

2023年度未踏 芝山PJ開発進捗報告(9月)

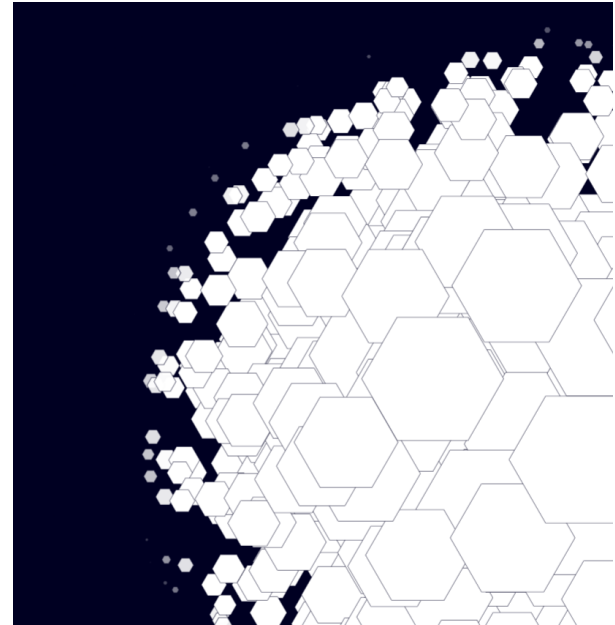
Pythonにトランスパイル可能な静的型付け プログラミング言語の開発

芝山駿介

自己紹介

芝山駿介

早稲田大学先進理工学部物理学科4年
大学では場の量子論、量子情報などを
勉強しています



背景

Pythonは人口に膾炙しているが、とても奇妙な言語

```
Python 3.11.0 (main, Jul 12 2023, 00:27:36) [Clang  
Type "help", "copyright", "credits" or "license" fo  
>>> class C:  
...     x: int  
...     def __init__(self, x):  
...         self.x = x  
...  
>>> c = C(1)  
>>> c == c # must be True  
True  
>>> C(1) == C(1) # !?  
False  
>>> 
```

なぜこのような挙動になるか？

A. `__eq__` を実装していないから

下のようにする必要があった

```
class C:
    x: int
    def __init__(self, x):
        self.x = x
    def __eq__(self, other):
        return self.x == other.x
```

でも、`__eq__` を実装していないならエラーにしてほしい
(願わくば、実行前に)

というか、Pythonに静的型システムが欲しい

理想

```
C = Class { .x = Int }
```

```
c = C.new { .x = 1 }
```



```
print! c == c
```

the type of `==`::rhs (the 2nd argument) is mismatched

expected: Eq

but found: test::C els(E1941)

```
: ::c(: test::C)
```

[問題の表示 \(F8\)](#) [クイック フィックス... \(F8\)](#)

でも、ただ新しい言語を作るだけではダメ

Pythonのコード資産がそのまま使えるような言語が欲しい

それと、Python代替を目指すならエコシステムも揃っていて欲しい(パッケージマネージャ, フォーマッタ, Language Server, etc.)

→ Pythonの公式エコシステムはあんまり出来も良くないので、後発の強みを活かして改善されているとなお良い

また、せっかく静的検査をやるのだからネイティブコードも出力したい

...と、いうことを実現する言語 **Erg** を作っています

これまでの進捗(6~8月)

Language Server

未踏期間前から基本的な機能は実装済み

- [x] Completion
- [x] Diagnostics
- [x] Hover
- [x] Go to definition
- [x] Find references
- [x] Renaming
- [x] Inlay hint
- [x] Semantic tokens
- [x] Code actions
- [x] Code lens

未踏期間中の主な進展はLanguage Serverの並列化

パッケージマネージャ

実用的なアプリケーションを作るにはパッケージマネージャが不可欠
パッケージマネージャはErg自身を用いて実装した

現在使用可能なコマンド:

- publish: 作成したパッケージを検証し、GitHub上で管理されるレジストリに登録
- build: コードをコンパイルして成果物をbuildディレクトリに置く
- clean: buildディレクトリをclean-upする
- help: ヘルプを表示する
- init: パッケージを初期化する
- run: アプリケーションパッケージを実行する
- install: (シェルスクリプトを使った擬似)実行可能ファイルを作って `$ERG_PATH/bin` に置く
- metadata: パッケージのメタデータを表示

ネイティブコードバックエンド

ネイティブコードバックエンドは、Rustコードをターゲットとする方式で実装

$$Erg \text{ スクリプト } \xrightarrow{\text{Ergコンパイラ}} Rust \text{ コード } \xrightarrow{\text{Rustコンパイラ}} \text{バイナリ}$$

ネイティブコードバックエンドは、構成的にはErg to RustトランスパイラとRustコンパイラを呼び出す部分からなる

トランスパイラ

トランスパイラ(Galと命名)の方は、現在

- 関数呼び出し(print, assert)
- 変数・関数・メソッド定義
- コントロールフロー(if, for, while, match)
- 基本的な二項演算
- クラス定義
- import(Ergモジュールのみ)

を変換可能

コンパイル(トランスパイル)実行例

```
~/Documents/GitHub/gal git:(main) (0.064s)
```

```
cat test.er
```

```
id x = x  
add x, y = x + y
```

```
world = "world"  
print! "Hello, {}! ", id world  
print! "1 + 2 = {} ", add 1, 2
```

```
~/Documents/GitHub/gal git:(main) (0.811s)
```

```
cargo run -- compile test.er
```

```
Finished dev [unoptimized + debuginfo] target(s) in 0.06s
```

```
Running `target/debug/gal compile test.er`
```

```
~/Documents/GitHub/gal git:(main) (0.292s)
```

```
./test
```

```
Hello, world!  
1 + 2 = 3
```

Rustに変換するので当然ではあるが、fibonacci関数での簡易ベンチマークではPythonよりも10倍以上高速に実行された

```
~/Documents/GitHub/gal git:(main) (0.063s)
cat test_fib.er
fib 0 = 0
fib 1 = 1
fib(n: Int): Nat = fib(n-1) + fib(n-2)

assert fib(10) == 55
print! fib 35

~/Documents/GitHub/gal git:(main) (13.893s)
hyperfine "python test_fib.py"
Benchmark 1: python test_fib.py
Time (mean ± σ): 1.371 s ± 0.005 s [User: 1.309 s, System: 0.011 s]
Range (min ... max): 1.365 s ... 1.379 s 10 runs

~/Documents/GitHub/gal git:(main) (0.541s)
time cargo run -- compile test_fib.er
Finished dev [unoptimized + debuginfo] target(s) in 0.02s
Running `target/debug/gal compile test_fib.er`
cargo run -- compile test_fib.er 0.20s user 0.07s system 55% cpu 0.479 total

~/Documents/GitHub/gal git:(main) (1.545s)
hyperfine "./test_fib"
Benchmark 1: ./test_fib
Time (mean ± σ): 74.6 ms ± 21.3 ms [User: 69.4 ms, System: 0.6 ms]
Range (min ... max): 69.4 ms ... 159.8 ms 18 runs
```

これからはErgとの互換性を高めるためにバインディングライブラリやPython APIを模倣するRustライブラリを作っていく

言語機能

主要なものとしては

- スライスの追加
- ユーザー定義再帰型の追加
- パターンマッチの機能強化

コンパイラ

- 型システムのバグ修正
- コード生成器のバグ修正

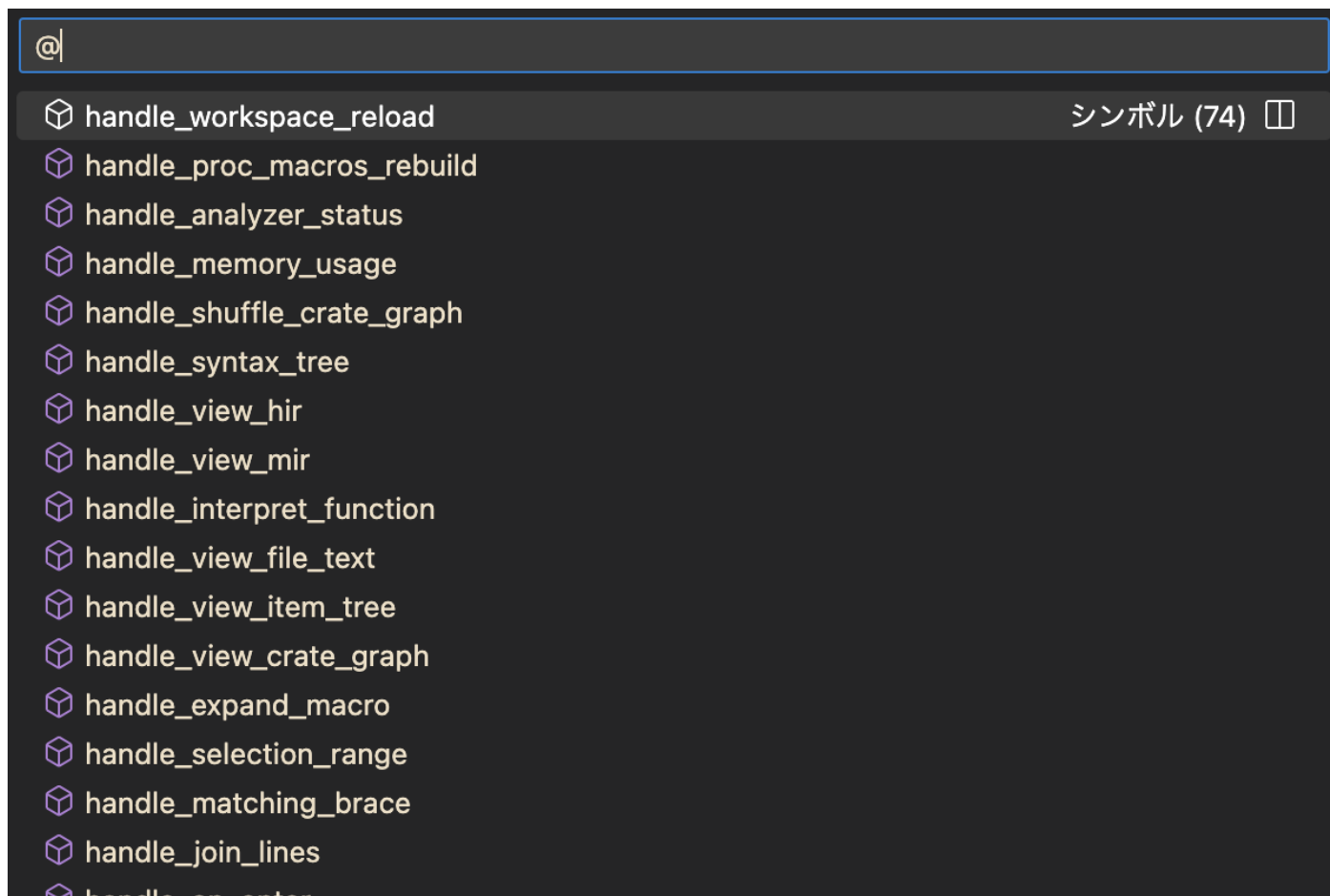
9月の進捗

Language Server

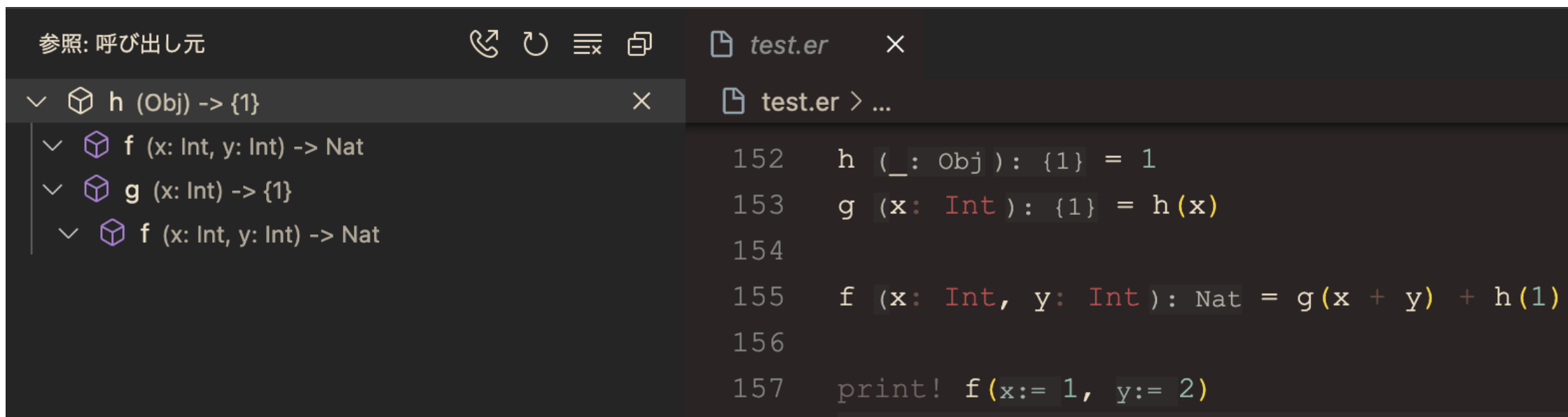
- Workspace diagnosticsを実装
プロジェクトを開くと自動的にパッケージ全体の検査をやってくれる機能
実用的にはかなり欲しかった機能なので、実装できて良かった

- Workspace symbolを実装

プロジェクト内でセマンティックにシンボルを検索できる機能



- Call hierarchy機能を実装
関数の呼び出し階層を表示できる機能



The screenshot shows a code editor with a dark theme. On the left, a call hierarchy window is open, titled '参照: 呼び出し元' (Reference: Caller). It shows a tree structure of function calls. The root is 'h (Obj) -> {1}', which is expanded to show its callers: 'f (x: Int, y: Int) -> Nat', 'g (x: Int) -> {1}', and 'f (x: Int, y: Int) -> Nat'. The right pane shows the source code for 'test.er', with line numbers 152 to 157. The code defines functions 'h', 'g', and 'f', and a 'print!' statement. The function 'h' is defined as 'h (_: Obj): {1} = 1'. The function 'g' is defined as 'g (x: Int): {1} = h(x)'. The function 'f' is defined as 'f (x: Int, y: Int): Nat = g(x + y) + h(1)'. The 'print!' statement is 'print! f(x:= 1, y:= 2)'.

```
参照: 呼び出し元
h (Obj) -> {1}
  f (x: Int, y: Int) -> Nat
  g (x: Int) -> {1}
  f (x: Int, y: Int) -> Nat

test.er
152 h (_: Obj): {1} = 1
153 g (x: Int): {1} = h(x)
154
155 f (x: Int, y: Int): Nat = g(x + y) + h(1)
156
157 print! f(x:= 1, y:= 2)
```

- Folding range機能を実装
importリストを折りたためるようになった

- Language Serverのテスト基盤を開発・公開

サーバーと通信を行うダミークライアントを実装、これを用いてテストも実装
コンパイラ本体と分離できたので別リポジトリで公開



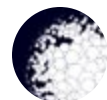
[erg-lang/molc](#)

A mock language client for testing language servers

★ 0

🔗 0

また未踏とは関係ありませんが、LSPに関する知見を本にまとめました
近日公開予定([Zenn/Leanpub](#))

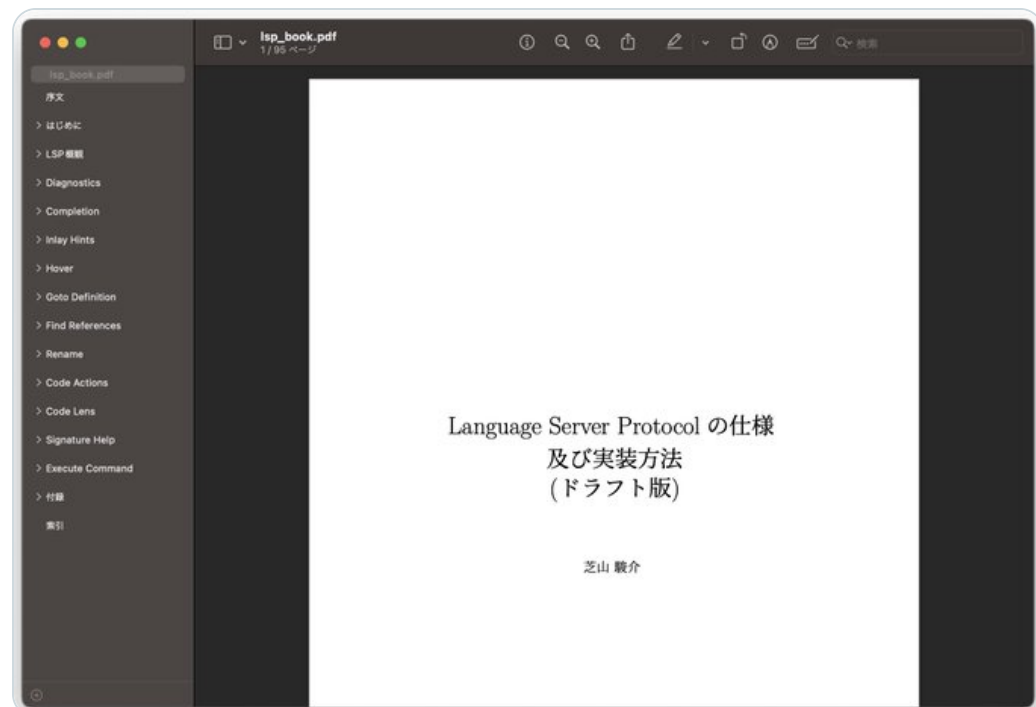


Shiba

@s_sbym · [フォローする](#)



春から書いていたLSPの本がようやくまとまりました



午後9:05 · 2023年8月31日



180



返信



リンクをコピー

[1件の返信を読む](#)

フォーマッタ

- スタイルガイドの策定
- フォーマッタ本体の実装(実装中)

インデントの処理がかなり面倒で、完成まではまだしばらくかかりそう

パッケージマネージャ

残念ながら目立った進展なし

言い訳: パッケージマネージャの実装を進めようとするたびコンパイラとLanguage Serverの
バグ及び機能不足が立ちはだかるため

ひと段落した感じはあるので、10月の実装を進められそう？

言語機能

- パターンマッチ機能の改善
ネストしたパターンに対応
- デコレータ (Python と同名の言語機能) の実装
- 内包表記 (言語機能) の実装

```
y = match [1, 2]:
    [1, 2, 3] -> 6
    [x, 2] -> x
assert y == 1

a = match { .x = 1 }:
    { y; } -> y + 100
    { x; y } -> 200 + x + y
    { x; } -> x
assert a == 1

b = match { .x = [1, 2]; }:
    { y; } -> y + 100
    { x; y; } -> 200 + x + y
    { x = [1, y]; } -> y
assert b == 2

c = match [[1, 2], [3, 4]]:
    [x, [3, 5]] -> x + 100
    [x, y, z] => 200 + x + y + z
    [[1, x], [3, y]] -> x + y
assert c == 6
```

コンパイラ

- CIの改善

CIはPython 3.11だけで行っていたが、3.7~3.11まで全てサポート(このために3.7特有のバグとかを必死に直した)

- その他、バグ修正

Jobs

- ✓ test (windows-latest, 3.7)
- ✓ test (windows-latest, 3.8)
- ✓ test (windows-latest, 3.9)
- ✓ test (windows-latest, 3.10)
- ✓ test (windows-latest, 3.11.3)
- ✓ test (ubuntu-latest, 3.7)
- ✓ test (ubuntu-latest, 3.8)
- ✓ test (ubuntu-latest, 3.9)
- ✓ test (ubuntu-latest, 3.10)
- ✓ test (ubuntu-latest, 3.11.3)
- ✓ test (macos-latest, 3.7)
- ✓ test (macos-latest, 3.8)
- ✓ test (macos-latest, 3.9)
- ✓ test (macos-latest, 3.10)
- ✓ test (macos-latest, 3.11.3)
- ✓ build-check (windows-latest)
- ✓ build-check (ubuntu-latest)
- ✓ build-check (macos-latest)

問題

9月に予定した作業が(特に新規実装部分で)あまり進められなかった
Language Serverは概ね完成したので、これからは新規実装を進められそう

今後の予定

- パッケージマネージャでは、依存関係解決器を実装し、ライブラリの依存関係を管理できるようにする
 - その他、公開に向けて必要な機能を実装する
 - 8合目会議までに一通り動かせるようにしたい
- ネイティブコードバックエンドでは、より多くのコードを変換できるよう機能強化を進めるほか、バインディング機構の実装を行う
 - パッケージマネージャをバイナリにコンパイルするのが目標
- フォーマッタの実装を進める
- コンパイラ本体では、パッケージマネージャの実装で必要になった機能やバグ修正等を行う

進捗チャート

灰色が完了、青が進行中(2023/9/30現在)

