



下载APP



## 13 | YouCompleteMe: Vim 里的自动完成

2020-08-26 吴咏炜

Vim 实用技巧必知必会

[进入课程 >](#)**讲述：吴咏炜**

时长 19:06 大小 17.50M



你好，我是吴咏炜。

在集成开发环境里，自动完成是一个非常重要的功能。可是 Vim 并不能真正理解你输入的代码，因此它自身无法提供自动完成的功能。不过，Vim 仍然提供了一些接口，允许第三方的软件实现这样的功能，并和 Vim 自身进行集成。🔗 [YouCompleteMe](#)（简称 YCM）就是这样的一个第三方软件，今天，我就为你详细介绍一下它。

YCM 对 C++ 程序员最为适合，它可以提供其他工具实现不了的功能。而且，它也适用于很多其他语言，包括 C 家族的各种语言和其他常用的语言，如 Python、Java 和 Go。🌟 即使在 YCM 不直接支持你使用的语言的时候，它仍然能通过标识符完成功能提供比没有 YCM（和其他语言支持插件）时更好的编辑体验。因此，我推荐你使用这个插件。

# YouCompleteMe

## 功能简介

首先我来介绍一下 YCM 的基本功能吧。根据它的主页（我的翻译）：

YouCompleteMe 是一个快速、即输即查、模糊搜索的 Vim 代码完成引擎。它实际上有好几个完成引擎：

- 一个基于标识符的引擎，可以在任何编程语言中工作

- 一个强大的基于 clangd 的引擎，可以为 C/C++/Objective-C/Objective-C++/CUDA（C 家族语言）提供原生的语义代码完成

- 一个基于 Jedi 的完成引擎，可以支持 Python 2 和 3

- 一个基于 OmniSharp-Roslyn 的完成引擎，用来支持 C#

- 一个基于 Gopls 的完成引擎，支持 Go

- 一个基于 TSServer 的完成引擎，支持 JavaScript 和 TypeScript

- 一个基于 rls 的完成引擎，支持 Rust

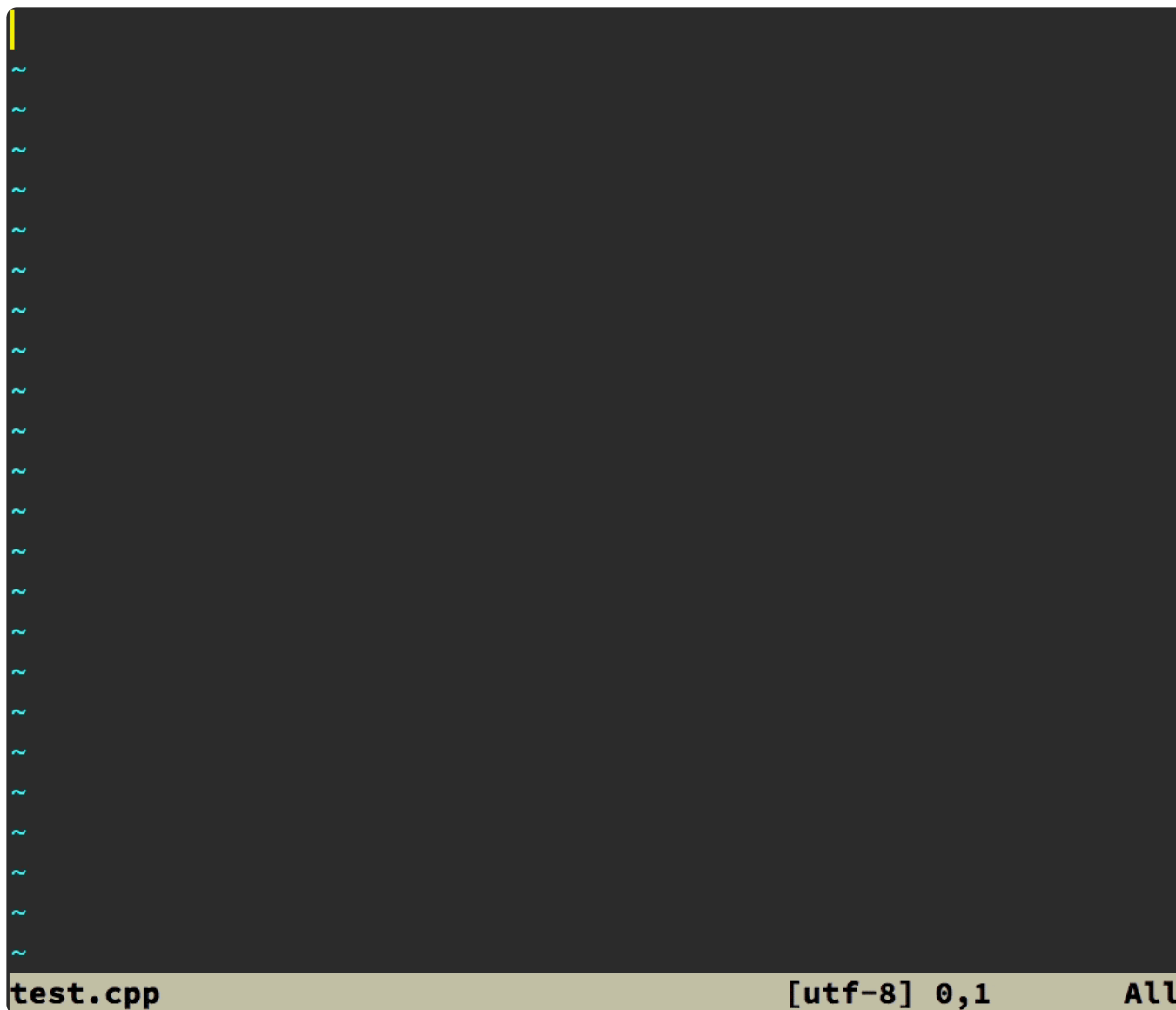
- 一个基于 jdt.ls 的完成引擎，支持 Java

- 一个通用的语言服务器协议（LSP）实现，用来支持任何其他有 LSP 服务器的语言

还有一个基于 omnifunc 的完成器，使用 Vim 的全能补全（omnicomplete）系统提供的数据来为很多其他语言提供语义完成

其实，Vim 里的自动完成插件并不止这一个，但 YCM 是比较成熟也比较全面的。虽说它的安装配置有一定的复杂性，但比起另外一些要求你独立安装、配置语言服务器的方案，它至少能一次性搞定插件和你需要的语言支持，所以反而算是简单的了。我最近的主要开发语言是 C、C++、Python 和 Vim 脚本，因此这也算是个很完美的匹配了。

下面是一个简单的示例，展示了 YCM 的效果。



YCM 的 C++ 示例

总体上，你只要在 YCM 给你提示的时候，敲 <Tab> 来选择合适的选项，然后继续往下输入就行。由于 YCM 使用模糊匹配，你只要输入你希望的标识符中的每一段中的若干字符，就可以快速把候选项减到你要的内容敲一两下 <Tab> 就能出来。事实上，我后来发现，在 `std::` 后只要输入 `mu` 就足以让 `make_unique` 成为第一选择了。

不过，这里面最让我吃惊的还是，`clangd` 引擎居然能在我只提供部分头文件的情况下（完全不提供是不行的），自动帮我插入正确的头文件并保持其字母序排列。这个功能我以前还真还没有见过！

## 安装

### Ubuntu 下的 apt 安装

如果你使用一个较新的 Linux 发布版，有可能系统本身已经自带了 YCM。虽然这个版本多半会有点老，但对于有些人来说，可能也够用了。毕竟，Linux 下的包安装确实方便。我们就先以 Ubuntu 为例，来介绍 Linux 包管理器下的安装过程。

首先，我们需要使用 apt 命令来安装 YCM，命令是：

```
1 sudo apt install vim-youcompleteme
```

[复制代码](#)

这步成功之后，YCM 就已经被安装到了你的系统上。不过，在你个人的 Vim 配置里，仍然还没有启用 YCM。要启用的话，可以输入下面的命令：

```
1 vim-addon-manager install youcompleteme
```

[复制代码](#)

这个命令之后，你会在你的 ~/.vim/plugin 目录下看到 youcompleteme.vim 的符号链接。这样，安装就算完成了。

## 手动安装

如果你的系统不直接提供 YCM，或者你想要使用最新版本的 YCM，那你就需要手工编译安装了。安装之前，你需要确保你的系统上有 CMake、Python 3 和平台主流的 C++ 编译器，即 Linux 上的 GCC，macOS 上的 Clang，及 Windows 上的 MSVC。如果要安装其他语言（如 Java 和 Go）的支持，也同样要准备好相应语言的环境，这些在 YCM 的主页上有介绍，我就先不多说了。

因为 YCM 是一个需要编译组件的插件，所以我不建议你用 Vim 的包管理器来安装，那样会出什么错都搞不清楚。大致安装过程是：

1. 选择安装目录
2. 签出 YCM
3. 根据你需要使用的语言使用合适的选项，来进行编译安装

下面，我们就快速地过一下。

首先，我们需要给 YCM 一个独立的安装目录。这个目录应该在 pack 下面，但不要放在包管理器使用的目录下，以免发生冲突。我的选择是“我的”，my。因为希望 YCM 直接启动，所以最后需要放到这个目录的 start 子目录下。换句话说，Unix 上的 `~/.vim/pack/my/start`，Windows 上的 `~\vimfiles\pack\my\start`。

然后，我们就应当在这个目录下签出 YCM。可以在进到这个子目录里面后，使用下面的命令（Windows 下面去掉“\”全部写一行，或者把“\”换成“^”）：

 复制代码

```
1 git clone --recurse-submodules \  
2     --shallow-submodules \  
3     https://github.com/ycm-core/YouCompleteMe.git
```

最后就是编译安装了。主要工作由 `install.py` 来完成，但如果我们不提供额外的选项，YCM 不会安装上面说的那些特定语言的语义完成引擎。我们需要显式地提供相应语言的选项：

`--clang-completer`，基于 `libclang` 的老 C 族语言引擎

`--clangd-completer`，基于 `clangd` 的新实验 C 族语言引擎

`--cs-completer`，C# 引擎

`--go-completer`，Go 引擎

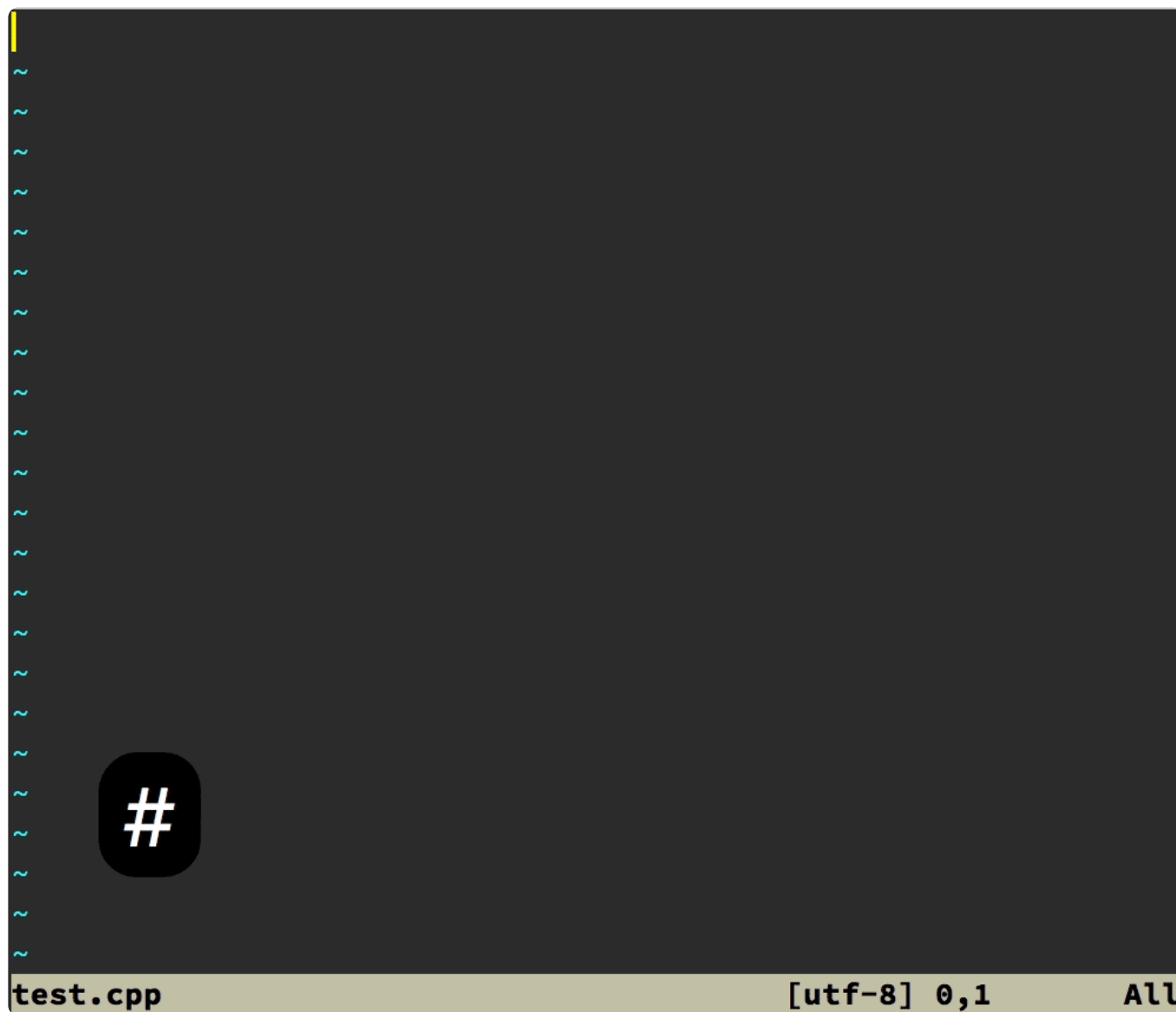
`--rust-completer`，Rust 引擎

`--java-completer`，Java 引擎

`--ts-completer`，JavaScript 和 TypeScript 引擎

`--all`，除 `clangd` 外的上述索引引擎

关于 `clangd`，我多说一句。虽然这个引擎被标为实验状态，但它的易用性和功能确实比老的引擎有了巨大的提升。同样是上面的代码，如果用老的 `libclang` 引擎的话，效果是这样的：



使用 libclang 的自动完成示例

我们可以看到：

老版本不会添加 `#include` 结束的 `>` 或 `"`

老版本不会自动添加头文件

老版本不会提供函数原型提示

老版本不会在输入中时刻提醒当前有错误（这倒不算是件坏事）

所以，如果你编译和使用 `clangd` 支持没有问题，那就用它吧。对我来说，使用 `libclang` 引擎可能有两个理由：

`clangd` 支持编译不过（我遇到过）

机器配置低, clangd 太慢了 (我的机器上能感到性能差异, 但 clangd 的响应速度完全可以接受)

你可以同时安装这两个引擎, 然后通过你的 vimrc 配置文件来选择使用哪一个, 使用 `let g:ycm_use_clangd = 0` 就是使用老引擎, 这个值设为 1 或者干脆不设, 则是使用新引擎。

另外一个要提醒你的地方是, 编译环境应尽可能干净, 不要暴露出自己用的第三方库的路径。我就碰到过因为环境变量里设了 Boost 库的包含路径, 从而导致 YCM 编译出错的情况。YCM 自己已经包含了所需的依赖库, 系统的和用户自己安装的类似库如果版本不合适的话, 反而会对 YCM 造成干扰。

(此外, 也告诉你一下我编译 clangd 时遇到的失败情况, 也供你参考一下。我的原因是自己编译了 Python 3, 由于系统上缺了一些开发包, 导致 Python 功能不完整, 到运行 YCM 的安装脚本时才暴露出来。我很高兴我后来花点时间解决了问题, 因为 clangd 的功能真的强大很多。)

## 配置

### 项目配置

像我上面的简单例子, YCM 是可以不需要配置就能直接工作的。但如果环境稍微复杂一点, C/C++ 程序就可能会出现识别错误。原因通常是以下三种:

头文件没找到, 可能是因为项目内部路径比较复杂, 也可能是因为编译器的头文件不是在 Clang 查找的默认位置下面

项目需要特别的宏定义

项目需要特定的 C 或 C++ 标准, 或特定的编译器选项

这些情况如果出现的话, 你需要让 YCM 了解项目的相应信息。

YCM 在 clangd 下的推荐做法是在源代码或其某个父目录下放一个 CMake 输出的 `compile_commands.json` 文件 (可在 cmake 命令行上加上 `-DCMAKE_EXPORT_COMPILE_COMMANDS=1` 来产生此文件)。这种方式最为通用和严格, 因为 CMake 输出的这个编译命令文件里包含每一个源文件的编译命令, 因此只要你的

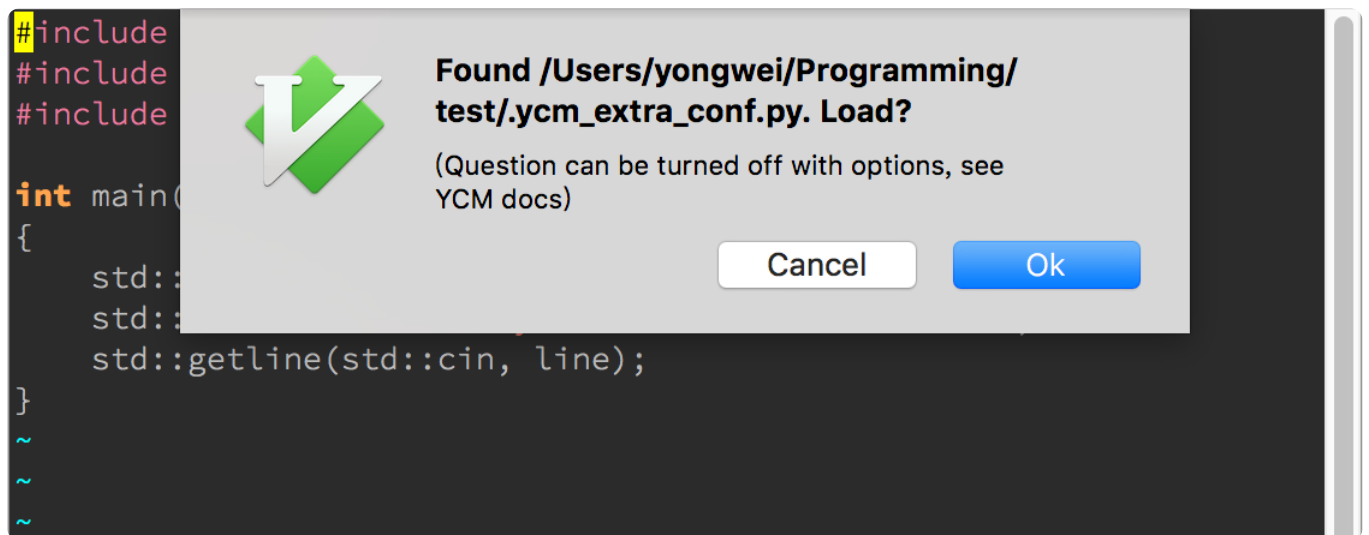
CMake 配置是正确的，YCM 通常就能正确识别，哪怕你每个文件的编译选项不同都没有关系。它可以给 YCM 提供完整的项目编译信息，使得查找一个符号的引用成为可能。如果你的工程里，这个文件产生在一个 build 目录下的话，别忘了你需要在项目的根目录下执行类似下面的命令：

```
1 ln -s build/compile_commands.json .
```

[复制代码](#)

使用 compile\_commands.json 也有缺点。CMake 在这个编译命令文件里放置的是绝对路径，因此，把源代码放在一个共享位置供不同的系统使用就会有麻烦。此外，只有少数 CMake 产生器支持 CMAKE\_EXPORT\_COMPILE\_COMMANDS，特别是，Windows 上默认的 Visual Studio 产生器不支持输出这个文件。抛开这些问题，如果项目比较大的话，clangd 的“编译”会消耗 CPU 和内存，也是一个可能的负面因素。但我们同时要记住，这种开销是让 YCM 能看到整个项目而不只是单个文件的代价。

如果因为某种问题你决定不使用编译命令文件这一方法，那 YCM 的经典做法是在文件所在目录或其父目录下寻找一个名字叫 .ycm\_extra\_conf.py 的文件。找到之后，它就会弹出一个提醒，提示用户是不是要载入这个文件，运行其中的代码来得到需要的配置信息。



YCM 的配置载入提示

虽然这个恼人的提示可以关掉，但这实际上会带来潜在的安全问题，毕竟 YCM 是会运行其中的 Python 代码的。此外，写这个文件也算是件麻烦事吧，尤其对不熟悉 Python 的人而言。



在 2014—2015 年期间，有人维护了一个叫 ycmconf 的插件，以一种我喜欢的方式解决了这个问题。不过，今天再直接用这个插件，就有点问题了。所以我在 GitHub 上复刻了这个插件，并进行了更新。如果你使用 YCM 进行 C 系语言的自动完成，那我推荐你安装 adah1972/ycmconf 这个插件。

这个插件支持两种简单的方式来配置 C/C++ 的自动识别：


1. 使用 CMake 输出的 compile\_commands.json 文件（由于 YCM 目前已经直接支持该文件，这只对较老版本的 YCM 有意义）
2. 使用上一讲（拓展 2）里用的 .clang\_complete 配置文件

目前我当然是以第 2 种方式来使用这个插件了：手写一个 .clang\_complete 很简单，非常适合临时写的小程序。但它的问题是，这个文件对整个目录有效。所以如果你在其中写了像 `-std=c++14` 这样的选项，选择某一特定 C++ 标准，那这个选项对于 C 文件就是错了，YCM 就会抱怨。不过解决起来也很容易，像上面说的情况，让 C 文件单独占一个子目录或者平行目录都可以消除此问题。

不管是哪种方法，你都需要确保配置文件在源代码的目录下或其父目录下。YCM 和 ycmconf 的搜索规则都是从源代码的所在位置往上找有没有满足文件名约定的文件，并在找到第一个时终止。

## 全局配置

YCM 有很多命令，但它默认只对少量的功能进行了键映射，其中最重要的就是 `<Tab>` 了。在使用中，我觉得自动修正和跳转功能值得单独进行一下键映射：

 复制代码

```
1 noremap <Leader>fi :YcmCompleter FixIt<CR>
2 noremap <Leader>gt :YcmCompleter GoTo<CR>
3 noremap <Leader>gd :YcmCompleter GoToDefinition<CR>
4 noremap <Leader>gh :YcmCompleter GoToDeclaration<CR>
5 noremap <Leader>gr :YcmCompleter GoToReferences<CR>
```

自动修正功能可以参考下图的演示。

```

class iterator {
public:
    typedef ptrdiff_t          difference_type;
    typedef _Tree              value_type;
    typedef _Tree*             pointer;
    typedef _Tree&             reference;
    typedef std::forward_iterator_tag iterator_category;

    iterator() {}
    explicit iterator(pointer root)
        : _M_this_level({root})
    {
        _M_current = _M_this_level.begin();
    }

    reference operator*() const
    {
        assert(!empty());
        return **_M_current;
    }
    pointer operator->() const

```

tree.h [+] [utf-8] 241,9 43%

YCM 自动修正功能的演示


自动修正的范围很广，小到修正一个拼写错误或者漏写名空间这样的问题，大到提供安全方面或代码风格现代化的调整（它可以利用你的 [Clang-Tidy](#) 配置来向你报告错误和提供修正建议）。有了它，编码工作真的轻松了许多。

我们这儿有四种跳转：

1. GoTo，无脑跳转，最常用的就是这个功能，如果能跳转到定义，就跳转到定义，否则就跳转到声明。
2. GoToDefinition，顾名思义，就是跳转到定义。
3. GoToDeclaration，专门跳转到声明。
4. GoToReferences，可以用来查出一个符号被引用的地方（libclang 引擎不支持该命令）。

注意，对 C 族语言来说，只有让 clangd 看到你的项目的 compile\_commands.json 文件，才能使用 GoToReferences 这个命令查找整个项目里符号被引用的地方；否则，你只能查出当前 Vim 里可见的引用，而非整个项目。

YCM 有很多配置参数，有些在默认状态下工作得不是很好。我通常会配置下面这些：

 复制代码

```
1 let g:ycm_auto_hover = ''
2 let g:ycm_complete_in_comments = 1
3 let g:ycm_filetype_whitelist = {
4     \ 'c': 1,
5     \ 'cpp': 1,
6     \ 'python': 1,
7     \ 'vim': 1,
8     \ 'sh': 1,
9     \ 'zsh': 1,
10    \ }
11 let g:ycm_goto_buffer_command = 'split-or-existing-window'
12 let g:ycm_key_invoke_completion = '<C-Z>'
```

第一项 `ycm_auto_hover` 用来禁用光标在一个符号上长期停留出现的自动文档提示。未禁用时的效果如下图：

```
ptr->next = &new_ptr_list;
Declared in nwa

static void print_position(const void* ptr, int line)

Prints the position information of a memory operation point. When \c
_DEBUG_NEW_USE_ADDR2LINE is defined to a non-zero value, this
function will try to convert a given caller address to file/line
information with \e addr2line.

@param ptr    source file name if \e line is non-zero; caller address
              otherwise
@param line   source line number if non-zero; indication that \e ptr
              is the caller address otherwise

    print_position(ptr->file, ptr->line);
} else {
    print_position(ptr->addr, ptr->line);
}
fprintf(new_output_fp, "%s\n");
}
total_mem_alloc += size;
```

YCM 的自动浮动提示

这个自动提示不能说一点用都没有，但它很容易成为写代码时的干扰，所以我还是把它禁用了。

第二项 `ycm_complete_in_comments` 表示我希望在写注释的时候也能启用自动完成——毕竟注释里通常也要写代码里的变量、函数名什么的。

第三项 `ycm_filetype_whitelist` 用来仅对白名单列表里的文件类型才启用 YCM。没有这一项，YCM 在打开一些特殊类型的文件时可能会报错，有时候也会导致打开的延迟。我就明确一下，让它只在我常用的源代码类型里才启用。

第四项 `ycm_goto_buffer_command` 用来告诉 YCM，当跳转的目的文件尚未打开时，用分割窗口的方式打开新文件；如果已经打开则跳转到相应的窗口。其他的可能值是 `'same-buffer'`，在同一个缓冲区的位置打开（除非这个位置因为文件修改的原因不能被替换），及 `'split'`，除非跳转目的在同一个文件，永远在新分割的窗口打开。

第五项 `ycm_key_invoke_completion` 用来定义手工启用语义完成的按键。在你输入时，YCM 会自动尝试标识符匹配，而当你输入 `..`、`->`、`::` 或这个按键时，YCM 则会启用语义完成，来给出当前上下文中允许出现的符号。这个按键默认是 `<C-Space>`，在某些操作系统上是不能用的（如 Mac 和老的 Windows），所以我改成了 `<C-Z>`。你也可以选择你自己喜欢的按键（但要注意映射冲突问题：Vim 里在插入模式下的可用键不多，事实上只有在终端下容易出问题的 `<C-S>` 和 `<C-Z>` 在 Vim 里没有默认功能）。

## 使用

说了这么多，实际上 YCM 大部分使用方法我也已经提到了。它基本上只要你使用 `<Tab>` 就能使用，你如果不理睬它的提示，它也不会对你造成什么干扰（我遇到过一些 Vim 的插件，虽然能提供些有用的提示，但是会侵入式地影响正常输入，那就只能删除 / 禁用了）。其他最重要的功能，我们也已经进行了按键映射，上面也都有了初步的描述。

还有一个可能有用的命令，不能通过按键映射，那就是重命名的重构。这个命令需要你先把光标移到要修改的符号上，然后输入的命令里要有新的名称。比如，如果你要把一个 `foo` 符号重命名成 `bar`，需要把光标移动到 `foo` 上面，然后输入：

```
1 :YcmCompleter RefactorRename bar
```

[复制代码](#)

命令虽然略长，但你一样可以用 `<Tab>` 来自动完成，所以你多打不了几个字符的。

YCM 默认只在屏幕底部显示当前行的问题，并且显示很可能被截断。要看到所有的代码问题，可以使用命令 `:YcmDiags`。

此外 YCM 还有一些调试命令，一般不需要使用，我这边就不介绍了。你可以自己在帮助文档里查看。


## RTags (选学)

因为有段时间我只能用 libclang 引擎，不能查找符号的引用，因此我安装了另外一个开源工具，[RTags](#)，来弥补这一缺憾。在使用 clangd 的情况下，RTags 已经不那么必要了。它仍然提供了一些特别的功能，并且 Linux 发布版里可能仍只提供了 libclang 引擎的 YCM，因此我就把这部分作为选学提供了，相当于本讲内部的一个小加餐。爱折腾并且使用非 Windows 环境（RTags 尚不支持 Windows）的小伙伴们可以把这部分读完，其他人就跳到内容小结吧。

## 安装

这次我就只讲自己构建安装的大概过程了。如果你的平台支持二进制安装，相信你应该可以自己搞定了。


首先我们要安装 RTags 本身。安装前，我们需要确认所在的平台有 CMake、C++ 编译器和 libclang 的开发包。这些都有了之后，我们选一个目录，执行下面的命令就可以编译安装了：

 复制代码

```
1 git clone --recursive https://github.com/Andersbakken/rtags.git
2 cd rtags
3 mkdir build
4 cd build
5 cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=1 -DCMAKE_BUILD_TYPE=Release ..
6 make -j4
7 sudo make install
```

在这些命令都正常执行之后，你就已经把 RTags 的命令安装到了 `/usr/local` 下面。

随后我们安装 RTags 和 Vim 集成的插件。这个比较简单，用你的包管理器安装 `lyuts/vim-rtags` 即可。在配置文件里，我通常只做一处调整：

 复制代码

```
1 let g:rtagsUseLocationList = 0
```

这是因为 vim-rtags 默认使用位置列表 (location list) 而不是快速修复窗口。我们之前没有介绍过位置列表，我就快速引用一下文档：

位置列表是一个窗口局部的快速修复列表。由


:lvimgrep、:lgrep、:lhelpgrep、:lmake 等命令产生，它们生成位置列表而不是对应的 :vimgrep、:grep、:helpgrep、:make 生成的快速修复列表。

位置列表和窗口相关联，而每个窗口都要单独的位置列表。一个位置列表只能和一个窗口相关联。位置列表和快速修复列表相互独立。

我通常不怎么使用位置列表，主要是为了简单，可以使用固定的快捷键。

## 运行和项目配置


RTags 是一个客户端 / 服务器端架构的程序。但它不是用 TCP/IP，而是 Unix 域套接字，一个用户只能运行一个服务器，可以支持多个客户端。启动服务器端的命令非常简单，就是：

 复制代码

```
1 rdm &
```

我这儿用了 &，让 rdm 在后台运行，但输出仍能直接从终端上看到。在我们刚开始学习使用 RTags 时，我们仍需多多监控 rdm 的输出。

在某个项目中启用 RTags，最简单的方式就是使用 CMake 输出的 compile\_commands.json 文件。我们只需要在这个目录下执行以下命令：

 复制代码

```
1 rc -J .
```

随后我们就能看到运行 rdm 的那个窗口屏幕哗哗地翻滚，忙着“编译”代码。等到 rdm 忙完了，项目索引就算完成了。而当你修改代码时，rdm 会看到你修改，然后就会自动编译相关的文件，保持索引为最新。

有没有注意到我上面编译 RTags 的命令已经生成了 compile\_commands.json 文件？所以，如果你没有现成的其他 CMake 项目，你可以用这个项目本身来进行搜索。

## 使用

如果我们使用默认的 vim-rtags 键映射的话，我们只需要把光标移到一个符号上面，然后输入 \rf（理解为 find references）即可。下面是一个示例：

```
#define SHAPE_H

#include <stdexcept> // std::logic_error
#include <stdio.h>    // puts

enum class shape_type {
    circle,
    triangle,
    rectangle,
};

class Shape {
public:
    virtual ~shape() {}
};

class circle : public shape {
public:
    circle() { puts("circle()"); }
    ~circle() { puts("~circle()"); }
};

class triangle : public shape {
```

13,7

3%

Vim-rtags 的使用示例

这个结果出来的速度比用 :grep 可快多了！

它还有很多其他命令，可以用来查找定义、**查找父类**、**查找子类**、**显示调用树**，等等。有些功能在 YCM 里并没有对应物，这也是 RTags 的价值了。你可以在 [vim-rtags 的主页](#) 上查看学习所有这些命令及其快捷键。

## 内容小结



在本讲中，我们主要介绍了 YouCompleteMe 这个重量级插件，包括其安装和配置。我们可以看到，在插件的帮助下，我们可以获得不输于集成开发环境的自动完成体验，同时，仍然享受 Vim 的快速启动和强大编辑功能。最后我们花了一点点时间介绍了 RTags 工具和 vim-rtags 插件，它在你写 C 族语言而不能使用 clangd 引擎时会特别有帮助。

YCM 的参数和键映射我写到了示例配置文件里，对应的标签是 l13-unix 和 l13-windows。

## 课后练习

今天学完之后的主要任务，当然就是把 YouCompleteMe 装起来、配置好了。它的使用反而是相当简单的，大部分情况下使用 <Tab> 就行。

至于 RTags，Unix 下的 C++ 程序员们可以根据自己的兴趣，决定是否捣腾一下。这个工具还是有点好玩性的。

如果遇到什么问题，欢迎留言和我讨论。我们下一讲再见！

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 12 | 语法加亮和配色方案：颜即正义

下一篇 14 | Vim 脚本简介：开始你的深度定制

## 精选留言 (12)

写留言



leaf

2020-11-07

请问老师，我想给ycm的GoToSymbol命令也定义一个快捷键，但发现这个命令不是默认



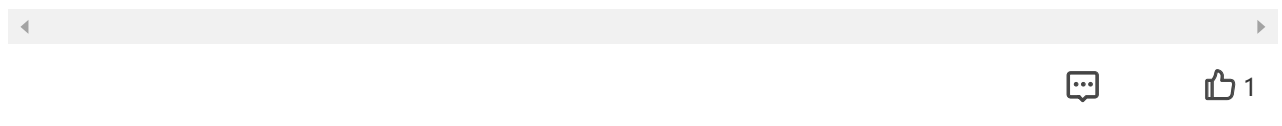
取当前光标下的符号，而是要自己输入带查找的symbol，这样的快捷键应该如何定义？

作者回复: 这个命令本来就是设计用来输入符号的，如果是取光标下的符号，用 GoToDefinition 子命令不挺好？

如果是不想手工输入前面这串，可以类似这样定义：

```
:nnoremap <Leader>gs :YcmCompleter GoToSymbol
```

这样，敲入 \gs 即可自动输入命令的前面部分。



**doge**

2020-09-03

说一下用后感吧，跳转的速度和精度上比vscode和clion要迅速很多，就是快捷键比较多，得多用熟能生巧才行。

另一个就是得ctags+cscope+YCM+rtags一起用才能得到最好的跳转体验。

基本就是C-] \gt \rj \rT轮番上阵，哈哈！

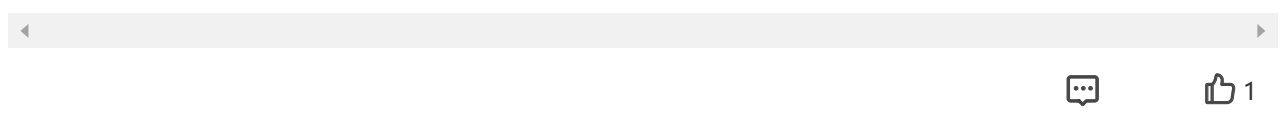
整体感觉还是非常良好的，不过如果CLion没那么卡和吃内存就好了，CLion的类继承关...  
展开 ✓

作者回复: 无论如何，cscope没有使用的必要了吧，如果你用C++的话。如果用上编译数据库的话，YCM基本已经满足需求了。ctags和rtags勉强有部分补缺的作用，cscope我看不出来有啥用处。

类继承你是说查看有什么子类吗？这个在rtags里有，但如果要跟商业软件比界面的漂亮，那肯定是没有的。

我试了你说的这个插件，还有点意思。但是，跟其他类似插件一样，单纯依赖名字的加亮，在C++里玩不转——把我的很普通的成员变量next都当成标准库里的函数名称来加亮了。

朴素点，不是为了朴素，而只是不愿看到错误行为时的无奈之举。如果有插件能进行完全正确的加亮，又不会拖慢编辑的速度，我干吗不用呢？;-)



**supakito**

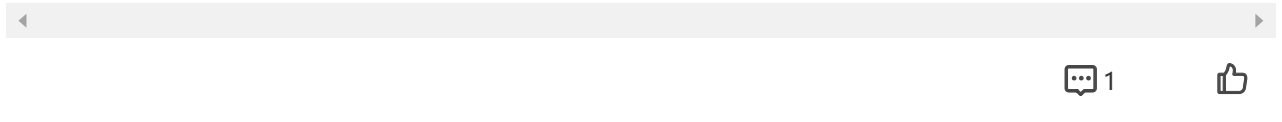
2020-11-08

老师，感觉最近tabnine好像很火的样子，不知道和ycm相比，哪个更好用一些？

作者回复: 没有比较。这个补全的方法不同, 比较耗算力, 而且对大项目是需要付费的, 我没有尝试。

对于这类工具我持审慎怀疑: 如果它真能预测得很好, 那是不是说明代码中的重复有点多了?

不过, 那只是个人没有使用过的瞎猜测。实践是检验真理的标准。你自己试试不就得了? 适合自己的就是好的, 适合你的不一定适合其他人, 反过来也一样。



浩浩

2020-10-25

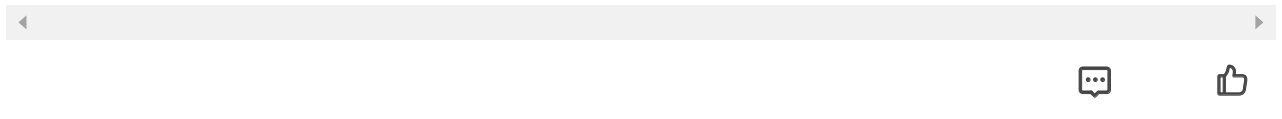
brew list. 安装了一下软件

=====

```
autoconf  cmake    fzf    gettext    icu4c    lua    node    pkg-confi
g  python@3.9  ruby    universal-ctags
automake  cscope    gdbm    go    libyaml    macvim    openssl@1....
```

展开 ∨

作者回复: 可能就是下载/网络出了问题。我目前试下来是可以成功的, clangd的URL跟你的完全相同。



gigglesun

2020-10-18

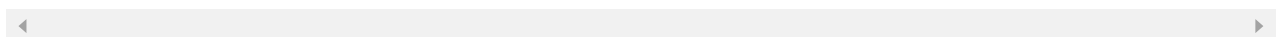
YouCompleteMe可以通过离线的方式安装吗? 公司的服务器不能连接外网, 我可以在自己的电脑通过

```
git clone --recurse-submodules \
    --shallow-submodules \...
```

展开 ∨

作者回复: 在同样软件配置的机器上, 使用git命令和安装命令, 完全结束了(建议测试一下)再拷贝结果。

安装过程中也会下载东西的, 所以不能在目标机上执行安装命令。



**瀚海星尘**

2020-10-11

老师我这里 ubuntu 使用 `apt install vim-youcompleteme` 后，提示找不到引擎，估计是还需要单独安装，最后还是用了编译安装的方式。我也安装了 `vim-autopairs`，确实带来了许多困扰，最头痛的就是需要只输入一对的第一个符号的情况，每次输入都会自动输出两个，然后删除第二个又会自动把第一个给删除...看了老师的留言，果断把它删除了。💎💎💎

展开 ∨

作者回复: 如果报的错是跟clang相关的话，可能是vim-youcompleteme没有正确地依赖到libclang1上。我在Ubuntu 20.04 LTS上碰到过，运行`sudo apt install libclang1`可以解决该问题。



💬 2

**doge**

2020-08-31

ubuntu环境按照老师的命令安装rtags失败，报错  
`rtags/src/rct/rct/Apply.h:46:10: error: unknown type name 'size_t'; did you mean 'std::size_t'?`  
最后直接下载release 2.3.8的tarball编译成功。。

作者回复: 如果是编译不过，那你可能得向作者报告 bug 了。



💬

**pyhhou**

2020-08-31

安老师给的步骤安装了 YCM，试着编辑了 javascript 文件，感觉有几个地方不是太清楚：

1. YCM 中的跳转(GoTo)貌似只能在单个文件中跳转，不能跳转到其它的文件中去？有些时候，如果需要跳转到函数，只能跳转到前面的定义和声明，无法跳转到后面的定义和...

展开 ∨

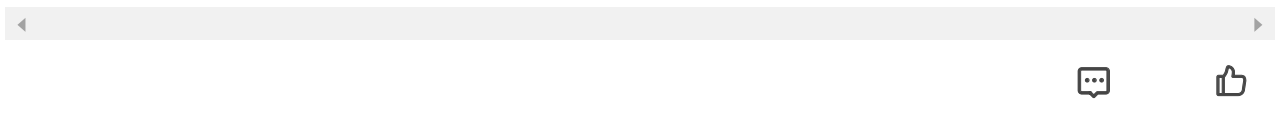
作者回复: 我对 YCM 的 JavaScript 支持没有细节了解，只能凭经验随便说两句了。

1. 在 C++ 里，如果没有编译命令文件的支持，也只能支持到差不多程度，但从来没见过跳不到“后面”的情况，毕竟，工具应该是把整个文件的符号分析出来的。

2. Vim 的 <C-O> 键是跳回，这个跟插件无关，哪儿都可以用。

应该按照 YCM 的要求，安装时使用 --ts-completer 选项就行。如果效果不好，那是能力问题。像 C++ 用 rtags 属于老版本引擎能力不足，rtags 本身和 YCM 没有任何集成关系。用了 clang d 引擎就基本上不需要 rtags 了。

对于 JavaScript，也许你可以严肃考虑一下 coc.nvim 这个选项。完成引擎本身的开发方式也决定了它的重点，YCM 明显是偏向 C++ 的。而 coc.nvim 本身就依赖于 node.js，在 JavaScript 上如果强于 YCM，我完全不会意外，虽然我没有用过。



**YouCompleteMe**

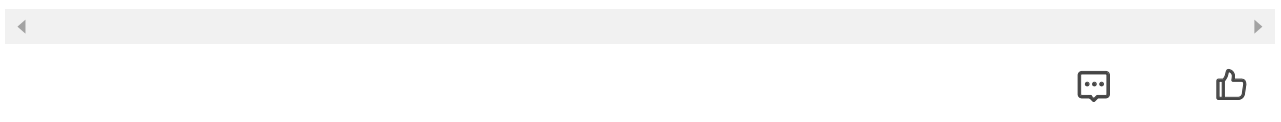
2020-08-26

老师，我有两个疑问：

1. 截图里是否使用了 statusline 的插件，我的用了老师的配置，没有显示 [+] / utf-8 / 当前行百分比。
2. 老师平时写代码会使用括号自动补全吗 ~

作者回复: 1. 不是插件，就是手工设置了 statusline。没讲这个是因为大部分人应该会去使用 Airline (后面会讲)，大概不会手工设了。现在手机上看不了当前配置，应该就是 [http://wyw.dcweb.cn/vim/\\_vimrc.html](http://wyw.dcweb.cn/vim/_vimrc.html) 的样子加上 fugitive 提供的 Git 信息。

2. 括号补全是个看似很小、实际很复杂的功能。你希望它很自动，同时又希望可以盲打，还能不影响删除之类的操作，都做好不容易。前不久我看到一个看似不错的补全插件，都已经写了一段推荐文字了，结果在编辑某段 Vim 脚本时发现会导致我根本无法删除我想要删除的部分。那就卸载了。不好用的功能不如不用。



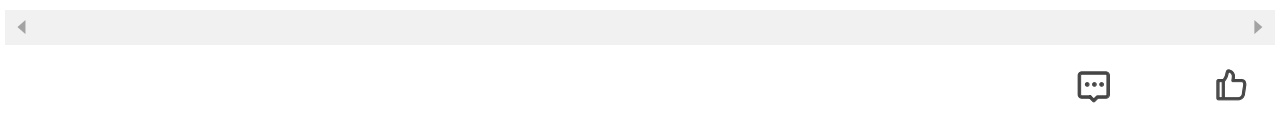
**know-one**

2020-08-26

请问python有没有哪个插件能显示继承关系？

展开 ∨

作者回复: 这个我不知道 (Python写对象继承似乎都很少)。看看有没有其他同学知道吧。



**return**

2020-08-26

ycm确实难装，以前总想着通过插件方式来装，今天才知道 其中的原理。

**我来也**

2020-08-26

有些功能确实很吸引人。

比如自动补全时自动用正则过滤候选词。

曾经我也折腾过ycm，但由于有些东西不会用，自己没调通，最终也放弃了。  
要是能早些看到这篇文章，应该会少走不少弯路。...

展开 ∨

作者回复: 又最快。😊

如果重命名常用的话，也可以自己定义个短命令，如：

```
:command -nargs=1 RR YcmCompleter RefactorRename <args>
```

