

<转载>vim map nmap

有五种映射存在

- 用于普通模式: 输入命令时。
- 用于可视模式: 可视区域高亮并输入命令时。
- 用于操作符等待模式: 操作符等待中 ("d", "y", "c" 等等之后)。

见下: |omap-info|。

- 用于插入模式: 也用于替换模式。

? 用于命令行模式: 输入 ":" 或 "/" 命令时。

下表是map绑定中, 对应的模式代号。现在先了解一下, 等看完之后再回过头看这个模式代号就会明白了。

字符模式 ~

<Space> 普通、可视、选择和操作符等待

n 普通

v 可视和选择

s 选择 (在可视模式下Ctrl+G进入)

x 可视

o 操作符等待

! 插入和命令行

i 插入

l 插入、命令行和 Lang-Arg 模式的 ":lmap" 映射

c 命令行

我主要讲解一下“n(普通模式)”下的两个绑定命令, 等看完之后就对应的明白别的模式下的命令了。

适用于普通模式的映射命令主要有:

1. :map

[语法] :map {lhs} {rhs} [mapmode-nvo] *:map*

1.1 作用模式: n、v、o (普通、可视和选择、操作符等待)

1.2 命令格式:

:map {lhs} {rhs}

含义：在`:map`作用的模式中把键序列 `{lhs}` 映射为 `{rhs}`，`{rhs}`可进行映射扫描，也就是可递归映射。

1.3 举例：

`:map td :tabnew .<cr>`

含义：在其作用模式（普通、可视、操作符）下，输入`td`等价于输入 `:tabnew . <cr>`。而普通模式下输入`:tabnew . <cr>`就是打开当前目录

如果再定义绑定 `:map ts td`，就是指在其作用模式下输入`ts`等价于`td`，也就是打开当前目录。不过如果没有特殊需要，一般不建议递归映射。

2. :noremap

`:noremap`和`:map`命令相对，作用模式和命令格式都相同，只不过不允许再对`{rhs}`进行映射扫描，也就是`{lhs}`定义后的映射就是`{rhs}`的键序列，不会再对`{rhs}`键序列重新解释扫描。它一般用于重定义一个命令，当然如果`:map`不需要递归映射的话，建议试用`:noremap`

比如：

`:noremap ts td`

它的意思是在其作用模式下，输入`ts`就是输入`td`，但是和`:map`不同的是，此时`td`再不会做进一步扫描解释。虽然之前已经定义了`td`，但是不会对`td`再做扫描

3. :unmap

`:unmap`是对应取消`:map`绑定的 `{lhs}`，作用模式相同，命令格式 `:unmap {lhs}`。

例如：

`:unmap td`

就是取消在其作用模式中`td`的绑定，比如之前`td`被绑定为`:tabnew .<cr>`，此时此绑定消失。

4. :mapclear

`:mapclear`时对应取消所有`:map`绑定的，慎用！

5. :nmap

`:nmap`是`:map`的普通模式板，也就是说其绑定的键只作用于普通模式。

例如：

`:nmap td :tabnew .<cr>` 和 `:map td :tabnew .<cr>` 在普通模式下等效

6. :nnoremap

`:nnoremap`和`:nmap`的关系和`:noremap`和`:map`的关系一样，只是`:nmap`的非递归版

7. :nunmap

`:nunmap`和`:nmap`的关系和`:unmap`和`:map`的关系一样，取消`:nmap`的绑定。

8. :nmapclear

`:nmapclear`是对应取消所有`:map`绑定的，慎用！

看完以上，应该可以发现一个规律，前4个是一组，后4个是一组，后一组比前一组多一个`n`就是指只作用于普通模式。其中每组内`*nore*`是其对应的非递归版、`*un*`是取消绑定某个`<lhs>`绑定、`clear`后缀是取消所有绑定。发现了

这个规律，再翻到前面的模式代号表，你大体可以猜到vmap、xmap、smap、omap是什么意思了吧，以及相对应的nore版本、un版本、clear版本。

另外：

{rhs} 之前可能显示一个特殊字符：

* 表示它不可重映射

& 表示仅脚本的局部映射可以被重映射

@ 表示缓冲区的局部映射

到这一步你可以轻松的长吸一口气，因为相关的命令已经都了解了，记不住没关系，可以随时:help map一下。不过别急，后面还有map更多的选项等着去攻克。

键表 |key-notation|

<k0> - <k9> 小键盘 0 到 9 *keypad-0* *keypad-9*

<S-...> Shift + 键 *shift* *<S-*

<C-...> Control + 键 *control* *ctrl* *<C-*

<M-...> Alt + 键 或 meta + 键 *meta* *alt* *<M-*

<A-...> 同 <m-...> *<A-*

<t_xx> termcap 里的 "xx" 入口键

特殊参数：

1. <buffer>
2. <silent>
3. <special>
4. <script>
5. <expr>
6. <unique>

它们必须映射命令的后边，在其他任何参数的前面。

<buffer> 如果这些映射命令的第一个参数是<buffer>，映射将只局限于当前缓冲区（也就是你此时正编辑的文件）内。比如：

```
:map <buffer> ,w /a<CR>
```

它的意思是在当前缓冲区里定义键绑定，“w”将在当前缓冲区里查找字符a。同样你可以在其他缓冲区里定义：

```
:map <buffer> ,w /b<CR>
```

比如我经常打开多个标签(:tabedit)，想要在各个标签里定义",w"键绑定，那么你只要在每个标签页里分别定义就可，其作用域也只在各自的标签里。同样要清除这些缓冲区的键绑定也要加上<buffer>参数，比如：

```
:unmap <buffer> ,w
```

```
:mapclear <buffer>
```

<silent>是指执行键绑定时不在命令行上回显，比如：

```
:map <silent> ,w /abcd<CR>
```

你在输入,w查找abcd时，命令行上不会显示/abcd，如果没有<silent>参数就会显示出来

<special>一般用于定义特殊键怕有副作用的场合。比如：

```
:map <special> <F12> /Header<CR>
```

<unique>一般用于定义新的键映射或者缩写命令的同时检查是否该键已经被映射，如果该映射或者缩写已经存在，则该命令会失败

<expr>. 如果定义新映射的第一个参数是<expr>，那么参数会作为表达式来进行计算，结果使用实际使用的<rhs>，例如：

```
:inoremap <expr> . InsertDot()
```

这可以用来检查光标之前的文本并在一定条件下启动全能 (omni) 补全。

一个例子：

```
let counter = 0
```

```
inoremap <expr> <C-L> ListItem()
```

```
inoremap <expr> <C-R> ListReset()
```

```
func ListItem()
```

```
let g:counter += 1
```

```
return g:counter . ' '
```

```
endfunc
```

```
func ListReset()
```

```
let g:counter = 0
```

```
return "
```

```
endfunc
```

在插入模式下，CTRL-L插入顺序的列表编号，并返回；CTRL-R复位列表编号到0，并返回空。

<Leader> mapleader

mapleader变量对所有map映射命令起效，它的作用是将参数<leader>替换成mapleader变量的值，比如：

```
:map <Leader>A oanother line<Esc>
```

如果mapleader变量没有设置，则用默认的反斜杠代替，因此这个映射等效于：

```
:map \A oanother line<Esc>
```

意思时输入\A键时，在下一行输入another line并返回到普通模式。

如果设置了mapleader变量，比如：

```
let mapleader = ","
```

那么就等效于:

```
:map ,A oanother line<Esc>
```

<LocalLeader> maplocalleader

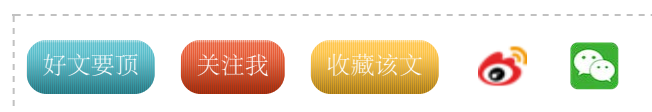
<LocalLeader>和<Leader>类似,只不过它只作用于缓冲区。

因此在设置mapleader和maplocalleader时最好区分开,不要出现冲突。

大体上映射的主要部分已经都提到了,还有很多具体的映射相关的内容可以参见:help map

分类: [Linux shell](#)

标签: [Linux](#), [shell](#), [vim](#), [map](#)



 [lq0729](#)
关注 - 0
粉丝 - 6

+ 加关注

« 上一篇: [Linux学习之"exit函数"](#)


» 下一篇: [<转载>linux下解压命令大全](#)

0
推荐

0
反对

posted @ 2011-12-24 10:33 [lq0729](#) 阅读(10909) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 注册用户登录后才能发表评论,请 [登录](#) 或 [注册](#), [访问](#)网站首页。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库



最新IT新闻:

- 余承东谈消费升级和消费者业务：华为不仅是技术、踏实的“理工男”
 - 人人视频宣布转型海外短视频内容社区
 - 马云：假货既是阿里之痛，更是中国之痛
 - 苹果正起诉Swatch的“Tick different”商标涉侵权
 - 淘汰锂电 中国教授发明磷纳米电池：手机续航增7倍
- » 更多新闻...



最新知识库文章:

- 如何打好前端游击战
 - 技术文章的阅读姿势
 - 马拉松式学习与技术人员的成长性
 - 程序员的“认知失调”
 - 为什么有的人工作多年还是老样子
- » 更多知识库文章...

公告

昵称: [lq0729](#)
园龄: 5年5个月
粉丝: 6
关注: 0
[+加关注](#)

<	2011年12月						>
日	一	二	三	四	五	六	
27	28	29	30	1	2	3	
4	5	6	7	8	9	10	
11	12	13	14	15	16	17	
18	19	20	21	22	23	24	
25	26	27	28	29	30	31	
1	2	3	4	5	6	7	

搜索

常用链接

- [我的随笔](#)
- [我的评论](#)

[我的参与](#)

[最新评论](#)

[我的标签](#)

我的标签

[Linux\(22\)](#)

[线程\(10\)](#)

[互斥量\(6\)](#)

[MFC\(4\)](#)

[条件变量\(3\)](#)

[封装\(2\)](#)

[接口\(2\)](#)

[优化\(2\)](#)

[extern\(2\)](#)

[windbg\(2\)](#)

[更多](#)

随笔分类

[C/C++\(7\)](#)

[Driver\(1\)](#)

[Exploit](#)

[Linux shell\(25\)](#)

[MFC\(1\)](#)

[Reverse](#)

[设计模式\(2\)](#)

随笔档案

[2012年12月 \(2\)](#)

[2011年12月 \(3\)](#)

[2011年10月 \(30\)](#)

文章分类

[Debug\(1\)](#)

[Driver\(2\)](#)

[MFC\(2\)](#)

[它山之石\(2\)](#)

调试

最新评论

1. [Re:Linux学习之互斥量的封装一：封装创建\(pthread_mutex_init\)和销毁\(pthread_mutex_destroy\)](#)

不错，学习了

--罗维

2. Re:MFC中控件的TAB顺序

很有用。。。

--aoguren

3. Re:Linux学习之线程封装七：基于接口的再封装

@小鹏的园地见"Linux学习之线程封装四：基于接口的封装"...

--lq0729

4. Re:Linux学习之线程封装七：基于接口的再封装

你好，你的文章很强大，学习了。但是CLEXecutiveFunctionProvider呢，可以发给我吗，1660405808@QQ.COM,谢谢了！

--小鹏的园地

5. Re:Linux学习之"实现Windows消息机制"

问下CLEvent是什么呢？基于哪个库？

--悠米海

阅读排行榜

1. <转载>vim map nmap(10910)
2. Unix/Linux下的open函数(O_CREAT和O_EXCL)(8265)
3. C/C++的流(stream)对象(4233)
4. Linux学习之"exit函数"(2767)
5. Linux学习之互斥量的封装一：封装创建(pthread_mutex_init)和销毁(pthread_mutex_destroy)(2763)

评论排行榜

1. Linux学习之线程封装七：基于接口的再封装(2)
2. Linux学习之"实现Windows消息机制"(1)
3. Linux学习之互斥量的封装一：封装创建(pthread_mutex_init)和销毁(pthread_mutex_destroy)(1)

推荐排行榜

1. Linux学习之互斥量的封装一：封装创建(pthread_mutex_init)和销毁(pthread_mutex_destroy)(1)