コードを自在に操るための PHP文法入門

Introduction to PHP syntax for mastering code manipulation

pixiv Inc.
USAMI Kenta

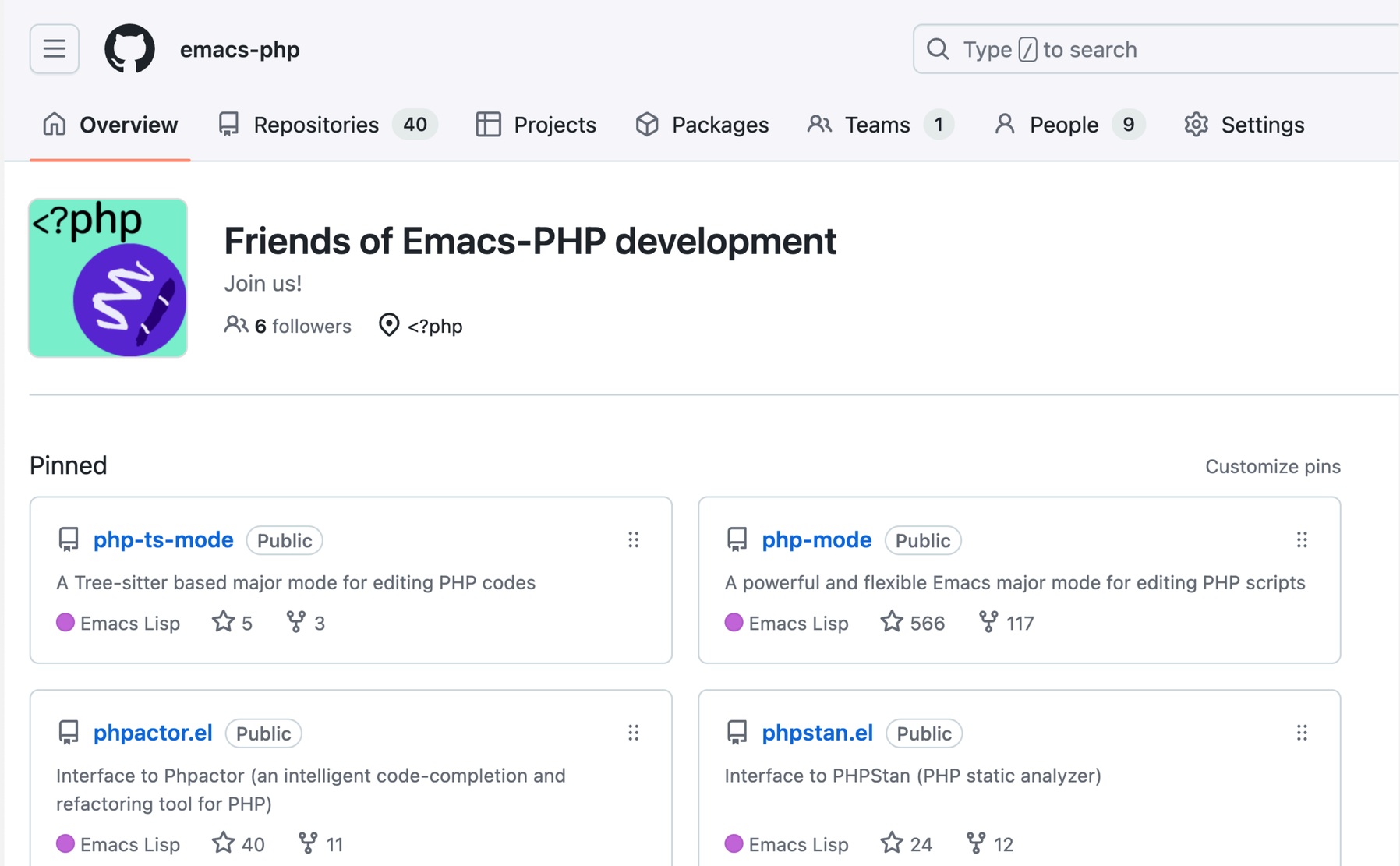
pixiv

PHPCon Kansai
2023-02-08 At Grand Front Osaka

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 Webエンジニアリングチーム PHPer
  - 2012年末から現職、APIとかCIとかいろいろなところを見つめてきました
  - 最近はピクシブ百科事典(dic.pixiv.net)も開発しています
- Emacs PHP Modeを開発しています（2017年-）
- プログラミング言語にちょっとこだわりのある素人（spcamp2010）

emacs-php



pixiv

PHPカンファレンス関西2017

pixiv

PHPカンファレンス関西2018

「ふつうのPHP」が pixiv になるまで

pixiv.inc
USAMI Kenta @tadsan

pixiv

2018-07-14 PHPカンファレンス関西 #php

x

pixiv

今回のお題

PHPカンファレンス関西2024

採択　2024/02/11 13:20〜　$room['A']　レギュラートーク（40分）

コードを自在に操るためのPHP文法入門

うさみけんた　tadsan

☆ 19

PHPのソースコードを正確に検査したり、ソースコードの一部を書き換えたいと思ったことはありませんか？

PHPにはPHP-Parserという構文解析ライブラリがあり、静的解析ツールのPHPStanやリファクタリングツールのRectorはPHP-Parserをベースにしたプラグインでソースコードを検査したり、ソースコードを書き換えたりすることができます。

しかしながら、構文木を操作するには普段何気なくPHPコードを書く以上のプログラミング言語についての知識が求められます。この発表では構文木を取り扱う前提となるプログラミング言語についての知識、PHP-Parserの構造、PHPStanとRectorそれぞれの拡張方法と実例についても紹介します。

世はまさに
静的解析時代！

静的解析

プログラムの性質を
プログラムを実行せずに
検査する技術の総称

2010年~
PhpStormリリース

pixiv inside [archive]

pixiv insideは移転しました！ » https://inside.pixiv.blog/

2016-11-11

PHP

Phan静的解析がもたらす大PHP型検査時代

こんにちは、pixivでPHPをやってるうさみです。健全なコードベースは黙っても降ってこないので、チーム全体で開発効率を高めるような改善をするのがお仕事です。

テキストエディタはmicro推しです ヾ(〃＞＜)ﾞ☆

さる11月3日に大田区産業プラザ PiOで開催されたPHPカンファレンス 2016にて~~大怪獣に蹂躙されながら~~Phanについて30分のセッション発表をいたしましたので、その内容を紹介します！

pixiv insideとは

https://inside.pixiv.blog/ に移転しました。こちらは2016年末までのアーカイブです。

読者です 234

このブログについて

月別アーカイブ

▼ 2016 (50)
- 2016 / 12 (27)
- 2016 / 11 (6)
- 2016 / 10 (1)
- 2016 / 9 (3)
- 2016 / 8 (1)

pixiv

2018年〜
PHPStanの普及

PHPStan

Home Try Documentation Blog Search ⌘ K

Meet The Next Member of Your Team!

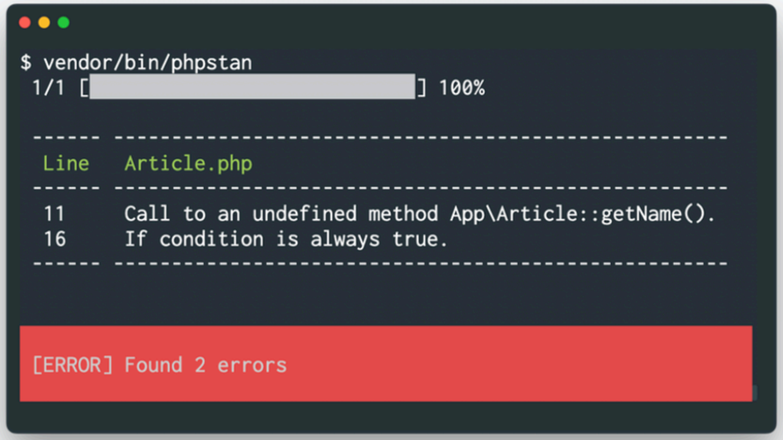
PHPStan finds bugs in your code without writing tests. It's open-source and free.

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.



pixiv

PHPStanの特徴

- PHPで書かれたオープンソースの静的解析ツール
  - ルールや型付けなどの拡張をPHPで柔軟に記述可能
- 変数スコープや型推論が実装されており、プログラムを実行しなくても
  実行時の状態をかなり正確に推測できる
  - レガシーコードを解析するために実行時情報(ランタイムリフレクション)を
    利用することもできる

PHPを正確に
理解したいと思った
ことはありますか？

PHPを正確に
理解したいと思った
ことはありますか？

正確じゃない理解
#とは

pixiv

ソースコードを
リファクタリングする
ことを考えましょう

例えば:
クラス名を変更したい

パッケージ全部ではなく
1クラスだけ移動したい

Hakone\Runner

↓

Olympic\Athlete

pixiv

シェル芸で一発

```
$ find . -type f -name "*.php" -exec gsed -I \
    's/Hakone\\\\Runner/Olympic\\\\Athlete/g' {} +
```


pixiv

シェル芸で一発

```
$ find . -type f -name "*.php" -exec gsed -I \
    's/Hakone\\\\Runner/Olympic\\\\Athlete/g' {} +
```

完

pixiv

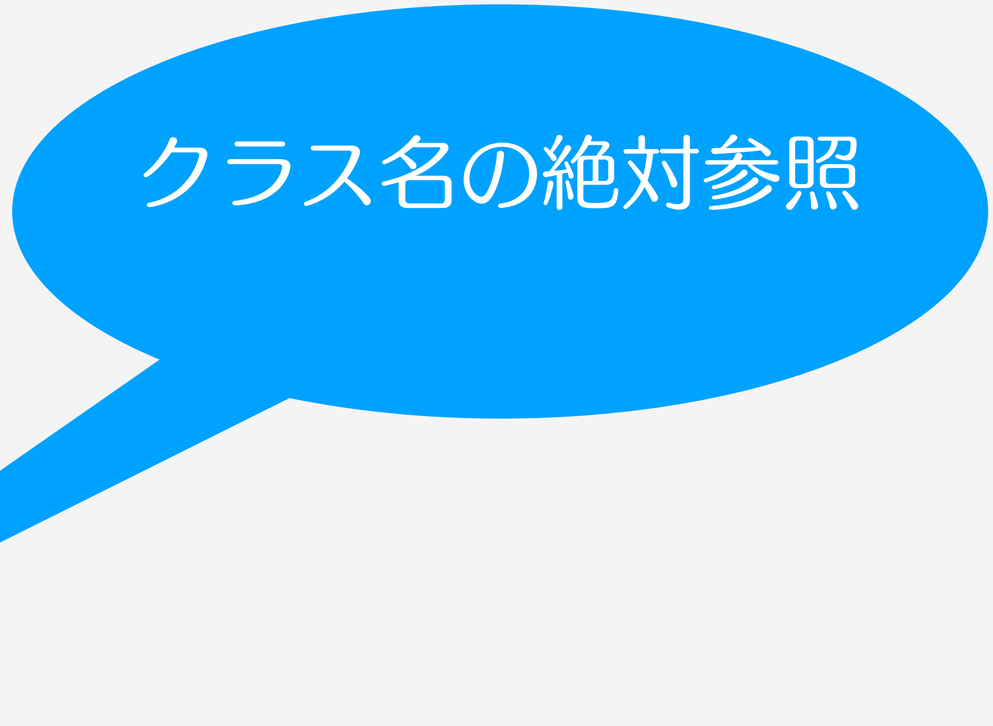
…そう簡単には
いきませんよね？

なぜ単純に置換してはいけないのか

- 単純置換でいける場合もある
  - クラス名が常にFQCN(完全修飾クラス名)で書かれているなら
- PHPのクラス名は常にFQCNが書かれるわけではない
- そもそも、ただの出力文字列としてコードに書かれたものと区別ができない
  - 例:クラスについてのHTMLドキュメントをPHPファイルに埋め込む

PHP名前空間のクラス名解決

```php
<?php
namespace Foo;

$now = new Date();

// Fatal error: Uncaught Error: Class "Foo\DateTime" not found
```




```php
<?php
namespace Foo;

$now = new \Date();

// No errors.
```


pixiv


```php
<?php
namespace Foo;
use Date;
$now = new Date();

// No errors.
```


pixiv

Qiita ログイン 新規登録

この記事は最終更新日から1年以上が経過しています。

PHP Advent Calendar 2022 15日目

@tadsan (園島 ぬぬ)

名前空間をさっくり理解する

PHP

最終更新日 2022年12月16日 投稿日 2022年12月16日

名前、つけてますか？

PHPには**namespace**(名前空間)という言語機能があります。

原初のPHPにはなかったのですが、PHP 5.3くらいからあるので、まあ平安時代には成立していたということです。それ以前の時代は `App_Http_Controllers_User` のような `_` 区切りの擬似名前空間が用いられていたことがありました。現在では `App\Http\Controllers\User` のような `\` 区切りの名前空間が利用できます。

PHPの名前空間は名字のようなものだということで説明ができます。

```php
<?php

namespace 山田;
```

ファイルの冒頭でこのような記述があるファイルは**山田家**だと思ってください。

script.php

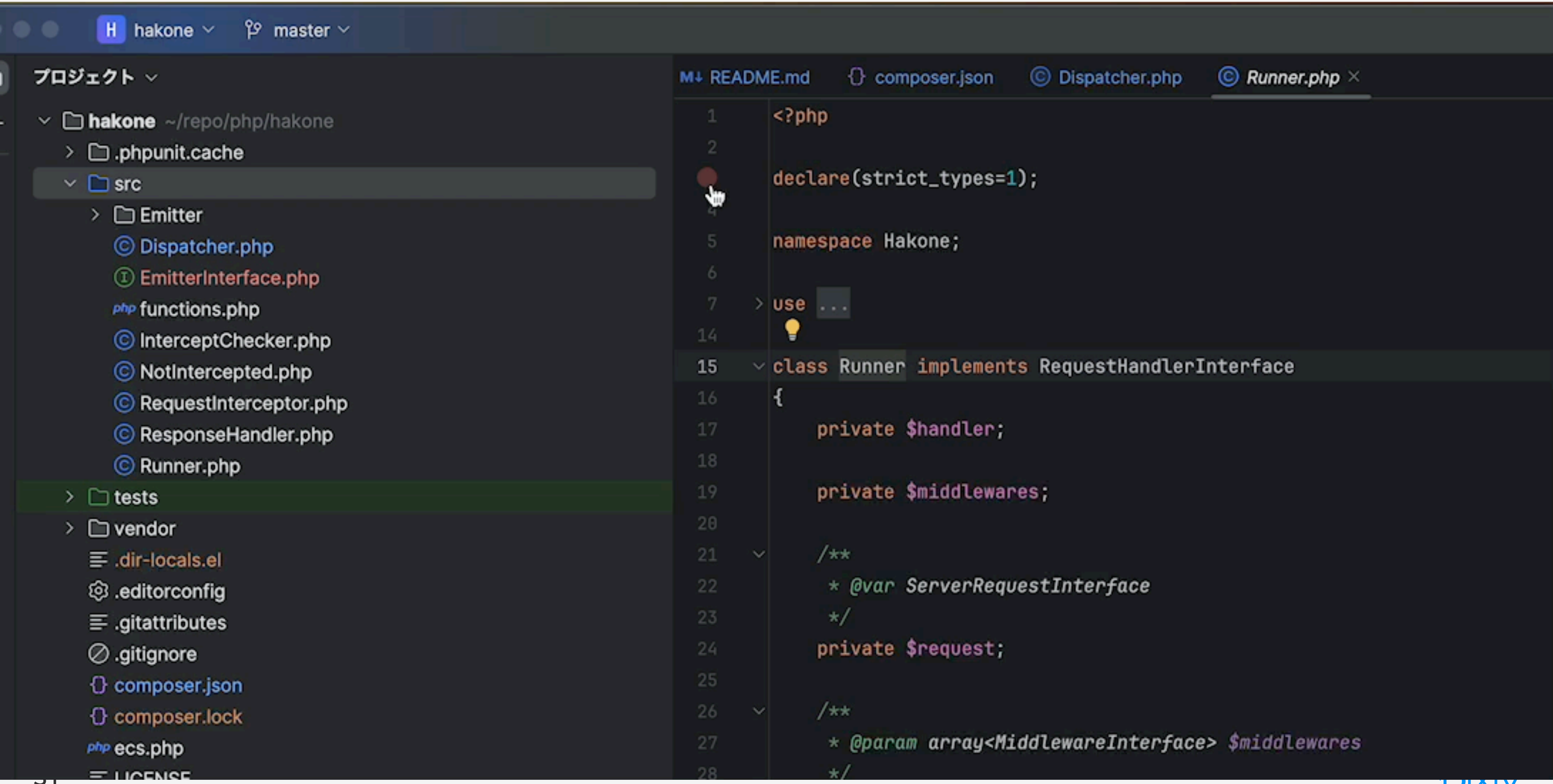
```php
<?php

namespace 山田;

$man = new 一郎();
```

おもむろに一郎を呼び付けます。*「おい一郎！」*

山田家で一郎といえば誰でしょうか。

そうです、われらが山田家の長男「`山田\一郎`さん」のことを指すに決まっています。みんな知ってるね。

もっと簡単に


hakone
master
プロジェクト
hakone ~/repo/php/hakone
.phpunit.cache
src
Emitter
Dispatcher.php
EmitterInterface.php
functions.php
InterceptChecker.php
NotIntercepted.php
RequestInterceptor.php
ResponseHandler.php
Runner.php
tests
vendor
.dir-locals.el
.editorconfig
.gitattributes
.gitignore
composer.json
composer.lock
ecs.php
README.md
composer.json
Dispatcher.php
Runner.php
<?php
declare(strict_types=1);
namespace Hakone;
use ...
class Runner implements RequestHandlerInterface
{
private $handler;
private $middlewares;
/**
* @var ServerRequestInterface
*/
private $request;
/**
* @param array<MiddlewareInterface> $middlewares
*/

PhpStorm最高！

PhpStorm最高！

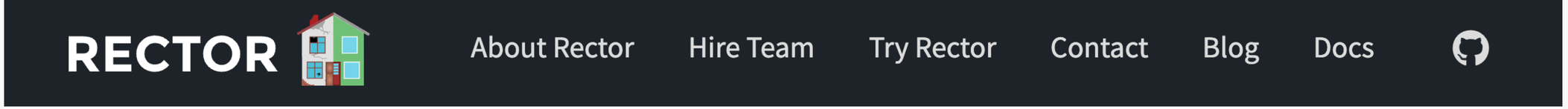
2024年、全PHPerは リファクタリングの 全フローをPhpStorm に委ねるしかないのか？

pixiv

そうではない
(もちろんPhpStormも
優れた製品だが)

ということで

Rectorを使おう



The Way You Can Finally Upgrade PHP

We help successful and growing companies to get the most of the code they already have.

Reduce maintenance cost, make feature delivery cheaper
and turn legacy code into sustainable code.

Hire Upgrade Team

Try It Online

pixiv

Rectorとは

- Tomas Votruba氏が開発する自動リファクタリングツール
  - ソースコード https://github.com/rectorphp/rector-src
- Composerで簡単に導入できる
  - PHP 7.2以上で動作し、PHPStan以外への依存関係を持たない
- 非常に数多くのルールを内蔵しているので、そのままで多くのユースケースに対応できる（PHPやPHPUnitのバージョンアップなど）

Rectorの特徴

- PHP-ParserとPHPStanをベースにカスタムルールを構築できる
  - 単純な構文木を扱えるだけでなく、PHPStanが分析する変数スコープや型情報にアクセスできるため、(コードに適切な型がついているという前提において)かなり信頼性の高いリファクタリングが可能
- あくまで構文木を扱うツールなので、出力されるコードの細かいスタイルの制御には対応していない（ほかのフォーマットツールや同作者のECSを併用）

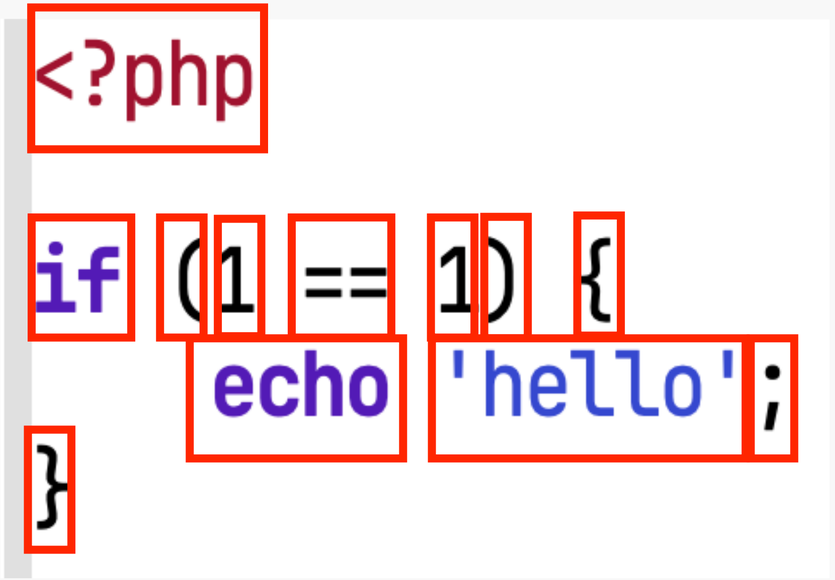
構文木(こうぶんき)ってなんだ

プログラムは
どうやって動くの？

ソースコード

```php
<?php

if (1 == 1) {
    echo 'hello';
}
```

字句解析(tokenize)



pixiv

Change language: Japanese

Submit a Pull Request Report a Bug

token_get_all

(PHP 4 >= 4.2.0, PHP 5, PHP 7, PHP 8)

token_get_all — 指定したソースを PHP トークンに分割する

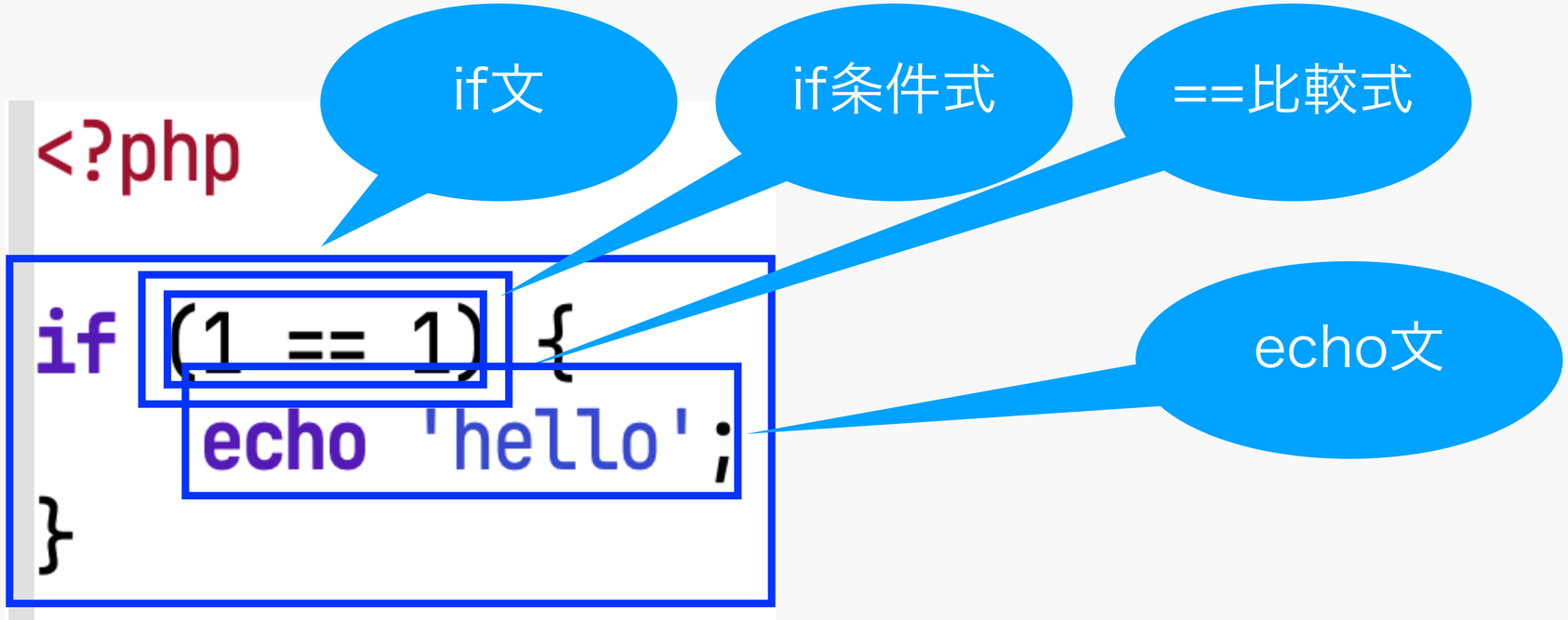
説明

```
token_get_all(string $code, int $flags = 0): array
```

token_get_all() は指定した文字列 **code** をパースし、Zend engine の 字句解析スキャナにより PHP 言語のトークンに分割します。

パーサトークンの一覧を得るには、パーサトークンの一覧 を参照するか、 あるいは token_name() でトークン値を文字列表現に 変換します。

構文解析(parsing)



pixiv

トークン列

```
[
    {"T_OPEN_TAG": "<?php\n"},
    {"T_WHITESPACE": "\n"},
    {"T_IF": "if"},
    {"T_WHITESPACE": " "},
    "(",
    {"T_LNUMBER": "1"},
    {"T_WHITESPACE": " "},
    {"T_IS_EQUAL": "=="},
    {"T_WHITESPACE": " "},
    {"T_LNUMBER": "1"},
    ")",
    {"T_WHITESPACE": " "},
    "{",
    {"T_WHITESPACE": "\n    "},
    {"T_ECHO": "echo"},
    {"T_WHITESPACE": " "},
    {"T_CONSTANT_ENCAPSED_STRING": "'hello'"},
    ";",
    {"T_WHITESPACE": "\n"},
    "}",
    {"T_WHITESPACE": "\n"}
]
```

PHPはじまったな

if文はじまったな

条件式はじまったな

こいつの前後は==比較だな

条件式終了のお知らせ

ここからブロックだな

echoだな

文字列だな

; で echo ＼(^o^)／ ｵﾜﾀ

} で if文 ＼(^o^)／ ｵﾜﾀ

README BSD-3-Clause license

PHP Parser

coverage 98%

This is a PHP parser written in PHP. Its purpose is to simplify static code analysis and manipulation.

Documentation for version 5.x (current; for running on PHP >= 7.4; for parsing PHP 7.0 to PHP 8.3, with limited support for parsing PHP 5.x).

Documentation for version 4.x (supported; for running on PHP >= 7.0; for parsing PHP 5.2 to PHP 8.3).

Features

The main features provided by this library are:

- Parsing PHP 7, and PHP 8 code into an abstract syntax tree (AST).
  - Invalid code can be parsed into a partial AST.
  - The AST contains accurate location information.
- Dumping the AST in human-readable form.
- Converting an AST back to PHP code.
  - Formatting can be preserved for partially changed ASTs.
- Infrastructure to traverse and modify ASTs.
- Resolution of namespaced names.
- Evaluation of constant expressions.
- Builders to simplify AST construction for code generation.
- Converting an AST into JSON and back.

PHP-Parser

- PHPの(元)コア開発者のNikita PopovによるPHP構文解析ライブラリ
  - PHPで書かれておりComposerで導入可能
  - PHPStan, Rector, PHPUnitなどはじめ多くのツールの基盤
- 2024年1月8日にPHP-Parser 5.0がリリースされた
  - ただし現行のPHPStan, RectorはまだPHP-Parser 4.xを利用
  - PHPStanは2024年内にリリース予定の2.0で5.xに移行予定

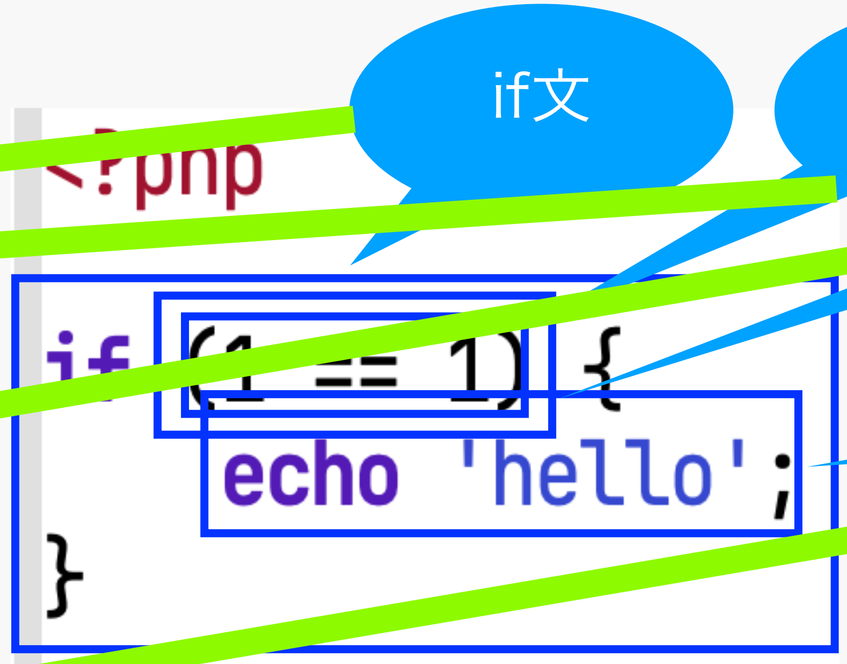
構文木 (4.x)

```
.composer/vendor/bin/php-parse ifa.php
====> File ifa.php:
==> Node dump:
array(
    0: Stmt_If(
        cond: Expr_BinaryOp_Equal(
            left: Scalar_LNumber(
                value: 1
            )
            right: Scalar_LNumber(
                value: 1
            )
        )
        stmts: array(
            0: Stmt_Echo(
                exprs: array(
                    0: Scalar_String(
                        value: hello
                    )
                )
            )
        )
        elseifs: array(
        )
        else: null
    )
)
```


pixiv

構文解析(parsing)

```
.composer/vendor/bin/php-parse ifa.php
====> File ifa.php:
==> Node dump:
array(
    0: Stmt_If(
        cond: Expr_BinaryOp_Equal(
            left: Scalar_LNumber(
                value: 1
            )
            right: Scalar_LNumber(
                value: 1
            )
        )
        stmts: array(
            0: Stmt_Echo(
                exprs: array(
                    0: Scalar_String(
                        value: hello
                    )
                )
            )
        )
        elseifs: array(
        )
        else: null
    )
)
```



pixiv

構文木 (4.x)

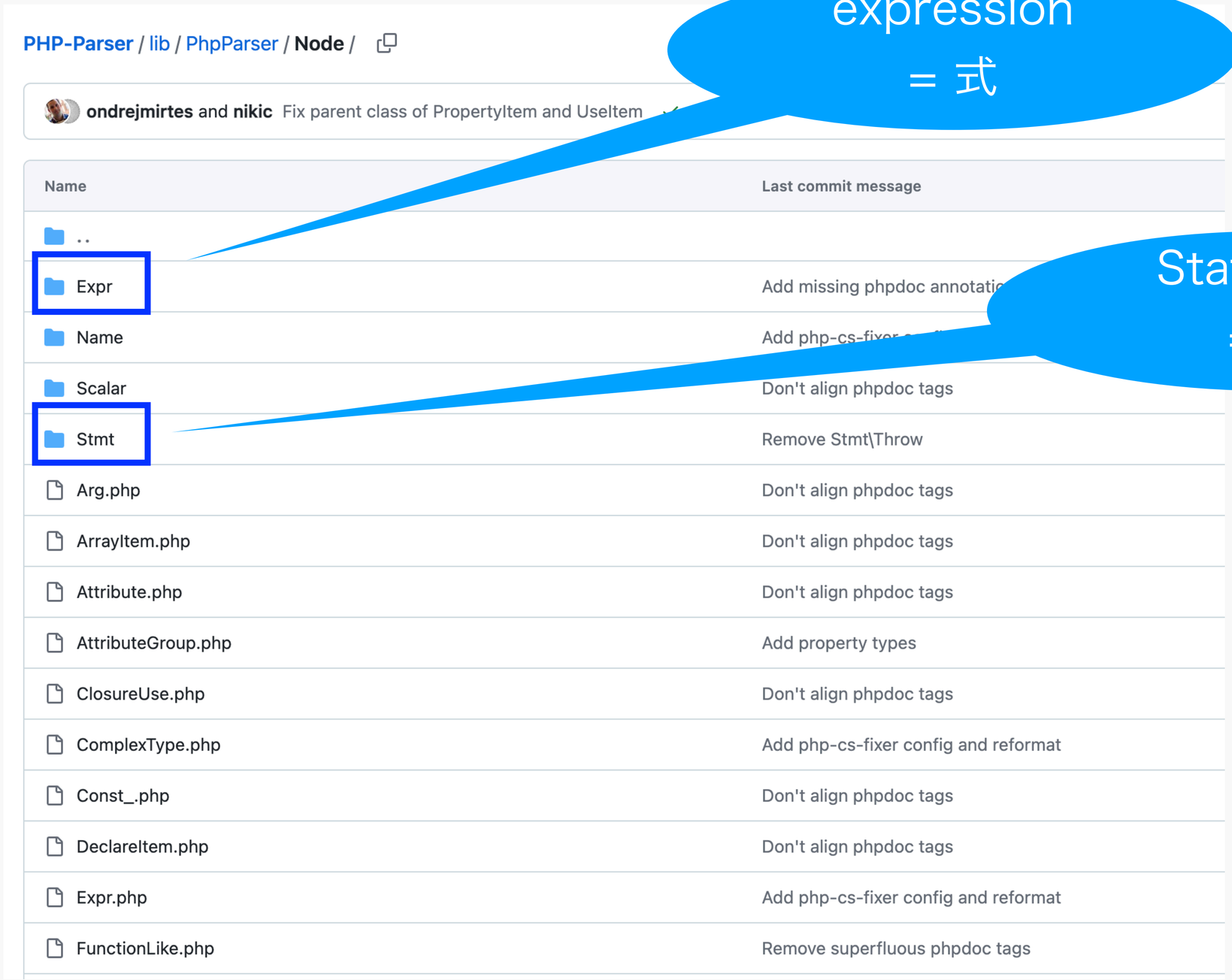
```
.composer/vendor/bin/php-parse ifa.php
====> File ifa.php:
==> Node dump:
array(
    0: Stmt_If(
        cond: Expr_BinaryOp_Equal(
            left: Scalar_LNumber(
                value: 1
            )
            right: Scalar_LNumber(
                value: 1
            )
        )
        stmts: array(
            0: Stmt_Echo(
                exprs: array(
                    0: Scalar_String(
                        value: hello
                    )
                )
            )
        )
        elseifs: array(
        )
        else: null
    )
)
```

構文木 (5.x)

```
.composer/vendor/bin/php-parse ifa.php
====> File ifa.php:
==> Node dump:
array(
    0: Stmt_If(
        cond: Expr_BinaryOp_Equal(
            left: Scalar_Int(
                value: 1
            )
            right: Scalar_Int(
                value: 1
            )
        )
        stmts: array(
            0: Stmt_Echo(
                exprs: array(
                    0: Scalar_String(
                        value: hello
                    )
                )
            )
        )
        elseifs: array(
        )
        else: null
    )
)
```

構文木 (syntax tree)

- プログラムの構造を解析した結果を木構造にしたもの
- PHPの場合は <?php foo(); bar(); のように ; 区切りでいくつも文を書けるので、トップレベル(一番外側)は配列 [] になっている
- その内側に入っているものの一個一個のことを「ノード」と呼ぶ
  - PHPのクラスとして定義されている
- PHP-ParserのノードのGitHubで見るのがわかりやすい



expression
= 式

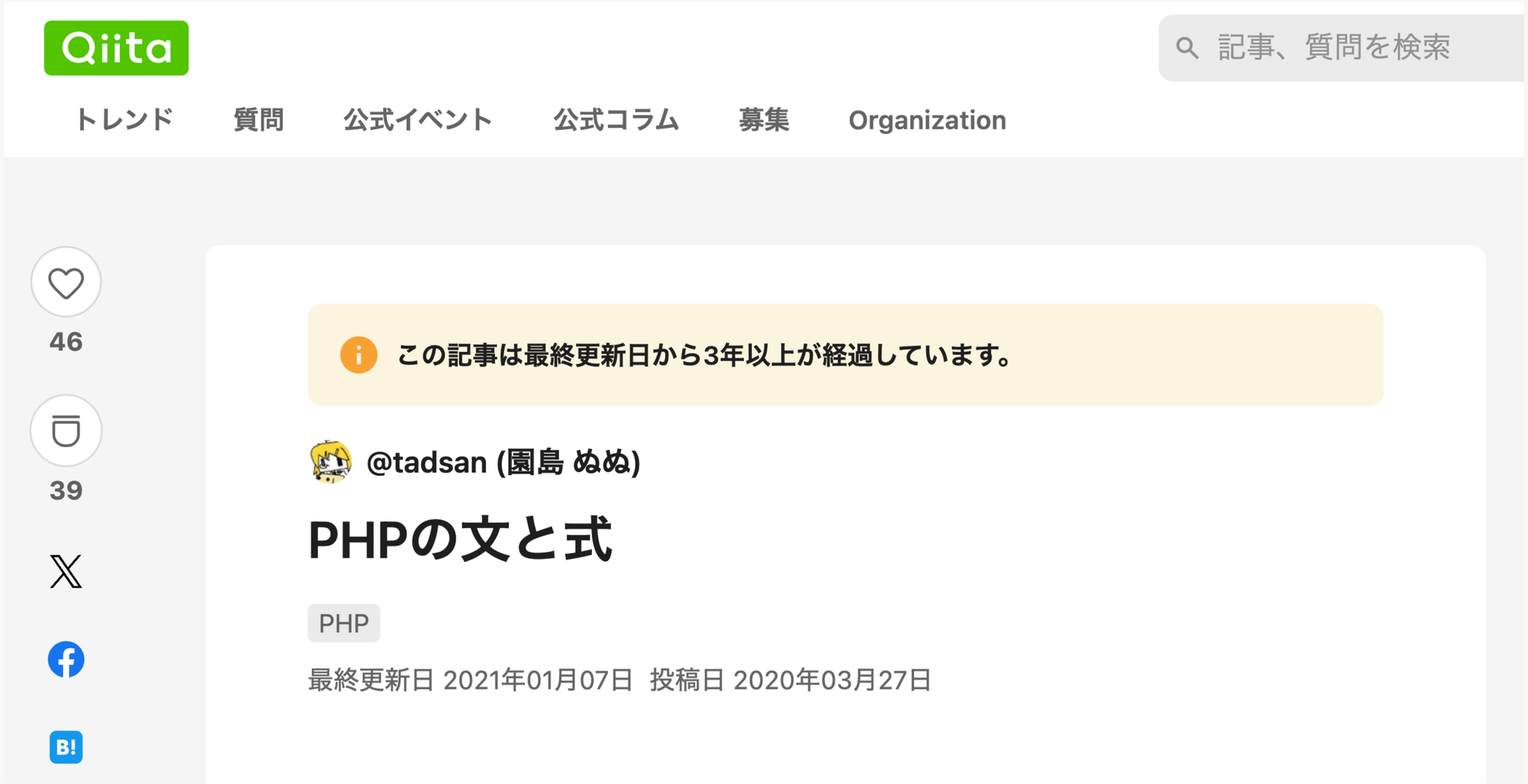
Statement
= 文

pixiv

PHP構文での echoとprintの 違いは？

式 vs 文

- 式（expressions）：演算子や関数呼び出しなど組み合わせて書ける
- 文（statement）：制御構造など、式の内側に書けないもの
  - if/switchのような制御構造やclass、functionの宣言文など
  - var_dump($v);　関数呼び出しや代入式だけの文は「式文」と呼ばれる


Qiita
記事、質問を検索
トレンド
質問
公式イベント
公式コラム
募集
Organization
46
39
この記事は最終更新日から3年以上が経過しています。
@tadsan (園島 ぬぬ)
PHPの文と式
PHP
最終更新日 2021年01月07日 投稿日 2020年03月27日
B!

pixiv

PHPにはいろんな文と式がある

キーワードのリスト

PHP のキーワード				
__halt_compiler()	abstract	and	array()	as
break	callable (PHP 5.4 以降)	case	catch	class
clone	const	continue	declare	default
die()	do	echo	else	elseif
empty()	enddeclare	endfor	endforeach	endif
endswitch	endwhile	eval()	exit()	extends
final	finally (PHP 5.5 以降)	for	foreach	function
global	goto (PHP 5.3 以降)	if	implements	include
include_once	instanceof	insteadof (PHP 5.4 以降)	interface	isset()
list()	namespace (PHP 5.3 以降)	new	or	print
private	protected	public	require	require_once
return	static	switch	throw	trait (PHP 5.4 以降)
try	unset()	use	var	while
xor	yield (PHP 5.5 以降)	yield from (PHP 7.0 以降)		

式と文と式文

- $a = 1「代入文」と呼ばれがちだが、代入そのもの式
- include_once や require_once も実は式ではなく文
- PHP 8系では fn() や match () {} など式が使いやすくなった
  - PHP 7までは「throw文」だったが、PHP 8からは「throw式」になった
- var_dump($v); 関数呼び出しや代入式だけの文は「式文」と呼ばれる

PHP-Parser 4.x → 5.0

- 実行環境要件としてPHP 7.4以上を要求
- PHP 5.x専用パーサーを削除
- 変な名前のクラスをわかりやすいようにリネーム

PHP7 vs 8: トークナイズの違い

- https://wiki.php.net/rfc/namespaced_names_as_token
  - PHP 8.0からnamespaceの一部に予約語を含められるようになった
  - その代りにトークンの間にスペース空けるのが許容されなくなった
- PHP-Parser 5.0ではこのトークナイズの結果が反映される

token_get_all()の違い

```
// PHP 7.x
[
    {"T_OPEN_TAG": "<?php "},
    {"T_NAMESPACE": "namespace"},
    {"T_WHITESPACE": " "},
    {"T_STRING": "Foo"},
    {"T_NS_SEPARATOR": "\\"},
    {"T_STRING": "Bar"},
    {"T_NS_SEPARATOR": "\\"},
    {"T_STRING": "Buz"},
    ";"
]

// PHP 8.x
[
    {"T_OPEN_TAG": "<?php "},
    {"T_NAMESPACE": "namespace"},
    {"T_WHITESPACE": " "},
    {"T_NAME_QUALIFIED": "Foo\\Bar\\Buz"},
    ";"
]
```

PHP 7.xで動いてたコード

```php
<?php

namespace Foo\Bar;

class Buz
{
    public static function f(): void
    {
        echo 'Called ', __METHOD__, PHP_EOL;
    }
}

\ Foo \ Bar \ Buz :: f ();

```


pixiv

PHP 8.xで動くコード

```php
<?php

namespace Foo\Bar\String;

class Buz
{
    public static function f(): void
    {
        echo 'Called ', __METHOD__, PHP_EOL;
    }
}

\Foo\Bar\String\Buz::f();
```

スライドここまで

隣の世界でも
構文解析時代の到来


gihyo.jp
サイトについて
カテゴリー
検索
Ruby 3.3リリース！新機能解説
Lrama LRパーサジェネレータが切り開く、Rubyの構文解析の未来
金
金子雄一郎（かねこゆういちろう）
2024-01-24
113
シェア
シンプルで強力な文法はRubyの特徴のひとつだと言われています。その文法を技術的に支えているのがパーサです。Ruby 3系のひとつの目標として、LSPやRBS、TypeProfをはじめとした各種ツールの拡充があります。それらのツールは多くの場合AST（抽象構文木）というプログラムをパースした結果を対象に解析を行います。そこでこれらのツールに対してより良いAPIを提供するべく、Rubyのパーサを刷新する動きが活発になっています。

pixiv

OSS-Vision Official
@OssVision

Ruby3.3 リリースが目前に迫りました！
リリースに向けまつもとさんから
「大構文解析時代！ Parser age」
のキーワードを頂きTシャツにアレンジしました。
多くの皆さんで Ruby3.3 リリースを祝し、関係各位に感謝しましょう。

当コレクションも、収益の一部を Rubyアソシエーションに寄付させて頂きます。

Translate post

7:57 AM · Dec 15, 2023 · **5,344** Views

Ruby Prize 2023

Prize Winner is

Kevin Newton

受賞理由／まつもとゆきひろ委員長より

Ruby言語をサポートするツールにとって、Rubyの構文解析機能は重要ですが、これまでは個別のツールで実装する必要があり、バージョンアップへの対応が困難でした。Ruby本体と共有できる構文解析器をツールから使いやすい形で提供できれば、これらの問題は解決できるはずですが、その達成は困難でした。Kevinはこの困難さに挑戦し、大きな成果をあげました。彼が開発した再起下降構文解析器である Prism は Ruby 3.3でRuby本体に組み込まれる予定です。また、この挑戦は、既存のRubyの構文解析部の急速な進化も促しました。

The Ruby parser is important for tools that support the Ruby language, but until now it has had to be implemented in a separate tool, making it difficult to keep up with version upgrades. Kevin took on this challenge and achieved great results. His recursive descent parser, Prism, will be integrated into Ruby 3.3. This challenge also prompted a rapid evolution of the existing Ruby parser.