh

強力なライブラリ

<?hh

hhvm/hhvm-autoload

<?hh HHVM-Autoload

- HHVM環境専用のComposer Plugin

  A Composer plugin for autoloading classes, enums, functions, typedefs, and constants on HHVM.

- vendor/autoload.php を vendor/hh_autoload.php に
- hh_autoload.jsonを設置

<?hh HHVM-Autoload

- 前述したhhiファイルなどをvendorディレクトリから見つけ出し、TypeChecker対応だけではなく、Nuclide, VSCの補完としても作用
- Hackで実装する場合は必ず入れておきましょう

<?hh Composer install

- php7依存のものはiniファイルかコマンドで解決

```
hhvm -d xdebug.enable=0 -d hhvm.jit=0 -d
hhvm.php7.all=1\
-d hhvm.hack.lang.auto_typecheck=0 \
$(which composer) require vendor/package
```

Zend Expressive with HHVM/Hack

<?hh Zend Expressive 1.0

- HHVM/Hackでも、何も気にせず利用可能
- Zend ServiceManager + HHVM/Hackで実装・稼働
- PHP感覚で手軽にHHVM/Hackアプリケーションを開発する場合にオススメ

<?hh Zend Expressive >= 2.0

- zend-config-aggregator の一部のコードが動作せず

  __invoke, yieldが動かず (Generator<Tk, +Tv, -Ts>)

- fast-routeのhhiがメンテされていないためType Error

  PHPとの分離が進むためPRせず

- 動かない部分を

  Hack実装のライブラリと入れ替えることに

<?hh

- PHPライブラリの実装を気にするのは・・・

- PHP並みにコード補完ができ、

Hackならではの実装をするにはHackで作るしかない

<?hh

hhvm/hack-router

<?hh hack-router

- PSR-7対応のHack専用Routerライブラリ

- 以前はfast-route拡張だったが除外

- Genericsを使い、

  Routerの実装を知らずに利用できるライブラリ

- PHPサポートをやめることにあたり、

  このライブラリベースに移行

指定したクラス、IFを継承、実装したクラス名宣言

```
type TResponder = ImmVector<classname<MiddlewareInterface>>;

final class Router extends BaseRouter<TResponder> {
```

Routerに格納し、マッチ時にTResponderを返却

```
  <<__Override>>
  protected function getRoutes(
  ): ImmMap<HackRouterHttpMethod, ImmMap<string, TResponder>> {
    $i = $this->routeMap->getIterator();
    $map = [];
    while ($i->valid()) {
      $map[$this->convertHttpMethod($i->key())] = $i->current();
      $i->next();
    }
    return new ImmMap($map);
  }
```

<?hh

ytake/hh-container

<?hh PSR-11

- PHPライブラリを使い続ける意味もあまりないため、
  Hack専用のPimpleライクなコンテナを開発

- PSR-11 hhiを用意し、Hackで実装

- Hackならではの方法でSingletonは <<__Memoize>>

- インスタンス解決方法定義は自由

```
public function set(
  string $id,
  (function(FactoryContainer): mixed) $callback,
  Scope $scope = Scope::PROTOTYPE,
): void {
  if (!$this->locked) {
    $this->bindings->add(Pair {$id, $callback});
    $this->scopes->add(Pair {$id, $scope});
  }
}
```

Closure Type

enums

Map

```
use Ytake\HHContainer\FactoryContainer;

$container = new FactoryContainer();
$container->set(
  MessageClass::class,
  $container ==> new MessageClass(‘testing')
);
```

Lambda

```
$container->parameters(
  MessageClient::class,
  'message',
  $container ==> $container->get(‘message.class’)
);

$instance = $container->get(MessageClient::class);
```

PSR-11実装

```
final class TestingInvokable {
  public function __invoke(FactoryContainer $container): int {
    return 1;
  }
}

$container = new \Ytake\HHContainer\FactoryContainer();
$container->set(TestingInvokable::class, $container ==>
  $container->callable(
    new \Ytake\HHContainer\Invokable(
      new TestingInvokable(), '__invoke', $container
    )
  )
);
```

Zend ServiceManager Factory風

インスタンス生成時に実行する

<?hh PSR-11

- Zend ServiceManager互換を目指したライト版

- Zend Expressive対応したところ、
  動的メソッドコールは推奨されていないため、
  Type Error

- /* UNSAFE_EXPR */

厄介なmixed

PSR-11の例(strict)

```
<?hh // strict

namespace Psr\Container;

interface ContainerInterface
{
    public function get(string $id): mixed;

    public function has(string $id): bool;
}
```

PSR-11 get (): mixed

```
$container = require 'config/services.php';
$app = $container->get('Zend\Expressive\Application');
```

mixedなため、何が返却されるかわからない

```
$config = $container->get('config_array');
if (is_array($config)) {
  if (array_key_exists('config_array', $config)) {
    return $config['config_array'];
  }
}
```

arrayであることを示す

```
$logger = $container->get(‘LoggerInterface');
invariant(
  $logger instanceof LoggerInterface,
  "Interface '\Psr\Log\LoggerInterface' is not implemented by this class",
);
```

invariantはHackで用意されている関数

第一引数の条件がfalseの場合にType Error

```
class TypeAssert {
  const type Tk = LoggerInterface;
  public static function assert<Tk>(Tk $t): this::Tk  {
    invariant(
      $logger instanceof LoggerInterface,
      "Interface '\Psr\Log\LoggerInterface' is not implemented by this class",
    );
    return $t;
  }
}
```

<?hh mixed

- 様々な値が返却されるため、mixedは使うべきではない
- PSRにこだわる必要もない
  (準拠しないことを検討中)
- 大規模な開発などではなるべくStrict(弊社談)

Forget PHP! Facebook's HHVM engine switches to Hack instead

<?hh

hhvm/type-assert

```
use namespace Facebook\TypeAssert;

class Foo {
  const type TAPIResponse = shape(
    'id' => int,
    'user' => string,
    'data' => shape(
      /* ... */
    ),
  );

  public static function getAPIResponse(): self::TAPIResponse {
    $json_string = file_get_contents('https://api.example.com');
    $array = json_decode($json_string, true);
    return TypeAssert\matches_type_structure(
      type_structure(self::class, 'TAPIResponse'),
      $array,
    );
  }
}
```

Shapeでフィールド指定

あいまいな型返却APIを検査

RouterとContainer、

簡単なValidationがあれば

最低限の動作が可能に

Dependency / Container

Build Application

Routing/Dispatcher

Middleware

hack routerのTResponder

```
type TResponder =
ImmVector<classname<\Psr\Http\Server\MiddlewareInterface>>;
```

リクエストに対応するものが登録されて入れば、
TResponderで指定した型を返却する

GETに対応させるRouteを記述

```
\Nazg\Http\HttpMethod::GET => ImmMap {
  '/' => ImmVector {App\Action\IndexAction::class},
},
```

'/' にアクセス時に起動するミドルウェアを指定

ImmVector<classname<\Psr\Http\Server\MiddlewareInterface>>
記述した通りに実行される

Actionクラスも同インターフェース実装のため区別なし

```
public function run(ServerRequestInterface $serverRequest): void {
  $container = $this->getContainer();
  $this->bootstrap($container);
  $router = $container->get(BaseRouter::class);
  invariant(
    $router instanceof BaseRouter,
    "%s class must extend %s",
    get_class($router),
    BaseRouter::class,
  );
list($middleware, $attributes) = $router->routePsr7Request($serverRequest);
```

hack-router継承クラスを取り出し

from PSR-7

```
public function __construct(
  Traversable<classname<MiddlewareInterface>> $queue,
  ?Resolvable $resolver = null,
){
  $this->queue = new Vector($queue);
  $this->resolver =
    (is_null($resolver)) ? new InstanceResolver() : $resolver;
}
```

Collection (Vector)

```
public function shift(): MiddlewareInterface {
  $this->queue->reverse();
  $current = $this->queue->pop();
  return $this->resolver->resolve($current);
}
public function cancel(int $index): Vector<classname<MiddlewareInterface>> {
  return $this->queue->removeKey($index);
}
```

Vector 操作

一つずつ取り出して実行

Vector 操作

指定したインデックスのアイテムを削除

https://github.com/ytake/nazg-skeleton

https://github.com/nazg-hack/framework

<?hh 小さなライブラリの集合体へ

- 最近のPHPと同様に疎結合に
- PSR-11, PSR-7, PSR-15を利用したため、

  PHPのライブラリもHackで実行可能なものはOK
- routerはhack-router以上のものがないため、

  hack-routerにのみ依存
- PSR-7はzend-diactorosをデフォルトに

  (Symfony Component 4以降はHack はサポートされない)

<?hh 小さなライブラリの集合体

- Request BodyなどのValidationにtype-assert
- mixedをケアする

<?hh

- Hackだから何か特別なもの、というのは無い

  *ランタイム云々は除く

- バグが少ない堅実なアプリケーションへ
- コードレビュー負荷軽減
- みんなが思うほど難しくない！
- PHP気分でかんたんなマイクロフレームワーク開発

<?hh おまけ

- Goでやらなかったのか？

  -> Go(Goa, Echo)は多くのAPIで導入済み

  大きく言語を変えずに、

  PHPの開発者が使えるものを増やしたかった

- segmentation faultつらくないですか？

  hhvm-dbg + straceでなんとかなります