Использование разработчиками языковых серверов в дереве исходного кода FreeBSD

Содержание

1. Введение 1
2. Требования 1
3. Настройки редактора 2
4. База данных компиляции 4
5. Последние шаги 6

1. Введение

Это руководство посвящено настройке дерева исходных кодов FreeBSD с использованием языковых серверов для индексации исходного кода. В руководстве описаны шаги для Vim/NeoVim и VSCode. Если вы используете другой текстовый редактор, вы можете использовать это руководство в качестве справочника и найти эквивалентные команды для вашего любимого редактора.

2. Требования

Для следования этому руководству необходимо установить определённые требования. Нам понадобится языковой сервер, `ccls` или `clangd`, а также, опционально, база данных компиляции.

Установка языкового сервера может быть выполнена через `pkg` или через порты. Если мы выберем `clangd`, нам нужно установить `llvm`.

Использование `pkg` для установки `ccls`:

```
# pkg install ccls
```

Если мы хотим использовать `clangd`, нам необходимо установить `llvm` (в примере команды используется `llvm15`, но вы можете выбрать нужную версию):

```
# pkg install llvm15
```

Для установки через порты выберите предпочтительную комбинацию инструментов из

каждой категории ниже:

- Реализации языковых серверов
  - devel/ccls
  - devel/llvm12 (Другие версии также подходят, но чем новее, тем лучше. Замените `clangd12` на clangdN в случае использования других версий.)
- Редакторы
  - editors/vim
  - editors/neovim
  - editors/vscode
- Генератор базы данных компиляции
  - devel/python (Для реализации scan-build-py из llvm)
  - devel/py-pip (Для реализации scan-build от rizsotto)
  - devel/bear

3. Настройки редактора

3.1. Vim/Neovim

3.1.1. Плагины клиента LSP

Встроенный менеджер плагинов используется для обоих редакторов в этом примере. Плагин LSP-клиента, который используется, — это prabirshrestha/vim-lsp.

Для настройки клиентского плагина LSP для Neovim:

```
# mkdir -p ~/.config/nvim/pack/lsp/start
# git clone https://github.com/prabirshrestha/vim-lsp
~/.config/nvim/pack/lsp/start/vim-lsp
```

и для Vim:

```
# mkdir -p ~/.vim/pack/lsp/start
# git clone https://github.com/prabirshrestha/vim-lsp ~/.vim/pack/lsp/start/vim-lsp
```

Чтобы включить плагин LSP-клиента в редакторе, добавьте следующий фрагмент в ~/.config/nvim/init.vim при использовании Neovim или в ~/.vim/vimrc при использовании Vim:

Для ccls

```
au User lsp_setup call lsp#register_server({
```

```
    \ 'name': 'ccls',
    \ 'cmd': {server_info->['ccls']},
    \ 'allowlist': ['c', 'cpp', 'objc'],
    \ 'initialization_options': {
    \     'cache': {
    \         'hierarchicalPath': v:true
    \     }
    \ }})
```

Для clangd

```
au User lsp_setup call lsp#register_server({
    \ 'name': 'clangd',
    \ 'cmd': {server_info->['clangd15', '--background-index', '--header-
insertion=never']},
    \ 'allowlist': ['c', 'cpp', 'objc'],
    \ 'initialization_options': {},
    \ })
```

В зависимости от установленной версии `clangd` может потребоваться обновить `server-info`, чтобы указать на правильный бинарный файл.

Обратитесь к https://github.com/prabirshrestha/vim-lsp/blob/master/README.md#registering-servers, чтобы узнать о настройке сочетаний клавиш и автодополнения кода. Официальный сайт clangd находится по ссылке https://clangd.llvm.org, а репозиторий ccls — https://github.com/MaskRay/ccls/.

Ниже приведены эталонные настройки сочетаний клавиш и автодополнения кода. Поместите следующий фрагмент в ~/.config/nvim/init.vim или ~/.vim/vimrc для пользователей Vim, чтобы использовать его:

```
function! s:on_lsp_buffer_enabled() abort
    setlocal omnifunc=lsp#complete
    setlocal completeopt-=preview
    setlocal keywordprg=:LspHover

    nmap <buffer> <C-]> <plug>(lsp-definition)
    nmap <buffer> <C-W>] <plug>(lsp-peek-definition)
    nmap <buffer> <C-W><C-]> <plug>(lsp-peek-definition)
    nmap <buffer> gr <plug>(lsp-references)
    nmap <buffer> <C-n> <plug>(lsp-next-reference)
    nmap <buffer> <C-p> <plug>(lsp-previous-reference)
    nmap <buffer> gI <plug>(lsp-implementation)
    nmap <buffer> go <plug>(lsp-document-symbol)
    nmap <buffer> gS <plug>(lsp-workspace-symbol)
    nmap <buffer> ga <plug>(lsp-code-action)
    nmap <buffer> gR <plug>(lsp-rename)
    nmap <buffer> gm <plug>(lsp-signature-help)
endfunction
```

```
augroup lsp_install
    au!
    autocmd User lsp_buffer_enabled call s:on_lsp_buffer_enabled()
augroup END
```

3.2. VSCode

3.2.1. Плагины клиента LSP

Для работы демона языкового сервера необходимы клиентские плагины LSP. Нажмите `Ctrl+Shift+X`, чтобы открыть панель поиска расширений в сети. Введите `llvm-vs-code-extensions.vscode-clangd` при использовании clangd или `ccls-project.ccls` при использовании ccls.

Затем нажмите `Ctrl+Shift+P`, чтобы открыть палитру команд редактора. Введите `Preferences: Open Settings (JSON)` в палитру и нажмите `Enter`, чтобы открыть settings.json. В зависимости от реализации языкового сервера, добавьте одну из следующих пар ключ/значение JSON в settings.json:

Для clangd

```
[
    /* Begin of your existing configurations */
    ...
    /* End of your existing configurations */
    "clangd.arguments": [
        "--background-index",
        "--header-insertion=never"
    ],
    "clangd.path": "clangd12"
]
```

Для ccls

```
[
    /* Begin of your existing configurations */
    ...
    /* End of your existing configurations */
    "ccls.cache.hierarchicalPath": true
]
```

4. База данных компиляции

База данных компиляции содержит массив объектов команд компиляции. Каждый объект определяет способ компиляции исходного файла. Файл базы данных компиляции обычно называется compile_commands.json. База данных используется реализациями языковых

серверов для целей индексирования.

Пожалуйста, обратитесь к https://clang.llvm.org/docs/JSONCompilationDatabase.html#format для получения подробностей о формате файла базы данных компиляции.

4.1. Генераторы

4.1.1. Использование scan-build-py

Установка

Инструмент `intercept-build` из scan-build-py используется для создания базы данных компиляции.

Установите пакет devel/python, чтобы получить интерпретатор python. Для получения `intercept-build` из LLVM:

```
# git clone https://github.com/llvm/llvm-project /path/to/llvm-project
```

где /path/to/llvm-project/ — это желаемый путь для репозитория. Для удобства создайте алиас в файле конфигурации оболочки:

```
alias intercept-build='/path/to/llvm-project/clang/tools/scan-build-py/bin/intercept-build'
```

rizsotto/scan-build можно использовать вместо LLVM's scan-build-py. LLVM's scan-build-py был объединён в дерево LLVM из rizsotto/scan-build. Эту реализацию можно установить с помощью `pip install --user scan-build`. Скрипт `intercept-build` по умолчанию находится в ~/.local/bin.

Использование

В корневом каталоге исходного кода FreeBSD создайте базу данных компиляции с помощью `intercept-build`:

```
# intercept-build --append make buildworld buildkernel -j`sysctl -n hw.ncpu`
```

Флаг `--append` указывает `intercept-build` прочитать существующую базу данных компиляции (если она существует) и добавить результаты в базу данных. Записи с дублирующимися ключами команд объединяются. Сгенерированная база данных компиляции по умолчанию сохраняется в текущем рабочем каталоге как compile_commands.json.

4.1.2. Использование devel/bear

Использование

В корневом каталоге исходного кода FreeBSD, чтобы создать базу данных компиляции с помощью `bear`:

```
# bear --append -- make buildworld buildkernel -j`sysctl -n hw.ncpu`
```

Флаг `--append` указывает `bear` прочитать существующую базу данных компиляции, если она есть, и добавить результаты в неё. Записи с дублирующимися ключами команд объединяются. Сгенерированная база данных компиляции по умолчанию сохраняется в текущем рабочем каталоге как compile_commands.json.

5. Последние шаги

После создания базы данных компиляции откройте любой исходный файл в дереве исходного кода FreeBSD, и серверный демон LSP также запустится в фоновом режиме. Первое открытие исходных файлов в дереве src занимает значительно больше времени, прежде чем сервер LSP сможет предоставить полный результат, из-за первоначального фонового индексирования сервером LSP, который компилирует все перечисленные записи в базе данных компиляции. Однако демон языкового сервера не индексирует исходные файлы, отсутствующие в базе данных компиляции, поэтому полные результаты не отображаются для исходных файлов, которые не компилировались во время выполнения `make`.