Kaede

迅速な開発と Rust 連携を可能にするプログラミング言語

平本 一桐
サイボウズ・ラボ ユース生
言語処理系開発コース

自己紹介 👋

平本 一桐

🎓 背景

- **大学**: 放送大学 教養学部 教養学科 情報コース 3 年
- **技術領域**: 言語処理系・Web バックエンド・Web フロントエンド
- **興味**: 低レイヤ全般（今後は OS 自作とか CPU 自作をやりたい）

最近やってること 🧐

- 不動産システムに **Elasticsearch** を導入して検索の高速化
- Java のレガシーコードを警告して、モダンコードに変換する **LSP サーバ** の開発
- 軽登山

軽登山 ⛰️

乗鞍岳（3,026m）

2,702m までバスが走ってるので、手軽に綺麗な景色が見れる！

Kaede の目的 📌

- Rust は安全高速で理想だけど、迅速な開発には向かない
- 他言語から FFI で呼び出せばいい話ではあるが、ボイラプレートが必要だったり自然な言語切り替えが難しい問題

Kaede ってどんな言語？🤔

- **迅速な開発を重視**
  - GC 搭載で所有権の制約なし
- **でも「速さ」も捨てない**
  - 必要な部分は Rust を呼び出せる
  - ボイラープレート不要でシームレスに利用可能
- **Rust との言語切り替えが簡単**
  - 大まかな言語仕様を Rust に合わせることで脳の負荷を軽減

使用技術 🛠️

🦀 実装言語

- **Rust**: コンパイラ本体の実装
- **Python**: インストールスクリプト

🔧 依存関係

- **LLVM**: コード生成バックエンド
- **Boehm GC**: ガーベジコレクター

開発状況 💻

- 基本的な制御構文
- Python のようなファイルベースのモジュールシステム
- ボイラープレート不要で Rust の関数をインポート
- 構造体・列挙型（Tagged Union って呼ばれてるやつ）
  - メソッド
  - ジェネリクス
- コマンドライン引数

プロジェクトを作成 🗂️

```
$ kaede new test_proj
```

以下のような構成になる

```
test_proj/
   ├── src/
   │    └── main.kd
   └── rust/
        ├── Cargo.toml
        ├── build.rs
        └── src/
             └── lib.rs
```

プロジェクトをビルド 🛠️

```
$ cd test_proj
$ kaede build
```

自動で Rust プロジェクトとリンクされた実行ファイルが生成される

```
$ ./build/main
```

Rust との連携 🦀

Rust コード

```rust
pub fn is_even(n: i32) -> bool {
    !is_odd(n)
}

fn is_odd(n: i32) -> bool {
    a % 2 != 0
}
```

Cargo.toml にも自由に依存を追加可能

Rust との連携 🦀

Rust コード

```rust
pub fn is_even(n: i32) -> bool {
    !is_odd(n)
}

fn is_odd(n: i32) -> bool {
    a % 2 != 0
}
```

Kaede コード

```
import krb_generated

use krb_generated.*

fn main(): i32 {
    return if is_even(10) {
        // 偶数
    } else {
        // 奇数
    }
}
```

pub な関数だけエクスポートされるので、is_odd は非公開

コード例 1 💡

ジェネリック構造体

```
struct Student<T> {
    name: T
}

impl<T> Student<T> {
    fn new(s: T): Student<T> {
        return Student<T> { name: s }
    }

    fn get_name(self): T {
        return self.name
    }
}
```

コード例２ 💡

ジェネリック列挙型

```
enum Fruit<T> {
    Apple(T),
    Orange
}

fn main(): i32 {
    let fruit = Fruit<i32>::Apple(128)

    return match fruit {
        Fruit::Apple(n) => n,
        Fruit::Orange => 256
    }
}
```

コード例３ 💡

m.kd

```
pub struct Apple {
    size: i32
}

impl Apple {
    pub fn new(n: i32): Apple {
        return Apple { size: n }
    }

    pub fn get_size(self): T {
        return self.size
    }
}
```

main.kd

```
import m

use m.Apple

fn main(): i32 {
    let apple = Apple<i32>::new(123)
    return apple.get_size()
}
```

今後の展望 🚀

🔗 Rust 連携の拡張

- **現在**: 関数レベルでの連携（一部の組み込み型のみ）
- **今後**: Vec や String など、構造体、トレイト、マクロなどへの対応拡大

⚡ 並列処理機能の強化

- **Goroutine 風の軽量スレッド実装**
- **チャネルベースの通信機構**

今後の展望 🚀 (続き)

🛠️ 開発体験の向上

- LSP サーバの開発
  - 補完・エラー表示・リファクタリング支援

🔧 インライン Rust 機能

- Kaede コード内で直接 Rust を記述
  - パフォーマンスクリティカルな部分を簡潔に最適化
  - FFI のボイラープレートを完全に排除

計算機デモ 🧮

- 🐙 GitHub: https://github.com/itto-hiramoto/kaede/tree/main/example/calculator

ありがとうございました! 🙏

- 📧 itto.hiramoto@gmail.com
- 🐙 GitHub: @itto-hiramoto
- 🐦 Twitter: @itto_hiramoto