# 7款优秀Vim插件帮你打造完美IDE

原创 乐乐QvQ 夕小瑶的卖萌屋 6月3日



一只小狐狸带你解锁 炼丹术&NLP 秘籍

集成开发环境(IDE, Integrated Development Environment )是用于提供程序开发环境的应用程序,不管是Java、C还是 Python,使用IDE编程可以帮你**检查语法、自动补全、后台编译**等,写代码就变得容易许多。另外,如果经常在服务器上工作的小伙伴一定知道Vim在进行文本编辑的时候是多么的方便。那么当vim的高效编辑性和IDE辅助编程性相融合,岂不是天下无敌?



下面介绍7款常用插件,帮你打造高可用性的VIM-IDEヾ(®°▽°®)ノ゙

# Vim-plug

既然是要安装插件,自然是少不了插件管理工具。当没有插件管理器时,Vim 用户必须手动下载 tarball 包形式的插件,并将它们解压到 ~/.vim 目录中。在少量插件的时候可以。但当他们安装更多的插件时,就会变得一团糟。所有插件文件分散在单个目录中,用户无法找到哪个文件属于哪个插件。此外,他们无法找到他们应该删除哪个文件来卸载插件。这时 Vim 插件管理器就可以派上用场。插件管理器将安装插件的文件保存在单独的目录中,因此管理所有插件变得非常容易。

Vim-plug 是一个自由、开源、速度非常快的、极简的 vim 插件管理器。它可以并行地安装或更新插件。你还可以回滚更新。它创建shallow clone最小化磁盘空间使用和下载时间。它支持按需加载插件以加快启动时间。其他值得注意的特性是支持分支/标签/提交、post-update 钩子、支持外部管理的插件等[1]。

## 1.安装

安装和使用起来非常容易。你只需打开终端并运行以下命令:

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

### 2.配置

要安装插件,你必须如下所示首先在 Vim 配置文件中声明它们。一般 Vim 的配置文件是 ~/.vimrc 。请记住,当你在配置文件中声明插件时,列表应该以 call plug#begin(PLUGIN\_DIRECTORY) 开始,并以 plug#end() 结束。

例如,我们安装 "lightline.vim" 插件。为此,请在 ~/.vimrc 的顶部添加以下行

```
1 call plug#begin('~/.vim/plugged')
2 Plug 'itchyny/lightline.vim'call
3 plug#end()
```

## 3.使用

```
1 $ vim #打开vim
2 :PlugStatus #查看插件状态
3 :PlugInstall #安装之前在配置文件中声明的插件
```

### 安装完的状态如下所示

```
1 Finished. 0 error(s).
2 [======]
3
4 - LeaderF: OK
5 - vim-airline: OK
6 - vim-airline-themes: OK
7 - vim-nerdtree-tabs: OK
8 - nerdtree: OK
9 - auto-pairs: OK
10 - nerdcommenter: OK
11 - coc.nvim: OK
```

## 还有一些常用操作如下:

```
1 :PlugUpdate #更新插件
2 :PlugDiff #查看插件的变化状态,简单地回滚有问题的插件。
3 :PlugClean #删除插件
```

更多详细细节参考: https://github.com/junegunn/vim-plug

## coc.nvim

智能补全类的插件有很多,常见的有**YouCompleteMe、deoplete.nvim、coc.nvim**等,其中YCM被誉为传说中最智能的vim补全插件了,其包括语法智能补全、语法检错、函数跳转等功能。但是其依赖环境复杂,而公司内网的开发机无法连接外网,一个个手动升级/安装依赖实在伤不起②,逐放弃。最终在对比多款智能补全插件后,为同时兼顾易安装性和易用性的前提下,选择了智能补全新秀coc.nvim。该插件亮点如下[2]:

• 多种触发方式,同时支持手工触发。默认使用 always 自动模式表示输入单词首字母以及 trigger character 时触发补全,可配置为 trigger 模式,表示仅在输入 trigger character 时触发,或者配置为 none,表示禁用自动触发。任何触发模式下都可使用 绑定的快捷键进行手工触发。

- 模糊匹配,智能大小写。同 YCM 等知名插件。
- 多 source异步并发获取。同时异步获取不同 source 结果,效率更高。
- **支持通过删除字符纠正错误输入**。为了提高过滤的效率,除非清空当前过滤字符,否则删除过多的字符不会导致补全停止,而 是触发一次针对已有补全缓存的重新过滤。

来看看补全效果吧~

### 1.安装

coc.nvim依赖nodejs,所以首先要安装nodejs

```
1 curl -sL install-node.now.sh/lts | bash
```

安装coc.nvim,同样需要用到vim-plug,在~/.vimrc文件中配置

```
1 Plug 'neoclide/coc.nvim', {'branch': 'release'}
```

在vim命令行中输入:CocInfo ,若有类似以下信息弹出表示插件安装成功

```
## versions

vim version: NVIM v0.5.0-508-gc37d9fa3d

node version: v12.16.3

coc.nvim version: 0.0.78-0033e4e624

term: xterm-color

platform: linux
```

### 2.配置

coc.nvim只是一个平台,它能够允许你安装各种语言插件,达到不同语言的补全效果。因此我们只有安装了对应的语言插件才能实现补全。以C/C++为例:

通过在vim内的命令模式输入:CocConfig 来配置coc.nvim的配置文件 coc-settings.json

```
1 {
2 "languageserver": {
3    "clangd": {
4      "command": "clangd",
5      "rootPatterns": ["compile_flags.txt", "compile_commands.json"],
6      "filetypes": ["c", "cc", "cpp", "c++", "objc", "objcpp"]
7      }
8   }
```

Ps:这种配置模式下,coc主要是依赖clangd进行代码自动化补全,所以需要事先安装好clangd。

对更多其他语言的支持可以参考: https://github.com/neoclide/coc.nvim/wiki/Language-servers#supported-features

配置完成后,就可以直接进行自动补全啦。

### 3.进阶

刚才说了coc.nvim作为一个平台,其本身也包含各种扩展,可以通过如下命令安装一些该平台的高阶扩展插件。比如,你不想配置上文说的 CocConfig ,也可以直接通过安装coc-clangd插件,就可以完成对C/C++的自动补全了。

安装命令:CocInstall 插件名

移除命令:CocUninstall 插件名

查看已安装:CocList extensions

更新命令:CocUpdate

更多有趣的coc插件参考 https://github.com/neoclide/coc.nvim

进一步的,你可以对coc进行映射扩展,使得更符合你的操作习惯。在~/.vimrc中进行如下配置

```
" Use `[g` and `]g` to navigate diagnostics
2 nmap <silent> [g <Plug>(coc-diagnostic-prev)
3 nmap <silent> ]g <Plug>(coc-diagnostic-next)
5 " GoTo code navigation.
6 nmap <silent> gd <Plug>(coc-definition)
   nmap <silent> gy <Plug>(coc-type-definition)
   nmap <silent> gi <Plug>(coc-implementation)
   nmap <silent> gr <Plug>(coc-references)
   " Use K to show documentation in preview window.
   nnoremap <silent> K :call <SID>show_documentation()<CR>
   function! s:show_documentation()
     if (index(['vim', 'help'], &filetype) >= 0)
       execute 'h '.expand('<cword>')
       call CocAction('doHover')
     endif
20 endfunction
```

## NERDTree

这个插件是几乎所有研发人员都会安装的一个插件——目录树,可以支持在不退出vim的编辑器的前提下,在文件中快速切换,同时能让开发人员快速掌握项目目录结构,是提升开发效率必不可少的工具。预览结果如下图所示:

```
Press ? for help
                                                CLASS: TreeDirNode
 (up a dir)
doc/
                                       | 10 | let s:TreeDirNode = copy(g:NERDTreeFileNode)
| 1 | let g:NERDTreeDirNode = s:TreeDirNode
 NERD tree.txt
  tags
  bookmark.vim
                                               Class method that returns the highest ca
unction! s:TreeDirNode.AbsoluteTreeRoot()
  creator.vim
                                                  let currentNode = b:NERDTree.root
while currentNode.parent != {}
  flag_set.vim
  key_map.vim
  menu_controller.vim
                                                            currentNode = currentNode.parent
   menu_item.vim
                                                return currentNode
  nerdtree.vim
  notifier.vim
  path.vim
                                            unlet s:TreeDirNode.activate
function! s:TreeDirNode.activate(...)
  tree_file_node.vim
  ui.vim
                                                  call self.toggleOpen(opts)
call self.getNerdtree().render()
nerdtree_plugin/
 exec_menuitem.vim
  fs_menu.vim
                                                   call self.putCursorHere(0, 0)
plugin
  NERD_tree.vim
                                              ' FUNCTION: TreeDirNode.addChild(treenode, inOrder) {{{I
' Adds the given treenode to the list of children for th
  nerdtree.vim
CHANGELOG
LICENCE
README.markdown
                                                                                                                                       dos | latin1 | tw=78 | vim | 1 | 10/636 (Top
```

### 1.安装

有了plug-vim安装插件就是如此的简单

```
1 call plug#begin()
2 Plug 'preservim/nerdtree'
3 call plug#end()
```

### 2.配置

NERDTree默认无须配置即可直接使用,当然更改部分映射后,可以使得目录树试用起来更加得心应手。最常见的配置在~/.vimrc添加如下命令,即可使用Ctrl+n快速开启目录树。

```
1 map <C-n> :NERDTreeToggle<CR>
```

## 3.使用

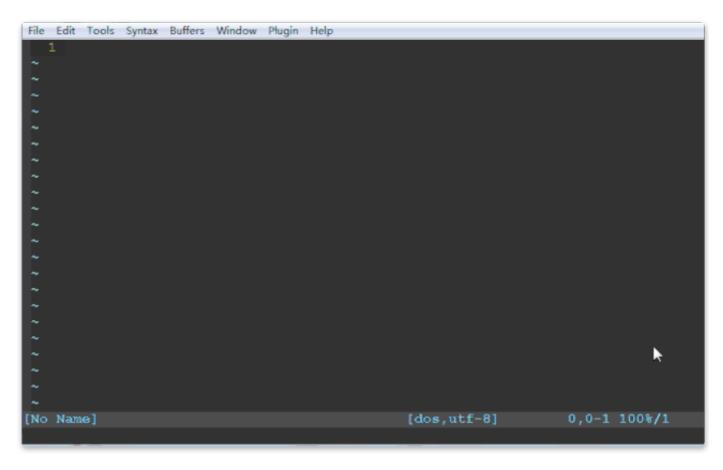
目录树的使用主要通过在vim的command模式下键入如下命令,即可达到相应的效果。

?:快速帮助文档

- o: 打开一个目录或者打开文件,创建的是 buffer,也可以用来打开书签
- go: 打开一个文件, 但是光标仍然留在 NERDTree, 创建的是 buffer
- t: 打开一个文件, 创建的是Tab, 对书签同样生效
- T: 打开一个文件,但是光标仍然留在 NERDTree,创建的是 Tab,对书签同样生效
- i:水平分割创建文件的窗口,创建的是 buffer
- gi:水平分割创建文件的窗口,但是光标仍然留在 NERDTree
- s:垂直分割创建文件的窗口,创建的是 buffer
- gs:和gi,go类似
- x: 收起当前打开的目录
- X: 收起所有打开的目录
- e: 以文件管理的方式打开选中的目录
- D:删除书签

## leaderF

提到vim的模糊查找插件,很多人第一反应是ctrlp.vim,ctrlp知名度很高,但跟其它的同类插件相比,它的唯一优点是用vimL编写(这让它的性能在同类插件中并不算优秀)。这里向大家推荐一款模糊查找插件——LeaderF,无论是从性能还是匹配精度上,都远远超越ctrlp[3]。



衍生功能:快速打开或定位某个buffer、最近使用的文件(mru)、tags(包括函数、类、变量等)、命令历史、文件中的某一行、vim的help、marks等等。

#### 1.安装

安装一如既往的简单

```
1 Plug 'Yggdroot/LeaderF', { 'do': './install.sh' }
```

#### 2.使用

leaderF几乎所有的搜索特性都不需要额外的配置,只要装好LeaderF插件就可以使用了,不像有的插件,配置就像一门新的脚本语言。下面说一下常用的命令:LeaderfFile 搜索当前目录下的文件:LeaderfBuffer 搜索当前的Buffer:LeaderfMru 搜索最近使用过的文件(search most recently used files)就是Mru

:LeaderfLine 搜索当前文件中有的某个单词

:LeaderfFunction 搜索当前文件的函数(这个很有意思,如下图列出该文件中所有的函数和变量)

```
"""Various utility functions.
  from collections import namedtuple, Counter
 4 from os.path import commonprefix
6
  __unittest = True
8 \text{ _MAX\_LENGTH} = 80
9 _PLACEHOLDER_LEN = 12
10 _MIN_BEGIN_LEN = 5
11 _{\text{MIN}}_{\text{END}}_{\text{LEN}} = 5
12 MIN_COMMON_LEN = 5
13 _MIN_DIFF_LEN = _MAX_LENGTH - \
14
                   (_MIN_BEGIN_LEN + _PLACEHOLDER_LEN + _MIN_COMMON_LEN +
1 v
         unittest = True
                            [:6 1]
       _{MAX}_{LENGTH} = 80
                            [:8 1]
 2 v
       _PLACEHOLDER_LEN = 12
3 v
                                [:9 1]
4 V
       _MIN_BEGIN_LEN = 5 [:10 1]
 5
       _{MIN}_{END}_{LEN} = 5
  V
                            [:11 1]
       _{MIN} COMMON_LEN = 5 [:12 1]
 6
  V
 7
       (_MIN_BEGIN_LEN + _PLACEHOLDER_LEN + _MIN_COMMON_LEN + [:14 1]
       def _shorten(s, prefixlen, suffixlen): [:18 1]
 8
       def _common_shorten_repr(*args):
 9
                                            [:24 1]
10
       def safe_repr(obj, short=False):
                                             [:45 1]
11
       def strclass(cls): [:54 1]
12 f
       def sorted_list_difference(expected, actual):
                                                          [:57 1]
13
       def unorderable_list_difference(expected, actual): [:98 1]
14
  f
       def three_way_cmp(x, y):
                                    [:115 1]
15
       _Mismatch = namedtuple('Mismatch', 'actual expected value') [:119 1]
  ٧
       def _count_diff_all_purpose(actual, expected): [:121 1]
16
       def _count_diff_hashable(actual, expected): [:156 1]
```

# auto-pairs

这个就是插件的功能简单而实用: 在输入/删除左括号时,能自动补上/删除右括号。

## 具体功能如下:

功能	支持	原文本	按键	新文本
成对插入	{}, [], (), "", ", "		]	[1]
成对删除	{}, [], (), "", ", "	foo[ ]	BACKSPACE	fool
换行并自动缩进	(}, □, ()	node{ }	ENTER	node{     
在括号内两侧各插入空格	{}, [], ()	foo{ }	SPACE	foo{ }
词后单引号不成对插入	,	fool	,	foo'
跳过右括号	{}, [], ()	[ foo  ]	1	[ foo ]
在转义符\后禁用插件	{}, [], (), "", ", "	foo\	{	foo\{
对字符串加小括号	C风格字符串	l'foo'	ALT+e	('foo')
删除重复成对符号	{}, [], (), ", "", "	foo"' "'	BACKSPACE	fool
飞行模式,跳出括号对而不插入	(), [], ()	if(a[3 ])	)	if(a[3])
撤销飞行模式,插入而不是跳出括号对	{}, [], ()	if(a[3])	ALT+b	if(a[3])

## 1.安装

```
1 Plug 'jiangmiao/auto-pairs'
```

## 2.使用

开箱即用的插件,无需过多的配置。

```
1 au FileType FILETYPE let b:AutoPairs = {"(": ")"}
2 au FileType php let b:AutoPairs = AutoPairsDefine({'<?' : '?>', '<?php': '?>'})
```

## vim-airline

vim-airline是vim的底部状态增强/美化插件,很好的贯彻了代码能力怎么样咱先不提,这个逼格一定要先上来。



具体效果如图所示,值得一提的是,当该插件搭配具备代码检测功能的插件时,可以实时提示该文件有多少个报错和警告等有用 信息。

#### 1.安装

```
1 Plug 'vim-airline/vim-airline'
2 Plug 'vim-airline/vim-airline-themes'
```

其中vim-airline-themes是主题插件,可以使得状态栏的颜色更加丰富。

#### 2.配置

```
1 set laststatus=2 "永远显示状态栏
2 let g:airline_powerline_fonts = 1 " 支持 powerline 字体
3 let g:airline#extensions#tabline#enabled = 1 "显示窗口tab和buffer
4 let g:airline_theme='moloai' " murmur配色不错
5
6 if !exists('g:airline_symbols')
7 let g:airline_symbols = {}
8 endif
9 let g:airline_left_sep = '▶'
10 let g:airline_left_alt_sep = '>'
11 let g:airline_right_sep = '◄'
12 let g:airline_right_alt_sep = '\{'}
13 let g:airline_symbols.linenr = '¶'
14 let g:airline_symbols.branch = '₂'
```

## NerdCommenter

如果你是一个酷爱写注释的程序员的话,那么你一定要用一下nerdcommenter(当然,如果你不希望你的代码被后人吐槽的话, 还是多写注释吧)。



NerdCommenter和Vim的Visual模式结合可以快速的注释/取消注释多行代码,同时在行尾追加注释并自动进入Insert模式可以方便的进行行内注释[4]。

### 1.安装

```
1 Plug 'preservim/nerdcommenter'
```

#### 2.配置

```
" Add spaces after comment delimiters by default
let g:NERDSpaceDelims = 1
" Use compact syntax for prettified multi-line comments
let g:NERDCompactSexyComs = 1
" Align line-wise comment delimiters flush left instead of following code indentation
let g:NERDDefaultAlign = 'left'
" Set a language to use its alternate delimiters by default
let g:NERDAltDelims_java = 1
" Add your own custom formats or override the defaults
let g:NERDCustomDelimiters = { 'c': { 'left': '/**', 'right': '*/' }
" Allow commenting and inverting empty lines (useful when commenting a region)
let g:NERDCommentEmptyLines = 1
" Enable trimming of trailing whitespace when uncommenting
```

```
20 let g:NERDTrimTrailingWhitespace = 1
21
22 " Enable NERDCommenterToggle to check all selected lines is commented or not
23 let g:NERDToggleCheckAllLines = 1
```

## 3.使用

- 1 \cc 注释当前行和选中行
- 2 \cn 没有发现和\cc有区别
- 3 \c<空格> 如果被选区域有部分被注释,则对被选区域执行取消注释操作,其它情况执行反转注释操作
- 4 \cm 对被选区域用一对注释符进行注释,前面的注释对每一行都会添加注释
- 5 \ci 执行反转注释操作,选中区域注释部分取消注释,非注释部分添加注释
- 6 \cs 添加性感的注释,代码开头介绍部分通常使用该注释
- 7 \cy 添加注释,并复制被添加注释的部分
- 8 \c\$ 注释当前光标到改行结尾的内容
- 9 \cA 跳转到该行结尾添加注释,并进入编辑模式
- 10 \ca 转换注释的方式,比如: /\*\*/和//
- 11 \cl \cb 左对齐和左右对其,左右对其主要针对/\*\*/
- 12 \cu 取消注释

更多功能参考: https://github.com/preservim/nerdcommenter



## 参考文献

[1]: Vim-plug: 极简 Vim 插件管理器 https://linux.cn/article-9751-1.html

[2]:Coc.nvim 系列: 为了更好的补全体验 https://zhuanlan.zhihu.com/p/39302327

[3]:让人相见恨晚的vim插件: 模糊查找神器LeaderF https://www.jianshu.com/p/

[4]:Vim十大必备插件 https://www.open-open.com/lib/view/open1414227253419.html



#### 夕小瑶的卖萌屋

关注&星标小夕,带你解锁AI秘籍 订阅号主页下方**「撩一下」**有惊喜哦

声明:pdf仅供学习使用,一切版权归原创公众号所有;建议持续关注原创公众号获取最新文章,学习愉快!