

Language Update PHP

Learn Languages 2026



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 Platform Div > WebTechnology Team 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
 - 最近チームが再編されてインフラっぽい仕事もしてます
- 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)

コミュニティでは



- 2016年から毎年PHPカンファレンスで発表しています
 - PHPの言語仕様と構文解析・標準関数などについて
 - PHPのWebフレームワークとコンポーネントの設計・実装について
 - PHPのテストと静的解析・型について
- Emacs JPというコミュニティをやっています
- Emacs-PHPというコミュニティでPHP Modeなどをメンテしています

Lispは関数型言語 (ではない)

Lisp is (not) a functional language



pixiv Inc.
USAMI Kenta

pixiv

2025-06-15 #fp_matsuri
関数型まつり 2025

入門ラムダ計算 with λ -1グランプリ！

The Ultimate λ -1 Grand Prix:
An Interactive Intro to Lambda Calculus!



pixiv Inc.
USAMI Kenta

pixiv

x

2026-07-12 #fp_matsuri
中野セントラルパークカンファレンス



にゃんだーすわん

ソフトウェア・ハードウェア



プロフィールを編集

プロフィール

投稿

プラン

ショップ

Rigor Playground

No errors

```
1  # Rigor Playground – edit this code and see diagnostics in real time
2
3  def greet(name)      String
4    "Hello, " + name   String
5  end                  :greet
6
7  world = if rand == 0  false | true
8            "World"     "World"
9            else
10             "Ruby"      "Ruby"
11            end          "Ruby" | "World"
12
13 result = greet(world) "Hello, Ruby" | "Hello, World"
14 result.upcase         "HELLO, RUBY" | "HELLO, WORLD"
15
```

ちょっとすごいRubyの型チェッカーを作ります

編集

2026年5月25日 23:20 · 全体公開

これは自画自賛しても許されるくらいちゃんとすごい技術だと思っているのですが...

私はLispやRubyのような動的言語が大好きで、2016年くらいからPHPに型をつけながら暮らしている者です。もともとは2010～2014年くらい前までShibuya.rbやRuby会議に顔を出していたものの、仕事でPHPをやっていることもあってPHP方面に軸足を移していました。



リクエスト

pixivで作品をリクエストできます

リクエストを送る

ゆるふわEmacs

QiitaなどにEmacsの記事を書くモチベーションが上がります

¥200 /月

ゆるふわPHP

QiitaとかにPHPの記事を投稿するモチベーションが上がります

¥300 /月

Emacs+PHP

PHPとEmacsが両方そなわり最強に見える

¥1,000 /月

PHPやTypeScriptの 最新の動的言語の型検査 型推論の知見を寄せ集め

■ **Rigor** Playground WASM ● No errors

```
1  # Rigor Playground (WASM) – runs entirely in your browser.
2
3  def greet(name)                                String
4      "Hello, " + name                            String
5  end                                              :greet
6
7  you = rand == 0 ? "Ruby" : "World"             "Ruby" / "World"
8  result = greet you                             "Hello, Ruby" / "Hello, World"
9  result.upcase                                  "HELLO, RUBY" / "HELLO, WORLD"
10
11
```

<?php

PHPという言語について

- HTML+Perlっぽい変数\$、ちょっとC言語っぽい構文、JavaっぽいOOP
- RubyもPerl, Smalltalk, Eiffel, Ada, Lisp, …の構文を融合させた独自の言語を構築できていたことと対比して、PHPはメジャー言語の構文の不器用なデッドコピー感が強かった。

ここまでの進化の
歴史をまとめましょう

HTMLテンプレート

1. PHPはC言語用ライブラリ
+テンプレートエンジンだった
(mSQL付き)

+ 手続き型

2. PHP/FIとして、
ほんのりC言語/Perl風味
+HTMLテンプレートな
スクリプト言語になった

+ オブジェクト指向

3. Java風味なクラス／ オブジェクト指向機能が追加

結局PHPとは何か

Perlっぽいスクリプト

HTMLテンプレート

C言語構文
libc関数

JavaっぽいOOP



いろいろな人の期待と
欲望を取り込んでPHPは
さらに進化

Request for Comments

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

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

In voting phase

This section is for RFCs that are currently being voted on.

- [Exempt input type and value validation from BC Break policy](#) (Created 2026-03-02, Voting started: 2026-03-21, Voting ends: 2026-04-04)
- [enum SortDirection](#) (Created 2026-03-02)
- [PHP License Update](#) (Created 2024-05-09, discussion started 2025-07-10)
- [DocComments For Function Parameters](#) (Created 2026-02-21)
- [array_only_keys\(\) and array_except_keys\(\)](#) (Created 2026-02-20)



PHP 5.3

- [Closures](#)

This RFC proposes Closures and lambda functions for PHP.

- [Namespace resolution order](#)

- [RFC on what namespace separator to choose](#)

- [HEREDOC with double quotes](#)

This RFC is about allowing double quote syntax similar to single-quotes in NOWDOC as a HEREDOC syntax variant (in PHP 5.3)

- [LSB parent::/self:: forwarding](#)

Changes how parent:: and self:: work with LSB by making them forwarding.

- [call_time_pass_by_reference](#)

This RFC proposes that call_time_pass_by_reference be switched off by default in PHP 5.3.

- [New error level E_USER_DEPRECATED](#)

This RFC proposes to introduce a new userspace error level similar to E_DEPRECATED for user level deprecation warnings

- [Rounding in PHP](#)

Make round() produce predictable results on all platforms and fix some problems in the process.

- [Alternative to include/require for autoloaders](#)

This RFC aims to offer an alternative solution to the well known fopen() “hack” used in autoloaders in order to verify the existence of files inside the include path

- [FPM SAPI](#)

This RFC discusses FPM SAPI and its inclusion to the core PHP distribution.

- [FPM INI syntax](#)

This RDV discusses the replacement of FPM configuration file from XML to INI.

- [New INI Files](#)

A proposal for two new INI files which will replace the current INI's packaged with PHP. One focused on production deployments and the other focused on development deployments.

- [count_elements handler vs count\(\)](#)

Internal change to allow some optimization and add consistency.

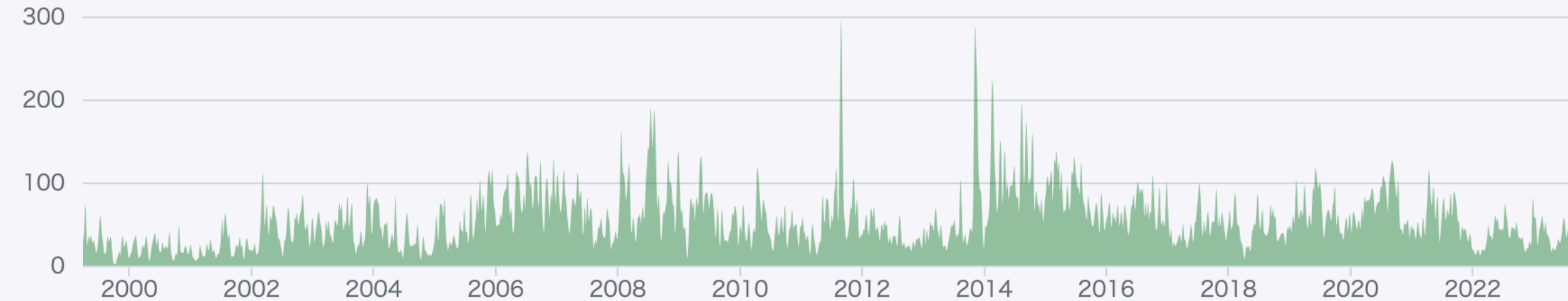


PHPは誰が開発しているのか

Apr 4, 1999 – Aug 26, 2023

Contributions: Commits ▾

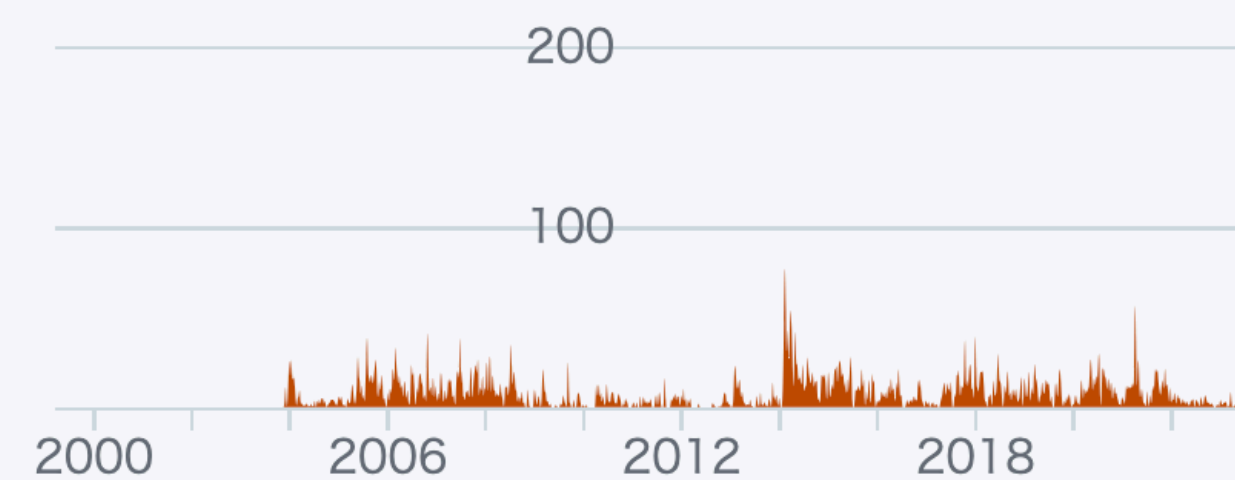
Contributions to master, excluding merge commits and bot accounts



dstogov

8,207 commits 950,566 ++ 587,128 --

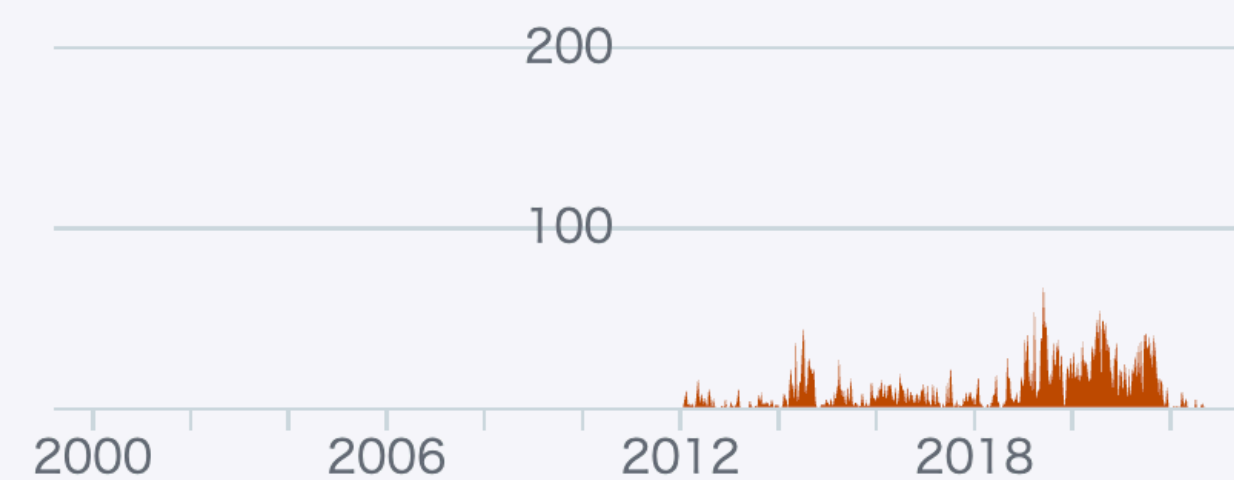
#1



nikic

5,832 commits 532,032 ++ 773,423 --

#2



weltling

4,917 commits 3,415,877 ++ 1,934,631 --

#3



helly25

4,491 commits 229,926 ++ 104,171 --

#4

Tuesday, 4 May 2021

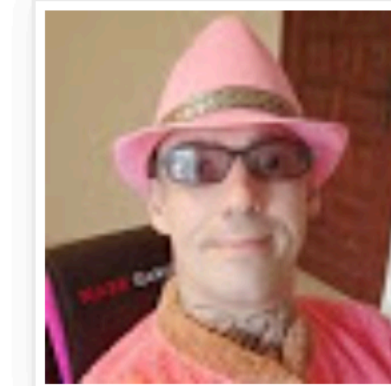
Avoiding Busses



Fig 1. A Bus

It's always been the case that there are certain parts of PHP source code that only a few people understand. The Karma system used to help us determine where a contributor could commit code in the source tree; If you had /Zend karma, you had a clue about Zend. Among those people with /Zend karma, some people understood more than others.

About Me



 Joe Watkins

[View my complete profile](#)

Blog archive

▼ 2021 (2)

▼ May (2)

[Avoiding Busses](#)

[Worthiness](#)

► 2020 (2)

► 2019 (12)

► 2018 (4)

► 2017 (1)

► 2016 (7)

► 2015 (11)

► 2014 (17)

**We support,
advance,
and develop
the PHP Language**

**The PHP
Foundation**

Our Mission

The PHP Foundation is a collective of people and organizations relying on the PHP language. Its mission is to ensure the long-term prosperity of the PHP language.

PHP 8.5.10 Released!

✓ 2025/11/20 リリース

php8.5

より賢く、より速く。未来へ向けて。

PHP 8.5 は、PHP 言語のメジャーアップデートです。URI 拡張モジュール、パイプ演算子、クローン時のプロパティ変更機能などの新機能が含まれています。

8.5 の新機能

PHP 8.5 にアップグレード

アトリビュート

PHPDoc のアノテーションの代わりに、PHP ネイティブの文法で構造化されたメタデータを扱えるようになりました。

PHP 7



```
class PostsController
{
    /**
     * @Route("/api/posts/{id}", methods={"GET"})
     */
    public function get($id) { /* ... */ }
}
```

PHP 8 [RFC ↗](#) [Doc ↗](#)



```
class PostsController
{
    #[Route("/api/posts/{id}", methods: ["GET"])]
    public function get($id) { /* ... */ }
}
```


Union 型

PHPDoc のアノテーションを使って型を組み合わせる代わりに、実行時に検証が行われる union型 をネイティブで使えるようになりました。

PHP 7



```
class Number {  
    /** @var int|float */  
    private $number;  
  
    /**  
     * @param float|int $number  
     */  
    public function __construct($number) {  
        $this->number = $number;  
    }  
}  
  
new Number('NaN'); // Ok
```

PHP 8 [RFC ↗](#) [Doc ↗](#)



```
class Number {  
    public function __construct(  
        private int|float $number  
    ) {}  
}  
  
new Number('NaN'); // TypeError
```


Match 式

`match` は `switch` 文に似ていますが、以下の機能があります:

- `match` は式なので、結果を返したり、変数に保存したりできます。
- `match` の分岐は一行の式だけをサポートしており、`break` 文は不要です。
- `match` は、型と値について、厳密な比較を行います。

PHP 7

```
switch (8.0) {  
    case '8.0':  
        $result = "Oh no!";  
        break;  
    case 8.0:  
        $result = "This is what I expected";  
        break;  
}  
echo $result;  
//> Oh no!
```

PHP 8 [RFC ↗](#) [Doc ↗](#)

```
echo match (8.0) {  
    '8.0' => "Oh no!",  
    8.0 => "This is what I expected",  
};  
//> This is what I expected
```


PHP 8.1

第一級callable

任意の関数へのリファレンスを取得できるようになりました。第一級callableと呼ばれます。

PHP < 8.1



```
$foo = [$this, 'foo'];  
  
$fn = Closure::fromCallable('strlen');
```

PHP 8.1 [RFC](#) [ドキュメント](#)



```
$foo = $this->foo(...);  
  
$fn = strlen(...);
```


PHP 8.4

括弧なしの `new MyClass()->method()`

新しくインスタンス化されたオブジェクトのプロパティとメソッドへのアクセスが、`new` 式を括弧で囲むことなくできるようになります。

PHP < 8.4



```
class PhpVersion
{
    public function getVersion(): string
    {
        return 'PHP 8.3';
    }
}

var_dump((new PhpVersion())->getVersion());
```

PHP 8.4 [RFC](#) [ドキュメント](#)



```
class PhpVersion
{
    public function getVersion(): string
    {
        return 'PHP 8.4';
    }
}

var_dump(new PhpVersion()->getVersion());
```


プロパティフック

プロパティフックは算出プロパティの機能を提供します。実態とずれやすい docblock コメントを書かずとも、IDEや静的解析ツールがネイティブに理解してくれます。さらに、対応するゲッターやセッターがそのクラスに存在するか確認することなく、確実に値の前処理・後処理を行うことができます。

PHP < 8.4

```
class Locale
{
    private string $languageCode;
    private string $countryCode;

    public function __construct(string $languageCode, string $countryCode)
    {
        $this->setLanguageCode($languageCode);
        $this->setCountryCode($countryCode);
    }

    public function getLanguageCode(): string
    {
        return $this->languageCode;
    }
}
```

PHP 8.4 [RFC](#) [ドキュメント](#)

```
class Locale
{
    public string $languageCode;

    public string $countryCode
    {
        set (string $countryCode) {
            $this->countryCode = strtoupper($countryCode);
        }
    }

    public string $combinedCode
    {
        get => \sprintf("%s_%s", $this->languageCode, $this->countryCode);
        set (string $value) {
            $this->languageCode = substr($value, 0, 3);
            $this->countryCode = substr($value, 4, 3);
        }
    }
}
```


PHP 8.4

非対称可視性

プロパティへの書き込みの範囲が、読み込みの範囲と独立して制御できるようになります。これによって、クラス外からのプロパティの変更を防ぎ値の取得のみを行えるゲッターメソッドのボイラープレートを書く必要がなくなります。

PHP < 8.4



```
class PhpVersion
{
    private string $version = '8.3';

    public function getVersion(): string
    {
        return $this->version;
    }

    public function increment(): void
    {
        [$major, $minor] = explode('.', $this->version);
        $minor++;
        $this->version = "{$major}.{$minor}";
    }
}
```

PHP 8.4 [RFC](#) [ドキュメント](#)



```
class PhpVersion
{
    public private(set) string $version = '8.4';

    public function increment(): void
    {
        [$major, $minor] = explode('.', $this->version);
        $minor++;
        $this->version = "{$major}.{$minor}";
    }
}
```


パイプ演算子

パイプ演算子を使うと、中間変数を扱うことなく複数の関数呼び出しを繋げることができます。これによってたくさんの「入れ子呼び出し」を置き換え、中から外ではなく先に向かって読むことができるようになります。

この機能の背景について詳しくは [PHP Foundation のブログ](#) をお読みください。

PHP 8.4 以前

```
$title = ' PHP 8.5 Released ';
```

```
$slug = strtolower(  
    str_replace('.', '',  
        str_replace(' ', '-',  
            trim($title)  
        )  
    )  
);
```

```
var_dump($slug);  
// string(15) "php-85-released"
```

PHP 8.5 NEW [RFC](#)

```
$title = ' PHP 8.5 Released ';
```

```
$slug = $title  
    |> trim(...)  
    |> (fn($str) => str_replace(' ', '-', $str))  
    |> (fn($str) => str_replace('.', '', $str))  
    |> strtolower(...);
```

```
var_dump($slug);  
// string(15) "php-85-released"
```

Deprecate semicolon after case in switch statement

- Author: Theodore Brown [✉ theodorejb@php.net](mailto:theodorejb@php.net)
- Implementation: <https://github.com/php/php-src/pull/19215>

It is possible to terminate case statements with a semicolon instead of the standard colon:

```
switch ($value) {  
    case 'foo';  
    case 'bar':  
    case 'baz';  
        echo 'foo, bar, or baz';  
        break;  
    default;  
        echo 'Other';  
}
```


2007年 MySQL Conference

*“We have things like protected properties.
We have abstract methods. We have all this
stuff that your computer science teacher told
you you should be using. I don't care about this crap at all.”*

–Rasmus Lerdorf

https://en.wikiquote.org/wiki/Rasmus_Lerdorf

2007年 MySQL Conference

“PHPには`protected`プロパティも
抽象メソッドもありますよ。計算機科学の
教授が「使え」と言ってるものは全部。
そんなことはクソ興味ないですけど”

–Rasmus Lerdorf (*tadsan*による超訳)

ここでいくつかの違いが簡単にわかります。波括弧がないことに気づくでしょう。

その `while` ループは、セミコロンが後に付いているので構文エラーのように見えます。でも、私はパーサーを書くのが本当に下手だったんです。今でも下手です。

だから、`if` や `endif` といったトークンがあると、ずっと読みやすくなります。つまり、どの構造を閉じているのかが正確にわかるので、波括弧は全く使いませんでした。

Rasmusの作る言語は
構文への関心が薄かった

[はてな匿名ダイアリー](#) > [anond:20100427231539](#)

< <http://anond.hatelabo.jp/20100...> | [フィクションに興味がある](#) >

2010-04-27

■ 伝説のPHP作者「Rasmus Lerdorf」名言集を聞くと嫌PHP厨がファビョる

- 今のPHPを作ったのは、何十人もの開発者ですよ。私は1人目の開発者だったに過ぎません。
- 問題を解くのが好きなだけで、プログラミングは大嫌いです。
- いかにプログラミングを避けるかを考えていたら、コードを再利用するためのツールとしてPHPができました。
- PHPは、歯ブラシみたいなものですね。毎日使うものですけど、だから何でしょう？ 誰が歯ブラシの本なんて読みたがります？
- パーザを書くのは苦手です。本当にダメなんです。今でもね。

PHP RFC: PHP License Update

- Date: 2025-07-10
- Author: Ben Ramsey, [✉ ramsey@php.net](mailto:ramsey@php.net)
- Proposed Version: Next PHP 8.x
- Status: Accepted
- First Published at: [🌐 http://wiki.php.net/rfc/php_license_update](http://wiki.php.net/rfc/php_license_update)

Introduction

PHP has a long history of confusion, concerns, and disagreements regarding its custom open source license, and the Zend Engine License, which covers the sources in the Zend/ directory, adds to this confusion and additionally complicates matters, since it is not an Open Source Initiative Approved License. This RFC proposes a pragmatic simplification to the PHP license that alleviates this confusion, preserves the copyrights owned by all PHP contributors, and grants users the same rights as the original licenses. The proposed license to accomplish this is the [🌐 Modified BSD License](#), often referred to as the 3-clause BSD license.

Proposal

PHPのライセンス変更

- いままでPHPはApache-1.0派生の独自ライセンスだった
 - 一般的な寛容な自由ソフトウェアライセンスの条件は満たしているが、OSIの承認を得られていないので微妙に使いにくいものだった
- PHP 8.6からは修正BSDライセンス(BSD-3-Clause)に変更された
 - “PHP License 4”としてBSDLの文面を指定することで、PHP License 3.0のアップデート条項を適用して軟着陸した

[Experimental Core · v0.9.6](#)

True Asynchronous inside PHP

Write sync. Run async.

Coroutines, non-blocking I/O, and structured concurrency, built into the language core. Write high-performance concurrent code with familiar functions and minimal changes.

[Get Started →](#)[How Coroutines Work](#)[Download](#)

concurrent-io.php

```
// three coroutines run at the same time
$page = spawn ? (fn() => file_get_contents($url));
$rows = spawn ? (fn() => $pdo->query($sql)->fetchAll());
$stats = spawn ? (fn() => file_get_contents($api));

// wait for all three, cancel after 2s
[$html, $data, $meta] = await_all_or_fail(
    [$page, $rows, $stats], timeout(2000));
```

[Concurrent I/O](#)[PDO Pool](#)[Threads](#)[Web Server](#)

KEY FEATURES

Production-oriented API



Coroutines

Lightweight coroutines for efficient concurrent execution.



Non-blocking I/O

fread, fwrite, file_get_contents, ob_start

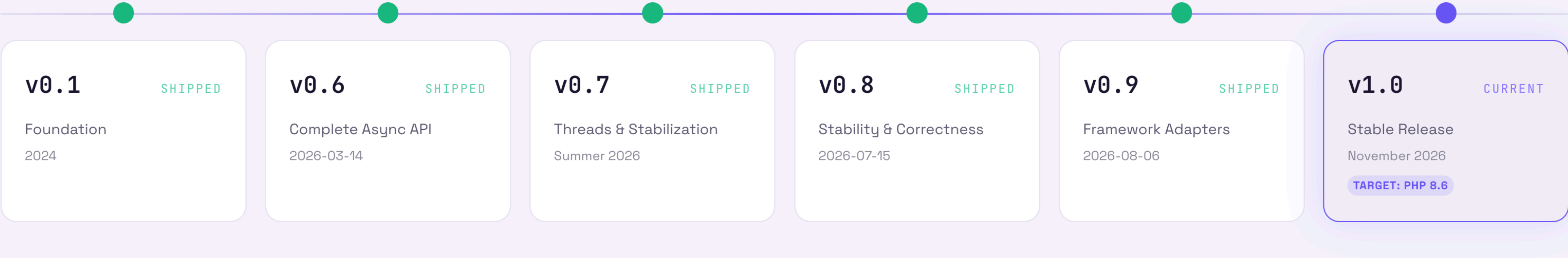


TrueAsync Server

Native HTTP/1.1, HTTP/2, and HTTP/3 web server

Roadmap

// current release: v0.9.6



[View full roadmap →](#)

True-Async

- シングルスレッドで動くPHPに「真の非同期」を入れようぜというプロジェクト
 - 全てのPHP関数は基本的に全てI/Oでブロックする
 - What Color is Your Function? 問題
 - よくある言語では Promise や async/await で非同期処理
 - True-AsyncではGoroutineに近い方法で処理できるように
- PHP 9.0とかで入るといいなあ…