Vim自动补全

# Vim配置文件

Vim默认搜索和加载用户自定义配置文件的地址顺序是

$HOME/.vimrc

$HOME/.vim/vimrc

所以之后的一些配置建议大家都是添加到这两个文件中，当然推荐使用第二种，因为可以有一个.vim目录来放置一些第三方包

# vundle

vim自动补全利用开源Vim插件管理器安装，gmarik/Vundle.vim是目前被推荐次数更多的Vim插件管理器。可以参照官方介绍

<https://github.com/VundleVim/Vundle.vim>

1. 下载Vundle.vim（vim安装插件的工具）.

Vundle是一个第三方的vim插件自动管理工具，简单好用，它能够利用github上面的资源来管理vim。当vim加载了vundle，只需要告诉vundle，要使用哪些第三方

的插件，那么vundle自动会从github上面下载相关源代码放置到对应的目录中

如果没有git要先安装，或者直接使用已下载好的Vundle.vim，这里使用~/.vim/bundle作为放置vim第三方包的目录

mkdir –p ~/.vim/bundle

git clone https://github.com/VundleVim/Vundle.vim.git ~/.vim/bundle/Vundle.vim

root@kali:~/.vim/bundle# git clone https://github.com/VundleVim/Vundle.vim.git ~/.vim/bundle/Vundle.vim

正克隆到 '/root/.vim/bundle/Vundle.vim'...

remote: Enumerating objects: 3136, done.

remote: Total 3136 (delta 0), reused 0 (delta 0), pack-reused 3136

接收对象中: 100% (3136/3136), 933.42 KiB | 602.00 KiB/s, 完成.

处理 delta 中: 100% (1105/1105), 完成.

1. 添加相应插件

在vim配置文件 $HOME/.vim/vimrc 中添加以下代码

" 加载默认的vim配置

source $VIMRUNTIME/defaults.vim

" 添加vundle的配置，实现vundle包管理功能

" 注意以下路径必须跟你Vundle的路径一致

let &rtp.=',~/.vim/bundle/Vundle.vim'

" 以下调用表示开始输入需要的第三方插件

call vundle#begin()

" 把Vunle自身也添加到插件列表中来是为了实现自动更新

Plugin 'VundleVim/Vundle.vim'

" AutoComplPop插件是为了自动弹出提示框

Plugin 'vim-scripts/AutoComplPop'

" YCMD插件是主要的自动补全插件

Plugin 'Valloric/YouCompleteMe'

" 以下调用是为了结束第三方插件的输入

call vundle#end()

1. 执行插件安装命令

重启vim，然后再命令模式中输入以下命令

:PluginInstall

最后就是漫长的等待，vundle自动从github上面下载第三方插件，结束之后如下

  " Installing plugins to /root/.vim/bundle

. Plugin 'VundleVim/Vundle.vim'

. Plugin 'vim-scripts/AutoComplPop'

. Plugin 'Valloric/YouCompleteMe'

\* Helptags

说明已经下载完成，这时候进入到

$HOME/.vim/bundle可以看到以下目录

root@kali:~/.vim/bundle# pwd

/root/.vim/bundle

root@kali:~/.vim/bundle# ls

AutoComplPop  Vundle.vim  YouCompleteMe

说明插件代码都下载好了

# 编译YouCompleteMe

YouCompleteMe(简称ycmd)是一个vim/emacs等编辑器自动补全的插件，在使用之前首先要编译插件为动态库供编辑器使用

root@kali:~/.vim/bundle# cd $HOME/.vim/bundle/YouCompleteMe

root@kali:~/.vim/bundle/YouCompleteMe# ls

appveyor.yml  ci           CODE\_OF\_CONDUCT.md  COPYING.txt  install.py  plugin          python     run\_tests.py  tox.ini

autoload      codecov.yml  CONTRIBUTING.md     doc          install.sh  print\_todos.sh  README.md  third\_party

确定已经安装了以下库

sudo apt install build-essential cmake python3-dev python-dev

执行YouCompleteMe目录中的./install.py

root@kali:~/.vim/bundle/YouCompleteMe# ./install.py --clang-completer

最终确定没有报错，并且成功生成ycmd\_core.so说明正常

# YCMD项目配置文件

正如很多IDE，比如visual studio 项目中会有相应的.sln和.vcxproj项目文件，qt项目有相应的.pro文件一样。使用YCMD也要有一个配置文件，不然自动补全不知道去哪个目录查找头文件去哪个目录查找cpp文件，都会使自动补全功能不能达到我们想要的效果。所以需要一个文件来规范项目的查找路径，编译规则，用C还是C++等内容。

需要在你的根目录中创建一个文件.ycm\_extra\_conf.py，是一个简单的脚本文件，vim编辑项目中的C/C++文件的时候会一直顺着目录往上查找有没有这个文件，该文件具体配置内容看文档<https://github.com/Valloric/ycmd>以及一个样本<https://github.com/Valloric/ycmd/blob/master/.ycm_extra_conf.py>

但是为了方便我们自身开发，应该懂得使用工具来构造这个配置文件，

该工具可以在github上面找到<https://github.com/rdnetto/YCM-Generator>

还是使用git来将它克隆下来

git clone <https://github.com/rdnetto/YCM-Generator.git>

最终里边会有类似于这种文件

root@kali:~/Downloads/YCM-Generator# ls

config\_gen.py  fake-toolchain  LICENSE  plugin  README.md  template.py

root@kali:~/Downloads/YCM-Generator# ./config\_gen.py

usage: config\_gen.py [-h] [-v] [-f] [-m MAKE]

                     [-b {cmake,autotools,qmake,make}] [-c COMPILER]

                     [-C CONFIGURE\_OPTS] [-F {ycm,cc}] [-M MAKE\_FLAGS]

                     [-o OUTPUT] [-x {c,c++}] [--out-of-tree]

                     [--qt-version {4,5}] [-e]

                     PROJECT\_DIR

config\_gen.py: error: too few arguments

其中config\_gen.py是一个python脚本，可以根据当前目录中的makefile或者cmake等文件进行自动生成一个项目配置文件，最好将这个文件添加到path目录中以便可以在任何目录使用。所以以下我们来进行一个实验

# 实践

创建一个目录，里边文件大致如下

root@kali:~/Downloads/test# tree .

.

├── CMakeLists.txt

├── include

│   └── Person.h

├── main.cpp

└── src

    └── Person.cpp

其中Person头文件和cpp文件分别放置于不同的目录include和src目录

其中Person.h内容如下

#ifndef \_PERSON\_H

#define \_PERSON\_H\_

class Person

{

public:

        void doSth();

};

#endif

Person.cpp内容如下

#include "Person.h

#include <iostream>

using namespace std;

void Person::doSth()

{

        cout<<"Person Hello World"<<endl;

}

main.cpp作为程序主入口，以及一个CMake文件，内容如下：

#添加要编译的文件以及最后生成目标

add\_executable(test src/Person.cpp main.cpp)

#添加要包含的目录

include\_directories(./include)

使用config\_gen.py生成ycmd的配置文件

root@kali:~/Downloads/test# ../YCM-Generator/config\_gen.py .

Running cmake in '/tmp/tmp01g0Tf'...

$ cmake /root/Downloads/test

Running make...

$ make -i -j4

Cleaning up...

Build completed in 1.96 sec

Collected 0 relevant entries for C compilation (0 discarded).

Collected 3 relevant entries for C++ compilation (0 discarded).

Created YCM config file with 1 C++ flags

表明已经生成成功，可以打开.ycm\_extra\_conf.py文件来查看一下

import os

import ycm\_core

flags = [

    '-x',

    'c++',

    '-I/root/Downloads/test/./include',

]

看到该文件开始部分有以下标志信息，其中flags数组里边都是一些字符串，这些字符串其实就是gcc命令编译执行需要的选项，可以看到已将将之前我们的include目录包含进搜索目录选项中了。当然也可以手动修改这个文件，做一些选项的调整。

最后我们再来验证一下自动补全的功能

vim打开main.cpp进行编辑可以发现有以下提示

"main.cpp" 5L,

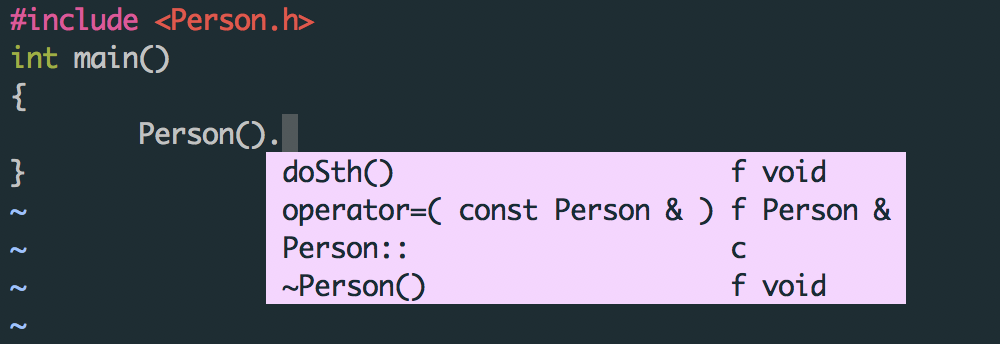
Found /root/Downloads/test/.ycm\_extra\_conf.py. Load?

(Question can be turned off with options, see YCM docs)

[O]k, (C)ancel:

意思是说找到了一个ycmd项目配置文件 .ycm\_extra\_conf.py，要不要加载？选择o就可以了

这时候再main函数中调用Person的成员方法已经有了自动提示



# 总结

使用vim的插件管理vundle 是为了方便管理第三方插件，少走很多配置的弯路。

使用YouCompleteMe需要加载响应的项目配置文件以确定要从哪个目录去搜索头文件等信息。而配置文件可以手写（比较费力，还要关注python语法以及对应接口），使用开源的工具直接生成该配置文件就会轻松很多，当然前提是项目有自己的makefile、cmake、qmake等自动构造的文件。所以以后想要使用自动配置就先写cmake等构造文件然后再使用ycmd自动补全。