**第一部分Vim 基本使用**

**Vim初步**

a：当前字符的后一个位置插入

i：当前字符的前一个位置插入

o：当前编辑位置下面新起一行

A：在行最后位置插入

I：在行最前的位置插入

O：在当前编辑位置的上面新起一行

w：保存

wq：保存并退出(w and q)

**Visual(可视)模式**

**Visual模式一般用来块状选择文本**

Normal 模式下使用 v 进入 visual 选择

使用 V 选择行

使用ctrl + v 进行方块选择

**Vim插入模式小技巧**

**补充在一般终端下的快捷键**

ctrl + a 移动到开头使用 V 选择行

ctrl + e 移动到结尾

ctrl + b 往前(左)移动

ctrl + f 往后移动

**Normal/insert模式的切换**

**不方便按ESC前提下的替换方案**

insert -> normal 除了使用 ESC 之外还可以使用 Ctrl+c 或者 Ctrl+[

使用 V 选择行

normal -> insert 可以使用 gi 快速跳转到最后一次编辑的地方进入插入模式

**如何进行快速纠错**

**代码就像人生，总是经常出错，需要我们快速修正**

进入Vim之后首先使用进入insert模式

Ctrl+h删除上一个字符，ctrl+w删除上一个单词，ctrl+u删除当前行

**在单词之间飞舞**

w/W移动到下一个word/WORD开头。e/E下一个word/WORD尾

b/B回到上一个word/WORD开头，可以理解为backword

word指的是以非空白符分割的单词，WORD是以空白符分割的单词

常用w/b

**行间搜索**

f{char}可以移动到char字符上，t移动到char的前一个字符，落到的终点不一样

如果第一次没搜到，可以使用；或者，来继续搜索该行的下一个/上一个

f是从行首开始搜索，而将f改成F后是从行尾进行搜索

**vim水平移动**

0移动到行首的第一个字符，^移动到第一个非空白字符

$移动到行尾，g\_移动到行尾的非空白字符

0和$比较常用，且可以用组合键0+w($+b)来替代^(g\_)

**Vim垂直移动**

( ) 在句子间移动，可以使用 :help (来查看帮助

{ } 在段落之间移动

sentence 和 paragraph 的定义都可以在help命令中找到

还可以使用 easy-motion 插件

**Vim页面移动**

gg/G移动到文件的开头和结尾，可以使用 ctrl+o 快速返回

H/M/L跳转到屏幕的开头，中间，结尾（不常用）

ctrl+u. ctrl+f. 上下翻页。zz把屏幕置为中间

**vim 快速删除**

**如何快速删除一个字符或者单词呢？**

VIM在normal 模式下使用x删除一个字符

使用d(delelte)配合文本对象快速删除一个单词，可以用daw(delete aound word)或者dw

结合文本对象指的是(dtx)delete to x就是将xxx之前的东西都删除掉

 删除一行 dd

删除4个字符4x，前面加字母即是多次删除执行一个命令

**Vim快速修改**

**相比删除，更常用修改，一般是删除之后改成我们期望的文本**

r(replace)就是将当前字符（光标所在处），R是逐词语替换

c(change)可以和上面的d对应起来，都可以结合文本对象，删除指定范围后进入插入模式

s(substitute)是将当前字符删除然后进入插入模式，而S是将当前行删除再进入插入模式

**Vim 查询**

**查询单词也是常用操作的一种，注意其与行间搜索的差别**

使用 / 或者 ？进行前向或者反向搜索

使用n / N跳转到下一个或者上一个匹配

使用 \* 或者 # 进行当前单词的前向或者后向匹配

使用:noh 可以取消对搜索结果的高亮

**Vim替换命令**

**substitude命令允许我们查找并且替换掉文本，并且支持正则表达式**

:[range]s[ubstitute]/{pattern}/{string}/[flags]

range表示范围 比如：10，20表示10-20行，%表示全部

pattern是要替换的模式，string是替换后的文本

flags有几个常用的标志

g(global)表示全局范围内执行

c(confirm)表示确认，可以确认或者拒绝修改

n(number)报告匹配到的次数而不替换，可以用来查询匹配次数

**使用正则表达式的场景**

例子

比如对于文本

class Duck:

def \_\_init\_\_(self, name):

self.name = name;

def quack(self):

print("gua gua")

class Man:

def \_\_init\_\_(self, name):

self.name = name

def quack(self):

print("man gua gua“)

def do\_quack(ducker):

ducker.quack();

if \_\_name\_\_ == '\_\_main\_\_':

d = Duck('duck')

m = Man('man')

do\_quack(d)

do\_quack(m)

**如果使用命令:% s/quack/jiao/g 则会将存在有quack子串的文本全部替换为jiao，但是我们要的是将单词quack进行替换，这时候就要进行正则表达式匹配，即使用命令:% s/<\quack\>/jiao/g**

**Vim多文件操作**

**Buffer-什么是缓冲区？**

Vim打开一个文件后会加载文件内容到缓冲区

之后的修改都是针对内存中的缓冲区，并不会直接保存到文件

直到我们执行:w (write)的时候才会把修稿内容写入到文件里面区

**Buffer切换**

使用:ls会列举当前缓冲区，然后使用:b n跳转到第n个缓存区

:bpre :bnext :bfirst :blast

或者使用:b buffer\_name 加上tab补全来跳转

**Window窗口**

一个缓存区可以分割成多个窗口，每个窗口也可以打开不同缓冲区

<ctrl+w>s水平分割，<ctrl+w>v垂直分割。或者:sp和:vs

每个窗口可以继续被无限分割(看你屏幕是否足够大)

重排窗：可以查看h window-resize文档

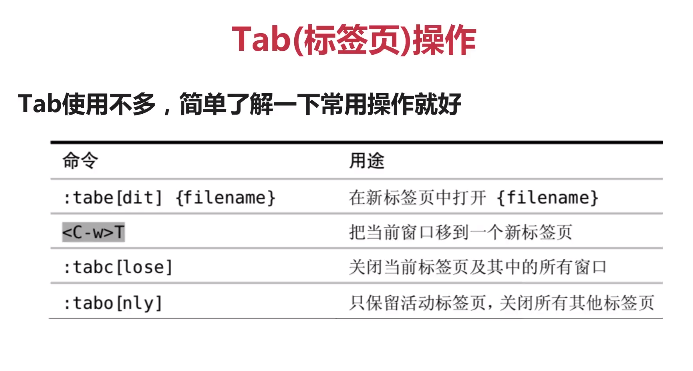
**Tab（标签页）将窗口分组**

Tab是可以容纳一系列窗口的容器(:h tabpage)

vim 的Tab和其他编辑器的不太一样，可以想象成Linux的虚拟桌面

比如一个Tab全用来编辑Python文件，一个Tab全是HTML文件

想比窗口，Tab一般用的比较少，Tab太多管理起来比较麻烦





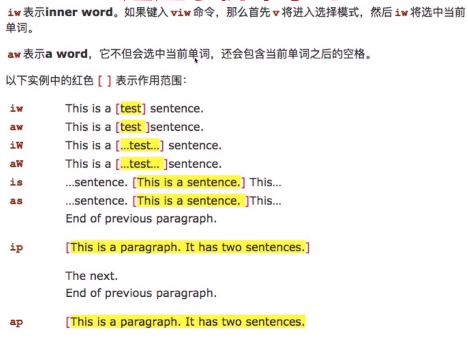
**文本对象(Text Object)**

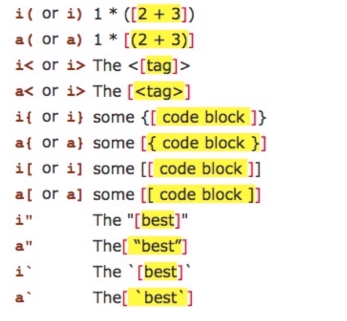
**三种文本对象单词w，句子s，段落p**

操作：[number]<command>[text object]

number表示次数，command是命令，d(elete)，c(hange)，y(ank)

例子：





**Vim复制粘贴与寄存器的使用**

**Vim Normal模式复制粘贴**

初学者会感觉Vim复制粘贴比较奇怪，先从normal模式学起

normal模式下复制站体分别使用y(ank)和p(ut)，剪切d和p

我们可以使用v(isual)命令选中所要复制的地方，然后使用p粘贴

配合文本对象:比如使用yiw复制一个单词，yy复制一行

**Vim Insert模式下的复制粘贴**

这个和其他的文本编辑器差不多，但是粘贴代码有个坑

很多人在vimrc中设置了autoindent，粘贴Python代码缩进错乱

这个时候需要使用:set paste和:set nopaste解决

**深入寄存器(register)**

Vim不适用单一剪切板进行剪切、复制与粘贴，而是多组寄存器

通过 “{register} 前缀可以指定寄存器，不指定默认用无名寄存器

比如使用 “ayiw复制一个单词到寄存器a中，“bdd 删除当前行到寄存器b中

Vim中 ““ 表示无名寄存器，缺省使用。”“p其实就等于p

**其他常见寄存器**

除了有名的寄存器a-z，Vim中还有一些其他常见寄存器

复制专用寄存器 “0 使用复制文本会同时被拷贝到复制寄存器0

系统剪切板 “+ 可以在复制前加上”+复制到系统剪切板

其他一些寄存器比如 “% 当前文件名，”.上次插入的文本

**强大的Vim宏(macro)**

**什么是Vim宏**

宏可以看成是一系列命令的集合

我们可以使用宏录制一系列操作，然后用于回放

宏可以非常方便地把一系列命令用在多行文本上

**如何使用宏**

宏的使用分为录制和回放，使用q来录制，同时也是q结束录制

使用q{register}选择要保存的寄存器，把录制的命令保存其中

使用@{register}回访寄存器中保存的一系列命令

**Vim 补全大法**

**常见的三种补全类型**

使用ctrl+n和ctrl+p补全单词

使用ctrl+x ctrl+f补全文件名

使用ctrl+x ctrl+o补全代码，需要开启文件类型检查，安装插件

插入当前文件名r! echo %

插入当前文件路径r! echo %:p

批量缩进

1. 使用V进入 visual line 模式，选中要进行缩进的行，然后按住ctrl + < 实现向左缩进，同理 ctrl + > 实现向右缩进。

**第二部分 Vim配置**

**Vim配置初步**

**首先是常用的Vim设置**

我们可以把常用的设置写到~/.vimrc里面避免每次打开vim重新设置

比如设置行号set nu; 设置主题colorscheme hybrid

Vim里有非常多这种配置，你可能需要参考下别人的配置

**Vim中的映射比较复杂，源于vim有多种模式**

设置一下leader键let mapleader = “,”,常用的是都好或空格

比如用inoremap<leader>w <Esc>:w<cr> 在插入模式保存

Vim中的映射概念稍微复杂，但是非常强大，下一章单独讲

**Vim 脚本**

Vim脚本对于Vim高级玩家来说可以实现强大的vim插件

初学者知道这个概念就好，Vim脚本是一种简单的简本语言

可以通过vimscript实现更多vim的控制，开发自己的插件

可以使用:h list查找所有设置的选项

**Vim中的映射**

**基本映射**

基本映射指的是normal模式下的映射

使用map就可以实现映射。比如:map-x 然后按-就会删除字符

:map <space> viw 告诉vim按下空格的时候选中整个单词

:map <c-d> dd 可以使用ctrl+d 执行dd删除一行

**模式映射**

Vim常用模式normal/visual/insert都可以定义映射

用nmap/vmap/imap定义映射只在normal/visual/insert分别有效

**递归与非递归映射**

例子:nmap – dd

:nmap \ -

当按下\时，Vim会解释其为-。我们又映射了-！Vim会继续解析-为dd，即它会删除整行。（类似递归）

\*map系列命令有递归的风险，如果安装一个插件，插件映射了同一个按键的不同行为，有冲突就会有一个失败

想要保证插件映射没有冲突会非常痛苦

使用\*map对应的nnoremap/vnoremap/inoremap

任何时候你都应该使用非递归映射，拯救自己和插件作者

**第三部分 Vim插件**