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

Tab(标签页)操作

Tab使用不多，简单了解一下常用操作就好

命令	用途
:tabe[dit] {filename}	在新标签页中打开 {filename}
<C-w>T	把当前窗口移到一个新标签页
:tabc[lose]	关闭当前标签页及其中的所有窗口
:tabo[nly]	只保留活动标签页，关闭所有其他标签页

Tab(标签页)切换操作

如何切换不同的标签页，一般建立两个就好，太多不好操作

Ex 命令	普通模式命令	用途
:tabn[ext] {N}	{N}gt	切换到编号为 {N} 的标签页
:tabn[ext]	gt	切换到下一标签页
:tabp[revious]	gT	切换到上一标签页

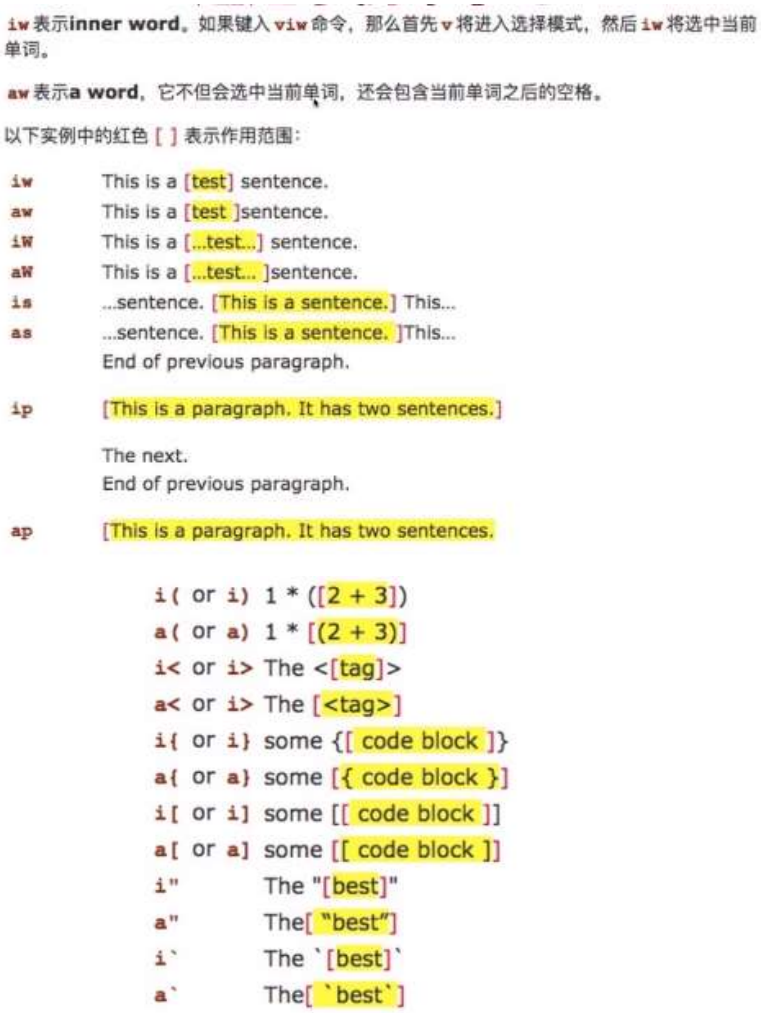
文本对象(Text Object)

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