**[Vim as Python IDE on windows](http://www.cnblogs.com/renrenqq/archive/2010/09/09/1813669.html)**

下载安装Python。

从[Vim的网站](http://www.vim.org/download.php#pc)下载vim,建议下**Self-installing executable**的版本。

编辑vim的配置文件\_vimrc

拷贝vimrc\_example里面的东东到\_vimrc

去掉\_vimrc里面的source $VIMRUNTIME/vimrc\_example.vim

设置中文支持

" 设置编码自动识别, 中文引号显示    
"set fileencodings=utf-8,cp936,big5,euc-jp,euc-kr,latin1,ucs-bom    
set fileencodings=utf-8,gbk    
set ambiwidth=double

设置ColorScheme，我选择的是desert，你可以在GVim的Edit-->Color Scheme看到其他的Color Scheme。

"ColorScheme  
colorscheme desert

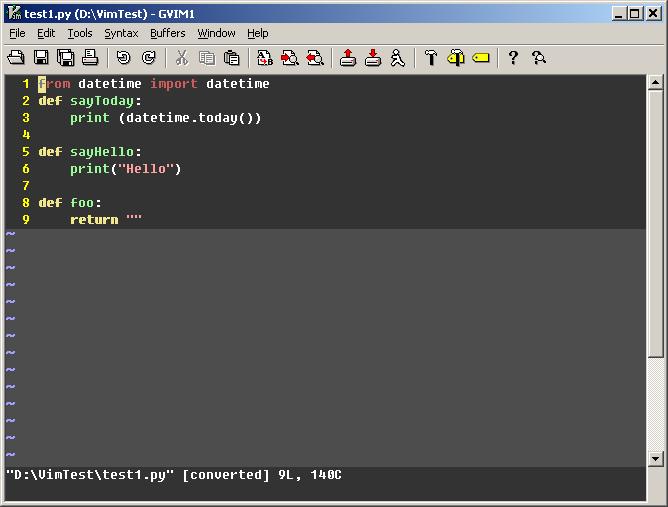
设置tab插入空格，缩进，删除键删除缩进，页面文字宽度

" 允许退格键删除和tab操作    
set smartindent    
set smarttab    
set expandtab    
set tabstop=4   
set softtabstop=4   
set shiftwidth=4   
set backspace=2  
set textwidth=79

启用鼠标，启用行号

" 启用鼠标    
set mouse=a    
  
" 启用行号    
set nu

完成以上步骤后用GVim打开Python代码的样子。

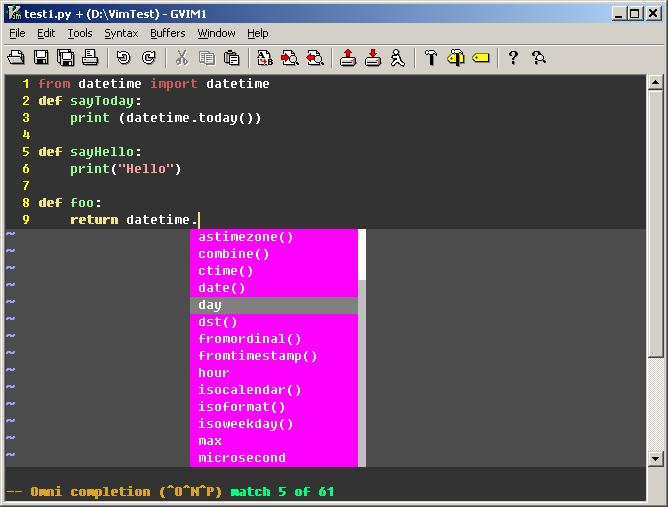


启用omni-completion的代码补全功能 ,这个系列的组件是随着Vim一起安装好的，我顺便把Ruby等其他语言的支持也加上了。现在按Ctrl-X,Ctrl-O可以补全代码

如下修改\_vimrc

"自动补全    
filetype plugin indent on  
set completeopt=longest,menu  
"自动补全命令时候使用菜单式匹配列表    
set wildmenu  
autocmd FileType ruby,eruby set omnifunc=rubycomplete#Complete  
autocmd FileType python set omnifunc=pythoncomplete#Complete  
autocmd FileType javascript set omnifunc=javascriptcomplete#CompleteJS  
autocmd FileType html set omnifunc=htmlcomplete#CompleteTags  
autocmd FileType css set omnifunc=csscomplete#CompleteCSS  
autocmd FileType xml set omnifunc=xmlcomplete#CompleteTags  
autocmd FileType java set omnifunc=javacomplete#Complet

omni-completion实现代码补全的效果

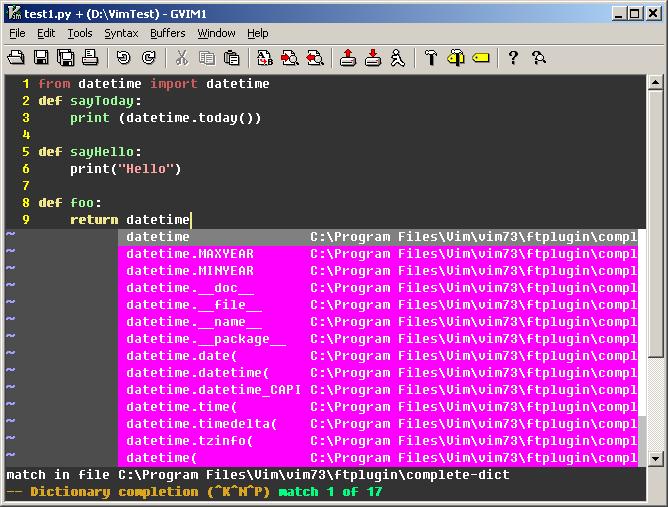


omni-completion可以实现部分的代码补全功能，但是不是很全，所以我们启用[pydiction](http://www.vim.org/scripts/script.php?script_id=850)，更丰富的的代码补全功能，按tab补全代码。

下载[pydiction](http://www.vim.org/scripts/script.php?script_id=850)，解压后有4个文件，拷贝python\_pydiction.vim和complete-dict到ftplugin目录，修改\_vimrc

"Pydiction  
let g:pydiction\_location = 'C:\Program Files\Vim\vim73\ftplugin\complete-dict

现在按Tab键可以实现代码补全，如图



下面我们需要分析源代码的工具[Ctags](http://ctags.sourceforge.net/)，从网站下载For Windows版本的[Ctags](http://ctags.sourceforge.net/)，解压缩后拷贝ctags.exe到某处，确保windows的path包含这个目录。

打开源代码文件，点击GVim上的buildtags按钮。或者在控制台中到你放源码的目录手动执行ctags -R命令。我们可以发现在当前目录下生成了tags文件。这个就是我们需要的。我们可以在方法上按Crtl+],Vim会帮你找到方法定义的地方。

下面我们需要使用这个文件查看源代码。我们需要用到另外一个插件[TagList](http://www.vim.org/scripts/script.php?script_id=273)，配合Ctags可以高效的查看源代码。下载[TagList](http://www.vim.org/scripts/script.php?script_id=273)解压缩后是doc和plugin和两个目录，把这两个目录拷贝到你的Vim安装目录（我的路径是C:\Program Files\Vim\vim73），覆盖原目录（不用担心这样覆盖不会影响已存在的东西），修改\_vimrc如下

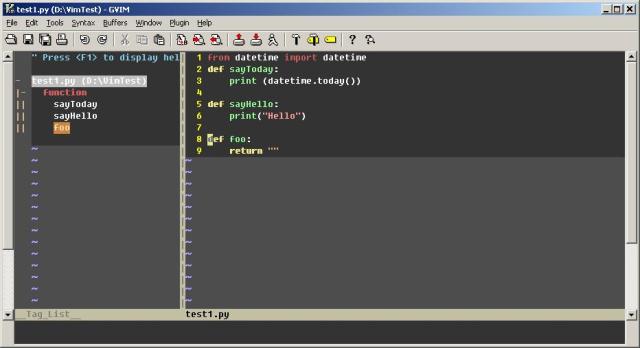
"TagList  
let Tlist\_Show\_One\_File=1  
let Tlist\_Exit\_OnlyWindow=1

我们需要制定tags目录给Vim,在Vim中执行命令:set tags=你的代码所在的路径\tags，如果你一直使用同一个代码路径可以把他写死在\_vimrc中，例如我的代码在

D:\VimTest配置如下

"tags  
set tags=D:\VimTest\tags

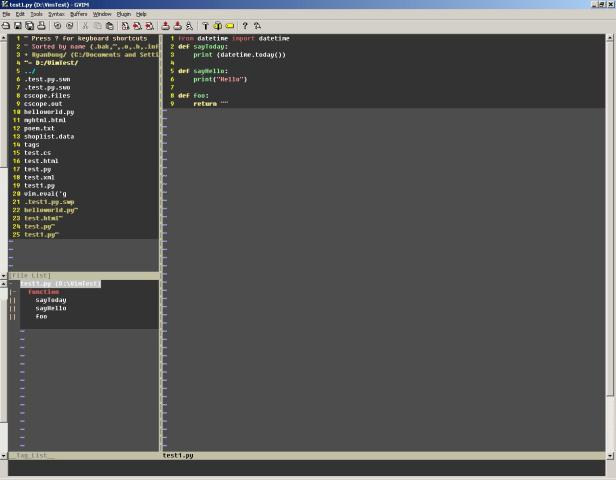
然后在Vim中执行:Tlist就可以打开TagList的窗口了，如下图左边的窗口，试着在方法上点击，可以发现右侧的光标会自动跳到你点中的方法。



下面我们添加的是文件浏览的功能，下载[WinManager](http://www.vim.org/scripts/script.php?script_id=95)，加压缩后有三个文件，放到plugin目录里。配置如下可以同时显示FileExplorer和TagList

"WinManager  
let g:winManagerWindowLayout='FileExplorer|TagList'  
nmap wm :WMToggle<cr>

下面我们输入wm命令就可以看到了，左上是FileExplorer，左下是TagList



启用[Cscope](http://cscope.sourceforge.net/)，更高级的分析源代码，支持反向查找

下载windows下的可用版本<http://sourceforge.net/projects/mslk/>，解压缩后拷贝所有文件到windows的path指向到的某个目录。

下载[cscope\_maps.vim](http://cscope.sourceforge.net/cscope_maps.vim)，放到Vim的plugin目录。

修改\_vimrc，让查找结果显示在quickfix窗口而不是主窗口。

"cscope show in quickfix  
set cscopequickfix=s-,c-,d-,i-,t-,e-

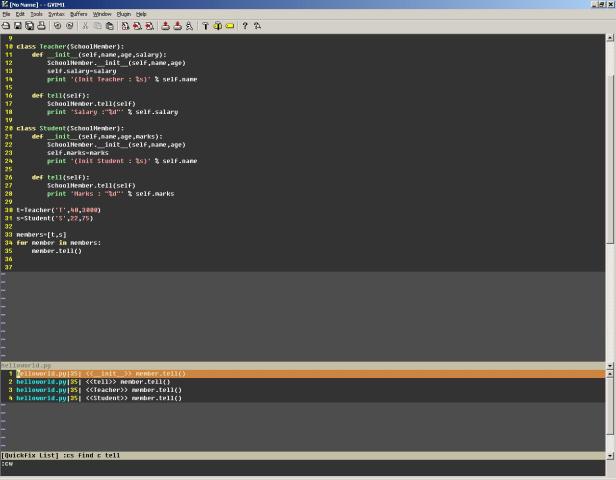
好了，配置完成，下面我们看看怎么用。在vim中输入:help if\_cscop.txt可以看到关于[Cscope](http://cscope.sourceforge.net/)的帮助。

看完文档，我们可以在vim中输入命令:cs find c foo用来查找foo方法在哪里被调用过。

[cscope\_maps.vim](http://cscope.sourceforge.net/cscope_maps.vim)映射了一些键盘的快捷键，我们可以快捷的使用查找功能。

我比较喜欢使用Ctrl+\,让光标停留在你想查找的内容，然后输入Ctrl+\后迅速的输入c,就可以在quickfix窗体看到查找的结果。如果结果有多条你可以输入命令:cw展开quickfix窗体。(这个快捷操作第一次使用可能会有些别扭，多试几遍就好了)

下面是我的一个查找结果



启用[MiniBufExplorer](http://www.vim.org/scripts/script.php?script_id=159)，管理vim的buffer(缓冲区)，快速的在多个同时编辑的文件间切换。

编辑\_vimrc如下

"MiniBufExplorer  
let g:miniBufExplMapWindowNavVim = 1   
let g:miniBufExplMapWindowNavArrows = 1   
let g:miniBufExplMapCTabSwitchBufs = 1   
let g:miniBufExplModSelTarget = 1

配置好后打开多个文件[MiniBufExplorer](ttp://www.vim.org/scripts/script.php?script_id=159)窗口就出现了。点击上面的文件名可以在不同的文件间快速切换。



启用Grep，在工程中查找

Grep是linux中命令，如果你不熟悉可以看[Grep的文档](http://www.gnu.org/software/grep/manual/)。

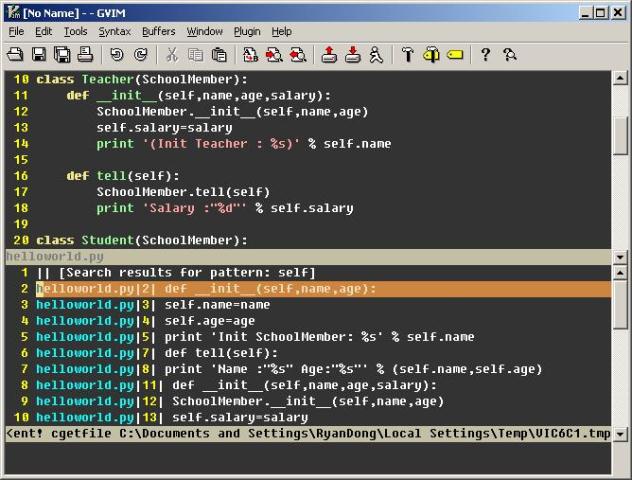
下载安装[Grep for windows](http://gnuwin32.sourceforge.net/packages/grep.htm)

下载Vim的Grep插件，[grep.vim](http://www.vim.org/scripts/script.php?script_id=311)放在plugin目录

配置\_vimrc,设置F3为快捷键

"Grep  
nnoremap <silent> <F3> :Grep<CR>

把光标放在你需要搜索的内容，按下F3,可以看到Search for pattern: xxx,按下回车，出现Search in files,输入\*.py，只查找python的源代码文件，按下回车就可以看到搜索的结果了。见下图

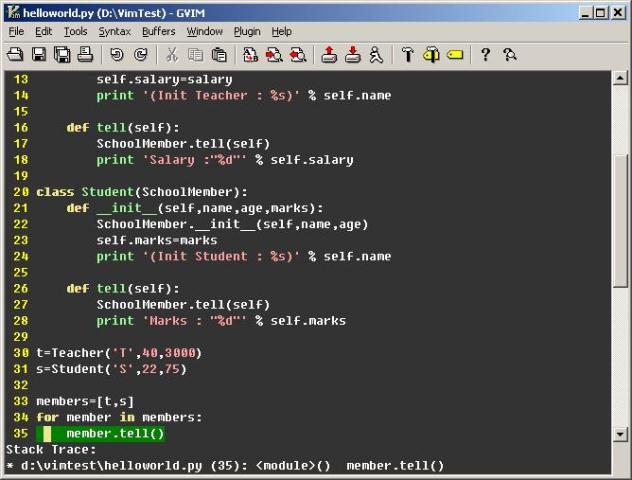


启用[VimPdb](http://code.google.com/p/vimpdb/)，可以调试Python程序。

下载[VimPdb](http://code.google.com/p/vimpdb/)，解压后拷贝VimPdb.py，VimPdb.vim到plugin即可。

用vim代开python代码文件，按F5运行，然后按F2设置断点，在运行到断点后可以用按F12可以查看Stack Trace，F3查看变量和参数值。

更高级的使用可以查看文档。

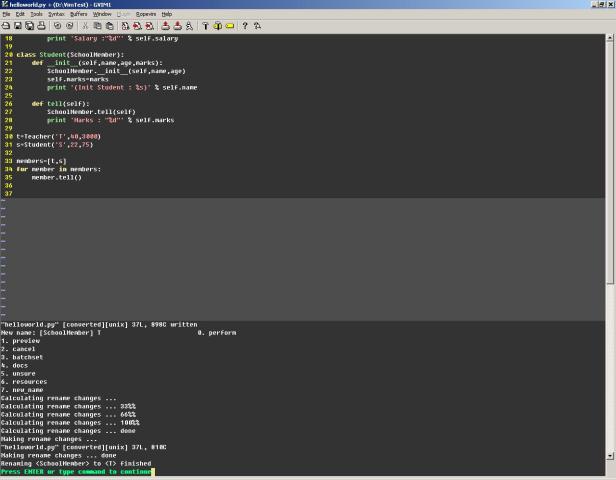


启用ropevim ，重构Python程序

下载r[ope](http://pypi.python.org/pypi/rope),[ropemode](http://pypi.python.org/pypi/ropemode)，[ropevim](http://pypi.python.org/pypi/ropevim)，分别解压缩，打开控制台，cd到解压缩目录，执行python setup.py install安装。

然后拷贝ropevim.vim文件到plugin目录。这样就安装好了。我们试试重构一个类名，光标放在类名上，按Ctrl-c 然后连按两下r。

可以看到quickfix窗体要求你输入New name:，输入新的名称回车，再回车，重构就完成了。如下图

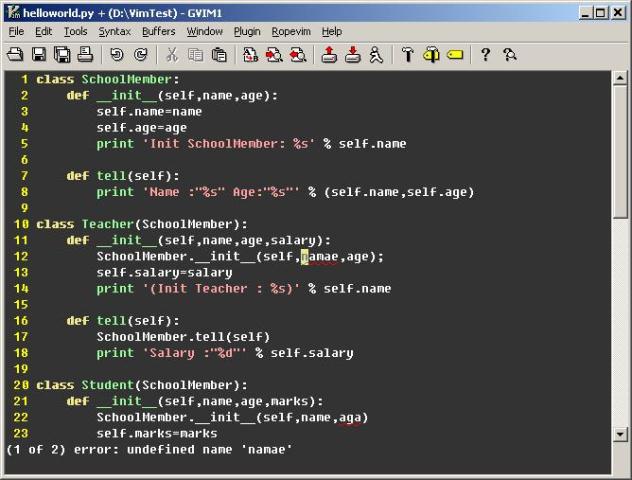


Python代码检查

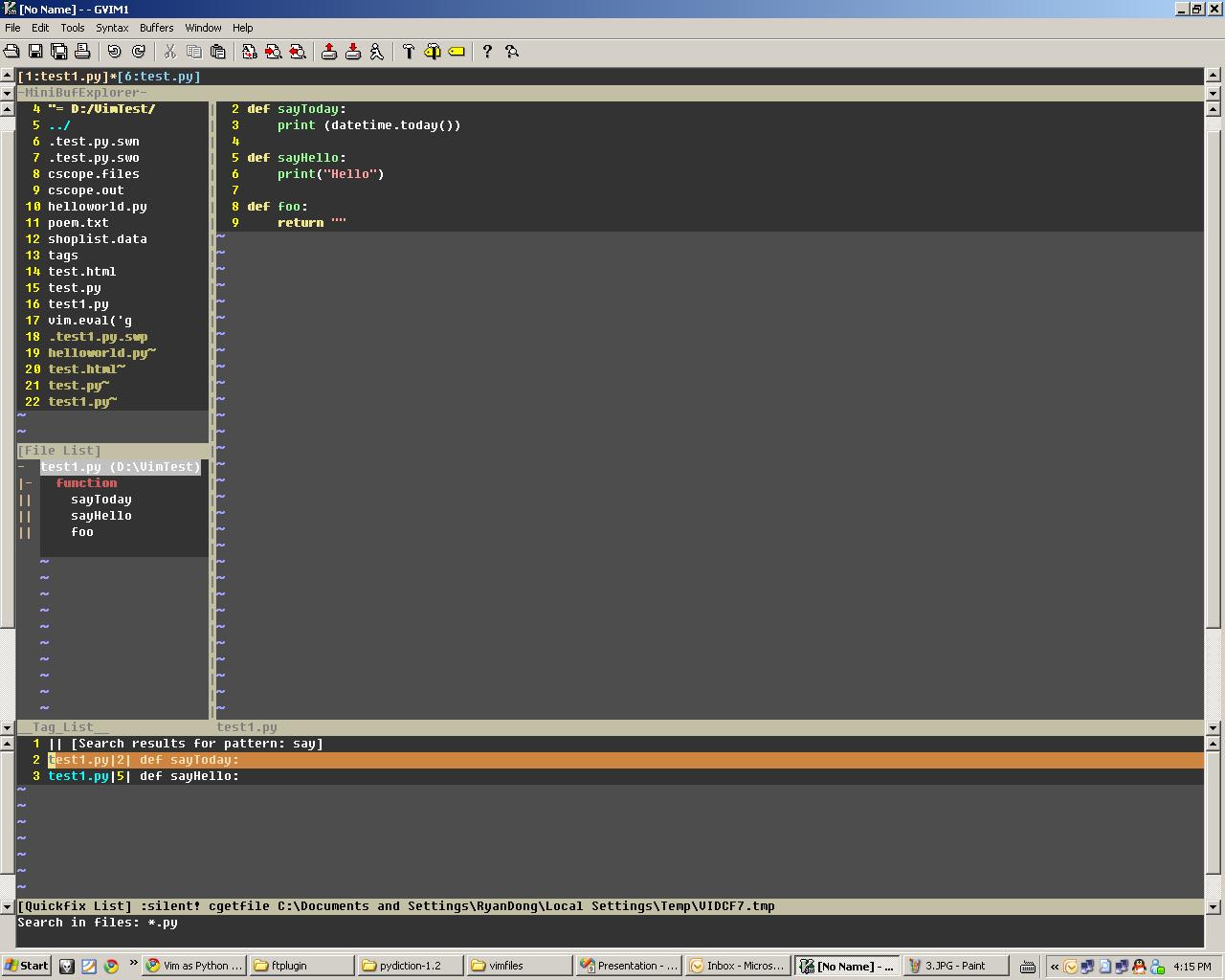
在Vim的ftplugin目录下新建一个python目录。

下载[pyflakes](http://www.vim.org/scripts/script.php?script_id=2441)，解压缩后把pyflakes.vim文件和pyflakes目录拷贝套ftplugin\python目录中。

打开一个有问题的python源代码文件，执行命令:cc，即可进行代码检查，如下图中找到了两处错误。



好了终于写完了，真累，最后放个大图，显示下整体效果。



参考文档:

[手把手教你把Vim改装成一个IDE编程环境(图文)](http://blog.csdn.net/wooin/archive/2007/10/31/1858917.aspx)

来源： <<http://www.cnblogs.com/renrenqq/archive/2010/09/09/1813669.html>>