

rfmt Ruby layer

Wakate.rb #3 / fujitani sora

fujitani sora

👤 2001 (24)

🏢 株式会社トリドリ

✂ @_fs0414

🔄 github.com/fs0414

🔍 sorafujitani.me



最近のtips

ミセスの影響で葬送のフリーレンを見始めた、
ラヴィーネ推し

rfmt

- 高速なRuby code formatter
- RubyGems Rust Extentionを使用し、Code ModuleをRustで実装
 - <https://bundler.io/blog/2023/01/31/rust-gem-skeleton.html>
- GitHub: <https://github.com/fs0414/rfmt>
 - いまStar数55くらい
- RubyGems: <https://rubygems.org/gems/rfmt>
 - 資料書いてる時点で9950 installs
- Zenn: <https://zenn.dev/soramarjr/articles/0b2464bc09b643>

format実行フロー

1. Ruby source code を受け取る
2. Prism でパースして **AST** を取得
3. PrismBridge で AST を走査し、Rust が処理できる **JSON** に変換
4. JSON を FFI 経由で Rust に渡す
5. Rust から返ってきた結果をファイル書き込み or 標準出力

Architecture

- Ruby layer
 - CLI, LSP Integration
 - Config
 - Cache
 - PrismBridge Module によるAST Parse
- FFI Boundary
 - Magnus + rb_sys で Ruby ↔ Rust をJSON経由で接続
- Rust layer
 - Emitterによる具体的なformat処理

今日はこのRuby layerの全体的な話



TOC (技術詳細の話)

PrismBridge

Foreign Function Interface

Command Line Interface

Configuration & Cache

Native Extension & Editor Integration

E2E test

PrismBridge

`lib/rfmt/prism_bridge.rb` / `lib/rfmt/prism_node_extractor.rb`

- source codeを入力にPrism parserを呼び出し、ASTを受け取る
- 各ノードからRfmt内部が利用したいドメインモデルに合わせたメタデータを抽出
 - クラス名、メソッド名、パラメータ数、etc.
 - ロケーション情報 (行番号、カラム、オフセット)
- コメント情報の収集とシリアルライズ

PrismBridgeの中間層を入れることで、Prismとの依存を吸収し、Parserを差し替えられる設計

AST変換 — 具体例

このRubyコードを入力すると...

```
1  def greet(name)
2    puts "Hello, #{name}"
3  end
```

PrismBridge がこのような JSON に変換する

```
1  {
2    "node_type": "def_node",
3    "metadata": {
4      "name": "greet",
5      "parameters_count": "1"
6    },
7    "children": [
8      { "node_type": "required_parameter_node" },
9      { "node_type": "statements_node" }
10   ]
11 }
```

Rust 側はこの JSON を受け取り、AST として再構築してフォーマットを行う

PrismとのIntegration

- PrismはRuby標準のパースーGem
- Ruby側でPrismを呼ぶのが最も自然
- RustからPrismを呼ぶにはRuby VMを経由するFFIが必要で複雑になる
- Ruby側でPrism ASTを走査し、Rust側が処理しやすい形式への事前処理を行う役割

```
lib/rfmt/prism_bridge.rb
```

```
1 def self.parse(source)
2   result = Prism.parse(source)
3   handle_parse_errors(result) if result.failure?
4   serialize_ast_with_comments(result)
5 end
```

TOC

PrismBridge

Foreign Function Interface

Command Line Interface

Configuration & Cache

Native Extension & Editor Integration

E2E test

Magnus — Ruby bindings for Rust

- Rust で Ruby の拡張 Gem を書くためのライブラリ
- Rust の関数を Ruby のメソッドとして公開できる
- Ruby ↔ Rust 間の型変換を自動で処理
- 引数のバリデーションやエラーハンドリングも Ruby の慣習に沿って動作
- rfmt では Magnus 経由で Rust のフォーマッタを Ruby から呼び出している

Ruby-Rust 間のFFI境界

```
1  # Ruby側 (lib/rfmt.rb)
2  def self.format(source)
3    prism_json = PrismBridge.parse(source)
4    format_code(source, prism_json)
5  end
```

- Magnus crate による Ruby-Rust FFI
- Ruby から呼べる3つの関数を公開
 - フォーマット
 - パース
 - バージョン取得
- データ型変換は Magnus が自動で処理

```
1  // Rust側 (ext/rfmt/src/lib.rs)
2  #[magnus::init]
3  fn init(ruby: &Ruby) → Result<(), Error> {
4      let module =
5          ruby.define_module("Rfmt")?;
6
7      module.define_singleton_method(
8          "format_code",
9          function!(format_ruby_code, 2))?;
10     module.define_singleton_method(
11         "parse_to_json",
12         function!(parse_to_json, 1))?;
13     module.define_singleton_method(
14         "rust_version",
15         function!(rust_version, 0))?;
16     Ok(())
17 }
```

Emitterとの疎結合なInterface

- rfmtの価値はtool自体ではない
 - 高速なEmitter Module（formattingの実装詳細）が優位性
 - Ruby layerはあくまでもClient
 - 外部toolが詳細を意識せずにrfmt Emitterを利用できる状態が理想的である
 - 例えば、oxfmtも部分的にPrettierのAPIに依存している
 - [RFC: Formatter #13608](#)
- ※ この為にも、もっとRust側に実装を寄せていきたいと考えている
 - Ruby layerの実装が重たいことは、Emitterを利用するユーザーの実装コストになる

TOC

PrismBridge

Foreign Function Interface

Command Line Interface

Configuration & Cache

Native Extension & Editor Integration

E2E test

CLI の概要

`lib/rfmt/cli.rb`

- Thor ベースのCLI – Thor Gem で宣言的なコマンド定義
- コマンド: `format / check / version / config / cache / init`
- 並列処理の自動判定 (ファイル数・サイズに基づくヒューリスティクス)
 - 余談で、10fileほどであれば並列化しない方が速い
- diff表示: `diffy / diff-lcs` Gem (`unified / side_by_side / color`)
- `progress`表示

CLI の処理フロー

1. 設定の読み込み – プロジェクトごとのルールを尊重するために、YAMLから設定を探索
2. 対象ファイルの決定 – 全ファイルを毎回処理するのは遅いので、キャッシュで変更があったものだけに絞る
3. 並列/逐次の自動判定 – ファイル数・サイズから最速の戦略を選ぶ。少量なら逐次の方が速い
4. フォーマット実行 – Prism で AST に変換し、Rust に渡して整形
5. 結果の出力 – モードに応じてファイル書き込み / diff表示 / 違反チェックを切り替え
6. キャッシュ更新 – 次回の実行を速くするために、処理済みファイルの情報を保存

TOC

PrismBridge

Foreign Function Interface

Command Line Interface

Configuration & Cache

Native Extension & Editor Integration

E2E test

Configuration & Cache

Configuration - 設定管理

`lib/rfmt/configuration.rb`

- YAML設定ファイルの探索・読込・バリデーション
- ファイルglobパターンによるinclude/exclude
- デフォルト設定とのマージ

Cache - キャッシュシステム

`lib/rfmt/cache.rb`

- mtime (ファイル更新日時) ベースの変更検知
- `~/.cache/rfmt/cache.json` にJSONで永続化
- `clear / prune / stats` 操作

TOC

PrismBridge

Foreign Function Interface

Command Line Interface

Configuration & Cache

Native Extension & Editor Integration

E2E test

Gem配布 & ネイティブ拡張ロード

```
lib/rfmt/native_extension_loader.rb
```

- rfmt.gemspec + ext/rfmt/extconf.rb による標準的なnative extension gem構造
- gem install rfmt で Rust 拡張含めてビルド&インストール
- rb_sys / magnus による Ruby-Rust FFI の標準的なパターン
- Ruby 3.0〜3.3+ のバージョン別パス対応

ビルド: extconf.rb → Rust の Makefile を生成 → Cargo でリリースビルド → 共有ライブラリ (.bundle / .so) を出力

ロード: Ruby のバージョンに応じたパスからshared libraryを自動検出して読み込み

Ruby LSP Integration

```
lib/ruby_lsp/rfmt/addon.rb / lib/ruby_lsp/rfmt/formatter_runner.rb
```

- Ruby LSP の Addon として登録
- format-on-save で rfmt を呼び出し
- FormatterRunner インターフェースに準拠

Ruby LSP の Addon は Ruby で書く必要があるため、Ruby layer に実装

TOC

PrismBridge

Foreign Function Interface

Command Line Interface

Configuration & Cache

Native Extension & Editor Integration

E2E test

テスト構成

- フォーマットテスト – Rubyコードを入力し、期待する整形結果と比較
 - 条件分岐、ループ、ブロック、rescue、lambda、パターンマッチなど構文ごとに網羅
- 設定テスト – YAML設定の探索・読込・親ディレクトリからの継承を検証
- CLIテスト – コマンド実行の正常系・異常系
- LSP連携テスト – Ruby LSP Addon の登録と format-on-save の動作
- ネイティブ拡張テスト – Rubyバージョン別のロードパス切り替え

Ruby テストと Rust テスト (`cargo test`) を `rake dev:test_all` で一括実行

テストの具体例

```
1  it 'formats if with elsif and else' do
2    source = "if x > 0\nputs \"positive\"\nelsif x < 0\n... "
3
4    result = Rfmt.format(source)
5
6    # フォーマット後、正しくインデントされていることを検証
7    expect(result).to eq(expected)
8  end
```

入力のRubyコードをフォーマットし、期待するインデントや構造と一致するかを検証

まとめ

- **Rust Extension:** Ruby = parse + I/O + 外部とのIntegration、Rust = 具体的なformattingや計算負荷の高い処理
- **Prism:** Rubyの標準parserをRuby側で呼び、JSONでRustに渡す
- **Gemエコシステム:** Thor, diffy, parallel, ruby_lsp などのGemを活用

see you later 🖐️

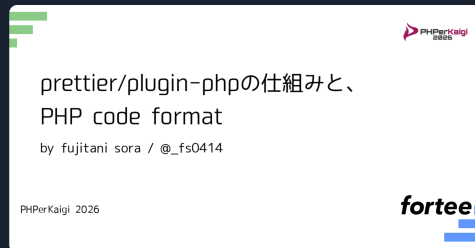
Rails Girls Tokyo #18

コーチで参加するよ

<https://railsgirls.com/tokyo-2026-02-13>

PHPerKaigi 2026

Day1に登壇するよ



rfmt

GitHub Starしてね

<https://github.com/fs0414/rfmt>