neovimで作る最新Ruby開発環境 2023

自己紹介

- joker1007
- Repro inc. チーフアーキテクト
- 最近はembulkのプラグインのバージョンアップをしまくっていた
- ルビコンで闘争に明け暮れる日々は一段落
- Vimmer

大事なことは始めに言っておく

とりあえず、ここに書いてあるプラグインを入れること。

- nvim-treesitter: パーサーベースのシンタックスハイライト
- nvim-lspconfig: LSの起動、アタッチをやってくれる
- mason.nvim: LSのインストーラー
- nvim-cmp: 補完(多分、一番楽)
- cmp-nvim-lsp: nvim-cmpのLSP対応
- nvim-dap: DAPサーバーの立ち上げとアタッチを行う
- nvim-dap-ui: デバッガとしてのUIを提供する
- nvim-dap-virtual-text: デバッガ内の変数の内容をソースコード中に表示する

後は時間が無くなりそうになるまでLSPとDAPの話をしてDEMO

何故今Parserが熱いのか

「世は正に大パーサー時代」
(ところでパーサーなのかパーザーなのか。どっちでも良いと思うけど)

Language Serverの重要性

昨今の開発環境のエコシステムにおいて、LSはほぼ必須となっている。

- tsserver (TypeScript)
- rust-analyzer (Rust)
- gopls (Go)
- jdtls (Java)
- sqlls, dockerls, json-lsp, terraform-ls
- etc

Language Serverの本質

Language Server Protocolで通信するサーバープロセスで、
「編集中のソースコードを認識」し、補完やリネーム機能を提供するもの。

不完全なソースコードをパースする必要がある
外部ツールで使い易い、フォールトトレラントなパーサーが必要

LSPのリクエスト形式

- JSON-RPCとHTTPでやり取りするシンプルなテキストプロトコル

```
{
    "jsonrpc": "2.0",
    "id" : 1,
    "method": "textDocument/definition",
    "params": {
        "textDocument": {
            "uri": "file:///p%3A/mseng/VSCode/Playgrounds/cpp/use.cpp"
        },
        "position": {
            "line": 3,
            "character": 12
        }
    }
}
```

LSP Capabilities

サーバー側、クライアント側それぞれの定義があり、それぞれがサポートしている機能が何なのかを示すもの。
Language Serverが提供する機能として期待されるものが何なのかはCapabilitiesの仕様を見ると理解しやすい。

see. https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#languageFeatures

Capabilityの例 1

- Completion
- Hover
- Declaration
- Definition
- TypeDefinition
- Reference
- Document Highlight

Capabilityの例 2

- Code Action
- Rename
- Folding
- Semantic Token
- Inlay Hint
- Diagnostic

RubyのLanguage Server

- Solargraph (補完、Rename, ドキュメント表示)
- Steep (補完、型チェック)
- typeprof (型シグネチャ、codelens)
- ruby-lsp (色々)

LSP活用のための型

SteepやtypeprofもLSとしての役割を重要視する様になっている。
現在型を付けるインセンティブとして補完性能の向上は無視できない。

ちゃんとRubyをパースしてメソッドチェインの先の結果に対して補完が効く様になり、
ドキュメントやシグネチャ表示の充実など開発体験がかなり向上する。

SteepのsilentモードPR

https://github.com/soutaro/steep/pull/800 で取り込まれた。
型チェックの結果を全て黙らせるモード。
型チェックを動かすと大量のエラーが出る場合でも補完のためのLSPとして活用するというユースケースに対応する。

neovimとLSP

大体のLSはVSCodeが1st support対象だが、neovimでも割となんとかなる。
但し、VSCodeみたいに拡張を入れたら勝手に何とかなるみたいな感じではない。

neovimでLSPを使うには

素のneovimだと手動でプロセス立てて編集画面にアタッチする必要があるので、いくつかのプラグインが必要になる。

- nvim-lspconfig: LSの起動、アタッチをやってくれる
- mason.nvim: LSのインストーラー
- nvim-cmp: 補完(多分、一番楽)
- cmp-nvim-lsp: nvim-cmpのLSP対応

必須じゃないけどオススメ

- fidget: LSPのprogress UI
- vim-illuminate: カーソル下のhighlight
- lspsaga.nvim: LSPのためのUI拡張

設定例

```
local lspconfig = require "lspconfig"

lspconfig.steep.setup({
  on_attach = function(client, bufnr)
    on_attach(client, bufnr)
    -- 手動で型チェックリクエストを送るカスタムコマンド
    vim.keymap.set("n", "<space>ct", function()
      client.request("$/typecheck", { guid = "typecheck-" .. os.time() }, function() end, bufnr)
    end, { silent = true, buffer = bufnr })
  end,
  on_new_config = function(config, root_dir)
    -- bundler環境下で起動する場合の調整
    add_bundle_exec(config, "steep", root_dir)
    return config
  end,
})
```

補完設定 (最小限)

```
local cmp = require "cmp"
local default_config = require "cmp.config.default"()
cmp.setup({
  sources = cmp.config.sources({
    { name = "nvim_lsp" },
    { name = "nvim_lsp_signature_help" },
  }),
})
```

Debug Adapter Protocol

より良い開発環境のためのもう1つのプロトコル。
仕組みはLSPと大体同じだが、名前の通りデバッガを操作する。

LSPとの大きな違いは双方向通信が必要なこと。
例えば、実行中にbreakpointで止まった場合はデバッガ側からUIに通知が必要。
UIでbreakpointを設定した時は、UI側からデバッガにbreakpointの設定を指示する必要がある。

RubyにおけるDAPの実装

ko1さん作のdebug.gemがDAP(とCDP)に対応している。
debug.gemではUnix Domain SocketかTCP Serverを立てることでDAPの通信を行う。

neovimでDAPを使うには

以下のプラグインを入れることで、DAPに対応したデバッガをneovimのUIで操作できる様になる。

- nvim-dap: DAPサーバーの立ち上げとアタッチを行う
- nvim-dap-ui: デバッガとしてのUIを提供する
- nvim-dap-virtual-text: デバッガ内の変数の内容をソースコード中に表示する

設定例

```
local dap = require "dap"
dap.adapters.ruby = function(callback, config)
  callback({
    type = "server",
    host = "127.0.0.1",
    port = "${port}",
    executable = {
      command = "bundle",
      args = {
        "exec", "rdbg", "-n", "--open", "--port", "${port}", "-c", "--",
        "bundle", "exec", config.command, config.script,
      },
    },
  })
end
```

```
dap.configurations.ruby = {
  {
    type = "ruby",
    name = "run current spec file",
    request = "attach",
    localfs = true,
    command = "rspec",
    script = "${file}",
  },
}
```

DEMO

俺のnvim/init.lua

https://github.com/joker1007/dotfiles/blob/master/nvim/init.lua

オススメのプラグインなどを聞きたい人は後で懇親会で