

実践PHPStan

Practical PHPStan



pixiv Inc.
USAMI Kenta

pixiv

お前誰よ



- うさみけんた (@tadsan) / Zonu.EXE / にゃんだーすわん
- ピクシブ株式会社 pixiv事業本部 エンジニア
 - 最近はピクシブ百科事典(dic.pixiv.net)を開発しています
- Emacs Lisper, PHPer
 - Emacs PHP Modeを開発しています(2017年-)

tadsanのあれこれが読める場所



- <https://tadsan.fanbox.cc/>
- <https://scrapbox.io/php/>
- <https://www.phper.ninja/>
- <https://zenn.dev/tadsan>
- <https://qiita.com/tadsan>
- <https://github.com/bag2php>

https://www.phper.ninja/

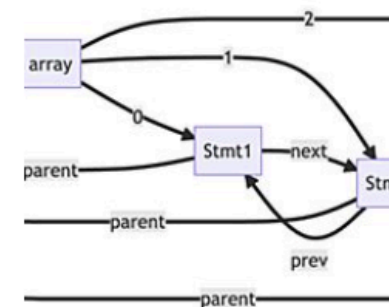
超PHePerになろう

Enjoy PHP Programming

📅 2022-04-28

条件付き戻り値型とPHPStan 1.6.0の新機能

🔖 PHPStan



この記事はPHPStan開発者のOndřej Mirtesによって2022年4月26日にPHPStan Blogに書かれた記事を翻訳したものです。phpstan.org 条件付き戻り値型 (Conditional return types) この機能の大部分はRichard van Velzenが開発しました。PHPStanは初リリース以...

47 users 🌟 🌟 🌟 🌟 🌟 🌟 🌟

📅 2021-04-09

PHPerKaigi 2021に参加して、それから

去年に引き続きPHPerKaigiにコアスタッフ・発表者として参加してました。phperkaigi.jp そういえば一年前にこんな記事も書いたりもしてました。www.phper.ninja 去年の暮れから今年にかけて体調を崩しておりタイミング的にぎりぎりだった気がします。(在宅...

1 user 🌟 🌟 🌟 🌟

📅 2021-04-05

戻り値の記憶と忘却

プロフィール



zonu_exe PRO

すごいPHePerになろう

<http://qiita.com/tadsan>

✓ 読者です 30

[このブログについて](#)

検索

記事を検索



最新記事

条件付き戻り値型とPHPStan 1.6.0の新機能

PHPerKaigi 2021に参加して、それから

戻り値の記憶と忘却

https://scrapbox.io/php/

USAMI Kenta

PHP

Date modified

Ruby on Rails

Rubyで実装されたWebアプリケーションフレームワーク。Rackに対応したアプリケーションサーバで動作する。

Flash(Webフレームワークの機能)

Ruby on Railsなどが備えている機能。Railsに影響を受けたCakePHPも同名の機能を持つ。

トップレベル名前空間

階層化されていない名前空間のこと
ファイルにnamespaceと書かれていなければ

Queuesadilla

GitHub:
<https://github.com/josegonzalez/php-queuesadilla>
PHPで実装されたジョ

PHP本格入門の私教版正誤表

このページには「PHP本格入門」に含まれる誤りを集めています。
この本には2020年の現

PHP処理系

PHPを実行またはコンパイルするソフトウェアの総称。ランタイムとも。
一般的にはThe PHP

-1

0より1少ない数。
ただの負の整数なのだが、PHPにおいては2の補数によりすべてのビットフラグが立つ特別

Nette

Nette Foundationが開発するPHPコンポーネント群。
公式サイト:
<https://nette.org/en/>

APM

英語: Application Performance Management または Application Performance

空文字

C言語などで、文字列の終端として機能する
\0(NUL)のような文字のこと。PHPのnullとは異なる。

空文字列

長さ0の文字列のこと。
#英語: empty string
空文字とは異なる。
`$s = '';`
`var_dump(strlen($s))`

交叉型

英語: intersection types
日本語: 交叉型 (こうさがた) または 交差型 (読みは同じ)、インターセクション型

DNF型

英語: DNF Types / Disjunctive Normal Form Types
日本語: DNF型 / 選言標準形型 (せんげん ひょう)

ユニオン型

英語: union types
日本語: ユニオン型 / 合併型
A|Bと書くことで、「AまたはB」を示す型。

Pure intersection types

実装されたバージョン:
PHP 8.1
PHP RFC: Pure intersection types

選言標準形型

DNF型のこと。せんげんひょうじゅんけいがた。

Disjunctive Normal Form Types

DNF型のこと。日本語では

組み込み型

データ型と特殊型のうち、型宣言に記述可能な型のこと
一般的に、組み込み型はすべて小文字で表記

データ型

英語: data types / primitive types
値の分類要素。
PHPの全ての値は何らかのデータ型に属す

型エイリアス

データ型の別名(エイリアス)のこと
キャストに使うことは可能
PHP: 型の相互変換 -

古い型エイリアス

キャストのために
boolean
integer
binary
double

今回のお題

プロポージャー



うさみけんた



[tadsan](#)

近年、PHPプロジェクトの品質を高めるためのツールとしてPsalmやPHPStanのような静的解析が開発現場にも取り入れられています。

このトークでは静的解析ツールの中でもPHPStanの型と機能にフォーカスして、PhpStormと共存しながら、詳細な型を付けるための手法について具体的な事例と改善例を挙げて紹介します。

- 外部との入出力の型付け
- 配列との付き合い方
- 曖昧な型を狭める方法
- 古いPHPとの互換性

先日、型についての話もしました

FTECHFEED
CONFERENCE
2022

5分でわかるPHPの型システム

Introducing PHP's type system in 5 minutes

にゃんだーすわん @tadsan / USAMI Kenta うさみけんた

#tfcon

x

PHPカンファレンスでも話しました

配列、ジェネリクス、 PHPで書けない型

Arrays, Generics, and Types that cannot be type-declared.



pixiv Inc.
USAMI Kenta

pixiv

x

2021.10.02

今回のゴール

PHPStanで型を付けて
既存コードの型を検査

今回やらないこと

PHPStanの詳細な設定
CI環境の組み方
ジェネリクス/条件型

ジェネリクスについて

超PHErになるう

Enjoy PHP Programming

📅 2020-03-01

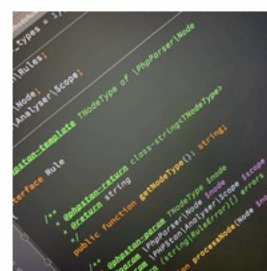
PHPDocを使ったPHPのジェネリクス

🔖 PHPStan

この記事はPHPStan開発者の[Ondřej Mirtes](#)によって2019年12月2日に書かれた記事を翻訳したものです。記事の末尾には訳者(@tadsan)の観点によるPhan, Psalm, PhpStormとの互換性についての情報も記述しています。

Generics in PHP using PHPDocs

A couple of years I wrote an impactful article on union and intersection types. It helped the PHP community to familiarize themselves with...



👤 medium.com

[medium.com](#)

2年前、私(Ondřej Mirtes)は[ユニオン型と交差型](#)についての衝撃的な記事を書きました。PHPコミュニティがこれらの概念に馴染むのを手助けし、[PhpStormでの交差型サポート](#)につながりました。

ユニオン型と交差型の違いは開発者が認識すべき静的解析に役立つ重要な概念なので、私はその記事を書きました。今回は同様に、[PHPStan 0.12で導入された](#)ジェネリクスについて、それが何であるかを説明したいと思います。

プロフィール



zonu_exe PRO

すごいPHErになるう
<http://qiita.com/tadsan>

✓ 読者です 30

[このブログについて](#)

検索

記事を検索



最新記事

条件付き戻り値型とPHPStan 1.6.0の新機能

PHPerKaigi 2021に参加して、それから

戻り値の記憶と忘却

PHPDocを使ったPHPのジェネリクス

条件付き戻り値型

超PHPerになろう

Enjoy PHP Programming

📅 2022-04-28

条件付き戻り値型とPHPStan 1.6.0の新機能

🔖 PHPStan

この記事はPHPStan開発者の[Ondřej Mirtes](#)によって2022年4月26日にPHPStan Blogに書かれた記事を翻訳したものです。

PHPStan 1.6.0 With Conditional Return Types and More!



🌐 phpstan.org **1 user**

phpstan.org

条件付き戻り値型 (Conditional return types)

この機能の大部分は[Richard van Velzen](#)が開発しました。

PHPStanは初リリース以来、関数呼び出しで渡された引数によって様々な型を返す方法を提供してきました。いわゆる[動的戻り値型拡張\(dynamic return type extensions\)](#)は非常に柔軟です。実装できる任意のロジックによって型を解決できます。しかし、[PHPStan拡張の核心となるコンセプト](#)には学習コストがかかります。

[PHPStan 0.12](#)ではジェネリクスが導入されました。これはPHPDocの特別な記法によって動的戻り値型拡張が必要だったケースの一部をカバーしています。

プロフィール



zonu_exe

PRO

すごいPHPerになろう

<http://qiita.com/tadsan>

✓ 読者です 30

[このブログについて](#)

検索

記事を検索



最新記事

条件付き戻り値型とPHPStan 1.6.0の新機能

PHPPerKaigi 2021に参加して、それから

戻り値の記憶と忘却

PHPDocを使ったPHPのジェネリクス

あなたが今年PHPPerKaigi 2020に参加しなければいけない理由

このスライドは
本日中に公開されます

この発表では知っておくべき機能を雑に取り上げるので本番運用／提案の前にドキュメントを自身でしっかり読み込みましょう。

それでもわからないことがあれば
TwitterかSlackのphpusers-ja
#type-safeで尋ねてください

PHPStan
使ってますか？

Meet The Next Member of Your Team!

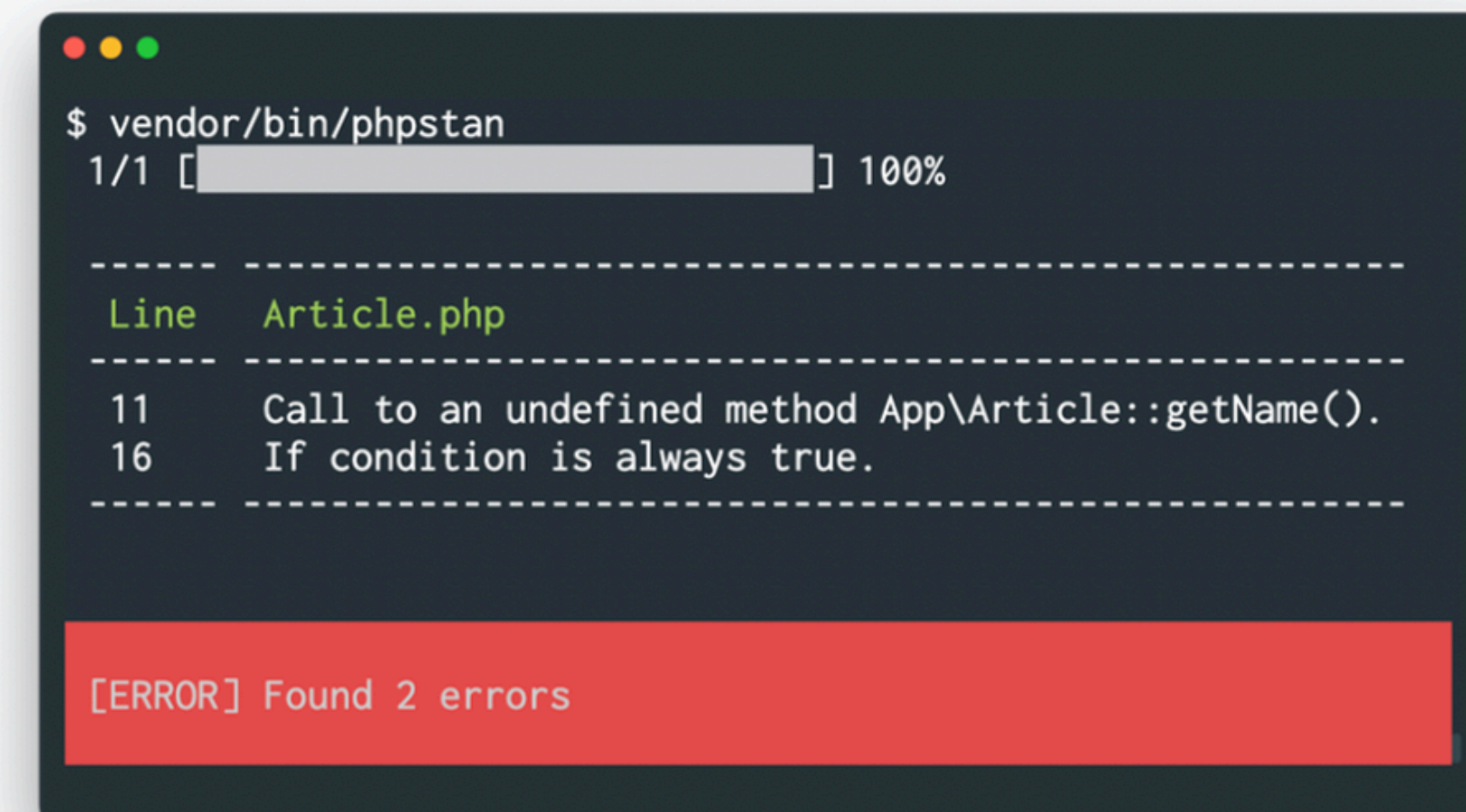
PHPStan finds bugs in your code without writing tests. It's open-source and free.

[Get Started](#)[Try It Online](#)

Find bugs before they reach production

PHPStan scans your whole codebase and looks for both obvious & tricky bugs. Even in those rarely executed if statements that certainly aren't covered by tests.

You can run it on your machine and in CI to prevent those bugs ever reaching your customers in production.



```
$ vendor/bin/phpstan
1/1 [████████████████████] 100%

-----
Line   Article.php
-----
11     Call to an undefined method App\Article::getName().
16     If condition is always true.
-----

[ERROR] Found 2 errors
```


PHPStanとは何か

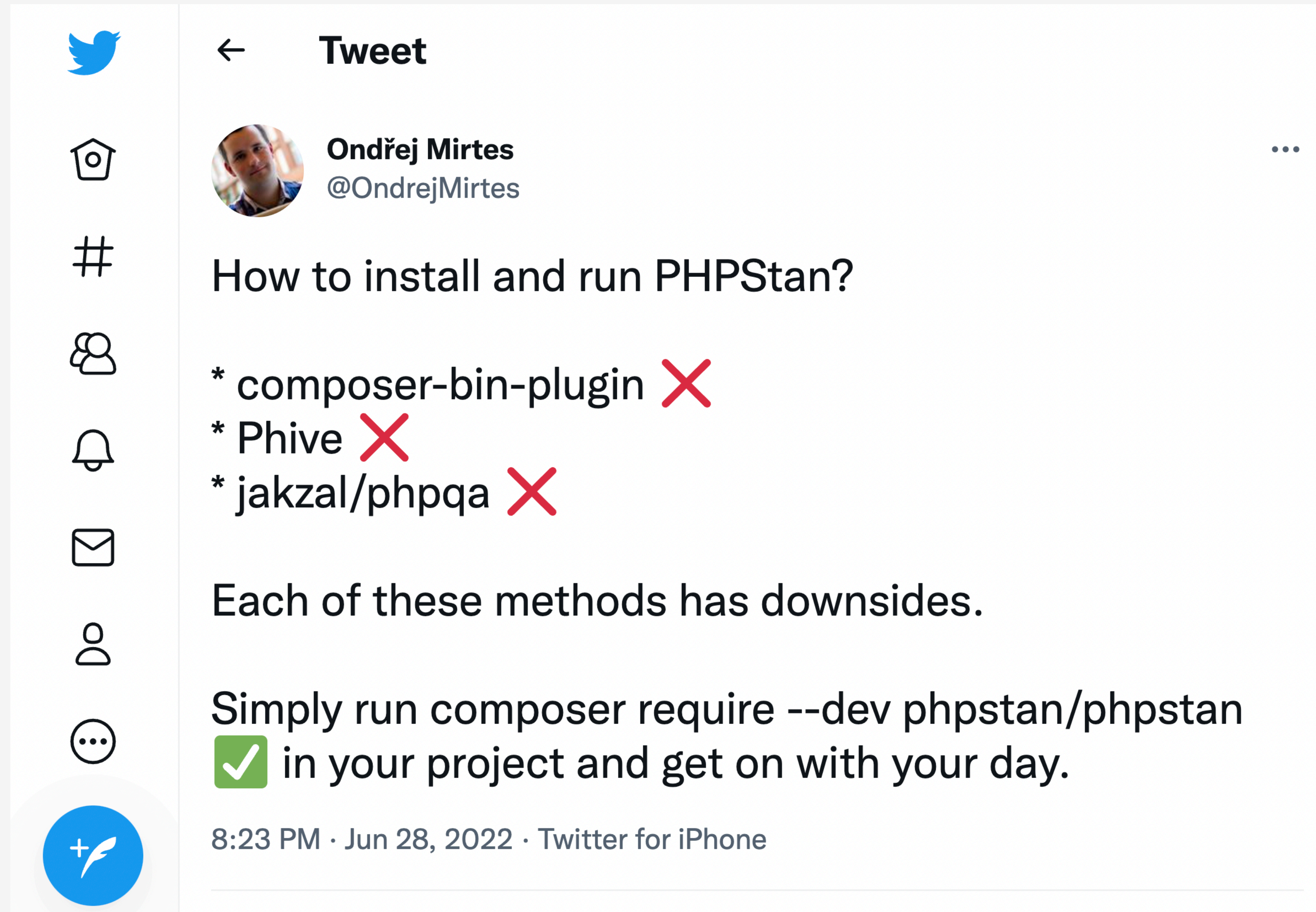
- 静的に(=プログラムを動作させずに)コードの状態を解析するツール
 - 広い意味ではコードのフォーマッターやタグ生成なども含む
 - PHPStanは静的解析の中でも^{リッ}**Lint**^{ター}と呼ばれるジャンルのツール
- ⇔ 動的解析（静的解析の対義語）
 - XdebugやPHPUnitはプログラムを動かしてみて性質を確かめる
 - 従来のPHPStanは静的と動的のハイブリッドだった

なに? 使ってない?

あなたとPHPStan
いますぐダウンロード

どうやって？

Ondřej said...



PHP 7.2以上ならプロジェクトに追加しよう

```
composer require --dev  
phpstan/phpstan
```


プロジェクトに追加できないなら

composer global require
phpstan/phpstan

require --dev以外の方法を検討

- 要は何らかの事情があってcomposer.jsonに追加できないなら仕方ない
 - プロジェクトがPHP 7.2以前に依存しているとき
 - 権限の問題などでcomposer.jsonに追加できないとき
- それ以外は原則Composerに追加しましょう
 - PhpStormの最新版なら、いい感じに勝手にPHPStanを動かしてくれる
 - もっと詳細な話はぺちこん沖縄当日までの間にzennに書きます

PHPStanを入れてどうするの

- Composerの実行ファイルは `./vendor/bin/phpstan` にある
 - 以下、パスを省略して単に `phpstan` コマンドとして表記
- 基本は `phpstan analyze` のように動かすと解析できる
 - `phpstan analyze src/Foo/Target.php` でファイル単位解析
- **PHPStan Pro**というWebインターフェイスもある
 - 弊社では応援の意味も込めてProを契約しています

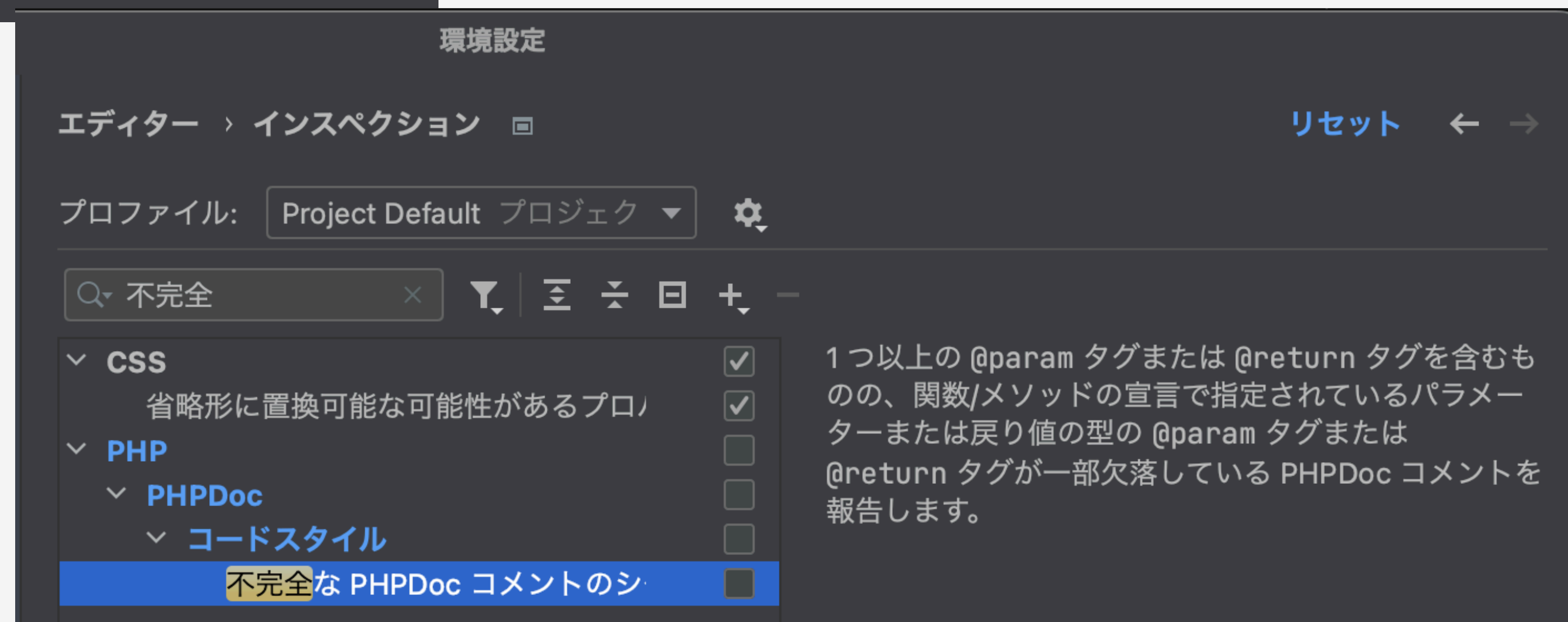
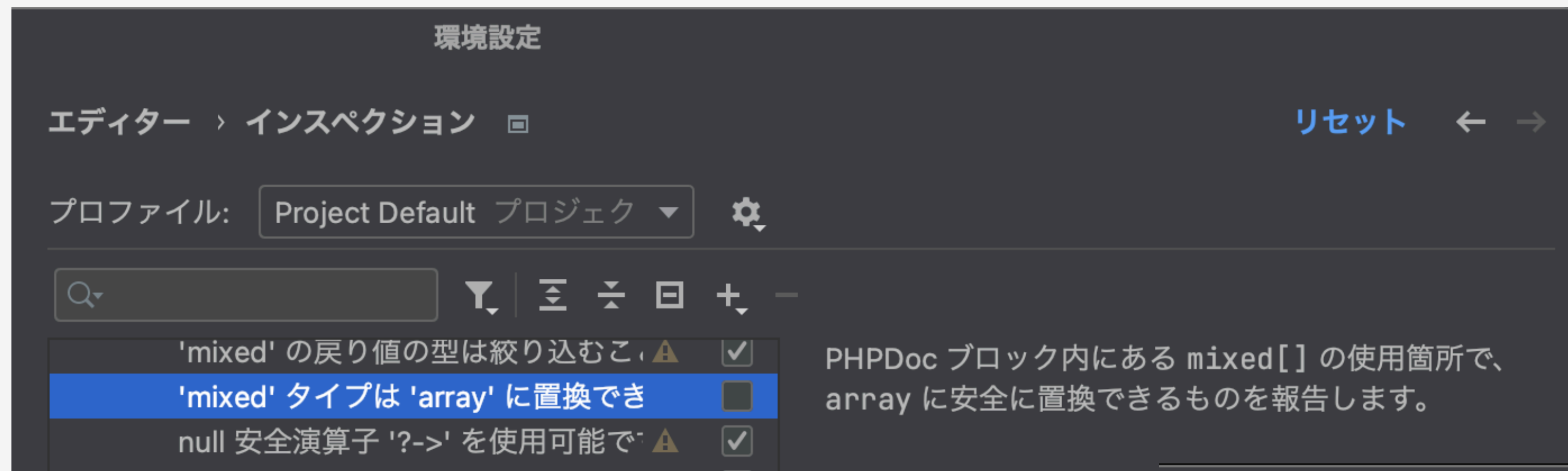
PhpStormがあるのに必要…?

- PhpStormもがんばってPHPStanやPsalmとの互換性を高めているが、解析能力に関しては専門のツールの方が強い
 - PHPStanとPsalmはユーザーがPHPで自由に拡張できる
- PhpStormの方がレガシー寄りのPHPバージョンに強い
 - PHPStanはPHP 7.2を以上サポート
- PhpStormのヘッドレス版のQodanaというツールがCI用途に使える

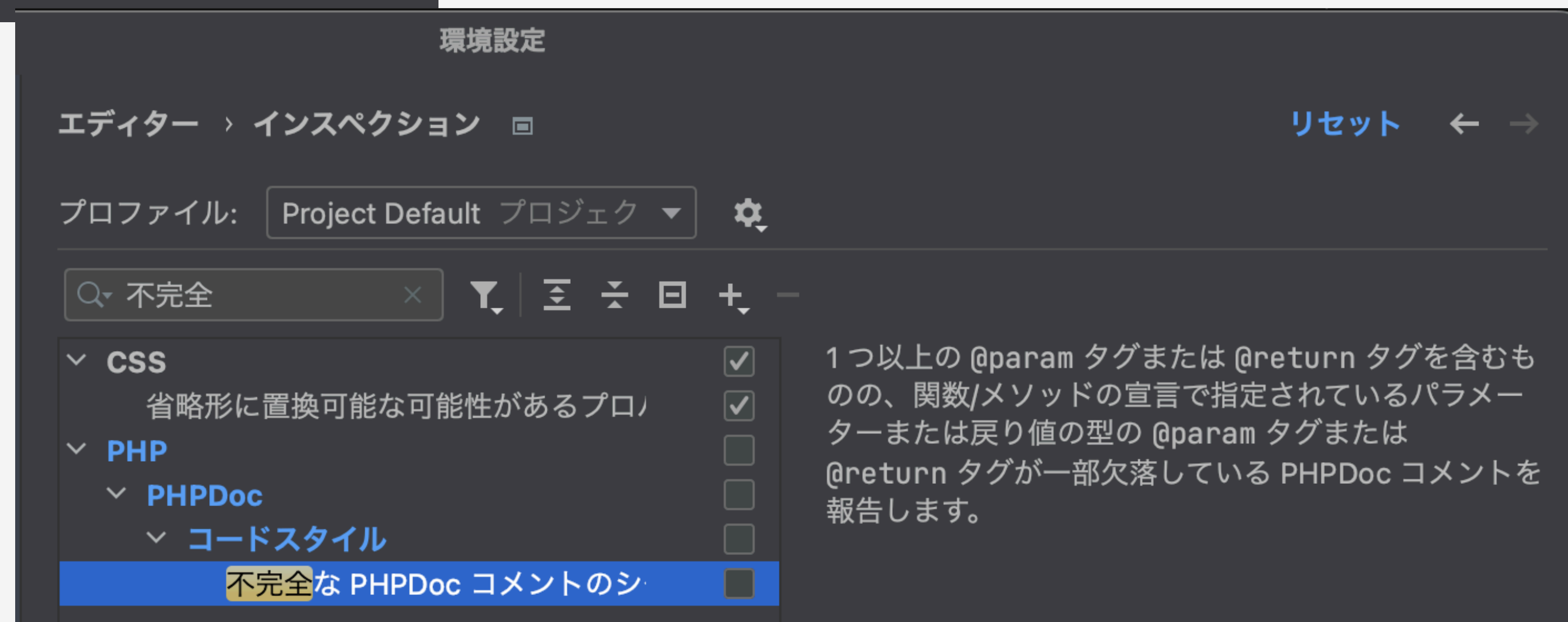
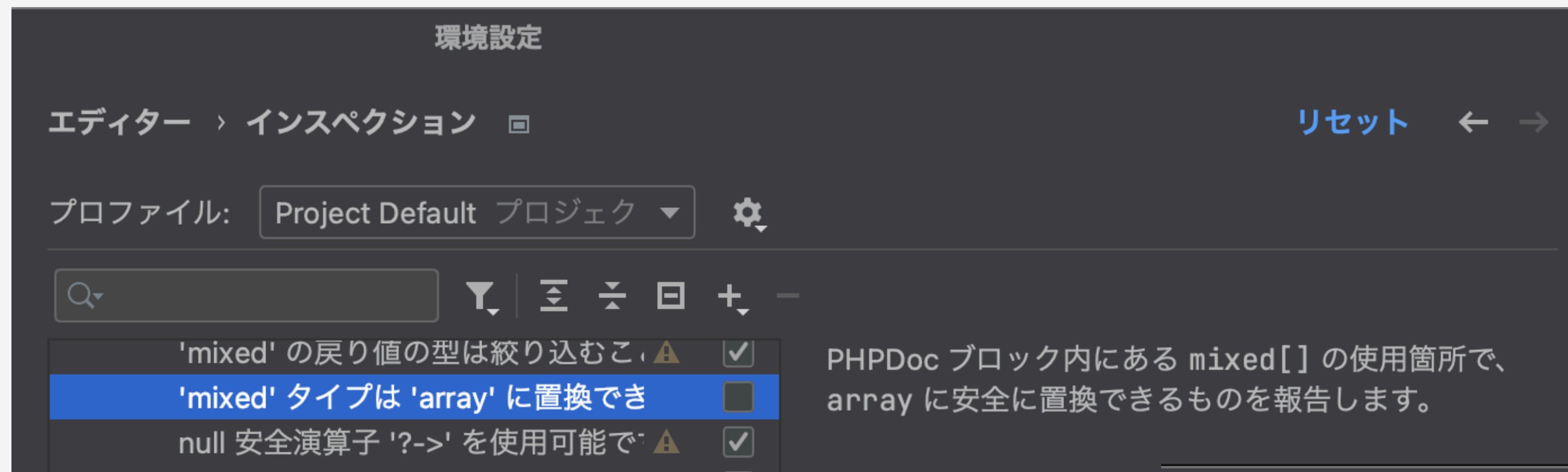
解析結果はリアルタイム表示しよう

- **PhpStorm:** 最近のバージョンでは標準組み込み（場合によって要設定）
- **VS Code:** sordev.phpstan+ErrorLens（Marketplaceには古いのもあるので注意）
- **Vim/NeoVim:** coc.nvimとかALEを入れるといいと思います
- **Emacs:** flycheck-phpstan（私が作っています）
- その他のLSP対応エディタ: efm-langserver（汎用言語サーバー）

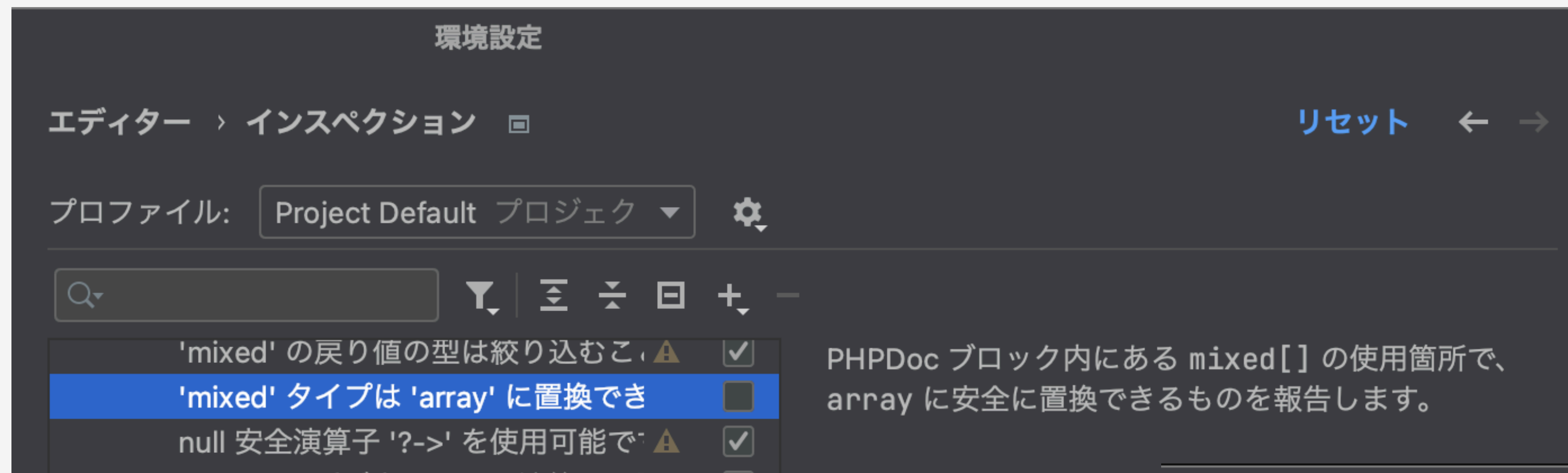
外した方がいいPhpStorm設定



外した方がいいPhpStorm設定



外した方がいいPhpStorm設定



@param とか @return を
執拗に全部埋めてくる

arrayと
array<mixed>と
mixed[] を混同しない



ここ最近のPHPStan

- 完全静的解析モードがデフォルトになりました
 - いままでのPHPStanは実行時リフレクションで解析コストを抑えていた
 - 改善の積み重ねによりパフォーマンスがものすごく向上している
 - 従来通りの実行時リフレクションも使える
- PHPStanはハイペースで改善されているのでガンガン更新しましょう
 - ただし今月は開発者が夏休みを宣言している 🌂

PHPStanは何ではないか

- PHPのあらゆる関数に対して完璧な型を付けてくれるわけではない
 - 不具合や意図しない挙動を発見したら報告してほしい
 - Twitterに書くだけでも作者か日本人の誰かが反応してくれると思う
- オープンソースソフトウェアであり、商用サポートはない
 - チェコ在住の個人(Ondřej Mirtes)が開発している
 - コミュニティによるバグ修正や機能追加の貢献も受け入れている

PHPの型の基礎

型の基本的な考え方

- 基本は型宣言できるPHPの型を覚える
 - クラス/インターフェイス名 (PHP 5)
 - 複合型 array / callable (PHP 5)
 - スカラー値 int / float / string / bool ($\geq$ PHP 7.0)
 - 特別な戻り値型 void ($\geq$ PHP 7.0) / never ($\geq$ PHP 8.1)
 - そのほかの型表記

ユニオン型

A | B

AまたはB
(複雑な式は書けない)

交叉型 **A&B**

AおよびB
(複雑な式は書けない)

DNF型

`int | A | (B & C) | null`

intまたはAまたは「BおよびC」またはnull
(一定の制約下で複雑な型も書ける)

nullable型 ?A

Aまたはnull
デフォルトで許容しない

謎のPHP用語

mixed

任意の(すべての)型を表す型
(ほかの言語でのany)

型をどこに書くか

型宣言 (type declaration)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```

型宣言 (type declaration)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```


型宣言 (type declaration)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```

型宣言 (type declaration)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```


型宣言 (type declaration)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```

型注釈 (type annotation)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```

型注釈 (type annotation)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```


型注釈 (type annotation)

```
class Book
{
    public readonly string $title;
    /** @phpstan-var non-empty-list<Author> */
    public readonly array $authors;

    /**
     * @param non-empty-string $title
     * @phpstan-param non-empty-list<Author> $authors
     */
    public function __construct(string $title, array $authors)
    {
```

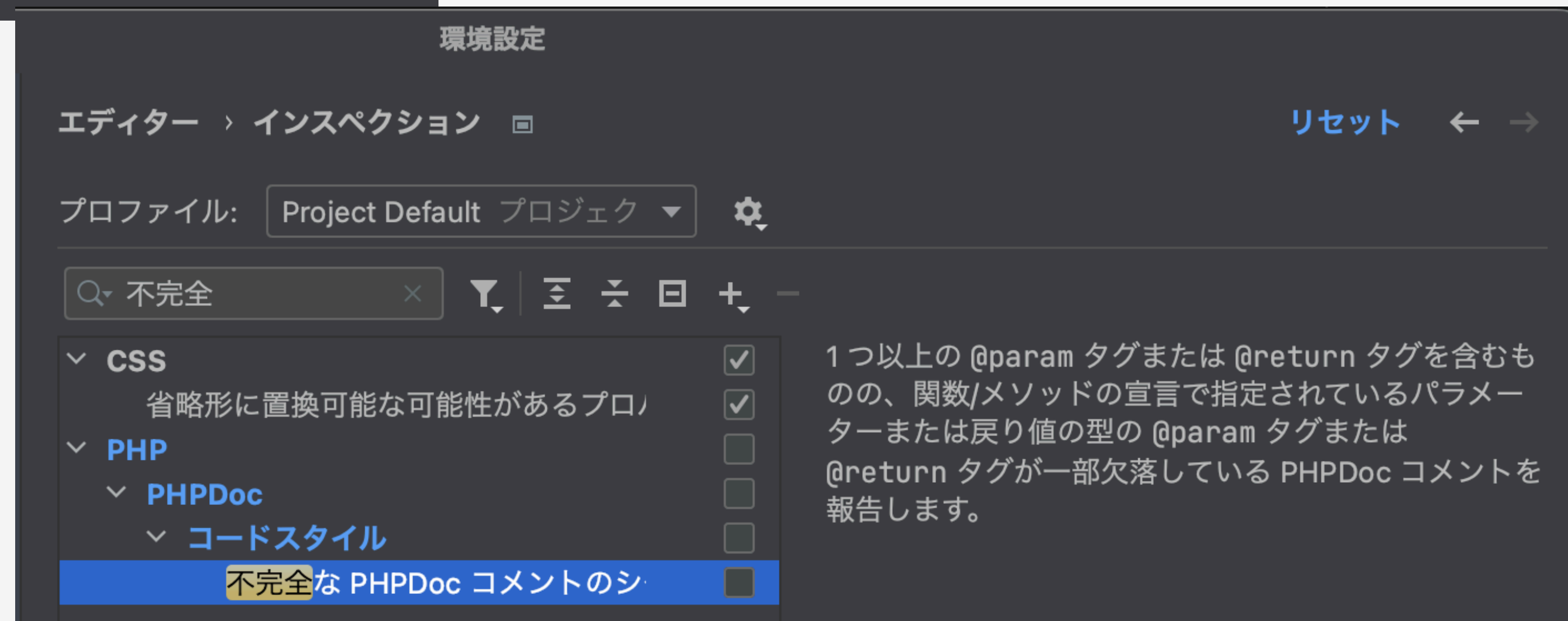
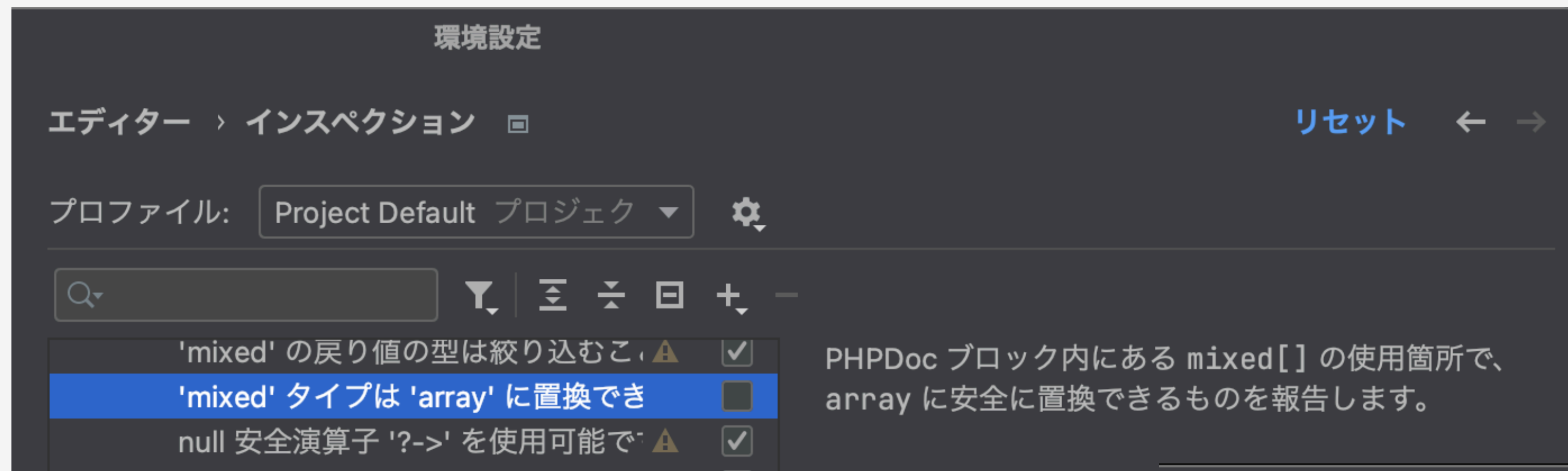
型宣言 vs PHPDoc(型注釈)

- 型宣言された型は、実行時に100%確実に保証される実効性のある型です
- PHPDocは `/** ... */` 形式のブロック内に書かれる単なるコメントです
 - 実行時にはその通りの型であることは保証されず、型は口約束に過ぎません
- この発表内で「型を付ける」というときは型宣言と型注釈のどちらも使います
 - 実効性のある型と口約束に過ぎない型の二枚舌の使い分けが動的言語における現実的な型の付け方になります

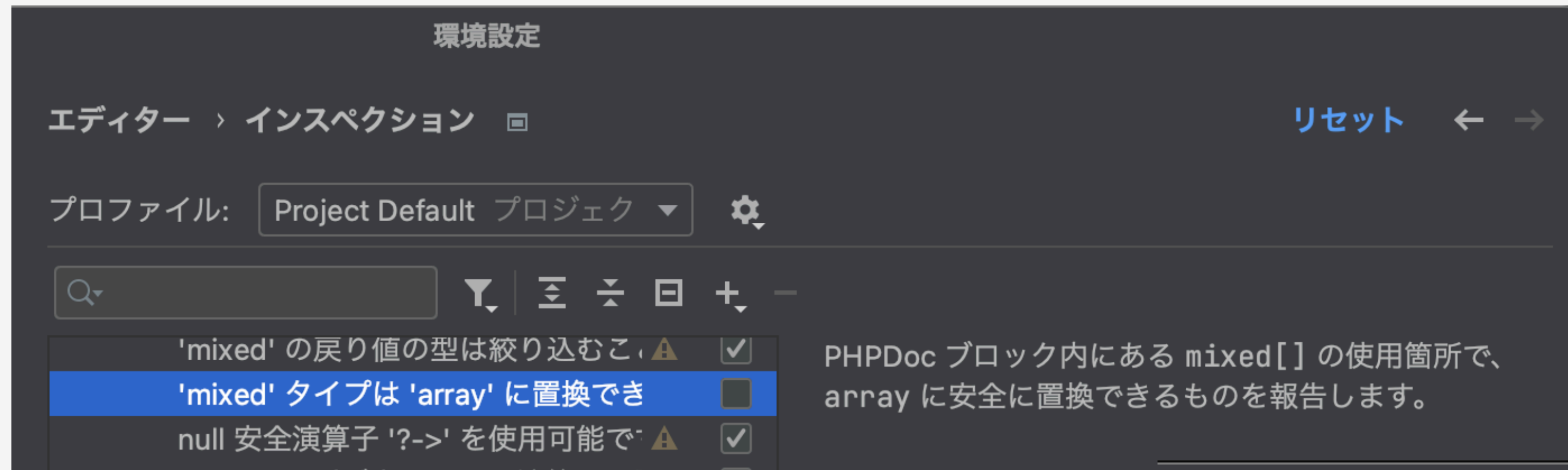
無駄なPHPDocは書かない

- 型宣言と同じ内容をPHPDocに書き写さない
 - 自然言語で説明を書きたいときは書く
 - `int`→`positive-int` `string`→`定数` のように型を強化する場合は書く
- 継承元と同じ型の場合は何も書かない（共変型付けをしたいときは書く）
 - PHP: 共変性と反変性 - Manual
- 几帳面に埋めようとしてくるPhpStormの言うことを聞かない

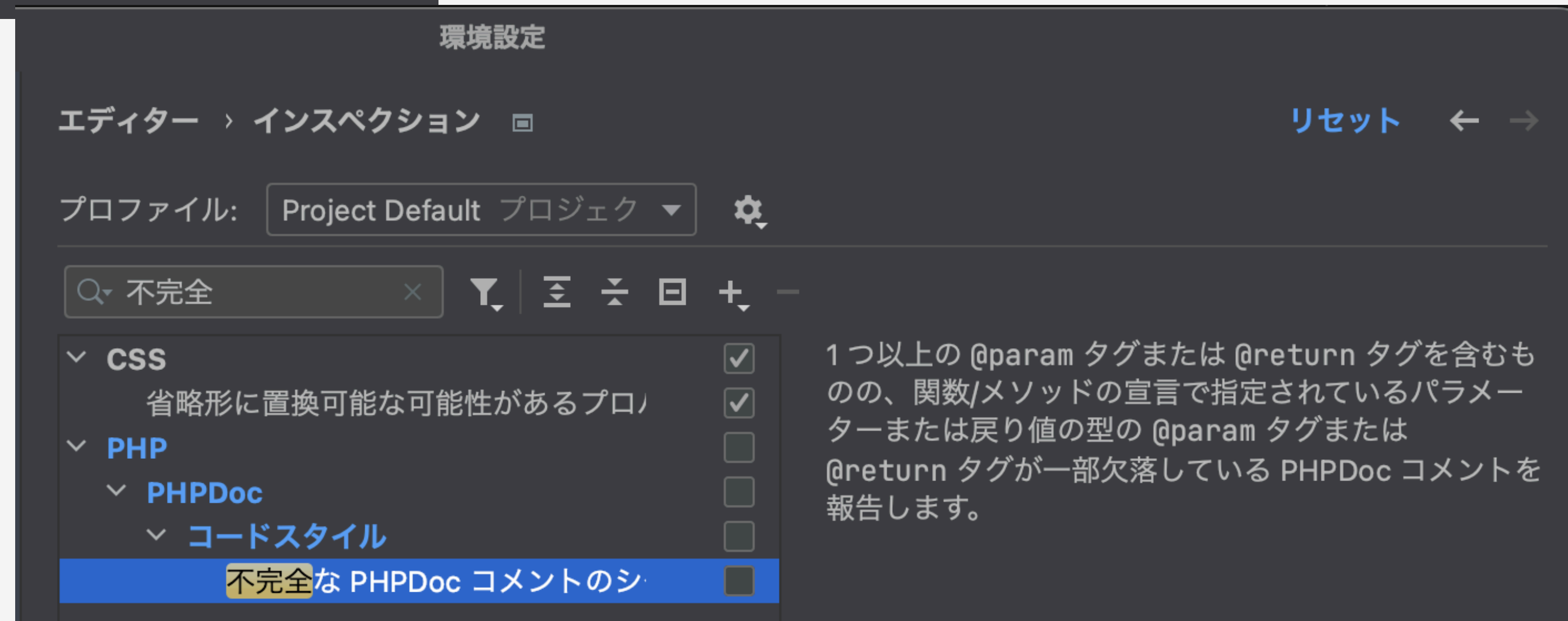
外した方がいい PhpStorm 設定 (再掲)



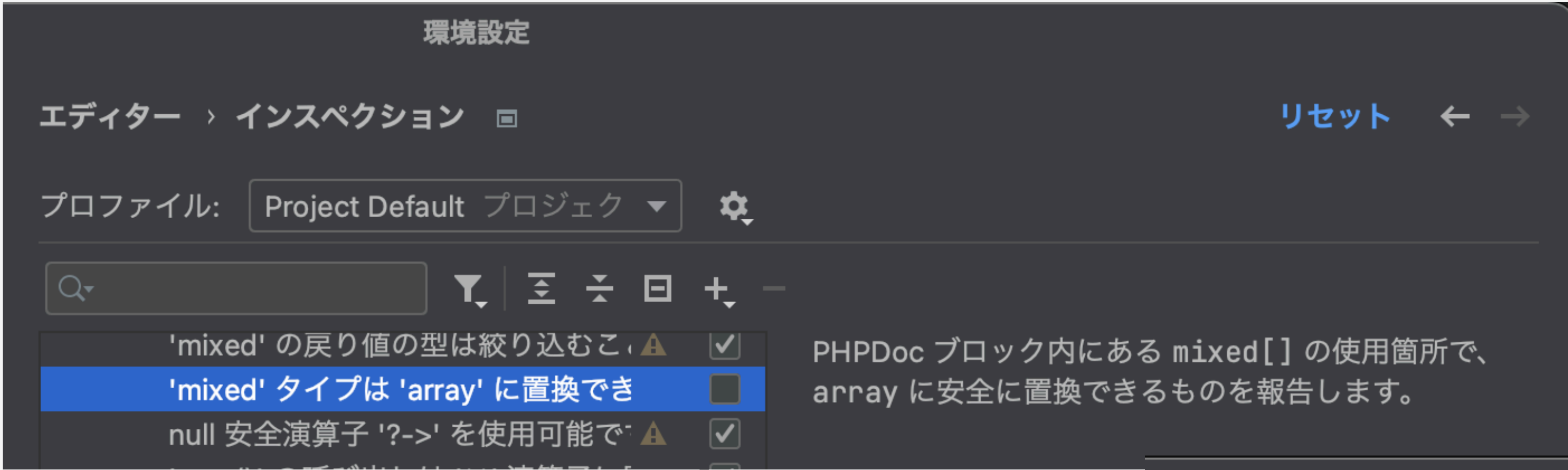
外した方がいいPhpStorm設定(再掲)



arrayと
array<mixed>と
mixed[] を混同しない

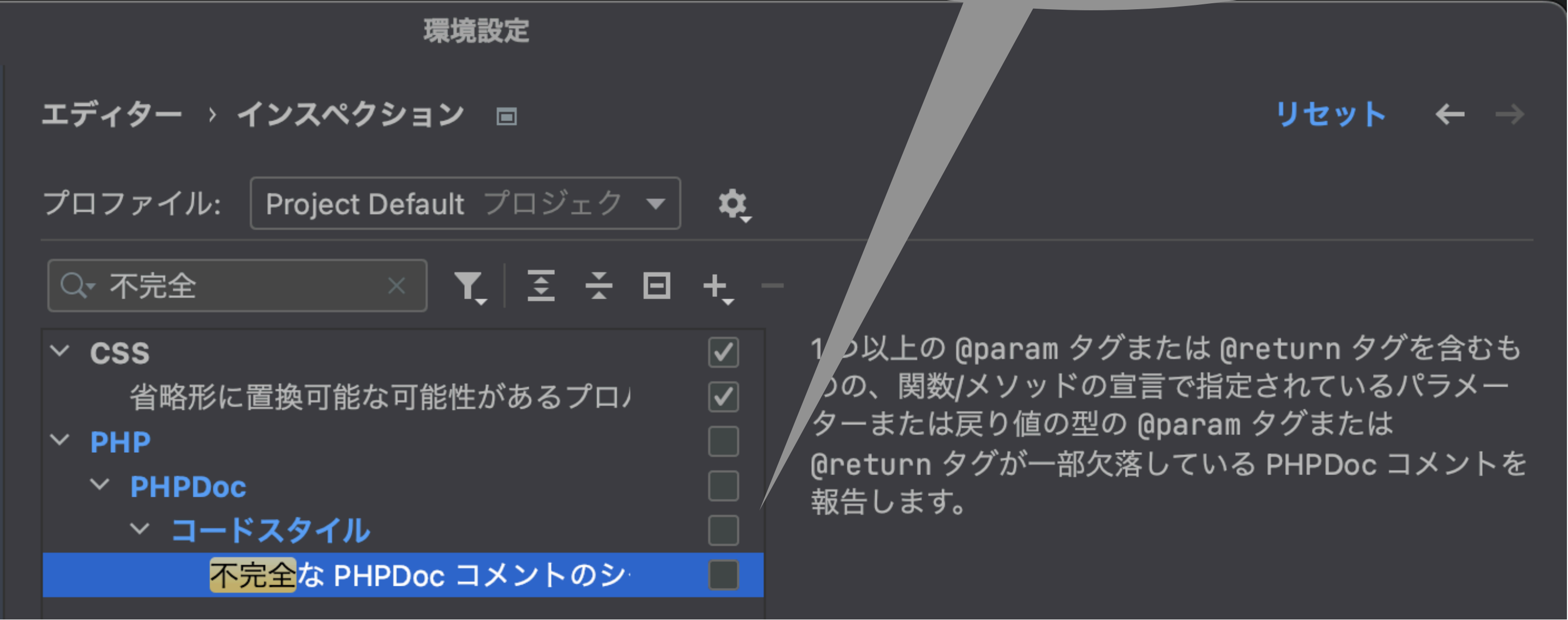


外した方がいい PhpStorm 設定 (再掲)



@param とか @return を
執拗に全部埋めてくる

arrayと
array<mixed>と
mixed[] を混同しない



型宣言があれば
PHPDocなくとも
足りるんじゃないの？

PHPStanには
もっと強力な型がある

PHPStanを 使ってみよう

オンライン サンドボックス

いますぐダウンロード

いますぐダウンロー
ド

しなくてもいい

<https://phpstan.org/try>

 **PHPStan** [Home](#) [Try](#) [Documentation](#) [Blog](#) 

Try out PHPStan and all of its features here in the editor. [Learn more about PHPStan »](#)

```
1 <?php declare(strict_types = 1);
2
3 class HelloWorld
4 {
5     public function sayHello(DateTimeImmutable $date): void
6     {
7         echo 'Hello, ' . $date->format('j. n. Y');
8     }
9 }
```

 Share Level 9 ☐ Strict rules ☐ Bleeding edge ☒ Treat PHPDoc types as certain

Found 2 errors

Line	Error
5	Parameter \$date of method HelloWorld::sayHello() has invalid typehint type DateTimeImmutable.
7	Call to method format() on an unknown class DateTimeImmutable.

© 2016–2022 Ondřej Mirtes  



こういう関数を
考えてみましょう

型宣言で型のついた完璧な関数

```
1 <?php declare(strict_types = 1);  
2  
3 // @phpstan-ignore-next-line  
4 function searchBook(string $word, array $options = []): array  
5 {  
6     return [];  
7 }  
8
```



Share

Level 9 ▼



Strict rules



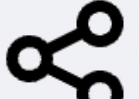
Bleeding edge

No errors!

型宣言で型のついた完璧な関数

本を検索する関数

```
1 <?php declare(strict_types = 1);  
2  
3 // @phpstan-ignore-next-line  
4 function searchBook(string $word, array $options = []): array  
5 {  
6     return [];  
7 }  
8
```

 Share

Level 9 ▼

☐ Strict rules

☐ Bleeding edge

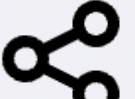
No errors!

型宣言で型のついた完璧な関数

本を検索する関数

検索ワード

```
1 <?php declare(strict_types = 1);  
2  
3 // @phpstan-ignore-next-line  
4 function searchBook(string $word, array $options = []): array  
5 {  
6     return [];  
7 }  
8
```

 Share

Level 9 ▼

☐ Strict rules

☐ Bleeding edge

No errors!

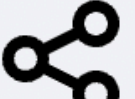
型宣言で型のついた完璧な関数

本を検索する関数

検索ワード

何これ

```
1 <?php declare(strict_types = 1);  
2  
3 // @phpstan-ignore-next-line  
4 function searchBook(string $word, array $options = []): array  
5 {  
6     return [];  
7 }  
8
```

 Share

Level 9 ▼

☐ Strict rules

☐ Bleeding edge

No errors!

型宣言で型のついた完璧な関数

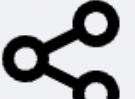
本を検索する関数

検索ワード

何これ

```
1 <?php declare(strict_types = 1);  
2  
3 // @phpstan-ignore-next-line  
4 function searchBook(string $word, array $options = []): array  
5 {  
6     return [];  
7 }  
8
```

何これ

 Share

Level 9 ▼

☐ Strict rules

☐ Bleeding edge

No errors!

PHPStan第一の関門(レベル6)

Found 2 errors

Line	Error
3	Function searchBook() has parameter \$options with no value type specified in iterable type array.
3	Function searchBook() return type has no value type specified in iterable type array.

PHPStan第一の関門(レベル6)

\$options
配列の中身わからん

Found 2 errors

Line	Error
3	Function searchBook() has parameter \$options with no value type specified in iterable type array.
3	Function searchBook() return type has no value type specified in iterable type array.

PHPStan第一の関門(レベル6)

Found 2 errors

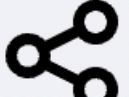
Line	Error
3	Function searchBook() has parameter \$options with no value type specified in iterable type array.
3	Function searchBook() return type has no value type specified in iterable type array.

\$options
配列の中身わからん

returnされる
配列の中身わからん

再度完璧に型を付けた関数

```
1 <?php declare(strict_types = 1);  
2  
3 /**  
4  * @param array<mixed> $options  
5  * @return array<mixed>  
6  */  
7 function searchBook(string $word, array $options = []): array  
8 {  
9     return [];  
10 }
```

 Share

Level 9 ▼

☐ Strict rules

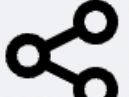
☐ Bleeding edge

No errors!

再度完璧に型を付けた関数

```
1 <?php declare(strict_types = 1);  
2  
3 /**  
4  * @param array<mixed> $options  
5  * @return array<mixed>  
6  */  
7 function searchBook(string $word, array $options = []): array  
8 {  
9     return [];  
10 }
```

配列の中身には
あらゆる可能性

 Share

Level 9 ▼

☐ Strict rules

☐ Bleeding edge

No errors!

型を表示してみる

取り出してきた値の配列

```
8 $result = searchBook('');
9 \PHPStan\dumpType($result);
10 foreach ($result as $book) {
11     \PHPStan\dumpType($book);
12 }
13 |
```

Line	Error
10	Dumped type: array
12	Dumped type: mixed

取り出してきた値の配列

```
8 $result = searchBook('');
9 \PHPStan\dumpType($result);
10 foreach ($result as $book) {
11     \PHPStan\dumpType($book);
12 }
13 |
```

配列の中身わからん

Line	Error
10	Dumped type: array
12	Dumped type: mixed

型がついていないとはどういうことか

- プログラムで具体的なことをするために、必要な情報が揃っていない
 - 今回の場合だと「検索して取得した本の情報を画面に出力する」など
 - 配列の中にどのような値が入っているのか内部構造が明示されていない
- 多くの場合、mixed型の場合は具体的な操作をするための情報が欠けている
- PHPStanは値を引数として渡すときに、要求する型に対して渡す型が十分に絞り込まれていないと容赦なく叱ってくれる

このプログラムが正しいか判断できない

```
9 $result = searchBook("");
10 foreach ($result as $book) {
11     echo "書名: {$book['name']}\n";
12     echo "価格: ¥{$book['price']}\n";
13     echo "\n";
14 }
```

必要のないところにmixedは書かない

- アプリケーション実装においてmixedと適切な場面はあまりない
 - mixed = いかなる種類のデータも受け入れるということ
- ビジネスロジックというよりユーティリティ的なものが該当する
 - mixedが適切なのはvar_dump(), is_string() のような関数
 - array<mixed> を受け入れるのはcount()とかsort()とかだけ
- ジェネリクスを使えばmixedと型付けを両立できるが、今回は割愛

コードに型を付ける

型宣言があれば
足りんじゃないの？

型宣言だけでは
表現力が不足している

array “配列”の曖昧さ

別途arrayの話もしました

PHP8時代に array型と向き合う

Faced with array type in the PHP 8.x era



pixiv Inc.
USAMI Kenta

pixiv

x

2021.12.21

配列はひとつ！じゃない！！

- ユーザIDがならんだリスト [123, 456]
- ユーザIDと名前の対応表 [123 => '野比のび太', 234 => '源静香']
- ユーザを表す構造体 ['id' => 123, 'name' => '野比のび太']
- ユーザを表す構造体がならんだリスト
[
 ['id' => 123, 'name' => '野比のび太'],
 ['id' => 234, 'name' => '源静香'],
]

配列はひとつ！じゃない！

list<int>

- ユーザIDがならんだリスト [123, 456]
- ユーザIDと名前の対応表 [123 => '野比のび太', 234 => '源静香']
- ユーザを表す構造体 ['id' => 123, 'name' => '野比のび太']
- ユーザを表す構造体がならんだリスト
[
 ['id' => 123, 'name' => '野比のび太'],
 ['id' => 234, 'name' => '源静香'],
]

配列はひとつ！じゃない！

list<int>

array<int,string>

- ユーザIDがならんだリスト [123, 456]
- ユーザIDと名前の対応表 [123 => '野比のび太', 234 => '源静香']
- ユーザを表す構造体 ['id' => 123, 'name' => '野比のび太']
- ユーザを表す構造体がならんだリスト
[
 ['id' => 123, 'name' => '野比のび太'],
 ['id' => 234, 'name' => '源静香'],
]

配列はひとつ！じゃない！！

list<int>

array<int,string>

- ユーザIDがならんだリスト [123, 456]
- ユーザIDと名前の対応表 [123 => '野比のび太', 234 => '源静香']
- ユーザを表す構造体 ['id' => 123, 'name' => '野比のび太']
- ユーザを表す構造体がならんだリスト
[
 ['id' => 123, 'name' => '野比のび太'],
 ['id' => 234, 'name' => '源静香'],
]

array{id:int, name:string}

配列はひとつ！じゃない！

list<int>

array<int,string>

- ユーザIDがならんだリスト [123, 456]
- ユーザIDと名前の対応表 [123 => '野比のび太', 234 => '源静香']
- ユーザを表す構造体 ['id' => 123, 'name' => '野比のび太']
- ユーザを表す構造体がならんだリスト
[
 ['id' => 123, 'name' => '野比のび太'],
 ['id' => 234, 'name' => '源静香'],
]

array{id:int, name:string}

list<array{id:int, name:string}>

array-shapes
(またはobject-like arrays)

arrayに詳細に型を付ける

```
3 /**
4  * @param array{mode?: 'full'|'partial'} $options
5  * @return list<array{title: string, price: int}>
6  */
7 function searchBook(string $word, array $options = []): array
8 {
9     $mode = $options['mode'] ?? 'partial';
10    return [];
11 }
```

arrayに詳細に型を付ける

fullかpartialのみ

```
3 /**
4  * @param array{mode?: 'full'|'partial'} $options
5  * @return list<array{title: string, price: int}>
6  */
7 function searchBook(string $word, array $options = []): array
8 {
9     $mode = $options['mode'] ?? 'partial';
10    return [];
11 }
```


取り出してきた値の配列

```
13 $result = searchBook('', ['mode' => 'perfect']);  
14 foreach ($result as $book) {  
15     echo "書名: {$book['name']}\n";  
16     echo "価格: ¥{$book['price']}\n";  
17     echo "\n";  
18 }
```


取り出してきた値の配列

```
13 $result = searchBook('', ['mode' => 'perfect']);
14 foreach ($result as $book) {
15     echo "書名: {$book['name']}\n";
16     echo "価格: ¥{$book['price']}\n";
17     echo "\n";
18 }
```

Line	Error
13	Parameter #2 \$options of function searchBook expects array{mode?: 'full' 'partial'}, array{mode: 'perfect'} given.
15	Offset 'name' does not exist on array{title: string, price: int}.

取り出してきた値の配列

```
13 $result = searchBook('', ['mode' => 'perfect']);
14 foreach ($result as $book) {
15     echo "書名: {$book['name']}\n";
16     echo "価格: ¥{$book['price']}\n";
17     echo "\n";
18 }
```

fullかpartialなのに
perfect

Line	Error
13	Parameter #2 \$options of function searchBook expects array{mode?: 'full' 'partial'}, array{mode: 'perfect'} given.
15	Offset 'name' does not exist on array{title: string, price: int}.

取り出してきた値の配列

```
13 $result = searchBook('', ['mode' => 'perfect']);
14 foreach ($result as $book) {
15     echo "書名: {$book['name']}\n";
16     echo "価格: ¥{$book['price']}\n";
17     echo "\n";
18 }
```

fullかpartialなのに
perfect

ほんとはtitleなのに
nameでアクセス

Line	Error
13	Parameter #2 \$options of function searchBook expects array{mode?: 'full' 'partial'}, array{mode: 'perfect'} given.
15	Offset 'name' does not exist on array{title: string, price: int}.

PHPStanの 強力な型

型名について

- 型名は一般的に `array`, `int`, `string`, `bool` などすべて小文字で書く
- クラス名は慣習的に `DateTime`, `App\Book` のようなキャメルケース（先頭と単語区切りを大文字にする）が一般的だが、`book` のように小文字も定義可
- そのためツールによっては `@return never` と書くと `App\never` と解釈
- これを避けるには `@param` `@return` `@var` に代えて、
`@phpstan-param` `@phpstan-return` `@phpstan-var` にする

整数範囲型

- `int<0, 6>` や `int<1, 10>` のような範囲を指定できる
- `positive-int`, `negative-int` という型もある
 - それぞれ `int<1, max>` / `int<min, -1>` のエイリアス
- 「0以上の数」はユニオン型を組み合わせで `0|positive-int` と書く
 - たとえば「配列の要素数」にマイナスはありえないのでこの型になる
- `int` `===` `int<min, max>` `===` `positive-int|0|negative-int`

リテラル型・定数型

- "full" や "partial" のような文字列や 0, 1 のようなスカラー値そのもの
- ふつうはユニオン型と組み合わせて "full"|"partial" や 1|2|9 などと書く
- constやdefineで定義された定数・クラス定数も使える
 - SEARCH_MODE_FULL|SEARCH_MODE_PARTIAL
 - SEARCH_MODE_* でまとめて参照もできる
 - 新たな SEARCH_MODE_FUZZY が増えても変更の必要がない

key-of型・value-of型

- 定数に格納された配列のキーまたは要素
- `const MODES = ['a', 'b'];` のとき `value-of<MODES> = 'a'|'b'`
- `const MESSAGES = ['x' => 'あ', 'y' => 'ん']` のとき
 - `key-of<MESSAGES> = 'x'|'y'`
 - `value-of<MESSAGES> = 'あ'|'ん'`
- 定数の * と同じく、定数の配列を編集すれば個別の箇所を修正しなくていい

PHPDocに書ける配列の型について

- 最近では `string[]` や `Book[]` のような書かれかたは好まれない
 - JavaScript(TypeScript)のように配列/リストと連想配列 (Map/ハッシュテーブル)の区別がない
- `list<string>` や `array<string,Book>` のように書くのが望ましい
- 配列の中身に同じ構造のものが繰り返されるときは `array<...>`
配列の中身がキーごとに別のデータが格納される場合は `array{...}`
- この `<>` と `{}` のカッコの種類は瞬時に混乱しないようにしておきましょう

array-shapes (Object-like arrays)

- `array{key: type}` `key`というキーが入っている連想配列
- `array{key?: type}` `key`というキーが省略されている連想配列
- `array{0: typeA, 1: typeB}` 長さ2の配列(リスト)
 - `array{typeA, typeB}` この形状ではキーを省略して型だけを列挙できる
- `array{}` 中に何も入っていない配列(空配列)

array<Type>

- **array<Type>** 配列の全ての値はType（キーの型が何かは明示しない）
- **array<string, Type>** キーの型がstring, 値の型がType
- **array<array{key: value}>**
 - 配列の中にarray-shapeが入っている

list<Type>

- ['x', 'y', 'z'] のようにキーが0, 1, 2, ... のような連番の配列
 - PHP 8.1で追加された array_is_list() 関数が根拠
- list<Type> リストの全ての値はType
 - 現在のPHPStanは array<int, Type> のエイリアス扱い
- list<array{key: value}>
 - 配列の中にarray-shapeが入っている

non-empty-*型

- **non-empty-string**: 長さ1以上の文字列（=空文字列ではない）
- **non-empty-array**: 長さ1以上の配列（キーの型は問わない）
- **non-empty-list**: 長さ1以上のリスト

先程のコードを拡張する

```
3 const SEARCH_MODES = ['full', 'partial', 'fuzzy'];
4
5 /**
6  * @param non-empty-string $word
7  * @param array{mode?: value-of<SEARCH_MODES>} $options
8  * @return list<array{title: non-empty-string, price: positive-int}>
9  */
10 function searchBook(string $word, array $options = []): array
11 {
12     $mode = $options['mode'] ?? 'partial';
13     return [];
14 }
```


先程のコードを拡張する

検索モードのリスト

```
3 const SEARCH_MODES = ['full', 'partial', 'fuzzy'];
4
5 /**
6  * @param non-empty-string $word
7  * @param array{mode?: value-of<SEARCH_MODES>} $options
8  * @return list<array{title: non-empty-string, price: positive-int}>
9  */
10 function searchBook(string $word, array $options = []): array
11 {
12     $mode = $options['mode'] ?? 'partial';
13     return [];
14 }
```

先程のコードを拡張する

検索モードのリスト

value-of<定数>

```
3 const SEARCH_MODES = ['full', 'partial', 'fuzzy'];
4
5 /**
6  * @param non-empty-string $word
7  * @param array{mode?: value-of<SEARCH_MODES>} $options
8  * @return list<array{title: non-empty-string, price: positive-int}>
9  */
10 function searchBook(string $word, array $options = []): array
11 {
12     $mode = $options['mode'] ?? 'partial';
13     return [];
14 }
```


先程のコードを拡張する

検索モードのリスト

value-of<定数>

```
3 const SEARCH_MODES = ['full', 'partial', 'fuzzy'];
4
5 /**
6  * @param non-empty-string $word
7  * @param array{mode?: value-of<SEARCH_MODES>} $options
8  * @return list<array{title: non-empty-string, price: positive-int}>
9  */
10 function searchBook(string $word, array $options = []): array
11 {
12     $mode = $options['mode'] ?? 'partial';
13     return [];
14 }
```

価格は正の整数

先程のコードを拡張する

検索モードのリスト

value-of<定数>

空文字列を
検索禁止する

```
3 const SEARCH_MODES = ['full', 'partial', 'fuzzy'];
4
5 /**
6  * @param non-empty-string $word
7  * @param array{mode?: value-of<SEARCH_MODES>} $options
8  * @return list<array{title: non-empty-string, price: positive-int}>
9  */
10 function searchBook(string $word, array $options = []): array
11 {
12     $mode = $options['mode'] ?? 'partial';
13     return [];
14 }
```

価格は正の整数

呼出し側のコード

```
16 $result = searchBook('', ['mode' => 'full']);
17 foreach ($result as $book) {
18     echo "書名: {$book['title']}\n";
19     if ($book['price'] === 0) {
20         echo "無料商品\n";
21     } else {
22         echo "価格: ¥{$book['price']}\n";
23     }
24
25     echo "\n";
26 }
```

Line	Error
16	Parameter #1 \$word of function searchBook expects non-empty-string, "" given.
19	Strict comparison using === between int<1, max> and 0 will always evaluate to false.

呼出し側のコード

```
16 $result = searchBook('', ['mode' => 'full']);
17 foreach ($result as $book) {
18     echo "書名: {$book['title']}\n";
19     if ($book['price'] === 0) {
20         echo "無料商品\n";
21     } else {
22         echo "価格: ¥{$book['price']}\n";
23     }
24
25     echo "\n";
26 }
```

(警告)
空文字列を検索

Line	Error
16	Parameter #1 \$word of function searchBook expects non-empty-string, "" given.
19	Strict comparison using === between int<1, max> and 0 will always evaluate to false.

呼出し側のコード

```
16 $result = searchBook('', ['mode' => 'full']);
17 foreach ($result as $book) {
18     echo "書名: {$book['title']}\n";
19     if ($book['price'] === 0) {
20         echo "無料商品\n";
21     } else {
22         echo "価格: ¥{$book['price']}\n";
23     }
24
25     echo "\n";
26 }
```

(警告)
空文字列を検索

(警告)
price === 0 はありえない

Line	Error
16	Parameter #1 \$word of function searchBook expects non-empty-string, "" given.
19	Strict comparison using === between int<1, max> and 0 will always evaluate to false.

ばっちり型が
つきましたね

基本的な型の付け方

- 関数／メソッドとクラス定義に絞って詳細な型を付けていく
 - パラメータ（仮引数）
 - 戻り値
 - プロパティ
- ただしクロージャ（無名関数）はがんばらなくていい
- 基本はこれだけに絞っていけばコードに型は付く （理想論）

array{} とか
覚えなくてよくない？

そういう意見もある

DTO (Data Transfer Object)

- データを格納して持ち運ぶことを目的としたクラス
- C言語などの構造体(struct)をクラスで表現したものとも考えられる
- 典型的には単なるPOPOで実装できる
- PHP 8.0からはコンストラクタプロモーションで、異様に簡潔に定義できる
 - 現代PHPでは小さいイミュータブルなクラスをバンバン切っていく方が責務が小さく解析もしやすくなるが今回の本題ではないので割愛

PHP 8.1時代のコード

```
3 enum SearchMode: string {  
4     case FULL = 'full';  
5     case PARTIAL = 'partial';  
6 }  
7 class BookSearchOptions{  
8     public function __construct(  
9         public readonly SearchMode $mode = SearchMode::PARTIAL  
10    ) {}  
11 }
```

PHP 8.1時代のコード

検索モード

```
3 enum SearchMode: string {  
4     case FULL = 'full';  
5     case PARTIAL = 'partial';  
6 }  
7 class BookSearchOptions{  
8     public function __construct(  
9         public readonly SearchMode $mode = SearchMode::PARTIAL  
10    ) {}  
11 }
```


PHP 8.1時代のコード

検索モード

オプションは配列
ではなくクラス

```
3 enum SearchMode: string {  
4     case FULL = 'full';  
5     case PARTIAL = 'partial';  
6 }  
7 class BookSearchOptions{  
8     public function __construct(  
9         public readonly SearchMode $mode = SearchMode::PARTIAL  
10    ) {}  
11 }
```


PHP 8.1時代のコード

```
13 class Author {
14     public function __construct(
15         public readonly string $name,
16     ) {}
17 }
18
19 class Book {
20     /**
21      * @param non-empty-string $title
22      * @param non-empty-list<Author> $authors
23      */
24     public function __construct(
25         public readonly string $title,
26         public readonly array $authors,
27     ) {}
28 }
```

PHP 8.1時代のコード

著者

```
class Author {
14     public function __construct(
15         public readonly string $name,
16     ) {}
17 }
18
19 class Book {
20     /**
21      * @param non-empty-string $title
22      * @param non-empty-list<Author> $authors
23      */
24     public function __construct(
25         public readonly string $title,
26         public readonly array $authors,
27     ) {}
28 }
```

PHP 8.1時代のコード

著者

```
class Author {
14     public function __construct(
15         public readonly string $name,
16     ) {}
17 }
18
19 class Book {
20     /**
21      * @param non-empty-string $title
22      * @param non-empty-list<Author> $authors
23      */
24     public function __construct(
25         public readonly string $title,
26         public readonly array $authors,
27     ) {}
28 }
```

書籍

PHP 8.1時代のコード

著者

```
class Author {  
14   public function __construct(  
15       public readonly string $name,  
16   ) {}  
17 }  
18  
19 class Book {  
20     /**  
21      * @param non-empty-string $title  
22      * @param non-empty-list<Author> $authors  
23      */  
24     public function __construct(  
25         public readonly string $title,  
26         public readonly array $authors,  
27     ) {}  
28 }
```

書籍

コンストラクタ
プロモーション

PHP 8.1時代のコード

```
30 /**
31  * @param non-empty-string $word
32  * @return list<Book>
33  */
34 function searchBook(string $word, BookSearchOptions $options): array
35 {
36     $result = [];
37     return $result;
38 }
```


PHP 8.1時代のコード

オプション型宣言

```
30 /**
31  * @param non-empty-string $word
32  * @return list<Book>
33  */
34 function searchBook(string $word, BookSearchOptions $options): array
35 {
36     $result = [];
37     return $result;
38 }
```

PHP 8.1時代のコード

オプション型宣言

戻り値PHPDoc

```
30 /**
31  * @param non-empty-string $word
32  * @return list<Book>
33  */
34 function searchBook(string $word, BookSearchOptions $options): array
35 {
36     $result = [];
37     return $result;
38 }
```

クラス vs array

- 一般論としては何でも格納できる配列より、クラス定義したオブジェクトの方が静的解析もしやすく、実行時に意図しない値も紛れにくい
- 特にPHP8ではコンストラクタプロモーションによって定義のハードルが一気に下がり、名前付き引数によってコード上に意図も込めやすくなった
- クラスは名前を付けて定義するので、似て非なる別のクラスをいくつも作ることになるとしんどくなってくる
- 単なるデータとして名無しのarray-shapesが良いこともある

かくしてPHPに
完璧な型がつけました

めでたしめでたし



これで済むなら
PHPは型なしとか
呼ばれてない

型なしは
どこから来るの？

動的型 vs 型なし

- 型宣言しなくても、全ての値は実行時に必ず具体的な型情報を持っている
 - `get_debug_type()` や `is_string()` など動的に正しく判別できる
 - 型宣言すると
- ここで言いたい「型なし」はソースコードを見て具体的な型が定まらないもの
 - 実行してみれば「動く」とわかるが、実行するまで正しいとは判断しきれない

型なしの値

- PHPではいろんなところから型がついてない値が吹き出してくる
 - そのまま使えない `mixed`, `array<mixed>`, `string`, `string|array`, ...
- スーパーグローバル変数(`$_GET`)、PDO、`$args`、`fgetcsv()`、などなど...
- そのまま使えろと信じ込んで(=間違った値は来ないと祈って)使う 🙏
 - → 一種の割り切りではあるが、最後の手段にしたい

型を絞り込もう

PHPStan\dumpType()

dumpType()関数で型を表示

```
1 <?php declare(strict_types = 1);  
2  
3 function test(string $v): void {  
4     $n = filter_var($v, FILTER_VALIDATE_INT);  
5     \PHPStan\dumpType($n);  
6 }  
7
```


dumpType()関数で型を表示

関数に渡す

```
1 <?php declare(strict_types = 1);  
2  
3 function test(string $v): void {  
4     $n = filter_var($v, FILTER_VALIDATE_INT);  
5     \PHPStan\dumpType($n);  
6 }  
7
```

dumpType()関数で型を表示

関数に渡す

```
1 <?php declare(strict_types = 1);  
2  
3 function test(string $v): void {  
4     $n = filter_var($v, FILTER_VALIDATE_INT);  
5     \PHPStan\dumpType($n);  
6 }  
7
```

Dumped type: int|false

dumpType()関数で型を表示

関数に渡す

```
1 <?php declare(strict_types = 1);  
2  
3 function test(string $v): void {  
4     $n = filter_var($v, FILTER_VALIDATE_INT);  
5     \PHPStan\dumpType($n);  
6 }  
7
```

Dumped type: int|false

もちろんPhpStorm上
で表示できる

check & throw

```
function test(string $v): void {  
    $n = filter_var($v, FILTER_VALIDATE_INT);  
    \PHPStan\dumpType($n);  
  
    if ($n === false) {  
        throw new DomainException("{ $n }は数値文字列ではありません");  
    }  
  
    \PHPStan\dumpType($n);  
}
```

Error

Dumped type: int|false

Dumped type: int

check & throw

```
function test(string $v): void {  
    $n = filter_var($v, FILTER_VALIDATE_INT);  
    \PHPStan\dumpType($n);  
  
    if ($n === false) {  
        throw new DomainException("{ $n }は数値文字列ではありません");  
    }  
  
    \PHPStan\dumpType($n);  
}
```

Error

Dumped type: int|false

Dumped type: int

falseだったら
関数を抜けることで

check & throw

```
function test(string $v): void {  
    $n = filter_var($v, FILTER_VALIDATE_INT);  
    \PHPStan\dumpType($n);  
  
    if ($n === false) {  
        throw new DomainException("{ $n }は数値文字列ではありません");  
    }  
  
    \PHPStan\dumpType($n);  
}
```

falseだったら
関数を抜けることで

Error

Dumped type: int|false

Dumped type: int

falseが混じる
可能性を排除

assert (表明)

```
function test(string $v): void {  
    $n = filter_var($v, FILTER_VALIDATE_INT);  
    \PHPStan\dumpType($n);  
  
    assert($n !== false);  
  
    \PHPStan\dumpType($n);  
}
```

Error

Dumped type: int|false

Dumped type: int

assert (表明)

```
function test(string $v): void {  
    $n = filter_var($v, FILTER_VALIDATE_INT);  
    \PHPStan\dumpType($n);  
  
    assert($n !== false);  
  
    \PHPStan\dumpType($n);  
}
```

Error

Dumped type: int|false

Dumped type: int

falseではないと表明

assert (表明)

```
function test(string $v): void {  
    $n = filter_var($v, FILTER_VALIDATE_INT);  
    \PHPStan\dumpType($n);  
  
    assert($n !== false);  
  
    \PHPStan\dumpType($n);  
}
```

falseではないと表明

Error

Dumped type: int|false

Dumped type: int

falseが混じる
可能性を排除

PHPStanは分岐ごとに型を保持

```
if ($n > 0) {  
    \PHPStan\dumpType($n);  
} else {  
    \PHPStan\dumpType($n);  
}  
  
\PHPStan\dumpType($n);  
,
```

Error

Dumped type: int<1, max>

Dumped type: int<min, 0>

Dumped type: int

PHPStanは分岐ごとに型を保持

```
if ($n > 0) {  
    \PHPStan\dumpType($n);  
} else {  
    \PHPStan\dumpType($n);  
}  
  
\PHPStan\dumpType($n);  
,
```

Error

Dumped type: int<1, max>

Dumped type: int<min, 0>

Dumped type: int

分岐ごとに範囲が分かれ
合流すると戻る

DIコンテナ

任意の値を格納できるコンテナ

```
1 <?php declare(strict_types=1);  
2  
3 interface ContainerInterface {  
4     public function get(string $id): mixed;  
5     public function has(string $id): bool;  
6 }  
7
```

超適当に実装して

```
class Container implements ContainerInterface {  
    /** @param array<string,mixed> $values */  
    public function __construct(private array $values) {}  
  
    public function has(string $id): bool {  
        return array_key_exists($id, $this->values);  
    }  
    public function get(string $id): mixed {  
        return $this->values[$id];  
    }  
}
```


値をコンテナに詰め込む

```
$container = new Container([  
    'XXX_ENDPOINT' => 'https://example.dev/api',  
    'XXX_ACCESS_TOKEN' => 'xxxxxxxxxxxxxxxxxx',  
    HttpClient::class => new HttpClient,  
]);
```

値をコンテナから取り出す

```
$url = $container->get('XXX_ENDPOINT');  
$token = $container->get('XXX_ACCESS_TOKEN');  
$http_client = $container->get(HttpClient::class);  
$response = $http_client->send($url, [  
    'Authorization' => "Bearer: {$token}",  
]);
```


値をコンテナから取り出す

```
$url = $container->get('XXX_ENDPOINT');  
$token = $container->get('XXX_ACCESS_TOKEN');  
$http_client = $container->get(HttpClient::class);  
$response = $http_client->send($url, [  
    'Authorization' => "Bearer: {$token}",  
]);
```


値をコンテナから取り出す

```
$url = $container->get('XXX_ENDPOINT');  
$token = $container->get('XXX_ACCESS_TOKEN');  
$http_client = $container->get(HttpClient::class);  
$response = $http_client->send($url, [  
    'Authorization' => "Bearer: {$token}",  
]);
```

Error

Cannot call method send() on mixed.

Part \$token (mixed) of encapsed string cannot be cast to string.

値をコンテナから取り出す

変数すべてmixed

```
$url = $container->get('XXX_ENDPOINT');  
$token = $container->get('XXX_ACCESS_TOKEN');  
$http_client = $container->get(HttpClient::class);  
$response = $http_client->send($url, [  
    'Authorization' => "Bearer: {$token}",  
]);
```

Error

Cannot call method send() on mixed.

Part \$token (mixed) of encapsed string cannot be cast to string.

値をコンテナから取り出す

変数すべてmixed

```
$url = $container->get('XXX_ENDPOINT');  
$token = $container->get('XXX_ACCESS_TOKEN');  
$http_client = $container->get(HttpClient::class);  
$response = $http_client->send($url, [  
    'Authorization' => "Bearer: {$token}",  
]);
```

型が不定のメソッド
呼び出し

Error

Cannot call method send() on mixed.

Part \$token (mixed) of encapsd string cannot be cast to string.

assertによるインライン型付け

```
$url = $container->get('XXX_ENDPOINT');  
assert(is_string($url));  
$token = $container->get('XXX_ACCESS_TOKEN');  
assert(is_string($token));  
$http_client = $container->get(HttpClient::class);  
assert($http_client instanceof HttpClient);
```


@varによるインライン型付け

```
/** @var string $url */  
$url = $container->get('XXX_ENDPOINT');  
/** @var string $token */  
$token = $container->get('XXX_ACCESS_TOKEN');  
/** @var HttpClient $http_client */  
$http_client = $container->get(HttpClient::class);
```


if/assert vs @var

- if文/assertなどは単純な型宣言よりも詳細に型付けできる
 - 本番でassert式の内容を評価するかは設定で制御できる([マニュアル参照](#))
 - 私は無効化しない方が安全だと思います
- @varはarray-shapesなどを使って配列の構造も詳細に型を付けられる。
ただし単なるコメントなので実態に反する型も付けられてしまうので要注意

if/assert vs @var の使い分け案

- 変数を生成/取り出した直後に引数としてそのまま渡す場合は、実装側の責任とする／型宣言による自動型チェックで保証されるので @var
- DBのクエリ結果などは@varで済ませるか、hydratorライブラリを使って期待通りの構造の配列／クラスにマッピングする
- 変数を生成/取り出した直後にそのオブジェクトのメソッド呼び出しをする場合は、assert(\$var instanceof Foo) のようにチェックしておくことでエラーメッセージがわかりやすくなる

標準関数 vs 型

- 失敗したらfalseを返す標準関数が多い
- `file_get_contents(): string|false`
 - 存在しないファイルにアクセスするとWarningを発生
 - Warningなどのエラーは`set_error_handler()`で振る舞いが変わる
- `$content = file_get_contents(); assert($content !== false);`
- Safe PHP: 全ての標準関数の失敗を例外に変換してくれるライブラリ

簡潔だが型安全ではないコード

But most of us are too lazy to check explicitly for every single return of every core PHP function.

```
// This code is incorrect. Twice.  
// "file_get_contents" can return false if the file does not exists  
// "json_decode" can return false if the file content is not valid JSON  
$content = file_get_contents('foobar.json');  
$foobar = json_decode($content);
```


几帳面にチェックする実装

The correct version of this code would be:

```
$content = file_get_contents('foobar.json');  
if ($content === false) {  
    throw new FileLoadingException('Could not load file foobar.json');  
}  
$foobar = json_decode($content);  
if (json_last_error() !== JSON_ERROR_NONE) {  
    throw new FileLoadingException('foobar.json does not contain valid JSON:');  
}
```

Obviously, while this snippet is correct, it is less easy to read.

実行時／型安全なラッパー関数

```
use function Safe\file_get_contents;  
use function Safe\json_decode;  
  
// This code is both safe and simple!  
$content = file_get_contents('foobar.json');  
$foobar = json_decode($content);
```

まとめ

PHPStanは
オンラインで
型チェックできる

<https://phpstan.org/try>

 **PHPStan**

HomeTryDocumentationBlog

Try out PHPStan and all of its features here in the editor. [Learn more about PHPStan »](#)

```
1 <?php declare(strict_types = 1);
2
3 class HelloWorld
4 {
5     public function sayHello(DateTimeImmutable $date): void
6     {
7         echo 'Hello, ' . $date->format('j. n. Y');
8     }
9 }
```

Share

Level 9 ▾

☐ Strict rules

☐ Bleeding edge

☒ Treat PHPDoc types as certain

Found 2 errors

Line	Error
5	Parameter \$date of method HelloWorld::sayHello() has invalid typehint type DateTimeImmutable.
7	Call to method format() on an unknown class DateTimeImmutable.

© 2016–2022 Ondřej Mirtes



PHPStanは
完璧じゃない

期待した型が見つからない
と思ったら最小の
コードをオンラインで
チェックしましょう

バグの場合もあるし
実装してPRRを送る
チャンスかもしれない

今回スルーした型

ジェネリクス

超PHErになるう

Enjoy PHP Programming

2020-03-01

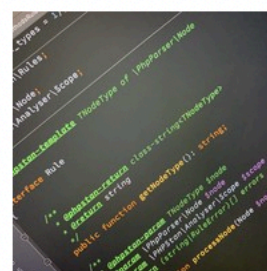
PHPDocを使ったPHPのジェネリクス

PHPStan

この記事はPHPStan開発者の[Ondřej Mirtes](#)によって2019年12月2日に書かれた記事を翻訳したものです。記事の末尾には訳者(@tadsan)の観点によるPhan, Psalm, PhpStormとの互換性についての情報も記述しています。

Generics in PHP using PHPDocs

A couple of years I wrote an impactful article on union and intersection types. It helped the PHP community to familiarize themselves with...



medium.com

[medium.com](#)

2年前、私(Ondřej Mirtes)は[ユニオン型と交差型](#)についての衝撃的な記事を書きました。PHPコミュニティがこれらの概念に馴染むのを手助けし、[PhpStormでの交差型サポート](#)につながりました。

ユニオン型と交差型の違いは開発者が認識すべき静的解析に役立つ重要な概念なので、私はその記事を書きました。今回は同様に、[PHPStan 0.12で導入された](#)ジェネリクスについて、それが何であるかを説明したいと思います。

プロフィール



zonu_exe PRO

すごいPHErになるう
<http://qiita.com/tadsan>

読者です 30

[このブログについて](#)

検索

記事を検索



最新記事

条件付き戻り値型とPHPStan 1.6.0の新機能

PHPerKaigi 2021に参加して、それから

戻り値の記憶と忘却

PHPDocを使ったPHPのジェネリクス

条件付き戻り値型

超PHPerになろう

Enjoy PHP Programming

📅 2022-04-28

条件付き戻り値型とPHPStan 1.6.0の新機能

🔖 PHPStan

この記事はPHPStan開発者の[Ondřej Mirtes](#)によって2022年4月26日にPHPStan Blogに書かれた記事を翻訳したものです。

PHPStan 1.6.0 With Conditional Return Types and More!



🌐 phpstan.org **1 user**

phpstan.org

条件付き戻り値型 (Conditional return types)

この機能の大部分は[Richard van Velzen](#)が開発しました。

PHPStanは初リリース以来、関数呼び出しで渡された引数によって様々な型を返す方法を提供してきました。いわゆる[動的戻り値型拡張\(dynamic return type extensions\)](#)は非常に柔軟です。実装できる任意のロジックによって型を解決できます。しかし、[PHPStan拡張の核心となるコンセプト](#)には学習コストがかかります。

[PHPStan 0.12](#)ではジェネリクスが導入されました。これはPHPDocの特別な記法によって動的戻り値型拡張が必要だったケースの一部をカバーしています。

プロフィール



zonu_exe **PRO**
すごいPHPerになろう
<http://qiita.com/tadsan>

✓ 読者です 30

[このブログについて](#)

検索

記事を検索



最新記事

条件付き戻り値型とPHPStan 1.6.0の新機能

PHPPerKaigi 2021に参加して、それから

戻り値の記憶と忘却

PHPDocを使ったPHPのジェネリクス

あなたが今年PHPPerKaigi 2020に参加しなければ
いけない理由

そもそも自明な型は付けたくない

```
/** @var HttpClient $http_client */  
$http_client = $container->get(HttpClient::class);
```

こういう型を付けられると嬉しい

```
/**  
 * @template T  
 * @phpstan-param string|class-string<T> $id  
 * @phpstan-return ($id is class-string<T> ? T : mixed)  
 */  
public function get(string $id);
```

現在のPHPStanの
仕様上このような型は
つけられない

Conditional return type for container #7141

New issue

Open

snapshotpl opened this issue on 28 Apr · 7 comments

snapshotpl commented on 28 Apr



Bug report

I'd like to create stub for Container with conditional return type, however looks like is not possible to do that because bug or missing feature.

Code snippet that reproduces the problem

<https://phpstan.org/r/b559c7cd-8851-4953-a7ec-0d8b5b3cc714>

Expected output

No error and correct type

Assignees

No one assigned

Labels

feature-request

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

Notifications

Customize

Unsubscribe

You're receiving notifications because you



ondrejmirtes commented on 28 Apr

Member



`class-string<T>|string` gets normalized to `string`: <https://phpstan.org/developing-extensions/type-system#type-normalization>

そのうち
(誰かが実装完了すれば)
改善する見込みはある

PHPStanの実装は
超簡単だとは言えないが
仕事でPHPやってたら
手を出せないほど難解と
いうほどではない

みなさんもチャレンジ
してみましよう

PHPStanで
楽しい型付けライフを