

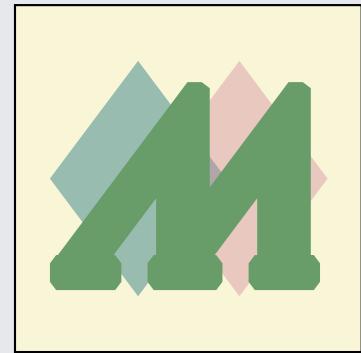
SATySF_I Language Server の 現状と今後

monaqa

2021 年 6 月 26 日

自己紹介

- ◆ 名前： monaqa
 - ▶ Twitter: https://twitter.com/mo_naqa
 - ▶ GitHub: <https://github.com/monaqa>
- ◆ SATySFI 歴： 2年ぐらい
- ◆ 技術系コミュニティ所属
 - ▶ SATySFI Slack
 - ▶ vim-jp Slack etc.



今までに作った SATySFI 関連のプログラム

satysfi-enumitem SATySFI (package)

豊富なスタイルの箇条書きを組むためのパッケージ

SLyDIF_I SATySFI (class file)

スライド作成のためのクラスファイル（このスライドも SLyDIF_I 製）

satysfi-easytable SATySFI (package)

シンプルな表を簡単に組むためのパッケージ

satysfi-azmath SATySFI (package)

アクセント、括弧類、 \LaTeX の amsmath パッケージ を真似た数式環境などを提供するパッケージ

今までに作った SATySFI 関連のプログラム

satysfi-figbox SATySFI (package)

図表を自由に組み合わせ、適切な場所に配置するためのパッケージ

satysfi-railway SATySFI (package)

グラフィックスを描きやすくするためのパッケージ

satysfi-parser Rust

Parsing expression grammar (PEG) を使用した SATySFI のパーサ

satysfi-language-server Rust

SATySFI の言語サーバ

目次

- ◆ Language Server Protocol
- ◆ SATγSF_I Language Server の現状
- ◆ SATγSF_I Language Server の今後

Language Server Protocol

Гандыңде сервер үргөсөл

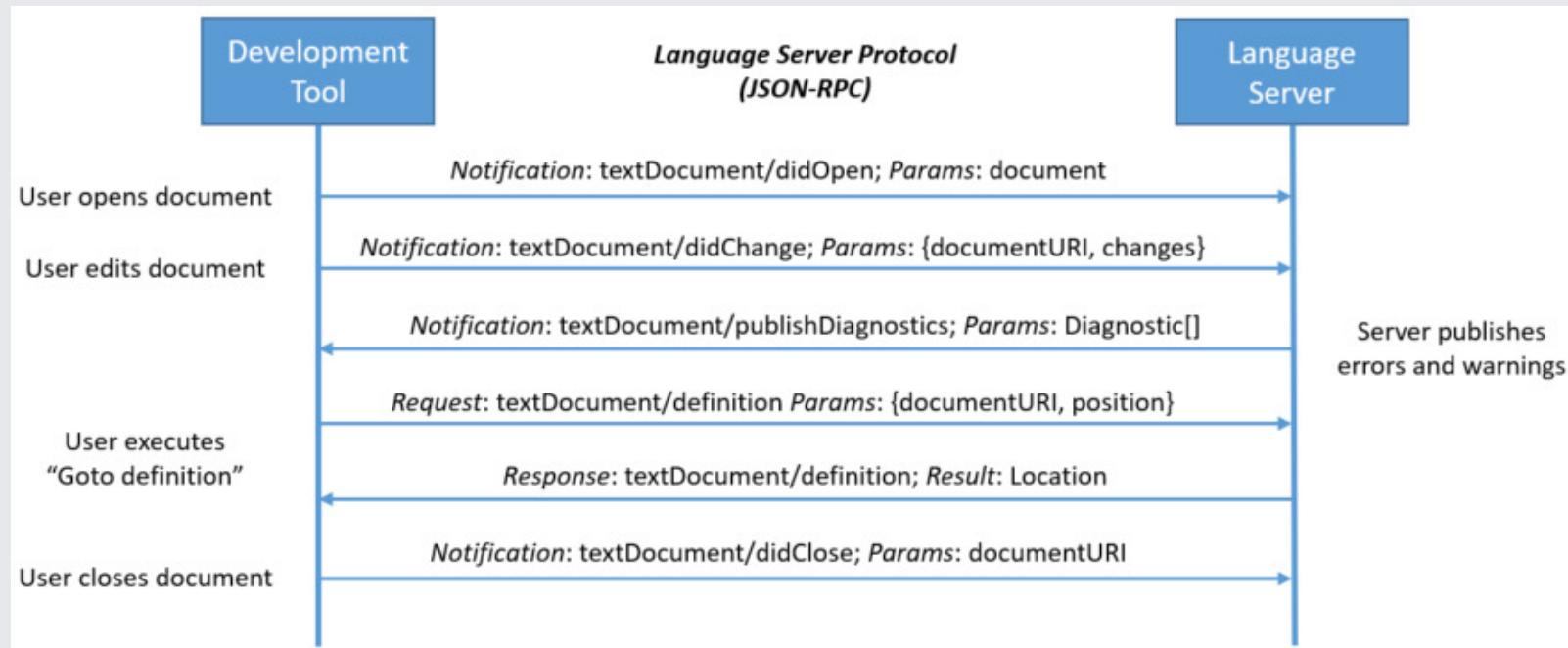
Language Server (言語サーバ)

言語サーバ：ある言語の開発に役立つ機能をサービスとして提供したもの。
言語別に実装され、以下のような機能を提供する。

補完 (completion)	ユーザが挿入しようとしている変数名・メソッド名などを予測して候補を表示する (挿入内容に応じスニペットを展開する場合も)
型情報の表示 (hover)	カーソル上の変数の情報を表示する
定義ジャンプ (goToDefinition)	カーソル上の変数が定義されている場所に移動する
リネーム (rename)	カーソル上の変数名を変更する
コードアクション (code action)	言語に応じて様々な操作を実行する (対応するモジュールのインポート、getter/setter の自動作成など)

Language Server Protocol (LSP)

言語サーバとクライアントは LSP という通信仕様に従って通信する。



<https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/>

なぜ LSP なのか：LSP の登場前

テキストエディタ

VSCode 

Vim 

Emacs 

マイナー
エディタ 

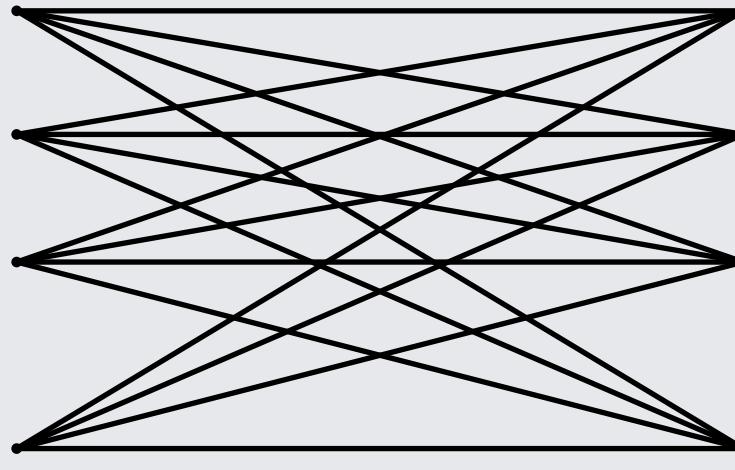
プログラミング言語

 python™

TypeScript



SATySFI



エディタ・言語毎に各々プラグインや拡張を作る必要があった

なぜ LSP なのか：LSP の登場前

テキストエディタ

VSCode 

Vim 

Emacs 

マイナー
エディタ 

プログラミング言語

 python™

TypeScript



SATySFI

マイナーなエディタ・言語では使用者が少なく実装も進まない

なぜ LSP なのか：LSP の登場後

テキストエディタ

VSCode 

Vim 

Emacs 

マイナー
エディタ 

Language Server
Protocol

プログラミング言語

 python™

TypeScript 



SATySFI 

各言語・エディタの実装が 1 つで済むため、開発が楽

なぜ LSP なのか：LSP の登場後

テキストエディタ

VSCode 

Vim 

Emacs 

マイナー
エディタ 

Language Server
Protocol

プログラミング言語

 python™

TypeScript 



SATySFI 

今日のお話

各言語・エディタの実装が 1 つで済むため、開発が楽

SATySFI Language Server の現状

SATySFI ランゲージサーバーの現状

SATySFI Language Server

- ◆ このリポジトリで開発中 (WIP!)

 [monaqa/satysfi-language-server](#)

The SATySFI Language Server

 Rust  15  1

- ◆ 現在、Neovim + coc.nvim のペアで動作することを確認
 - ▶ VSCode などのエディタで試せる方、お待ちしています

現在提供している機能

- ◆ 補完
- ◆ 定義ジャンプ
- ◆ ホバー
- ◆ 診断情報の表示

現在提供している機能

- ◆ 補完
- ◆ 定義ジャンプ
- ◆ ホバー
- ◆ 診断情報の表示

機能 1：補完機能

- ◆ コマンド名の補完
 - ▶ 編集中のファイル、又は require/import しているパッケージ内で定義されたコマンドを補完
 - ▶ 現在のモード（インライン / ブロック / 数式）に合わせて候補を変更
 - 数式モードではインラインコマンドが出ない、逆も然り
 - ▶ 宣言時にシグニチャが明示されていればそれを活用
 - 補完候補の詳細画面に表示
 - シグネチャの型に合わせてスニペット展開

機能 1：補完機能

- ◆ 変数名の補完

- ▶ 編集中のファイル、又は require/import しているパッケージ内で定義された変数を補完
- ▶ 現在のモードがプログラムモードのときに表示
 - インライン / ブロック / 数式モードでは表示されない
- ▶ 宣言時にシグニチャが明示されていればそれを活用
 - 補完候補の詳細画面に表示
 - スニペット展開は行わない（部分適用の可能性があるため）
- ▶ プリミティヴも補完の対象
- ▶ プリアンブルで宣言された変数のみ対象（`let in` 式束縛などは対象外）

機能 1：補完機能

◆ コマンド名の補完

```
279
280 +frame{機能 1：補完 }<
281
282 +listing{
283     * 変数名、コマンド名を補完してくれる
284 }
285
286 +fig-
287 +fig-center► v [LS]
288 +fig-block► v [LS]
289 +fig-on-left► v [LS]
290 +fig-abs-pos► v [LS]
291 +fig-on-right► v [LS]
292
293 >
```

機能 1：補完機能

- ◆ コマンド名の補完

```
279
280 +frame{機能 1：補完}<
281
282 +listing{
283     * 変数名、コマンド名を補完してくれる
284 }
285
286 +fig-center|
287 +fig-center> v [LS] [figbox] block-cmd
288
289                                     _____
290                                     block-cmd defined in package figbox/figbox
291
292
293 >
```

機能 1：補完機能

- ◆ コマンド名の補完

```
279
280 +frame{機能 1：補完}<
281
282 +listing{
283     * 変数名、コマンド名を補完してくれる
284 }
285
286 +fig-center();
287
288
289
290
291
292
293 >
```

機能 1：補完機能

◆ 変数名の補完

```
279  
280 +frame{機能1：補完}<  
281  
282 +listing{  
283     * 変数名、コマンド名を補完してくれる  
284 }  
285  
286 +fig-center(include-image);  
287     include-image          f [LS] int?-> length -> string -> figbox  
288     include-image-with-size f [LS] _____  
289     include-image-with-height f [LS] variable defined in package figbox/figbox  
290     include                [A]  
291  
292  
293 >
```

現在提供している機能 (再掲)

- ◆ 補完
- ◆ 定義ジャンプ
- ◆ ホバー
- ◆ 診断情報の表示

機能 2：定義ジャンプ

- ◆ コマンドや変数の定義箇所にジャンプできる機能

```
348
349 +frame{機能 2：定義ジャンプ }<
350
351 +listing{
352     * コマンドや変数の定義箇所にジャンプできる機能
353 }
354
355 >
356
357 +frame{実装の話 }<
358
359 +listing{
360     * 実装言語：\Rust;
361 }
```

NORMAL slide.saty

機能 2：定義ジャンプ

- ◆ コマンドや変数の定義箇所にジャンプできる機能

```
543 let-block ctx +frame ?:n-frame title inner =
544   let () = page-num |> SlydifiParam.set (SlydifiParam.get page-num + 1) in
545   let n-frame = n-frame |> Option.from 1 in
546   read-block ctx '<
547     +SlydifiScheme.genframe(frame-normal)(n-frame)(|title = title; inner = inner; footer = footer|
548   >
549
550
551 let-block ctx +frame-nofooter ?:n-frame title inner =
552   let () = page-num |> SlydifiParam.set (SlydifiParam.get page-num + 1) in
553   let n-frame = n-frame |> Option.from 1 in
554   read-block ctx '<
555     +SlydifiScheme.genframe(frame-normal)(n-frame)(|title = title; inner = inner; footer = footer|
556   >
```

NORMAL

~/ghq/github.com/monaqa/slydifi/src/theme/arctic.satyh

- ◆ 別ファイルで定義されていてもそこにジャンプする

現在提供している機能 (再掲)

- ◆ 補完
- ◆ 定義ジャンプ
- ◆ ホバー
- ◆ 診断情報の表示

機能 3：ホバー

- ◆ コマンドや変数の情報を参照できる機能
- ▶ 型がユーザにより明示されている場合は型も表示

```
147 >
148
149 +section{|Language Server Protocol |}<%
150
151 +frame{Language Server (言語サーバ) }<
152
153 +p{
154   \emph{言語サーバ}: ある言語の開発に役立つ機能をサービスとして提供したもの。
155 } inline command
156
157 +p{ [(int -> bool)?; inline-text] inline-cmd
158   言語別に実装され、以下のような機能を提供する。
159 }
160
161 +listing{
162   * 補完 (completion)
163     ** ユーザが挿入しようとしている変数名・メソッド名などを予測して候補に出力する機能
```

機能 3：診断情報の表示

- ◆ 構文解析がエラーとなったときにエラー箇所と原因を表示
 - ▶ いくつかのエラーについては具体的なエラー原因を表示
 - ▶ パーサの手に負えないものはエラー箇所と期待される字句のみ表示

```
376
377 +frame{機能3：診断情報の表示}<
378
379 +listing{
380     ★ 構文解析がエラーとなったときにエラー箇所と原因を表示
381     ★★ いくつかのエラーについては具体的なエラー原因を表示
382     ★★ パーサの手に負えないものはエラー箇所と期待される字句のみ表示
383
384     }           \emph{★ Incomplete inline command. \ Try adding semicolon or a
385     }           [Syntax Error] [E] Incomplete inline command.
386     Try adding semicolon or arguments after the command name.
387
388 +fig-center ★ Incomplete block command. \ Try adding semicolon or arg
389
```

SATySF_I Language Server の実装

- ◆ 実装言語：**Rust**

- ◆ 理由：

- ▶ Rust を学びたかった
- ▶ 優れたパフォーマンス・省メモリ性・安全性に惹かれた
- ▶ 静的型付け言語であること由来の書きやすさに惹かれた
- ▶ Rust で書かれた代表的な言語サーバが複数あった

Rust [rust-analyzer](#)

Deno [Deno Language Server](#)

L^AT_EX [TexLab](#)

実装：パーサ

- ◆ parsing expression grammar (PEG) パーサを使用
 - ▶ rust-peg を使用
- ◆ 現在は字句解析と構文解析を同時に行う
 - ▶ エラー回復やカーソル位置のモードの把握に難あり

実装：言語サーバ概観

- ◆ `lspower` を使用
 - ▶ `lsp-types` などが使いやすい
- ◆ 文書構造として、構文木だけでなく各ファイルで定義されている変数、コマンド、型、モジュールなどの情報を保持

実装：各リクエストに対する処理の流れ

- ◆ ファイルが開かれたとき / 変更されたとき (didOpen/didChange)
 - ▶ ファイルを構文解析し、成功したらそのファイルの `Environment` を作成
 - ▶ 依存ファイル (`@require` / `@import`) のうち未解析のものがあれば解析し、成功すれば先程と同様に `Environment` を作成
- ◆ 補完リクエストがあったとき (completion)
 - ▶

SATySFI Language Server の 今後の課題と発展性

今後の課題と発展性
SATySFI Language Server の

SATySFI と Language Server の相性

- ◆ 基本的に、静的型付け言語と LSP は相性が良い
 - ▶ 静的解析で型が判明することにより、賢い補完や診断情報をユーザに提示できるため
 - ▶ SATySFI も例外ではない
- ◆ 開発者の実力とリソースさえ揃えば、かなり便利な機能が実現できる
 - ▶ TeX や LATEX にはない SATySFI の強み

型検査・型推論の追加

- ◆ まだ型検査・型推論器は実装されていない
- ◆ 変数・コマンドの型推論・型検査によって実現できそうなこと
 - ▶ 型エラーの診断情報表示
 - ▶ シグニチャの無いコマンド・変数の補完、詳細情報表示
 - ▶ レコードのフィールド名での補完候補の絞り込み
 - ▶ パイプライン演算子直後の補完候補の絞り込み
 - ▶ etc.

補完の改善

- ◆ パーサのエラー回復機構をより強力にする
 - ▶ パーサが回復不能な構文エラーが起きると、適切な補完候補が出せない
 - 現在のモード（プログラムモード・水平モード・垂直モードなど）や依存パッケージが分からぬいため
 - ▶ しかし「ユーザの入力途中」のテキストはたいてい構文エラー
 - ▶ そのため現在はユーザの入力途中でもある程度パーサが対処できるように書いているが、完璧ではない
 - ▶ ユーザの入力途中でもエラー回復可能、またはカーソル位置のモードや依存ファイルの情報を取得できるパーサへと改良する

その他の発展性

- ◆ doc comment のホバー表示
 - ▶ 現在 doc comment 機能が本家で構想されている
- ◆ Auto import
 - ▶ コマンド補完の際やコードアクションによって必要な依存を自動解決
 - ▶ 予め `~/.satysfi/dist/packages` などの全ファイルを見ておけば可能
- ◆ コードアクション
 - ▶ 未定義のコマンドがあるとき、その定義の雛形をプリアンブルに追加
 - ▶ 定義した関数・コマンドのシグニチャをモジュールの `sig` に追加

まとめ

まとめ

まとめ

- ◆ SATySF_I language server を開発中
 - ▶ 補完機能、