PSRを実装してみる

Let's implement PSR(PHP Standards Recommendations)

pixiv

2019-01-26 PHP Conference Sendai

Sendai, Miyagi, Japan **#phpconsen**

お前誰よ

- うさみけんた（@tadsan） / Zonu.EXE
  - GitHub/Packagistでは id: zonuexe
- ピクシブ株式会社 pixiv運営本部
- Emacs Lisper, PHPer
  - Emacs PHP Modeのメンテナ引き継ぎました
  - 好きなリスプはEmacs Lispです
- Qiitaに記事を書いたり変なコメントしてるよ

pixiv

お絵描きがもっと楽しくなる場所を創る

プレスリリース

Friends of Emacs-PHP development

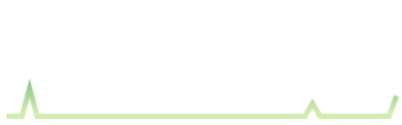
Join us!

<?php

Find a repository... Type: All Language: All Customize pinned repositories New

php-runtime.el

PHP-Emacs bridge, call PHP function from Emacs



php emacs melpa

Emacs Lisp ★ 2 1 GPL-3.0 Updated 2 days ago

phpactor.el

Interface to Phpactor (an intelligent code-completion and refactoring tool for PHP)

Emacs Lisp ★ 8 4 1 issue needs help Updated 3 days ago

php-mode

A PHP mode for GNU Emacs

Top languages

Emacs Lisp PHP

Most used topics

Manage

emacs melpa php major-mode

People

5 >

Invite someone

今回の目的

pixiv

PSRって何のためにあるの

pixiv

強い人たちと話すと
PSRの不満についての
愚痴を聞くけれども、
なるほどわからん

pixiv

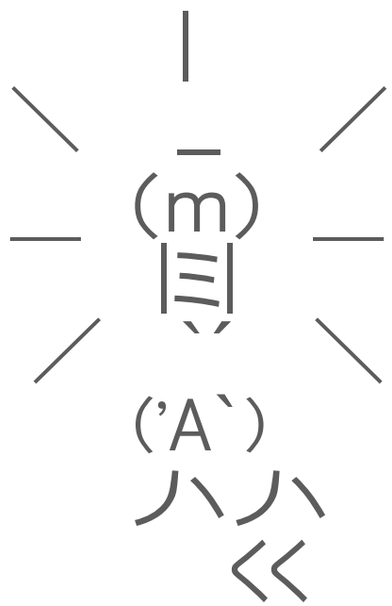
わかりたい

pixiv

それを明らかにしたい
と思い、われわれ取材
班は仙台へと旅立った

でも、英語も
仕様書も読むのも
苦手だお…

pixiv



そうだ、実装しながら理解しよう

pixiv

実装

pixiv

仕様を雑に読む→
よくわからないので
ぐぐる→
既存実装を読んでみる→
雑に実装してみる→
（以下ループ）

pixiv

背景

pixiv

PHPのコンポーネントを
作って提供してる組織は
意外とたくさんある

pixiv

Documentation

Zend Framework 3

Zend Framework is a collection of **60+ packages** for professional PHP development. Each package is available on GitHub and can be installed via Composer.

Tutorials

We provide tutorials for getting started with zend-mvc, understanding advanced topics of many components, and migration from version 2 to version 3.

Read tutorials

zend-authentication

zend-barcode

You are browsing the **Symfony 4 documentation**, which changes significantly from Symfony 3.x. If your app doesn't use Symfony 4 yet, browse the Symfony 3.4 documentation.

Symfony Components

The Components implement common features needed to develop websites. They are the foundation of the Symfony full-stack framework, but they can also be used standalone even if you don't use the framework as they don't have any mandatory dependencies.

Download as PDF

The League of Extraordinary Packages is a group of developers who have banded together to build solid, well tested PHP packages using modern coding standards.

★ ★ ★

We comply to the standards of the PHP-FIG.
We adhere to the best-practices put forward by PHP The "Right" Way.
We distribute code via Packagist and Composer.

Hoa is a modular, extensible and structured set of PHP libraries. Moreover, Hoa aims at being a bridge between industrial and research worlds.

Compose and extend libraries to create applications or your own libraries!

- *Acl*
- Bench
- *Cli*
- Compiler
- *Consistency*
- Console
- Database
- *Devtools*
- Dispatcher
- Dns
- *Event*
- Eventsource
- *Exception*
- Fastcgi
- File
- *Graph*
- Http
- *Irc*
- *Iterator*
- *Json*
- *Locale*
- *Mail*
- *Math*
- Mime
- *Praspel*
- Promise
- *Protocol*
- Realdom
- *Regex*
- Registry
- Router
- Ruler
- *Session*
- *Socket*
- *Stream*
- Stringbuffer
- Test
- Ustring
- View
- *Visitor*
- Websocket
- Worker
- Xml
- Xyl
- Zformat
- Zombie

They trust us

They support us

Category:PHP libraries

Help

Translate this page

Other languages:	Deutsch · English · Hawai'i · Nederlands · Scots · català · español · français · italiano · polski · português · português do Brasil · sicilianu · suomi · سنڌي · 中文 · 日本語

This category contains PHP libraries that do not add functionality to a wiki like most extensions do.

Pages in category "PHP libraries"

The following 33 pages are in this category, out of 33 total.

- Manual:External libraries

A

- Extension:Ask
- At-ease

B

- Base convert

C

- CDB
- CLDRPluralRuleParser
- Css-sanitizer
- CSSJanus

D

- Extension:DataTypes
- Extension:DataValues
- Extension:DataValuesCommon
- IPSet

O

- Oauthclient-php
- ObjectFactory

P

- Phan-taint-check-plugin
- Php-session-serializer
- Purtle

R

- RelPath
- RemexHtml
- RunningStat

S

- ScopedCallback
- Extension:Serialization

T

Main page
Get MediaWiki
Get extensions
Tech blog
Contribute

Support
User help
FAQ
Technical manual
Support desk
Communication

Development
Bug tracker
Code repository
Code docs
Statistics
Wikimedia technology
Wikimedia audiences

MediaWiki.org
Community portal
Recent changes
Translate content
Random page
Current issues
Sandbox

Print/export
Create a book

Laravel PHP Components

Report abuse

The components that make up the Laravel PHP framework.

http://laravel.com

Repositories 33 People 2 Projects 0

Find a repository... Type: All Language: All

translation

[READ ONLY] Subtree split of the Illuminate Translation component (see laravel/framework)

PHP ★ 31 6 Updated 7 minutes ago

support

[READ ONLY] Subtree split of the Illuminate Support component (see laravel/framework)

PHP ★ 285 101 Updated 7 minutes ago

pagination

[READ ONLY] Subtree split of the Illuminate Pagination component (see laravel/framework)

PHP ★ 51 20 Updated 7 minutes ago

mail

[READ ONLY] Subtree split of the Illuminate Mail component (see laravel/framework)

Top languages

PHP

People 2 >

daylerees
Dayle Rees

taylorotwell
Taylor Otwell

たくさんあるのはいいが
大同小異、同じような
ものが乱立してると困る

pixiv

使い回せるものが
あるなら折角だから
相互運用したい

pixiv

そのような現状でPSRは
何を提供しているのか、
実装しながら見てみよう

pixiv

おしながき

pixiv

PHP-FIGについて
PSR-0 PSR-3
PSR-1 PSR-6
PSR-2 PSR-13
そのほかのPSR

pixiv

ここから
駆け足になるので
最初にまとめ

pixiv

PSRにナンバリング
されてるからと言って
高品質で優先すべき
対象とは限らない

pixiv

PHP-FIG

フレームワーク相互運用グループ

pixiv

PHP-FIG

Home Blog PSRs Personnel Bylaws FAQs Get Involved

Moving PHP forward through collaboration and standards.

Welcome to the PHP Framework Interop Group! We're a group of established PHP projects whose goal is to talk about commonalities between our projects and find ways we can work better together.

View recommendations (PSRs) Frequently asked questions

PSRを策定する団体

pixiv

フレームワーク/CMS/そのほかPHP関連開発者の寄合所帯

pixiv

さっき挙げた団体の全てが加盟してるわけではない

pixiv

抜けちゃったと
ころもしろしろ
ある

pixiv

Zend,CakePHP,
Yii, Drupal, Slim
など多数

pixiv

Guzzle, Doctrine,
Symfony, Laravel,
The PHP Leagueは
脱退済み

pixiv

脱退したからと言って
PSRを尊重してないわけではない

pixiv

PSR

PHP標準勧告

pixiv

「勧告」と名前はついてるがPHP本体とは関係なく、有志による規格

pixiv

グループの背景にある考えは、代表者たちがプロジェクト間の共通点について話し合うことで協働する方法を探ることです。

グループの議論の主な観客は参加プロジェクト同士ですが、そのほかのPHPコミュニティも注目していることはよく承知しています。ほかの人々がPSRの成果物を採用することは歓迎しますが、それは目的ではありません。

投票メンバーは標準を遵守しなければいけませんか？

> いいえ。PHP-FIGの投票メンバーになることで、すべての、あるいは任意のPSRを実装することを強制されることはありません。プロジェクトはしかるべき時にアップグレードする際に後方互換性を考慮する必要があります。ほとんどのプロジェクトで最終的に採用されると想定されますが、必須ではありません。

勧告される対象は我々
下々のPHPer各位
ではなくPHP-FIGに
任意で参加した団体

pixiv

勧告された参加団体が
PSRと準拠することも
また任意

pixiv

脱退したはずの団体が
PSRを準拠することも
また任意

pixiv

PSR自体が、ベストプラクティスを標榜してたりもするけど…？

pixiv

それはそれ

これはこれ

pixiv

AUTOLOADING

Autoloaders remove the complexity of including files by mapping namespaces to file system paths.

PSR-4: Improved Autoloading

```php
<?php

use Vendor\Package\ClassName;

$object = new ClassName();
```

INTERFACES

Interfaces simplify the sharing of code between projects by following expected contracts.

PSR-3: Logger Interface
PSR-6: Caching Interface
PSR-11: Container Interface
PSR-13: Hypermedia Links
PSR-16: Simple Cache

```php
<?php

namespace Psr\Log;

/**
 * Describes a logger instance
 */
interface LoggerInterface
{
```

HTTP

Interoperable standards and interfaces to have an agnostic approach to handling HTTP requests and responses, both on client and server side.

PSR-7: HTTP Message Interfaces
PSR-15: HTTP Handlers
PSR-17: HTTP Factories
PSR-18: HTTP Client

```php
<?php

namespace Psr\Http\Message;

/**
 * Representation of an outgoing, client-side request.
 */
interface RequestInterface extends MessageInterface
{
```

CODING STYLES

Standardized formatting reduces the cognitive friction when reading code from other authors.

PSR-1: Basic Coding Standard
PSR-2: Coding Style Guide

```php
<?php

namespace Vendor\Package;

class ClassName
{
    public function fooBarBaz($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

PSR-0

Autoloading Standard

pixiv

失効済み！！！

~~PSR-0~~

~~Autoloading Standard~~

pixiv

オートロードに適合したクラスの命名規則とファイルの配置

pixiv

前提

- PHP 5.0からクラスのオートローディング(遅延ロード)に対応
  - =必要あるまで読み込まない
  - それ以前はinclude_pathからinclude/require(_once)が主流

ざっくりPSR-0

- クラス名の構造の規則

  \<Vendor Name>\(<Namespace>\)*<Class Name>

- _ 区切りの擬似名前空間も許容

  \<Vendor Name>_(<Namespace>_)*<Class Name>

ディレクトリ配置

```
.
└── Hoge
    └── Fuga
        └── Piyo.php
```

```php
class Hoge_Fuga_Piyo {}

// または

namespace Hoge\Fuga;
class Piyo{}
```

実装依存になる部分

- ファイルシステム依存
  - 大文字小文字の差異
- Unicode NFD問題

曖昧な部分

- 「ベンダー名」って何…？
  - 自由に決められるので、ただの紳士協定に過ぎない
- トップレベルのクラス定義を戒める効果はある(?)

実装方式

- A: クラスマップ方式
  - 実行前にディレクトリ構造をスキャンして対応表を作成
- B: 逐次確認
  - 実行時にファイルシステムを検査

PSR-0準拠クラスローダー関数

```
function add_directory(string $base_dir): void {
  $directory = rtrim($base_dir, '/\\');

  spl_autoload_register(function ($fqcn) use ($base_dir) {
    $namespaces = explode('\\', $fqcn);
    $class = array_pop($namespaces);
    $class_file = strtr($class, ['_' => '/']) .
'.php';
    $ns_dir = implode('/', $namespaces);
    $file_path = implode(‘/',
array_filter([$base_dir, $ns_dir, $class_file], 'strlen'));

    if (is_file($file_path)) {
      require_once $file_path;
    }
  });
}
```

現状

- 以前はライブラリごとに
  クラスローダーを提供してた
  - Requests::register_autoloader() など
- 最近はComposerがオートロード機能を提供してくれるので
  composer.jsonに書くだけ

重要

- PSR-0はPSR-4で代替されたので廃止済み

Composer以前

- PEARでインストールする
- zipをダウンロードして、
  そのディレクトリを
  include_pathに追加する
- 同じホスト上で動作する別のPHP
  プロジェクトと共用されることも

Composer以後

- 依存パッケージはプロジェクト内のサブディレクトリにインストール
- 同じホスト上で動作する別のプロジェクトと混在することは原則ない

PSR-1

Basic Coding Standard

pixiv

基本的な(＝相互運用のため重要度の高い)コーディング規約

pixiv

要件

- 名前空間・クラス名は
  PSR-4/0に適合するように
- 文字コードはUTF-8(BOM無)
- クラス定義で副作用を
  起こしてはいけない

命名規則

- クラス名はUpperCamelCase
- 定数名はUPPER_SNAKE_CASE
- メソッド名はlowerCamelCase
- 変数・プロパティは規定しない

理由

- 「あるクラスがロードされた副作用として別のファイルが読み込まれる」のような挙動はわかりにくい
- 最低限のもの以外はinclude/requireせずに遅延ロードに任せるのが無難

こういうコードはだめ

```php
<?php /** Foo/Bar.php */

// 別のファイルを読み込もうとする
include_once __DIR__ . "/foo.php";

// エラーレベルを変更
error_reporting(0);

class Foo
{
    // ...
}
```

「実装します」
といった感じの
ものではないので
次

pixiv

PSR-2

Coding Style Guide

pixiv

コーディング スタイルの ガイドライン

pixiv

要件

- プロジェクト内で表記を一貫して意識のずれを低減する
- PSR-1に準拠することが前提
- さまざまなプロジェクトのコーディングスタイルを調査した上で策定

ホワイトスペース・インデント

- インデントは4スペースが基本
- 単項演算子以外の演算子や()の前後にはスペースを入れる
- 改行コードはLF
- 一行あたりの文字数は80文字を推奨、120文字以上は警告

ざっくり

```
class Foo ← 改行する
{
    public function bar($xs)  ← 改行する
    {
        foreach ($xs as $x) { ← 改行しない
            $f = function ($a) use ($x) {  ← 改行しない
                // ...
            };
        }
    }
}
```

{} と改行位置の癖

- 宣言文(class, function, namespaceなど)の開始行と { の間は改行する
- それ以外の文(if, foreachなど)は改行せず、開始行に { を書く
- 無名関数は文ではなく式なので改行せず、開始行に { を書く

重要なこと

pixiv

> スタイルルールは、様々なプロジェクトの共通内容から生み出されています。 様々な作者が複数プロジェクトを横断して協力しあうことで、全てのプロジェクトで有用な**ガイドライン策定の助け**となります。 従って、このガイド本来の利点は、ルール自体にはなくルールを共有することにあります。

http://www.infiniteloop.co.jp/docs/psr/psr-2-coding-style-guide.html

> 7. その他 本スタイルガイドでは、意図的に省略しているスタイルやプラクティスが多くあります。 例えば下記のような幾つかについては、ここでは明記していません。
>
> （中略）
>
> なお、本スタイルガイドは将来的に様々なスタイルやプラクティスの登場に応じて改定・拡張をできるものとします。

http://www.infiniteloop.co.jp/docs/psr/psr-2-coding-style-guide.html

PSR-2は
コーディングスタイル策定の
ガイドライン

pixiv

PSR-2をカスタマイズする

- 定義文であっても { を同じ行で書くとか
- インデントを8タブにするとか
- 改行コードをCR+LFにするとか

策定したガイドラインを守らせる

- PHP-CS-Fixerを導入する
- EditorConfigを導入する
- PhpStormの設定ファイルを共有
- StyleCI https://styleci.io

PHP-CS-Fixerの設定例

```
<?php

$finder = PhpCsFixer\Finder::create()
    ->in(__DIR__ . '/src')
    ->in(__DIR__ . '/tests');

return PhpCsFixer\Config::create()
    ->setRules([
        '@PSR1' => true,
        '@PSR2' => true,
        'braces' => [
            'position_after_functions_and_oop_constructs' => 'same',
        ],
    ])
    ->setFinder($finder)
    ->setIndent("\t")
    ->setLineEnding("\n");
```

定義開始行と { を同じ行に

タブでインデント

.editorconfig

```
# EditorConfig is awesome: http://EditorConfig.org

# top-most EditorConfig file
root = true

# Unix-style newlines with a newline ending every file
[*]
end_of_line = lf
insert_final_newline = true
charset = utf-8

# PHP - http://php.net/
[*.php]
indent_style = tab
indent_size = 8
```

タブでインデント
幅8で表示

PhpStormの設定ファイルを共有

- PhpStormが生成する.ideaは一部ファイルを除き、共有した方がいいらしい
  - github/gitignore: JetBrains.gitignore
- 特に.idea/codeStyles/Project.xmlがコーディングスタイル

大事なこと

- コーディングスタイルは人間がレビューであれこれと指図したりするよりも、機械的に変換したほうがギクシャクせずに済む
- 「人間が気をつける」は破綻する
- 「自然にそうなる」状態が理想

PSR-2の現状

- 初版が2012年6月なので、PHP7の時代には不足気味
  - 可変長引数、無名クラスとか
- 現在、後継となるPSR-12 Extended Coding Style Guideが策定作業中

PSR-3

Logger Interface

pixiv

Syslog風のLoggerインターフェイス

pixiv

要件

- Syslogで定義された8種類のレベルごとにログを記録
  - 緊急度が高い順から
    - Emergency, Alert, Critical, Error, Warning, Notice, Informational, Debug

実装

- psr/logパッケージに含まれるLoggerTrait/AbstractLoggerを利用して最小限で実装可能
- 今回実装したPSRのなかでは機能がわかりやすく実装が簡単なので初心者にオススメ

雑感

- 普通はMonologを使えばいいので代替実装に変更するメリットが全然感じられない
- Monologのハンドラーを実装した方が拡張しやすいので、そちらの方がオススメ

さらに雑感

- WebアプリからSyslogの8種類のレベルはミスマッチ感がある
- 正直、Error, Debug, Infoの3レベルくらいにまとめた方が扱いやすいのでは

ヘルパー関数

```php
<?php declare(strict_types=1);

namespace ZoPsr\Log3;

use Psr\Log\LogLevel;

function to_num(string $level): int {
  return [
    LogLevel::EMERGENCY => 0,
    LogLevel::ALERT     => 1,
    LogLevel::CRITICAL  => 2,
    LogLevel::ERROR     => 3,
    LogLevel::WARNING   => 4,
    LogLevel::NOTICE    => 5,
    LogLevel::INFO      => 6,
    LogLevel::DEBUG     => 7,
  ][$level];
}
```

行区切りJSON出力するロガー

```
final class FileLogger implements LoggerInterface {
   use LoggerTrait;

   /** @var resource */
   private $_fp;
   /** @var string */
   private $_min_level;

   /**
    * @param resource $fp
    * @param ?string $min_level
    */
   public function __construct($fp, $min_level = null) {
      $this->_fp = $fp;
      $this->_min_level = $min_level ?? LogLevel::WARNING;
   }

   public function log($level, $message, array $context = []) {
      if (to_num($level) <= to_num($this->_min_level)) {
         $log = json_encode(['level' => $level, 'message' => $message, 'context' => $context],
            JSON_UNESCAPED_SLASHES) ?: '{"json_encode_error":true}}';
         fputs($this->_fp, $log);
         fputs($this->_fp, "\n");
```

PSR-4

Autoloader

pixiv

オートロードに適合したクラスの命名規則とファイルの配置

pixiv

PSR-00の後釜

pixiv

オートロードに適合したクラスの命名規則とファイルの配置

pixiv

PSR-0との差異

- _を使った擬似名前空間は許容されなくなった
- PSR-0は名前空間に対応するディレクトリを用意しなければならなかったが、PSR-4ではディレクトリを浅くできる

PSR-0との差異

- prefixとbase directory という概念が導入された
  - \A\B\C\D というクラス
    - PSR-0: src/A/B/C/D.php
    - PSR-4: src/D.php
- Composerを使ってたらお馴染

ディレクトリ配置

```
.
└── src
    └── Fuga
        └── Piyo.php
```

```
.
└── src
    └── Piyo.php
```

```php
<?php

namespace Hoge\Fuga;

class Piyo{}
```

クラスローダの実装(1)

```
final class ClassLoader {
  /** @var array<string,string> */
  private $directories = [];

  public function add(string $prefix, string $directory): void {
    $this->directories[$prefix] = $directory;
  }

  public function register(bool $prepend = false) {
    spl_autoload_register([$this, 'autoload'], true, $prepend);
  }

  public function autoload(string $class): void {
    $path = $this->_resolve($class);

    if ($path !== null) {
      require_once $path;
    }
  }
```

クラスローダの実装(2)

```
private function _resolve(string $class): ?string {
   foreach ($this->directories as $prefix => $dir) {
      if (strpos($class, $prefix) !== 0) {
         continue;
      }

      $path = $this->_makePath($class, $prefix, $dir);

      if (is_file($path)) {
         return $path;
      }
   }

   return null;
}

private function _makePath($class, $prefix, $directory): string {
   $relative = substr($class, strlen($prefix));
   $path = strtr($relative, '\\', '/') . '.php';

   return $directory . DIRECTORY_SEPARATOR . $path;
}
}
```

ディレクトリの追加

```
<?php

use ZoPsr\Autoload4\ClassLoader;
use function ZoPsr\TestHelper\capture;

require_once __DIR__ . '/../../vendor/autoload.php';

$loader = new ClassLoader;
$loader->register(false);

$loader->add('Foo\\', __DIR__ . '/1');
$loader->add('Hoge\\', __DIR__ . '/2');
```

PSR-4を実装する意味はあるか

- クラス名の規則として重要
- クラスローダは自作せずにComposerを使った方がパフォーマンスの最適化できて便利
- PHPのクラスロードの仕組みを学ぶ上では重要

PSR-13

Link definition interfaces

pixiv

要件

- ハイパーメディアの「リンク」を抽象化したクラス
  - HTMLのとか<link>とか
  - HTTPのLinkヘッダとか(RFC5988/8288 WebLinking)
- ……なんだけど

仕様

- 基本はPSR-7と同じ
  イミュータブルなオブジェクト
- 関連するRFCがやたらある
  - 特にRFC6570(URI Template)、
    それ混ぜる必要あったか…？
    - 実装として混ざってるわけではないが、isTemplated()という状態がある

URI Template

- RFC6570
  - URIに変数展開するための
    テンプレート言語
- これ自体は需要ある規格だが、
  Linkでテンプレートありなしの
  hrefを混在させて嬉しいことは
  ……ある？

既存実装

- crell/htmlmodel
  - PSR-13提案者の概念実証
- php-fig/link-util
  - PHP-FIGによる実装
  - Linkクラスも実装されてる
- あのSymfonyも独自実装してない

仕様への文句

- isTemplated()
  - 必死にユースケースを考えてみたけど、積極的にこの分岐をしたいケースが一切見当たらない
- Symfony\WebLinkでも事実上黙殺されてるように見える

実装への文句

- isTemplated()
  - hrefに { か } のどちらかが含まれてたらURI Template文字列って判定してる
- すごく怪しい

URIの仕様

- https://example.com/{ のようなURIは完全に合法
- {と}はreservedな文字でもunreservedな文字でもないので、パーセントエンコードされてるとは限らない

とてもバグりそうな雰囲気が

- 文字列を場当たり的な判定して実行時に分岐するのは、どう考えてもアンチパターン
- せめてTemplatedLinkInterfaceのような別の型で表現するなら気持ちはわかる(けど嬉しくはない)

そのほかのPSR

pixiv

ここから実装が間に合わなかったPSRについて雑に話します

pixiv

PSR-7 message interface

- HTTPのリクエスト／レスポンスを抽象化するクラス
- GuzzleのようなHTTPクライアントライブラリには好適
- しかしHTTPミドルウェアの媒介としては辛くないですか

PSR-7とわたし

- Guzzle使ったらPHPでめちゃんこ簡単にHTTPプロキシ鯖を実装できたので好き
- PHPでプロキシサーバーを作る
- PHPのプロキシサーバーを説明する

PSR-6 Caching Interface

- 実は今回は実装してみたけど、特に感情がない
- 業務ではRedisなどのKVSをラップした独自の抽象クラスを使ってるけど、移行する気は特に湧かなかった

PSR-5 PHPDoc (Draft)

- PSRの番号が若いのに長年マージされてない代表格
- 放棄されかけたがAPIドキュメント自動生成よりもIDEや静的解析のデファクト標準として重要なので議論復活した

PSR-5とわたし

- WEB+DB PRESS Vol.87に書いた（総集編に収録されてるから買って）
- 2018年のPHPDoc事情とPSR-5
- PhpStorm 2018.3でPSR-5ジェネリクス記法はサポートされない

PSR-5と静的解析

- コード上の記述とPHPDocのコメントを手がかりにして、怪しい部分を高精度で検出してくれる
- Phan, PHPStan, Psalm, PhpStormなどのツールがある
- 今回の実装でも助けられました

PSR-8 Mutually Assured Hug

- 2014年4月1日に提案された
- 「この標準は、オブジェクトが互いの感謝と支持を表明するための、一般的な方法を確立します」
- こんなのにPSRの番号の帯域を降ってよかったの…？
- PSR-8 Hug（日本語訳）