VIM 学习笔记

阿左 [1]　　Nobody [2]

June 18, 2012

[1]感谢读者
[2]感谢国家

Contents

List of Figures

List of Tables

Abstract

part of vim tech

摘要

部分 vim 技巧

Part I

调用 python 脚本

Chapter 1

调用 python 脚本

1.1 基本介绍

先把函数定义于~/.vimrc 配置文件中。可以直接:so % 来重新载入，但是前提是所有在 vimrc 中的自定义函数都要定义成function! 这种形式。

1.1.1 在状态栏显示信息

一个简单的例子：定义一个在 vim 的状态行中显示"Eat Me" 的消息的函数。然后绑定到F7 键上。

```
" show message in the state bar

function! ShowEatMe()

python << EOF
print 'EAT ME'
EOF
endfunction

map <f7> :call ShowEatMe() <cr>
```

1.1.2 取得打开文件的缓存

下面函数从打开文件的缓存中取得文件内容，然后统计空白行的行数。

```
" count blank lines in buffer

function! CountBlankLine()
python << EOF
import vim
count = 0
for line in vim.current.buffer :
  if len(line) == 0:
    count += 1
print "there are " + str(count) + " blank lines in this file"
EOF
endfunction

map <f7> :call CountBlankLine() <cr>
```

1.1.3 调用 python 脚本

对于一个已经存在的 python 脚本：

```
print 'EAT ME'
```

可以通过:pyfile < 文件名> 来调用已经存在的 python 脚本。

同样对于 python 脚本：

```
import vim

count = 0
for line in vim.current.buffer :
  if len(line) == 0:
    count += 1
print "there are " + str(count) + " blank lines in this file"
```

也可以通过pyfile 命令调用。

1.2 vim 模块

现在有了 vim 模块提供 python 程序对 vim 的操作。

1.2.1 基本常量

基本常量

```
vim.buffers           *python-buffers*
  A sequence object providing access to the list of vim buffers.  The
  object supports the following operations: >
      :py b = vim.buffers[i]  # Indexing (read-only)
      :py b in vim.buffers  # Membership test
      :py n = len(vim.buffers)  # Number of elements
      :py for b in vim.buffers: # Sequential access

vim.windows           *python-windows*
  A sequence object providing access to the list of vim windows.  The
  object supports the following operations: >
      :py w = vim.windows[i]  # Indexing (read-only)
      :py w in vim.windows  # Membership test
      :py n = len(vim.windows)  # Number of elements
      :py for w in vim.windows: # Sequential access

vim.current           *python-current*
  An object providing access (via specific attributes) to various
  "current" objects available in vim:
    vim.current.line  The current line (RW)   String
    vim.current.buffer  The current buffer (RO)   Buffer
    vim.current.window  The current window (RO)   Window
    vim.current.range The current line range (RO) Range
```

Python 脚本的全部sys.stdout 输出都在 vim 的消息区，正常输出像是提示信息。所以的sts.stderr 错误信息像是错误提示。

vim 调用的 python 脚本不支持输入sys.stdin(包括inout()、raw_input())，调用时很可能会发生错误。

1.2.2 错误对象

vim 模块中的错误会抛出vim.err 类型的异常：

```
try:
  vim.command("put a")
except vim.error:
  # nothing in register a
```

1.2.3 缓冲区对象

缓冲区对象可以通过以下途径获得：

- via vim.current.buffer (|python-current|)

- from indexing vim.buffers (|python-buffers|)

- from the "buffer" attribute of a window (|python-window|)

缓冲区对象可以作为一个序列对象。注意在索引与分片操作时结果会有不同：

b[:] 的结果为None，会清空整个缓冲区；b = None 仅仅更新了变量，不会影响缓冲区。

缓存对象的常用的操作：

```
b.append(str)      Append a line to the buffer
b.append(str, nr)  Idem, below line "nr"
b.append(list)     Append a list of lines to the buffer
                Note that the option of supplying a list of strings to
                 the append method differs from the equivalent method
                    for Python's built-in list objects.
b.append(list, nr) Idem, below line "nr"
b.mark(name)      Return a tuple (row,col) representing the position
                    of the named mark (can also get the []"<> marks)
b.range(s,e)       Return a range object (see |python-range|) which
                 represents the part of the given buffer between line
                    numbers s and e |inclusive|.
```

注意：用append() 添加一个新行时，不可以加上换行符'\n'。行尾可以有'\n' 但是会被忽略。所以以下的操作是被允许的：

```
:py b.append(f.readlines())
```

常用的例子：

```
:py print b.name            # write the buffer file name
:py b[0] = "hello!!!"       # replace the top line
:py b[:] = None             # delete the whole buffer
:py del b[:]                # delete the whole buffer
:py b[0:0] = [ "a line" ]   # add a line at the top
:py del b[2]                # delete a line (the third)
:py b.append("bottom")      # add a line at the bottom
:py n = len(b)              # number of lines
:py (row,col) = b.mark('a') # named mark
:py r = b.range(1,5)        # a sub-range of the buffer
```

1.2.4 范围对象

范围对象代表了一部分的缓冲区，取得方法有：

- via vim.current.range (|python-current|) - from a buffer's range() method (|python-buffer|)

常用属性有：

```
r.start     Index of first line into the buffer
r.end       Index of last line into the buffer
```

常用方法有：

```
r.append(str)      Append a line to the range
r.append(str, nr)  Idem, after line "nr"
r.append(list)     Append a list of lines to the range
                Note that the option of supplying a list of strings to
                 the append method differs from the equivalent method
```

```
                    for Python's built-in list objects.
r.append(list, nr) Idem, after line "nr"
```

例子（r 是当前的范围）：

```
# Send all lines in a range to the default printer
vim.command("%d,%dhardcopy!" % (r.start+1,r.end+1))
```

1.2.5 窗口对象

窗口对象代表了一个 vim 窗口。取得窗口对象的方法有：

- via vim.current.window (|python-current|) - from indexing vim.windows (|python-windows|)

窗口对象没有方法，只能通过属性来操作。常用窗口属性：

```
buffer (read-only)  The buffer displayed in this window
cursor (read-write) The current cursor position in the window
                     This is a tuple, (row,col).
height (read-write) The window height, in rows
width (read-write)  The window width, in columns
```

只有水平屏时才能重设调试；只能垂直分屏时才能设置调试对象。