

今日からできる安心型付け入門

Introduction of typing in PHP

2021-05-29 YouTube
PHPカンファレンス沖縄

！ 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, Rubyist
 - 2018年からEmacs PHP Modeのメンテナ
 - 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

本日のお題



fortee

今日からできる安心型付け入門

by うさみけんた / @tadsan



—今日からできる安心型付け入門 - PHPカンファレンス沖縄

<https://fortee.jp/phpcon-okinawa-2021/proposal/b90e3c04-ae04-4430-a28c-3f231e703c37>

レギュラートーク (30分)

今日からできる安心型付け入門

PHPカンファレンス沖縄2021

☆ 4



うさみけんた

 [tadsan](#)

PHP 7.0以降は引数にint, stringなどの型が書けるようになるなど静的型についての機能が充実しています。
またPhpStormのようなIDEやPHPStanのような実行しなくてもコードを解析・補完入力できるツールが揃ってきています。

今回のトークでは、できる限り型についての事前知識がなくても使えるよう以下の内容について扱います。

- プロジェクトに段階的に静的解析を導入するための考えかた
- PHPの型の特性とPHPDocでつける詳細な型
- 自分が使おうとしている「配列」が何者なのかを見極める
- 外部入力値を静的な値として扱うための方法
- 関数やオブジェクトの型を複雑しすぎないための考えかた

| アジェンダ

- PHPの型について
- なぜ型をつけるのか
- PHPの入出力型の特性

| 今回含まないもの

- 各静的解析ツールやIDE (PhpStorm)固有の型・機能
- メタプログラミングに対応した型付け

用語について

- 単に関数と呼ぶとき、メソッドとクロージャを含む
- 定義に直接記述する型を型宣言 (type declaration)、PHPDocを型注釈 (annotation) と呼びます

PHPの型について

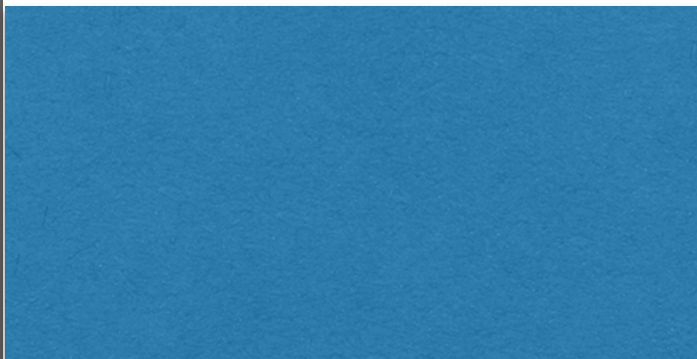
型の前に

プログラムには
依存があるという
ことを認める

依存とは

| 簡単なコードを考える

```
<?php
```



```
echo $a + $b;
```

| 簡単なコードを考える

```
<?php
```

```
$a = 1;
```

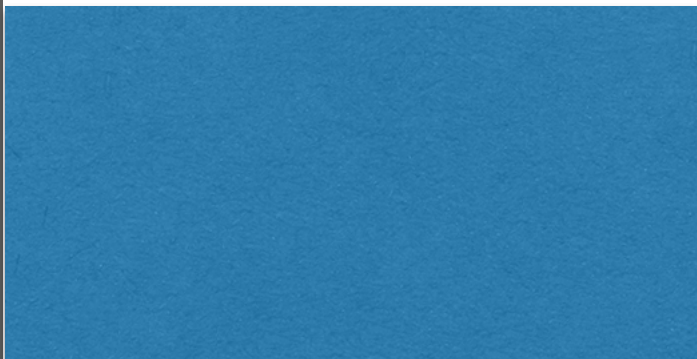
```
$b = 2;
```

```
echo $a + $b;
```

マスクされている
という状態を
コードに反映する

| 簡単なコードを考える

```
<?php
```



```
echo $a + $b;
```

| 簡単なコードを考える

```
<?php

function print_add($a, $b)
{
    echo $a + $b;
}

print_add(1, 2);
```

汎用的な処理と
実際の値を
切り離すことができた

| 簡単なコードを考える

```
<?php
```

```
function print_add($a, $b)
{
    echo $a + $b;
}
```

```
print_add(1, 2);
```

こいつらが
何者かわからん

Untitled

```
1 <?php
2
3 function print_add($a, $b)
4 {
5     echo $a + $b;
6 }
7
8 print_add([1], [2]);
9
```

Preview

Output for 8.0.3 | released 2021-03-04 | took 35 ms, 16.68 MiB

Warning: Array to string conversion in /in/unZ6j on line 5
Array

型をつけよう

| 簡単なコードを考える

型が明確になった

```
<?php
```

```
function print_add($a, $b)
{
    echo $a + $b;
}
```

```
print_add(1, 2);
```

そもそも
型とは

| データ型

- PHPで使える値の種類のこと
- あたい 値 = へんすう 変数・ていすう 定数にセットしたり、関数呼び出しに渡せるもの
- 値は何らかのデータ型に属する

| PHPと型

- PHPは型システムの分類として弱い動的型付けあるいは「型なし」と分類されがち
- しかしPHPは単なる型なしと呼ぶには型を使ってできることが多い

| PHPの実行時型検査

- 関数のパラメータと戻り値、プロパティに型宣言できる
- 強力な実行時型検査
 - 実行時にコードの型宣言通り型で実行されることが保障
 - TSやPythonとは異なる

| PHP5の型ヒント

- Type Hintingと呼ばれていた
- `array`, `callable`あるいはクラス/インターフェイスしか書けなかった
- PHP7.0以降ではスカラー型に対応し、型宣言に改称

| PHPの静的型検査

- Psalm, PHPStanなどの検査ツールが豊富
- PhpStormは検査能力は劣るが簡単に導入できる
- 今回の発表では個別のツールの導入方法には触れません

| データ型(1) int 整数型

- 0, 1, 2や-100のような数
- 最大値はPHP_INT_MAXで
最小値はPHP_INT_MIN

| データ型(2) string 文字列型

- "a" "xyz" "あいいう" のような文字の並びだと思ってください
- 長さ0の ""^{から} (空文字列) も存在する

PHPの文字列値はエンコーディングを持たないバイト列

ここで簡単な
コードを考えて
みましょう

簡単なコードを考える

```
<?php
```

```
$a = 1;
```

```
$b = "2";
```

```
$c = $a + $b;
```

```
echo $c;
```

値
1

値
"2"

値
1

値
"2"

式
1 + "2"

値
3

細かくいうと $\$a = 1$
のような変数代入も式

値
3

リテラルと式

- コードに直接書く値を
リテラル(literal)といいます
- 1 2.3 "abc" みたいなもの
- $\$a + \b のようなコードを
式(expression)といいます

| 値には必ず型がある

```
<?php
```

```
$a = 1;
```

```
$b = "2";
```

```
$c = $a + $b;
```

```
echo $c;
```

int(1)

string("2")

int(1)

string("2")

式
1 + "2"

int(3)

int(3)

| データ型(3) float 浮動小数点数型

- 0.1, 10.0, -0.3, INF, -INF
のような値
- PHPでは整数の範囲を外れた
数はfloatになる
- `PHP_INT_MAX+1` は **float**

| データ型(4) bool 論理型

- true と false の二種類だけの値からなる型
- `$a==$b` や `$a < $b` のような比較式の結果はbool
- PHPでは `$a && $b` もbool

｜ スカラー型のまとめ

- bool, int, float, stringの4種をスカラー(scalar)と呼ぶ
- 後述する複合型でも特殊型でもないものという共通性

| 複合型

- 内部に別の値を持てる型
 - `array` (配列)
 - `object` (オブジェクト)

※ PHPマニュアルは`callable`特殊型と`iterable`特殊型も複合型として記載

| データ型(5) array 配列型

- 他言語のリストと辞書(ハッシュ)を兼ねた型だが、基本は辞書的
- `[1, 2, 3]` や `[]` (空配列)
`['name' => 'taro', 'birthday' => '09-13']`
のように複数の値を格納できる

| データ型(5) array 配列型

- 作成時にキーを指定しないと0からの連番になる
 - `[1]`と `[0 => 1]` は同じ
 - `$a = [3 => 'a']` は
`[null, null, 'a']`ではない

| データ型(6) object

- プロパティ(メンバー変数)を持ち、`$obj->prop` のような構文でアクセスできる値の型
- objectには属するクラスがあり、組み込みクラス、ユーザー定義、`stdClass`がある

| 型宣言とクラス

- 型宣言にはクラス名またはインターフェイスを記述できる
- 事実上のユーザー定義型
- インターフェイスを活用すると実装と宣言を分離できる
- 依存性逆転の原則(DIP)

| 全てinterfaceで指定すべきか

- 抽象化でたしかに実装詳細への依存を避けることができる
- 全ての依存をインターフェイスにすべきかは悩ましい
- DTOやValueObjectなど
- DateTimeInterface

| 特殊型

- 単体で型宣言できない特殊な値
 - resource (リソース)
 - null (空値)

※ nullはnullableの一部やPHP8のユニオン型の一部として記述可能

| 型ではないが特殊な戻り値

- `function(): void {}`
- 値を返さないことを示す
- 値を返さないということは、副作用があるというマーク
- DBにデータを記録する、メールを送る、例外送出

| 複合的な型表記

- nullable型
 - ?DateTimeとか
- ユニオン型
 - A|Bとかstring|falseとか
 - PHP7ではPHPDocコメントに書く必要がある

クラスの型付けを
考えてみましょう

| Book (本)

- プロパティ(メンバー変数)として、name(書名)とAuthor(著者オブジェクト)を持つ
- メソッドは今回は割愛

| クラスの型付けを考える 5.0

```
<?php // PHP 5.x~7.3
```

```
class Book {  
    /** @var string */  
    private $name;  
    /** @var Author */  
    private $author;  
  
    public function __construct(  
        string $name, Author $author  
    ) {  
        $this->name = $name;  
        $this->author = $author;  
    }  
}
```

| クラスの型付けを考える 7.4

```
<?php // PHP 7.4
```

```
class Book
{
    private string $name;
    private Author $author;

    public function __construct(
        string $name, Author $author
    ) {
        $this->name = $name;
        $this->author = $author;
    }
}
```

| クラスの型付けを考える 8.0

```
<?php // PHP 8.0以降
```

```
class Book
{
    public function __construct(
        private string $name,
        private Author $author
    ) {
        // プロパティ代入は不要

        // $this->name = $name;
        // $this->author = $author;
    }
}
```

仕様変更



著者が複数の本って
いっぱいあるんじゃない

著者が複数…
配列で受け取ろう

\$author



\$authors_

| 变更前 7.4

```
<?php // PHP 7.4
```

```
class Book
{
    private string $name;

    private Author $author;

    public function __construct(
        string $name, Author $author
    ) {
        $this->name = $name;
        $this->author = $author;
    }
}
```

| 変更後 7.4

```
<?php // PHP 7.4
```

```
class Book
```

```
{
```

```
    private string $name;
```

```
    private array $authors;
```

```
    public function __construct(
```

```
        string $name, array $authors
```

```
    ) {
```

```
        // ...
```

```
    }
```

arrayって…
情報量減っちゃってね？

| array型宣言は結構困る

- 今回欲しいのはAuthorだけが
入った配列
- arrayだけでは不十分
 - ['name' => '太宰治'] みたいな
配列がほしいわけではない

ここから出てくる
のがPHPDoc

| 变更前 7.4

```
<?php // PHP 7.4
```

```
class Book
{
    private string $name;

    private array $authors;

    public function __construct(
        string $name, array $authors
    ) {
        // ...
    }
}
```

| PHPDocでプロパティ型付け

```
<?php // PHP 7.4
```

```
class Book
```

```
{
```

```
    private string $name;
```

```
    /** @var Authors[] */
```

```
    private array $authors_;
```

```
    public function __construct(
```

```
        string $name, array $authors_
```

```
    ) {
```

```
        // ...
```

```
    }
```

| PHPDocでメソッド型付け

```
<?php // PHP 7.4
```

```
class Book
```

```
{
```

```
    private string $name;
```

```
    /** @var Authors[] */
```

```
    private array $authors_;
```

```
    /**
```

```
        * @param Authors[] $authors
```

```
        */
```

```
    public function __construct(
```

```
        string $name, array $authors_
```

```
    ) {
```

```
        // ...
```

| PHPDoc(型注釈)について

- DocComment `/** ... */`に型情報を書くことで、型宣言できない詳細な型を付ける
- PHPが対応しない型や、実行時に検査しにくい型を記述できる

| PHPDoc(型注釈)にしか書けない型

- コレクションの型
- ジェネリクス
- ユニオン型はPHPDocで
長年使われていたがPHP8で
型宣言に取り入れられた

| PHPDoc(型注釈)にしか書けない型

- リテラル型・定数型
- array-shapes
(Object-like arrays)

パラメータの型

```
<?php
```

```
/**
```

```
 * 検索エンジンを単語検索する
```

```
 */
```

```
function search(string $word, string $mode) {
```

```
    // ...
```

```
}
```

ユーザーが検索したいワード
なんでも入れていい

検索モードはどんな文字列で
も入れたいわけではない

| パラメータのリテラルユニオン型

```
<?php
```

```
/**
```

```
 * 検索エンジンを単語検索する
```

```
 * @phpstan-param 'exact' | 'partial' $mode
```

```
 */
```

```
function search(string $word, string $mode) {
```

```
    // ...
```

```
}
```

期待値をResultでそのまま書ける

パラメータの定数プレフィクス型

```
<?php
const SEARCH_MODE_EXACT = 'exact';
const SEARCH_MODE_PARTIAL = 'partial';

/**
 * 検索エンジンを単語検索する
 * @phpstan-param SEARCH_MODE_* $mode
 */
function search(string $word, string $mode) {
    // ..
}
```

ユーザーが検索したいワード
なんでも入れていい

型なしは
どこから来るの？

ここまでは値をリテラル
で記述するから数経由で
期待通りの値が渡される
前提で話をしてきた

リテラルは実行時変化しない

```
<?php
```

```
function print_add(int $a, int $b): int|float  
{  
    echo $a + $b;  
}
```

```
print_add(1, 2);
```

| 外部入力

```
<?php
```

```
function print_add(int $a, int $b): int|float  
{  
    echo $a + $b;  
}
```

```
print_add($_GET['a'], $_GET['b']);
```

クエリが
きちんと渡されたら動く

期待する形式の値が渡
されなければエラー

| 厳密な入力

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

function print_add(int $a, int $b): int|float
{
    echo $a + $b;
}

print_add($_GET['a'], $_GET['b']);
```

厳密な型付けを有効化

?a=1&b=2 が渡されようが
型が違うので絶対動かない

| DB(PDO)から取得した型

- PDOはデフォルトでプリペアドステートメントのエミュレーションが有効
- その場合は全てのカラムが文字列になってしまう

| デシリアライズした値

- `unserialize()`,
`json_decode()`など
- これらも何が入っているかわからない

まとめ

今回はPHPに組み込みの型の使いかたに絞って説明しました

実際に厳密に型を付けて
いくと、これだけでは力
不足なポイントが多く出
てきます

“

配列の型と向き合う

”

2020-02-26 PHP勉強会@東京
#phpstudy

-arrayの型と向き合う #phpstudy
<https://tadsan.fanbox.cc/posts/854598>



28



20



@tadsan が2021年08月02日に更新 6296 views

array shapes記法(Object-like arrays)と旧PSR-5記法で型をつける

🏷️ PHP, PSR-5, array-shape

この数年でPHPでの開発でもCI(継続的インテグレーション)はかなり活発に行われるようになり、PHPUnitなどのテストフレームワークのほか、PHPStanやPhanなどのツールによる静的解析も浸透しつつあります。

関数/メソッドの引数と返り値、そしてオブジェクトのプロパティは比較的に型がつけやすいところですが、現状で無法地帯な箇所があります。そうです、配列の内部構造です。実際のところ、PHPDocに `@param array` や `@return array` と書くことは `mixed` と書くのとあまり大きな違いはありません。

–array shapes記法(Object-like arrays)と旧PSR-5記法で型をつける
<https://qiita.com/tadsan/items/bfa9465166c351da37e5>

“

Qiita

キーワードを入力

LG TM

170

143

Twitter

f

...

⚠️ この記事は最終更新日から1年以上が経過しています。

@tadsan が2020年01月29日に更新 14257 views

このPHPがテンプレートエンジンのくせに慎重すぎる (前篇)

PHP

LG TM

👍 👤 🌐 🌐 🌐

この記事ではPackagistで公開可能な形式のPHPのライブラリ(Composerパッケージ)を公開するための道具立てを紹介します。あと、現代のPHPerはツールを組み合わせでさくっと開発しているんだという自慢です。

タイトルは「この TypeScript が Hello, world! のくせに慎重すぎる」と「この勇者が俺TUEEEくせに慎重すぎる」のぱく r... パロディです。

テンプレートエンジンのくせに型安全なんてなまいきな。

”

–このPHPがテンプレートエンジンのくせに慎重すぎる (前篇)
<https://qiita.com/tadsan/items/bf61520eb2d455e0e8b4>