PHP8 <<Attribute>>

2020-05-16 #phpkansai
【オンライン】関西PHP勉強会

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
  - 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

Friends of Emacs-PHP development

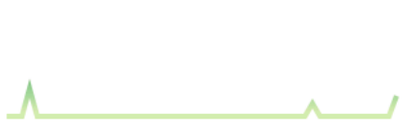
Join us!

<?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

emacs-jp

Japan http://emacs-jp.github.com/

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

reading-init.el

init.el読書会のページ

html dotfiles emacs emacs-lisp

Ruby ★ 6 1 Updated 6 days ago

helm-c-yasnippet

Helm source for yasnippet

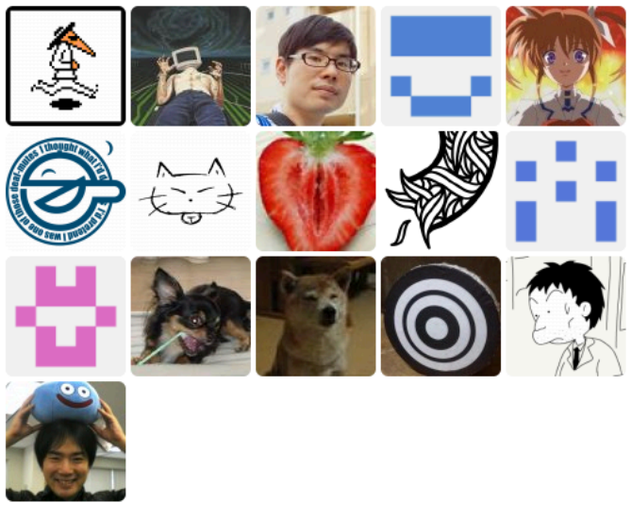
Emacs Lisp ★ 14 5 Updated on 21 Mar

replace-colorthemes

Top languages

Emacs Lisp Ruby TeX

People 37 >



近況

3月から

氏んでた

最近は

やや元気

PHPカンファレンス関西にはPHP-Parserの話をしようと応募してました

はじめに

今回紹介する内容に
よって今すぐ何かが変
わることはあまりない

少なくともコードを PHP7系から8に移行 できないと使えない

これから一年くらい
かけて対応するツールや
フレームワークも徐々に
出てくると思う

PHP8時代に心の準備だけしておこう

さて

Request for Comments

RFC

This page gives an overview of the current RFCs for PHP.

To create a new RFC, see How To Create a RFC.

Note: An RFC is effectively “owned” by the person that created it. If you want to make changes, get permission from the creator. If no agreement can be found, the only course of action is to create a competing RFC. In this case, the old RFC page should be modified to become an intermediate page that points to all the competing RFC's.

A new page in this RFC namespace will automatically be loaded with an RFC template. Customize as needed.

PHPへの変更提案

全てのPHPへの変更はここで議決

ここを観察すれば
次のPHPがわかる

ウォッチ
してますか？

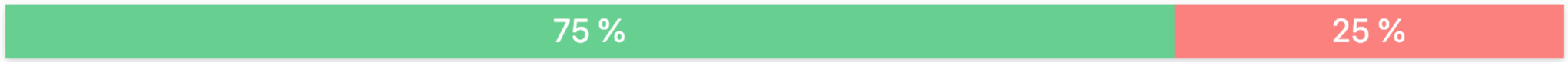
php RFC Watch

Subscribe to RFC vote notifications

CURRENTLY ACTIVE RFCS

Active **Constructor Property Promotion** PHP 8.0

Add support for declaring properties in the constructor signature?

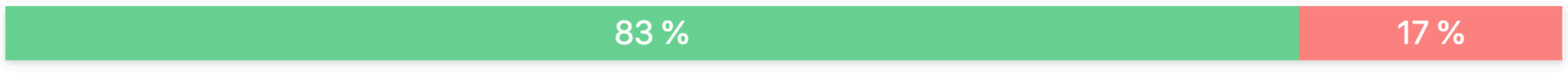


- Yes: 21
- No: 7

Total number of votes cast: 28

Active **Mixed Type v2** PHP 8.0

Add mixed as a type to be used as parameter, return and class property types?



- Yes: 44


PHP RFC Bot
762 Tweets
PHP RFC Bot
@PHPRFCBot
Keep track of new and updated @php_net RFCs. Managed by @michaeldyrynda.
Also check out php-rfc-watch.beberlei.de for voting status.
Adelaide, South Australia
Joined May 2016
1 Following
2,332 Followers
Followed by Y-Kanoh@PhpCon関西実行委員長, 虚無, and 36 others you follow
@PHPRFCBot

今年12月にリリース予定のPHP8.0に向けてさまざまな機能が提案

Feature freezeは7月だが既にPHP8向けにいろいろ決まってる

さて

PHP RFC: Attributes v2

- Version: 0.5
- Date: 2020-03-09
- Author: Benjamin Eberlei (beberlei@php.net), Martin Schröder
- Status: Accepted
- Target: 8.0
- First Published at: http://wiki.php.net/rfc/attributes_v2
- Implementation: https://github.com/php/php-src/pull/5394

Large credit for this RFC goes to Dmitry Stogov whose previous work on attributes is the foundation for this RFC and patch.

RFCは受理されたが、実装はまだマージされてない

過去何度も提案されてきたが今回の仕様でようやく受理された

そもそも

これは何だ

Introduction

This RFC proposes Attributes as a form of structured, syntactic metadata to declarations of classes, properties, functions, methods, parameters and constants. Attributes allow to define configuration directives directly embedded with the declaration of that code.

Similar concepts exist in other languages named **Annotations** in Java, **Attributes** in C#, C++, Rust, Hack and **Decorators** in Python, Javascript.

So far PHP only offers an unstructured form of such metadata: doc-comments. But doc-comments are just strings and to keep some structured information, the @-based pseudo-language was invented inside them by various PHP sub-communities.

“このRFCはクラス・プロパティ・関数・メソッド・引数および定数の宣言に対する構文的メタデータの構造化された形式として「アトリビュート」を提案します。アトリビュートによって宣言に直接埋め込まれた構成ディレクティブを定義できます。”

Similar concepts exist in other languages named **Annotations** in Java, **Attributes** in C#, C++, Rust, Hack and **Decorators** in Python, Javascript.

So far PHP only offers an unstructured form of such metadata: doc-comments. But doc-comments are just strings and to keep some structured information, the @-based pseudo-language was invented inside them by various PHP sub-communities.

“似たようなコンセプトは「**アノテーション**」としてJavaに、「**アトリビュート**」としてC#・C++・Rust・Hackに、そして「**デコレータ**」としてPython・JavaScriptにあります。”

attributeの定訳としては「属性」。直接関係ないが同じ用語としてはHTMLでも非常に重要な概念。<a href="...">

> これまでPHPは構造化されていない形式のメタデータとしてDoc-commentだけを提供しています。ただ、Doc-commentは単なる文字列であり、構造化情報を保持するための @ ベースの擬似言語がさまざまなPHPサブコミュニティによって提案されてきました。

Doc-commentは
リフレクションAPI経由で
実行時に文字列として
取得できる

実行時に影響することもあるし、しないこともある特殊なやつ

/**
 * これ
 */

```
/**
 * @return int
 */
```

```
/**
 * @phpstan-return 1|2
 */
```

```
/**
 * @deprecated
 */
```

```
/**
 * @Inject
 */
```

```
/**
 * @Inject(optional=true)
 */
```

各プロジェクトが
雰囲気で定義してる

各プロジェクト固有タグは
@phan-param みたいに
共通のタグにprefixを
つけたりアノテーションを
\ で名前空間で区切ってる

文字列を解析してるだけ
だからPHPに存在しない
名前付き引数も好き勝手
に使えるぜ！！！１１

– PHPerKaigi 2020 (Feb 9, 2020)

大雑把に言うと
IDEや静的解析のヒント
実行時メタプログラミング
に分類できる

なんとなくの風潮として
@tag はタグ、
@Anno("arg", attr=true)は
アノテーションと呼び分け

あくまで「なんとなく」で
あって、各サブコミュニティ
が勝手に実装してるだけ
なので厳密な定義はない

区別はかなり曖昧だが
PSR-19はタグカタログ
であり、アノテーション
カタログではない

どちらにせよ、PHPスクリプトに書く以上の付加情報を記述できる

あるいは実装から分離して
アノテーションとして
記述することでAOPを実現

実行時メタプログラミングの温床

例:

Go! AOP

そんなわけで

Attributeが入ると

```
/**
 * @ExampleAttribute
 */
class Foo
{
    // ...
}
```

```
<<ExampleAttribute>>
class Foo
{
    // ...
}
```

```
// この記法は不採用
@:ExampleAttribute
class Foo
{
    // ...
}
```

Doc-commentから取得できるのは文字列なので、実行時に活用するにはそれを構文解析する必要

AttributeはDoc-commentと同じように実行時に取得でき、構造化されており構文解析不要

好き勝手な記法が使えるタグ/アノテーションと違ってAttributes v2仕様は単純な関数呼び出し風構文と静的な式

静的な式 = PHPで有効な式のうちリテラルと定数と演算子式で解決できる(実行時に変わらない)

たぶんPHP-Parserから
もリフレクションと似た
APIが用意されて
簡単に取得できるだろう

アトリビュートはクラスと同じようにuseで名前解決できる

アトリビュートはクラスにマッピングし、Reflectionからインスタンスがとれる

アトリビュートクラス

```
<<PhpAttribute>>
class MyAttr
{
    function __construct($arg){
        // 特にやることがなければ
        // 実行時には何もしなくていい
    }
}
```

ある意味ではコメントと
同じで、取得されない限り
Attributeは実行に影響を
及ぼさない

先程のAttributeクラスは静的解析のためのヒントであって必須ではない

その意味でPythonのデコレータとは全然違う

アトリビュートを加味してインスタンスを生成する実行時のファクトリーまたはクラスローダーで近似させることはできる

明確な分類はできない
が、タグよりもアノテー
ション系の方がアトリ
ビュートで表現しやすい

PHPコンパイラ(処理系)のためのアトリビュートがあることが示唆されている(詳細はRFC範囲外)

PHP

やるじゃん

ほんとに？

誰のための
アトリビュート?

コンパイラアトリビュート(処理系への指示)はなんとなくわかる

ユーザーランドアトリビュートは…？

ユースケース

は…？

RFCで実例が紹介されている

これはわかる

“

```php
<?php
use Doctrine\ORM\Attributes as ORM;
use Symfony\Component\Validator\Constraints as Assert;

<<ORM\Entity>>
/** @ORM\Entity */
class User
{
```

”

これも… まあまあ

“

```
/** @ORM\Id @ORM\Column(type="integer") @ORM\GeneratedValue */
<<ORM\Id>><<ORM\Column("integer")>><<ORM\GeneratedValue>>
private $id;
```

”

これは… そうなるね

“

```
/**
 * @ORM\Column(type="string", unique=true)
 * @Assert\Email(message="The email '{{ value }}' is not a valid e
 */
<<ORM\Column("string", ORM\Column::UNIQUE)>>
<<Assert\Email(array("message" => "The email '{{ value }}' is not
private $email;
```

”

ここまでくると…

```
/**
 * @ORM\Column(type="integer")
 * @Assert\Range(
 *      min = 120,
 *      max = 180,
 *      minMessage = "You must be at least {{ limit }}cm tall to enter
 *      maxMessage = "You cannot be taller than {{ limit }}cm to enter
 * )
 */
<<Assert\Range(["min" => 120, "max" => 180, "minMessage" => "You must
<<ORM\Column(ORM\Column::T_INTEGER)>>
protected $height;
```

もうつらたん

```
/**
 * @ORM\ManyToMany(targetEntity="Phonenumber")
 * @ORM\JoinTable(name="users_phonenumbers",
 *      joinColumns={@ORM\JoinColumn(name="user_id", referencedColumnN
 *      inverseJoinColumns={@ORM\JoinColumn(name="phonenumber_id", re
 *      )
 */
<<ORM\ManyToMany(Phonenumber::class)>>
<<ORM\JoinTable("users_phonenumbers")>>
<<ORM\JoinColumn("user_id", "id")>>
<<ORM\InverseJoinColumn("phonenumber_id", "id", JoinColumn::UNIQUE)>>
private $phonenumbers;
```

DSLを喜んで覚え直したいひとは居るのか？

移行対象ではなく例として考えれば、アノテーションベースDSLの新規実装コストは多少下がってそう

とはいえライブラリは既にあるし、コードインジェクションもキャッシュ機構があるので実行時コストには影響しない

実はPHP RFC: Named Argumentsも復活してPHP8に向けて議論されそうな雰囲気があるので影響あるかもね

<<Deprecated>>

PHP RFC: <<Deprecated>> Attribute

- Version: 1.0
- Date: 2020-05-06
- Author: Benjamin Ebelei
- Status: Under Discussion
- First Published at: http://wiki.php.net/rfc/deprecated_attribute

Introduction

With the Attributes RFC accepted for PHP 8, this is a first proposal for an internal attribute hooking into the Compile cycle.

It also serves as a good prototype and first step, as usually all languages with attributes also have a Deprecated attribute in their language core.

アトリビュートの実際の利用例としての新しいRFC

trigger_error(E_DEPRECATED) 相当のコードを生成する実行エンジン組み込みのアトリビュート

言語組み込みのアトリビュートがある意義もわかるし、ユーザーランドで再現もめんどいので助かるといえば助かる

関数やメソッド以外にプロパティや定数の非推奨化もできる

が、メーリングリストでは存在意義について割と揉めてる

ついでに引数へのアトリビュートの付けかたについても議論

普通の実装

```
function f(
    $arg1,
    $arg2
){}
```

引数の非推奨化

```
function f(
    $arg1,
    <<Deprecated>>$arg2
){}
```

引数のアトリビュート
ってほんとに
必要なんですか…

そもそも名前空間 \Php\Attributes \Deprecated じゃなくていいの？

Attributesのfuture scopeの段階では名前空間がついてたのに、しれっとグローバル名前空間に移動している

Attributeが解決できない問題

/** @duplicated */
アノテーション名を
タイプミスすると
(上手に静的解析しない限り)
実行時に無視される

<<Duplicated>>
アトリビュート名を
タイプミスすると
(静的解析しない限り)
実行時に無視される

あと単純に名前空間との
相性が悪くて、IDEや
静的解析を使ってないと
useし忘れて無意味な
おまじないとして残り…

この問題はアノテーションベースAOPで結構深刻だと思うんですけど皆さんどうやって解決してるんだろ

全部のタグ/アノテーションがアトリビュートに移行されるの？

されない

と思う

いままでのタグ

```
/** @param 1|2 $arg */
function f(
    int $arg1
){}
```

アトリビュートで置換(イメージ)

```
use PHPStan\Param as param;

function f(
<<param("1|2")>>int $arg1
){}
```

IDEの入力補完があったとしても書きにくい

Attribute()

明日はどっちだ

PHP先生の次回作にご期待ください