

なぜ新しい型チェッカーを 書きはじめたのか

How I Ended Up Writing a New Type Checker for Ruby



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)

はじめましての方は
はじめまして

2014年くらいまで
Ruby界隈にも
出入りしていました

日本Ruby会議2011 初めて参加

札幌Ruby会議2012 レポート班

RubyKaigi 2013 レポート班

RubyKaigi 2014を 最後にRubyと疎遠に

趣味でいろいろな
動的言語をやっています
(Lisp, shell, ...)

特に動的言語／
スクリプト言語が好き

動的言語には
無限の可能性()が
あります

みなさん
型は書きたいですか！

「素晴らしい」 言語は
世の中にいっぱいある

動的言語で
フリーダムに書かれた
プロジェクトは
熱的死を迎えて崩壊する

業務で動的言語を
正当化したい



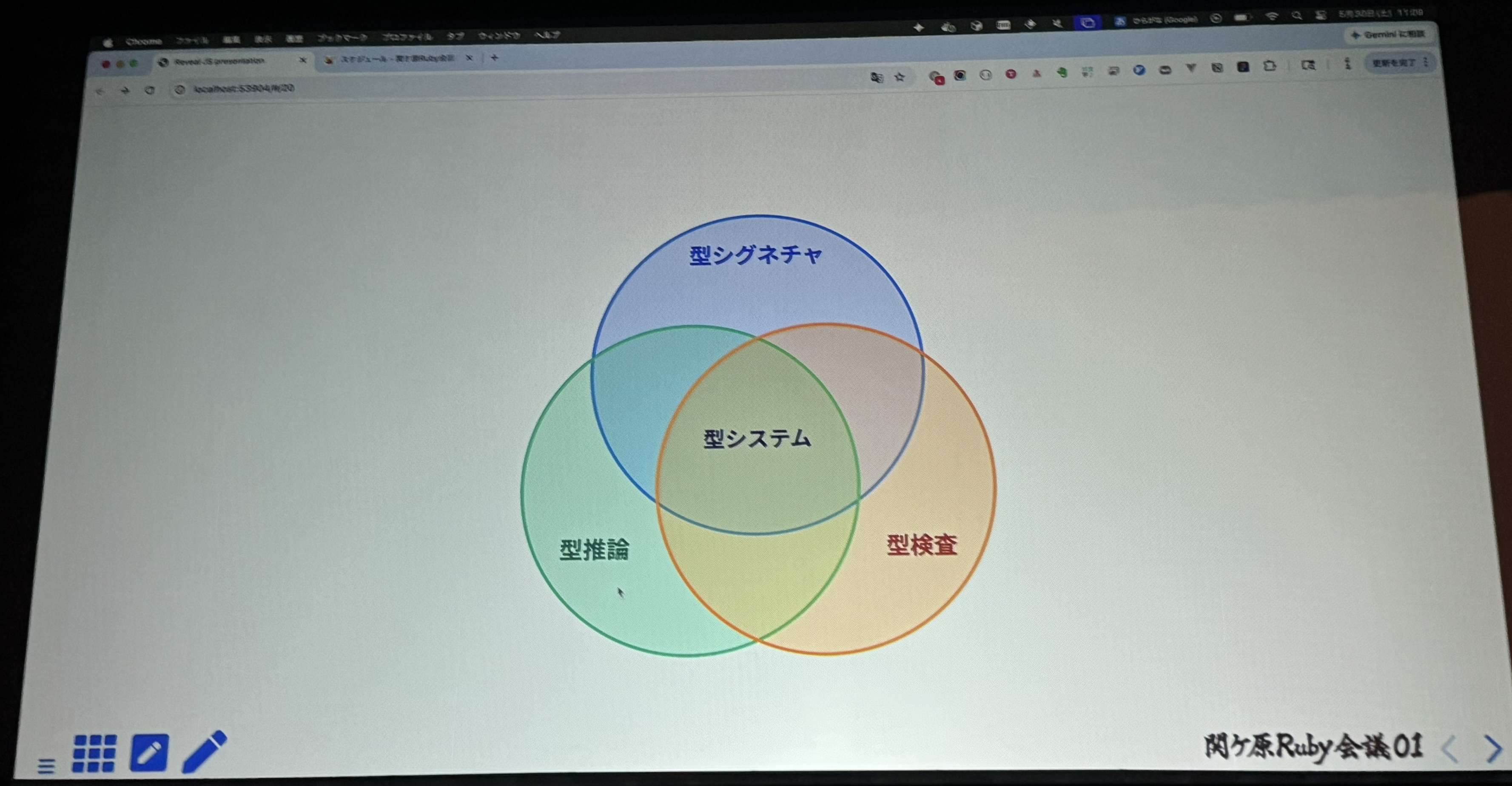
関ヶ原Ruby会議01

拙者、
『型は欲しいが型は書きたくない』
者たちとの和睦を結び、
るびいにおける型の領地安堵を
実現せんと欲す者也

大江森塚三矢大坂守小輔次郎真年

@sanfrecce_osaka

2026/05/30 #sekigahara01



PHP界隈で型の話を 始めて10年くらい

pixiv inside [archive]



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 ム(//><)ノ ☆

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にて大怪獣に蹂躪さ
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！

Phanとは

PhanはPHPの静的解析ツールです。開発元はハンドメイドのマーケットサービスを運営し、現在
PHP作者のRasmus Lerdorf氏する米Etsy社です。もちろんRasmus Lerdorf氏も開発に参加してま
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 236

[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)
 - 2016 / 7 (2)
 - 2016 / 6 (1)
 - 2016 / 5 (4)
 - 2016 / 4 (1)
 - 2016 / 2 (2)

10年前は遅くて
CIで使うのも非実用

PHPでは10年くらいで
型を書くのが当たり前に



Meet The Next Member of Your Team!

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

Documentation

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.

```
$ vendor/bin/phpstan
1/1 [ ] 100%

-----
Line   Article.php
-----
11     Call to an undefined method App\Article::getName().
16     If condition is always true.
-----
```

PHPStanは動的解析と ハイブリッドで現実的速度



PHPStanクイックガイド2023

2023/09/17に公開 ↻ 2024/12/13



83



10



php PHP



PHPStan



Tech

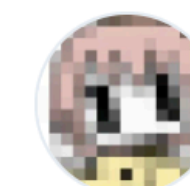
PHPStan（PHP Static Analysis Tool）はコードを実行せずに検査できるツールです。本稿では業務アプリケーションにPHPStanを導入するまでに押さえておきたい事柄を記述します。



この記事の初出は PHPerKaigi 2023 のパンフレット原稿です。

導入

PHPStanは本稿記述時点の1.9.x系において、PHP 7.2以降で実行できます。PHPStanは `composer require --dev phpstan/phpstan` でのインストールが基本です。



にゃんだーすわん

フォロー



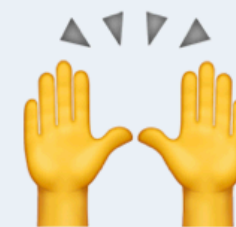
にゃーん

バッジを贈る

バッジを贈るとは →

目次

- 導入
 - ・ IDEの有効化
- PHPと型
- 型付けの基礎



キミにも作れるPHPStan拡張

2024/04/13に公開



10



1



php PHP

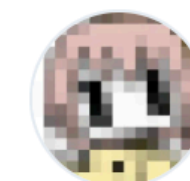
PHPStan

Tech

こんにちは！ PHPStanを活用していらっしゃいますでしょうか。

PHPStanは今日では言わずと知れた静的解析ツールですが、プラグイン機構を備えており、型付けのためのさまざまな機能を提供しています。PHPStanはできるだけコードを書かずPHPDocによる記述だけで多くのケースには型をつけられるようになっていますが、残念ながら本稿執筆時点のPHPStan 1.10系ではPHPDocだけですべてのニーズを満たすことはできないため、より深くPHPStanを利用するにはプラグインの理解が必要になります。

プラグイン機構は外部パッケージによる拡張だけでなく、PHPStan本体に組み込まれている標準関数への型付けなどにも使われています。つまり、PHPStanがどのような原理で解析対象のコードに型をつけているのかを知るにはプラグインAPIへの理解が必要になります。本稿ではPHPStanの拡張APIについて簡単に説明し、実用的に使える拡張プラグインを作れるところまでまとめます。



にゃんだーすわん

フォロー



にゃーん

バッジを贈る

バッジを贈るとは →

目次

- PHPStan拡張の種類
- PHPStanのソースコードにアクセスする
- PHPStanに同梱されている既存拡張を読む
- 簡単なルールを実装する

PHPStan extensions

PHPDoc

型レベルでかなりのの
ことができるように

PHPDocと
拡張を書きまくれば

エージェントで
PHPDocを
増やしまくればいいのでは

RBSが普及してない
のは実装と宣言が
分離しているからでは

RubyKaigi 2024

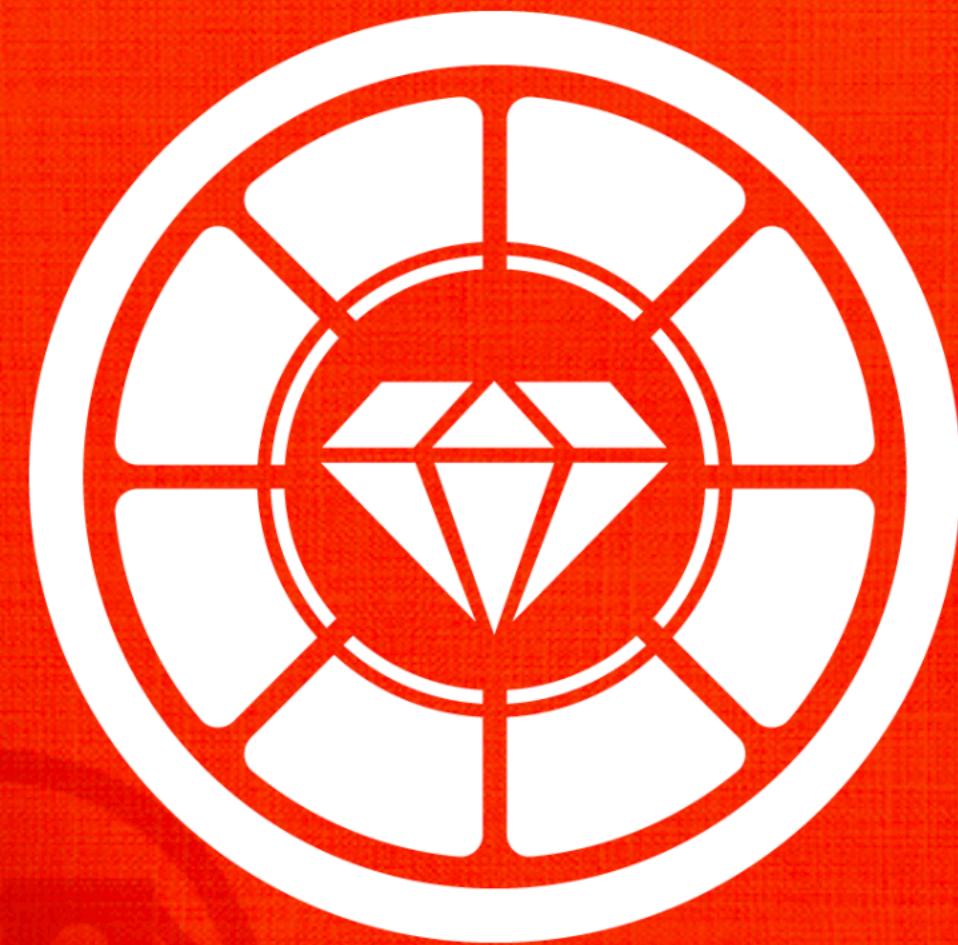
Embedding it into Ruby code

Soutaro Matsumoto
Timee, Inc.

素晴らしい
Rubyでも型を
書きましょう

完

時を遡って...



RubyKaigi 2025

April 16th - 18th, 2025

Ehime Prefectural Convention Hall
(愛媛県県民文化会館), Matsuyama, Ehime

Keynote Speakers

Yukihiro "Matz" Matsumoto
Mari Imaizumi
Ivo Anjo

RubyKaigi 2025で
ほぼ10年ぶりに参加

Automatically generating types by running tests

RubyKaigi 2025 Day1

RubyKaigi 2026

Hakodate,
Hokkaido, Japan

2026 April Wed **22**, Thu **23**, Fri **24**

Hakodate Arena 函館アリーナ · [Map](#)

Hakodate Citizen Hall 函館市民会館 · [Map](#)

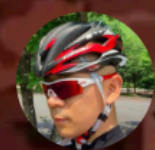
Keynote Speakers



Yukihiro "Matz" Matsumoto



Charles Nutter



Satoshi Tagomori

RubyKaigi 2026 is over. Thank you everyone who made it.

See you at RubyKaigi 2027 in Miyazaki!

This work by RubyKaigi 2026 Team is licensed under [CC BY 4.0](#) Speaker, session, and sponsor information is provided by respective parties and is not covered by this license.

楽しかったですね！

RubyKaigiは エコシステムを作っている 人たちの成果発表会

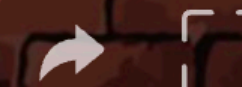
個人的に
おもしろかったこと



RubyKaigi 2026

No Types Needed, Just Callable Method Check

Daichi Kamiyama @_dak2_
2026-04-23 RubyKaigi 2026





Good Enough Types: Heuristic Type Inference for Ruby

RubyKaigi 2026

Shia@STORES



にゃんだーすわん

@tadsan



型の方々に詰め寄っているいろいろ話せておもしろかった [#rubykaigi](#)



その前に

Fukuoka Rubyist Kaigi 05

2026.02.28 Sat. 10:00–18:00

主催: Fukuoka.rb

会場: リファレンス駅東ビル 3 階 会議室H-2



Claude Code に向いているプログラミング言語

2026/03/05に公開 12件



AI



プログラミング言語



ベンチマーク



llm



Claude Code



Tech

[English version →](#)

TL;DR

Claude Code に 13 言語で簡易 git を実装させたところ、Ruby・Python・JavaScript が最速・最安・最安定でした。静的型付けの言語は 1.4～2.6 倍遅く、生成コストがかかる傾向がありました。

Cost for Claude Code to Generate a Mini-Git (v1+v2, 20 trials)

**Yusuke Endoh**

フォロー

World No.1 IOCCC player.

http://ioccc.org/winners.html#Yusuke_Endoh

バッジを贈る

バッジを贈るとは →

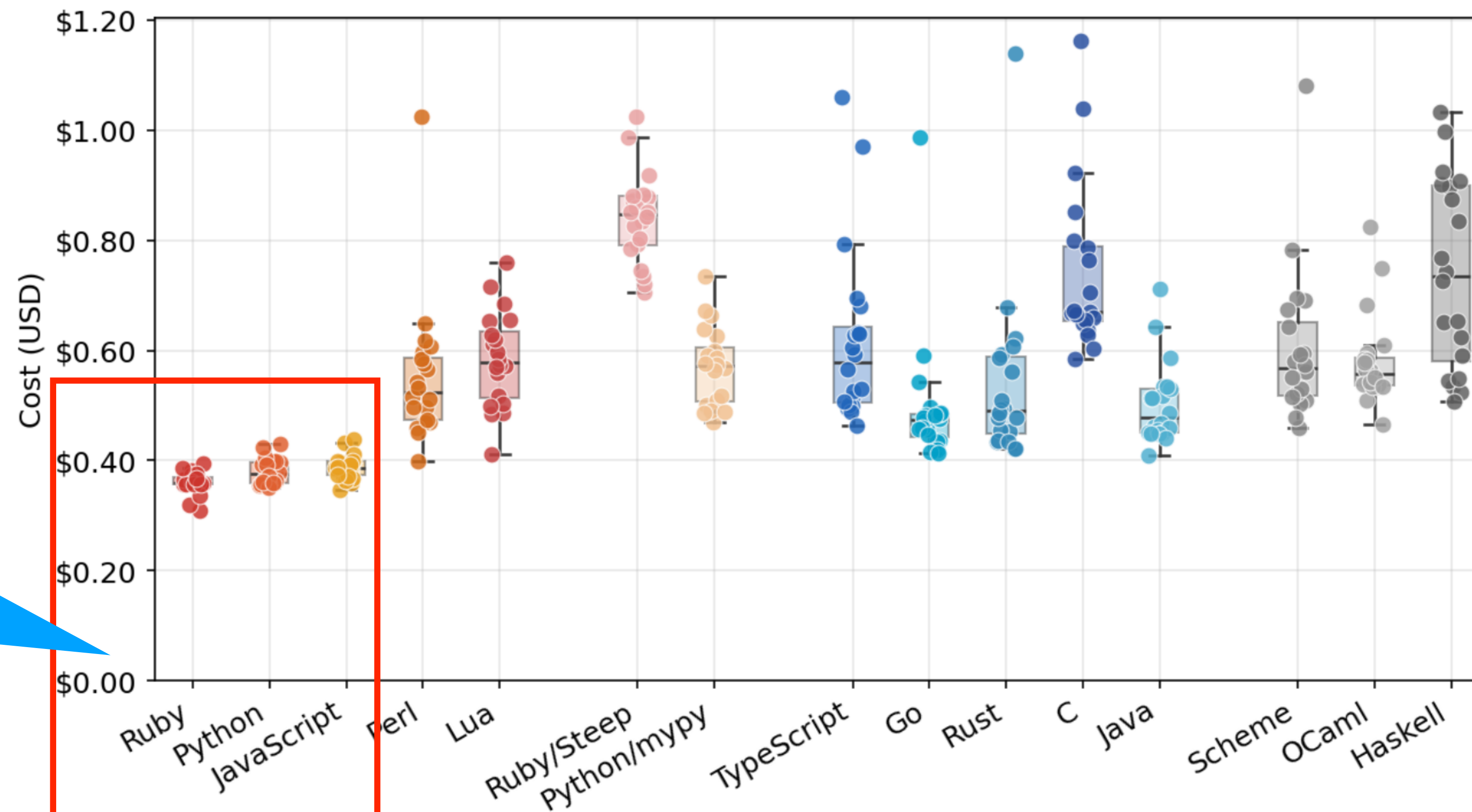
目次

もしかして…
型って

TL;DR

Claude Code に 13 言語で簡易 git を実装させたところ、Ruby・Python・JavaScript が最速・最安・最安定でした。静的型付けの言語は 1.4～2.6 倍遅く、生成コストがかかる傾向がありました。

Cost for Claude Code to Generate a Mini-Git (v1+v2, 20 trials)



シンプル
型なし

書かない方が
いいのでは…？

「自分の未来予測を信じてちょっと意地を張ってみる」 まつもとゆきひろ氏が Rubyに型宣言を入れない理由



今のRubyに型宣言を入れるのは
賢い選択ではないんじゃないかなと
思っているんですね

Ruby開発者
まつもとゆきひろ氏



型宣言がなくても静的型宣言のメリットは実現できる

例えば、Ruby3.0から静的型チェックを導入するようになったんですけども、型宣言はなかったんですね。つまり、型情報はそのプログラムとは別のところに持っていて、その情報を使って型チェックを行おうと。ないものについては推論をしようというアプローチですね。

もちろん、今のRubyの静的型チェックは誕生したばかりでまだ未成熟なんですけれども、原理的には、型宣言なしで型チェックの恩恵をある程度受けることができると言われています。がんばっている途中ですね。ただ、型宣言はないので、ドキュメントとしての役割は果たせません。

ただ、そうはいっても型情報はいろいろあるので、そこによる推論結果みたいなものをIDEで表示することによって、ドキュメントとしての役割をある程度果たせますし、Rubyには、インラインドキュメント。つまり型がなくてもいずれにしても振る舞いは書かないといけないのでインラインドキュメントをする習慣があって、型情報を書いたらそれを取り込んでくる。型情報として取り込んでくるというツールも今開発中です。

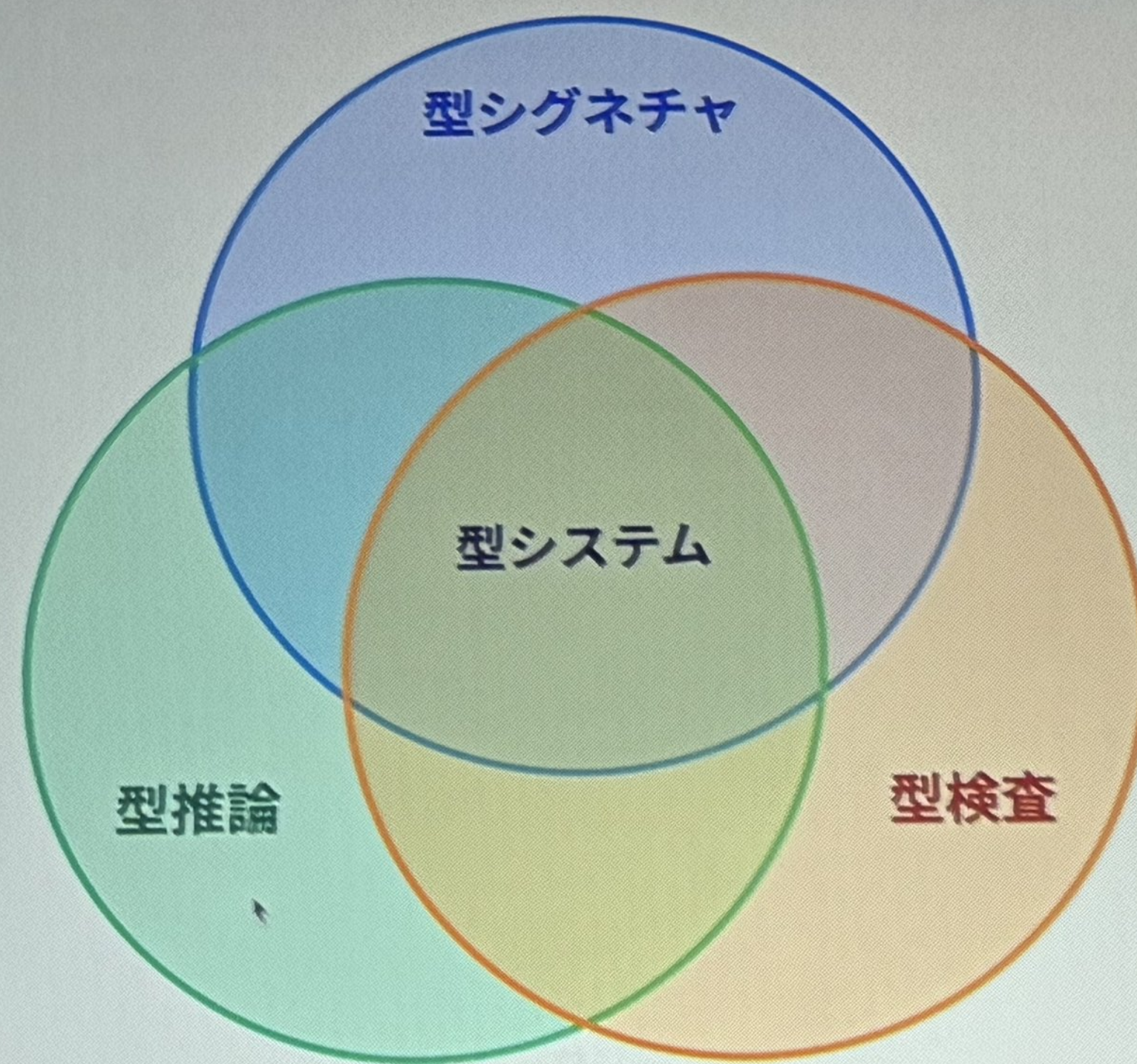
Rubyに型宣言を入れない理由

ただ、一度言語に型宣言を持たせると、もう取り除けないわけですよ。型宣言のあるプログラムがもうそこにたくさんできちゃうわけです。さらに、コミュニティの中におそらくですが、型警察みたいなものが出てくると思うんですね。

ジェムとかライブラリを作っている人のところに行って、「お前の使っているこのライブラリに型宣言がないんだけど、型宣言は絶対にあるべきだ」みたいに言って、型宣言を付けていないプログラムに対して文句をつけまくるみたいな人たちがコミュニティに登場しそうな気がするんですね。実際によその言語ではそういうことが発生していると聞くんですね。

今年の年初まで
RBSは良くないと
思ってた

RBS::Inlineの方が
実装と分離しにくい



長年の経験則で
どこから「型なし」が
湧いてくるのか
理解できていた

既存のアプローチより
良いやりかたが
あるのでは？



にゃんだーすわん
クリエイターページ

投稿する

ダッシュボード

投稿管理

プラン管理

クリエイター設定

ファン一覧

おたより

支援金管理/振込

FANBOXプリント

FANBOXプリントとは

コンテンツ提供者申請

クリエイター優待

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。

Rigor

Ruby ソースに注釈を書かなくても、フローに応じた実用的な型フィードバックを返す静的解析器です。

[ハンドブックを読む →](#)[ソースを見る](#) **English**[Read in English](#)**日本語**[日本語で読む](#)**最新リリース v0.1.18** ・ 2026年6月11日

Rigor はソースに型注釈を一切書かずに `rigor check` で Ruby コードを解析し、型付きレシーバーへの未定義メソッド呼び出し、引数個数の誤り、証明可能な `nil` デリファレンスなどのバグを報告します。型はすべてコードが実際に生成する値から推論されます。チームに型注釈を書いてもらう必要はありません。

```
1  # Rigor Playground (WASM) – runs entirely in your browser.
2
3  def greet(name)                               String
4    "Hello, " + name                             String
5  end                                             :greet
6
7  result = greet("World")                       "Hello, World"
8  result.upcase                                "HELLO, WORLD"
9
10 # Try introducing an error:
11 x = nil                                       nil
12 x.upcase # possible-nil-receiver           Dynamic[top]
13
14 # Inline-RBS works without a magic comment (rigor-rbs-inline is
15 # pre-loaded with require_magic_comment: false, per ADR-32 WD10):
16 class AscDesc                                singleton(AscDesc)
17   # @rbs asc_or_desc: :asc | :desc
18   def ascdesc(asc_or_desc)                   :asc | :desc
19     asc_or_desc                               :asc | :desc
20   end                                         :ascdesc
21 end                                           :ascdesc
22
23 AscDesc.new.ascdesc(:bad)                   Dynamic[top]
24
```

DIAGNOSTICS 2

ERROR Line 12, col 3
undefined method `upcase' for nil
call.undefined-method

ERROR Line 23, col 21
argument type mismatch at
parameter `asc_or_desc' of
`ascdesc' on AscDesc: expected
:asc | :desc, got :bad
call.argument-type-mismatch

■ **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  world = rand == 0 ? "ruby" : "world"           "ruby" / "world"
8  result = greet world                          "Hello, ruby" / "Hello, world"
9  result.upcase                                "HELLO, RUBY" / "HELLO, WORLD"
10
11
```


■ Rigor Playground WASM ● 1 error



```
1  # Inline-RBS works without a magic comment (rigor-rbs-inline is
2  # pre-loaded with require_magic_comment: false, per ADR-32 WD10):
3  class AscDesc                               singleton(AscDesc)
4      # @rbs asc_or_desc: :asc | :desc
5      def ascdesc(asc_or_desc)                 :asc | :desc
6          asc_or_desc                           :asc | :desc
7      end                                       :ascdesc
8  end                                         :ascdesc
9
10 ● AscDesc.new.ascdesc(:bad)                Dynamic[top]
11
```

型絞り込みになる
ヒントをかきあつめる

具体的な型情報を
保ち続ける

$$\begin{aligned}a &= 1 \\ b &= 2 \\ c &= a + b\end{aligned}$$

Rubyで任意の
型推論ロジックを
注入できる

ActiveRecord ActiveSupport

プラグインでひたすら
型をつける

100% Claude Code

80% Claude Code
10% Gemini
10% Grok

リアルワールドの Railsアプリでテスト しまくる

Fableのお蔭で
最適化がめっちゃ進んだ

設計思想

AIエージェントに
SKILLで判断させる

PHPStanで 学んだ経験則

ユーザーは適切な
設定を選べない

無理してmaxレベルを
目指そうとする

PHPStan自体も
発展途上なので
コードを曲げることに
なりかねない

偽陽性に対して
過剰なコードを
書かされる

Rails 向け Rigor: mise を使ったステップバイステップ セットアップ

このウォークスルーでは、Rails プロジェクトをゼロから最初の `rigor check` 実行まで導きます。`mise` を使って Ruby 4.0 と合わせて Rigor をインストールし、解析器をプロジェクトの `Gemfile` の外に置きます。

セットアップの進め方は2通りあります。

アプローチ	向いているケース
A <code>rigor-project-init</code> スキル	ほとんどのプロジェクト。スキルがスタックを検出し、プラグインを提案し、設定を書いてくれます。
B 手動のステップバイステップ	各決定を明示的に制御したい場合。

どちらも同じ結果になります。迷ったら Path A を選んでください。

呼び出し方

AI コーディングエージェントに次のいずれかを伝えてください。

“Set up Rigor in this project.” “Configure Rigor for this Rails app.” “Add type checking.”

エージェントは次のコマンドを実行して応答するはずです。

```
rigor skill rigor-project-init
```

このコマンドはSKILL 定義を標準出力に出力します。SKILL ファイルとその `references/` ディレクトリの絶対パスを含む短いヘッダー、続いてSKILL 本文です。エージェントはその手順に従い、ヘッダーが示すディレクトリから `references/NN-*.md` ファイルを順に読み込みます。

エージェントが自動的にコマンドを実行しない場合は、明示的に依頼してください: “`rigor skill rigor-project-init`” を実行して、表示される手順に従ってください。”

このフローは任意のバンドルスキルに対して使用できます。

お使いの言語でセットアップ

上のプロンプトはただの自然言語です。どの言語で頼んでも、エージェントは同じリンク先の手順に従います。つまりセットアップの対話全体を母語で進められます。下のすぐ使えるプロンプトはそれぞれ、エージェントにその言語で対話しながらセットアップを一緒に進めるよう明示的に依頼します:

✓ お使いの言語でセットアップ

✓ [ja] 日本語

日本語で対話しながら、このプロジェクトに Rigor をインストールする作業を一緒に進めてください。次の手順
<https://raw.githubusercontent.com/rigortype/rigor/refs/heads/master/docs/install.md>

› [zh-Hans] 简体中文

› [zh-Hant] 繁體中文

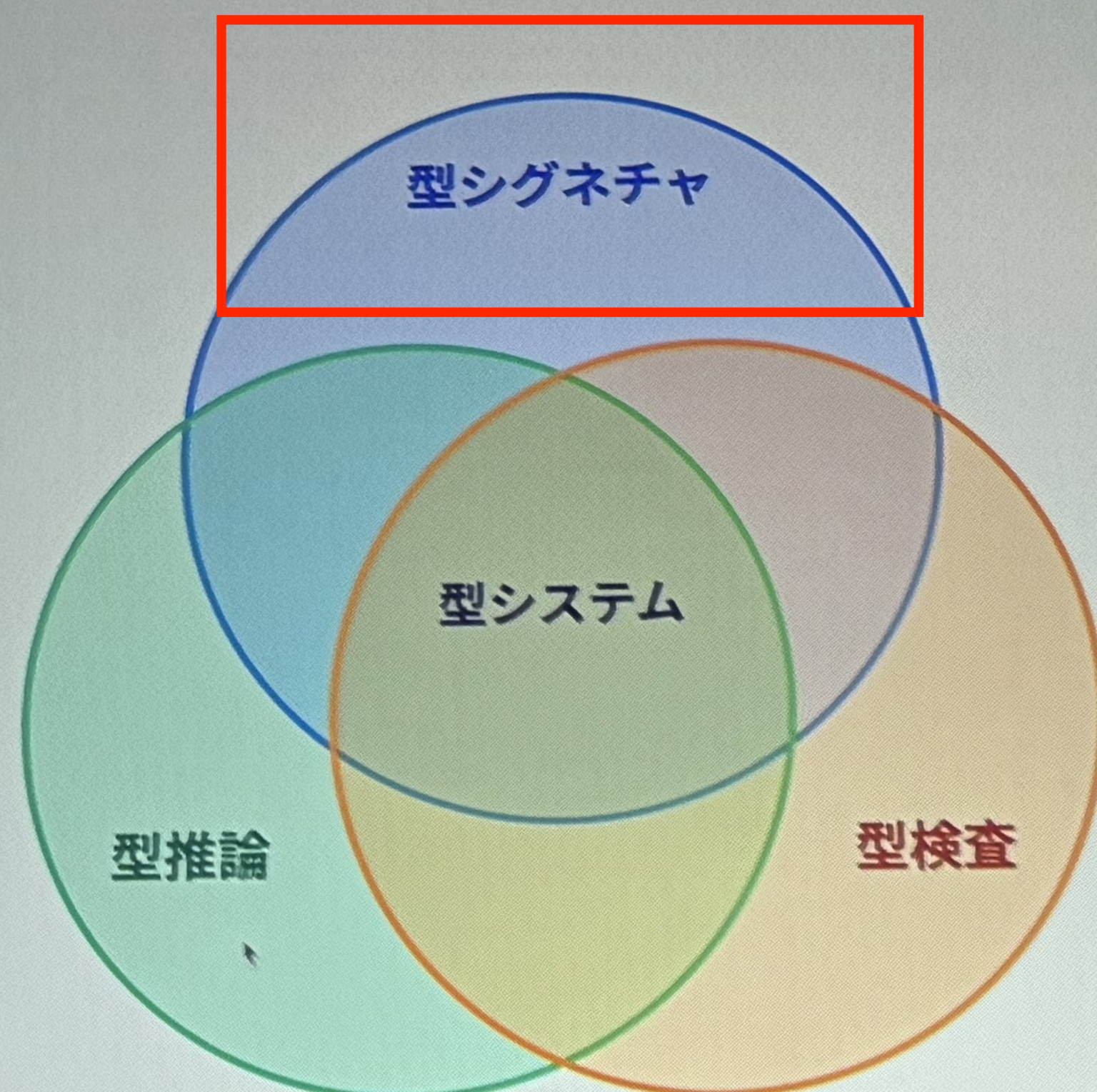
› [ko] 한국어

› [pt-BR] Português (Brasil)

› [pt] Português

プロジェクトの段階に
よって無視していい
エラーも存在する

型とはつまり
なんなのか



実行時にとりうる
予測可能な状態

Rubyには構文上
要求される型宣言は
ないのだから
どう書いてもいいはず

RailsのCoCも 人間が期待する型

型チェッカーが
人間の期待に
応えられないのが悪い

メンテナンスとオーサリング

- **rigor-upgrade** : 新しい Rigor バージョンをクリーンに採用します。ベースラインとの差分を取り、本物の新規キャッチを sig 品質の偽陽性から仕分けし、再生成します。
- **rigor-doctor** : セットアップが健全であること(設定が解決し、プラグインがロードされ、ベースラインが新鮮で、RBS 環境が壊れていないこと)を検証します。[トラブルシューティング](#)を参照してください。
- **rigor-plugin-author** : 自分のリポジトリに新しい Rigor プラグインをスキャフールドして、Rigor が推論できないアプリケーション DSL やメタプログラミングパターンについて教えます。バンドルされたプラグインがプロジェクトの依存するフレームワークや DSL をカバーしていないときに使ってください。

理屈上は任意の項に
型をつけられる

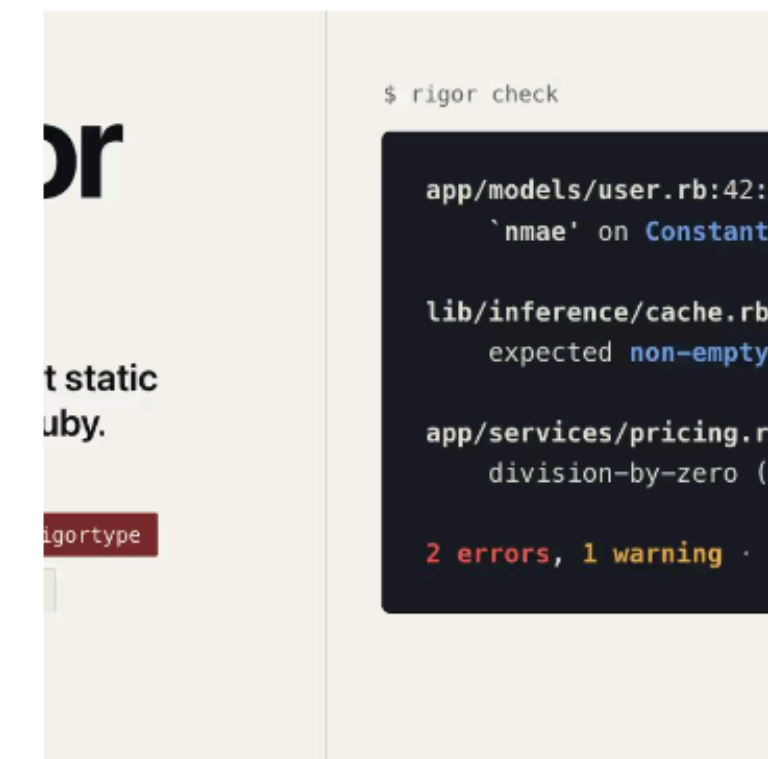
型推論がうまくいくのは単機能のちっちゃいスクリプトだけでは...？と思われるかもしれませんが、現実的なプロジェクトでもちゃんとしっかり型がつくように頑張っています。

秘訣はPHPStan譲りの**プラグインシステム**です。

正攻法で型を推論できないメタプログラミングやDSLは全部これでカバーできます。

rigor/plugins at master · rigortype/rigor

Inference-first static analysis for Ruby. Contribute to rigortype/rigor development by creating an account on GitHub.



サンプルコードも結構いっぱい考えていて、Lispみたいな式を静的に評価したり、Rack的なマイクロフレームワークのコントローラに暗黙的なインターフェイス(プロトコル)を与えて、パラメータに型をつけるなんてもこともできます。

プラグインの書き方は
覚えなくていい

プログラムがとりうる
作用は戻り値だけでは
ない

Effect system (実装済み)

Rigor最大の問題

tadsanは
Ruby開発者でも
Rails開発者でもない

圧倒的当事者 意識の欠如

実際に使っている人の
フィードバックが欲しい！

もっとというと
マジレスが欲しい！



にゃんだーすわん
クリエイターページ

投稿する

ダッシュボード

投稿管理

プラン管理

クリエイター設定

ファン一覧

おたより

支援金管理/振込

FANBOXプリント

FANBOXプリントとは

コンテンツ提供者申請

クリエイター優待

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。

この記事でいちばん
強調したかったこと

ということで、そろそろ一ヶ月分の**Claude Code Max**契約が切れそうなので、ちょっと動的言語の静的解析の未来を見てみたいという方はドシドシGitHub Sponsorsで応援していただきたく... 何卒...

Sponsor @zonuexe on GitHub Sponsors

Support zonuexe's open source work



あとはPIXIV FANBOXからのご支援など、ご無理のない範囲でよろしくをお願いします。私自身は**フリーソフトウェア**という概念が大好きなのでこのソフト自体をプロプライエタリ化するつもりはないため、**応援してくださるみなさまの支援が頼りです**。もし業務で活用してくださる会社さんがあれば、まとまった金額を投げていただけると大変ありがたい限りです。

ということでよろしくをお願いします！

構造上
メンテが高コスト

ということで、そろそろ一ヶ月分の**Claude Code Max**契約が切れそうなので、ちょっと動的言語の静的解析の未来を見てみたいという方はドシドシGitHub Sponsorsで応援していただきたく... 何卒...

Sponsor @zonuexe on GitHub Sponsors

Support zonuexe's open source work



あとはPIXIV FANBOXからのご支援など、ご無理のない範囲でよろしくをお願いします。私自身は**フリーソフトウェア**という概念が大好きなのでこのソフト自体をプロプライエタリ化するつもりはないため、**応援してくださるみなさまの支援が頼りです**。もし業務で活用してくださる会社さんがあれば、まとまった金額を投げていただけると大変ありがたい限りです。

ということでよろしくをお願いします！

Claude Max 20x
トークン消費を
抑えながら作っても
ちよつと足りない

今月のClaude契約は
今朝切れました！

それはさておき

Rigorステッカー
持ってきています





fkuMnk2 突如現わる —————...
@fku_mnk



ALT

rigor.typedduck.fail

動的言語の未来に
ご期待ください