

The Brass Language

AIと作る実用的な言語処理系

@cordx56

2026-08-22 Kernel/VM探検隊@東京 No19

▶ おま誰

- cordx56、かおるこたや、など
 - aka: たやちゃん、たやさん、など
 - 56.ax や @cordx56 で検索
- 東京都内の大学で博士課程学生
 - Rustの型システムと開発支援が専門
 - 博士論文が....白紙！w
- 様子のおかしいRustオタク
 - 最近はRustコンパイラ型システムバグ直し屋さんとか



▶ みんな大好き自作言語

- 自作「プログラミング」言語の話
- 既存言語の言語処理系を自作、ではない
- 言語の構文、型の性質、実行時の振る舞いまで、自由自在
 - 見た目をCっぽくしたり、Rubyっぽくしたりできる
 - `1 + "hello"` を禁止したり、しなかったりできる
 - 実行前にコンパイルをしたり、実行時にコンパイルしたりできる

▶ Q. なぜ今更自作言語を？

- A. 発表者の知るすべての言語に不満があるから
 - 背景: 発表者は元々Pythonオタクです
- ありとあらゆる言語にそれぞれの利点欠点がある
 - Pythonはサッと書けて型付けは強いが、動的なので大変
 - Rustは型付けも強く静的だが、書くためのコストが大きい
 - TypeScriptは `as any`、JavaScriptはやはり型付けが....
 - Haskellは型推論が嬉しいが純粹関数型前提はちょっと....
 - Scalaは良さそうではあるが、もう少し雑に実行したい（欲張り）

▶ 真に欲しい機能を言葉にしよう

- 自分が真に欲しい言語とは何かを、AIと壁打ちする
 - これはFable 5登場前だったのでOpus 4.8とやった気がする
- 結果得られたものは以下のとおり
 - 静的型付けで、できるだけ人間が注釈せずにpolymorphismを実現
 - 注釈なしのDuck typing可能な型システム
 - 実行時はJITコンパイルをし、一度コンパイルされたら実行が速い
 - あくまでスクリプト言語として、AOTコンパイルはしない

▶ それって○○○かな～？

- それらを満たす言語機能を調べる
 - これもOpus 4.8と共に調査
- 結果得られたものは以下のとおり
 - 型推論はいつもの、一般にHindley-Milnerと呼ばれるやつを参照
 - Record polymorphismはWandや大堀先生らが導入している
 - JITコンパイルはPythonのNumbaなどの手法が良さそう（後述）
- 山ほど先行研究があり、あとは積み木を積み上げるだけ

▶ 出来上がったもの



- The Brass Language: <https://brass-cz.org>
- 名前やロゴは人の手で生成されたため、ほんのり温かい
- 実装コマンド: インタプリタ `brass`、LSPサーバー `czls`、パッケージマネージャ `czpm`

▶ どんなコードが書ける？

- 以下のようなコードが書ける

```
fun show(person) {  
    println(person.name)  
}  
show({ name: "cordx56" })
```

- 実行すると「cordx56」が出力される

▶ 型定義

- 当然（？）型の定義もできる

```
type Person = {  
    name: string  
}
```

- 直和型も表現可能

```
type Person = Student { name: string }  
             | Teacher { name: string }
```

▶ Subtyping

```
type Person = { name: string, age: int32 }  
fun show(person) {  
    print(person.name)  
}
```

```
show({ name: "cordx56" })           // OK
```

```
show({ name: "cordx56", id: 1 })    // OK
```

```
show(Person { name: "cordx56" })    // ERROR
```

▶ メソッド

- 自称オブジェクト指向なのでメソッド実装もできる

```
fun Person.show(self) {  
    println(self.name)  
}
```

- Associated function (?) も

```
fun Person.new(name: string) {  
    return Person { name: name, age: 0 }  
}
```

▶ Nullable / Result

- Null安全かつNullable

```
fun Person.null() -> Person? {  
    return null  
}
```

- Resultはstd実装

▶ LSPサーバー

- 型を書かなくていいことは、型を知らなくていいことを意味しない
- 型推論の隆盛は開発環境で型が表示されることが大きい
 - と、勝手に思っている
- 開発環境で型を知るためには、LSPサーバーが必須
 - インタプリタの実装を拝借して実装

▶ パッケージマネージャ

- パッケージマネージャ `czpm` はBrass自身で実装
- 実装は `std`、別に実行ファイルを配置

▶ 実装

- 実装はRustとBrass自身でやった
- 実装期間はおよそ2週間
 - 以降はバグフィックスや細かいメンテが中心
- 実装はLLMが主体、私はテストコード作成や設計に集中
 - 利用モデルはOpus 4.8, Fable 5, gpt-5.6-sol
 - モデルに応じてClaude CodeとCodexを併用
 - ほとんどはFable 5が設計、gpt-5.6-solが実装

▶ 実装手順

1. Opus 4.8とチャット画面で設計を詰める
 - ここはめちゃくちゃ口を出した
 - Rustコンパイラ `rustc` の設計を参考に
 - 最初に適切な実装や規約を用意すると、その後はよく回る
2. LLMで実装
3. 別セッションのLLMと大まかに目でレビュー
4. テストコードを投入し、バグと原因を特定
5. 2に戻る

▶ という変な言語を実装しました

- 試していただいて、いろいろ意見を伺いたいです！
- またドメイン取得しちゃった: <https://brass-cz.org>
- 特にissueや欲しい機能の大募集
 - github.com/brass-cz/brass