Vim

The Life Changing Editor

主要内容

- 掌握 **vim** 编辑器的使用方法
- 熟悉 **vim** 编辑器的各种功能
- 能熟练使用 **vim** 编辑文件

切记要在 *使用中学习*，而不是在记忆中学习。

Vim 介绍

- 全屏幕文本编辑器

  - 世界上只有三种编辑器，EMACS、VIM和其它
  - **VIM is the God of editors, EMACS is God's editor**
  - **EMACS is actually an OS which pretends to be an editor**

  VIM >= SUM(现代编辑器)

Vim 特性

- 语法高亮
- 自动对齐
- **代码折叠**
- **自动补全**
- 显示行号

重定义Tab

十六进制编辑

列编辑模式

快速注释

...

Visual Studio Profession 2012 : 11645元
UtralEdit : 420元
Source Insight : 2500元
...

Normal 模式

- 无论什么时候，不管用户处于何种模式，只要按一下 **Esc 键**，即可使 **vim** 进入 **Normal 模式**。
- 在shell中输入 **vim** 启动编辑器时，即进入该模式。
- 在该模式下，用户可以输入各种合法的 **vim** 命令，用于管理自己的文档。此时从键盘上输入的任何字符都被当做编辑命令来解释。
- 若输入的字符是合法的 **vim** 命令，则 **vim** 在接受用户命令之后完成相应的动作。但需注意的是，所输入的命令并不在屏幕上显示出来。若输入的字符不是 vim 的合法命令，**vim** 会响铃报警。

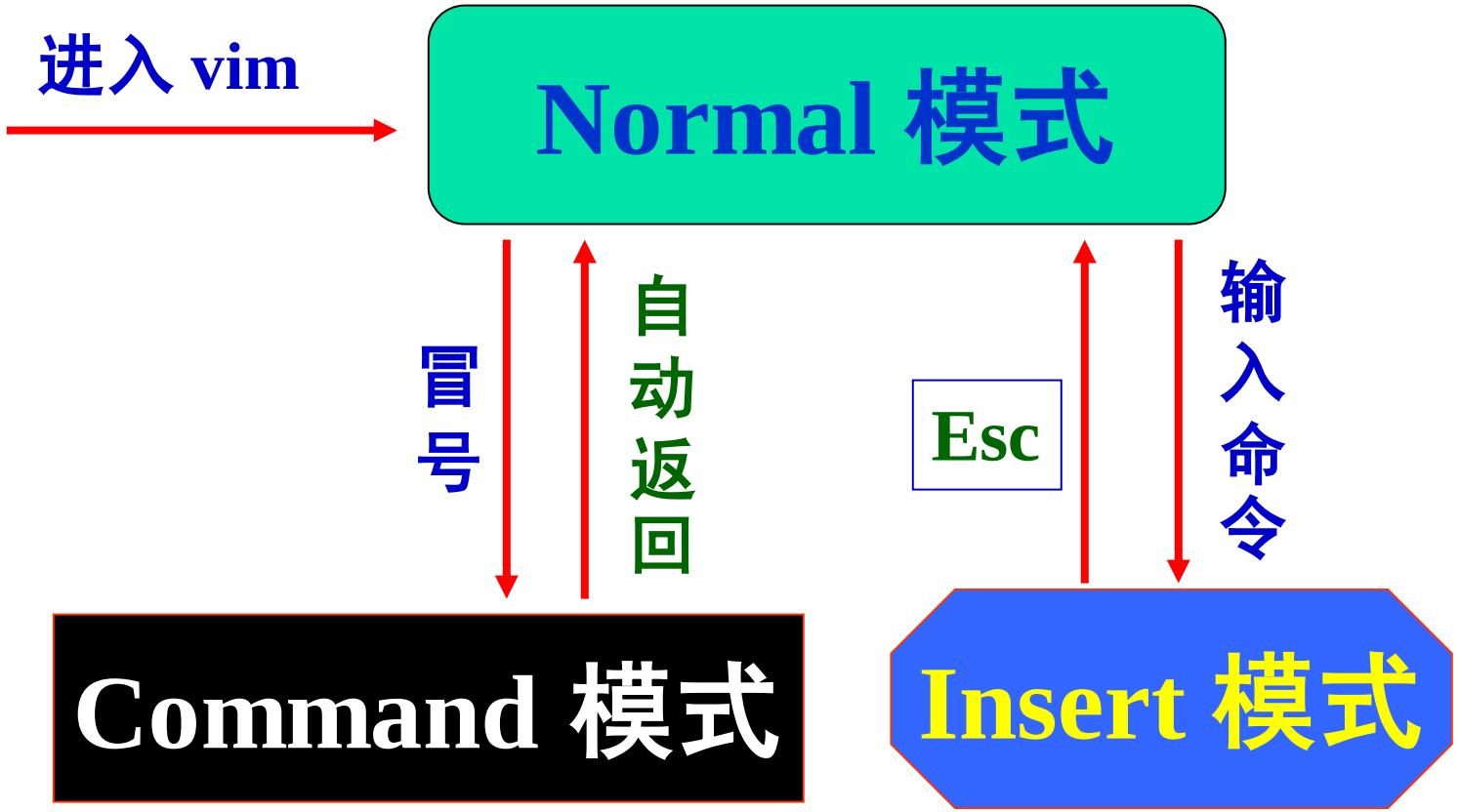
Insert 模式

- 在 **Normal** 模式下输入**插入命令 i**、附加命令 **a** 、打开命令 **o**、修改命令 **c**、取代命令 **r** 或替换命令 **s** 等都可以进入 Insert 模式。

- 在该模式下，用户输入的任何字符都被vim当做文件内容保存起来，并将其显示在屏幕上。在文本输入过程中，若想回到Normal模式下，按 **Esc** 键即可。

Command 模式

- Normal 模式下，用户按冒号 “:” 即可进入 Command 模式，此时 vim 会在显示窗口的最后一行 (屏幕的最后一行) 显示一个 “:” 作为 Command 模式的提示符，等待输入命令。
- 多数文件管理都是在此模式下执行的 (如保存文件等)
- Command 模式中所有的命令都必须按 **<回车>** 后执行
- 命令执行完后，vim 自动回到 Normal 模式。
- 若在 Command 模式下输入命令过程中改变了主意，可按 Esc键，或用退格键将输入的命令全部删除之后，再按一下退格键，即可使 vi 回到 Normal 模式下。


进入 vim
Normal 模式
冒号
自动返回
Esc
输入命令
Command 模式
Insert 模式

vim 的进入

◆ vim 的进入：在命令行提示符后键入 vim 和想要编辑或新建的文件名，便可进入 vim

```
vim example.c
```

◆ 通常在 Linux 下为 vim 定义了别名 vi，所以只需输入 vi 即可

```
vi example.c
```

vim 的进入

- 如果所编辑的文件已经存在，则在屏幕上显示出该文件的内容，并且光标停在第一行的首位，在状态行显示出该文件的文件名、行数和字符数

- 进入 vim 时，用户不仅可以指定一个待编辑的文件名，而且还有许多附加操作：

  - 如果希望在进入 vim 之后，光标处于文件中特定的某行上，可在 vim 后加选项 **+n**，其中 n 为指定的行数
  - 如果希望在进入 vim 之后光标处于文件最末行，则只需把命令中附加项 “**+**” 后面的数字 **n** 省略即可

vim 的进入

◆ 在进入 vim 时，除了可以指定一个光标起始行号之外，还可以在命令中指定一个模式串，此时在进入 vim后，光标就处于文件中第一个与指定模式串相匹配的那行上。

◆ vim 可以同时编辑多个文件

```
vi +3 example.c
```

```
vi + example.c
```

```
vi +/int example.c
```

```
vi example1.c  example2.c
```

vim 的进入

命令	功能
vi *filename*	从第一行开始编辑 filename 文件
vi +*n filename*	从第 n 行开始编辑 filename 文件
vi + *filename*	从最后一行开始编辑 filename 文件
vi +/*pattern filename*	从包含 pattern 的第一行开始编辑 filename 文件
vi –r *filename*	在系统崩溃之后恢复 filename 文件
vi –R *filename*	以只读方式编辑 filename 文件

vim 的退出

- 需要保存文件并退出 vim 时，可以使用以下几种方法

  - 连按两次大写字母 **Z**，若文件被修改过，则保存后退出。
  - **:w** 保存当前编辑文件，但并不退出。
  - **:q** 系统退出 vim 返回到 shell。
  - **:wq** 表示存盘并退出。
  - **:q!** 放弃所作修改而直接退到 shell 下。
  - **:x** 同 Normal 模式下的 **ZZ** 命令功能相同。
  - **:w newfile** 存为另一个文件。

光标移动操作

- 全屏幕文本编辑器中，光标的移动操作无疑是最经常使用的操作了。用户只有熟练地使用移动光标的这些命令，才能迅速准确地到达所期望的位置处进行编辑。

- vim 中的光标移动既可以在 Normal 模式下，也可以在 Insert 模式下，但主要是在 Normal 模式下。

- 在 Insert 模式下：可直接使用键盘上的四个方向键移动光标。

Normal 模式光标移动

- 在 Normal 模式下的光标移动方式

光标左移一格	向左方向键，h，BACKSPACE
光标右移一格	向右方向键，l，SPACE
光标上移一行， 所在的列不变	向上方向键，k，Ctrl+p
光标下移一行， 所在的列不变	向下方向键，j，Ctrl+n

Normal 模式光标移动

^	光标移到所在行的开始
0	同 ^
$	光标移到所在行的末尾 (前面可加重复因子)
nG	光标移到第 n 行，若不指定 n，则移到最后一行
:$	光标移到最后一行
-	光标移到上面一行的开始 (前面可加重复因子)
+	光标移到下面一行的开始 (前面可加重复因子)
Enter	同 +

Ctrl+g：显示光标所在行位置以及文件状态信息

Normal 模式光标移动

- 按字 (word) 移动光标 (Normal 模式下)

w	光标右移一个单词
W	右移一个以空格作为分隔符的单词
b	光标左移一个单词
B	左移一个以空格作为分隔符的单词
e	光标右移到一个单词的结尾
E	右移一个以空格作为分隔符的单词结尾

注：这些命令前都可以加 重复因子。

Normal 模式光标移动

- 按句 (sentence) 移动光标 (Normal 模式下)

)	光标移到下一句 (sentence) 的开始
(	光标移到当前句 (sentence) 的开始

- 按段 (paragraph) 移动光标 (Normal 模式下)

}	光标移到下一段落(paragraph)的开始
{	光标移到当前段落(paragraph)的开始

- 按节 (section) 移动光标 (Normal 模式下)

[[	光标移到下一节(section)的开始
]]	光标移到本节(section)的开始

注：这些命令前都可以加 重复因子。

光标在屏幕移动

- 光标在屏幕移动 (Normal 模式下)

H	**光标移到当前屏幕的顶部**
M	**光标移到当前屏幕的中部**
L	**光标移到当前屏幕的底部**

其它方式

- 光标移动的其它方法 (Normal 模式下)

fc	将光标移到当前行的下一个字符 c 上
Fc	将光标移到当前行的前一个字符 c 上
tc	将光标移到当前行的下一个字符 c 之前（左边）
Tc	将光标移到当前行的前一个字符 c 之后（右边）
;	重复最近一次的 f、F、t、T 命令

屏幕滚动

- 屏幕滚动(Normal 模式下)

Ctrl+d	向前（下）移动半个屏幕
Ctrl+u	向后（上）移动半个屏幕
Ctrl+f	向前（下）移动一个屏幕
Ctrl+b	向后（上）移动一个屏幕

文本插入

- 在 Normal 模式下用户输入的任何字符都被 vim 当作命令加以解释执行，如果用户要将输入的字符当作是文本内容时，则应将 vim 的工作模式从 Norma l模式切换到 Insert 模式。

- 插入 (insert) 命令

i	从光标所在位置前开始插入文本，插入过程中可以使用 BACKSPACE 键删除错误的输入
I	从当前行的行首前开始插入文本

文本插入

- 附加 (append) 命令

a	在光标当前所在位置之后追加新文本
A	从光标所在行的行尾开始插入新文本

- 新开行 (open) 命令

o	在光标所在行的下面新开一行，并将光标置于该行的行首，等待输入文本
O	在光标所在行的上面插入一行，并将光标置于该行的行首，等待输入文本

文本删除

- 在 Insert 模式下，可以用 BACKSPACE 或 Delete 键将输错或不需要的文本删除，但此时有一个限制就是只能删除本行的内容，无法删除其它行的内容。

- 在 Normal 模式下，vim 提供了许多删除命令。这些命令大多是以 d 开头的。

◆ 删除单个字符（删除少量字符的快捷方法）

x	删除光标处的字符。若在 **x** 之前加上一个数字 **n**，则删除从光标所在位置开始 向右的 **n** 个字符。
X	删除光标前的那个字符。若在 **X** 之前加数字 **n**，则删除从光标前那个字符开始 向左的 **n** 个字符。

文本删除

- 删除多个字符

dd	删除光标所在的整行。在 **dd** 前可加上一个数字 **n**，表示删除当前行及其后 **n-1** 行的内容。
D, d$	删除从光标所在处开始到行尾的内容
d0	删除从光标前一个字符开始到行首的内容。
dw	删除一个单词。若光标处在某个词的中间，则从光标所在位置开始删至词尾。可在 **dw** 之前加一个数字 **n**，表示删除 **n** 个指定的单词。

类似的命令还有：**dH**、**dM**、**dL** ...

取消上一命令（Undo）

- 取消上一命令，也称复原命令，是非常有用的命令，它可以取消前一次的误操作或不合适的操作对文件造成的影响，使之回复到这种误操作或不合适操作被执行之前的状态。

u	取消前一次的误操作，可连续使用，以取消更前的误操作。
U	将当前行恢复到该行的原始状态。如果用 U 命令后再按 U 键时，表示取消刚才 U 命令执行的操作，也就是又恢复到第一次使用 U 命令之前的状态，结果是什么都没做。
Ctrl＋r	撤消以前的撤消命令，恢复以前的操作结果。

文本恢复

- 寄存器：vim 内部有 9 个用于维护删除操作的寄存器，分别用数字 1，2，...，9 表示，它们分别保存最近用 **dd** 删除的内容。这些寄存器组成一个队列，采“先进先出”的模式存放，例如最近一次用**dd**删除的内容被放到寄存器 1 中；当下次用 **dd** 删除文本内容时，vim 将把寄存器 1 的内容转存到寄存器 2 中，而寄存器 1中存放最近一次删除的内容。以此类推，vim保存最近九次用**dd** 删除的内容，而前面用 **dd** 删除的内容则被抛弃。

- 恢复命令用 **"np**，其中 n 为寄存器号。

文本内容的修改

- 文本内容的修改是指在编辑过程中，可以对文本中的某些字符，某些行进行修改，即用新输入的文本代替需要修改的老文本，它等于先用删除命令删除需要修改的内容，然后再利用插入命令插入新的内容。所以在使用修改命令后，vim 进入到 Insert 模式，当输入完新的内容后，再按 Esc 键回到 Normal 模式。

ctc	修改到位于当前行的下一个字符*c*
C, c$	修改到行尾
cc	修改当前行

文本内容的修改

- 在命令 **C**（大写）之前可以加上数字 **n**，表示修改指定行数的内容。例如，**3C** 就表示把光标所在位置的字符之后（注意不是整行）直到下面两个整行的内容删除，然后由随后输入的内容代替。

- **cc** 之前也可以加上数字 **n**，表示要从光标当前行算起修改 **n** 行的内容。例如，**5cc** 表示先删除光标所在行及其下面的 **4** 行，然后输入要修改的内容。

复制与粘贴

- 复制与粘贴 (Normal模式下)

nyy，nY	复制当前行开始的 n 行
y0	复制至行首，不含光标所在处字符
y$	复制至行尾，含光标所在处字符
nyw	复制 n 个字（word）
yG	复制至文件尾，包含当前行
y1G	复制至文件首，包含当前行
p	粘贴至光标后，若整行复制则贴在当前行下面
P	粘贴至光标前，若整行复制则贴在当前行上面

文本的替换

- 文本的替换即是用新输入的文本代替原已有的文本。它同文本修改一样，也是先执行删除操作，再执行插入操作。

  - 取代命令：**r** 和 **R**

r	用随后输入的一个字符代替当前光标处的那个字符。替换执行完后自动回到 Normal 模式
R	用随后输入的文本取代从当前光标处及其后面的若干字符，每输入一个字符就取代原有的一个字符，直到按 **Esc** 键 结束这次取代，返回 Normal 模式

文本的替换

- 字 (word) 替换： **cw**
  - 如果我们只希望将某个字 (word) 用其他文本串替换，则可用 **cw** 命令。
  - 命令 **cw** 所替换的是一个狭义的字。
  - 输入 **cw** 命令后，vim 将把光标所在位置至光标所在的那个字的结尾删除，然后用户可输入任何替换文本内容。输入完成之后按 Esc键，结束替换，返回 Normal 模式。

文本行的合并

- 文本行的合并
  - vim 提供了将文本中的某些行进行合并的命令。该命令用 **J** (大写) 表示，其功能是把光标所在行与下面一行合并为一行。
  - 如果在 **J** 命令之前加一个数字 **n**，则表示把光标当前行及其后面的 **n-1** 行 (共 **n** 行) 合并为一行。

编辑多个文件

- vim 可以同时打开 (编辑) 多个文件，可用命令 **:n** 和 **:N**，在不同文件之间进行切换

```
vi file1 file2 file3
```

- **:split file** 切分出一个新窗口，用来打开指定的文件。若省略文件名，则打开当前文件，可用于同时观察文件的不同部分。若要在这两个 (上下) 窗口间进行切换，可用 **Ctrl+w Ctrl+w** (连按两次)。
- **:vsplit file** 作用同上，只是分成左右并列的两个窗口

搜索字符串

- 搜索字符串

/rexp 回车	向前（下）搜索 **rexp**
?rexp 回车	向后（上）搜索 **rexp**
n	重复最近一次的搜索
N	重复最近一次的搜索，但搜索方向相反
/回车	向前重复最近一次的搜索
?回车	向后重复最近一次的搜索

表中 **rexp** 是正则表达式，可以是单个字符串

如果查找已经到达文件末尾 (开头)，查找会自动从文件头部 (末尾) 继续查找。

替换字符串

- 替换字符串 (Command 模式下)

:s/old/new :ns/old/new	用字符串 new 替换当前行内头一个字符串 old（可指定行号 n ）
:s/old/new/g :ns/old/new/g	用字符串 new 替换当前行内所有字符串 old（可指定行号 n ）
:m,ns/old/new/g	用字符串 new 替换第 m 行到第 n 行内所有的字符串 old（“.”表示当前行，“$” 表示最后一行）
:%s/old/new/g	将文件内所有的字符串 old 替换为新的字符串 new
:%s/old/new/gc	进行替换时要求用户确认每个替换需添加 c 选项

配对括号的查找

- 配对括号的查找 (Normal 模式下)
  - 在程序调试时，用来查找不配对的括号是很有用的。
  - 利用 % 可以查找配对的括号：)、]、}。
  - 把光标放在任何一个 (、[或 { 处， 接着按 % 字符，此时光标的位置应当是在配对的括号处， 再次按 % 就可以跳回配对的第一个括号处。

插入文件

- 在光标所在位置处插入另外的文件的内容
  (Command 模式下)

```
:r filename
```

执行外部命令

- 在 vim 中执行外部命令 (Command 模式下)
  - 输入 **:!** 然后紧跟着输入一个外部命令可以执行该外部命令。

```
:!ls -l /etc/
```

    - 所有的外部命令都可以以这种方式执行

  - 输入 **:sh** 暂时退出 vim，回到 Shell 中，结束时输入 exit 或按 Ctrl+d 返回 vim。

在线帮助命令

- vim 拥有一个细致全面的在线帮助系统。可在 vim 中的 Command 模式下用下面的命令启动该帮助系统

```
:help
```

  - 关闭帮助窗口命令 (Command 模式下)

```
:q
```

Command 模式下的所有命令都必须输入 <回车> 运行！

- 在 shell 命令行中输入 vimtutor 可打开一个关于 vim 的简短中文教程，退出时用 `:q` 命令。

vim 启动脚本

- 创建 vim 启动脚本

vim 的功能特性要比 vi 多得多，但在缺省状态下大部分功能都没有激活。为了启动更多的功能，用户需要创建一个 vim 的 配置文件 .vimrc 。

```
cp /usr/share/vim/vim63/vimrc_example.vim  ~/.vimrc
```

- 你可以把你喜欢的其它功能设置添加到这个 .vimrc 配置文件中。

vim 的一些使用技巧

- 插入补全

在插入模式下，为了减少重复的击键输入，vim 提供了若干快捷键，当你要输入某个上下文曾经输入过的字符串时，你只要输入开头若干字符，使用快捷键，vim 将搜索上下文，找到匹配字符串，把剩下的字符补全，你就不必敲了。这样，编程序时你起多长的变量名都没关系了，而且还可以减少输入错误。

技巧：插入补全

- **Ctrl+p**：向上搜索，补全一个字 (word)。

例如：上文中出现过 **filename** 这个词，当你想再输入 **filename** 时，只要先输入 **f**，如何按 **ctrl+p** 即可。假如 **vim** 向上搜索，找到以 **f** 开头的第一个匹配不是 **filename**，你可以继续按 **ctrl+p** 搜索下一个匹配进行补全。当然，如果你想一次 **ctrl+p** 就成功，你可以多输入几个字符比如 **fil** 再按 **ctrl+p** 补全。

- **Ctrl+n**：向下搜索，补全一个字 (word)。

技巧：插入补全

- **Ctrl+x Ctrl+l**：补全一行

比如你写过一行

```
for (int i = 0; i < 100; i++)
```

想输入一个相同的文本行，只要输入

for Ctrl+x Ctrl+l

即可。如果补全出来的不是你想要的那一行，你可以按 **Ctrl+p** 或 **Ctrl+n** 选择上一个或下一个匹配行。

- **Ctrl+x Ctrl+f**

  在文件系统中搜索，补全一个文件名

技巧：大小写切换等

- 大小写自动切换 (Normal 模式下)

`~`	切换光标所在字母的大小写
`u/U`	切换大小写

- 加密当前文件 (Command 模式下)

```
:X
```

技巧：行首/行尾插入文本

- 行首/行尾插入文本 (Command 模式下)

`:s/^/string/g`	在当前行的行首插入字符串 **string**
`:s/$/string/g`	在当前行的末尾插入字符串 **string**

- 也可在多行或所有行的行首（末尾）插入字符串

```
:m,ns/^/string/g
```

```
:%s/^/string/g
```

- 一些有用的扩展：

vs + vi plugin

eclipse + vi plugin

pycharm + vi plugin

chrome + vi plugin

firefox + vi plugin

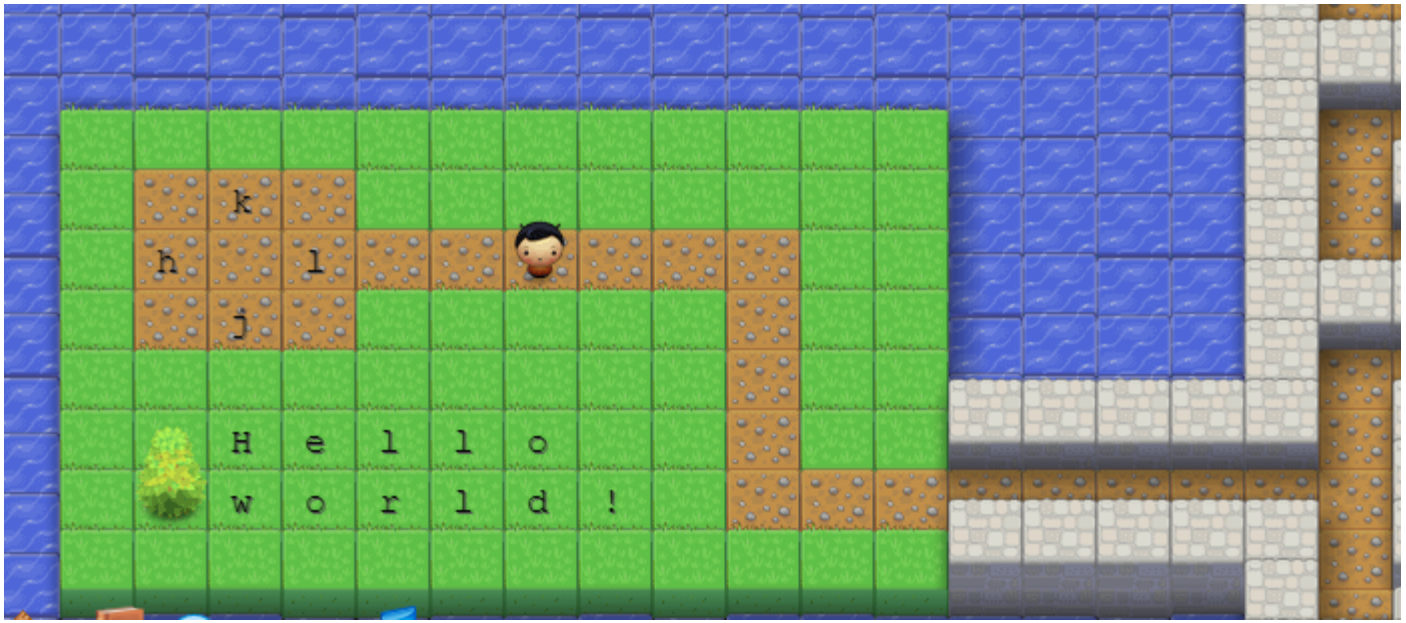
sublime + vi plugin

vscode + vi plugin

...

- Vi练习：

  https://vim-adventures.com/



作业

- 参考以下扩展，配置VIM

  https://github.com/wklken/k-vim
  https://github.com/amix/vimrc

- 请完成 VIM-练习.pdf 中的操作练习