A. 什么是VIM

1. 文本编辑器

类Unix系统通常自带的编辑器

- 在命令行中，通过“vim + 文件路径”的指令可以使用VIM打开文本文件
- 初次使用VIM的用户想必都遇到过“无法退出”的问题
- VIM以其学习代价高、学习曲线陡，但熟练之后能够提升效率著称

2. 输入方式

VIM的主要魅力并不在软件本身，而是其定义的输入方式

- VIM软件是命令行软件，综合下来的使用体验不一定比现代的图形化编辑器和IDE强
- 但是VIM所定义的输入方式可以应用在其他软件中

许多编辑软件可以实现VIM的输入方式

- 图形化IDE、代码编辑器可通过安装插件，实现VIM逻辑
- Overleaf、Jupyter等在线编辑器带有VIM映射功能

B. 为什么选择VIM

1. 更符合人体工学

传统输入方式

- 键盘+鼠标，右手需要反复在键盘与鼠标之间移动
- 靠键盘方向键移动光标，难以处理大幅度的移动需求，同时方向键的位置也比较偏

VIM输入方式

- 相关键位集中在键盘字母区，常用的指令尽量设计在标准指法附近，双手无需移动即可满足所有输入需求

2. 更高效的定位与修改方式

定位在编程中实际上比输入占有更大的比重

- 输入工作的逻辑相对简单
- 定位与修改的逻辑复杂且繁多
  - 需要定位到某一个函数或者某一行的位置，又可能需要定位到某一个字符或者某一个词的位置
  - 需要修改一个词中某个字符、修改整个词、修改括号中的内容

传统定位方式

- 使用鼠标定位，然而鼠标的移动精度有限，移动到特定的位置需要“对准”的时间
- 定位与修改在人脑中实际上可以构成一组清晰的逻辑描述，然而使用鼠标去实现这些逻辑并不直观
  - 在编程中，修改一个词和修改一个字符的频率可能几乎相同，然而使用鼠标框选一个词所用的时间将长于选中一个字符

VIM定位与修改方式

- 定位与修改的逻辑可以直接转化为VIM快捷键组合，实现更直观的操作
- 常用的定位、修改操作均能用较短的快捷键组合实现，提升操作效率

C. 如何使用VIM

1. 模式与切换

普通模式

- VIM的默认模式，输入绝大多数VIM指令的模式
- 实现大部分定位和修改逻辑的模式
  - VIM“哲学”的最佳体现，定位与修改比起输入占有更大的比重
- 光标将呈现特殊的状态，能够覆盖一个字符

输入模式

- 在普通模式下输入一些指令进入，具体的指令及其区别后续介绍
- 非VIM用户比较熟悉的模式，光标也是正常的状态
- 在此模式下，按ESC返回普通模式

底线命令模式

较少使用的模式，大多情况用于实现文件级别的指令

- 在IDE、编辑器环境中，使用软件的文件操作即可，VIM的这些指令用处不大

在命令模式下输入冒号进入，继续输入实现文件操作，按ESC返回命令模式

- :q退出，:w保存，:wq退出并保存

2. 定位指令

页定位

- 实现大跨度定位/滚动，或是对屏幕显示的部分进行调整

非定量页定位：适用于对文件内容的快速检索

- Ctrl+f/b (Forward, Backward) 使光标向后/前移动一页
- Ctrl+d/u 使光标向后/前移动半页
- 操作系统通常自带这些快捷键，并不是不可替代

定量页定位

- gg使光标移动到文件第一行，G (Shift+g) 使光标移动到最后一行

页面位置调整

- z+z/b/t (bottom, top) 平移页面位置，使得当前光标所在行在屏幕中间/最底/最上
- 编程中比较常用的指令。平移到中间的操作最常用，所以设置为z键连按两下

行定位

- 将光标跳转到需要的行

基本的行移动

- j/k，将光标向上/向下移动一行
- 类似方向键中的上和下，非常常用所以放在右手标准键位处

精准行定位

- 数字+G / 数字+gg，跳转到数字标示的行
- Shift+h/m/l (H/M/L) 跳转到屏幕上的第一行/中间行/最后一行
- 适合可见部分的代码，实现快速跳转

段落间移动

- Shift+[/]，即{/}，跳转到上一个/下一个段落
- 段落的定义是没有空行隔开的多行字符
- 在代码段落间跳转，前提是你的编程习惯良好，会给不同功能性的代码分段

返回功能

- `` 或 '' 返回本次跳转前光标所在的行

词定位

在词之间跳转

- 词的定义是中间无分隔符的连续字符串

基础指令

- w将光标移动到下一个单词的开头
- b将光标移动到上一个单词的开头

- e将光标移动到当前所在单词的末尾，如果已经在则移动到下一个单词的末尾

大写指令

- w, b, e均有对应的大写指令，效果类似
- 唯一区别是对词的定义不同，中间只要无空格的字符串均看做一个词

词查找

- 按下/后进入词查找模式
- 输入词汇后按下Enter，跳转到光标后首次出现词的位置
- 按n (next) 跳转到下一个出现该词的地方
- 按shift+n (N) 跳转到上一个出现该词的地方

字符定位

- 跳转到一行的某一个字符处

基本的字符移动

- h/l 将光标向左/右移动一个字符
- 类似方向键的左和右，非常常用所以也放在右手标准键位处

查找式字符定位

- f (find) +字符，跳转到当前行光标右侧第一个字符位置
- Shift+f (F) + 字符，跳转到当前光标左侧向左第一个字符位置
- t (till) + 字符，跳转到跳转到当前行光标右侧第一个字符位置的左一位
- Shift + t (T) + 字符，跳转到当前光标左侧向左第一个字符位置的右一位

行间字符位置定位

- 0移动到当前行的一个字符处
- Shift+6 (^) 移动到当前行的第一个非空字符处
- Shift+4 ($) 移动到当前行的最后一个字符处
- 作为理应常用的指令，都需要修饰键Shift似乎有点不方便，但后续我们会讲到，大部分情况下其实不需要用到这几条指令

定位指令的应用

- 仅凭上述定位指令，我们实际上已经可以实现在代码文件中的高效、精准定位

场景示例

- 137gg zz f)， 跳转到某一行，将此行移到屏幕中央，定位到希望输入的位置
- gg跳转到文件第一行，添加import package之后，``回到刚才的行

- 将光标移动到我们希望的位置之后，接下来的问题就是如何进行输入

3. 插入指令

插入指令两方面的功能

- 一是进入输入模式，让我们可以输入内容
- 二是快速将光标移动到希望开始输入的位置

基础插入指令

基础插入指令插入的位置和当前光标所在字符对应位置有关

- i (input) 在光标所在字符前插入
- a (append) 在光标所在字符后插入
- s 删除光标所在字符并进入输入模式

- 可以配合定位指令，首先将光标定位到指定位置

行内插入指令

- 进入输入模式的同时，光标移动到常用位置
  - Shift+i (I) 在当前行的开头插入
  - Shift+a (A) 在当前行的末尾插入
  - Shift+s (S) 将当前行清除后进入输入模式
- 对应编程中常见的场景，只要进行行定位即可

行间插入指令

- 对应常用的另起一行输入的场景
  - o 在当前行下新起一行，并开始输入
  - Shift+o (O) 在当前行上新起一行，并开始输入
  - 省去了将光标移动到行尾，再按Enter的麻烦

插入指令的应用

- 插入指令移动光标的功能有限，必然需要配合定位指令使用
- 场景示例
  - 137gg zz A，跳转到某一行，将此行移动到屏幕中央，从句末开始输入

4. 选中操作

“选中”实际上是一种模式

- 类似按住Shift之后框选一部分文字
- 官方的称呼叫Visual Mode，因此大部分操作和v键有关

字符级别选中 (Visual)

- 按下v即进入字符选中模式，按ESC或者再次按v退出
- 此时选中的部分为，从按下v时光标所在字符开始，到目前光标所在字符为止
- 可以配合定位指令快速划定选中范围
  - 首先定位到范围开头所在字符，按下v，再定位到范围末尾所在字符

行级别选中 (Visual Line)

- 按下Shift+v (V) 即进入行选中模式，同样按ESC或V退出
- 此时选中的部分为，从按下V时光标所在行开始，到目前光标所在行为止的所有行
  - 常用于整段的选择
- 同样可以配合定位指令快速划定选中范围

列级别选中 (Visual Block)

- 按下Ctrl+v 即进入列选择模式，按ESC或Ctrl+v退出
- 此时选中的部分为，从按下Ctrl+v时光标所在行和列开始，到目前光标所在行和列为止的一个“矩形区域”
  - 常用于对整齐划一的多行内容进行统一修改
- 此时按下I/A进入多行同时编辑模式
  - 同时在多行的对应位置插入
    就像上小学时老师让抄一句话N遍，把N支笔并排握着，写一遍即可完成任务

指定范围选中

- 使用vi (in) + 指定范围的字符实现
  - 光标移动到希望指定的范围内的任意一个字符内（包括范围的头尾）
  - 输入指令直接完成框选
- 支持选中括号内所有内容
  - 输入vi + (/)/[/]/{/}/</>
- 支持选中引号内内容

- 输入vi + ‘/“

5. 删除操作

删除光标所在位置的内容

删除光标所在字符

- 插入指令中的s实际上就是删除光标所在字符，并进入输入模式
- 如果只是想删除字符，不进入输入模式，按x

删除光标所在行

- 同理，插入指令中的S可以做到，但顺带进入了插入模式
- 如果只是想删除行，不进入输入模式，按dd

删除当前行光标后所有内容

- Shift+d (D)即可，光标所在的那一个字符也会被删除

删除指定范围的内容

删除选中的内容

- 在“选中操作”中使用任意方式选中一部分内容后，按下d/x删除选中的内容

删除括号/引号内的内容

- 和选中操作类似，使用 di+字符 完成

特殊的快捷键

- XML语言和HTML语言以类似 <tab> contents </tab> 的格式组织，使用dit指令可快速删除 <tab> 内的内容（其他标签名称也均可）
- Shift + j (J) 将当前行和下一行拼接，相当于将下一行行首的换行符和空白字符删除

删除与定位指令结合

相比选中后删除更便捷的方式

- 要利用定位指令实现更加精准的范围删除，当然我们可以先用v进入选中模式，使用定位指令框选范围后删除。但一些定位指令是可以直接和删除指令结合的。

删除直到某个字符

- df + 字符 或 dt + 字符，效果类似 vf + 字符 + d 或 vt + 字符 + d

删除多行

- dj 删除当前行以及下一行，dk删除当前行以及上一行

删除后进入输入模式

将删除指令中的d换成c即可，适用于上述所有删除指令

- 例如，删除小括号内内容后进入输入模式的指令是ci(或ci)

6. 剪贴板与寄存器

剪贴板

- VIM存在内置的剪贴板
- 使用复制指令能够将指定内容复制到剪贴板中，使用粘贴指令将剪贴板中的内容取出
- 在前述删除操作中，任何删除的内容都会自动进入剪贴板

复制操作

复制选中的内容

- 在选中模式下，按 y (yank) 即可将选中的内容拷贝到剪贴板中

复制整行内容

- 按 yy 即可复制光标所在行

复制也是一种模式

- 在非选中模式下，按下y后实际上我们就进入了“复制待命”模式

- 其操作逻辑和按下v后进入的选中模式非常相似，因此可以直接使用 yt yi yf 等方式复制一定范围内的内容
- 需要留意的是在这个模式下，如果发生了行跳转，其表现和选中模式是有差异的
  - 按下v后的行跳转依然是字符级别的框选
  - 按下y后的行跳转会直接复制跨越过的整个行

粘贴操作

粘贴剪贴板中的内容

- 按下p (put? paste?) 将剪贴板中的内容粘贴到光标所在处之后
- 按下Shift + p (P) 将剪贴板中的内容粘贴到光标所在处之前

粘贴并覆盖选中内容

- 在选中模式下，按p，相当于将选中的内容删除并粘贴剪贴板的内容

命名寄存器

剪贴板的覆盖问题

- 既然复制操作和删除操作都会将内容拷贝到剪贴板中，默认的剪贴板将被反复覆盖
- 如果先前复制了一段需要反复复用的代码到剪贴板，在编程过程中难免会使用删除操作，如此就把剪贴板内容覆盖了

将剪切内容存到命名寄存器而非默认剪贴板中

- 在复制/删除操作前，指定寄存器，内容就会被拷贝到寄存器中
- 除非再次将其他内容拷贝到相同的寄存器中，否则寄存器中的内容会一直保留

寄存器指定指令

- “+字母，字母就成为了寄存器的代号
- 在任意复制/删除指令前使用小写字母指定寄存器，内容就会覆盖指定的寄存器
- 使用大写字母指定寄存器，内容就会添加到指定的寄存器现有内容之后

读取寄存器

- 在任意粘贴操作之前再次输入”+字母，即可指定寄存器
- 粘贴内容将是从寄存器中读取的

命名寄存器的应用

- 寄存一段经常复用的代码段备用
- 调换两个代码段的顺序
  不过，由于粘贴覆盖操作会更新默认剪贴板，所以调换顺序并不一定要用到命名寄存器

一些特殊的寄存器

默认寄存器

- 就是所谓的默认剪贴板，而且也有代号”，可以通过””调用
- 即使在复制操作前指定了命名寄存器，这个寄存器也会被覆写

数字寄存器

- 代号0-9，相当于剪贴板记录
- 实际上的运作模式稍微复杂一些，有兴趣可以自行查阅资料

系统寄存器

- 代号+，直接读取Windows/Linux/Mac系统的Ctrl+C复制记录

- :registers指令可以查看所有寄存器的内容

7. 书签系统

标记位置实现快速跳转

- 顾名思义，书签就是在代码中标记行列位置，方便之后快速跳转到对应位置

标记书签

- m (mark) +字母，将光标所在位置记录到字母标记的书签
- 小写字母仅在本文件中有效，大写字母在整个工程中有效

跳转书签

- `+字母，跳转到标记的精准位置
- ‘+字母，跳转到标记所在行的开头

显示所有书签

- :marks指令可以显示所有的书签及其位置

书签系统的应用

- 在代码中的关键位置打上标记，方便开发过程中跳转

8. 宏命令

重复执行多次任务，无需反复输入指令

- 一些指令需要重复多次，而手动重复键入过于繁琐
- VIM自带相当强大的宏命令功能，从简单但机械重复的操作，到复杂的指令集，都可以胜任

重复执行指令

可以在几乎所有指令前添加数字实现重复执行

- 数字+指令，执行指令N次
- 指令可以是单字符，前提是这个指令键入单字符即完成
  - 例如，10j 代表向下移动10行
- 指令也可以是多字符，前提是这个指令是多字符组成的
  - 例如，2dti 相当于执行“删除直到i字符”两次

在能够快速预估指令执行次数的情况下很高效

记录宏命令

- 记录从开始到结束键入的所有指令，方便后续再次自动执行

开始宏记录

- 按下q+字母，字母表示存储宏的寄存器
  当然，这个寄存器和之前的剪贴寄存器不冲突
- VIM状态栏显示”recording @<字母>”
- 开始之后的指令都会按照键入顺序，记录在寄存器中
  当然，键入的速度是无所谓的

停止宏记录

- 再次按下q，停止指令记录

调用宏命令

单次调用

- 键入@+字母，VIM自动按顺序执行对应寄存器存储的指令

多次调用

- 配合“重复执行指令”的描述，键入数字+@+字母即可执行宏命令N次

宏命令的应用

宏记录可以录制几乎一切指令，因此通过设计，可以实现相当复杂的自动化

- 例如，批量修改单词

D. 回顾和总结

1.