;f easymotion 搜索全局 string
;s easymotion 搜索全局 char
Q 保存并退出文件
;c 复制全文

插入模式 Ctrl+W 删一个 word

- h: 向前移动一个字符
- l: 向后移动一个字符
- j: 向下移动一行
- k: 向上移动一行
- b: 向前移动一个单词
- w: 向后移动一个单词
- e: 移动光标到单词尾部
- ^: 移动光标到行首第一个非空字符处
- $: 移动光标到行末最后一个非空字符处
- gg: 移动光标至文件首行的首个非空格字节
- G: 移动光标至整个文本最后一行的首个非空格字节()
- %: 移动光标到匹配括号的另一端
- 0: 移动光标至行首

> - `5dd` 删除5行
> - `4>>` 将4行代码进行缩进
> - `2cc` 更改接下来的两行，并进入插入模式

w	word 词
s	sentence 句
p	paragraph 段
t	tag(HTML/XML) 标签
[或]	由[]包裹的部分
(或)	由()包裹的部分
<或>	由<>包裹的部分
{或}	由{} 包裹的部分
"	由""包裹的部分
'	由''包裹的部分
`	由``包裹的部分

- `数字 + gg`，表示跳转到文件的第几行
- `数字 + G`，也表示跳转到文件的第几行。但是我平时都使用 gg 这种跳转方式，毕竟在一个键位上按两次可比按它的大写字母要快的多
- `数字 + L`，移动到窗口的倒数第几行
- `数字 + H`，移动到窗口的第几行
- `数字 + M`，与单纯的使用 `M` 效果一样

复合命令	等效长命令	含义
A	$a	在行尾进入插入模式
I	^i	在行首进入插入模式
o	A<CR>	在下一行进入插入模式
O	ko	在上一行进入插入模式
S	^c$	删除当前行并进入插入模式
C	c$	删除当前光标位置到行尾的内容并进入插入模式
s	cl	删除光标后的一个字符并进入插入模式

我们也可以使用 `t` 来跳转光标到搜索的字符处，它与 `f` 的区别在于 `f` 直接跳转光标到指定字符处，`t` 跳转光标到指定字符的前一个位置

行内查找字符使用 `f{char}` 来进行，即使用f后面加一个字符，会快速跳转到行内第一个出现该字符的位置，例如下面一段话

我们当然可以使用 `cw` 来删除5，然后在插入模式下写入10。但是这里要介绍一个新的方法——使用 `Ctrl + a` 来在数字文本上进行递增操作，它也是一个操作符。符合之前的公式。这里假设光标在 5 的位置，然后执行 `5 Ctrl+a` 来实现将5这个数字文本递增5个的操作

命令	含义
i	在当前光标前进入插入模式
I	在行首进入插入模式
a	在当前光标后面进入插入模式
A	在行尾进入插入模式
o	在下一行进入插入模式
O	在上一行进入插入模式
s	删除当前光标所在字符并进入插入模式
S	删除光标所在行并进入模式
c + motion	删除指定范围的字符，并进入插入模式
C	删除光标所在位置至行尾的字符，并进入插入模式

- >: 右缩进
- <: 左缩进
- =: 自动缩进
- y: 复制
- p: 粘贴
- gu: 变为小写
- gU: 变为大写
- g~: 反转大小写
- 示例：使用 `>G` 将当前行至文件尾部的代码进行缩进

> 其实在插入模式中是可以进行删除操作的，例如如果要删除的字符刚好在光标前面，我们可以使用退格键删除它。另外 `vim` 提供了其他几种方式在插入模式中向前删除单词、行
>
> - <Ctrl + h>: 删除光标前一个词(与退格键相同)
> - <Ctrl + w>: 删除光标前一个单词
> - <Ctrl + u>: 删除至行首

针对这种情况 `vim` 提供了一种新的模式 `插入–普通模式` ，在插入模式中使用 `Ctrl + o` 来进入该模式。该模式运行用户暂时回到普通模式，然后在执行一条普通模式的命令之后自动回到插入模式中。

在进入可视模式后，使用 `<Ctrl +g>` 进入选择模式，此时左下角的字样已经变为 `SELECT` ，表示此时进入了选择模式。在选择模式中，随便输入一个字符，它会删除选中然后输入对应的内容。

处理字符的可视模式与普通的 `motion` 配合使用，可以选中光标移动所经过的字符。可以在普通模式下按 `v` 进入

处理行的可视模式可以与行操作的 `motion` 配合，一次选中一行，可以在普通模式下按 `V` 进入

如果选择到一半发现我们选择错了该怎么办呢？一种方式是退回到普通模式下，然后再重新进入选择模式。但是在这里要介绍一种新的方式——可以按 `o` 重新选择选区的活动段。选择模式下选区一段固定，另一端可以移动。通过多次按下 `o` 来变更需要移动哪一段

```
command = {startpoint},{endpoint} + cmd
```

命令	简写	用途
:[range]delete [x]	d	删除指定范围内的行[到寄存器 x 中]
:[range]yank [x]	y	复制指定范围的行[到寄存器 x 中]
:[line]put [x]	pu	在指定行后粘贴寄存器 x 中的内容
:[range]copy {address}	t	把指定范围内的行拷贝到 {address} 所指定的行之下
:[range]move {address}	m	把指定范围内的行移动到 {address} 所指定的行之下
:[range]join	j	连接指定范围内的行
:[range]normal {commands}	narm	对指定范围内的每一行执行普通模式命令 {commands}
:[range]substitute/{pattern}/{string}/[flags]	s	把指定范围内出现{pattern}的地方替换为{string}
:[range]global/{pattern}/[cmd]	g	对指定范围内匹配{pattern}的所有行，在其上执行Ex 命令 {cmd}

范围符号

符号	用途
.	当前行
$	文件末尾
0	虚拟行，位于文件第一行的上方
1	文件第一行
'm	包含标记m的行
<	高亮选区的起始行
>	高亮选区的结尾行
%	整个文件，相当于 :1,$

之前针对这个例子给出了不同的解决方案，在学习 `.` 命令的时候，是使用 `.` 命令，在学习可视模式的时候使用处理列的可视模式。现在再介绍一种新的方式，通过在命令行模式中使用 `normal` 关键字来告诉vim，我们将要使用普通模式的命令。

它使用的方式为： `range + normal + operator` ，它支持范围操作，表示我们将要针对某个范围来执行普通模式的操作

针对这个例子，首先要构造一个可以使用 `.` 命令的操作，即我们在首行使用 `A;` 在行尾添加分号，接着配合命令模式的范围，加上 `2,$normal.` 表示我们将要从第二行到尾行来执行 `.` 命令

重复上次的ex命令

在普通模式下 `.` 可以重复上一次的修改，但是某些ex命令并没有对其进行修改，如果我想重复通过 `.` 来重复上次的ex命令则无能为力了。而且通过实验也可以发现，它也无法重复由ex命令造成的修改。

可以使用 `@:` 重复上一次的命令。如果执行过 `@:` 进行重复，那么可以使用 `@@` 再次执行上次重复的命令，例如在编写代码时经常会一到的一个问题就是将当前行代码下移一行，但是也不是所有的行都会这么干，那么就可以先使用 `.m.+1` 将当前位置的代码移动到光标的下一行，然后移动光标，在下一个需要次操作的位置执行 `@:` ，后面就可以直接使用 `@@` 来重复上一次的操作了。这里就不再针对它来做演示了。各位小伙伴可以自己来尝试一下

按下 `:` 进入到命令模式之后，可以使用方向键向上或者向下查找历史命令。

单词与字串

在vim中一个单词由字母、数字、下划线或者其他非空白字符组成，单词间以空白字符分割。而字串是由非空白字符序列组成。这个感觉可能很抽象，但是多多练习和尝试应该就很容易明白了。

字串间的移动使用大写的 `W`， `B`。下面来看一个例子

利用标签，快速跳转

vim中提供了标签的方式进行跳转，事先可以在对应位置设置标签，后面通过标签访问该标签所在位置

可以使用 `m{a-z}` 来在任意位置设置标记，而后使用`{a-z}来回到对应标记位置。该命令可以回到之前设置标签时光标所在行和列。

下表列举出了，如何回到这些vim自动标记所在位置

位置标记	含义
``	当前文件中上次跳转动作之前所处的位置
`.	上次修改的地方
`^	上次进入插入模式的位置
`[	上次修改或者复制的起始位置
`]	上次修改或者复制的结尾位置
`<	上次高亮选区的起始位置
`>	上次高亮选区的结尾位置

在匹配的括号间进行跳转

可以使用 `%` 在一组括号中使用，可以跳转到下一个匹配的 `()`、`[]`、`{}`。例如下列操作

我们可以配合 `operator` 来使用，删除括号中的内容。例如下面的代码

使用全局书签在文件间跳转

之前介绍过在文件中可以使用标记，在文件不同位置进行跳转。那个时候说到使用小写字母设置标记，小伙伴们可能会产生疑惑，那大些字母去哪了呢，为什么只能使用小写字母，而大写字母被空着呢？文章写到这里了，我可以告诉大家，大写字母被用到了全局书签里面。

vim中提供了由a到z的有名寄存器，可以在使用 `operator` 的操作前面指定需要使用的寄存器，引用一个寄存器可以使用 `" + 寄存器名` 的格式。这个时候我们之前的公式就又可以扩展了

```
" + regester + operator + motion
```

例如在执行删除的时候 `"add` 将一行删除的内容放到a寄存器中，再次执行 `"bdd` 将内容放到b寄存器中，执行粘贴的时候，可以使用 `"ap` 和 `"bp` 来分别使用 a和b寄存器的内容。

复制寄存器

前面说到使用 `dd` 之类的命令会将被删除的内容放到无名的寄存器中，它的行为有点像普通编辑器中的剪切，那它是不是剪切呢，那么多教程都把它叫做删除，是不是有问题呢。它确实是删除指定，教程说的也没错，vim中有专门存储复制内容的寄存器。普通的删除命令会把被删除的内容保存到无名寄存器中，但是这些内容不会被保存到复制寄存器中。复制寄存器使用 0来表示。即我们可以使用 "0p来将复制寄存器的内容取出。也可以通过命令 :reg 0来查看这个寄存器的内容。

黑洞寄存器

前面说到 dd会将被删除内容放入到无名寄存器中，如果这段内容我确实不想要了，也不想它占用寄存器，有没有什么办法彻底删除呢，答案是使用黑洞寄存器，顾名思义，放入该寄存器中的内容都被吸走丢失了，无法使用了。黑洞寄存器使用 _作为标识符，执行删除指令的时候可以使用 "_dd这样就再也访问不到之前删除的内容了。

系统剪切板

之前我们在vim中复制粘贴的内容，只能在vim中使用。同样的系统中复制粘贴的内容只能在系统其它程序中使用，无法直接粘贴到vim中。我们可以在vim中使用系统剪切板。vim可以使用 `+` 来访问系统剪切板。例如使用 `"+yy` 将内容复制到系统剪切板中，供其他程序使用。

但是在有好的shell工具的加持下，我更喜欢用 `<Ctrl+v>` 这样的方式直接粘贴一大段文字到vim中。或者配合vim的可视模式，直接使用shell中的快捷键从vim中粘贴选中的内容到系统剪切板

表达式寄存器

前面介绍的几种寄存器都是被动的存储静态的内容，只有存储功能。表达式寄存器则可以接受一段vim脚本并执行它并输出结果。表达式寄存器使用 `=` 来表示。
例如在插入模式中可以使用 `<Ctrl+r>=6*6` 来进行数学计算并输出。

解法三：由于我们需要先删除之前的内容再复制，为了快速删除，所以会发生覆盖问题，我们只要不执行删除操作就不会覆盖了，为了一次性完成粘贴替换的操作，可以使用选择模式，之前介绍选择模式的时候说过，在选择模式下 `operator` 会将选中部分作为操作区域。可以使用 `vi"` 来选中引号内容，然后直接使用 `p` 完成复制

示例2：插入模式中使用寄存器

之前已经在介绍表达式寄存器的时候已经介绍了如何在插入模式中使用寄存器，可以使用 `<Ctrl + r> + register` 例如上面的例子可以使用 `<Ctrl + r>0` 来将复制寄存器中的内容写入到光标所在位置。

宏是存储在寄存器中的连续的操作指令，以便后续可以对这些指令进行回放。可以使用 `q` 进行录制，后面跟寄存器名称，表示将接下来的操作记录保存到这个寄存器中。例如使用 `qa` 表示将接下来的操作保存到 `a` 这个寄存器中。退出宏的录制可以直接输入 `q`

针对上面的例子，我们可以执行 `qa` 进行宏的录制，然后使用 `A` 在行尾进入插入模式，接着输入 `;` 完成行尾的操作。然后使用 `I` 进入行首，然后在行首输入 `var` 完成这部分的工作。最后使用 `q` 退出宏的录制。这样就将这个宏保存在了 `a` 寄存器.

我们可以使用 `:reg a` 来查看寄存器的内容。

以并行的方式执行宏

宏是存储在寄存器中的连续的操作指令，以便后续可以对这些指令进行回放。可以使用 `q` 进行录制，后面跟寄存器名称，表示将接下来的操作记录保存到这个寄存器中。例如使用 `qa` 表示将接下来的操作保存到 `a` 这个寄存器中。退出宏的录制可以直接输入 `q`

宏录制完成之后，可以使用 `@ + 寄存器` 来回放寄存器中保存的宏。在回放宏之后可以使用 `@@` 来快速回放上一次回放的宏。

给宏追加命令

还是上面的例子，假设在录制好了宏之后发现我们少了一个j，使用串行话的方式无法顺利执行。这种情况下不需要重新录制宏，只需要在对应寄存器中添加一条指令。

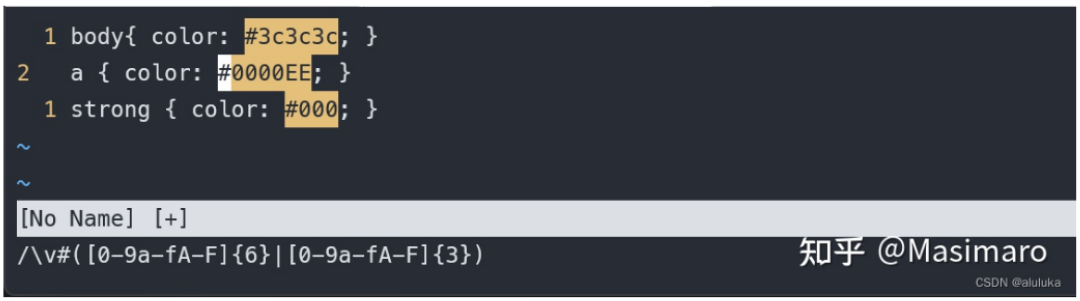
这里补充一下寄存器相关知识。在上一篇介绍寄存器的时候我们只演示了使用小写字母的寄存器，没有提到大写字母的寄存器。根据之前的惯例，大写字母与小写字母都可以使用，大写字母的功能比小写字母要强，例如大写的标签标示全局，小写的只能用于单个文件。这里大写的寄存器与小写的寄存器是同一个寄存器，使用大写时我们可以对寄存器内容进行追加操作。

宏是保存在寄存器中的，`q` 后面加字母表示宏的内容保存在哪个寄存器中，说到这里，聪明的你已经反应过来该如何将命令追加到寄存器中了。那就是使用 `q+大写字母` 。

我们将上述并行的操作改为串行，假设已经录好其他操作只差一个 `j` 了，我们可以使用 `qA` 进行追加，然后添加 `j` 操作即可

好家伙，反斜杠居然有7个，而且 `()` 、 `{}` 需要转义，而 `[]` 不需要转义。正则表达式就够麻烦的了，还得记住vim与其他编辑器的不同，用一次人就麻了。

好在vim提供了 `very magic` 模式，即除了 `_` 、数字、字母之外的所有字符都具有特殊含义，这样我们就不用纠结哪些需要转义，哪些不需要了。可以在搜索的开头添加 `\v` 来启用这一模式，即我们可以输入 `\v#([0-9a-fA-F]{6}|[0-9a-fA-F]{3})`



字符	含义
\x	十六进制数
\X	非十六进制数
\d	数字
\D	非数字
\o	八进制数
\o	非八进制数
\w	包括字母、数字和 _
\W	不包括 字母、数字和 _
\h	包括 字母和 _
\H	不包括字母和 _
\l	小写字母
\L	非小写字母
\u	大写字母
\U	非大写字母

但是这个时候我们发现匹配的结果并不是我们想要的，这是因为在匹配模式中 `/` 是具有特殊意义的特殊字符，我们需要告诉vim将其解释为普通字符，这个时候可以使用 `\V` 来进入 `very nomagic` 模式，该模式与 `very magic` 相反，将所有字符作为普通字符来解释。

我们会返现它只匹配到了 `https:`，并且模式中的字符串也变成了 `https:`，后面从/开始截断了，这时候我们可以使用 `\/` 对 `//` 进行转换。同时 `\` 本身也作为特殊字符，我们也需要对其进行转义。即整个匹配应该输入 `https:\/\/www.baidu.com\/search?q=\\\\\/`