readme

pengpengxp

2014 年 12 月 20 日

目录

This is My first repositry on GitHub

I'm pengpengxp, who study in CQUPT in Chongqing, China.I like use emacs.

I want to put my emacs configuration here

email:pengpengxppri@gmail.com

1 我的 emacs 配置

重新配置 emacs 也有一段时间了，今天早上也不想做事，不如就来补上这篇 emacs 的配置文件的说明把。

之前我的配置文件都是乱的。参考了网上别人的配置文件，将我的配置文件也“格式化”了一下。

1.1 init.el needtobeimproved

init.el 是总调用接口。首先在里面设置我的环境变量：

```
;;;设置环境变量
(setenv "PATH" (concat "/home/pengpengxp/peng_bin:" (getenv "PATH")))


(setq HOME (getenv "HOME"))
(setq DIR (concat HOME "/.emacs.d"))
(setq GTD (concat HOME "/gtd"))
(setq LISP (concat DIR "/lisp"))
(setq SITE-LISP (concat DIR "/site-lisp"))
```

然后设置 load-path：

```
(add-to-list 'load-path DIR)
(add-to-list 'load-path SITE-LISP)
(add-to-list 'load-path LISP)
(let ((default-directory SITE-LISP))    ;Don't add load-path after plugins every time
  (normal-top-level-add-subdirs-to-load-path))
```

这最后一句是将所有 SITE-LISP 目录下的子目录都加入到了 load-path 中去了。(我的理解是这样的，但是没有具体测试是不是所有子目录)。其中 SITE-LISP 是我放置第三方插件的地方。这样不用每次添加插件的的时候都使用 add-to-list 手动添加了。这里可能需要改进一下。

* 我觉得这样定义的 load-path 太多了。需要寻找的地方太多，可能影响速度。*

接下来是我 copy 别人的函数和自定义的函数：

```
(require 'init-peng-copyfun)            ;;;; some function I copyed from others
(require 'init-peng-prifun)             ;;;; load function wrote by pengpengxp
```

然后就是所有第三方插件：注意这些插件，必须首先加载 evil。因为下面有很多按键绑定都是在 evil 的基础上绑定的。当然，也可以每次在需要加载的时候都调用一次

```
(require 'evil)
```

最后加载 init-font.el 和 init-global.el;

1.2 site-lisp

site-lisp 目录用于存放所有第三方插件。直接下载下来放到这里就行了。我一般使用 git clone *

1.3 lisp

lisp 目录用于存放我所有的 init-*.el 文件，就是我实际上的所有配置文件。每个文件都以 provide 结尾，以供 init.el 调用。

1.3.1 init-evil.el question

1. 安装了一些基于 evil 的插件：

```
(require 'evil-surround)
(require 'evil-nerd-commenter)
```

```
(require 'evil-visualstar)
(require 'evil-leader)
(require 'evil-numbers)
```

设置：

```
(evilnc-default-hotkeys)
(global-evil-surround-mode 1)
(evil-mode 1)
(define-key evil-normal-state-map (kbd "C-c +") 'evil-numbers/inc-at-pt)
(define-key evil-normal-state-map (kbd "C-c -") 'evil-numbers/dec-at-pt)
(setq evilnc-hotkey-comment-operator ",,")
```

2. 一些 mode 的 initial-state evil 中的 state 就是 vi 中的 mode，emacs 默认的 mode 还是以前 mode 的意思。evil 总共有好几个 state，其中我经常使用的就是 normal,insert,emacs 三个。evil 对每个 mode 都是默认进入的 normal。有些 emacs 的 mode 我不想使用 evil。就设置为默认进入 emacs-mode：

   问题：有些 mode，比如 bm-show-mode 是在 init.el 后面才加载的。这样在前面设置也没有问题

```
;;; 设置这些模式的默认evil模式
(evil-set-initial-state 'ibuffer-mode 'emacs)
(evil-set-initial-state 'bookmark-bmenu-mode 'emacs)
(evil-set-initial-state 'Info-mode 'emacs)
(evil-set-initial-state 'compilation-mode 'emacs)
(evil-set-initial-state 'help-mode 'emacs)
(evil-set-initial-state 'dired-mode 'emacs)
(evil-set-initial-state 'compilation-mode 'emacs)
(evil-set-initial-state 'apropos-mode 'emacs)
(evil-set-initial-state 'magit-mode  'emacs)
(evil-set-initial-state 'magit-process-mode 'emacs)
(evil-set-initial-state 'mew-draft-mode 'emacs)
```

```
(evil-set-initial-state 'mew-summary-mode 'emacs)
(evil-set-initial-state 'mew-message-mode 'emacs)
(evil-set-initial-state 'bm-show-mode 'emacs)
(evil-set-initial-state 'Man-mode 'emacs)
```

esc 就是退出 insert 模式。不再是 emacs 中的 esc

```
(setq evil-esc-delay 0)
```

3. 定制 insert-state **基本都是从网上找到的**

   我认为 emacs 在进行纯输入的时候是很强大的，不需要 evil，所有我
   首先把所有 evil-insert-state 中的按键绑定都去掉：

```
;; remove all keybindings from insert-state keymap,it is VERY VERY important
(setcdr evil-insert-state-map nil)
```

   然后把 emacs 模式下的所有按键绑定到 insert 模式下：

```
(define-key evil-insert-state-map
  (read-kbd-macro evil-toggle-key) 'evil-emacs-state)
```

   在 insert-mode 下，esc 需要能退出 insert-mode 到 normal-mode 中：

```
;; but [escape] should switch back to normal state
(define-key evil-insert-state-map [escape] 'evil-normal-state)
```

   之前我喜欢在 vi 中使用 kj 来退出 insert 回到 normal 中，这里抄了
   一个别人的函数：**（暂时还没有看懂）**

```
(define-key evil-insert-state-map "k" #'cofi/maybe-exit)
(evil-define-command cofi/maybe-exit ()
  :repeat change
  (interactive)
```

```
  (let ((modified (buffer-modified-p)))
    (insert "k")
    (let ((evt (read-event (format "Insert %c to exit insert state" ?j)
   nil 0.5)))
      (cond
       ((null evt) (message ""))
       ((and (integerp evt) (char-equal evt ?j))
(delete-char -1)
(set-buffer-modified-p modified)
(push 'escape unread-command-events))
       (t (setq unread-command-events (append unread-command-events
      (list evt)))))))
```

4. 按键绑定 主要是定义 evil-normal-state-map 和 evil-motion-state-map。刚开始的想法是把',' 定义成这两个 map 中的 prefix-key：

   PS: 但是我发现，直接使用init-evil-leader.el比较简单了，不过 evil-leader 使用到最后感觉定制性没有那么强，现在暂时使用着 evil-leader

```
(define-prefix-command 'peng-evil-global-map)
(define-key evil-normal-state-map (kbd ",") 'peng-evil-global-map)
(define-key evil-motion-state-map (kbd ",") 'peng-evil-global-map)
```

   然后就是实际定义，使用 define-key 这样定义：

```
;; ;;; normal-map
(define-key evil-normal-state-map (kbd "DEL") 'delete-other-windows)
(define-key evil-normal-state-map "ei " 'find-file)
(define-key evil-normal-state-map ",," 'evilnc-comment-operator)
(define-key evil-normal-state-map ",1" 'delete-other-windows)
(define-key evil-normal-state-map ",0" 'delete-window)
(define-key evil-normal-state-map ",2" 'split-window-below)
(define-key evil-normal-state-map ",3" 'split-window-right)
```

```
(define-key evil-normal-state-map ",u" 'winner-undo)
(define-key evil-normal-state-map ",r" 'winner-redo)
(define-key evil-normal-state-map ",h" 'eshell)
(define-key evil-normal-state-map ",p" 'switch-to-buffer)
(define-key evil-normal-state-map ",n" 'save-buffer)
(define-key evil-normal-state-map ",k" 'kill-buffer)
(define-key evil-normal-state-map ",w" 'eshell)
(define-key evil-normal-state-map ",b" 'ibuffer)
(define-key evil-normal-state-map "m" 'point-to-register)
(define-key evil-normal-state-map "'" 'jump-to-register)
(define-key evil-normal-state-map "-" 'split-window-below)
(define-key evil-normal-state-map "|" 'split-window-right)
(define-key evil-normal-state-map "q" 'View-quit)
(define-key evil-normal-state-map (kbd "C-n") 'evil-next-line)
(define-key evil-normal-state-map (kbd "C-r") 'isearch-backward)
(define-key evil-normal-state-map (kbd "C-p") 'evil-previous-line)
(define-key evil-normal-state-map (kbd "C-e") 'move-end-of-line)
(define-key evil-normal-state-map (kbd "M-.") 'find-tag)
(define-key evil-normal-state-map (kbd "C-b") 'backward-char)
(define-key evil-normal-state-map (kbd "C-f") 'forward-char)
(define-key evil-normal-state-map (kbd "K") 'man)

;; ;;; motion map
(define-key evil-motion-state-map "ei " 'find-file)
(define-key evil-motion-state-map ",1" 'delete-other-windows)
(define-key evil-motion-state-map ",0" 'delete-window)
(define-key evil-motion-state-map ",2" 'split-window-below)
(define-key evil-motion-state-map ",3" 'split-window-right)
(define-key evil-motion-state-map ",u" 'winner-undo)
(define-key evil-motion-state-map ",r" 'winner-redo)
```

```
(define-key evil-motion-state-map ",h" 'eshell)
(define-key evil-motion-state-map ",p" 'switch-to-buffer)
(define-key evil-motion-state-map ",n" 'save-buffer)
(define-key evil-motion-state-map ",k" 'kill-buffer)
(define-key evil-motion-state-map ",w" 'eshell)
(define-key evil-motion-state-map ",b" 'ibuffer)
(define-key evil-motion-state-map "-" 'split-window-below)
(define-key evil-motion-state-map "|" 'split-window-right)
(define-key evil-motion-state-map "m" 'point-to-register)
(define-key evil-motion-state-map "'" 'jump-to-register)
(define-key evil-motion-state-map (kbd "C-n") 'evil-next-line)
(define-key evil-motion-state-map (kbd "C-r") 'isearch-backward)
(define-key evil-motion-state-map (kbd "C-p") 'evil-previous-line)
(define-key evil-motion-state-map (kbd "C-e") 'move-end-of-line)
(define-key evil-motion-state-map (kbd "M-.") 'find-tag)
```

visual-mode 也需要一些 emacs 的移动方式以适合我的习惯：

```
(define-key evil-visual-state-map (kbd "C-e") 'move-end-of-line)
(define-key evil-visual-state-map (kbd "C-b") 'backward-char)
(define-key evil-visual-state-map (kbd "C-f") 'forward-char)
```

最后一些是 evil quit，我不是很懂，从网上抄过来的：

```
;; evil quit
(define-key evil-normal-state-map [escape] 'keyboard-quit)
(define-key evil-visual-state-map [escape] 'keyboard-quit)
(define-key minibuffer-local-map [escape] 'helm-keyboard-quit)
(define-key minibuffer-local-ns-map [escape] 'helm-keyboard-quit)
(define-key minibuffer-local-completion-map [escape] 'helm-keyboard-quit)
(define-key minibuffer-local-must-match-map [escape] 'helm-keyboard-quit)
(define-key minibuffer-local-isearch-map [escape] 'helm-keyboard-quit)
```

1.3.2 init-org.el

1. 环境变量 主要是为了方便我进行 gtd 设置，gtd 文件主目录：

```
(setq ORG-HOME "/home/pengpengxp/gtd")
```

org-agenda 文件

PS：org-agenda 应该会只在这些文件中寻找事件，这样的目的是不再 org-agenda 中显示那些已经完成或者已经删掉的文件

```
(setq ORG-AGENDA-FILES (list (concat ORG-HOME "/inbox.org")
     (concat ORG-HOME "/book.org")
     (concat ORG-HOME "/dreams.org")
     (concat ORG-HOME "/note.org")
     (concat ORG-HOME "/test.org")
     (concat ORG-HOME "/Tips.org")
     ))
```

org-refile

PS：refile 的时候又需要在所有文件中都能 refile

```
(setq ORG-REFILE-FILES (list (concat ORG-HOME "/book.org")
     (concat ORG-HOME "/dreams.org")
     (concat ORG-HOME "/finished.org")
     (concat ORG-HOME "/inbox.org")
     (concat ORG-HOME "/note.org")
     (concat ORG-HOME "/README.org")
     (concat ORG-HOME "/test.org")
     (concat ORG-HOME "/Tips.org")
     (concat ORG-HOME "/trash.org")
     ))
```

2. hook org-mode-hook 进行主要的 org 的设置：

```
(add-hook 'org-mode-hook '(lambda ()
    (interactive)
    (local-set-key (kbd "<tab>") 'org-cycle)
    (local-set-key (kbd "<C-tab>") 'other-window)
    (local-set-key (kbd "<C-return>") 'org-insert-heading-respect-content)
    (setq truncate-lines nil)
    (yas-minor-mode -1)
    (auto-fill-mode 1)
    ;; (hl-line-mode 1)
    (local-set-key (kbd "C-c a") 'org-agenda)
    (setq org-agenda-files ORG-AGENDA-FILES)
    (setq org-directory ORG-HOME)
    (org-indent-mode 1) ;不显示哪么多个*
    (when window-system
      (local-set-key (kbd "<s-return>") 'org-insert-subheading))
    ))
```

org-agenda-hook 定制一下我自己的东西:

```
(add-hook 'org-agenda-mode-hook '(lambda ()
   (delete-other-windows)
   (linum-on)
   (hl-line-mode 1)
   ))
```

3. GTD 设置 使得 refile 可以在所有 ORG-REFILE-FILES 中进行：

```
(setq org-refile-targets (quote ((nil :maxlevel . 9)
 (ORG-REFILE-FILES :maxlevel . 9))))
```

我的事件中所有可能的状态：

```
(setq org-todo-keywords '((sequence "TODO(t!)"
```

```
    "DOING(n)"
    "WAITING(w)" ;waiting for others
    "SOMEDAY(s)" ;I'll do it someday
    "Dreams(i)"
    "Tips(p)"
    "|"
    "DONE(d@/!)"
    "ABORT(a@/!)"
    )))
```

使用 org-capture-template 快速抓取事件：

```
(setq org-capture-templates
      '(("t" "News" entry (file+datetree (concat ORG-HOME "/inbox.org"))
 "*  TODO  [#A]  %?\n %T")

("i" "Dreams" entry (file+datetree (concat ORG-HOME "/dreams.org"))
 "*  Dreams  %?\n %T")

("s" "SOMEDAY" entry (file+datetree (concat ORG-HOME "/inbox.org"))
 "*  SOMEDAY  %?\n %T")

("p" "Tips" entry (file+datetree (concat ORG-HOME "/Tips.org"))
 "*  Tips  %?\n %T")

("b" "Book" entry (file+datetree (concat ORG-HOME "/book.org"))
 "*  SOMEDAY  %?\n %T")

("n" "Notes" entry (file+datetree (concat ORG-HOME "/note.org"))
 "*  TODO  %?\n %T")

("a" "Account" table-line (file+headline (concat ORG-HOME "/account.org.gpg") "W
```

```
 "|")

("k" "test" entry (file+datetree (concat ORG-HOME "/test.org") "Tasks")
 "* TODO  %?  \n %T")
))
```

设置默认的 org-default-note-file：

```
(setq org-default-notes-file (concat ORG-HOME "/inbox.org"))
```

设置使用方便使用 org 的全局按键绑定：

```
(global-set-key (kbd "C-c c") 'org-capture)
(global-set-key (kbd "C-c a") 'org-agenda)
(global-set-key (kbd "C-c l") 'org-store-link)
(global-set-key (kbd "C-c b") 'org-iswitchb)
```

定制自己的 org-agenda 选项：

```
;;; 定制自己的org-agenda选项。这样Ctrl-a以后可供选择。
(setq org-agenda-custom-commands
      '(
("d" "Agenda and Home-related tasks"
 (
  (agenda "")
  (todo "DOING")
  (todo "WAITING")
  ))
("w" "things WAITING"
 (
  (agenda "")
  (todo "WAITING")
```

```
  ))
("o" "things TODO"
 (
  (agenda "")
  (todo "TODO")
  ))
;; ("h" . "h for peng's dispatcher") ; description for "h" prefix
("ht" todo "TODO")
("hn" todo "DOING")
("hd" todo "DONE")
("hw" todo "WAITING")
("hi" todo "Dreams")
("hp" todo "Tips")
("hs" todo "SOMEDAY")
))
```

org-agenda 默认只显示一天的事件：

```
(setq org-agenda-span 'day)
```

1.3.3 init-org-export.el

设置 org 的导出，我现在使用的，一般还是导出成 html。所以主要还是针对 html 的设置，在导出 html 时，由于开启 auto-fill，一段话可能分成多行，英文中两行之间加入一个空格就刚刚好，但是中文就不行，会出现多余的空格。解决办法是使用版本新一点的 org-mode。然后加入下面这两个函数：

PS：因为貌似有冲突，我直接把 emacs 原来版本的 org-mode 删除了，然后使用下载的新版本原来版本的 org-mode 放在/usr/share/emacs/中，现在我的 org-mode 直接放在 SITE-LISP 中了

```
(defun clear-single-linebreak-in-cjk-string (string)
```

```
  "clear single line-break between cjk characters that is usually soft line-breaks"
  (let* ((regexp "\\([\u4E00-\u9FA5]\\)\n\\([\u4E00-\u9FA5]\\)")
 (start (string-match regexp string)))
    (while start
      (setq string (replace-match "\\1\\2" nil nil string)
    start (string-match regexp string start))))
  string)
(defun ox-html-clear-single-linebreak-for-cjk (string backend info)
  (when (org-export-derived-backend-p backend 'html)
    (clear-single-linebreak-in-cjk-string string)))
(add-to-list 'org-export-filter-final-output-functions
     'ox-html-clear-single-linebreak-for-cjk)
```

设置导出的 css 格式：

```
(setq org-html-head "<link href=\"css/org-manual.css\" rel=\"stylesheet\" type=\"text
```

我的理解其实就是默认引用准备导出 org 文件目录下 css 目录中的、org-maual.css 文件作为 css 格式。

当然，这需要该文件存在才行

1.3.4 init-eim.el eim

1. 基本配置 没什么好说的，直接照着 eim 中的 readme 配置好就能用了。把默认的输入法设置成 eim-py 就行：

   ```
   (setq default-input-method "eim-py")
   ```

   我觉得默认的 C-\ 开启输入法不方便。自己绑定到了 C-backspace 上

   ```
   (global-set-key (kbd "<C-backspace>") 'toggle-input-method)
   ```

   使用 ‘;’ 暂时输入中文，在 cc-mode 这些中不起作用，全局绑定成 s-;:

```
(global-set-key (kbd ";") 'eim-insert-ascii)
(peng-global-set-key (kbd "s-;") 'eim-insert-ascii)
```

其中 peng-global-set-key 是我自己写的函数，用于在有 evil 的情况下的按键绑定。这是在使用 evil 是真正的全局设置。

2. **DONE** 控制每页显示的词条数目 NEEDTOBEIMPROVED

   - State "DONE" from "TODO" *[2014-11-05 三 16:33]*
     最后还是在 py.txt 中写了 page-length=9, 然后调用 eim-build-table，结果莫名其妙的就好了。eim 没怎么吃透，有时间看看源码。
   - State "TODO" from "" *[2014-11-05 三 11:43]*

   刚开始我设置的是每页显示 10 个。但是后来发现一个 bug 是第 10 个条目我不能选择。所以现在想设置为 9 个。之前应该就在 eim 目录下的 py.txt 中写 page-length=9 就行了。但是每次重启 emacs 就会被重写为 10。py.txt 总是会被不明原因的重写。我不知道这是为什么??

3. **DONE** eim 中加入自己的词库!!!! EIM

   - State "DONE" from "DOING" *[2014-10-25 六 11:33]*
     首先想使用 pyword2tbl.pl。结果 perl 一直都没有配置好。所以不能转换一些指定的词库。

     然后试着直接修改了一下 py.txt 中的内容。但是发现每次重启 py.txt 都会被重置。最后发现了这个 otherpy.txt。修改这个词库文件没有问题。不会被重置。所以最后的解决方案是：

     下载搜狗词库，自己把里面的' 替换成为 -。然后直接添加在 otherpy.txt 后面，然后调用 eim-build-table（可能需要一会儿）。保存重启就可以了。

     现在就是使用 eim 来输入的。感觉还不错。*<2014-10-24 五 21:12>*

发现一个更加简单的添加词库的方式。在 py.txt 的开头有一个 other-files 选项。在其中加入词库文件就可以了。其中词库文件格式就参考 other_{py}.txt 中的就行。（其实我觉得应该就是 py.txt 中的格式）。

接下来就是自己制作自己的词库文件拉：拷贝 other_{py}.txt 到新词库文件中，把 Table 下面的内容换成自己的词库。然后调用 eim-build-table 自动就可以构建好这个词库文件。然后只需要将词库文件添加到 py.txt 的 other-files 选项中就可以了。

暂时就用着网上找到的搜狗基础词库。没有找到更好的。但是基本能满足使用需求了。

以后有需要可以自己构造自己的词库。

整个世界安静了。

4. **DONE** emacs 绑定中文标点到英文标点。 NEEDIMPROVE:BUG:EIM 结果就只是修改了 py.txt 中的 Punctuation 部份。将 * 对应的内容修改了一下，然后使用 eim-build-table 重新构造了一下就行了。但是有一个 bug，这样只能每次输入一个 *。同时按两下 * 也不行。

   <2014-10-27 一 11:31>

5. **TODO** 我开始使用五笔输入法

   - State "TODO" from "" *[2014-12-20 六 11:07]*

   希望以后能配置成单字模式，我不想打词组，为的就是以后能盲打。

   也许以后可以在 eim 的基础上自己定制一个输入法。

1.3.5 init-global.el

init-global.el 是我的全局设置。其中包括一些对 emacs 本身就有的功能的一些配置，全局的按键绑定等等。

1. MISC

   (a) 不需要开启自动备份产生令人讨厌的 ~ 文件

```
(setq backup-inhibited t)
```

(b) 开启 desktop-save-mode 下一次进入 emacs 的时候继续访问上次访问的文件。有利于工作的重新开展：

```
(desktop-save-mode 1)
```

(c) 设置我的个人信息：

```
(setq user-full-name "pengpengxp")
(setq user-mail-address "pengpengxppri@gmail.com")
```

(d) 光标不要闪动：

```
(blink-cursor-mode -1)
```

(e) 开始不需要使用 menu-bar，scroll-bar 这些

```
(menu-bar-mode -1)
(when window-system
  (scroll-bar-mode -1))
```

(f) 不是 root 用户的时候开启 server：

```
(unless (string-equal "root" (getenv "USER"))
  (require 'server)
  (unless (server-running-p) (server-start)))
```

(g) 关闭 process 的时候不需要询问。

```
(setq kill-buffer-query-functions
      (remq 'process-kill-buffer-query-function
    kill-buffer-query-functions))
```

(h) 从王垠的配置中借鉴的 使用更大的 kill-ring。默认 emacs 滚动都是半屏半屏的滚动，不流畅，使 emacs 滚动更流畅一点。默认的 mode 设置成 text-mode。当光标移动过来鼠标自动躲避到右上角：

```
(setq kill-ring-max 200)
(setq scroll-margin 3
      scroll-conservatively 10000)
(setq default-major-mode 'text-mode)
(mouse-avoidance-mode 'banish)
```

(i) 默认显示时间，开启对匹配括号的提示：

```
(display-time)
(show-paren-mode t)
```

(j) F11 直接全屏：

```
(when window-system
  (progn
    (peng-global-set-key [f11] '(lambda ()
     (interactive)
     (set-frame-parameter nil 'fullscreen
  (if (frame-parameter nil 'fullscreen) nil 'fullboth))
     ;; ;; If you want the fullscreen emacs to be very minimal (no tool bar,
     (progn
       (if (fboundp 'tool-bar-mode) (tool-bar-mode -1))  ;; no toolbar
       (menu-bar-mode -1) ;;no menubar
       ;; (scroll-bar-mode -1) ;; no scroll bar
       )))))
```

(k) 开启 winner-mode，为它定义两个按键绑定：

```
(winner-mode 1)
(peng-global-set-key (kbd "C-c u") 'winner-undo)
(peng-global-set-key (kbd "C-c r") 'winner-redo)
```

(l) 开启 recentf-mode 记录最近打开的文件：

```
(recentf-mode 1)
```

(m) 回答 yes-or-no 的时候可以简单使用 y 或者 n。这个我还没有测试成功：

```
(setq yes-or-no-p 'y-or-n-p)
```

(n) 使用 registor 更加方便：

```
(peng-global-set-key (kbd "C-x SPC") 'point-to-register)
(peng-global-set-key (kbd "C-x j") 'jump-to-register)
```

PS：我在 bookmark 配置中设置 C-c j 跳转 bookmark

(o) 主题设置

```
(when window-system
  (load-theme 'misterioso nil nil)
  (enable-theme 'misterioso)
  )
```

(p) 显示列号

```
(setq column-number-mode t)
```

(q) 每次分割窗口的时候都水平切割 也就是除非手动，禁止上下分割窗口。我不是很喜欢上下分割的窗口。

```
(setq split-height-threshold nil)
(setq split-width-threshold 0)
```

(r) 加密文件 使用 easypg 加密文件，默认每次修改加密文件都需要输入密码，感觉很麻烦，去掉了。这样每次只有第一次打开文件和重启 emacs 才需要输入密码。安全性没有那么高。但是考虑到我的 emacs 也就只有我用了。所以感觉还好。

```
(setq epa-file-cache-passphrase-for-symmetric-encryption t)
```

(s) 自动 revert-buffer 当文件被其他编辑器修改以后，一般都是在 terminal 中使用 vim 修改。自动 revert-buffer。

```
(global-auto-revert-mode 1)
```

2. 按键绑定 因为 evil 已经成了我很依赖的插件。所以这里全局绑定基本都使用的时我自己定义的 peng-global-set-key。

   (a) F5

```
;;; f5-map use for compiling and eye protection
(define-prefix-command 'F5-map)
(global-set-key (kbd "<f5>") 'F5-map)
;;; for compile
(peng-global-set-key (kbd "<f5> <f5>") 'compile)
(peng-global-set-key (kbd "<f5> r") 'recompile)

;;;F5 for eye protected
(peng-global-set-key (kbd "<f5> ee") 'peng-eyerest-show-rest)
(peng-global-set-key (kbd "<f5> er") 'peng-eyerest-restart)
(peng-global-set-key (kbd "<f5> ep") 'peng-eyerest-pause)
(peng-global-set-key (kbd "<f5> ec") 'peng-eyerest-continue)
(peng-global-set-key (kbd "<f5> eg") 'peng-eye-gymnistic)
(peng-global-set-key (kbd "<f5> es") 'peng-eyerest-reset)
(peng-global-set-key (kbd "<f5> ek") 'peng-eyerest-kill)
;;;F5 for eye protected
```

   (b) F6 主要负责和系统交互

```
;;; f6-map use for calling the system applications
(define-prefix-command 'F6-map)
(global-set-key (kbd "<f6>") 'F6-map)
;;; 在当前文件夹快速打开文件管理器
(peng-global-set-key (kbd "<f6> e") '(lambda ()
  (interactive)
  (save-window-excursion
    (save-restriction
      (shell-command (concat "gnome-terminal -x thunar " default-directory))
```

```
(peng-global-set-key (kbd "<f6> n") '(lambda ()
  (interactive)
  (save-window-excursion
    (save-restriction
      (shell-command (concat "gnome-terminal -x nautilus " default-directory
;;; 在当前文件夹快速打开终端
(peng-global-set-key (kbd "<f6> t") '(lambda ()
  (interactive)
  (save-window-excursion
    (save-restriction
      (shell-command "gnome-terminal&")))))
```

(c) F8 主要的 prefix-key

```
;; f8-map the global key binding are all here
(define-prefix-command 'F8-map)
(global-set-key (kbd "<f8>") 'F8-map)

(peng-global-set-key (kbd "<f8> j") 'bookmark-jump)
(peng-global-set-key (kbd "<f8> w") 'save-buffer)
(peng-global-set-key (kbd "<f8> f") 'find-file)
(peng-global-set-key (kbd "<f8> d") 'kill-this-buffer)
(peng-global-set-key (kbd "<f8> q") 'kill-buffer-and-window)
(peng-global-set-key (kbd "<f8> r") 'recentf-open-files)
(peng-global-set-key (kbd "<f8> a") 'org-agenda)
(peng-global-set-key (kbd "<f8> s") 'peng-toggle-gnome-terminal)
(peng-global-set-key (kbd "<f8> <backspace>") 'delete-other-windows)
(peng-global-set-key (kbd "<f8> <return>") 'delete-window)
(peng-global-set-key (kbd "<f8> gg") 'peng-goto-scratch)
(peng-global-set-key (kbd "<f8> gn") 'peng-toggle-gnome-terminal)
(peng-global-set-key (kbd "<f8> go") 'peng-ibuffer-filter-org-mode)
(peng-global-set-key (kbd "<f8> ge") 'peng-ibuffer-filter-emacs-lisp-mode)
```

```
(peng-global-set-key (kbd "<f8> gd") 'peng-ibuffer-filter-dired-mode)
(peng-global-set-key (kbd "<f8> gc") 'peng-ibuffer-filter-c-mode)
(peng-global-set-key (kbd "<f8> gp") 'peng-ibuffer-filter-c++-mode)
(peng-global-set-key (kbd "<f8> <tab>") 'switch-to-buffer)
(peng-global-set-key (kbd "<f8> e") 'eshell)
(peng-global-set-key (kbd "<f8> x") 'execute-extended-command)
(peng-global-set-key (kbd "<f8> h k") 'describe-key)
(peng-global-set-key (kbd "<f8> h f") 'describe-function)
(peng-global-set-key (kbd "<f8> h v") 'describe-variable)
(peng-global-set-key (kbd "<f8> h r") 'info-emacs-manual)
(peng-global-set-key (kbd "<f8> 1") 'delete-other-windows)
(peng-global-set-key (kbd "<f8> 0") 'delete-window)
(peng-global-set-key (kbd "<f8> 2") 'split-window-below)
(peng-global-set-key (kbd "<f8> 3") 'split-window-right)
(peng-global-set-key (kbd "<f8> c u") 'winner-undo)
(peng-global-set-key (kbd "<f8> c r") 'winner-redo)
(peng-global-set-key (kbd "<f8> c m") 'shell-command)
(peng-global-set-key (kbd "<f8> c c") 'org-capture)
(peng-global-set-key (kbd "<f8> b") 'ibuffer)
(peng-global-set-key (kbd "<f8> <f8>") '(lambda ()
  (interactive)
  (switch-to-buffer (other-buffer))))
```

(d) F9 本来想用来使用 register 方便一点，但是好像使用的比较少。

```
(define-prefix-command 'F9-map)
(global-set-key (kbd "<f9>") 'F9-map)
(peng-global-set-key (kbd "<f9> <f9>") 'jump-to-register)
(peng-global-set-key (kbd "<f9> <f10>") 'point-to-register)
```

(e) MISC

```
;;; MISC
```

```
(peng-global-set-key (kbd "C-M-0") 'delete-window)
(peng-global-set-key (kbd "C-M-1") 'delete-other-windows)
(peng-global-set-key (kbd "C-M-2") 'split-window-below)
(peng-global-set-key (kbd "C-M-3") 'split-window-right)
;; (peng-global-set-key (kbd "<C-tab>") '(lambda ()
;;                                        (interactive)
;;                                        (switch-to-buffer (other-buffer))))
(peng-global-set-key (kbd "<C-tab>") 'other-window)
(peng-global-set-key (kbd "s-v") 'view-mode)
(peng-global-set-key (kbd "C-+") 'text-scale-increase)
(peng-global-set-key (kbd "C-=") 'text-scale-increase)
(peng-global-set-key (kbd "C--") 'text-scale-decrease)
(peng-global-set-key (kbd "\C-cn") 'autopair-mode)
(peng-global-set-key (kbd "C-x C-b") 'ibuffer)
(peng-global-set-key (kbd "<C-up>") 'enlarge-window)
(peng-global-set-key (kbd "<C-down>") 'shrink-window)
(peng-global-set-key (kbd "<C-left>") 'shrink-window-horizontally)
(peng-global-set-key (kbd "<C-right>") 'enlarge-window-horizontally)
```

1.3.6 init-powerline.el

我开始使用 powerline 好像是可以模拟 vim 中的状态栏。我觉得还是比较好看的。但是有一个问题就是这种情况下，使用 evil 就不能显示出 evil 的 state。不使用 powerline 的时候是可以看见的。

其实 evil 对应的 state 记录再 evil-mode-line-tag 这个变量之中。查阅源码 powerline-default-theme 函数其实就是对 mode-line-format 进行了一下设置。我把 powerline-default-theme 中所有的内容拷贝出来，在其中加入了作了一点点小小的改动，就能达到有 evil+powerline 状态栏正常显示 evil-state 的目的了。真得只是作了一点点的修改！

```
(setq-default mode-line-format
      '("%e"
```

```
(:eval
 (let* ((active (powerline-selected-window-active))
(mode-line (if active 'mode-line 'mode-line-inactive))
(face1 (if active 'powerline-active1 'powerline-inactive1))
(face2 (if active 'powerline-active2 'powerline-inactive2))
(separator-left (intern (format "powerline-%s-%s"
powerline-default-separator
(car powerline-default-separator-dir))))
(separator-right (intern (format "powerline-%s-%s"
 powerline-default-separator
 (cdr powerline-default-separator-dir))))
(lhs (list (powerline-raw "%*" nil 'l)
   ;; add by pengpengxp
   (powerline-raw evil-mode-line-tag nil 'l)
   (powerline-buffer-size nil 'l)
   (powerline-raw mode-line-mule-info nil 'l)
   (powerline-buffer-id nil 'l)
   (when (and (boundp 'which-func-mode) which-func-mode)
     (powerline-raw which-func-format nil 'l))
   (powerline-raw " ")
   (funcall separator-left mode-line face1)
   (when (boundp 'erc-modified-channels-object)
     (powerline-raw erc-modified-channels-object face1 'l))
   (powerline-major-mode face1 'l)
   (powerline-process face1)
   (powerline-minor-modes face1 'l)
   (powerline-narrow face1 'l)
   (powerline-raw " " face1)
   (funcall separator-left face1 face2)
   (powerline-vc face2 'r)))
```

```
(rhs (list (powerline-raw global-mode-string face2 'r)
   (funcall separator-right face2 face1)
   (powerline-raw "%4l" face1 'l)
   (powerline-raw ":" face1 'l)
   (powerline-raw "%3c" face1 'r)
   (funcall separator-right face1 mode-line)
   (powerline-raw " ")
   (powerline-raw "%6p" nil 'r)
   (powerline-hud face2 face1))))
   (concat (powerline-render lhs)
   (powerline-fill face2 (powerline-width rhs))
   (powerline-render rhs))))))
```

1.3.7 init-sql.el

使用 sql-indent.el 进行 indent：

```
(eval-after-load "sql"
  '(load-library "sql-indent"))
```

然后做了一些基本的按键绑定，设置了 sql-interactive-mode 中 evil 为 evil-emacs-state：

PS：其实这个 evil-emacs-state 可以统一到 init-evil.el 中设置

```
(defun peng-sql-mode ()
  (add-to-list 'ac-modes 'sql-mode)
  (linum-mode 1)
  (local-set-key (kbd "<f5>") 'sql-send-buffer)
  (local-set-key (kbd "<C-return>") 'peng-sql-send-line)
  (local-set-key (kbd "<f8> s r") 'sql-send-region)
  (local-set-key (kbd "<f8> s s") 'sql-send-string)
  (local-set-key (kbd "<f8> s l") 'peng-sql-send-line)
  (local-set-key (kbd "<f10>") 'sql-send-region)
```

```
  (local-set-key (kbd "<f9>") 'peng-sql-send-remain)
)
(add-hook 'sql-mode-hook 'peng-sql-mode)

;;sqli-mode:the interactive mode of sql
(defun peng-sql-interactive-mode ()
  (evil-emacs-state)
  (sql-set-product 'ansi)
  (sql-set-sqli-buffer-generally))
(add-hook 'sql-interactive-mode-hook 'peng-sql-interactive-mode)
```

1.3.8 init-font.el

为了要使得 org 中的 table 同时输入中文和英文的不产生混乱，必须使用大小对应配套的中英文字体。

PS：等宽字体。且一个中文和两个英文的宽度相同

PS：这个问题只有在 X-windows 中才存在

这两个字体我还是找了很久，英文是 Monaco-13。中文是苹果的 Hiragino Sans GB W3. 总的来说还是比较好看的。

```
;;; 在X-window下才这样设置字体
(when window-system
    (progn
      (set-frame-font "Monaco-13")
      (set-fontset-font t 'han (font-spec :family "Hiragino Sans GB W3" :size 20))
      )
  )
```

1.3.9 init-guide-key.el

emacs 中按键太多了，使用这个插件可以在按键有中提醒你。开启 guide-key-mode。设置 delay 为 0.5s。最后定义需要提醒的 prefix-key。

PS：这个功能我其实比较少用到

```
(guide-key-mode 1)
(setq guide-key/idle-delay 0.5)
(setq guide-key/guide-key-sequence '("C-c" "C-c C-x" "C-x" "C-x r" "," ",c" ",l" "z")
```

1.3.10 init-bookmark-bmemu-mode.el

在 init-evil.el 中已经定义了 bookmark 为 evil-emacs-state-mode。这里进行简单配置：bookmark 修改时候自动保存。定义一个按键绑定：

```
(setq bookmark-save-flag 1)             ;let bookmark auto-saved when I change it
(define-key global-map (kbd "C-c j") 'bookmark-jump)
```

1.3.11 init-dired.el

使用到现在，越来越喜欢在 emacs 中打开目录进行操作了。虽然偶尔还是会用用 nautilus，thunar 这些。但是真得已经越来越少了。dired-mode 经过配置之后，确实很强大。配置文件里面，有很多东西是我在网上找的。

1. peng-dired-do-copy NEEDTOBEIMPROVED 我自己定义的函数，原来的函数在拷贝多个文件之后，总还是标记这这些文件。我这个函数，可以把 from 目录下所有的标记都删除。但是 to 目录下这些拷贝过去的文件还是有 C 的标记。想想，其实 to 目录下面，这些标记确实没有必要删除。确实是比较有用的提醒。

   ```
   (defun peng-dired-do-copy ()
     (interactive)
     (progn
       (dired-do-copy)
       (dired-unmark-all-marks)
       ))
   ```

   希望的改进：拷贝以后，直接跳转到 to 目录去。

2. hook

```
(defun peng-dired-mode ()
  (hl-line-mode 1)
  (dired-details-install)
  (local-set-key (kbd "<tab>") 'dired-details-toggle)
  (local-set-key (kbd "C") 'peng-dired-do-copy)
  (define-key evil-emacs-state-map (kbd "M-<up>") 'dired-up-directory)
  )
(add-hook 'dired-mode-hook 'peng-dired-mode)
```

3. 增强插件

```
(require 'dired-isearch)
(require 'wdired)
(require 'dired)
(require 'dired+)                       ;增强dired
(require 'dired-details)                ;Dired详细信息
(require 'dired-details+)               ;Dired详细消息切
换
(require 'dired-x)
```

这是增强 dired 的一些配置，代码的注释基本能解释。其中 my-dired-omit-extensions 我经常使用。这个可以使 dired 自动忽略一些后缀名文件。很实用。;my-dired-omit-regexp 需要会实用 eamcs 的正则表达式。这个我还不是很会。

```
(setq dired-recursive-copies t)             ;可以递归的
进行拷贝
(setq dired-recursive-deletes t)            ;可以递归的
删除目录
(setq dired-recursive-deletes 'always)      ;删除东西时
不提示
(setq dired-recursive-copies 'always)       ;拷贝东西时
不提示
```

```
;; (toggle-dired-find-file-reuse-dir 1)          ;使用单
一模式浏览Dired
(setq dired-details-hidden-string "[ ... ] ") ;设置隐藏
dired里面详细信息的字符串
(setq dired-listing-switches "-aluh")         ;传给 ls 的
参数，这里就可以设置显示大小的方式
(setq directory-free-space-args "-Pkh")       ;目录空间选
项
(setq dired-omit-size-limit nil)              ;dired忽略
的上限
(setq dired-dwim-target t)                    ;Dired试着
猜处默认的目标目录
(setq my-dired-omit-status t)                 ;设置默认忽
略文件
(setq my-dired-omit-regexp "^\\.?#\\|^\\..*") ;设置忽略文
件的匹配正则表达式
(setq my-dired-omit-extensions '(".cache" ".o" ".elc"))   ;
设置忽略文件的扩展名列表
(add-hook 'dired-after-readin-hook '(lambda ()
      (progn
(require 'dired-extension)
(dired-sort-method)))) ;先显示目录，然后显示文件
(add-hook 'dired-mode-hook '(lambda ()
      (progn
(require 'dired-extension)
(dired-omit-method)))) ;隐藏文件的方法
```

设置文件的默认的打开模式，这个是我最喜欢的，直接在上面用!。爽死：

```
(setq dired-guess-shell-alist-user                    ;
设置文件默认打开的模式
```

```
      (list
;; 图书
(list "\\.chm$" '(concat
  "firefox chm:"            ;执行特定的命令
  (replace-regexp-in-string ;替换空格为%20
   " " "%20" (w3m-expand-file-name-as-url (dired-get-filename))) ;
用URL的模式解析文件名
  " -q"))
;; (list "\\.pdf$" "wine /data/Backup/WindowsTools/FoxitReader/FoxitReader.exe")
(list "\\.pdf$" "evince")
(list "\\.pdg$" "wine /data/Backup/WindowsTools/MiniPDG/pdgreader.exe")
;; 多媒体
;; (list (format "\\(%s\\)$" (emms-player-get emms-player-mplayer 'regex)) "mpla
(list "\\.\\(jpe?g\\|png\\)$" "gthumb" )
;; 网页
(list "\\.html?$" "firefox")
;; 压缩包
(list "\\.rar$" "unrar e -ad")
(list "\\.tar.bz2$" "tar jxvf")
(list "\\.gz$" "gzip -d")
(list "\\.mkv$" "smplayer")
(list "\\.rmvb$" "smplayer")
(list "\\.mp4$" "smplayer")
(list "\\.avi$" "smplayer")
(list "\\.doc$" "wps")
;; 其他
(list "\\.exe$" "wine")))
```

4. **TODO** 剩下的这些还没有理解 这些应该是 dired 有关于排序的。我还没有看到这里。中间还有一些注释掉的东西我也还没看。

```
(defvar one-key-menu-dired-sort-alist nil
```

```
  "The ‘one-key’ menu alist for DIRED-SORT.")

(setq one-key-menu-dired-sort-alist
      '(
(("s" . "Size") . dired-sort-size)
(("x" . "Extension") . dired-sort-extension)
(("n" . "Name") . dired-sort-name)
(("t" . "Modified Time") . dired-sort-time)
(("u" . "Access Time") . dired-sort-utime)
(("c" . "Create Time") . dired-sort-ctime)))

(defun one-key-menu-dired-sort ()
  "The ‘one-key’ menu for DIRED-SORT."
  (interactive)
  (require 'one-key)
  (require 'dired-sort)                 ;排序 dired 文件
  (one-key-menu "DIRED-SORT" one-key-menu-dired-sort-alist t))
```

1.3.12 init-yasnippet.el needtobeimproved

使用模版。刚开始的时候我使用默认配置挺好的。

```
(require 'yasnippet)
(yas-global-mode 1)
(setq ac-source-yasnippet nil)          ;hope to use yasnippet with auto-complete
```

Question

---

1. 但是后来发现，我在 org-mode-hook 中这样设置了以后，info 这些的 tab 也变成了 yax/expand。这样设置的 [tab] 和 <tab> 这些是有什么区别？

```
;;; ------------------------------------------------------------------------
```

```
;;; 如果yas-expand失败后，tab作用就是之前的，一般是org-cycle
;;; ----------------------------------------------------------------------
(add-hook 'org-mode-hook
  (let ((original-command (lookup-key org-mode-map [tab])))
    `(lambda ()
       (setq yas-fallback-behavior
     '(apply ,original-command))
       (local-set-key [tab] 'yas-expand))))
;;; ----------------------------------------------------------------------
```

解决办法

---

1. 现在暂时还是这样设置，然后 init-yasnippet.el 中先不开启 yas-global-mode，而是在需要的 mode 中通过他们的 hook 单独调用 yas-minor-mode 来开启：

```
(require 'yasnippet)
(yas-reload-all)
(setq ac-source-yasnippet nil)          ;hope to use yasnippet with auto-complete
```

1.3.13 init-header2.el

自动添加 header。包括对文件的描述，作者，创建和修改时间等。我自己作了一些小小的改动，比如我现在不想加入版权信息等等。

```
(require 'header2)
;;;; 这是参考header2.el中的代码，把这些定义成宏
(defsubst peng-email-address ()
  "Insert my own email address"
  (insert header-prefix-string "Email: " "pengpengxppri@gmail.com" "\n"))
```

```
(defsubst peng-header-author ()
  "Insert my own email address"
  (insert header-prefix-string "Author: " "pengpengxp" "\n"))

;;;; 根据原始的‘make-header-hook’自己定制了一下
(setq make-header-hook '(
 header-title
 header-blank
 header-file-name
 header-description
 peng-header-author
 peng-email-address
 ;; header-maintainer
 ;; header-copyright
 header-creation-date
 header-version
 header-modification-date
 ;; header-pkg-requires
 ;; header-modification-author
 ;; header-update-count
 ;; header-url
 ;; header-doc-url
 ;; header-keywords
 ;; header-compatibility
 ;; header-blank
 ;; header-lib-requires
 ;; header-end-line
 ;; header-commentary
 ;; header-blank
 ;; header-blank
```

```
 ;; header-blank
 ;; header-end-line
 ;; header-history
 ;; header-blank
 header-blank
 header-end-line
 ;; header-free-software
 header-code
 header-eof
 ))
(provide 'init-header2)
```

1.3.14 init-auto-insert.el

其实和 header2.el 的作用差不多，我使用 auto-insert 来自动添加一些我不想重复输入的代码，配合 header2 提供的 make-header 来生成 header 注释信息。感觉还不错。全局自动打开 auto-insert-mode，每次新建文件的时候会自动提示是否添加 auto-insert template。也可以在空白文件中显式使用 auto-insert 来添加。

问题

---

1. 不是新建的文件就不会添加，这是怎么实现的呢？
2. 老样子，latex-mode 的模版没有弄好。

```
;;; init-auto-insert.el ---
;;
;; Filename: init-auto-insert.el
;; Description:
;; Author: pengpengxp
;; Email: pengpengxppri@gmail.com
```

```
;; Created: 六 12月 20 11:37:54 2014 (+0800)
;; Version:
;; Last-Updated:
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;; Code:


(define-auto-insert 'org-mode '(nil "#+OPTIONS: ^:{}
#+STARTUP: showall
"))
(define-auto-insert 'c-mode '(nil "#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int args,char *argv[])
{

}"))

;; (add-hook 'find-file-hook 'auto-insert)
(auto-insert-mode 1)

;; 不要每次都问我是否需要添加
(setq auto-insert-query nil)

(provide 'init-auto-insert)

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

```
;;; init-auto-insert.el ends here
```

1.3.15 TODO wait for writing

1. init-cc-mode.el
2. init-w3m.el
3. init-helm.el
4. init-auctex.el
5. init-auto-complete.el
6. init-bm.el
7. init-compilation.el
8. init-cuda.el
9. init-emacs-lisp.el
10. init-eshell.el
11. init-evil-leader.el
12. init-flycheck.el
13. init-global.el
14. init-ibuffer.el
15. init-icicles.el
16. init-ido.el
17. init-Info-mode.el
18. init-latex.el
19. init-lusty-explorer.el

20. init-magit.el
21. init-mew.el
22. init-paredit.el
23. init-peng-copyfun.el
24. init-peng-prifun.el
25. init-plugins.el
26. init-scheme.el
27. init-shell-pop.el
28. init-shell-script.el
29. init-slime.el
30. init-smartparens.el
31. init-smex.el
32. init-sundry.el
33. init-tags.el
34. init-test.el
35. init-uniquify.el
36. init-weibo.el
37. init-window-numbering.el
38. init-macro.el