いまさら！php お作法入門

Modern...? PHP manners guide

pixiv SUMMER BOOT CAMP 2017

東京都渋谷区千駄ヶ谷 #pixiv_BOOTCAMP

pixiv サービス開発インターンシップ 2017 夏

エントリーする

pixivとPawooを支える大規模サービス開発

ABOUT THE INTERNSHIP

2500万人のユーザーを抱える **pixiv** と、リアルタイム通信が行われる

マストドンインスタンス **Pawoo** の2つの大規模サービスを舞台とし、

pixiv サービス開発インターンシップ 2017 夏

エントリーする

この夏、ピクシブで

圧倒的猛者になる。

講義資料です

エントリー期間

7.3(月) - 7.31(月)

開催期間

9.4(月) - 9.8(金)

pixivとPawooを支える大規模サービス開発

ABOUT THE INTERNSHIP

2500万人のユーザーを抱える **pixiv** と、リアルタイム通信が行われる

マストドンインスタンス **Pawoo** の2つの大規模サービスを舞台とし、

このスライドは

Webで公開します

（メモはとらなくても良いです、の意味）

pixiv

お前誰よ

- うさみけんた（@tadsan）/ Zonu.EXE
  - GitHub/Packagistでは id: zonuexe
- 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

pixiv

おぼえて

ほしいこと

コードは動けばええんや

反論？

最高のコードは動くコード

- コードの美しさはユーザーに見えない
- 汚ないコードでも、動けばユーザーにとっては最大の価値を生み出す

最高のコードは動くコード

- コードの美しさはユーザーに見えない
- 汚ないコードでも、動けばユーザーにとっては最大の価値を生み出す

全ての物事には一長一短がある

- 読めないコードはメンテできない
  - 機能拡張しにくい
  - ある日とつぜん動かなくなる
  - 仕様変更に追従困難

動けばの前提が崩れた

コードは価値が大暴落

まとめると

- ソースコードの美醜は現在の
  プログラムの価値とは関係ない
- ただし美しくないコードは
  将来の可能性を閉ざしかねない

そのために「読める（リーダブル）」
コードの追求と
お作法の共有が必要

今回は大仰な設計の話で
はなく、単にコーディン
グルールの共有をします

PSR?

その話はあとで

<?phpからはじめよ

- ファイル先頭は必ず<?phpを書きます
- 必要ならnamespaceとuseを書きます
- include_once 必要なければ書きません
- そのあとクラス定義(だけ)を書きます
- ファイル最後に ?> は書きません

namespaceは必要か？

- namespaceを利用すべきかは、
  プロジェクトごとの事情に強く依存します
- pixiv-libでは原則として利用しません
  - テストコード(tests/)でだけ使ってる
- pixiv.git内のプロジェクト単位では、
  利用されることもあります

もしもnamespaceがなかったら

- User_Commonのような擬似名前空間
- namespaceとuseの解決は少し面倒
- 要はクラス名とファイル名の解決が
  できればオートローディングできます
  - User_Common → User/Common.php

require/include_onceは最低限だけ

- クラスローダーを登録することで、
  自動ローディングできる（遅延評価）
- 1ファイルにつき1クラスを定義する
- 関数定義などは基本的には
  初期化ファイルから読み込みます
  （composer.jsonのautoload.files）

?> を書かない理由

- 省略可能、書くメリットが皆無
- PHPは <?php ～ ?> の範囲外に書かれた文字列を出力します
- 不随意にコード中に変な文字が混入したとき、文法エラーにもならず出力されかねないリスクがあります

クラス定義

- クラスの概要を一行で要約すること
  - 一行で説明が不可能なら、クラスが持つ責任は明らかに多すぎます
- クラス継承は慎重に設計してください
  - 個人的な経験では継承よりも委譲やinterfaceの方がうまくいきます

PHPDocの構造

```
/**
 * 概要(summary, 要約) ↓ 下は空行
 *
 * 説明文(description)。クラスの説明を書きます。
 * ここには何行書いてもいいです。
 *
 * 説明不足よりも能弁を尊べ。
 *
 * @see https://zonuexe.github.io/phpDocumentor2-ja/references/phpdoc/basic-syntax.html
 */
final class Foo_BarModel implements ModelInterface
{
    // ...
```

クラスとPHPDoc

```php
<?php

/**
 * パラメータからURLを組み立てるメソッド群
 *
 * ## 規約
 *
 * - すべてのメソッドは配列引数 `$params` を受け取る
 * - 後から読み書きしやすくするために、メソッド定義はアルファベット順に並べておく
 * - プロトコルやドメイン名を含むurlを返すメソッド名には必ず先頭に `full` をつける
 * - それ以外のメソッド(fullで始まらないメソッド)は全て `/` から始まるパスを返す
 * - 同じドメイン内でのリンクには、ドメイン名がつかないメソッド(fullで始まらないメソッド)を使う
 * - メールや外部サイトに出力する場合、またはドメインをまたぐリンクの場合には、fullで始まるメソッド
 * - このクラスのメソッドはSmartyのmodifierからも利用される `{reverse_route page='wwwRankingReport
 * - 受け入れるGETパラメータはUtil_Assertを使ったチェックをする（省略可能なパラメータの有無はarray
 * - 既存のページに対応するメソッドを定義するときは、パラメータの順番を既存のものに合わせる
 * - オプショナルなパラメータだが組み合わせて指定する必要がある場合、そのパラメータを必須にした新し
 *
 * ## テスト
 *
 * メソッド定義のコメントに `@route\example` 形式のコメントを一行で書くとPHPUnitで実行されます
 *
 * ```
```

メソッド定義

- メソッドの概要を一行で要約し、
  引数と返り値の型をデザインします
- 一行で説明できないメソッドは
  その存在の軸が容易に揺らいでいく
- 複数の型の値を返すメソッドは
  機能が多すぎ、利用側が煩雑になる

メソッドとPHPDoc

```php
<?php

/**
 * ログ書き込み処理を分離する
 *
 * テスト時などに共有ディレクトリにログを書き込まないことが目的
 *
 * @package Log
 */
interface Log_WriterInterface
{
    /**
     * ログを記録する
     *
     * このメソッドは記録処理のみを分担するので、呼び出し側が責任を持って変換する
     *
     * @param  string $file   ログファイル名（path）
     * @param  string $string 書き込み文字列、典型的にはjson_encode済（改行コード含まず）
     * @return void
     */
    public function writeLogString($file, $string);
}
```

http://nico.ms/kn1774

はじめてのPHPDocと型

My first PHPDoc and Type

pixiv

2016-09-09 ピクシブ社内勉強会

これだけ覚えて帰ってね

タグ名	意味	例
@param	引数を定義	@param int $n1
@return	返り値を定義	@return int[]
@var	変数/プロパティを定義	@var int
@property	マジックプロパティを定義	@property int $id

PHPDocの型

- 「PHPの型」か「クラス名」を書く
- 基本的な型
  - string, int, bool, float, array, resource, null, true, false
- 擬似型
  - callable, void(返り値を返さない)

https://zonuexe.github.io/phpDocumentor2-ja/references/phpdoc/types.html

PHPDocの型

- 複合型
  - 「intまたは文字列」 `int|string`
- 値が並んだ配列
  - 「intが並んだ配列」 `int[]`

https://zonuexe.github.io/phpDocumentor2-ja/references/phpdoc/types.html

なんでPHPなのに
わざわざ型を書くの

バグるのと型書くの
どっちがええねん

PHPは動的プログラミング言語

- 静的型解析器(コンパイラ)は
  一般に実行前(コンパイル時)に
  変数や関数の型を特定する
- PHPのVM(インタプリタ)は、
  変数や関数の型は基本的に特定せず、
  実行時に動的型解析する

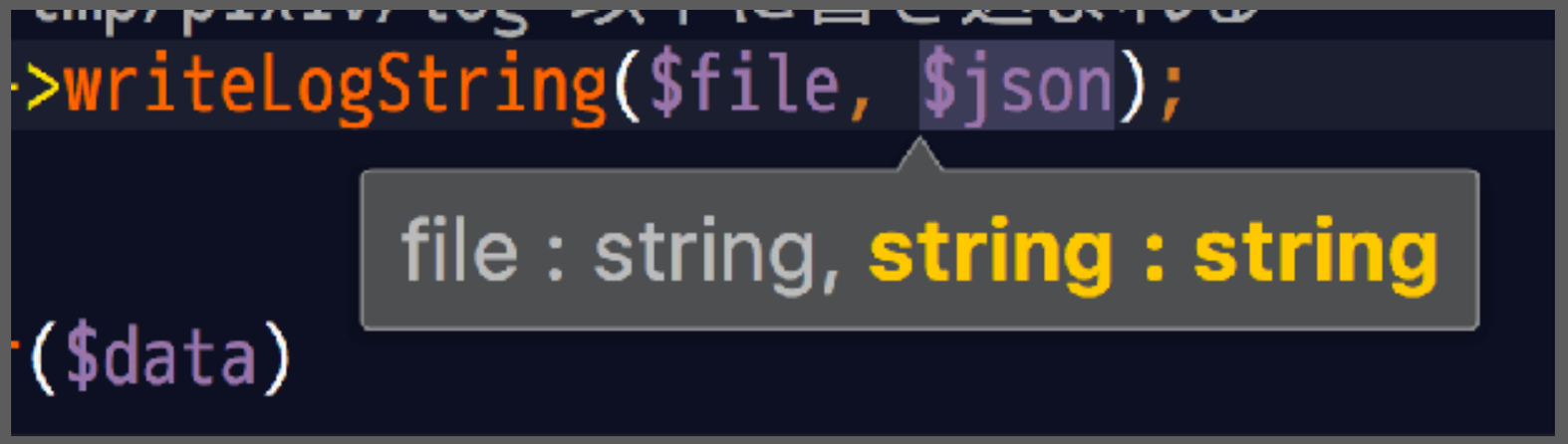
PHPの多くのバグは
実行時まで遅延する

要は動かさないと

わからない

動かさなくても
バグを検出できる
うれしみは大きい

引数一覧で型を表示するところ



型がめったくそなの叱ってくれ

```php
/** @var array $json */
$json = fopen( filename: 'php://stdin');
$logger->writeLogString($file, $json);
```

ループの話をしよう

PHPのfor文は原則禁止

```php
for ($i = 0; $i < count($novels); $i++) {
    $values[] = $novels[$i]->id();
}
```

PHPの配列は歯抜けになる

```php
$a = range( low: 1,  high: 5);
unset($a[2]);
for ($i = 0; $i < count($a); $i++) {
    var_dump($a[$i]);
}
```

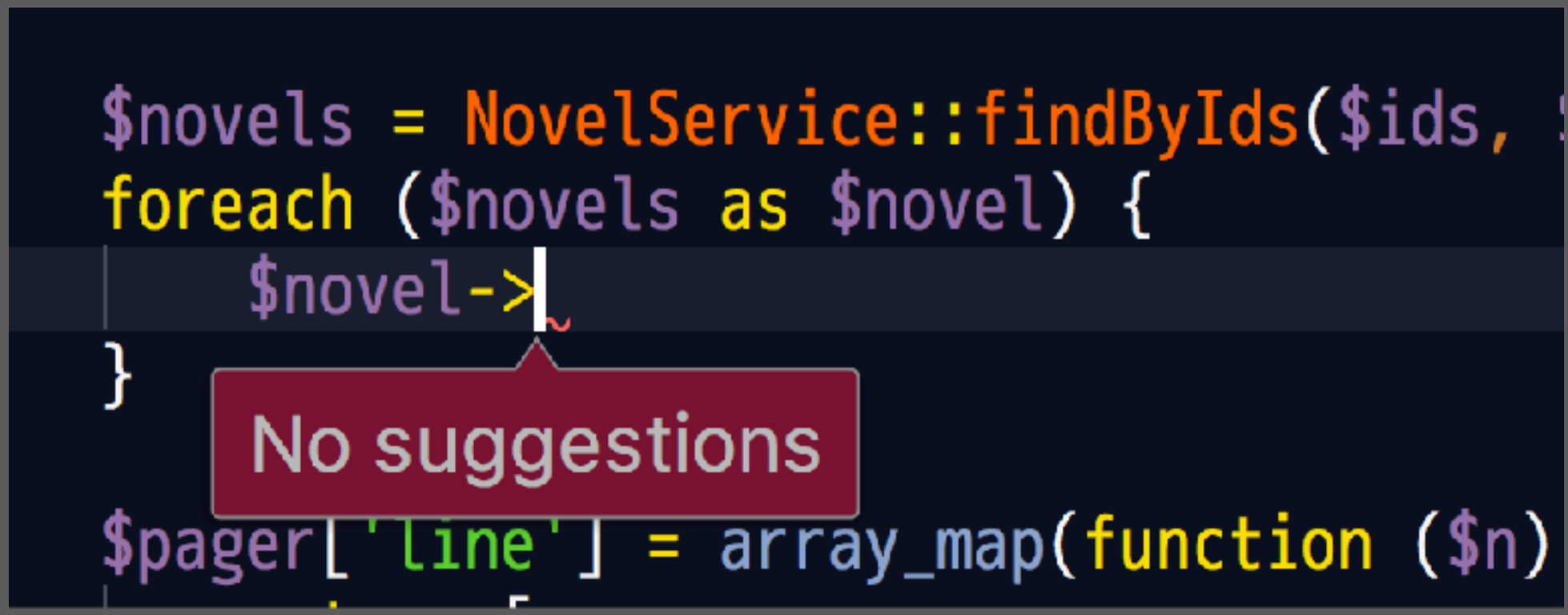
foreachなら大丈夫

```php
$a = range( low: 1,  high: 5);
unset($a[2]);
foreach ($a as $i => $v) {
    var_dump($v);
}
```

引数一覧で型を表示するところ

```
/**
 * @param  int[] $ids
 * @param  \MobileNovel\Session\SessionInter
 * @param  array $options
 * @return array
 */
public static function findByIds(array $ids,
{
```

ループの中で補完が利かない



きちんと型をつけ直す

```php
/**
 * @param  int[] $ids
 * @param  \MobileNovel\Session\Sessio
 * @param  array $options
 * @return NovelModel[]
 */
public static function findByIds(array
{
```

ループの中でも補完が利くよ

PSR

PHP Standards Recommendations

PHP標準勧告

PHP-FIG

PHP Framework Interop Group

PHPフレームワーク相互運用グループ

フレームワークやCMSを開発する組織の寄合

2017年8月現在のメンバー

CakePHP, Composer, concrete5, Contao Open Source CMS, Drupal, eZ Publish, Horde, IBM i Toolkit, Icicle, Jackalope, Joomla, Lithium, Magento, PEAR, Phalcon, Phing, phpBB, phpDocumentor, PHPixie, Pimcore, PPI Framework, PrestaShop, PyroCMS, ReactPHP, Revive Adserver, Sculpin, SilverStripe, Slim, Stash, Stormpath PHP SDK, SugarCRM, Symfony, Neos and Flow, Wikibase and Semantic MediaWiki, Yii framework, Zend Framework 2, Zikula

http://www.php-fig.org/members/

相互運用??

Zend Framework Components

Documentation for the ZF components.

Tutorials

Learn Zend Framework in-depth via our tutorials.

Components

Authentication

Authenticate users via a variety of adapters, and provide the authenticated identity to your application.

Barcode

Programmatically create and render barcodes as images or in PDFs.

Cache

Caching implementation with a variety of storage options, as well as codified caching strategies for

Captcha

Generate and validate CAPTCHAs using Figlets, images, ReCaptcha, and more.

Symfony Components

Symfony Components are a set of decoupled and reusable PHP libraries. They are becoming the standard foundation on which the best PHP applications are built on. You can use any of these components in your own applications independently from the Symfony Framework.

Installation

Use Composer to install any of the Symfony Components in your PHP project:

```
$ composer require symfony/asset
```

COPY

Component List

Component	Description	Resources
Asset	Manages URL generation and versioning of web assets such as CSS stylesheets, JavaScript files and image files.	Code Docs
BrowserKit	Simulates the behavior of a web browser.	Code Docs

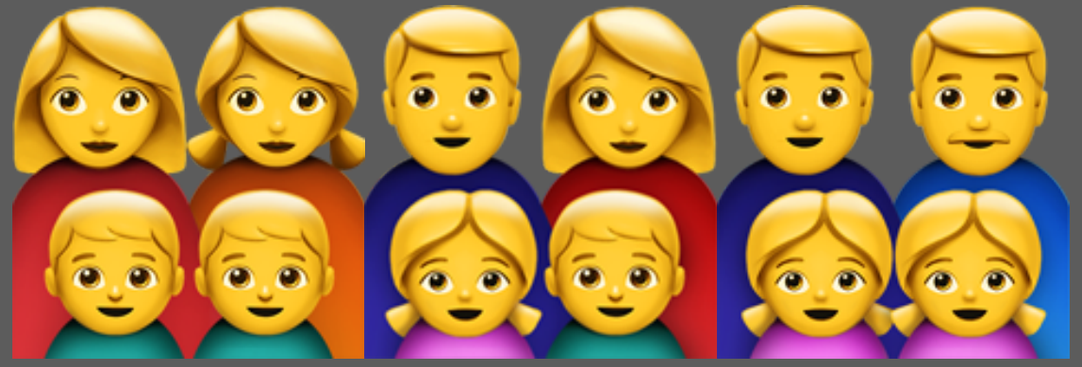
Namespaces

CakePHP 3.5

Namespaces summary

ばらばらに作ってたのを
仕様化して相互に
利用しやすくしようぜ！
……みたいな人類の夢

みんな仲良く



~~PSR-0 Autoloading Standard~~

- クラス名とファイルを一致させる規則
- ベンダー名を名前空間のトップにする
- \と_をDIRECTORY_SEPARATORに置換
- 2014年にPSR-4に置換されて廃止

PSR-1 Basic Coding Standard

- PHPコードは<?php … ?>または<?= … ?>
- 文字コードはUTF-8(BOMなし)
- クラスや定義のファイルは分けて、
  副作用のある処理とは分割する
- クラス名はStudlyCaps(UpperCamlCase)、
  メソッド名はlowerCamelCase、
  プロパティ名は規定しないけど一貫せよ

PSR-2 Coding Style Guide

- 具体的なコーディングスタイルガイド
- PSRに準拠したライブラリが必ず遵守しなければいけないルールではなく、プロジェクトごとにガイドラインを決めるための叩き台
- ↑ここ誤解してるひとが意外と多い

クラスや関数など定義の{は改行

```
final class Hoge implements Fizz
{
    function hoge($a, $b)
    {
        return $a + $b;
    }
}

function foo()
{
```

そのほかの文の{は改行しない

```
$is_hoge = hoge();
if ($is_hoge) {
    foo();
} elseif (piyo($a)) {
    bar();
}

foreach ($values as $i => $v) {
    $values[$i] = piyopiyo($v);
}
```

演算子などは、きっちり空白

```
$str = "もじれつ";
$str2 = $str . "！";


$n = 2;
$m = 5;
return foo($n, $m) * 2;
```

PSR-4 Autoloading Standard

- クラス名とファイルを一致させる規則
- ベンダー名を名前空間のトップにする
- \と_をDIRECTORY_SEPARATORに置換
- サブ名前空間とベースディレクトリの概念が導入

わしのPSRは17まで
あるぞ

ルールは破るためにある

所詮はフレームワーク開発者どもの間の規則

プライベートなプロジェクトがPSRに厳密に従っても得られるものは乏しい

プライベートなプロジェクトがPSRに厳密に従っても得られるものは乏しい

厳密に

都合のいいところだけ
持ってくればいい

規則とは理想的には
自分たちの足を撃ち
抜かないためにある

実態に即しない規則は改正すべし

よく聞く例

- PSR-2を~~魔改造~~カスタマイズ
  - インデントはスペースではなくタブ
  - { の改行位置が気に入らないので統一
- 要はコード内で統一感があれば良い

pixivの例

- PSR-1, PSR-2はそのまま採用
  - 行の長さの規制だけ緩めてる
- PSR-0, PSR-4は採用しない
  - 相互運用されないアプリケーションコードなどはvendorは厳密に意識しなくても別に問題が起こらない

クラスのオートロード

- PSR-0, PSR-4の規約通りに配置するだけで、Composerがいい感じにロードできる
- まあ文字列マッチしてるだけなので、ベンダー名とか省略しても実際動く
- そもそもローダーを自作してもいい

まとめ

まとめ

- (これに限らず)ルールやお作法は、われわれ人類が自爆しないための鎖
- ルールが作られた背景には事情があるので、それを共有していくのが重要
- 「どうしてこうなってるの」と感じたら質問していこう