Emacs自力求生指南 – 前言

Chen

# Eamcs是什么？

就我理解， Emacs 是一个长得像文本编辑器的 REPL ：

* 它是 Emacs lisp ──一个 Lisp 方言──的运行时。
* 每次你打开 Emacs ，它都会使用一个干净的[[1]](#footnote-20)运行时跑一遍 ~/.emacs.d/init.el 脚本。[[2]](#footnote-21)
* 你所看到的内容（窗口、文字、状态栏、光标）以及与它的互动（键盘、鼠标）所造成的内容改变，均为上述脚本执行函数所产生的 副作用 。

# 缺点？

先说缺点吧，客观一些。

## 学习成本高

想要随心所欲地使用 Emacs 的话，不可避免地需要用 Elisp 深度定制。

推荐参考书是 [ANSI Common Lisp](https://acl.readthedocs.io/en/latest/index.html)指南 。

把 Emacs 当作 CL 的 REPL 基本没啥问题，只要 (require 'cl) 就可以用 Common Lisp 的关键字了。

这么想，你在用一个编辑器的时候顺便还能把 LISP 大家族 Racket 、 Clojure ( ClojureScript )、 Scheme 都入了个门，何乐而不为啊。

## 有些操作会阻塞编辑器

虽然 Emacs 26+ 的 async 已经实现得很好了，日常操作基本不会被阻塞。但你依然还是可能会被什么套件里的同步调用卡一下。比如：

* 打开一个 parse 特别费劲的文件，比如一个超大的单行 JSON
  + 等 tree-sitter 完善后这将不会成为一个问题，应该……
* 有些网络 IO，比如 gnus 下载新闻
  + 现代的前后端分离型软件（比如 telega ）不会有这个问题
* 开一个大图片预览 Buffer 或打开一个 PDF
  + 惊人的事实： Emacs 通过把 PDF 转成图片来预览。所以如果你对一个上百 M 的大 PDF 做缩放操作会十分酸爽……

虽然上述都能被 C-g 打断，不过嘛，没有一个工具是万能的。如果你觉得有个功能 Emacs 干得不够好，那就立刻换一个工具吧。时间宝贵。

## 分量不够轻，导致默认装机量不够

这个是真的没办法了… vi （不是 vim ）几乎是每个服务器 Linux 的标配，但 Emacs 的基础包实在太大[[3]](#footnote-26)，甚至不少桌面版 Linux 都不会预装它。不过 macOS 居然预装了它，难道帮主爱用？

## 前后端互动麻烦

Emacs 作为前端与后端进程通信的场景，在极端情况下，性能不理想：后端 burst 大量数据时可能会卡死 Emacs ，比如 LSP 后端一次性给了太多的补全建议之类的。

如果写 Emacs 的外部动态库，弹性就不太够。

# 长处

## 万物皆文本

除了状态栏（mode line）外[[4]](#footnote-30)，所有你看到的文字都能使用统一的逻辑、统一的环境来互动：

* 你在写文档 (org-mode) ，文档里需要插入一段 C 。org-mode 提供一个函数，把这段嵌入代码映射到一个新的子窗口。你可以在这个子窗口里享受所有编辑 C 项目时你所使用的工具和环境，比如代码补全(LSP 和 company-mode)、预定义的代码片段（ yasnippet ）、语法查错（ flycheck ）等。
* 用 [helm grep](https://github.com/emacs-helm/helm/wiki/Grep) 可以搜索整个 Project ，搜索结果呈现在一个 Buffer 里。你可以 直接修改[这个 Buffer](https://github.com/mhayashi1120/Emacs-wgrep) 并「保存」 。同样，修改过程中你可以使用所有你早已熟悉的文本处理工具和流程，比如可视化正则文本处理器 anzu 、多光标 multiple-cursors.el ，甚至临时写个 elisp 函数并当场执行也可以。
* Emacs 自带的 dired 是一个文件浏览器。同样，你可以在它的 buffer 里「 直接修改并保存 」。从此批量更名再也不用找额外的软件或者记额外的命令。注意图中右下角的 Editable 。
* eval-expression （默认 M-: ） 可以把最底下一行（minibuffer）变成一个临时的 Elisp REPL，这里你可以执行任何 Elisp 函数，结果也会回显在里面。哪怕这个输入框只有一行高度，你会发现编辑体验和编辑一个 .el 文件是一致的：都有括号配平、都有函数名补完、一样能使用片段展开，甚至还能继续用 C-x C-e 来「临时执行表达式的一部分」。
* 这一行的可订制性和大 buffer 是一模一样的，很多软件，诸如 ivy 或者 smex ，都把这一行玩出了花。

## 文本皆结构，编辑文本实为操作结构

首先我强烈建议你花三分钟[看看这个视频](https://www.youtube.com/watch?v=D6h5dFyyUX0) 。

然后来重温一下这个经典的小故事：

转载自译言网 | <http://article.yeeyan.org/view/legendsland/209584> 在 ILC 2002 大会上，前Lisp大神，当今的Python倡导者 Peter Norvig，由于某些原因，做一个类似于马丁路德在梵蒂冈宣扬新教的主题演讲，因为他在演讲中大胆地声称Python就是一种Lisp。

讲完后进入提问环节，出乎我意料的是，Peter点了我过道另一侧，靠上面几排座位的一个老头，他衣着皱褶，在演讲刚开始的时候踱步进来，然后就靠在了那个座位上面。

这老头满头凌乱的白发，邋遢的白胡须，像是从旅行团中落下的游客，已经完全迷路了，闲逛到这里来歇歇脚，随便看看我们都在这里干什么。我的第一个念头是，他会因为我们的奇怪的话题感到相当失望；接着，我意识到这位老头的年纪，想到斯坦福就在附近，而且我想那人也在斯坦福 —— 难道他是……

“嗨，John，有什么问题？” Peter说。

虽然这只是10个字左右的问题，我不会假装自己记住了Lisp之父约翰麦卡锡说的每一个字。他在问Python程序能不能像处理数据一样，优雅地处理Python代码。

“不行。John, Python做不到。”

Peter就回答了这一句，然后静静地等待，准备接受教授的质疑，但老人没有再说什么了。此时，无语已胜千言。

什么叫「像处理数据一样处理代码」？我们知道整个 Emacs 都是用 Elisp 构建起来的，而 Lisp 的迷人之处就在于「代码即数据，数据即代码」： List 在没被求值之前是数据，被求值时就成了代码。

视频里使用的那些快捷键和函数，与其说是文本操作，不如说是操作了语法树后，又重新渲染回 buffer 文本。所以在 Emacs 里写 Lisp 、写 Clojure 、写 Elm 是非常非常享受的事情，心智负担和操作负担都比其它抽象语言好得多。

不要去玩那些括号玩笑了，差远了，用 paredit 写 lisp 根本不需要数括号，哪怕把括号全隐藏都能写出 valid 的程序。

代码是什么？是文本。数据是什么？是结构。「文本即结构」的血脉流淌在 Emacs 的各个角落，除了写 Lisp 之外：

* paredit-everywhere 可以把「编辑语法树」的思想扩展到几乎所有程序语言上
* # 举一个例子， || 表示光标所在位置  
  defmodule Test do  
  ||def abc do  
   "Hello World"  
   end  
  end  
  # C-k 为「删除到行尾」。  
  # 在上述光标所处位置，一般版本的 C-k 会立刻打破 do...end 平衡  
  # 如果使用 paredit-everywhere 提供的 paredit-kill 的话：  
  defmodule Test do  
   ||  
  end  
    
  # 如果光标在引号里呢？  
  defmodule Test do  
   def abc do  
   "Hello ||World"  
   end  
  end  
  # paredit-kill 后：  
  defmodule Test do  
   def abc do  
   "Hello ||"  
   end  
  end  
    
  # 其它括号也是一样  
  defmodule Test do  
   def abc do  
   some\_array = ||[  
   "1",  
   "2",  
   "3",  
   ]  
   end  
  end  
  # paredit-kill 后：  
  defmodule Test do  
   def abc do  
   some\_array = ||  
   end  
  end  
    
  # 我直接把 paredit-kill 绑定在 C-k 了。没这功能我写不了程序。
* org-mode 里的 org 指的是 organized plain text ，就是「文本即结构」的最直接体现。比如：
  + org-refile 会把整个标题及其所属内容移动到另一个标题之下，期间所有的级别变化、缩进都会自动完成
  + 调整标题或列表的上下顺序使用 M-up 和 M-down ，同样是以整个结构为单位的移动
  + 每个元素都分配有自己的 UUID。创建链接使用 UUID ，哪怕目标元素事后改变了位置或内容也不怕
  + 表格明明是纯文本写的，操作起来却和 Excel 差不多，甚至还能写自动计算公式
  + org-capture 可以快速往表格里 append 一行数据（而不用操心这个表格的边框有没有被打断之类的）
  + 甚至还有个 类似 SQL 的软件 可以以复杂的条件组合来查询你的文档库。

## GUI友好、鼠标友好、不反直觉

奇怪的是没几个 Emacs 介绍文提到这个的：Emacs 鼓励你使用它的 GUI 模式。

* 无参数启动 emacs 就是 GUI[[5]](#footnote-37)
* 自带了可深度自定义的 Menu 和 Toolbar
  + 大部分常用功能都能在菜单栏里找到，甚至还能显示当前快捷键组合。前期我建议你不要关掉菜单栏，找个功能还是相当方便的……
* 鼠标的框选、滚轮、双击、右键菜单等操作和你的使用习惯一致
* 外观、字体的颗粒度极细。可以使用多套字体、所有桌面色彩和花哨的 window decoration
* 甚至 Emacs 本身有一个类似控制面板的 GUI 配置界面 M-x customize ，可以不写 Elisp 、不碰配置文件也能轻度定制编辑器行为
* 看图、浏览网页、刷 Telegram 、预览 Markdown 等场景几乎只有在 GUI 内才 make sense

## 内置官方唯一指定软件包管理器

还是可视化的。可以直接点击 Install 按钮安装。

## 天生支持C/S模式

你肯定有过想在打开两个 vim 进程间互通剪贴板或光标互相跳转的场景，遗憾的是，不能。[[6]](#footnote-40)

Emacs 能以 server 模式启动， expose 到端口或 socket 文件。client 能随时连接它[[7]](#footnote-41)，还能主动抢占焦点。

## 文档又多又全还易读

M-x info 里的文档每一篇都可以拿来当小说读。

## 随时Hack， 彻底Hack

所有可见元素和变化都是 执行函数 所带来的副作用，所以你对编辑器的改造几乎没有场所和功能限制。

来几个例子体验一下 Emacs 的可定制性吧。这些例子很糙，接下来几章会更加系统。

### 自己造一个简单的 Vim 按键模式

在 Emacs 中，「按 j ，一个字母 j 出现在 Buffer 里」也是 函数调用带来的副作用 ！

按 C-h k j 可以看到调用的函数叫 (self-insert-command) 。

我们试着重新绑定 j 让它变成「跳到下一行」。

* 我们只知道按方向键下可以跳到下一行。首先用 C-h k (方向键下) 来查询对应的函数调用。我们可以得到很多信息：
  + 函数调用是 (next-line)
  + 方向键下在 Emacs 里写作 <down>
  + 这个函数也被绑定到 C-n 上了。
* 在 **scratch** Buffer 里试试看：

(global-set-key ;; 全局绑定设置  
(kbd "j") ;; 按键 j  
'next-line) ;; 调用 (next-line)

* 把光标移到最后一个括号的后面，按 C-x C-e （ (eval-last-sexp) ），现在按键盘上的 j ，你会发现光标真的向下跑了。
* 以此类推把 hjkl 都绑定了吧
* 我们这个例子太糙，执行了这个之后你的 j 就再也打不出字来了，最方便的复原法就是重启 Emacs……. 这个例子表明 Emacs 不仅能做 Vim 能做的任何事儿，而且甚至能做得更好。事实上，Emacs 的 evil-mode 是我用过的最接近 vim 原生的 vim style 实现。

### 定义自己的副作用函数

将光标向左移动一格，你会用 C-b 。我现在想写一个函数，让我能一次向前移动三格。

模拟击键 C-b 三次吗？不够鲁棒，万一有用户把它绑定到其它功能了怎么办。[[8]](#footnote-45)

那么我怎么精确定义这个函数呢？

* 使用 C-h k C-b 查询 C-b 究竟绑定了什么函数。结果是 (backward-char)
* 试试看，在 M-x 里使用 backward-char ，发现光标真的回退了一格
* 顺便在这个帮助文档里还看到了 (backward-char) 可以加一个参数用来表示回退几个字符
* 可以动笔写了：
* (defun my/backward-3-chars () ;; 该函数没有参数  
  "Backward 3 chars." ;; Docstring。以下是函数本体  
  (interactive) ;; 该函数可被 M-x 调用或绑定快捷键  
  (backward-char 3))
* 在 **scratch** Buffer 里粘贴这一段[[9]](#footnote-46)，把光标移到最后一个括号的后面，按 C-x C-e （ (eval-last-sexp) ），你会看到状态栏里出现了一个 my/backward-3-chars ，说明 defun 成功了[[10]](#footnote-47)。
* 试试在 M-x 里调用 my/backward-3-chars ，works as expected.
* 不妨把它绑定到一个快捷键上？
* (global-set-key (kbd "C-M-b") 'my/backward-3-chars) ;; Ctrl + Alt + b
* 把这些代码放到我的配置里，就能每次打开 Emacs 自动生效啦。

1. 其实带了一些 Emacs 预设的默认值。比如没有 ~/.emacs.d/init.el 文件时 Emacs 也依然能生成一个窗口来。 [↑](#footnote-ref-20)
2. Emacs 28 之后变成 $XDGCONFIGHOME/emacs/init.el 了，一般是 ~/.config/emacs/init.el [↑](#footnote-ref-21)
3. 如果精简太多就会让 Emacs 失去太多功能，最后变成跟 nano 差不多的存在…… [↑](#footnote-ref-26)
4. 但是这并不意味着状态栏无法定制了。事实上，不仅状态栏软件包 多如牛毛 ，而且甚至还有能 把状态栏和 minibuffer 二合一的软件 ，只能说定制 Emacs 是没有极限的。 [↑](#footnote-ref-30)
5. 也有 CLI 模式： emacs -nw [↑](#footnote-ref-37)
6. neovim 在 C/S 模式上做了不少努力，その努力を認めよう。 [↑](#footnote-ref-40)
7. 具体参考 emacsclient –help 和 C-h i m Emacs server 。简单地说，启动服务器是 emacs –daemon ，启动客户端是 emacsclient [↑](#footnote-ref-41)
8. 虽然不太可能，但因为 Emacs 什么都能做，所以如果真的有人这么干了，也请不要奇怪： Because he can. [↑](#footnote-ref-45)
9. 如果你用 Emacs 打开 org 格式的本文的话，你可以直接把光标放在 BEGINSRC 和 ENDSRC 内按 C-c C-c ，这段会自动执行，并将结果追加到这个代码块后面 [↑](#footnote-ref-46)
10. 状态栏里的显示是 (defun) 函数的求值结果：一个名叫 my/backward-3-chars 的 Symbol。如果你在 (+ 1 2) 的后面按 C-x C-e ，你会看到求值结果 3 [↑](#footnote-ref-47)