你将得到什么（目标）

- `~/.vimrc` 极简、只管**插件列表 + 全局小开关**。
- 各语言行为（键位、路径、tags）**放** `~/.vim/ftplugin/`，只在对应文件里生效，互不打架。
- 公共跳转逻辑统一放 `autoload/jump.vim`：
  - 先把当前位置压栈 → `<C-t>` 一定能回退；
  - **候选少** → 内置 `ptselect`（编号 + 预览）；
  - **候选多** → `fzf :Tags`（右侧/底部预览，可模糊过滤）；
  - 支持"优先项目内候选、去重"等优化。
- LSP（比如 C 以后用）多位置时 → **Quickfix 全屏**浏览（编号 + 预览键）。
- Python 不开 LSP，仅用 **ctags + fzf**；YAML 给一份实用 `ftplugin`（可选）。

一、装基础依赖

0）先说明两件事

- Vim 的命令**区分大小写**：是 `:CocInfo` 不是 `:cocinfo`。
- 我们**不用 coc-lua**，而是离线解压 **lua-language-server** 并让 coc 直接调用。

1）装基础依赖（Ubuntu 24.04）

```
apt-get update
apt-get install -y vim git curl nodejs npm universal-ctags || apt-get install -y
exuberant-ctags
```

（可选）第 0 步：代理

如需代理，把下面的地址换成你的代理并执行：

```
export http_proxy=http://192.168.16.106:7890
export https_proxy=$http_proxy
export no_proxy="localhost,127.0.0.1,::1,.local"
# Git 走代理
git config --global http.proxy  $http_proxy
git config --global https.proxy $https_proxy
# npm 走代理（可选换镜像）
npm config set proxy $http_proxy
npm config set https-proxy $https_proxy
# npm config set registry https://registry.npmmirror.com
```

删除代理：

```
# 取消环境变量
unset http_proxy
unset https_proxy
unset no_proxy

# Git 取消代理
git config --global --unset http.proxy
git config --global --unset https.proxy

# npm 取消代理
npm config delete proxy
npm config delete https-proxy
# 如果你之前换过 registry，这里可以还原（可选）
npm config delete registry

```

2）安装插件管理器（vim-plug）

```
curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
  https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

在 Vim 里自检：`:echo exists('*plug#begin')` 应返回 `1`。

3）离线安装 lua-language-server（仅下载 + 建软链）

> 只提供**可复制的下载命令**（不解压）。下载完成后，你再自行解压到 `/opt/lua-language-server`，最后执行 `ln -s` 建软链即可。

3.1 选择架构 & 版本

```
uname -m  # x86_64 → 选 linux-x64；aarch64/arm64 → 选 linux-arm64
VERSION=latest  # 也可改为固定版本号，如 3.10.5
```

3.2 一键下载（自动挑 glibc，排除 musl）

> 需要联网；若有代理：`export http_proxy=http://IP:PORT; export https_proxy=$http_proxy`

A) 最新版本（自动匹配架构）

```
#首先
cd /opt
```

```
ARCH=$(uname -m); PATTERN=$([ "$ARCH" = x86_64 ] || [ "$ARCH" = amd64 ] && echo linux-x64 || echo linux-arm64)
curl -sL https://api.github.com/repos/LuaLS/lua-language-server/releases/latest \
 | grep -Eo "https://[^\" ]*${PATTERN}.*\.tar\.gz" \
 | grep -vi musl \
 | head -n1 \
 | xargs -I{} sh -c 'echo downloading: {}; curl -L -o /opt/lua-language-server.tar.gz "{}"'
```

B) 指定版本（可重复构建）

```
VERSION=3.10.5   # ← 改成你要的版本
ARCH=$(uname -m); FILE=$([ "$ARCH" = x86_64 ] || [ "$ARCH" = amd64 ] && echo linux-x64 || echo linux-arm64)
URL="https://github.com/LuaLS/lua-language-server/releases/download/${VERSION}/lua-language-server-${VERSION}-${FILE}.tar.gz"
curl -L -o /opt/lua-language-server-${VERSION}.tar.gz "$URL"
```

3.3 建软链（解压后再执行）

> 你解压到 `/opt/lua-language-server` 后，再执行：

```
#解压
#sudo tar -vxf lua-language-server.tar.gz -C /opt/lua-language-server

ln -sf /opt/lua-language-server/bin/lua-language-server /usr/local/bin/lua-language-server
```

3.4 （可选）校验

```
sha256sum /opt/lua-language-server*.tar.gz   # 与发布页校验和对比（如需）
```

4）写 coc.nvim 的离线配置（`~/.vim/coc-settings.json`）

> 纯 JSON，不要注释。把 `Lua.workspace.library` 改成**你工程里模块的父目录**（例如 `src/lualib`）。

```
{
  "languageserver": {
    "lua": {
      "command": "/usr/local/bin/lua-language-server",
      "filetypes": ["lua"],
      "settings": {
        "Lua.runtime.version": "LuaJIT",
        "Lua.runtime.path": ["?.lua", "?/init.lua"],

        "Lua.workspace.checkThirdParty": false,
        "Lua.diagnostics.globals": ["vim", "skynet"],
```

```
        "Lua.workspace.library": [
          "${workspace}/src/lualib",
          "/opt/bmc/lualib",
          "/opt/bmc/app",
          "${workspace}/.lls/3rd/libmc"
        ],

        "Lua.workspace.ignoreDir": [
          "**/.git",
          "**/.cache",
          "**/luaclib"
        ],

        "Lua.workspace.maxPreload": 4000,
        "Lua.workspace.preloadFileSize": 100,
        "Lua.semantic.enable": true,
        "Lua.telemetry.enable": false
      }
    }
  }
}

```

> 如果你的模块位于 `lualib/` 而不是 `src/lualib/`，就改成 `${workspace}/lualib`；多个根就多写几条。

工程根执行

```
#manifest构建完成后，比如在bios组件下执行，或者其他组件下下执行，不要直接执行，没意义，一定要拉下代码在代码根目录执行
```

```
#!/usr/bin/env bash
set -euo pipefail

WS_ROOT="${WS_ROOT:-$(pwd)}"
LINK_DIR="$WS_ROOT/.lls/3rd"
LINK_PATH="$LINK_DIR/libmc"
mkdir -p "$LINK_DIR"

# 用法：
#   bash lls_link.sh                      # 默认找 libmc4lua
#   PKG=mdb_interface bash lls_link.sh    # 指定找某个包
#   bash lls_link.sh /abs/path/to/lualib  # 直接用显式路径
PKG="${PKG:-libmc4lua}"

have_conan() { command -v conan >/dev/null 2>&1; }
is_conan2() { conan --version 2>/dev/null | grep -qE 'Conan version 2'; }

pick_latest() {
  # 从参数列表中挑 mtime 最新的目录
  # 用 stat 便携取 mtime：Linux 用 %Y
  local latest=""
```

```
  local latest_m=0
  for d in "$@"; do
    [[ -d "$d" ]] || continue
    local m; m=$(stat -c %Y "$d" 2>/dev/null || echo 0)
    if (( m > latest_m )); then latest_m=$m; latest="$d"; fi
  done
  [[ -n "$latest" ]] && printf '%s\n' "$latest"
}

# 1) 显式路径
if [[ $# -ge 1 ]]; then
  best="$1"
else
  candidates=()

  # 2) Conan 2：优先用 conan 命令拿到包目录，再定位 lualib
  if have_conan && is_conan2; then
    # 列出本地缓存里该包的所有二进制 (package_id)
    # 示例：conan list "libmc4lua/*:*" 仅本地缓存
    while IFS= read -r ref; do
      # 过滤出以 PKG/ 开头的 recipe 行
      [[ "$ref" =~ ^[[:space:]]*$PKG/[^[:space:]]*$ ]] || continue
      # 对该 recipe 列出所有 package_id
      while IFS= read -r pkgid; do
        pkgid="${pkgid##* }"      # 行尾就是 package_id
        # 用 cache path 拿二进制包目录 ~/.conan2/p/<hash>/p
        pdir=$(conan cache path "${ref# }:${pkgid}" 2>/dev/null || true)
        [[ -n "${pdir:-}" && -d "$pdir" ]] || continue
        ll="$pdir/opt/bmc/libmc/lualib"
        [[ -d "$ll" ]] && candidates+=("$ll")
      done < <(conan list "${ref# }:*" 2>/dev/null | sed -n 's/^[[:space:]]\{10,\}\([0-9a-f]\{40\}\).*/\1/p')
    done < <(conan list "$PKG/*" 2>/dev/null | sed -n 's/^[[:space:]]\{6,\}\('"$PKG"'\S*\).*/\1/p')
  fi

  # 3) Conan 2 直接扫描（兜底）：~/.conan2/p/**/p/opt/bmc/libmc/lualib
  if ((${#candidates[@]}==0)); then
    while IFS= read -r d; do
      # 用 conan cache ref 反查名字再按 PKG 过滤（若有 conan）
      if have_conan; then
        ref=$(conan cache ref "$(dirname "$d")" 2>/dev/null || true)
        [[ -n "$ref" ]] && [[ "$ref" != "$PKG"* ]] && continue
      fi
      candidates+=("$d")
    done < <(find "$HOME/.conan2/p" -type d -path '*/p/opt/bmc/libmc/lualib' 2>/dev/null || true)
  fi

  # 4) Conan 1：原有扫描
  if ((${#candidates[@]}==0)); then
    while IFS= read -r d; do candidates+=("$d"); done < <(
```

```
      find "$HOME/.conan/data/$PKG" -type d -path '*/package/*/opt/bmc/libmc/lualib' 2>/dev/null | sort
    )
  fi

  if ((${#candidates[@]}==0)); then
    echo "not found for $PKG (conan2/1 均未找到)：请确认包已安装到本地缓存"
    echo "提示：可以先执行：conan install --requires=\"$PKG/*\" --build=missing"
    exit 1
  fi

  best="$(pick_latest "${candidates[@]}")"
fi

# 校验：必须含有一个常见文件（比如 mc/logging.lua）
if [[ ! -f "$best/mc/logging.lua" ]]; then
  echo "路径可疑：$best 里没有 mc/logging.lua"
  echo "请检查或改用显式路径：bash lls_link.sh /path/to/opt/bmc/libmc/lualib"
  exit 1
fi

ln -sfn "$best" "$LINK_PATH"
echo "[ok] linked $LINK_PATH -> $best"

```

```
# 指定只找 libmc4lua
PKG=libmc4lua bash lls_link.sh
```

二、一次性创建目录

```
mkdir -p ~/.vim/autoload ~/.vim/ftplugin ~/.vim/after/plugin
```

三、写入极简 `~/.vimrc`

> 这里只"声明插件 + 开基础开关"，**不要**写各语言的键位/autocmd——都放 `ftplugin/` 里。

```
cat <<'EOF' > ~/.vimrc
" ============================================================================
"  ~/.vimrc -- 极简主控台（只列插件 + 通用开关）
" ============================================================================

" 插件管理（你已有 ~/.vim/autoload/plug.vim）
call plug#begin('~/.vim/plugged')
  Plug 'neoclide/coc.nvim', {'branch': 'release'}   " LSP/补全框架（以后 C 用得到）
  Plug 'junegunn/fzf', { 'do': { -> fzf#install() } } " fzf 内核
  Plug 'junegunn/fzf.vim'                            " fzf 的 Vim 集成（提供 :Tags 等）
call plug#end()

" 小抄：:PlugInstall  安装插件；:PlugUpdate 更新；:PlugClean 清理
```

```
" 打开 filetype 体系（自动加载 ~/.vim/ftplugin/<lang>.vim）
filetype plugin indent on
syntax on

" 通用 tags 查找策略：工程内优先，向上查找；大小写跟随被查单词
set tags=./tags;,tags
set tagcase=match

" 让 LSP 浮窗更灵敏 / 提示更干净 / 显示符号栏
set updatetime=300
set shortmess+=c
set signcolumn=yes

" :ptselect 预览更舒服（fzf 预览布局在 after/plugin/fzf.vim 里设）
set showfulltag
set previewheight=18
set splitbelow

" 设定 <leader>（YAML 里会用到 ,yh；你也可以改成空格）
let mapleader = ","

" 重要约定：
" - 语言相关键位/路径只放 ftplugin/<lang>.vim（避免全局冲突）
" - 插件 UI/行为放 after/plugin/<name>.vim（确保插件加载后执行）
" - 公共跳转逻辑放 autoload/jump.vim（懒加载）
" ============================================================================
EOF
```

> 打开 Vim，执行 `:PlugInstall` 安装 fzf / fzf.vim / coc.nvim。

四、写入 `autoload/jump.vim`（公共跳转逻辑）

> **核心文件**：集中"入栈 + 智能分流 + 兜底"。所有语言共用这套逻辑。

```
" autoload/jump.vim
" 公共跳转工具（跨语言复用）
" - 单候选直跳；少候选 tselect（编号+预览）；多候选 fzf :Tags（可预览）
" - 兼容 Universal Ctags 的 kind 取值（f/m/c 与 function/member/class）

if exists('g:loaded_autoload_jump') | finish | endif
let g:loaded_autoload_jump = 1

" ---------- quickfix 小工具 ----------
function! jump#open_qf(height) abort
  execute 'botright copen ' . (a:height > 0 ? a:height : 8)
  try
    let w = getqflist({'winid':1}).winid
    if w > 0 | call win_gotoid(w) | endif
  catch
  endtry
```

```
endfunction

function! jump#set_qf(items, title) abort
  try
    call setqflist([], 'r', {'items': a:items, 'title': a:title})
  catch
    " 旧版 Vim 兼容
    call setqflist(a:items, 'r')
  endtry
  call jump#open_qf(8)
endfunction

" ---------- tag 匹配/排序 ----------
function! jump#tag_pattern(word) abort
  " 允许前缀（Class:method / pkg.method）；并对正则特殊字符做转义
  let w = escape(a:word, '\/.^$~[]*')
  return '^\%(.\+[:.]\)\=' . w . '$'
endfunction

function! jump#by_shorter_path(a, b) abort
  let la = len(a:a.filename)
  let lb = len(a:b.filename)
  return la == lb ? 0 : (la < lb ? -1 : 1)
endfunction

" ---------- 可调参数 ----------
if !exists('g:jump_smart_threshold')
  let g:jump_smart_threshold = 12     " <= 阈值 用 tselect；> 阈值 用 fzf :Tags
endif
if !exists('g:jump_prefer_cwd')
  let g:jump_prefer_cwd = 1           " 优先项目内候选
endif

" ---------- 统一入栈（确保 <C-t> 可回退） ----------
function! jump#push_tagstack(word) abort
  if exists('*settagstack')
    call settagstack(win_getid(), {
          \ 'items': [{'tagname': a:word, 'from': [bufnr('%'), line('.'), col('.'),
0]}]
          \ }, 'a')
  endif
endfunction

" ---------- 智能分流主逻辑（跨语言复用） ----------
function! jump#smart_pick_impl(word, lang_name, file_regex) abort
  " (1) 先入栈
  call jump#push_tagstack(a:word)

  " (2) 没有 tags 文件：直接 tselect 兜底（避免 E426）
  if empty(tagfiles())
    execute 'tselect ' . a:word
    return
  endif
```

```
69
70    let T = get(g:, 'jump_smart_threshold', 12)
71
72    " 拉取候选并按后缀/类型过滤（兼容 u-ctags 的 kind 取值）
73    let all   = taglist(jump#tag_pattern(a:word))
74    let expr  = "has_key(v:val,'filename') && v:val.filename =~# '".a:file_regex."'"
75    let expr .= " && index(['f','m','c','function','member','class'],
tolower(get(v:val,'kind',''))) >= 0"
76    let cands = filter(copy(all), expr)
77
78    " (2.1) 优先项目内候选（若有）
79    if get(g:, 'jump_prefer_cwd', 1)
80      let cwd = getcwd()
81      let in_cwd = filter(copy(cands), 'fnamemodify(v:val.filename, ":p")[:len(cwd)-1]
==# cwd')
82      if len(in_cwd) > 0
83        let cands = in_cwd
84      endif
85    endif
86
87    " (2.2) 去重：filename#line
88    let seen = {}
89    let uniq = []
90    for t in cands
91      let lnum = get(t, 'line', get(t, 'lnum', 1))
92      let key  = (has_key(t,'filename') ? t.filename : '') . '#' . lnum
93      if !has_key(seen, key)
94        let seen[key] = 1
95        call add(uniq, t)
96      endif
97    endfor
98    let cands = uniq
99
100   " (3) 分流
101   if len(cands) == 0
102     " 没有捕到候选：直接让 Vim 自己匹配（更宽松）
103     execute 'tselect ' . a:word
104     return
105   endif
106
107   if len(cands) == 1
108     let lnum = get(cands[0], 'line', get(cands[0], 'lnum', 1))
109     execute 'edit +' . lnum . ' ' . fnameescape(cands[0].filename)
110     return
111   endif
112
113   if len(cands) <= T
114     execute 'tselect ' . a:word
115     return
116   endif
117
118   " 候选很多：优先 fzf 的 :Tags（可预览/过滤）
119   if exists(':Tags')
```

```
    execute 'Tags ' . a:word
  else
    " 无 fzf.vim：回落 quickfix
    let qf = map(copy(cands),
          \ "{'filename':v:val.filename,'lnum':get(v:val,'line',
get(v:val,'lnum',1)),'col':1,'text':printf('[%s] %s',get(v:val,'kind','?'),a:word)}")
    call jump#set_qf(qf, a:lang_name . ' definitions: ' . a:word)
  endif
endfunction

" ---------- 语言包装 ----------
function! jump#smart_pick_python(word) abort
  return jump#smart_pick_impl(a:word, 'Python', '\.py$')
endfunction

function! jump#smart_pick_lua(word) abort
  return jump#smart_pick_impl(a:word, 'Lua', '\.lua$')
endfunction

```

在 Vim 里执行：`:source ~/.vim/autoload/jump.vim`

五、Python 专用 `ftplugin/python.vim`

不开 LSP；`<C-]>` 用公共"智能分流"；`<C-t>` 优先 tagstack，否则退到上一步。

```
cat <<'EOF' > ~/.vim/ftplugin/python.vim
" 仅对 Python buffer 生效

" gf/find 支持
setlocal isfname+=:
setlocal suffixesadd+=.py
setlocal path+=.,**

" 追加系统 site-packages 的 tags（若存在）
if filereadable('/usr/local/lib/python3.12/dist-packages/tags')
  setlocal tags+=/usr/local/lib/python3.12/dist-packages/tags
endif

" 若启用 venv，把 venv 的 site-packages/tags 也追加（存在才加）
if exists('$VIRTUAL_ENV')
  for p in split(globpath($VIRTUAL_ENV, 'lib/python*/site-packages/tags'), '\n')
    execute 'setlocal tags+=' . fnameescape(p)
  endfor
endif

" <C-]>：调用公共智能分流（单个直跳；少候选 ptselect；多候选 fzf）
silent! nunmap <buffer> <C-]>
nnoremap <buffer> <silent> <C-]> :call jump#smart_pick_python(expand('<cword>'))<CR>

" <C-t>：若有 tagstack 就 :pop，否则退回上一个位置（<C-o>）
```

```
silent! nunmap <buffer> <C-t>
nnoremap <buffer> <expr> <C-t> len(gettagstack(win_getid()).items)>0 ? ":pop\<CR>" : "\
<C-o>"
EOF
```

> 在 Vim 里执行：`:source ~/.vim/ftplugin/python.vim`

六、（可选）YAML 专用 `ftplugin/yaml.vim`

6.1 启用 YAML LSP（安装 + 追加偏好）

> 让 `gd / gr / K / 保存格式化` 真正工作：安装 `coc-yaml`，并在 `~/.vim/coc-settings.json` 顶层追加 3 个键（与 Lua 的 languageserver 并列）。

① 安装 coc-yaml（一次性）：

```
:CocInstall coc-yaml
```

② 在 `~/.vim/coc-settings.json` 顶层追加（与已有键并列；注意 JSON 逗号位置）：

```
{
  "coc.preferences.formatOnSaveFiletypes": ["yaml", "yml"],
  "coc.preferences.hoverTarget": "float",
  "yaml.schemaStore.enable": false
}
```

③（推荐）在需要跳转/校验的 YAML 文件首行声明真实 schema：

```
# yaml-language-server: $schema=file:///usr/share/bmcgo/schema/manifest.schema.json
```

④ 自检：

```
:CocRestart
:CocList services      " 应看到 * yaml [running]
```

6.2 UI/配色（Coc 悬浮窗不再白底）

> 把 hover 浮窗背景与边框调成和主题一致。放 **after/plugin** 层，自动随启动生效。
> 按所用编辑器选择其一（Vim8 或 Neovim）。

Vim 8：

```
cat <<'EOF' > ~/.vim/after/plugin/coc-ui.vim
" 打开真彩（支持就开）
set termguicolors

" 去掉引用高亮的黄底
hi! link CocHighlightText  Normal
```

```
hi! link CocHighlightRead  Normal
hi! link CocHighlightWrite Normal

if has('nvim')
  " -- Neovim：统一悬浮窗与边框（深色示例；浅色把颜色改浅或用 NONE）--
  hi NormalFloat guibg=#1e1e1e
  hi FloatBorder guibg=#1e1e1e guifg=#5c6370
else
  " -- Vim8：Coc 的浮窗组 --
  hi CocFloating guibg=#1e1e1e
endif

" 主题切换时保持生效（避免被 colorscheme 覆盖）
augroup CocFloatFix
  autocmd!
  if has('nvim')
    autocmd ColorScheme * hi NormalFloat guibg=#1e1e1e | hi FloatBorder guibg=#1e1e1e guifg=#5c6370
  else
    autocmd ColorScheme * hi CocFloating guibg=#1e1e1e
  endif
augroup END
EOF

```

只影响 YAML buffer；包含缩进/折叠、coc 按键、hover 自动弹等。

```
cat <<'EOF' > ~/.vim/ftplugin/yaml.vim
" 仅对 YAML buffer 生效

" 基础：2 空格缩进、按缩进折叠、不换行
setlocal expandtab
setlocal shiftwidth=2 softtabstop=2 tabstop=2
setlocal foldmethod=indent foldlevel=99
setlocal nowrap

" 保存前尽力格式化（没装 coc 也不会报错）
autocmd BufWritePre <buffer> silent! call CocAction('format')

" 补全/导航/确认
inoremap <buffer> <silent><expr> <C-y>  coc#refresh()
inoremap <buffer> <silent><expr> <Tab>   coc#pum#visible() ? coc#pum#next(1) : coc#refresh()
inoremap <buffer> <silent><expr> <S-Tab> coc#pum#visible() ? coc#pum#prev(1) : "\<S-Tab>"
inoremap <buffer> <silent><expr> <C-j>   coc#pum#visible() ? coc#pum#next(1) : "\<Ignore>"
inoremap <buffer> <silent><expr> <C-k>   coc#pum#visible() ? coc#pum#prev(1) : "\<Ignore>"
inoremap <buffer> <silent><expr> <CR>    coc#pum#visible() ? coc#pum#confirm() : "\<CR>"

nnoremap <buffer> <silent> K          :call CocActionAsync('doHover')<CR>
nnoremap <buffer> <silent> <leader>yi :call CocActionAsync('doHover')<CR>
```

```
nmap     <buffer> <silent> gd         <Plug>(coc-definition)
nmap     <buffer> <silent> gr         <Plug>(coc-references)

" 自动 hover（2s），移动/插入即关闭
if !exists('b:yaml_auto_hover') | let b:yaml_auto_hover = 1 | endif
setlocal updatetime=2000
autocmd CursorHold   <buffer> if b:yaml_auto_hover && exists('*CocAction') &&
CocAction('hasProvider','hover') | silent call CocActionAsync('doHover') | endif
autocmd CursorMoved,InsertEnter <buffer> silent! call coc#float#close_all()

" 去掉 coc 的“引用高亮”黄底色（不影响 hover）
hi! link CocHighlightText  Normal
hi! link CocHighlightRead  Normal
hi! link CocHighlightWrite Normal

" 快捷开关：,yh 切换自动 hover
nnoremap <buffer> <silent> <leader>yh :let b:yaml_auto_hover = !get(b:yaml_auto_hover,1)
\| echo 'YAML auto-hover: ' . (b:yaml_auto_hover ? 'ON' : 'OFF')<CR>
EOF
```

七 Lua 专用`ftplugin/lua.vim`

> 略

```
cat <<'EOF' > ~/.vim/ftplugin/lua.vim
" ============================================================================
"  Lua（最小可用 · 插件化）
"  流程：LSP → require 智能 → jump/ctags（或 tselect）；始终先压栈，<C-t> 可回退
"  依赖：已按文档配置 coc.nvim、autoload/jump.vim、通用 ctags 设置
" ============================================================================

" 让 gf/find 能按 Lua 模块找文件（a.b -> a/b.lua）
setlocal isfname+=:
setlocal suffixesadd+=.lua
setlocal path+=.,./lua,./src,**

" 在 require 行上，按模块名打开文件（a.b -> a/b.lua）
function! s:LuaJumpRequireUnderCursor() abort
  if &filetype !=# 'lua' | return 0 | endif
  let line = getline('.')

  " 1) 抓 require("mod") 或 require 'mod'
  let mod = matchstr(line, 'require\zs\s*(\=\s*["'']\zs[^"'' ]\+\ze["''])')
  if empty(mod) | return 0 | endif

  " 2) 可选：别名一致性（local X = require "xxx"，光标不在字符串时不触发）
  let alias = matchstr(line, '^\s*local\s\+\zs\k\+\ze\s*=\s*require')
  let word  = expand('<cword>')
  if word !=# alias && word !=# substitute(mod,. '.*[./]', '', '')
    if matchstr(line, '\%'.col('.').'c') !~# '["'']' | return 0 | endif
  endif
```

```
  " 3) 映射成文件并打开
  let file = substitute(mod, '\.', '/', 'g') . '.lua'
  try
    execute 'find ' . fnameescape(file)
    return 1
  catch
    return 0
  endtry
endfunction

" 智能跳转：LSP → require → jump/ctags（或 tselect）
function! s:LuaGoDef() abort
  " 0) 先入栈（无论走哪个分支，<C-t> 都能回退）
  if exists('*settagstack')
    let w = expand('<cword>')
    call settagstack(win_getid(), {
          \ 'items': [{'tagname': w, 'from': [bufnr('%'), line('.'), col('.'), 0]}]
          \ }, 'a')
  endif

  " 1) LSP（coc.nvim）
  if exists('*CocAction') && CocAction('hasProvider','definition')
    try | call CocAction('jumpDefinition') | return | catch | endtry
  endif

  " 2) require 智能（a.b -> a/b.lua）
  if s:LuaJumpRequireUnderCursor() | return | endif

  " 3) 兜底：你的“公共跳转分流”（少候选 tselect、多候选 :Tags）
  if exists('*jump#smart_pick_lua')
    call jump#smart_pick_lua(expand('<cword>'))
  elseif exists('*jump#smart_pick_generic')
    call jump#smart_pick_generic(expand('<cword>'))
  else
    " 没有公共分流函数时，回落到 tselect（更宽松）
    execute 'tselect ' . expand('<cword>')
  endif
endfunction

" 绑定键位（buffer-local）
silent! nunmap <buffer> <C-]>
nnoremap <buffer> <silent> <C-]> :<C-u>call <SID>LuaGoDef()<CR>

" 回退：优先 tagstack，否则退回上一个位置
silent! nunmap <buffer> <C-t>
nnoremap <buffer> <expr> <C-t> len(gettagstack(win_getid()).items)>0 ? ":pop\<CR>" : "\
<C-o>"
EOF

```

> 在Vim 里执行：source ~/.vim/ftplugin/lua.vim

八、fzf 的 UI 设置（让列表更宽、预览可切换）

> 仅影响 fzf 插件，不碰语言逻辑；**这段要在 Vim 里生效**（不是 bash）。

```
cat <<'EOF' > ~/.vim/after/plugin/fzf.vim
" 右侧预览窄一些；Ctrl-/ 开/关；长行换行；边框
let g:fzf_preview_window = ['right,38%,wrap,border', 'ctrl-/']

" 整体列表更大（底部分割占 80% 高度）
let g:fzf_layout = { 'down': '80%' }

" 重载 :Tags，让列表“路径在前、符号在后”，更容易看清来源
if exists('*fzf#vim#tags')
  command! -nargs=* -complete=tag Tags call fzf#vim#tags(
        \ <q-args>,
        \ fzf#vim#with_preview({'options': ['--delimiter=\t', '--with-nth=2..,1']}, 'right,38%')
        \ )
endif
EOF
```

> 在 Vim 里执行：`:source ~/.vim/after/plugin/fzf.vim`
> fzf 里按 **Ctrl-/** 可随时开/关预览；把预览移到底部可改成：
> `let g:fzf_preview_window = ['down,35%,wrap,border', 'ctrl-/']`

十、生成/维护 tags（Python）

> 推荐 universal-ctags。确保 `ctags --version` 可用。

```
# 在项目根生成项目 tags（函数/方法/类，不要变量/导入）
ctags -R --languages=Python --python-kinds=-iv --fields=+n -f tags .

# （可选）为系统 site-packages 生成 tags（路径按你的实际环境）
sudo ctags -R --languages=Python --python-kinds=-iv --fields=+n \
  -f /usr/local/lib/python3.12/dist-packages/tags /usr/local/lib/python3.12/dist-packages

# （可选）为 venv 的 site-packages 生成 tags（在激活 venv 后）
ctags -R --languages=Python --python-kinds=-iv --fields=+n \
  -f $VIRTUAL_ENV/lib/python*/site-packages/tags $VIRTUAL_ENV/lib/python*/site-packages
```

> `~/.vimrc` 里 `set tags=./tags;,tags` 会**自动从当前目录向上查找**名为 `tags` 的文件。
> Python 的 site-packages 专属 tags 我们在 `ftplugin/python.vim` 里用 `setlocal tags+=...` **按需追加**，不影响其他语言。

十一、日常用法小抄

- **跳到定义（Python，无 LSP）：**
  - `<C-]>`：
    - 候选 **≤ 12** → `ptselect`（编号 + 预览）
    - 候选 **> 12** → `fzf :Tags`（右侧/底部预览，可 `Ctrl-/` 开/关，直接敲字过滤）
  - **回退**：`<C-t>`（若没栈则自动退到 `<C-o>`）
- **fzf 列表**：
  - `Ctrl-/` 开/关预览；
  - 我们把 `:Tags` 列表改成“路径在前、符号在后”，更容易判别来源。
- **Quickfix（以后 LSP 用）**：
  - 多位置 → `,qq` 打开全屏 → `12G<CR>` 跳第 12 条；`p` 预览；`v/s/t` 分屏/标签打开。

十二、常见误区 & 排错

1. **别在 bash 里 `source` Vim 脚本**：
   `source ~/.vim/after/plugin/fzf.vim` 是**Vim 命令**，应该在 Vim 里 `:source`，不是在 shell 里。
2. **Perl 依赖**：`fzf.vim` 的 `:Tags` 使用 Perl 脚本解析 `tags`。容器没装 Perl 会报错。
   - Debian/Ubuntu：`apt-get update && apt-get install -y perl`
3. **fzf 改了终端 `<C-R>/<C-T>`？**
   那是 fzf 的 **shell** 绑定（跟 Vim 无关）。
   - 彻底关闭：注释 `~/.bashrc` / `~/.zshrc` 里 `source ~/.fzf.*`
   - 仅恢复按键（bash）：

     ```
     bind -r '\C-t'
     bind '"\C-r": reverse-search-history'
     ```

   - 仅恢复按键（zsh）：

     ```
     bindkey -r '^T'
     bindkey '^R' history-incremental-search-backward
     ```

4. **自检命令（在 Vim 里）：**

```
:echo exists(':Tags')            " =2 则 fzf 的 :Tags 可用
:echo executable('fzf')          " =1 则 fzf 二进制可用
:echo tagfiles()                 " 能看到项目/库的 tags
:verbose nmap <C-]>              " 映射应来自 ftplugin/python.vim（buffer-local）
:tags                            " 看 tagstack：有记录则 <C-t> 能回退
```

十三、可调参数（放 `~/.vimrc`）

```
" 候选阈值：<= 阈值 用 ptselect；> 阈值 用 fzf
let g:jump_smart_threshold = 12

" 是否优先项目内候选
let g:jump_prefer_cwd = 1
```

到这里，你的整套 Vim 配置就**完整上线**了：

- 结构清晰：**vimrc（主控台）** + **autoload（公共库）** + **ftplugin（语言内聚）** + **after/plugin（插件 UI）**。
- 交互顺手：**小候选编号预览**，**大候选 fzf 预览**，**随时可回退**。
  后续要给 Lua、C、Go 等语言加同款体验，只需在各自的 `ftplugin/<lang>.vim` 里绑定 `<C-]>` 到 `jump#smart_pick_<lang>`（或 LSP → quickfix），其它都不用动。