Composer解体新書

Anatomy of Composer, in Runtime

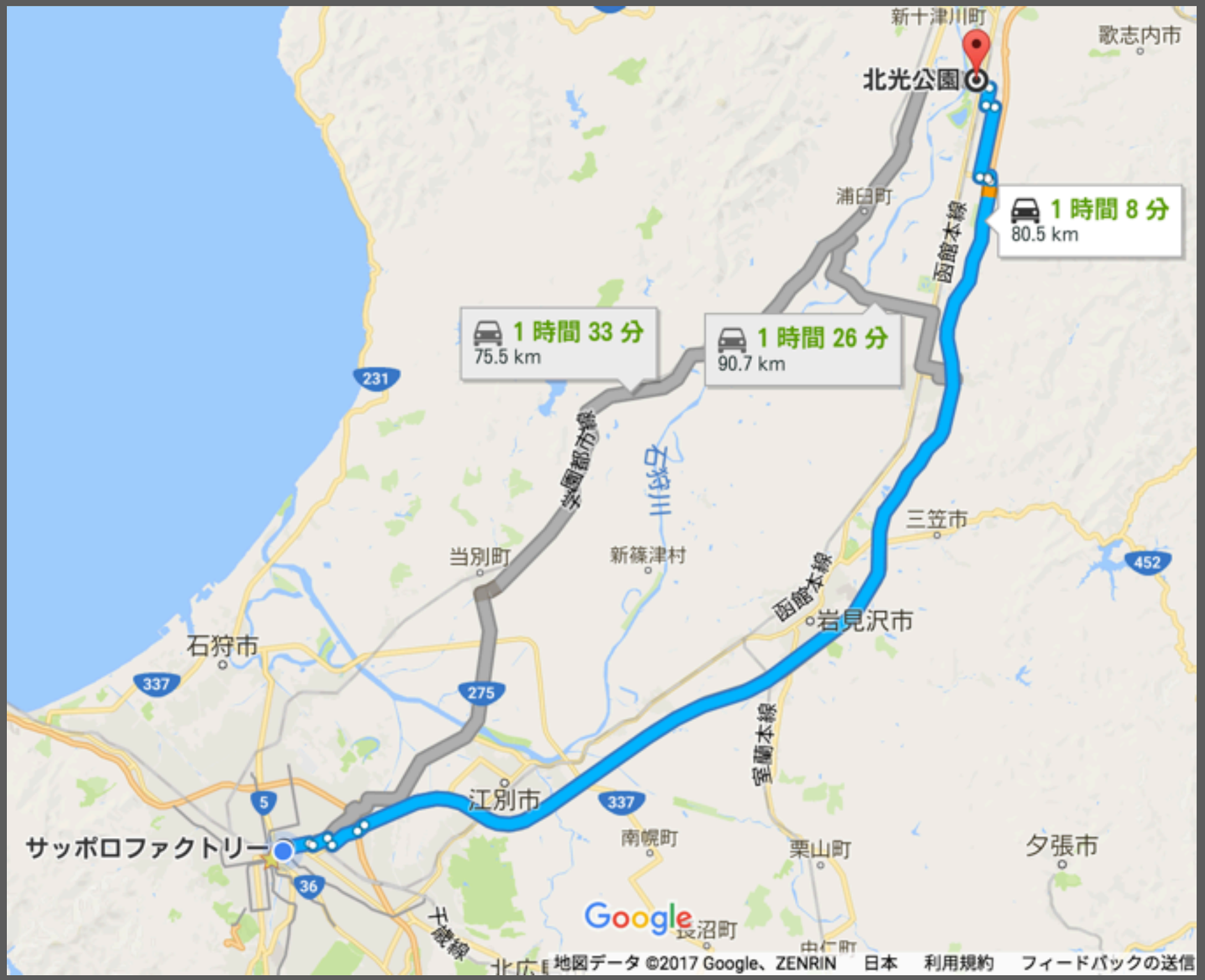
PHPカンファレンス北海道出張版(仮)

PHPオフ会 #phpcondo2017

お前誰よ

- うさみけんた（@tadsan）/ Zonu.EXE
  - GitHub/Packagistでは id: zonuexe
- 砂川出身、北海道工業大学
- 2012年11月にピクシブ株式会社に入社
- Emacs Lisper, PHPer
  - 入社前は自宅警備をしながらRuby書いてた
- Qiitaに記事を書いたり変なコメントしてるよ
- 2017年、PHPカンファレンス全部でしゃべった

pixiv


北光公園
新十津川町
歌志内市
浦臼町
函館本線
1 時間 8 分
80.5 km
1 時間 33 分
75.5 km
1 時間 26 分
90.7 km
231
札幌都市高速
石狩川
三笠市
当別町
新篠津村
452
函館本線
岩見沢市
石狩市
337
275
室蘭本線
江別市
337
5
南幌町
栗山町
夕張市
サッポロファクトリー
36
Google
長沼町
千歳線
地図データ ©2017 Google、ZENRIN
日本
利用規約
フィードバックの送信

Composer

使ってますか？

Dependency Manager for PHP

A. 使ってない

B. FW経由で

C. 直接使ってる

分かれそう

本日しない話

PHPのクラスのオートローディングの説明

プロジェクトでの Composerの運用方法

過去の発表を
読んでくださいい

仕事で使えるComposer

無修正版

ピクシブ株式会社
うさみけんた @tadsan

pixiv

PHPカンファレンス北海道2016

https://niconare.nicovideo.jp/watch/kn1371

Webアプリケーション開発のためのプログラミング技術情報誌

FOR ALL WEB APPLICATION DEVELOPERS ウェブDBプレス

WEB+DB PRESS

vol. 91 2016

すぐに使えて ずっと役立つ!

データ構造の基礎知識

- 配列
- 連結リスト
- ハッシュテーブル
- 木構造

Arduino、Raspberry Piで電子工作

はじめてのIoT

iOSアプリ開発最前線

iOS 9 Swift 2 Xcode 7 CI環境

Slack/Botkitでチームコミュニケーション

PHPの名前空間とオートローディング

Progressive Web Apps

Qiita キーワードを入力 News Advent Calendar 2017 もう登録した？ ユーザ登録 ログイン

ComposerでPHPの依存関係を管理する

PHP 10406 Composer 331

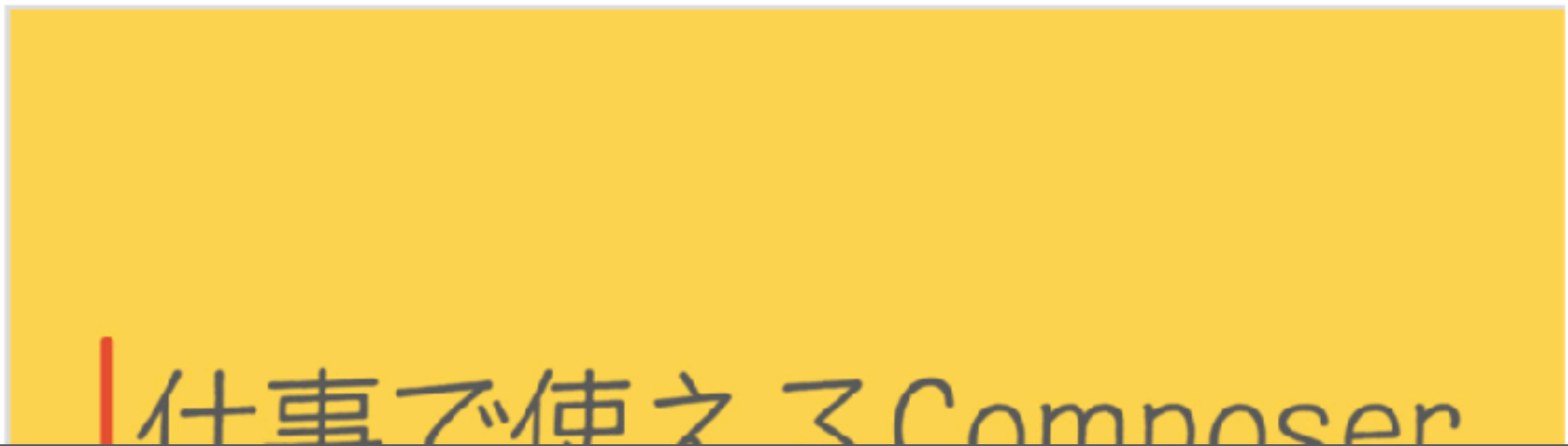
28 いいね 0 コメント

いいね

tadsan 2017年05月18日に投稿 ストック

モダンなPHPの依存管理(パッケージ管理)に既に欠かせないものとなったComposerの導入と運用方法について説明します。「仕事で使えるComposer」でもざっくりと紹介しましたが、今回はもうちょっとだけ詳細に書きます。

概要についてざっくりと知りたい型は、先にこちらのスライドをご覧ください。



tadsan
9156 Contribution
フォロー

人気の投稿

- ライセンスの選択を恐れる必要はありません
- 初心者を戒めるPHP
- モダンPHPアンチパターン
- PHPのモダンな開発環境を紹介する
- いま熱い最新スクリーンエディタmicro ことはじめ

Organization

https://qiita.com/tadsan/items/86099d44d12f1103b0a0

includeって書きたくない僕たちのためのオートローディングとComposer

PHP 10406 Composer 331

tadsan 2017年10月23日に更新 2

こんにちはこんにちは、PHP書いてますか？ include_once してますか？ それともキミは require_once 派？

ところで、現代的なPHPではクラスファイル(ここでは class , trait , interface を含む定義ファイル)では、わざわざファイルを include / require しなくても自動的に読み込む機能をカンタンに構築できる環境があるので、紹介いたします。

この記事は手を動かして動作確認しながら読めるように構成してありますので、斜め読みするだけではもったいないですよ ヾ(〃＞＜)ﾉﾞ

はじめに

今回の記事ではクラスの自動ロード(オートローディング)の概要に絞って解説しますが、名前空間の文法や細かい説明を含めて包括的に解説した記事は、既にWEB+DB PRESS Vol.91｜技術評論社にて「PHP大規模開発入門 第12回 名前空間とオートローディング」として発表済みです。 ~~現代仮名遣いで書かれてるので職場でもオススメしやすいですね！！！~~

tadsan
9156 Contribution
フォロー

人気の投稿
- ライセンスの選択を恐れる必要はありません
- 初心者を戒めるPHP
- モダンPHPアンチパターン
- PHPのモダンな開発環境を紹介する
- いま熱い最新スクリーンエディタmicro ことはじめ

Organization

Composerで
できること

ぎっくり

プロジェクトが
依存するパッケージを
ダウンロードしてくる

プロジェクトが依存するパッケージを利用可能にする

今日は実行時の
話をします

基本はComposerの
autoload.phpを読み込
めば準備完了する

Autoloading

For libraries that specify autoload information, Composer generates a `vendor/autoload.php` file. You can simply include this file and start using the classes that those libraries provide without any extra work:

```php
require __DIR__ . '/vendor/autoload.php';

$log = new Monolog\Logger('name');
$log->pushHandler(new Monolog\Handler\StreamHandler('app.log', Monolog\Logger::WAR
$log->addWarning('Foo');
```

復習

PHPスクリプトの読み込みかた

require, require_once

include, include_once

場合によるけど
基本だいたい同じ

どこから？

A. ファイルシステムの絶対パス
B. include_pathからの相対パス
C. プロトコル（ラッパー）

ファイルシステムの絶対パス

- `require '/path/to/file.php';`
  - ファイルシステム内のフルパス
- `require __DIR__.'/../Klass.php';`
  - マジック定数__DIR__で
    スクリプトのディレクトリが入る

include_pathからの相対パス

- include_pathにディレクトリを追加する
  - だいたいデフォルトに "." が入ってる

```
require 'Hoge.php';
require 'Fuga/Piyo.php';
```

  - ディレクトリ内も探索される

プロトコル(ラッパー)

- fopen('php://stderr', 'r') とか file_get_contents('http://example.com/') とかやるための仕組み
- ……が、セキュリティの懸念もあるので、デフォルトでは php.ini で allow_url_include = Off になってる

改めて Composer

オートロード
機能ついてる

autoload

Autoload mapping for a PHP autoloader.

`PSR-4` and `PSR-0` autoloading, `classmap` generation and `files` includes are supported.

PSR-4 is the recommended way since it offers greater ease of use (no need to regenerate the autoloader when you add classes).

PSR-4

Under the `psr-4` key you define a mapping from namespaces to paths, relative to the package root. When autoloading a class like `Foo\\Bar\\Baz` a namespace prefix `Foo\\` pointing to a directory `src/` means that the autoloader will look for a file named `src/Bar/Baz.php` and include it if present. Note that as opposed to the older PSR-0 style, the prefix (`Foo\\`) is **not** present in the file path.

Namespace prefixes must end in `\\` to avoid conflicts between similar prefixes. For example `Foo` would match classes in the `FooBar` namespace so the trailing backslashes solve the problem: `Foo\\` and `FooBar\\` are distinct.

どうやって
読み込んでるの？

autoload.phpを読もう

みなさまの手元のComposer
プロジェクトを
見比べながらお聞きくださいい

```
% composer --version
Composer version 1.5.2 2017-09-11 16:59:25
```

```
% composer install --no-dev --prefer-dist
```

vendor/autoload.php

```
<?php

// autoload.php @generated by Composer

require_once __DIR__ . '/composer/autoload_real.php';

return
ComposerAutoloaderInitbed8ae7d525347237f9cdad08c8930b3::getLoader();
```

vendor/autoload.php

1.autoload_real.phpを読み込む

2.謎のクラスの静的メソッドを実行

- たぶん変なバイトキャッシュを残さないためにcontent-hashが付いたクラス名

```php
<?php

// autoload_real.php @generated by Composer

class ComposerAutoloaderInitbed8ae7d525347237f9cdad08c8930b3
{
    private static $loader;

    public static function loadClassLoader($class)
    {
        if ('Composer\Autoload\ClassLoader' === $class) {
            require __DIR__ . '/ClassLoader.php';
        }
    }

    public static function getLoader()
    {
        if (null !== self::$loader) {
            return self::$loader;
        }

        spl_autoload_register(array('ComposerAutoloaderInitbed8ae7d525347237f9cdad08c8930b3', 'loadClassLoader'), true, true);
        self::$loader = $loader = new \Composer\Autoload\ClassLoader();
        spl_autoload_unregister(array('ComposerAutoloaderInitbed8ae7d525347237f9cdad08c8930b3', 'loadClassLoader'));

        $useStaticLoader = PHP_VERSION_ID >= 50600 && !defined('HHVM_VERSION') && (!function_exists('zend_loader_file_encoded') || !zend_loader_file_encoded());
        if ($useStaticLoader) {
            require_once __DIR__ . '/autoload_static.php';

            call_user_func(\Composer\Autoload\ComposerStaticInitbedBae7d525347237f9cdad08cB930b3::getInitializer($loader));
        } else {
            $map = require __DIR__ . '/autoload_namespaces.php';
            foreach ($map as $namespace => $path) {
                $loader->set($namespace, $path);
            }

            $map = require __DIR__ . '/autoload_psr4.php';
            foreach ($map as $namespace => $path) {
                $loader->setPsr4($namespace, $path);
            }

            $classMap = require __DIR__ . '/autoload_classmap.php';
            if ($classMap) {
                $loader->addClassMap($classMap);
            }
        }

        $loader->register(true);

        return $loader;
    }
}
```

autoload_real.php

1.$loaderが生成済みなら返すだけ

2.クラスローダーを読み込むだけのクラス

ローダーを登録して、使ったらすぐに

spl_autoload_unregister する

- なんか環境依存の
  ワークアラウンドっぽい気がする

autoload_real.php

```
if (null !== self::$loader) {
    return self::$loader;
}

// Composer\Autoload クラスを読み込むだけのローダー
spl_autoload_register(array('ComposerAutoloaderInitbedRy', 'loadClassLoader'),
true, true);
self::$loader = $loader = new \Composer\Autoload\ClassLoader();
spl_autoload_unregister(array('ComposerAutoloaderInitbedRy', 'loadClassLoader'));
```

autoload_real.php

1. PHPの実行時環境によってクラス定義の読み込みを変更
2. autoload_static.php VS 三兄弟
    - PHP5.6以上で、HHVMじゃなくて、zend_loader_file_encoded()が無効ならautoload_staticが利用される

autoload_real.php

```
$useStaticLoader = PHP_VERSION_ID >= 50600 && !defined('HHVM_VERSION') &&
     (!function_exists('zend_loader_file_encoded') || !zend_loader_file_encoded());
if ($useStaticLoader) {
    require_once __DIR__ . '/autoload_static.php';
    call_user_func(\Composer\Autoload\ComposerStaticInitbedRy::getInitializer($loader));
} else {
    $map = require __DIR__ . '/autoload_namespaces.php';
    foreach ($map as $namespace => $path) {
        $loader->set($namespace, $path);
    }
    $map = require __DIR__ . '/autoload_psr4.php';
    foreach ($map as $namespace => $path) {
        $loader->setPsr4($namespace, $path);
    }
    $classMap = require __DIR__ . '/autoload_classmap.php';
    if ($classMap) {
        $loader->addClassMap($classMap);
    }
}
```

autoload_static.php

- プロジェクトの依存パッケージと、composer installのオプションに依存
- デフォルトではComposer.jsonの定義通りのローディングを生成
  - --optimize-autoloader オプションでクラスマップを強制で生成する

autoload_static.php (PSR-4のみ)

```php
<?php // autoload_static.php @generated by Composer
namespace Composer\Autoload;
class ComposerStaticInitbed8ae7d525347237f9cdad08c8930b3 {
    public static $prefixLengthsPsr4 = array (
        'T' =>
        array (
            'Teto\\' => 5,
        ),
    );
    public static $prefixDirsPsr4 = array (
        'Teto\\' =>
        array (
            0 => __DIR__ . '/../..' . '/src',
        ),
    );
    public static function getInitializer(ClassLoader $loader) {
        return \Closure::bind(function () use ($loader) {
            $loader->prefixLengthsPsr4 = ComposerStaticInitbed8ae7d525347237f9cdad08c8930b3::$prefixLengthsPsr4;
            $loader->prefixDirsPsr4 = ComposerStaticInitbed8ae7d525347237f9cdad08c8930b3::$prefixDirsPsr4;

        }, null, ClassLoader::class);
    }
}
```

autoload_static.php (optimized)

```
    public static $classMap = array (
        'Teto\\Object\\Helper' => __DIR__ . '/../..' . '/src/Object/Helper.php',
        'Teto\\Object\\MethodAlias' => __DIR__ . '/../..' . '/src/Object/MethodAlias.php',
        'Teto\\Object\\ObjectArray' => __DIR__ . '/../..' . '/src/Object/ObjectArray.php',
        'Teto\\Object\\PrivateGetter' => __DIR__ . '/../..' . '/src/Object/PrivateGetter.php',
        'Teto\\Object\\PrivateStrictGetter' => __DIR__ . '/../..' . '/src/Object/
PrivateStrictGetter.php',
        'Teto\\Object\\PropertyLikeMethod' => __DIR__ . '/../..' . '/src/Object/
PropertyLikeMethod.php',
        'Teto\\Object\\ReadOnly' => __DIR__ . '/../..' . '/src/Object/ReadOnly.php',
        'Teto\\Object\\ToArrayInterface' => __DIR__ . '/../..' . '/src/Object/
ToArrayInterface.php',
        'Teto\\Object\\TypeAssert' => __DIR__ . '/../..' . '/src/Object/TypeAssert.php',
        'Teto\\Object\\TypeDefinition' => __DIR__ . '/../..' . '/src/Object/
TypeDefinition.php',
        'Teto\\Object\\TypedProperty' => __DIR__ . '/../..' . '/src/Object/TypedProperty.php',
    );
```

autoload_static.phpの大事なとこ

- Closure::bind()でスコープ束縛して、
  メソッド呼び出しをけちりながら
  privateメンバーにセットしてる

```
return \Closure::bind(function () use ($loader) {
    $loader->prefixLengthsPsr4 = ComposerStaticInitbedRy::$prefixLengthsPsr4;
    $loader->prefixDirsPsr4 = ComposerStaticInitbedRy::$prefixDirsPsr4;
    $loader->classMap = ComposerStaticInitbedRy::$classMap;

}, null, ClassLoader::class);
```

ClassLoader.php

- クラスローダーの実装本体
- いままで設定された情報をもとにクラス名の解決・ファイルロードをする
- 動的に変動する部分は分離されてるので、バイトコードがキャッシュされても特に問題なさそうに作ってるっぽい

ClassLoader.php

- spl_autoload_register()で呼ばれる
- クラスマップにあったらそれを読む
- なかったらpsr-4, psr-2を読む
- PEARに依存するなどの場合は include_pathを読む

これによって
得られる知見

Composerは、いろんなPHP
のバージョンとか実行環境
で動くように書かれてる

実行時にcomposer.jsonとか
composer.lockは使ってなく
て、静的なPHPスクリプト
ファイルが書き出される

サーバーでcomposer install
の必要はなく、手元でインス
トールしてvendorディレクト
リごと転送してやればいい

よくある誤解

「レンタルサーバーの最安プランでPHP動かしてるからComposer入らないんだ…」といったことはない

本筋から外れた話

Autoloader Optimization

https://getcomposer.org/doc/articles/autoloader-optimization.md

Level 1
本番運用時は
--optimize-autoloaderが基本

……そのほかは
効果あるのかな？
（私は使ってないです）

予告

PHP AdventCalendar 2017では
Composerとか全然考慮されてない
(zipで配布されてる)SDKをむりやり
Composerで管理する話を書きます