

# *php*ゆるふわCI入門

A casually guide for PHP Continuous Integration



2019-03-30 PHPerKaigi 2019 Day 1  
Nerima Coconeri Hall **#phperkaigi**

# ！ お前誰よ



- うさみけんた (@tadsan) / Zonu.EXE
  - GitHub/Packagistでは id: zonuexe
- ピクシブ株式会社サービスプラットフォーム事業部
- Emacs Lisper, PHPer
  - 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

# お前誰よ



- 2013年頃から仕事でPHPを書いています
- その前はRubyをちょっとやってました
- 普段はPHP勉強会@東京などでふざけた話をします
- 今回は実用的な話をします
- 去年は前夜祭トークでふざけた話をしました
  - 今回の発表の内容とは  
あんまり関連ないです
  - 無関係ではないけど

pixiv

PHPでテストングフレ…ryを  
実装する前に知っておきたい勘所  
Tips for implementing Testing Framework in PHP.

pixiv

2018-03-09 PHPerKaigi 2018 前夜祭  
Nerima Coconeri Hall #phperkaigi



お絵描きがもっと楽しくなる場所を創る

pixiv 2018年度新卒  
エントリー受付中



# Friends of Emacs-PHP development

Join us!

<?php

-  **Repositories** 25
-  **People** 5
-  **Teams** 1
-  **Projects** 0
-  **Settings**

Find a repository...

Type: All ▾Language: All ▾

Customize pinned repositories

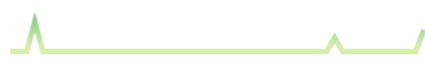
New

## 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



Top languages

Emacs Lisp PHP

## 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



Most used topics

Manage

emacs melpa php

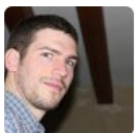
major-mode

## php-mode

A PHP mode for GNU Emacs



People 5 >



Invite someone



Search or jump to...



Pull requests

Issues

Marketplace

Explore



# emacs-jp

Japan

<http://emacs-jp.github.com/>

Repositories 18

People 37

Teams 8

Projects 0

Settings

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



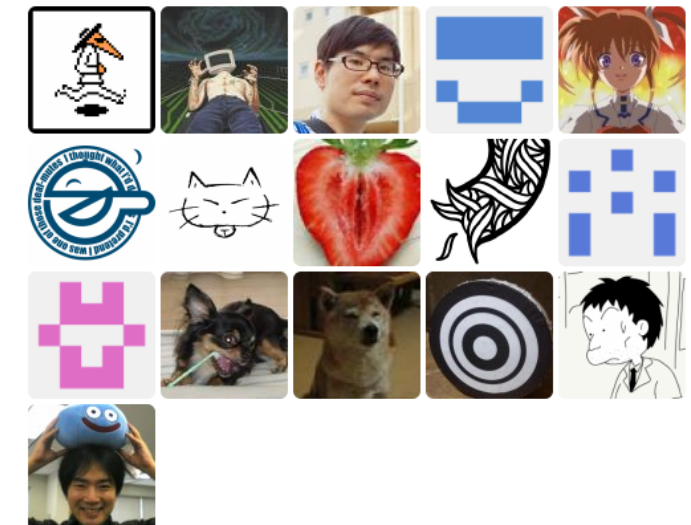
## replace-colorthemes

### Top languages

Emacs Lisp Ruby TeX

### People

37



アジエinda

CIとは何か

CIで何ができるか

CI実行環境の種類

ゆるふわにCIを始める

実際の事例について

CodeIgniter  
(Web Framework)の  
話はしません

CD (Continuous  
delivery/deployment)  
の

話もしません

個別のCI製品の  
設定の話も  
しません

CIとは何か

CIで何ができるか

CI実行環境の種類

ゆるふわにCIを始める

実際の事例について

本題の前に

今回紹介するツールと  
GitLab CIの設定をまとめて  
セットアップできるプロジェ  
クトを用意しています

<https://gitlab.com/zonuexe/phperkaigi-ci>

予め断っておきますが  
この構成では規模の大  
きなプロジェクトでは  
パフォーマンス目立ち  
ます(断言)

実プロジェクトに  
導入する際は各自  
の責任で取舍選択  
してください

あとすみません  
Windowsでは  
たぶん動きません

CIとは何か

# CIとは

単にシーアイ  
と呼びます



- Continuous Integration  
(継続的インテグレーション)の略
- もともとはXP(エクストリームプログラミング)  
コミュニティで明文化されたもの  
…らしい
- 有名な文章は2000年に書かれた

議論には登るが、実行に移されることの少ない「ベストプラクティス」がソフトウェア開発にはいっぱいある。その中でも、もっとも基本的かつ重要なものの一つが、「完全に自動化されたビルドとテストプロセス」だ。これは、プロジェクトチームが自分のソフトウェアのビルドやテストを一日に何度も行うことを可能にするものである。

–*Martin Fowler*, 継続的インテグレーション (オブラブ訳)

[http://objectclub.jp/community/XP-jp/xp\\_relate/cont-j](http://objectclub.jp/community/XP-jp/xp_relate/cont-j)

# Continuous Integration

*Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage.*

---

01 May 2006



**Martin Fowler**

**Translations:** [Portuguese](#) · [Chinese](#) · [Korean](#) · [French](#) · [Chinese](#) · [Czech](#)

Find **similar articles** to this by looking at these tags: [popular](#) · [agile](#) · [continuous delivery](#) · [extreme programming](#)

For more information on this, and related topics, take a look at my [guide page](#) for delivery.

[ThoughtWorks](#), my employer, offers consulting and support around Continuous Integration. CruiseControl, the first continuous integration server, was originally created at ThoughtWorks. ThoughtWorks Studios, our products group, created [Go](#) - a open source server for continuous integration and

## Contents

- Building a Feature with Continuous Integration
- Practices of Continuous Integration
  - Maintain a Single Source Repository.
  - Automate the Build
  - Make Your Build Self-Testing
  - Everyone Commits To the Mainline Every Day
  - Every Commit Should Build the Mainline on an Integration Machine
  - Fix Broken Builds Immediately
  - Keep the Build Fast
  - Test in a Clone of the Production Environment
  - Make it Easy for Anyone to Get the Latest Executable
  - Everyone can see what's happening
  - Automate Deployment
- Benefits of Continuous Integration
- Introducing Continuous Integration
- Final Thoughts
- Further Reading

# テスト自動化に必要なこと（引用）

---

- 全てのソースコードが格納され、誰でも最新の（そして過去の）ソースを取り出すことが可能な、ただ一つの出入口を設定すること
- ビルドプロセスを自動化して、誰でも、コマンド一つでソースからシステムを作り出すことができること
- テスト作業を自動化して、コマンド一つでいつでもテストスイートを実行できるようにすること
- 現行の実行可能モジュールで、もっとも適切であると自信を持てるようなものを、誰でも入手可能であるようにすること

–Martin Fowler, 継続的インテグレーション (オブラブ訳)

[http://objectclub.jp/community/XP-jp/xp\\_relate/cont-j](http://objectclub.jp/community/XP-jp/xp_relate/cont-j)

# テスト自動化に必要なこと (引用)

Git/GitHub

- 全てのソースコードが格納され、誰でも最新の(そして過去の)ソースを取り出すことが可能な、ただ一つの出入口を設定すること
- テスティングプロセスを自動化して、誰でも、コマンド一つでテストシステムを作り出すことができること
- テスト作業を自動化して、コマンド一つでいつでもテストスイートを実行できるようにすること
- 現行の実行可能モジュールで、もっとも適切であると自信を持てるようなものを、誰でも入手可能であるようにすること

Composer

テスト  
フレームワーク

git tag / master

–Martin Fowler, 継続的インテグレーション (オブラブ訳)

[http://objectclub.jp/community/XP-jp/xp\\_relate/cont-j](http://objectclub.jp/community/XP-jp/xp_relate/cont-j)

# テスト自動化に必要なこと (引用)

Git/GitHub

- 全てのソースコードが格納され、誰でも最新の(そして過去の)ソースコードを取り出すことが可能になる。各々の出入口を設定すること
- テスティングプロセスを自動化して、誰でも、コマンド一つでフレームワーク・システムを作り出すことができること
- テスト作業を自動化して、コマンド一つでいつでもテストスイートを実行できるようにすること
- 現行の実行可能モジュールで、もっとも適切であると自信を持てるようなものを、誰でも入手可能であるようにすること

CIの前提となる  
環境は揃ってるので  
基本的に組み合わせで  
かなりいける

Composer

git tag / master

–Martin Fowler, 継続的インテグレーション (オブラブ訳)

[http://objectclub.jp/community/XP-jp/xp\\_relate/cont-j](http://objectclub.jp/community/XP-jp/xp_relate/cont-j)

# | 銀の弾丸はない

---

- コードの規模や構造によって導入できるツール・できないツールがある
- 大規模なコードになるほど実行時間が掛かるようになり工夫が必要になってくる

CIで何ができるか

# ！ 継続的にされてると嬉しいこと

---

- いろいろあるだろうが、基本的にはコードが正常に保たれてることは確認し続けたい  
(コード上の異常発生を迅速に知る)
- そのほか属人化してる項目や忘れられがちなプロセスを盛り込めるとよい

# コードの異常を検知するには

---

- 実行してチェックする
  - 自動テスト(ユニットテスト, 受け入れテスト)
  - 人間が動作確認
- 実行せずコードの品質を検査する
  - 「QAツール」と総称される

# | 自動テスト(ユニットテスト)

---

- PHPUnit, phpspecなど
- 小さな単位でテストすることが理想だが、コードが古く結合度が高いプロジェクトではユニットテストに表現することが困難なこともある

# | 自動テスト(受け入れテスト)

---

- Codeception
- 実際にHTTPリクエストを行ってテストできるフレームワーク
- 古くテストが少ないプロジェクトではこちらの方が導入しやすいかもしれない

# | QAツール (Quality Assurance)

---

- 大まかに二種類に大別できる
  - 字句解析ベースで主にスタイルを検査するもの
  - 再帰的な型検査を行うもの

# | コーディングスタイルベース

---

- コードを字句的に解釈する
- PHPMD (PHP Mess Detector)
- PHP\_CodeSniffer & phpcbf
- PHP Coding Standards Fixer
  - 修正までやってくれるやつ

# | 型検査を含む精密な型検査

---

- 字句的な静的解析ツールはオブジェクトの未定義メソッド検出などにはできないのに対して、最近開発が活発なツールはコードパスを詳細に解析して、さまざまな情報を出してくれる

# | 型検査を含む精密な型検査

---

- それぞれ開発が活発なツール
- PHPStan, Psalm, Phan
- ぶつちやけキャラが被ってるけど、それぞれ検出してくれるポイントが違ったりする



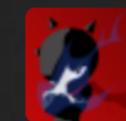
Speaker Deck

Published on Mar 10, 2018

# PHPStanによる 継続的静的解析

@hiraku

mercari



Hiraku NAKA...

11 Presentations

★ Star this Talk 7 Stars

Published in Technology

Stats Unavailable

## Share

Twitter, Facebook

Embed

Direct Link

Download PDF

PHPStanで始める継続的静的解析 #phperkaigi /php-static-analysis

by Hiraku NAKANO

Published March 10, 2018 in Technology

# | PHPStan vs Phan

---

- Phanは純粹な静的解析
  - プロジェクト全体スキャン
- PHPStanは実行時情報も利用し必要なファイルのみ解析
- 高速に結果を返したい場合はPHPStanの方が有利

# | PHPStanの制約

---

- クラスや関数・定数の定義を  
予め(または遅延)読み込みでき  
るようにする必要がある
- PSR-1(定義だけのファイルと  
副作用の処理を分離すること)  
が重要

# | PHPStanの制約

---

- ComposerとかPSR-1を考慮していない時代のPHPプロジェクトには厳しいが、ハードルを越えれば高速で頻繁なビルドが可能

# | PHPStanの効能

---

- 今回の本題とは少し離れるが、EmacsやVimでチェッカーのバックエンドとして適切に設定すると、PhpStormを凌ぐ情報量がリアルタイムで得られる

24



0



@tadsan

2018年07月31日に投稿

4867 views

編集する



# PHPで開発が捗るリアルタイムエラーチェック

PHP

Emacs

Vim

PHPStan

Pythonでも型チェックが捗ると噂をきいたのでPHPの環境構築について書きます。ちょっと眠いの  
で簡潔に... もしわからないことがあったら回答するのでコメントで聞いて<sup>1</sup>。

## 得られる利益

関数名を間違ってることに気付いたり

```
-- if (php_sapi_name() === 'cli') {
```

[得られる利益](#)[事前準備](#)[PHPStanのインストール](#)[プロジェクトの設定](#)[エディタの設定](#)[Vim](#)[Emacs](#)[Visual Studio Code](#)[まとめ](#)[脚注](#)

# Vim+ALE

```
    $a = 1;
    $b = [];

--  echo $a + $b;

    if (PHP_SAPI === 'cli') {
        try {
            $this->addPlugin('Bake');
        } catch (MissingPluginException $e) {
src/Application.php
Binary operation "+" between 1 and array() results in an error.
```

CIの実行環境  
いろいろ

# CIはいつ実行する

---

- 「たくさんやるほうが望ましい」
- 多くのCIサービスでGitHubと連携して、Pull Requestを作成したブランチにPushするたびに毎回ビルドを行うのが普通
- 結果はPRのページに表示される

# 汎用CIの実行環境

---

- テストプログラムを「自動的」かつ「継続的」に実行する必要がある
- 汎用のCIと特定用途のCIがある
- 現実的には下記のどちらか
  - 各種CI SaaSのどれかと契約
  - CI用のサーバを自前で用意

# | CI SaaS (hosted)

---

- サーバ自前管理の必要がない
- 典型的にはGitHubなどと連携
- CI側が用意したOS環境を利用するものと、Dockerイメージを利用するものがある

## ❖ プロフィール



[sue445 \(id:sue445\)](#)

フルスタックキュアエンジニアです

✓ 読者です 161

## ❖ 検索

記事を検索



## ❖ 最新記事

[Rails 5.2.2.1にしたら  
Errno::ENOENT: No such file or  
directoryのエラーになった](#)

[Dependabotの設定ファイルを  
置くようにした](#)

[gitlabci-bundle-update-mrを  
作った](#)

[nixit TECH SALONで「スペッ](#)

# くりにつき

フルスタックキュアエンジニアです

## CIマニアから見た各種CIツールの使い所

CircleCI

TravisCI

Wercker

GitLabCI

2018-12-07

社内外でちょいちょい聞かれるのでメモ。

## 目次

1. [前置き](#)
2. [GitHubを使ってる場合](#)
  1. [ライブラリを作ってる場合](#)
    1. [Travis CIを選択する理由](#)
    2. [Travis CIを選択しない理由](#)
  2. [アプリを作ってる場合](#)
    1. [CircleCIとWerckerの共通点](#)
    2. [CircleCIとWerckerの機能差異](#)
3. [GitLabを使ってる場合](#)
  1. [GitLab CIの優位点](#)
4. [Jenkinsなどを使った方がいい場合](#)
5. [追記：2018/12/8](#)

## 前置き

# | CI SaaSの制約

---

- GitHub EnterpriseやオンプレのGitLabは連携できない
- 各種CIのオンプレ版(有償)を利用することで利用できるが、サーバの調達・管理は自前で行う必要がある

# | オンプレミスのCI

---

- サーバを自前で調達・管理する必要がある
- 無料の製品ではJenkinsの牙城だった
- 自前でGitLab CEをCI機能を持ってるので連携することもできる

# | GitLabの場合

---

- GitLabには複数ある
  - SaaS版のGitLab.com
  - オンプレ版のGitLab
  - それぞれの有料・無料版
- どちらもCI機能が統合されてる

# | GitLab.com

---

- 無料でプライベートリポジトリ可
- 無料版でもCI機能が利用可能
- グループあたり月2000分まで
- 設定ファイルに書くだけで  
Dockerでビルドが実行される

# | 特定用途のCI

---

- PHP解析
  - SensioLabsInsight
  - Scrutinizer
- コーディングスタイル
  - StyleCI

御託はわかった

で、結局いろいろ準備  
しなくちゃいけなくて  
めんどくさいんでしょ？

ゆるふわに始める

CI

もっと気軽に手を  
付けられるところから  
やっていきましょう

継続して動かせる

サーバがない

さてどうするか

# テスト自動化に必要なこと（引用）

- 全てのソースコードが格納され、誰でも最新の（そして過去の）ソースを取り出すことが可能な、ただ一つの出入口を設定すること
- ビルドプロセスを自動化して、誰でも、コマンド一つでソースからシステムを作り出すことができること
- テスト作業を自動化して、コマンド一つでいつでもテストスイートを実行できるようにすること
- 現行の実行可能モジュールで、もっとも適切であると自信を持てるようなものを、誰でも入手可能であるようにすること

–Martin Fowler, 継続的インテグレーション (オブラブ訳)

[http://objectclub.jp/community/XP-jp/xp\\_relate/cont-j](http://objectclub.jp/community/XP-jp/xp_relate/cont-j)

複雑な反復作業があれば、  
一個のコマンド(スクリプト)で  
まとめてみることから始める

ところで

みなさんSyntaxErrorのPHP  
スクリプトを本番に反映して  
事故ったことはありますか？

私はたくさんあります



Syntax Errorになる  
ファイルを確実に  
捕捉してみましよう

```
git ls-files '*\*.php' |  
xargs -I{} php -l {}
```

Windowsにはxargsが  
ない(たぶん)のでPHPで

<https://gist.github.com/zonuxe/a513afd1d6933b3cc5cca0eb866f2834>

ファイルが数十個単位ならすぐ  
に終るが、数千単位になると  
全数の検査はかなり  
時間がかかるので注意

弊社での事例に  
ついて

# 「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

ファイル数 6264

行数 974419

毎回全ファイルを  
チェックするのは  
さすがにづらい

# | デプロイ時に常にgit tagをつける

---

```
tag=release/$(date +"%Y.%m.%d.%H.%M.%S").$USER  
git tag $tag  
nohup git push origin $tag &>/dev/null &
```

- デプロイスクリプトで、デプロイ成功したら日時と作業者名を含むタグを生成してoriginにpushする

# 最新のリリースタグとの差分をとる

---

```
if [ "${1:-}" = "" ]
then tag=$(git tag | grep -E '^release/' | tail -n 1)
else tag="$1"
fi

# phpファイルが無いときにエラーにしたくないので、 `|| true` で無理矢理正常終了にする
TARGET_FILE_LIST="$(git diff "$tag" --name-only --diff-filter=ACMR | grep -E '.php$' || true)"
```

- これで最新デプロイと手元の差分があるファイル抽出できる

# ！ デプロイ時にphp -lを行う

---

- 差分のあるファイル全部に対してチェックを実施し、一つでもエラーのファイルがあったらデプロイ処理を中断する
- 中断されるとtagはつかないので、差分が隠れることはない

# | MergeRequestと二種類のビルド

---

- PHPUnit
- PHPStanおよび独自Lint
- それぞれ実行時間は1分ちよつと

# | PHPStan

---

- PHPStan本運用のためbootstrap処理の軽いリファクタリングをした
- `_autoload_for_static_analysis.php`
- この作業によってPhpactorという別ツールも利用可能に

# | PHPUnit

---

- 直列実行で10分前後かかっていた
- テストが依存するMySQLを複数起動し、PHPUnitのプロセスを並列実行することで1分程度で完了

# | masterブランチへのテスト

---

- MRのテストを無視してmasterにマージされたときの防波堤
- この段階で失敗したテストはSlackで報告される
- 時刻に依存したテストのランダムフェイルがたまに報告される

# | 実行環境

---

- GitLab CI + Jenkins
  - 以前はPHPStanの警告はGitLabのMRへのコメントで報告
  - 現在はCI実行はAWSまたはJenkinsで、それぞれの結果をGitLabのパイプラインに統合

まとめ

# ！ どのようなCIを導入するか

---

- プロジェクトの現状・性質による
  - テストが書ける密結合度か
  - 定義ファイルは分離されてるか
  - (同名の関数や定数が複数定義されてるとはまる)

発表時に書き  
そびれたこと

# ！ 小姑CI

---

- コーディングスタイルのような重箱の隅は人間がコードレビューで小うるさく責めるより、Botが鉄の心で指摘した方が角が立たない
- PHP-CS-Fixerやphpcbfのようなコマンド一撃で修正されるようにすれば手間もかからない

# | どんな場合も静的解析を導入すべきか

---

- 普通の既存プロジェクトをPHPStanやPhanで解析すると異常な量の型の不整合の警告が出るのが普通
- 多すぎる警告を目のあたりにすると、人間は無力感到苛まれる  
(学習性無力感)

# 警告を深刻に受け止めない

---

- そのプロダクトが実際は動いてるのであれば、現状をあまり深刻に感じる必要はない
- 警告は記述の不整合や曖昧な点を教えてくれるだけなので、「警告を減らす」ことに価値がある

# | 注目する警告を減らす

---

- Phanであればmasterでのログ出力とブランチでの出力が減る(増えない)ことだけに注目する
- PHPStanであれば、ブランチで変更したファイル(と、そのクラス/関数を使ったファイル)だけを対象にしてみる

解析に怯えすぎず  
ご安全に