# UltiSnips

## 1 UltiSnips简介

UltiSnips是用于在vim中输入一个关键字就可以自动补全代码片的神奇。比如：输入main，触发UltiSnips后后就会自动输入一下内容：

int main(int argc,char \*argv[])

{

光标停留在在此处

return 0

}

### 1.1 使用要求

该插件适用于Vim版本7.4或更高版本。只有在'compatible'设置未设置的情况下才有效。

这个插件是针对Python 2.7,3.3或3.4进行测试的。所有其他版本都不受支持，但可能会起作用。

Python 2.x或Python 3.x界面一定可用。换句话说，Vim必须支持python或python3。以下命令显示如何测试是否在Vim中支持python。如果python版本被编译，他们打印'1'，如果没有，则打印'0'。

测试Vim是否使用python版本2.x编译：>

    ：echo has（“python”）

链接的python版本Vim可以找到：>

    ：py import sys; print（sys.version）

测试Vim是否使用python版本3.x编译：>

    ：echo has（“python3”）

链接的python版本Vim可以找到：>

    ：py3 import sys; print（sys.version）

请注意，Vim可能不是使用您的系统范围内安装的python版本，因此请确保检查Vim内的Python版本。

UltiSnips尝试自动检测哪些python版本被编译到Vim中。不幸的是，在某些版本的Vim中，这种检测功能不起作用。在这种情况下，您必须使用“UltiSnipsUsePythonVersion”全局变量明确告知UltiSnips要使用的版本。

使用python版本2.x：>

 let g：UltiSnipsUsePythonVersion = 2

使用python版本3.x：>

 let g：UltiSnipsUsePythonVersion = 3

### 1.2 致谢

UltiSnips的灵感来自于TextMate（http://macromates.com/）的片段功能，这是Mac OS X的GUI文本编辑器。管理Vim中的片段不是新的。 我想感谢snipMate的作者Michael Sanders，从他的插件借来的一些实现细节以及使用他的片段的许可。

## 2 安装和更新

获得UltiSnips的推荐方法是在github上跟踪SirVer / ultisnips。主分支总是稳定的。

### 2.1 使用Pathogen：

如果您是Pathogen用户，您可以在github上跟踪UltiSnips的官方镜像：

$ cd〜/ .vim / bundle && git clone git：//github.com/SirVer/ultisnips.git

如果您还想要默认的snippet，还可以跟踪

$ cd〜/ .vim / bundle && git clone git：//github.com/honza/vim-snippets.git

有关如何更新软件包的更多详细信息，请参阅Pathogen文档。

### 2.2 使用下载的数据包：

下载数据包并将其解压缩到您选择的目录中。然后将此目录添加到您的Vim运行时路径，将此行添加到您的vimrc文件。

设置runtimepath + =〜/ .vim / ultisnips\_rep

UltiSnips还需要来自ftdetect /目录的Vim源文件。不幸的是，Vim只允许在.vim目录中的这个目录。因此，您必须符号链接/复制文件：

mkdir -p〜/ .vim / ftdetect /

ln -s〜/ .vim / ultisnips\_rep / ftdetect / \*〜/ .vim / ftdetect /

重新启动Vim和UltiSnips。要访问帮助，请使用

helptags〜/ .vim / ultisnips\_rep / doc

帮助UltiSnips

UltiSnips没有snippet。默认的片段可以在这里找到：https：//github.com/honza/vim-snippets

## 3 设置和命令

### 3.1 命令

UltiSnipsEdit命令打开当前文件类型的专用代码段定义文件。如果不存在片段文件，则会创建一个新文件。如果用作UltiSnipsEdit！所有公用的snippet文件都将考虑在内。如果多个文件匹配搜索，用户将选择该文件。

有几个变量与UltiSnipsEdit命令相关联：

g:UltiSnipsEditSplit 定义打开编辑窗口的方式. 可能值为：

|normal| 默认。在当前窗口中打开。

|horizontal| 水平分割窗

|vertical| 垂直拆分窗户。

|context| 依赖于上下文，垂直或水平拆分窗口

g:UltiSnipsSnippetsDir 定义存储专用代码段文件的目录。例如，如果变量设置为“〜/ .vim / mydir / UltiSnips”，当前的“filetype”为“cpp”， 如果文件不为空则：UltiSnipsEdit将打开“〜/. vim / mydir / UltiSnips / cpp.snippets“，如果为空：UltiSnipsEdit将在g：UltiSnipsSnippetDirectories查找文件，如果没有找到，：UltiSnipsEdit将在g：UltiSnipsSnippetsDir打开新的文件。注意，名为“snippet”的目录是为snipMate片段保留的，不能使用。

g:UltiSnipsSnippetDirectories 定义查找代码段的目录。不要混淆这个变量与上面的变量。有关该变量的更多信息，请参见| UltiSnips-snippet-search-path |部分。

g:UltiSnipsEnableSnipMate 允许在＆runtimepath中查找SnipMate代码片段。 UltiSnips将只在名为“snippet”的目录搜索SnipMate代码片段。默认为“1”，所以UltiSnips会查找SnipMate代码片段。

UltiSnipsAddFiletypes命令允许显式合并当前缓冲区的其他片段文件类型。 例如，如果您编辑.rst文件，但也希望Lua片段可用，您可以发出命令

：UltiSnipsAddFiletypes rst.lua

使用点分语法。此列表中的第一个文件类型将是UltiSnipsEdit中使用的文件类型，因此顺序很重要，列表优先级排序。您可以将其添加到您的ftplugin / rails.vim>中

：UltiSnipsAddFiletypes rails.ruby

改命令表示：首先UltiSnips首先使用rails代码片段，而rails片段应该覆盖相同触发器的ruby片段。 所有的优先级都是rails - > ruby - > all。 如果您有一些特殊的编程代码片段，应该优先级低于您可以调用的ruby代码片段

：UltiSnipsAddFiletypes ruby.programming

那么优先级就是rails - > ruby - > programming - > all。

### 3.2 触发器

您可以通过设置全局变量来定义用于触发UltiSnips操作的键。用于展开代码段，在代码段中向前跳转并、向后跳转，列出当前展开上下文中的所有可用代码段。请注意，某些终端仿真器不会将<c-tab>发送到正在运行的程序。具有默认值的变量为：

g:UltiSnipsExpandTrigger="<TAB>" "展开代码片段

g:UltiSnipsJumpForwardTrigger="<c-j>" "后一个

g:UltiSnipsJumpBackwardTrigger="<c-k>" "前一个

g:UltiSnipsListSnippets="<c-tab>" "打开代码片段列表

UltiSnips只会在代码段处于活动状态时映射跳转触发器，以尽可能少地与其他映射进行干扰。

g：UltiSnipsJumpBackwardTrigger的默认值会干扰内置的补全功能。解决方法是将以下内容添加到您的vimrc重新映射键。

### 3.3 snippets搜索路径

UltiSnips的snippet定义文件存储在一个或多个目录中。有几个变量用于指示这些目录，并定义UltiSnips如何加载代码段。

snippet定义文件存储在snippet目录中。snippet目录必须是“runtimepath”选项中定义的目录的子目录。变量g：UltiSnipsSnippetDirectories定义了snippet目录的名称列表。请注意，“snippets”是为snipMate片段保留的，不能使用。默认值如下所示。

let g:UltiSnipsSnippetDirectories=["UltiSnips"]

UltiSnips将按照定义的顺序查找每个“runtimepath”目录中由g：UltiSnipsSnippetDirectories定义的子目录。例如，如果您将snippet保存在名为“mycoolsnippets”的.vim子目录中，并且要使用UltiSnips附带的默认代码段，请将以下内容添加到您的vimrc文件中。

let g:UltiSnipsSnippetDirectories=["UltiSnips", "mycoolsnippets"]

如果您不想使用插件附带的第三方代码段，请定义变量：

let g：UltiSnipsSnippetDirectories = [“mycoolsnippets”]

您还可以通过设置变量b：UltiSnipsSnippetDirectories，来重新定义缓冲区上的搜索路径。 该变量优先于全局变量。

参见| UltiSnips-added-snippets |，其说明snippets与文件类型的对应关系。

如果在此变量中仅指定一个目录，并且以绝对路径指定此目录，则UltiSnips将不会在＆runtimepath中查找代码段，这能显著的加速。 所以，通常设置是：

let g：UltiSnipsSnippetDirectories = $HOME.'/.vim / UltiSnips'

但是，您将无法使用开箱即用的第三方插件附带的片段。 您需要将它们复制到您选择的目录中。但是，您将无法使用该目录外的第三方插件附带的片段。 您需要将它们复制到您选择的目录中。

### **3.4 关于选择模式映射的警告**

Vim| mapmode-s |帮助文档写到：

注意：在选择模式下映射可打印字符可能会混淆用户。最好使用：xmap和：smap，或者在映射定义后使用：sunmap。

但是，大多数Vim插件，包括一些默认的Vim插件，都不遵守这一点。UltiSnips使用“select”模式来对片段中的tabstop变量进行替代覆盖。 现有的Visual +选择模式映射将会受到干扰，因此，UltiSnips发出一个|：sunmap | 命令来删除可打印字符在选择模式的映射。不会修改其他映射。 特别地，UltiSnips不会更改现有的normal，insert或visual模式的映射。

如果不需要此行为，您可以通过将下面的代码添加到您的vimrc文件来禁用它。

let g:UltiSnipsRemoveSelectModeMappings = 0

如果要仅对特定映射禁用此功能，请将其添加到要忽略的映射列表中。 例如，您的vimrc文件中有以下内容时，除了在其完整定义中包含字符串“somePlugin”或字符串“otherPlugin”的映射，如|:smap| command， 将取消所有选择模式的映射>

let g：UltiSnipsRemoveSelectModeMappings = 1

let g：UltiSnipsMappingsToIgnore = [“somePlugin”，“otherPlugin”]

### 3.5 函数

UltiSnips提供了一些扩展核心功能的函数。

#### 3.5.1 UltiSnips＃AddSnippetWithPriority

第一个函数是UltiSnips＃AddSnippetWithPriority(trigger, value, description,options, filetyp, priority)。 它将新的触发器，值，描述和选项的代码片段添加到当前的片段列表中。 请参阅| UltiSnips-syntax | 有关函数参数的含义的详细信息。 优先级是一个数字，定义哪个代码片段应该比其他代码片段优先。 请参阅| UltiSnips-add-snippets |中的priority关键字。

#### 3.5.2 UltiSnips＃Anon

第二个函数是UltiSnips＃Anon（value，...）。它展开一个匿名代码段。匿名片段在当场定义，展开后立即丢弃。匿名代码段不会添加到全局代码段列表中，因此无法再次展开，除非该函数再次被调用。该函数采用三个可选参数，顺序为：trigger,,description,options。参数与| UltiSnips＃AddSnippetWithPriority |的参数有相同功能。trigger, 和options选项可以更改代码段展开的方式。可以在代码段定义中指定options的参数。每个参数的功能请参阅| UltiSnips-snippet-options |的列表。这里的描述仅为注释作用。

例如：

inoremap <silent> $$ $$ <C-R> = UltiSnips＃Anon（'：latex：\`$ 1 \`'，'$$'）<cr>

每当键入两个$符号时，将展开该片段。

注意：映射的右侧以'$$'触发器的迅速重复键入开始，并将'$$'作为触发参数传递给该函数。这是必需的，以便UltiSnips能够访问键入的字符，以便它可以确定触发器是否匹配。

#### 3.5.3 UltiSnips＃SnippetsInCurrentScope

第三个函数是UltiSnips＃SnippetsInCurrentScope，该函数等价与snipmate的 GetSnipsInCurrentScope函数。

该函数返回触发器与当前词匹配的所有代码片段的vim型字典，如果您需要当前缓冲区的所有代码片段信息，您可以简单地将1（这意味着所有）作为此函数的第一个参数传递，并使用全局变量g：current\_ulti\_dict\_info获取结果（参见下面的示例）。

此函数不会直接为ultisnips添加任何新功能，但允许使用第三方插件来收集当前可用的片段。

这种第三方插件的一个例子是SnippetCompleteSnipMate，它使用GetSnipsInCurrentScope函数将当前可用的片段与用户定义的缩写收集整理，并提供相应补全菜单。

此脚本位于http://www.vim.org/scripts/script.php?script\_id=4276。

注意：如果你检查上面的网站，它列出了两个依赖关系：SnippetComplete插件和snipmate.你需要SnippetComplete插件，不需要snipmate，则你只需要定义函数GetSnipsInCurrentScope。把下面代码放入vimrc文件中：

function! GetSnipsInCurrentScope()

return UltiSnips#SnippetsInCurrentScope()

endfunction

如何使用此函数的第二个示例如下：

function! ExpandPossibleShorterSnippet()

if len(UltiSnips#SnippetsInCurrentScope()) == 1 "only one candidate...

let curr\_key = keys(UltiSnips#SnippetsInCurrentScope())[0]

normal diw

exe "normal a" . curr\_key

exe "normal a "

return 1

endif

return 0

endfunction

inoremap <silent> <C-L> <C-R>=(ExpandPossibleShorterSnippet() == 0? '': UltiSnips#ExpandSnippet())<CR>

如果您的代码段的触发器是lorem，若没有其他代码片段的触发器是lor时，当键入lor，然后敲击<C-L>将展开此片段。

关于如何使用此函数来提取当前缓冲区的所有片段的第三个示例：

function! GetAllSnippets()

call UltiSnips#SnippetsInCurrentScope(1)

let list = []

for [key, info] in items(g:current\_ulti\_dict\_info)

let parts = split(info.location, ':')

call add(list, {

\"key": key,

\"path": parts[0],

\"linenr": parts[1],

\"description": info.description,

\})

endfor

return list

endfunction

变量g：current\_ulti\_dict\_info是为了避免与存在的第三方插件冲突。 定义包含文件路径，和行号

### 3.6 关于缺少python支持的警告

当加载UltiSnips时，它将检查Vim是否支持python。 如果没有检测到支持，将显示警告并跳过UltiSnips的装载。

如果要禁止此警告消息，您可以将以下行添加到您的vimrc文件中。

let g：UltiSnipsNoPythonWarning = 1

如果您的Vim配置文件在多个系统中共享，那么其中一些vim可能吗，没有python支持。

## 4. UltiSnips的snippets文件的语法

本章介绍如何编写自己的片段和片段定义句法。示例用于帮助说明。

### 4.1添加snippet

请参阅| UltiSnips-snippet-search-path |以了解哪些目录与snippet定义应该位于哪里。

使用类似于Vim检测| ftplugins |的策略，UltiSnips遍历sinippet的目录，查找具有以下名称的文件：ft.snippets，ft \_ \*或ft / \*，其中“ft”是“ filetype“和”\*“是一个类似shell的通配符，匹配任何字符串包含空字符串。下表显示了一些典型的snippets文件名及其相关的文件类型。

snippet filename filetype ~

ruby.snippets ruby

perl.snippets perl

c.snippets c

c\_my.snippets c

c/a c

c/b.snippets c

all.snippets \*all

all/a.snippets \*all

注意：all文件类型是唯一的。它代表不管文件类型如何，都可以可以使用的片段。例如，日期插入代码片段适用于all.snippets文件。

UltiSnips支持Vim的点缀文件类型语法。例如，如果您为CUDA C ++框架定义了一个虚拟文件类型，例如“：set ft = cuda.cpp”，UltiSnips将搜索并激活cuda和cpp文件类型的片段。

snippets文件的语法很简单。以＃字符开头的所有行都被视为注释。被UltiSnips忽略。

以关键字'extends'开头的行提供了一种组合片段文件的方式。当“'extends”指令包含在某个snippets文件中时，该代码片段等效于该代码片段于列出文件类型的相应代码片段的组合。即此时可以使用列出文件类型的相应代码片段。

语法如下所示：

extends ft1，ft2，ft3

例如，cpp.snippets中的第一行如下所示：

extends c

当UltiSnips激活cpp文件的片段时，它首先查找所有c片段并激活它们。这是一种sinippet代码片跨文件类型重用的办法。在片段文件中允许使用多个“扩展”行，并且它们可以在文件的任何位置。

以关键字“priority”开头的行设置了此snippet文件中所有位于该行之后的代码片段的优先级。文件的默认优先级始终为0.当代码片段展开时，UltiSnips将从所有源中收集与触发器匹配的所有代码段定义，并仅保留优先级最高的。例如，所有默认片段都具有优先级<0，因此用户自定义的片段总是覆盖默认定义的片段。

以关键字“snippet”开头的行表示代码段定义的开始，以关键字“endsnippet”开头的行表示代码定义结束。代码段定义位于这两行之间。下面时shell文件的'if'语句的代码片段。

snippet if "if ... then (if)"

if ${2:[[ ${1:condition} ]]};

then

${0:#statements}

fi

endsnippet

起始行采用以下形式：

snippet tab\_trigger [ "description" [ options ] ]

tab\_trigger是必需的，但描述和选项是可省略的。'tab\_trigger'是用于触发代码段的单词或字符串序列，通常使用单个单词，也可以是正则表达式。但tab\_trigger不能含有空格，当tab\_trigger含有空格或者是正则表达式时必须加上双引号。如：

snippet "tab trigger" [ "description" [ options ] ]

双引号并不是触发器的一部分。要激活该片段时：输入tab trigger，按下tab键就会展开片段。

含有空格的触发器必须使用双引号并非技术需要。任何匹配的字符都可以。例如，这是一个有效的片段起始行：

snippet !tab trigger! [ "description" [ options ] ]

引号可以作为触发器的一部分，但必须在触发器外加上另一对匹配的字符。如：

snippet !"tab trigger"! [ "description" [ options ] ]

要激活此片段，键入：“tab trigger”

'description'是描述触发器的字符串。 用于记录代码片段并将其与其他片段具有相同触发器的代码片段进行区分。 当代码段被激活并且多个代码段匹配时（即有多个代码段触发器相同），UltiSnips会显示匹配的代码片段列表及其描述，用户根据这些描述来选择他们想要的片段。

'options'控制代码段的行为。 选项由单个字符指示。 代码段的“选项”字符可以是多个参数的组合。

基本格式：

snippet <tigger> "注解" <选项>

${1:name}

${0}

endsnippet

支持直接用shell script，也可以通过!v嵌入vimscript或者!p嵌入python

snippet <tigger> "注解" <选项>

`shellcode或者!v vimscriot 或者!p pythoncode`

endsnippet

python自动定义的变量

fn文件名

path文件全路径

t占位符t[1]相当于${1}

snip方法

也可用global !p直接嵌入通用的python脚本

global !p

endglobal

tigger可以是关键字或者时正则表达式。关键字snippet和endsnippet以及global和endglobal都需顶格书写

### 4.2 选项解释

除了互斥选项外，都可以组合：

b 仅当触发器是一行的第一个字或前面有空白字符时才触发。默认只要前面有空白符，任何位置都能触发

i 当触发器在词中或词后也能触发

w 标签触发器在非字母字符之前或之后也可触发

r 正则表达式 。使用此选项，该选项卡触发器是一个python正则表达式。 如果最近键入的字符与正则表达式匹配，则该片段将被扩展。 注意：正则表达式必须位于双引号内.

t 默认情况下如果片段定义包含前导制表符，UltiSnips会按照Vim 'shiftwidth'，'softtabstop'，'expandtab'和'tabstop'设置的制表符展开代码中的制表符。（例如，如果设置了'expandtab'，则制表符将被替换为空格。）如果设置了此选项，UltiSnips将忽略Vim设置并插入制表符。此选项对于与制表符分隔格式相关的片段非常有用。

s 在行尾的光标跳到下一个制表位删除空格。用于行尾有制表位和可选文本时。

m 删除每行最后一个字符后的所有空格。当代码片中有空格组成的行时，删除空格用空来代替。

e 上下文片段 - 使用此选项扩展代码片段不仅可以由以前的字符在行中进行控制，还可以通过任何给定的python表达式进行控制。 该选项可以与其他选项一起指定，如“b”。

A 输入关键字自动触发，不需tab键

### **4.3 转义字符**

在代码段定义中，字符“'”，“{”，“$”和“\”具有特殊的含义。 如果你想从字面上插入这些字符之一，请将其，加上反斜杠\转义。

### 4.4 明文

代码片中显示的是什么，补充的就是什么。相当于常量。

------------------- SNIP -------------------

snippet bye "My mail signature"

Good bye, Sir. Hope to talk to you soon.

- Arthur, King of Britain

endsnippet

------------------- SNAP -------------------

bye<Tab> -->

Good bye, Sir. Hope to talk to you soon.

- Arthur, King of Britain

### 4.5 视觉占位符

代码段可以包含名为$ {VISUAL}的特殊占位符。 $ {VISUAL}变量将用触发前选择的文本进行展开。

使用占位符定义代码段后，在Vim的Visual模式选择一些文本，然后按下tab键，再用于触发展开代码段的键，再按tab。所选文本将被删除，然后在代码片段中代替$ {VISUAL}占位符展开。

$ {VISUAL}占位符可以设置默认值    $ {VISUAL：默认文字}

$ {VISUAL}占位符也可以定义一个转换 $ {VISUAL：default / search / replace / option}。

例如：

------------------- SNIP -------------------

snippet t

<tag>${VISUAL:inside text/should/is/g}</tag>

endsnippet

------------------- SNAP -------------------

输入 this should be cool

将光标移到should上

按下viw 然后按<Tab>按 "t" 再按 <Tab>

结果为：

this <tag>is</tag> be cool

如果直接按t按tab结果为

<tag>inside text</tag>

### 4.6 插值

#### 4.6.1 UltiSnips支持用shell代码

使用shellcode的语法很简单：`shellcode`。 当一个代码段被触发时，UltiSnips首先将它写入一个临时脚本，然后执行该脚本来运行shellcode。随后 shellcode被执行的结果替代。 任何可以作为脚本运行的内容都可以在shellcode中使用。 包括一个shebang行，例如＃！/ usr / bin / perl，你的代码片段也可以使用其他程序来运行脚本，例如perl。

举列：此代码段使用shell命令插入当前日期。

------------------- SNIP -------------------

snippet today

Today is the `date +%d.%m.%y`.

endsnippet

------------------- SNAP -------------------

today<tab> ->

Today is the 15.07.09.

此代码段使用perl插入当前日期。

------------------- SNIP -------------------

snippet today

Today is `#!/usr/bin/perl

@a = localtime(); print $a[3] . '.' . $a[4] . '.' . ($a[5]+1900);`.

endsnippet

------------------- SNAP -------------------

today<tab> ->

Today is 15.6.2009.

#### 3.6.2 UltiSnips支持用VimScript代码

使用vimscript的语法为`!v vimscript`

例如：

------------------- SNIP -------------------

snippet indent

Indent is: `!v indent(".")`.

endsnippet

------------------- SNAP -------------------

输入4空格 indent<tab> ->

Indent is: 4.

#### 3.6.3.UltiSnips支持用python代码

使用python的语法类似于Vim。`!p pythoncode`或者`＃！/usr/bin/python pythoncode`

在python代码中自动定义的变量有：

fn:文件名

path:文件全路径

t:占位符t[1]相当于${1}，t[2]相当于${2}

snip:方法

context：条件的结果

sinp对象提供的方法有：

snip.mkline(line="", indent=None):用指定符号画一行。如果indent是None，那么mkline会添加当前'tabstop'和'expandtab'定义的空格。

snip.shift(amount=1): 将mkline 右侧使用的默认缩进级别移动'shiftwidth'，'amount'定义的空格数。

snip.unshift(amount=1): 将mkline左侧的缩进级别移动由'shiftwidth'，'amount'定义的空格数。

snip.reset\_indent():将缩进级别重置为其初始值。

snip.opt(var, default):检查Vim变量“var”是否已设置。如果设置，它返回变量的值;否则返回“default”的值。

snip对象还提供了一些属性：

snip.rv： 'rv'是返回值，其文本将替换代码段定义中的python块。它初始为空字符串。这里不推荐使用'res'变量。

snip.c：其值为定义中在python代码的当前位置。一旦插值完成，就将其设置为空字符串。因此，可以检查snip.c是否为空，以确保插值仅完成一次。这里不推荐使用“cur”变量。

snip.v：与$ {VISUAL}占位符相关的数据。有两个属性

snip.v.mode（'v'，'V'，'^ V'，参见 visual-mode）

snip.v.text 所选文本。

snip.fn：当前文件名。

snip.basename：当前已被删除文件名

snip.ft：当前文件类型。

snip.p：最后选定的占位符。将包含具有以下属性的占位符对象：

'current\_text' –选定的当前文本

'start' – 选定文本的开始

'end' – 选定文本的结束

“snip”对象还提供以下文本操作符：

snip >> amount:等效于snip.shift(amount)

snip << amount: 等效于snip.unshift(amount)

snip += line: 等效于"snip.rv += '\n' + snip.mkline(line)"

在python块中定义的任何变量都可以在同一个代码片段中的其他python块中使用。此外，python模块的vim'，'re'，'os'，'string'和'random'都是在代码段范围内预先导入的。可以使用python'import'命令导入其他模块。

Python代码允许非常灵活的代码段。例如，以下代码段将镜像同一行上的第一个tabstop值，但右对齐并以大写形式。

------------------- SNIP -------------------

snippet wow

${1:Text}`!p snip.rv = (75-2\*len(t[1]))\*' '+t[1].upper()`

endsnippet

------------------- SNAP -------------------

wow<tab>Hello World ->

Hello World

以下代码段使用正则表达式选项，并说明使用python的匹配对象的正则表达式分组。sippet展开内容取决于触发器，触发器也可变化。

------------------- SNIP -------------------

snippet "be(gin)?( (\S+))?" "begin{} / end{}" br

\begin{${1:`!p

snip.rv = match.group(3) if match.group(2) is not None else "something"`}}

${2:${VISUAL}}

\end{$1}$0

endsnippet

------------------- SNAP -------------------

be<tab>center<c-j> ->

\begin{center}

\end{center}

------------------- SNAP -------------------

be center<tab> ->

\begin{center}

\end{center}

第二种形式是第一种形式的变体; 都产生相同的结果，但它说明正则表达式分组的工作原理。 使用正则表达式有一些缺点，如果您使用<Tab>键来展开代码段，那么当你你输入“be form <Tab>”，期望得到“"be formatted"，但最终到上述SNAP，而不是你想要的。此外该片段很难阅读。

但是，最大的优点是您可以创建考虑了“触发器”前的文本的snippet片段。 这样，您可以使用它来创建后缀片段，这在某些IDE中很受欢迎。

------------------- SNIP -------------------

snippet "(\w+).par" "Parenthesis (postfix)" r

(`!p snip.rv = match.group(1)`$1)$0

endsnippet

------------------- SNAP -------------------

something.par<tab> ->

(something)

------------------- SNIP -------------------

snippet "([^\s].\*)\.return" "Return (postfix)" r

return `!p snip.rv = match.group(1)`$0

endsnippet

------------------- SNAP -------------------

value.return<tab> ->

return value

#### 3.6.4.Global Snippets

通用片段提供了在所有某个snippets的所有代码片中重用该代码的方法。相当于c中被调函数。目前，只支持python代码。 通用片段的执行结果放入每个片段文件中的python代码块的全局变量中。 要创建全局代码段，请使用关键字“global”替代“sinppet”，对于python代码，您可以使用“!p”作为触发器。 例如，以下代码片段生成与前面示例相同的结果。 但是，使用此语法，“upper\_right”代码段可以被其他代码段重用。

------------------- SNIP -------------------

global !p

def upper\_right(inp):

return (75 - 2 \* len(inp))\*' ' + inp.upper()

endglobal

snippet wow

${1:Text}`!p snip.rv = upper\_right(t[1])`

endsnippet

------------------- SNAP -------------------

wow<tab>Hello World ->

Hello World

Python全局函数可以存储在python模块中，然后导入。这使得所有代码段文件容易访问全局函数。 从Vim 7.4开始，您可以将python文件放到〜/ .vim / pythonx中，直接导入到您的代码片段中。 例如使用〜/ .vim / pythonx / my\_snippets\_helpers.py

global !p

from my\_snippet\_helpers import \*

endglobal

### 4.7 制表位和占位符

snippets用于快速将重复使用的代码插入到文档中。 通常文本中的某些内容是可变的。 Tabstops用于方便修改这些可变内容。 使用tabstops时，当代码片被触发后，将光标放在可变内容上，输入所需的内容，就可将此代码变为适合此文档的特定代码。然后跳转到下一个变量组件，输入该内容，然后继续，直到所有变量组件完成。

tabstop的语法是美元符号，后跟一个数字，例如“$1”。 Tabstops从1号开始，按顺序进行。 '$0'是一个特殊的tabstop。无论定义了多少个tabstops，它始终是片段中的最后一个tabstop。 如果没有定义“$0”，则在代码段末尾将自动定义“$ 0”。

------------------- SNIP -------------------

snippet letter

Dear $1,

$0

Yours sincerely,

$2

endsnippet

------------------- SNAP -------------------

letter<tab>Ben<c-j>Paul<c-j>Thanks for suggesting UltiSnips!->

Dear Ben,

Thanks for suggesting UltiSnips!

Yours sincerely,

Paul

可以使用<c-j>跳转到下一个tabstop，<c-k>跳转到前一个。 由于许多人（包括我自己）使用<Tab>来展开snippet，所以不能使用<Tab>键来向前跳。

tabstop可以设置默认值，语法为${number:默认文本}，默认文本可能是通常可变内容的值，也可能是一个单词或短语，用于提醒期待输入的内容。

下面示例说明了在编辑shell脚本中'case'语句的代码段。tabstops使用默认值来提醒用户预期的值。

------------------- SNIP -------------------

snippet case

case ${1:word} in

${2:pattern} ) $0;;

esac

endsnippet

------------------- SNAP -------------------

case<tab>$option<c-j>-v<c-j>verbose=true

case $option in

-v ) verbose=true;;

esac

tabstop可以嵌套，语法为${1:txet1${2:text2}。在第一个tabstop中键入任何文本将使用键入的文本替换默认值，包括第二个tabstop，第二个tabstop基本上被删除。当触发Tabstop跳转时，UltiSnips移动到'$0'或’$3’。

------------------- SNIP -------------------

snippet a

<a href="${1:http://www.${2:example.com}}"</a>

$0

</a>

endsnippet

------------------- SNAP -------------------

当此代码段展开时，第一个tabstop的默认值为“http://www.example.com”。 此时键入的内容会替换整个<http://www.example.com>，如只想替换“example.com”，请跳转到下一个tabstop。 它的默认值为“example.com”。

a<tab><c-j>google.com<c-j>Google ->

<a href="http://www.google.com">

Google

</a>

a<tab>#top<c-j>Top ->

<a href="#top">

Top

</a>

------------------- SNIP -------------------

snippet a

<a href="$1"${2: class="${3:link}"}>

$0

</a>

endsnippet

------------------- SNAP -------------------

在这个代码片段中，'$ 1'标志着第一个tabstop。表示总是要为'href'属性添加一个值。 输入网址并按<c-j>后，光标将跳转到'$2'。 此tabstop是可选的。 默认文本为“class = "link"”。 您可以按<c-j>接受该默认值，光标将跳转到'$3'，输入值，也可以按退格键，从而把$2替换为空字符串，等价于将其删除。 在任一情况下，按<c-j>表示接受默认值或者调制下一个tabstop，直到最终tabstop。

a<tab>http://www.google.com<c-j><c-j>visited<c-j>Google ->

<a href="http://www.google.com" class="visited">

Google

</a>

a<tab>http://www.google.com<c-j><BS><c-j>Google ->

<a href="http://www.google.com">

Google

</a>

当然默认文本也可以被镜像、转换或者插值。

### 4.8 镜像

镜像tabsto中的内容。 在代码片展开时，当您输入tabstop的值时，该tabstop的所有镜像都将替换为相同的值。 要镜像某个tabstop，只需使用“美元符号后跟数字”（例如“$ 1”）再次插入该tabstop。

一个tabstop可以在一个代码片段中多次进行镜像，并且可以在任一一个镜像处改变所有镜像的值。 镜像的tabstop可以定义默认值。 只有第一个需要一个默认值。其余镜像的tabstop将自动采用默认值。

镜像可用于开始标签，例如TeX'开始'和'结束'标签标签，XML和HTML标签以及C代码#ifndef块。 以下是一些片段示例。

------------------- SNIP -------------------

snippet env

\begin{${1:enumerate}}

$0

\end{$1}

endsnippet

------------------- SNAP -------------------

env<tab>itemize ->

\begin{itemize}

\end{itemize}

------------------- SNIP -------------------

snippet ifndef

#ifndef ${1:SOME\_DEFINE}

#define $1

$0

#endif /\* $1 \*/

endsnippet

------------------- SNAP -------------------

ifndef<tab>WIN32 ->

#ifndef WIN32

#define WIN32

#endif /\* WIN32 \*/

### 4.9 转换或替换

转换就像镜像，但不是只从原始tabstop逐字复制，而可以进行一些指定的变化：如大写变小写。

转换语法如下：

${<tab\_stop\_no/regular\_expression/replacement/options}

参数如下：

tab\_stop\_no 要引用的tabstop的编号

regular\_expression 判断tabstop内容是否匹配的正则表达式

replacement 替换字符串，下面详细说明

options 正则表达式的选项

选项可以是下列值的任意组合：

g 所有匹配项全部替换。配正则表达式的内容可能有多个，默认情况下只有第一个匹配项被替换，使用此选项则会所有匹配项都被替换。

i 不区分大小写。默认情况下，正则表达式匹配时区分大小写。使用此选项，将不区分大小写。

m 默认情况下，正则表达式中的'^'和'$'特殊字符仅适用于整个字符串的开头和结尾，当选择多行，则将其完全转换为整行单行字符串。而使用此选项，'^'和'$'特殊字符匹配字符串中任何行的开始或结束（用换行符 - '\ n'分隔），而不会将多行转为一行。

a ascii转换。默认情况下，在raw utf-8字符集上进行转换。使用此选项，可以在不同的ASCII字符集上进行匹配，例如'à'将变为'a'。此选项需要python包含'unidecode'。

正则表达式的语法超出了本文档的范围，参见htp://docs.python.org/library/re.html。

#### 4.9.1替换

替换字符串可以包含$number变量，例如$1，表示匹配正则表达式的第一个匹配项。 $0变量是特殊的，表示所有匹配项。 替换字符串还可以包含特殊的转义序列：

\ u - 大写下一个字母

\ l - 小写下一个字母

\ U - 大写一切，直到下一个\ E

\ L - 小写一切，直到下一个\ E

\ E – 结束\L或则\U产生的大写或小写，

\ n - 换行符

\ t - 文字标签

替换字符串可以使用条件替换。语法为使用语法（？no：text：other text），表示如果组$no与正则表达式匹配，插入“text”，否则插入“other text”。 “其他文本”是可选的，如果没有提供，默认为空字符串“”。 这个功能非常强大。 它允许您将可选文本添加到代码片段中。

#### 4.9.2.示例

例1:大写第一个字符

------------------- SNIP -------------------

snippet title "Title transformation"

${1:a text}

${1/\w+\s\*/\u$0/}

endsnippet

------------------- SNAP -------------------

title<tab>big small ->

big small

Big small

例2: 大写所有单词的第一个字母

------------------- SNIP -------------------

snippet title "Titlelize in the Transformation"

${1:a text}

${1/\w+\s\*/\u$0/g}

endsnippet

------------------- SNAP -------------------

title<tab>this is a title ->

this is a title

This Is A Title

例3: ASCII 转换

------------------- SNIP -------------------

snippet ascii "Replace non ascii chars"

${1: an accentued text}

${1/.\*/$0/a}

endsnippet

------------------- SNAP -------------------

ascii<tab>à la pêche aux moules

à la pêche aux moules

a la peche aux moules

例4: 正则表达式组

这是一个智能的c风格printf片段，第二个tabstop仅在第一个tabstop中有格式（％）字符时显示。

------------------- SNIP -------------------

snippet printf

printf("${1:%s}\n"${1/([^%]|%%)\*(%.)?.\*/(?2:, :\);)/}$2${1/([^%]|%%)\*(%.)?.\*/(?2:\);)/}

endsnippet

------------------- SNAP -------------------

printf<tab>Hello<c-j> // End of line ->

printf("Hello\n"); // End of line

But

printf<tab>A is: %s<c-j>A<c-j> // End of line ->

printf("A is: %s\n", A); // End of line

### 4.10 清除代码片段

#### 4.10.1清除整个snippets文件

使用clearsnippets来清除某一个文件类型的代码片段

------------------- SNIP -------------------

clearsnippets

------------------- SNAP -------------------

#### 4.0.2.按优先级清除代码片段

'clearsnippets' 也可以用于移除所有优先级低于当前优先级的代码片段。例如，以下例子清除所有优先级为<= 1的片段，即使示例代码段在“clearsnippets”之后定义

------------------- SNIP -------------------

priority 1

clearsnippets

priority -1

snippet example "Cleared example"

This will never be expanded.

endsnippet

------------------- SNAP -------------------

#### 410.3按触发器清除代码片段

'clearsnippets'也可以清除指定的某些代码片段，只需在后面只出要移除代码片的触发器。

------------------- SNIP -------------------

clearsnippets trigger1 trigger2

------------------- SNAP -------------------

### 4.11 上下文代码片段

context代码片可以在定义代码片时用e选项启用，语法有两种：

snippet tab\_trigger "description" "expression" option

或者

context "expression"

snippet tab\_trigger "description" options

“expression”可以是任何python表达式。如果'expression'为“True”，那么这个代码段将被扩展。 “表达式”必须用双引号包裹。

以下python模块将自动导入到评估'expression'的范围之内：'re'，'os'，'vim'，'string'，'random'。

全局变量`snip`将提供以下属性：

'snip.window' - 'vim.current.window'的别名

'snip.buffer' - 'vim.current.window.buffer'的别名

'snip.cursor'– 光标对象，其行为类似'vim.current.window.cursor'，但是零索引并且具有下附加方法：

 - 'preserve（）' - 执行pre / post / jump操作的特殊方法;

 - 'set（line，column）' - 将光标设置为指定的行和列;

 - 'to\_vim\_cursor（）' - 返回1索引光标，适合分配到'vim.current.window.cursor';

'snip.line'和'snip.column' – 光标位置（零索引）的别名;

'snip.visual\_mode' - （'v'，'V'，'^ V'，参见 | visual-mode |）;

'snip.visual\_text' – 可视模式下选择的最后文字;

'snip.last\_placeholder' - 前一个的代码段的最后一个活动占位符，具有以下属性：

-'current\_text' –选定的当前文本

-'start' – 选定文本的开始

-'end' – 选定文本的结束

举例：

------------------- SNIP -------------------

snippet r "return" "re.match('^\s+if err ', snip.buffer[snip.line-1])" be

return err

endsnippet

------------------- SNAP -------------------

只有前一行从'if err'前缀开始时，该代码段才会扩展为'return err'。

注意：上下文片段优先于非上下文片段。如果没有上下文匹配，则可以使用非上下文片段。

举例：

------------------- SNIP -------------------

snippet i "if ..." b

if $1 {

$2

}

endsnippet

snippet i "if err != nil" "re.match('^\s+[^=]\*err\s\*:?=', snip.buffer[snip.line-1])" be

if err != nil {

$1

}

endsnippet

------------------- SNAP -------------------

如果上一行匹配'err：='前缀，那么这个代码段将扩展为'if err！= nil'，否则默认的'if'片段将被扩展。

将上下文条件移动到单独的模块是一个好主意，因此可以由其他UltiSnips用户使用。在这种情况下，应使用'global'关键字导入模块，如下所示：

------------------- SNIP -------------------

global !p

import my\_utils

endglobal

snippet , "return ..., nil/err" "my\_utils.is\_return\_argument(snip)" ie

, `!p if my\_utils.is\_in\_err\_condition():

snip.rv = "err"

else:

snip.rv = "nil"`

endsnippet

------------------- SNAP -------------------

只有当光标位于return语句中时，该代码段才会扩展，然后根据它所在的“if”语句将扩展为“err”或“nil”。 'is\_return\_argument'和'is\_in\_err\_condition'是自定义python模块的一部分，在此示例中称为“my\_utils”。

上下文的条件可以返回任何可以用作python的'if'语句的条件的值，如果为True'，那么代码段将被扩展。 “条件”的值在代码段中的“snip.context”变量中可用：

------------------- SNIP -------------------

snippet + "var +=" "re.match('\s\*(.\*?)\s\*:?=', snip.buffer[snip.line-1])" ie

`!p snip.rv = snip.context.group(1)` += $1

endsnippet

------------------- SNAP -------------------

该代码片段将在行尾展开为'var1 + ='，从'var1：='开始。

您可以使用以下技巧从以前的代码段捕获占位符文本：

------------------- SNIP -------------------

snippet =“desc”“snip.last\_placeholder”Ae

`！p snip.rv = snip.context.current\_text` == nil

endsnippet

------------------- SNAP -------------------

只有当您在其他代码段中替换选定的tabstop（如在“$ {1：var}”时），并且将以'== nil'后面的tabstop值替换该tabstop，则该片段才会被扩展。

### 4.12 Snippets action

Snippets action是任意可以在代码片段的特定点执行的Python代码。

有三种类型的操作：

\* Pre-expand - 在触发条件匹配之后调用，但在片段实际扩展之前;

\* Post-expand - 在片段展开后和插值第一次被应用后引用，但在跳转到第一个占位符之前调用。

\* Jump - 在跳到下一个/ 前一个占位符之后调用。

指定的代码将在上面定义的阶段进行评估，并且在| UltiSnips-context-snippets | 列出的相同的全局变量和模块将可用。

注意：所有缓冲区修改都应使用名为“snip.buffer”的特殊变量。 不是'vim.current.buffer'，也不是'vim.command（“...”）'，因为在这种情况下，UltiSnips将无法跟踪缓冲区中的变化。

'snip.buffer'与'vim.current.window.buffer'具有相同的接口。

#### 4.12.1 Pre-expand actions

Pre-expand actions可以用来匹配一个位置的片段，然后在不同的位置展开它。 例如：纠正代码片段的缩进; 在另一个函数体中扩展函数声明代码片段，其移动的扩展点超出初始函数; 通过扩展代码片段在不同的地方摘取方法重构。

Pre-expand action声明如下：

pre\_expand "python code here"

snippet ...

endsnippet

缓冲区可以通过名为“snip.buffer”的变量在Pre-expand actions代码中进行修改，片段展开位置将被自动调整。

如果光标行（触发器触发时所在行）需要修改，那么必须调用特殊变量方法'snip.cursor.set（line，column）'。 在这种情况下，UltiSnips不会删除任何匹配的触发文本，它应该在操作代码中手动完成。

除了上述定义的变量之外，“snip.visual\_content”也将被声明，并将包含在片段扩展之前选择的文本（类似于$ VISUAL占位符）。

下面的代码段将扩展为4个空格的缩进级别，无论它在哪里被触发。

------------------- SNIP -------------------

pre\_expand "snip.buffer[snip.line] = ' '\*4; snip.cursor.set(line, 4)"

snippet d

def $1():

$0

endsnippet

------------------- SNAP -------------------

以下代码片段将所选代码移动到文件的末尾并创建它的新方法定义

------------------- SNIP -------------------

pre\_expand "del snip.buffer[snip.line]; snip.buffer.append(''); snip.cursor.set(len(snip.buffer)-1, 0)"

snippet x

def $1():

${2:${VISUAL}}

endsnippet

------------------- SNAP -------------------

#### 4.12.2 Post-expand actions

Post-expand actions可以用于一些基于展开代码片的操作。代码风格格式化（例如在方法声明之前和之后插入换行符），根据python插值结果应用动作。

Post-expand action 声明如下：

post\_expand "python code here"

snippet ...

endsnippet

缓冲区可以通过变量“snip.buffer”在Post-expand action代码中进行修改，代码段扩展位置将被自动调整。

变量“snip.snippet\_start”和“snip.snippet\_end”将在action代码范围内定义，并以'(line, column)'的形式指出代码片段的开始和结束位置。

注意：如果post-action在扩展之前插入或删除行，'snip.snippet\_start'和'snip.snippet\_end'会自动调整到正确的位置。

以下片段将展开以完成方法定义，并在代码片段结束后自动插入换行符。创建一个函数将非常有用，它将在特定上下文中插入任意多个换行符。这对创建一个会在特定上下文中插入多个换行符的函数很有用。

------------------- SNIP -------------------

post\_expand "snip.buffer[snip.snippet\_end[0]+1:snip.snippet\_end[0]+1] = ['']"

snippet d "Description" b

def $1():

$2

endsnippet

------------------- SNAP -------------------

#### 3.12.3 Post-jump actions

Post-jump actions可用于根据用户输入占位符的内容来触发的一些代码。例如：在跳转后或最后一次跳转后的匿名跳转（例如执行移动方法重构，然后插入新方法调用）;在最后一次跳转后插入TOC。

Jump-expand action 声明如下：

post\_jump "python code here"

snippet ...

endsnippet

缓冲区可以通过变量“snip.buffer”的在Post-jump actions代码进行修改，代码段扩展位置将被自动调整。

下一个变量和方法也将在actions代码范围内定义：

\*'snip.tabstop' – 跳至tabstop的序号

\*'snip.jump\_direction' - '1'如果向前跳，'-1'向后跳；

\*'snip.tabstops' - tabstop对象列表，见上文;

\*'snip.snippet\_start' – 代码片展开起始位置(line, column)

\*'snip.snippet\_end' - 代码片展开的结束位置(line, column)

\*'snip.expand\_anon（）' - 'UltiSnips\_Manager.expand\_anon（）'的别名;

Tabstop对象有几个有用的属性：

\*'start' - tabstop的起始位置(line, column)，（也可用'tabstop.line'和'tabstop.col'访问）。

\*'end' - tabstop的结束位置(line, column)

\*'current\_text' - tabstop内的文本。

以下代码片段将在vim-help文件的内容列表中插入新内容。

------------------- SNIP -------------------

post\_jump "if snip.tabstop == 0: insert\_toc\_item(snip.tabstops[1], snip.buffer)"

snippet s "section" b

`!p insert\_delimiter\_0(snip, t)`$1`!p insert\_section\_title(snip, t)`

`!p insert\_delimiter\_1(snip, t)`

$0

endsnippet

------------------- SNAP -------------------

'insert\_toc\_item'将在第一次跳转后调用，并将新输入的段添加到当前文件的TOC中。

注意：还可以使转动作触发代码段扩展。 在这种情况下，应该调用'snip.cursor.preserve（）'，以便UltiSnips知道游标已经在所需的位置。

以下示例将在用户跳出方法声明片段后在文件末尾插入方法调用。

------------------- SNIP -------------------

global !p

def insert\_method\_call(name):

vim.command('normal G')

snip.expand\_anon(name + '($1)\n')

endglobal

post\_jump "if snip.tabstop == 0: insert\_method\_call(snip.tabstops[1].current\_text)"

snippet d "method declaration" b

def $1():

$2

endsnippet

------------------- SNAP -------------------

### 4.13 自动触发

注意：vim 7.4.214以后才支持此功能。

许多语言结构只能在特定的地方出现，所以可以在不手动触发的情况下使用代码段。

代码段可以通过在代码段定义中指定“A”选项来标记为自动触发。

在将代码段定义为自动触发后，每个类型字符将检查代码段条件，如果条件匹配，则会触发代码段。

\*警告：\*使用此功能可能导致显着的vim减缓。

以下代码可以在Go 编程中使用:

------------------- SNIP -------------------

snippet "^p" "package" rbA

package ${1:main}

endsnippet

snippet "^m" "func main" rbA

func main() {

$1

}

endsnippet

------------------- SNAP -------------------

当“p”或“m”字符将在行的开头出现时，将自动触发，不需要按触发键TAB。