配置emacs

learn elisp in Y minutes

```
;; 15分钟学会Emacs Lisp (v0.2a)
;;（作者：bzg，https://github.com/bzg
;;  译者：lichenbo，http://douban.com/people/lichenbo）
;;
;; 请先阅读Peter Norvig的一篇好文:
;; http://norvig.com/21-days.html
;; （译者注：中文版请见http://blog.youxu.info/21-days/）
;;
;; 之后安装GNU Emacs 24.3:
;;
;; Debian: apt-get install emacs (视具体发行版而定)
;; MacOSX: http://emacsformacosx.com/emacs-builds/Emacs-24.3-universal-
10.6.8.dmg
;; Windows: http://ftp.gnu.org/gnu/windows/emacs/emacs-24.3-bin-i386.zip
;;
;; 更多信息可以在这里找到:
;; http://www.gnu.org/software/emacs/#Obtaining

;; 很重要的警告:
;;
;; 按照这个教程来学习并不会对你的电脑有任何损坏
;; 除非你自己在学习的过程中愤怒地把它砸了
;; 如果出现了这种情况，我不会承担任何责任
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; 打开emacs
;;
;; 按'q'消除欢迎界面
;;
;; 现在请注意窗口底部的那一个灰色长条
;;
;; "*scratch*" 是你现在编辑界面的名字。
;; 这个编辑界面叫做一个"buffer"。
;;
;; 每当你打开Emacs时，都会默认打开这个scratch buffer
;; 此时你并没有在编辑任何文件，而是在编辑一个buffer
;; 之后你可以将这个buffer保存到一个文件中。
;;
;; 之后的"Lisp interaction" 则是表明我们可以用的某组命令
;;
;; Emacs在每个buffer中都有一组内置的命令
;; 而当你激活某种特定的模式时，就可以使用相应的命令
;; 这里我们使用`lisp-interaction-mode'，
;; 这样我们就可以使用内置的Emacs Lisp（以下简称Elisp）命令了。

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
```

```
;; 分号是注释开始的标志
;;
;; Elisp 是由符号表达式构成的 (即"s-表达式"或"s式"):
(+ 2 2)

;; 这个s式的意思是 "对2进行加2操作".

;; s式周围有括号，而且也可以嵌套:
(+ 2 (+ 1 1))

;; 一个s式可以包含原子符号或者其他s式
;; 在上面的例子中，1和2是原子符号
;; (+ 2 (+ 1 1)) 和 (+ 1 1) 是s式.

;; 在 `lisp-interaction-mode' 中你可以计算s式.
;; 把光标移到闭括号后，之后按下ctrl+j（以后简写为'C-j'）

(+ 3 (+ 1 2))
;;           ^ 光标放到这里
;; 按下`C-j' 就会输出 6

;; `C-j' 会在buffer中插入当前运算的结果

;; 而`C-xC-e' 则会在emacs最底部显示结果，也就是被称作"minibuffer"的区域
;; 为了避免把我们的buffer填满无用的结果，我们以后会一直用`C-xC-e'

;; `setq' 可以将一个值赋给一个变量
(setq my-name "Bastien")
;; `C-xC-e' 输出 "Bastien" (在 mini-buffer 中显示)

;; `insert' 会在光标处插入字符串:
(insert "Hello!")
;; `C-xC-e' 输出 "Hello!"

;; 在这里我们只传给了insert一个参数"Hello!", 但是
;; 我们也可以传给它更多的参数，比如2个：

(insert "Hello" " world!")
;; `C-xC-e' 输出 "Hello world!"

;; 你也可以用变量名来代替字符串
(insert "Hello, I am " my-name)
;; `C-xC-e' 输出 "Hello, I am Bastien"

;; 你可以把s式嵌入函数中
(defun hello () (insert "Hello, I am " my-name))
;; `C-xC-e' 输出 hello

;; 现在执行这个函数
(hello)
;; `C-xC-e' 输出 Hello, I am Bastien

;; 函数中空括号的意思是我们不需要接受任何参数
;; 但是我们不能一直总是用my-name这个变量
;; 所以我们现在使我们的函数接受一个叫做"name"的参数
```

```
(defun hello (name) (insert "Hello " name))
;; `C-xC-e' 输出 hello

;; 现在我们调用这个函数，并且将"you"作为参数传递

(hello "you")
;; `C-xC-e' 输出 "Hello you"

;; 成功!

;; 现在我们可以休息一下

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; 下面我们在新的窗口中新建一个名为 "*test*" 的buffer:

(switch-to-buffer-other-window "*test*")
;; `C-xC-e' 这时屏幕上会显示两个窗口，而光标此时位于*test* buffer内

;; 用鼠标单击上面的buffer就会使光标移回。
;; 或者你可以使用 `C-xo' 使得光标跳到另一个窗口中

;; 你可以用 `progn'命令将s式结合起来:
(progn
  (switch-to-buffer-other-window "*test*")
  (hello "you"))
;; `C-xC-e' 此时屏幕分为两个窗口，并且在*test* buffer中显示"Hello you"

;; 现在为了简洁，我们需要在每个s式后面都使用`C-xC-e'来执行，后面就不再说明了

;; 记得可以用过鼠标或者`C-xo'回到*scratch*这个buffer。

;; 清除当前buffer也是常用操作之一：
(progn
  (switch-to-buffer-other-window "*test*")
  (erase-buffer)
  (hello "there"))

;; 也可以回到其他的窗口中
(progn
  (switch-to-buffer-other-window "*test*")
  (erase-buffer)
  (hello "you")
  (other-window 1))

;; 你可以用 `let' 将一个值和一个局部变量绑定:
(let ((local-name "you"))
  (switch-to-buffer-other-window "*test*")
  (erase-buffer)
  (hello local-name)
  (other-window 1))

;; 这里我们就不需要使用 `progn' 了， 因为 `let' 也可以将很多s式组合起来。
```

```
;; 格式化字符串的方法:
(format "Hello %s!\n" "visitor")

;; %s 是字符串占位符，这里被"visitor"替代.
;; \n 是换行符。

;; 现在我们用格式化的方法再重写一下我们的函数:
(defun hello (name)
  (insert (format "Hello %s!\n" name)))

(hello "you")

;; 我们再用`let'新建另一个函数:
(defun greeting (name)
  (let ((your-name "Bastien"))
    (insert (format "Hello %s!\n\nI am %s."
                    name       ; the argument of the function
                    your-name  ; the let-bound variable "Bastien"
                    ))))

;; 之后执行:
(greeting "you")

;; 有些函数可以和用户交互:
(read-from-minibuffer "Enter your name: ")

;; 这个函数会返回在执行时用户输入的信息

;; 现在我们让`greeting'函数显示你的名字:
(defun greeting (from-name)
  (let ((your-name (read-from-minibuffer "Enter your name: ")))
    (insert (format "Hello!\n\nI am %s and you are %s."
                    from-name ; the argument of the function
                    your-name ; the let-bound var, entered at prompt
                    ))))

(greeting "Bastien")

;; 我们让结果在另一个窗口中显示:
(defun greeting (from-name)
  (let ((your-name (read-from-minibuffer "Enter your name: ")))
    (switch-to-buffer-other-window "*test*")
    (erase-buffer)
    (insert (format "Hello %s!\n\nI am %s." your-name from-name))
    (other-window 1)))

;; 测试一下:
(greeting "Bastien")

;; 第二节结束，休息一下吧。

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; 我们将一些名字存到列表中:
(setq list-of-names '("Sarah" "Chloe" "Mathilde"))
```

```
;; 用 `car'来取得第一个名字:
(car list-of-names)

;; 用 `cdr'取得剩下的名字:
(cdr list-of-names)

;; 用 `push'把名字添加到列表的开头:
(push "Stephanie" list-of-names)

;; 注意: `car' 和 `cdr' 并不修改列表本身, 但是 `push' 却会对列表本身进行操作.
;; 这个区别是很重要的: 有些函数没有任何副作用（比如`car'）
;; 但还有一些却是有的 (比如 `push').

;; 我们来对`list-of-names'列表中的每一个元素都使用hello函数:
(mapcar 'hello list-of-names)

;; 将 `greeting' 改进，使的我们能够对`list-of-names'中的所有名字执行:
(defun greeting ()
    (switch-to-buffer-other-window "*test*")
    (erase-buffer)
    (mapcar 'hello list-of-names)
    (other-window 1))

(greeting)

;; 记得我们之前定义的 `hello' 函数吗？ 这个函数接受一个参数，名字。
;; `mapcar' 调用 `hello', 并将`list-of-names'作为参数先后传给`hello'

;; 现在我们对显示的buffer中的内容进行一些更改:

(defun replace-hello-by-bonjour ()
    (switch-to-buffer-other-window "*test*")
    (goto-char (point-min))
    (while (search-forward "Hello")
      (replace-match "Bonjour"))
    (other-window 1))

;; (goto-char (point-min)) 将光标移到buffer的开始
;; (search-forward "Hello") 查找字符串"Hello"
;; (while x y) 当x返回某个值时执行y这个s式
;; 当x返回`nil' (空), 退出循环

(replace-hello-by-bonjour)

;; 你会看到所有在*test* buffer中出现的"Hello"字样都被换成了"Bonjour"

;; 你也会得到以下错误提示: "Search failed: Hello".
;;
;; 如果要避免这个错误, 你需要告诉 `search-forward' 这个命令是否在
;; buffer的某个地方停止查找, 并且在什么都没找到时是否应该不给出错误提示

;; (search-forward "Hello" nil t) 可以达到这个要求:

;; `nil' 参数的意思是 : 查找并不限于某个范围内
```

```
;; `t' 参数的意思是: 当什么都没找到时，不给出错误提示

;; 在下面的函数中，我们用到了s式，并且不给出任何错误提示:

(defun hello-to-bonjour ()
    (switch-to-buffer-other-window "*test*")
    (erase-buffer)
    ;; 为`list-of-names'中的每个名字调用hello
    (mapcar 'hello list-of-names)
    (goto-char (point-min))
    ;; 将"Hello" 替换为"Bonjour"
    (while (search-forward "Hello" nil t)
      (replace-match "Bonjour"))
    (other-window 1))

(hello-to-bonjour)

;; 给这些名字加粗:

(defun boldify-names ()
    (switch-to-buffer-other-window "*test*")
    (goto-char (point-min))
    (while (re-search-forward "Bonjour \\(.+\\)!" nil t)
      (add-text-properties (match-beginning 1)
                           (match-end 1)
                           (list 'face 'bold)))
    (other-window 1))

;; 这个函数使用了 `re-search-forward':
;; 和查找一个字符串不同，你用这个命令可以查找一个模式，即正则表达式

;; 正则表达式 "Bonjour \\(.+\\)!" 的意思是:
;; 字符串 "Bonjour ", 之后跟着
;; 一组            |  \\( ... \\) 结构
;; 任意字符        |  . 的含义
;; 有可能重复的    |  + 的含义
;; 之后跟着 "!" 这个字符串

;; 准备好了？试试看。

(boldify-names)

;; `add-text-properties' 可以添加文字属性，比如文字样式

;; 好的，我们成功了!

;; 如果你想对一个变量或者函数有更多的了解:
;;
;; C-h v 变量 回车
;; C-h f 函数 回车
;;
;; 阅读Emacs Lisp官方文档:
;;
;; C-h i m elisp 回车
;;
```

```
;; 在线阅读Emacs Lisp文档:
;; https://www.gnu.org/software/emacs/manual/html_node/eintr/index.html

;; 感谢以下同学的建议和反馈:
;; - Wes Hardaker
;; - notbob
;; - Kevin Montuori
;; - Arne Babenhauserheide
;; - Alan Schmitt
;; - spacegoing
```

我的配置文件

```
(custom-set-variables
 ;; custom-set-variables was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(current-language-environment "UTF-8")
 '(delete-selection-mode 1)
 '(global-hl-line-mode 1)
 '(global-linum-mode 1)
 '(inhibit-startup-screen t)
 '(make-backup-files nil)
 '(package-selected-packages '(doom-themes company which-key try use-package))
 '(ring-bell-function 'ignore)
 '(visible-bell 1)
 '(which-key-side-window-location 'right))
(custom-set-faces
 ;; custom-set-faces was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(default ((t (:family #("思源黑体 HW" 0 7 (charset chinese-gbk)) :foundry
"outline" :slant normal :weight normal :height 139 :width normal)))))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;


;; 添加melpa软件源
(require 'package)
(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)
;; Comment/uncomment this line to enable MELPA Stable if desired.  See `package-
archive-priorities`
;; and `package-pinned-packages`. Most users will not need or want to do this.
;;(add-to-list 'package-archives '("melpa-stable" .
"https://stable.melpa.org/packages/") t)
(package-initialize)

;; 设置use-package
```

```
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))
;; try 可以尝试某个包
(use-package try
  :ensure t)
;; 快捷键按地慢时可以显示快捷键
(use-package which-key
  :ensure t
  :config (which-key-mode))
;; company 以及快捷键配置
(use-package company
  :ensure t
  :bind(("C-=" . company-complete)
    :map company-active-map
        ("C-n" . company-select-next)
        ("C-p" . company-select-previous))
  :config
  (global-company-mode t))
;; doom-theme  https://github.com/doomemacs/themes
(use-package doom-themes
  :ensure t
  :config
  ;; Global settings (defaults)
  (setq doom-themes-enable-bold t    ; if nil, bold is universally disabled
        doom-themes-enable-italic t) ; if nil, italics is universally disabled
  (load-theme 'doom-ayu-dark t)

  ;; Enable flashing mode-line on errors
  (doom-themes-visual-bell-config)
  ;; Enable custom neotree theme (all-the-icons must be installed!)
  (doom-themes-neotree-config)
  ;; or for treemacs users
  (setq doom-themes-treemacs-theme "doom-atom") ; use "doom-colors" for less
minimal icon theme
  (doom-themes-treemacs-config)
  ;; Corrects (and improves) org-mode's native fontification.
  (doom-themes-org-config))

;; org mode 设置
;; 在所有的模式下都可以打开 org-store-link org-agenda org-capture
(global-set-key (kbd "C-c l") #'org-store-link)
(global-set-key (kbd "C-c a") #'org-agenda)
(global-set-key (kbd "C-c c") #'org-capture)
;; org mode 默认显示图片
(defun turn-on-org-show-all-inline-images ()
  (org-display-inline-images t t))
(add-hook 'org-mode-hook 'turn-on-org-show-all-inline-images)
;;重设整个文件的缩进函数
(defun indent-whole ()
  (interactive)
  (indent-region (point-min) (point-max))
  (message "Full text indent successfully"))
;;绑定到F7键
(global-set-key [f7] 'indent-whole)
```