型なき言語に付ける型

Type in untyped languages

#virtual_study_group

バーチャル空間勉強会#0

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

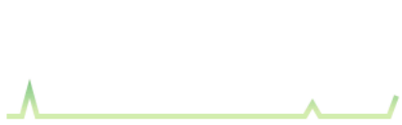
Join us!

<?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

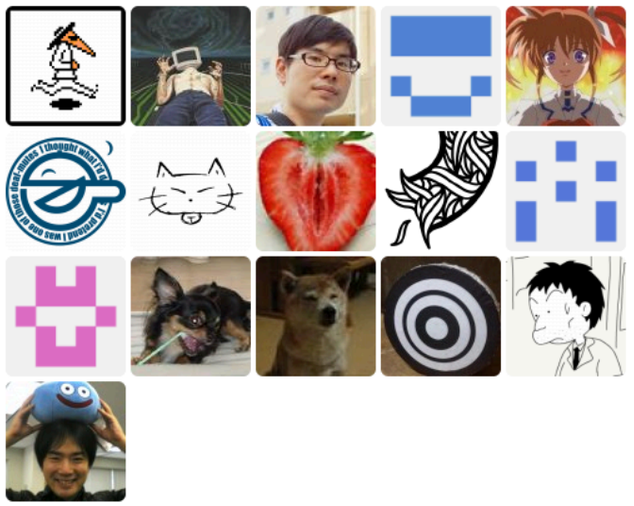
Emacs Lisp ★ 14 5 Updated on 21 Mar

replace-colorthemes

Top languages

Emacs Lisp Ruby TeX

People 37 >



近況

PIXIV TECH FES.
テックフェス

— BE CREATIVE

2020.02.17 MON

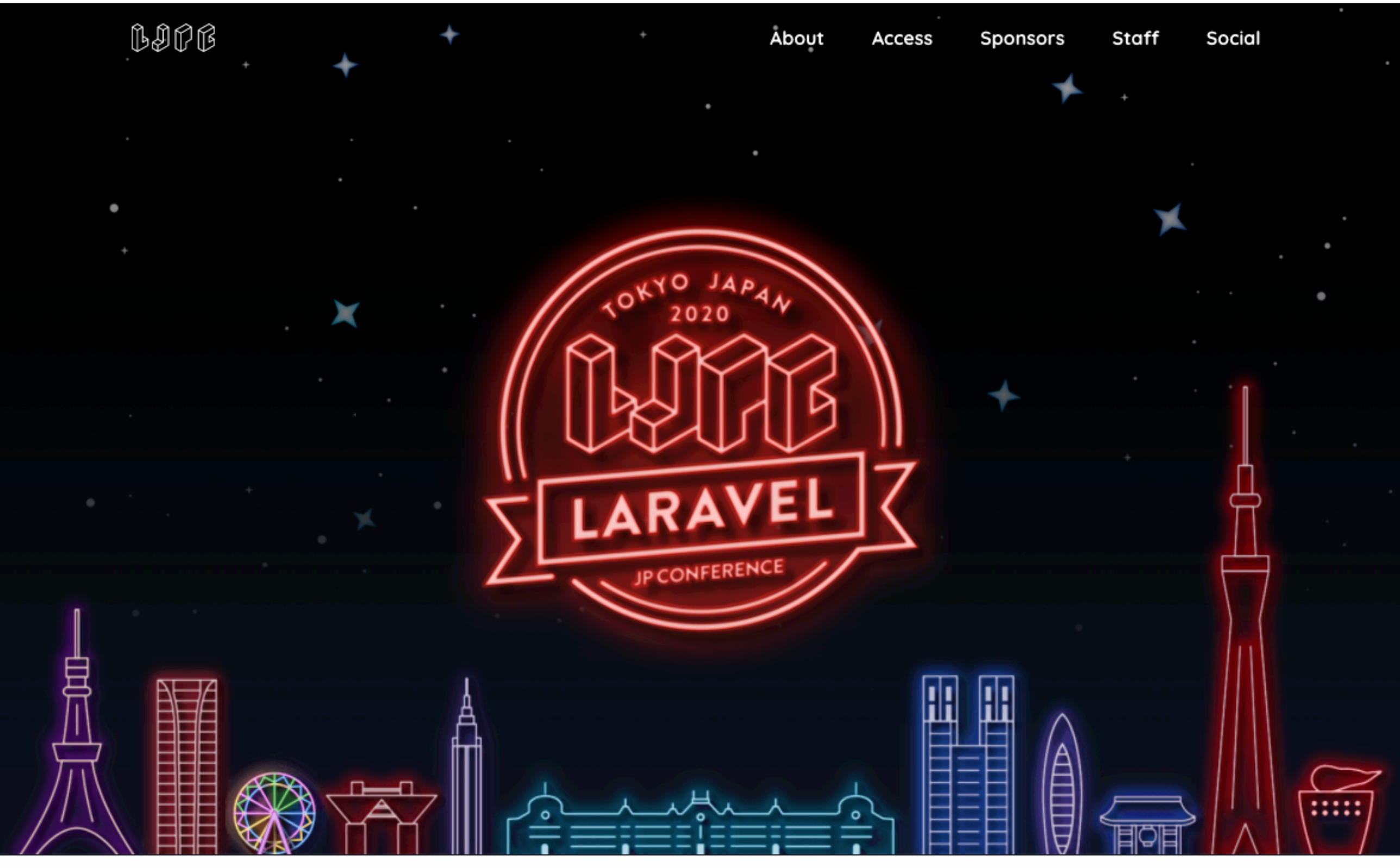
本イベントは完全招待制です。

tadsan / 宇佐美健太

「ふつうのPHP」がpixivになるまで前史

2007年に開設されたイラストSNS「pixiv」は当初からフレームワークなしのPHPで構築されています。改善事例などについては過去のPHPカンファレンスなどで紹介してきましたが、今回は私が入社する2012年以前のpixivで既に実践されていた、外部であまり類を見ないテクニックをいくつか紹介いたします。

話しました


TOKYO JAPAN
2020
LARAVEL
JP CONFERENCE

採択 ショートセッション(15分)

速習Composer

Laravel JP Conference 2020

 うさみけんた tadsan

現在のLaravelは近年の多くのPHPフレームワークと同様にComposerを基盤として構成されており、Packagistに登録された多数のPHPライブラリと簡単に相互運用できるようになっています。

この発表ではComposerの基本機能およびLaravelとComposerの関係、そしてComposerの設定方法などをまとめて解説します。

Laravel JP Conference 2020 開催中止のご案内

2020-02-21 18:00:00

Laravel JP Conference 2020 代表のblue-goheimochiです。

誠に残念ではございますが、表題の通りLaravel JP Conference 2020の開催を中止させていただくことに決定いたしました。

2月18日(火)に「Laravel JP Conference 2020の新型コロナウイルス感染症(COVID-19)への対応に関するお知らせ」をアナウンスさせていただきました。

上記アナウンス以降引き続き、新型コロナウイルス感染症(COVID-19)の国内の動向を慎重に注視しつつスタッフ内で議論を進めておりました。しかし、社会情勢の変化や所属企業によりイベントの参加制限が設けられる等、参加者およびスピーカーのみなさまのご参加が困難になる可能性があること、適切な安全確保が困難なことなどを理由に開催中止を決断させていただきました。

仕方がないので全
然違う話をします

本日のお題

型、ついて
ますか？

型については結構
雑な印象で見解を
語るひとが多い

俺も俺も

型があるから変更に強い
というひとも居れば、
型がないから変更に強い
というひとも居る

型

よくわからん

私もだ

intとboolとか

でしょ

型システム入門 プログラミング言語と型の理論

著者 ：Benjamin C. Pierce 著、住井 英二郎 監訳、遠藤 侑介 訳、酒井 政裕 訳、今井 敬吾 訳、黒木 裕介 訳、今井 宜洋 訳、才川隆文 訳、今井 健男 訳

定価 ：7,480円 （本体6,800円＋税）

判型 ：B5

頁 ：528頁

ISBN ：978-4-274-06911-6

発売日 ：2013/03/26

発行元 ：オーム社

読むがよい

“

第 1 章

はじめに

1.1　計算機科学における型

現代のソフトウェア工学において、多種多様な形式手法は、システムが期待された動作の（明示的または暗黙の）仕様に照らして正しく振る舞うことを保証する手助けになるものと評価されている。広範に及ぶ形式手法の一端は、Hoare 論理、代数的仕様記述言語、様相論理、表示的意味論といった強力な枠組みである。これらは、ソフトウェアの多種多様な正しさを記述できるが、利用するのはしばしば煩雑で、プログラマ側にかなり高度な知識を要求する。一方、それよりずっと控えめであるが、控えめであるゆえに、自動検査器をコンパイラやリンカ、あるいはプログラム解析器に埋め込むことができ、背景となっている理論に馴染みのないプログラマでも使えるような技術がある。このような**軽量形式手法**のよく知られた例に、チップ設計や通信プロトコルといった有限状態システムのエラーを探索するツールである**モデル検査器**と呼ばれるツールがある。別の現在普及しつつある例としては**実行時モニタリング**と呼ばれる一連の技術があり、これらはシステムの一部が仕様通りに動いていないことを動的に検知する。しかし、今のところそれらより遥かに普及していて、最も確立された軽量形式手法は、本書で中心的に扱う**型システム**である。

「型システム」とは何か、プログラミング言語の設計者や開発者たちが日常的に口にする際の意味合いをすべて反映し、なおかつ十分に意味のある具体的な定義を与えるのが難しいのは、複数の大きなコミュニティが共有する多くの用語と同様である。しかし、次のような定義はできよう。

”

–Benjamin C. Pierce 著／住井英二郎 監訳／遠藤侑介・酒井政裕・今井敬吾・黒木裕介・今井宜洋・才川隆文・今井健男 訳 『型システム入門　プログラミング言語と型の理論』（オーム社、978-4-274-06911-6、2013年）サンプル

> **型システム**とは、プログラムの各部分を、それが計算する値の種類に沿って分類することにより、プログラムがある種の振る舞いを起こさないことを保証する、計算量的に扱いやすい構文的手法である。
>
> 幾つか補足すべき点がある。まず、この定義は型システムを、**プログラム**に関する推論のためのツールだとみなしている。このような用語の使い方は、本書が、プログラミング言語で用いられる型システムに関するものであるという方向性を反映している。より一般的には、**型システム**（あるいは**型理論**）という用語は、論理学や数学、哲学の、もっと広範な研究分野に言及するものである。この意味での型システムは、1900 年代初頭、Russell のパラドックス [424] など、数学の基盤を揺るがした論理的パラドックスを回避する手段として、最初に形式化された。20 世紀を通じて、型は論理学、特に証明論（Gandy [167] や Hindley [209] を見よ）において標準的に用いられる道具となり、また哲学や科学の言語として浸透していった。この分野における主要な成果として、原点である Russell の**分岐階型理論** [493]、Ramsey による**単純階型理論** [394] （これは Church の単純型付きラムダ計算 [100] の基礎となった）、Martin-Löf の**構成的型理論** [299; 301]、Berardi、Terlouw、Barendregt による**純粋型システム** [43; 458; 40] などがある。

–Benjamin C. Pierce 著／住井英二郎 監訳／遠藤侑介・酒井政裕・今井敬吾・黒木裕介・今井宜洋・才川隆文・今井健男 訳 『型システム入門　プログラミング言語と型の理論』（オーム社、978-4-274-06911-6、2013年）サンプル

(一章はサンプルとしてPDFが無料でダウンロードできます)

正しい、完璧に

正しいのだが

最初から

用語多すぎ

説明の抽象度が
高すぎて現実との
ギャップ大きすぎ

この1章の内容を解きほぐして説明するだけで何時間も話せそう

そのほかの型とかプログラミング言語論の本を読むと数式みたいなのが出てきてワケワカラン

型について
ちょっと知りたい
だけなのに

世の中の人と型

型を使っているが
思案の対象として
向き合ってないひと

特定の言語の型とだ
け向き合ってる人

抽象概念の型と向き合ってるひと

いろんな
背景の人が居る

動的言語を使っていると
静的型付き言語利用者か
ら冷ややかに見られがち

やーい

型なし型なし

型はあるんだ！

実行時にあるんだ！

そういうことをまじめに
主張すると憐れみを混え
た目で見られはじめる

真の型なし言語も複数あるが今回は割愛しました

型
わかりたくないですか

このトークに

結論はない

実際の言語の型から
いきましょう

C言語

よくわからんけど型がだるい

```c
#include <stdio.h>

long add(long a, long b) {
    return a + b;
}

int main(void) {
    unsigned int x = 1, y = 2;

    printf("%d + %d = %d\n", x, y, add(x, y));
}
```

私たちFラン情報科学生は
情弱なのでまともな説明
もないカビの生えた教科
書でC言語を叩き込まれる

私の入門者時代のC
言語理解は曖昧
なので曖昧なままで
次の話に進めます

さておき

短くなってべんり

```
// JavaScript

function add(a, b) {
    return a + b;
}

const x = 1, y = 2;

console.log(`${x} + ${y} + ${add(x, y)}`);
```

もっと短くなってべんり

```ruby
# Ruby

def add(a, b)
  a + b
end

x = 1; y = 2

puts "#{x} + #{y} + #{add x, y}"
```

そういう俺たちが動的言語に触れるとこうなる

型とか

要らんやろw

そうだな

見ればわかるだろ

```ruby
# Ruby

# to_sオブジェクトが生えてたら何でもいい
def show(obj)
  puts obj.to_s
end
```

常識的に考えて

```ruby
# Ruby

# 足し算できるものならなんでもいいし
# ジェネリクスとかなくてもいいw
def add(a, b)
  a + b
end

def show(obj)
  puts obj.to_s
end
```

引数が少ないうちはな…

```ruby
# Ruby

user = User.findBy(id: params[:user_id])
```

引数が増えてくると

```
# Ruby

obj = palpunte(
  user_id,
  ????,
  ????,
  ????,
)

p obj
```

実装を直接見に
行く機会が増える

引数が増えてくると

```ruby
# Ruby

# 実装を見ないと期待するオプションに何を渡せばいいのか
def palpunte (id, foo, bar, buz)
    p [id, foo, bar, buz]
end
# というか見てもよくわからない
```

素朴に書くと引数がわからん

```ruby
# Ruby

# 実際にはキーワード引数だけで表現できる
def palpunte (id: nil, **options)
    raise TypeError if id.nil?

    foo ||= "default foo"
    bar ||= "default bar"
    buz ||= "default buz"

    p [id, foo, bar, bar]]
end
```

だんだん型チェックが増えてく

```ruby
# Ruby

  raise TypeError unless foo.instance_of?(String)
  raise TypeError unless bar.instance_of?(String)
  raise TypeError unless buz.instance_of?(String)
```

いわゆる動的言語にも型が付く流れ

Flowtype TypeScript

PEP484

mypy

Ruby3 rbi Sorbet Steep

PHP型宣言
PHPDoc
PhpStorm,Phan

身体は型を求める

型が付いていない
ということは
どういうことか

可能性は無限大

```php
<?php

/**
 * @param mixed $a 可能性は無限大
 * @param mixed $b
 * @return mixed
 */
function add($a, $b)
{
    return $a + $b;
}
```

仮引数宣言がない言語

```php
<?php

function add()
{
    $args = func_get_args();

    return array_pop($args)
        + array_pop($args);
}
```

無限大じゃ

困るんだよ

とはいえ+できる
値という前提がある
ので案外困らない

ほんとに？

PHPの演算子の暗黙的型変換

```php
<?php

error_reporting(0);

var_dump(add(1, 2)); // => int(3) 想定通り
var_dump(add(1.0, 2.0)); // => float(3.0) 許容範囲

var_dump(add('1', '2')); // => int(3) 許容範囲

var_dump(add('', '')); // => int(0) ?????
var_dump(add(1, 2, 3)); // => int(3) ?????
var_dump(add([], [])); // => [] えっ?????

var_dump(add(PHP_INT_MAX, 1));
// => float(9.223372036854E+18)
```

実行時に
型を調べよう

いわゆる動的言語は
実行時に型があるの
でそれを検査できる

JavaScriptでは typeof 演算子で 型名文字列がとれる

Python type()
Ruby Object#class

PHPではis_int(),
is_string()などの関数
(gettypeは使いにくい)

可能性を絞り込もう

```php
<?php

/**
  * @param mixed $a 可能性は無限大
  * @param mixed $b 可能性は無限大
  * @return mixed
  */
function add($a, $b)
{
    if (!is_int($a)) {
        throw new TypeError('Error');
    }
    if (!is_int($b)) {
        throw new TypeError('Error');
    }

    // 実装するときに int 以外の値が渡ってくる可能性を考えなくて良い
    return $a + $b;
}
```

実行時に変な値に対して処理が続行する可能性は避けられる

例外で処理が
停止するので

PHPとかいう静的型付きプログラミング言語

静的型付きPHP

```php
<?php declare(strict_types=1);

// このメソッドを実装するときに int 以外の値が
// 渡ってくる可能性は考えなくてよい
function add(int $a, int $b): int
{
    return $a + $b;
}
```

静的＝
実行しなくても
型が決まってる状態

静的型付き

Statically typed

既に型が決定してる

動的型付け
Dynamic typing
実行時に型を付ける

よくある誤解
静的型＝コンパイラ
動的型＝インタプリタ

ただしプログラム実行時のデータの表現方法がそういう関係になりがち

C言語は静的型付き

多くの動的言語はC言語で実装

必ずしも排他の
関係ではない

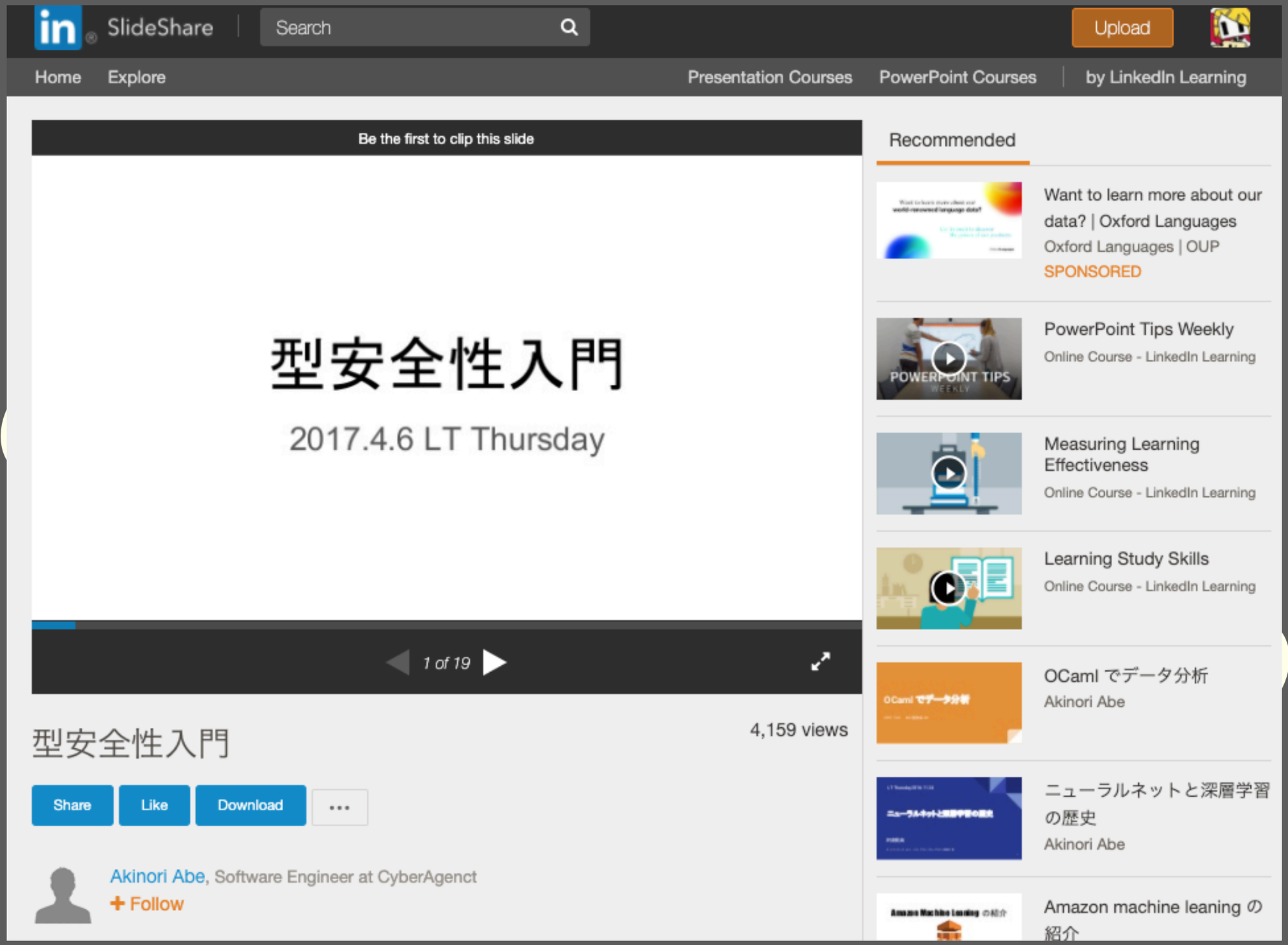
お使いの言語ランタイム(ruby, python, phpなど)のC言語実装がどうなっているか、変数がC言語の型でどう表現されてるか読んでみるとおもしろいと思います

PHPは実行時に型をチェックする

ただし関数とプロパティに静的に型を付けられる

ただし静的に型を付けて
も実行時にクラッシュす
る可能性があります

型で守られてると
はどういう状態か



–Akinori Abe 『型安全性入門』(Apr 6, 2017)

“

”

–Akinori Abe 『型安全性入門』(Apr 6, 2017)

関数を呼び出す側が間違って呼び出すことを PHPだけでは防げない

動的言語で完全かつ健全にするのは不可能

偽陽性(動かないのに動きそうなフリをしている)こ
とを最小にしていくこと
を目指す

コンパイラの代りに
静的型チェッカーを使
おうということになる

PHPでも複数の型チェッカーが揃っている

PhpStorm
Phan, Psalm
PHPStan

PHPのソースコードを解析しても判別できない型は注釈(annotate)してあげる必要がある

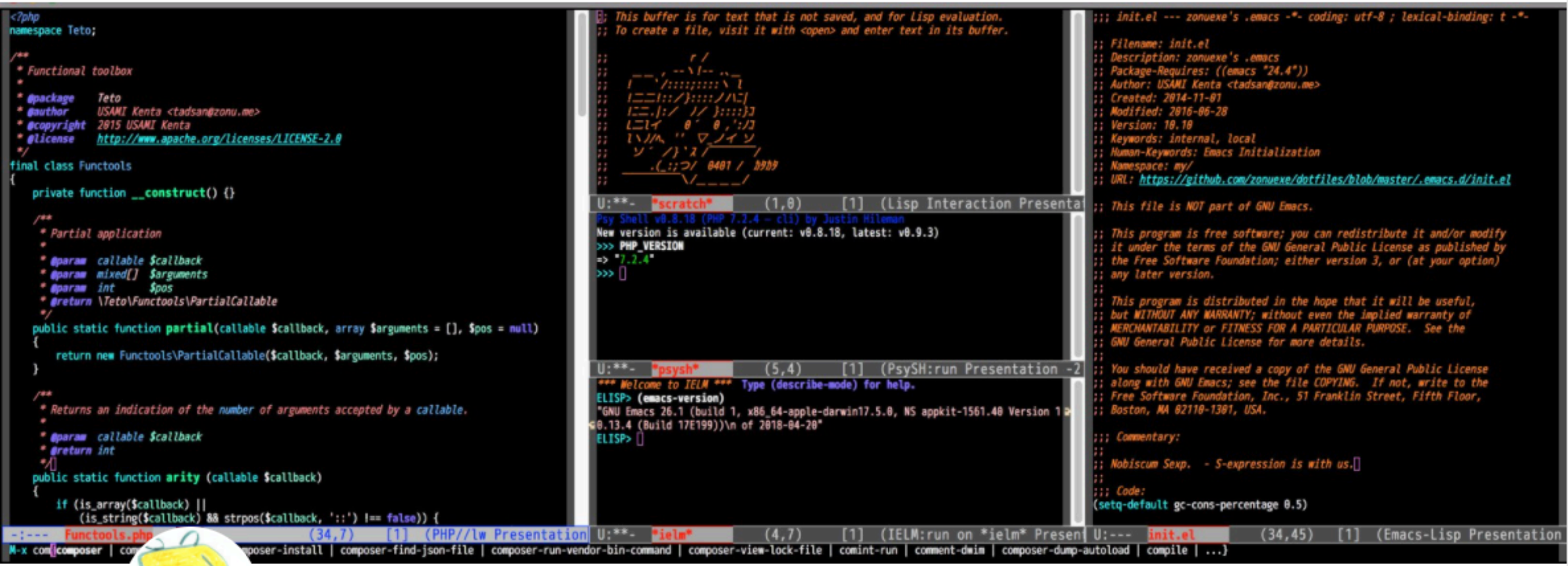
人間がコメントで型を書く

人間が型を間違って
つけると破綻するの
で実際簡単ではない

連想配列(Hash, dict)を構造体のつもりで多用されると静的解析を簡単に破滅できる

と思うでしょ？
PHPDocでなら
型をつけられます

PHPに型を付けたり
CIに載せる話は
FANBOXに記事やス
ライドを掲載してます



tadsan

プロフィール　投稿　ショップ

型について学ぶにはどうすればいい

型システム入門 プログラミング言語と型の理論

著者 ：Benjamin C. Pierce 著、住井 英二郎 監訳、遠藤 侑介 訳、酒井 政裕 訳、今井 敬吾 訳、黒木 裕介 訳、今井 宜洋 訳、才川隆文 訳、今井 健男 訳

定価 ：7,480円 （本体6,800円＋税）

判型 ：B5

頁 ：528頁

ISBN ：978-4-274-06911-6

発売日 ：2013/03/26

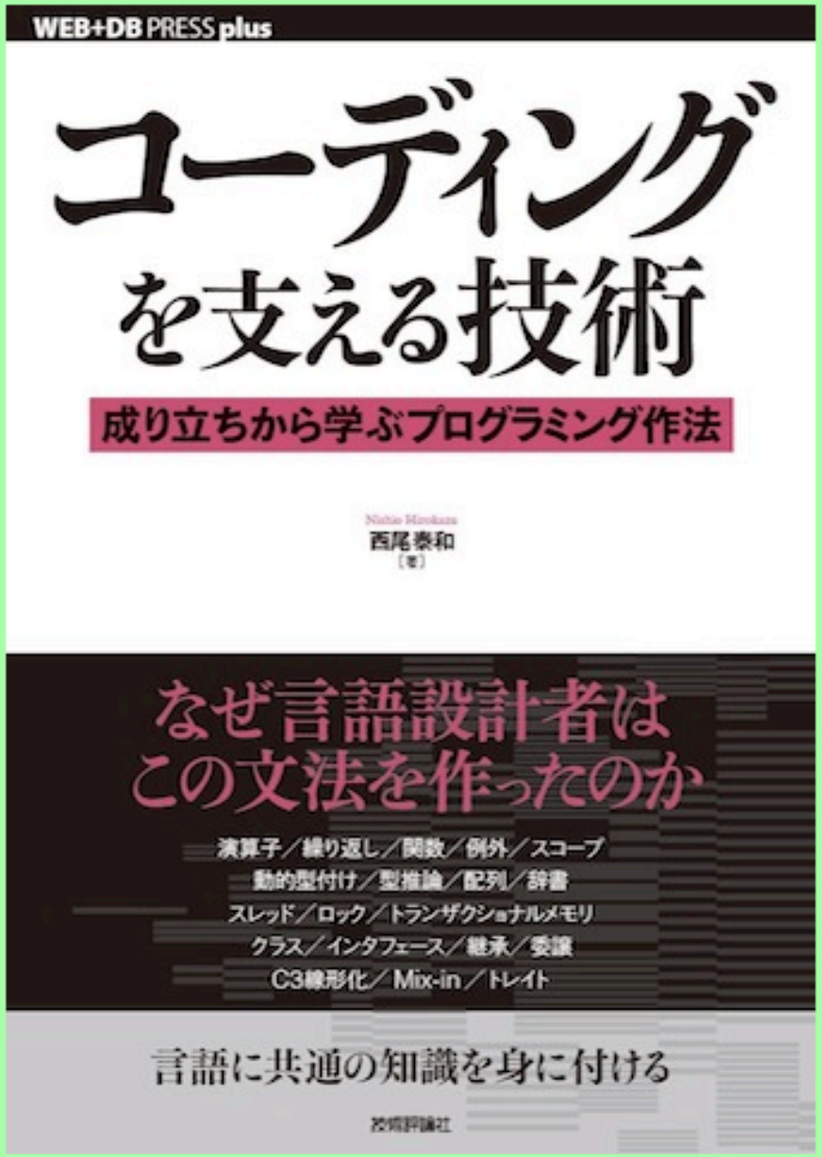
発行元 ：オーム社

ハードルが高い
(そもそも値段が…)

一章のPDFは無料なので読んでほしいが難しいので理解できなくてもいい

コーディングを支える技術

著者公式ページ



世の中にはたくさんのプログラミング言語があります。そしてプログラミングに関する概念も、関数、型、スコープ、クラス、継承など、さまざまなものがあります。多くの言語で共通して使われる概念もあれば、一部の言語でしか使われない概念もあります。これらの概念は、なぜ生まれたのでしょうか。本書のテーマは、その「なぜ」を理解することです。

そのために本書では、言語設計者の視点に立ち、複数の言語を比較し、そして言語がどう変化してきたのかを解説します。いろいろな概念が「なぜ」生まれたのかを理解することで、なぜ使うべきか、いつ使うべきか、どう使うべきかを判断できるようになるでしょう。そして、今後生まれてくる新しい概念も、よりいっそう理解しやすくなることでしょう。

型のみならずプログラミング言語についてのさまざまな概念を学べる

まとめ

仕事で使ってる言語を好きになるために目を逸らさず向き合っていこう