Neovim for Rust

Florian "Florob" Zeitz

2023-07-05

Neovim

- "just" a text editor
- very configurable
- Personalized Development Environment
- just showing Rust specific bits
- adding plugins one-by-one

Base setup

- just so nobody has to ask
- folke/lazy.nvim as plugin manager
- echasnovski/mini.nvim for some basic configuration
  - colorscheme: minischeme
  - `mini.basics`: Common configuration presets
  - `mini.statusline`: Statusline
  - `mini.comment`: Comment lines
  - `mini.ai`: Extend and create `a/i` textobjects
  - `mini.indentscope`: Visualize indent scope

LSP

- Language Server Protocol
- provides
  - completion
  - diagnostics
  - syntax highlighting
  - code actions
  - formatting
  - ...

Neovim LSP

- built-in as of Neovim 0.5
- configurations managed as a separate plugin for agility
  - neovim/nvim-lspconfig
- individual configuration per language server
- setup completion and key bindings when an LSP attaches

LSP Configuration

```
local lspconfig = require'lspconfig'
lspconfig.rust_analyzer.setup{
    -- Use nightly rust-analyzer
    cmd = {'rustup', 'run', 'nightly', 'rust-analyzer'},
    settings = {
        ['rust-analyzer'] = {
            standalone = true,
            checkOnSave = {
                command = 'clippy'
            }
        }
    },
}
```

LSP Attach

- cf. neovim/nvim-lspconfig documentation

```lua
vim.api.nvim_create_autocmd('LspAttach', {
    group = vim.api.nvim_create_augroup('UserLspConfig', {}),
    callback = function(args)
        vim.bo[args.buf].omnifunc = 'v:lua.vim.lsp.omnifunc'

        local opts = { buffer = args.buf }
        vim.keymap.set('n', 'gD', vim.lsp.buf.declaration, opts)
        vim.keymap.set('n', 'gd', vim.lsp.buf.definition, opts)
        vim.keymap.set('n', 'K', vim.lsp.buf.hover, opts)
        vim.keymap.set('n', 'gi', vim.lsp.buf.implementation, opts)
        vim.keymap.set('n', '<C-k>', vim.lsp.buf.signature_help, opts)
        vim.keymap.set('n', '<space>D', vim.lsp.buf.type_definition, opts)
        vim.keymap.set('n', '<F2>', vim.lsp.buf.rename, opts)
        vim.keymap.set({ 'n', 'v' }, '<space>ca', vim.lsp.buf.code_action, opts)
        vim.keymap.set('n', 'gr', vim.lsp.buf.references, opts)
        vim.keymap.set('n', '<space>f', function()
            vim.lsp.buf.format { async = true }
        end, opts)
    end,
})
```

Inlay Hints

- lvimuser/lsp-inlayhints.nvim
- show types and parameter names
- currently at the end of lines
- will be inline with Neovim 0.10

Inlay Hint Configuration

```lua
require'lsp-inlayhints'.setup()
vim.api.nvim_create_autocmd('LspAttach', {
    group = vim.api.nvim_create_augroup('LspAttach_inlayhints', {}),
    callback = function(args)
        if not (args.data and args.data.client_id) then
            return
        end

        local bufnr = args.buf
        local client = vim.lsp.get_client_by_id(args.data.client_id)
        require'lsp-inlayhints'.on_attach(client, bufnr)
    end,
})
```

Rust Tools

- simrat39/rust-tools.nvim
- automatically configures rust-analyzer
- provides its own inlay hints
- still needs completion and key bindings
- runnables

Rust Tools Configuration

```
require'rust-tools'.setup{
    server = {
        -- Use nightly rust-analyzer
        cmd = {'rustup', 'run', 'nightly', 'rust-analyzer'},
        settings = {
            ['rust-analyzer'] = {
                standalone = true,
                checkOnSave = {
                    command = 'clippy'
                }
            }
        },
    }
}
```

Diagnostics

- Neovim has a diagnostic API `vim.diagnostic`
- originally added for LSP, generalized since Neovim 0.6
- shows diagnostics at the end of lines per default
- needs some key bindings for convenient use

Diagnostics Key Bindings

```lua
vim.keymap.set('n', '<space>e', vim.diagnostic.open_float)

-- Only jump between diagnostics which are at least WARN
local min_warn = { min=vim.diagnostic.severity.WARN };
function prev_warning()
    vim.diagnostic.goto_prev({ severity=min_warn });
end
function next_warning()
    vim.diagnostic.goto_next({ severity=min_warn });
end
vim.keymap.set('n', '[d', prev_warning)
vim.keymap.set('n', ']d', next_warning)
```

Diagnostics Display

- default diagnostic display can be a bit unwieldy at times
- ~whynothugo/lsp_lines.nvim
  - uses virtual lines instead
  - can point to the exact location in a line
- disabling the default underlining for `HINT` diagnostics can help readability for `#[cfg()]`-heavy code

Diagnostics Display Configuration

```lua
require'lsp_lines'.setup(opts)

-- Disable virtual text since we have virtual lines
vim.diagnostic.config{ virtual_text = false }

-- Disable virtual_lines in Lazy.nvim (cf. lazy.nvim #620)
vim.diagnostic.config(
    { virtual_lines = false },
    require'lazy.core.config'.ns,
)

-- Don't underline HINTs, can be annoying with #[cfg()]
vim.diagnostic.config{
  underline = { severity = { min = vim.diagnostic.severity.INFO } },
}
```

Better LSP UI

- some plugins can provide better UI for LSP
- j-hui/fidget.nvim provides an LSP progress display
- stevearc/dressing.nvim provides dialogs for actions
  - other plugins can further improve these,
    e.g. nvim-telescope/telescope.nvim

Completion

- triggering omni complete is annoying
- there is a plethora of completion plugins
- generally support for (LSP) snippets is recommended
- e.g. hrsh7th/nvim-cmp with hrsh7th/cmp-nvim-lsp

Cargo.toml

- Saecki/crates.nvim
- provides features for managing `Cargo.toml`
- show currently used dependency version (SemVer)
- update/upgrade dependencies
- en-/disable features
- show crate info
- large API, needs key bindings for convenient use

crates.nvim Key Bindings

```lua
local crates = require'crates'
crates.setup()

vim.api.nvim_create_autocmd('BufRead', {
    pattern = "Cargo.toml",
    group = vim.api.nvim_create_augroup('BufRead_cargo.nvim', {}),
    callback = function(args)
        local opts = { buffer = args.buf }
        vim.keymap.set('n', '<leader>ct', crates.toggle, opts)
        vim.keymap.set('n', '<leader>cr', crates.reload, opts)

        vim.keymap.set('n', '<leader>cv', crates.show_versions_popup, opts)
        vim.keymap.set('n', '<leader>cf', crates.show_features_popup, opts)
        vim.keymap.set('n', '<leader>cd', crates.show_dependencies_popup, opts)

        vim.keymap.set('n', '<leader>cu', crates.update_crate, opts)
        vim.keymap.set('v', '<leader>cu', crates.update_crates, opts)
        vim.keymap.set('n', '<leader>ca', crates.update_all_crates, opts)
        vim.keymap.set('n', '<leader>cU', crates.upgrade_crate, opts)
        vim.keymap.set('v', '<leader>cU', crates.upgrade_crates, opts)
        vim.keymap.set('n', '<leader>cA', crates.upgrade_all_crates, opts)

        vim.keymap.set('n', '<leader>cH', crates.open_homepage, opts)
        vim.keymap.set('n', '<leader>cR', crates.open_repository, opts)
        vim.keymap.set('n', '<leader>cD', crates.open_documentation, opts)
        vim.keymap.set('n', '<leader>cC', crates.open_crates_io, opts)
    end,
})
```