

Emacs 实践笔记

aborn

2018-04-13 23:16

Contents

1	基本数据类型	1
1.1	列表	1
1.1.1	关联列表 alist (Association Lists)	1
1.1.2	属性列表 plist (Property Lists)	2
1.1.3	对列表进行排序	3
2	函数	3
2.1	什么是函数?	3
2.2	函数定义	3
2.3	检查一个函数是否定义	3
2.4	函数参数	4
2.5	函数调用	4
2.5.1	funcall	4
2.5.2	apply	4
3	文件	4
3.1	文件及访问	4
3.1.1	打开文件	4
3.1.2	文件保存	5
3.2	文件基本信息函数	5
3.3	文件与目录	5
3.3.1	创建、复制和删除目录	6
3.4	文件名	6
3.4.1	文件名扩展	6

4	org 实践	6
4.1	org 模式简介	6
4.2	文档结构	6
4.2.1	目录结构	7
4.2.2	显示与隐藏	7
4.2.3	列表	7
4.2.4	块结构	7
4.3	表格	7
4.4	超链接	7
4.4.1	链接格式	8
4.5	待办事项	8
4.6	日程表 (Agenda View)	8
4.6.1	日程文件 (Agenda files)	8
4.6.2	分发按钮	9
4.6.3	内建 Agenda 视图	9
4.6.4	计划 Schedule	9
4.7	Org 快速记录	9
4.7.1	如何使用 org-capture?	9
4.7.2	org 条目复制与移动	9
4.7.3	记录模板	10
4.8	Org 的导出功能	11
4.8.1	导出的 Dispatcher	11
4.9	org-capture.el	11
5	书签	11
5.1	emacs 的书签功能	11
5.1.1	设置一个书签	11
5.1.2	列出保存的书签	11
5.1.3	跳转到一个书签	12
5.1.4	删除一个书签	12
5.1.5	保存书签	12
5.1.6	其他设置	13
5.1.7	bookmark+	13
6	dired 实践	13
6.1	dired 文件管理	13
6.1.1	常用命令	13
6.1.2	标记与操作	14
6.1.3	批量执行 Shell 命令	15
6.1.4	dired 的扩展	15

7	magit 实践	15
7.1	magit 模式简介	15
7.2	常用命令	15
7.3	分支操作	16
8	包管理	16
8.1	Emacs 的 Package-Mode	16
8.2	包列表	16

1 基本数据类型

1.1 列表

列表是由零个或者多个元素组成的序列，列表中的每个元素都可由任意的对象组成。

1.1.1 关联列表 alist (Association Lists)

关联列表是一种特殊的列表，它的每个元素都是一个点对构成，如下示例：

```
(setq alist-of-colors
  '((rose . red) (lily . white) (buttercup . yellow)))
```

关联列表可以用来记录 key-value 这样的 map 结构；对每个元素做 car 操作拿到 key，做 cdr 操作即拿到相关系的 value。

1. 关联列表操作

- (assoc key alist) 获取列表第一个 key 所关联的值；下面是一个例子：

```
ELISP> (assoc 'rose alist-of-colors)
(rose . red)
```

注意：这里用得比较是 equal 函数，如想用 eq 函数，请采用 (assq key alist) 这个函数

- (rassoc value alist) 获取列表第一个 value 为 **value** 所关联的值；
- (assoc-default key alist) 获取列表中第一个 key 为 **key** 的 value；

```
ELISP> (assoc-default 'rose alist-of-colors)
red
```

1.1.2 属性列表 plist (Property Lists)

属性列表是由成对元素 (paired elements) 组成的列表，每个元素对关联着一个属性的名及其对应属性值。下面是一个例子：

```
(pine cones numbers (1 2 3) color "blue")
```

这里 pine 关联其值为 cons，numbers 关联其值为 (1 2 3)，一般每个元素对的关联值是由 symbol 类型组成的。

1. 属性列表的操作

- (plist-get plist property) 返回属性列表中属性名为 property 的属性值：

```
ELISP> (setq pl '(pine cones numbers (1 2 3) color "blue"))
(pine cones numbers
  (1 2 3)
  color "blue")
ELISP> (plist-get pl 'pine)
cones
ELISP> (plist-get pl 'numbers)
(1 2 3)
```

- (plist-member plist property) 如果属性列表 plist 中含有属性 property，则返回 non-nil。
- (plist-put plist property value) 保存属性 property 及值 value 的属性对

```
(setq my-plist '(bar t foo 4))           ;; => (bar t foo 4)
(setq my-plist (plist-put my-plist 'foo 69))   ;; => (bar t foo 69)
(setq my-plist (plist-put my-plist 'quux '(a))) ;; => (bar t foo 69 quux (a))
```

1.1.3 对列表进行排序

对列表进行排序可以采用 sort 这个函数 (sort list predicate)。不过这个函数是有副作用的，这个函数调用后会改变原有 list 的结构。第三个参数 predicate 传入的是一个比较函数，它接收两个参数。如果是想递增排序，当第一个参数小于第二个参数时返回 non-nil，否则返回 nil。注意这个 sort 函数对 list 的排序，始终保持 car 部分不变。下面是一个例子：

```

ELISP> (setq nums '(1 3 2 6 5 4 0))
(1 3 2 6 5 4 0)
ELISP> (sort nums '<)
(0 1 2 3 4 5 6)
ELISP> nums
(1 2 3 4 5 6)

```

注意这里的 `nums` 排序后，的 `car` 与原来 `list` 的 `car` 是一样的。所以一般采用重新赋值的方式 `(setq nums (sort nums '<))`

2 函数

2.1 什么是函数？

函数是有传入参数的可计算的规则。计算的结果为函数返回值。大部分计算机语言里，函数是有一个名字的。从严格意义来说，`lisp` 函数是没有名字的。注意：函数也是一个 `lisp` 对象，把这个对象关联到一个 `symbol`，这个 `symbol` 就是函数名

2.2 函数定义

定义一个函数的语法如下：

```
defun name args [doc] [declare] [interactive] body. . .
```

2.3 检查一个函数是否定义

检查一个变量是否绑定到函数，`fboundp symbol`，还有一个函数 (`functionp OBJECT`)

```

(fboundp 'info)                ; t
(fboundp 'setq)                ; t
(fboundp 'xyz)                 ; nil
(functionp (lambda () (message "Anonymous Functions"))) ; t
(fboundp (lambda () (message "Anonymous Functions"))) ; *** Eval error ***

```

2.4 函数参数

有些参数是可选的，当用户没有传是，设置一个默认值，下面是一个例子：
 “‘`elisp (defun piece-meal/fun-option-parameter (a &optional b) (when (null b) (message "paramete b is not provided") (setq b "ddd"))) ; set to default value (message "a=%s, b=%s" a b)`’“

2.5 函数调用

函数的调用有两种方式，**`**funcall**`** 和 **`**apply**`**

2.5.1 funcall

funcall 它的语法如下：

```
funcall function &rest arguments
```

因为 funcall 本身是一个函数，因此 funcall 在调用前，它的所有参数都将事先做求值运算。注意参数 **function** 必须为一个 Lisp 函数或者原生函数，不能为 Special Forms 或者宏 (可以为 lamda 匿名函数)。

2.5.2 apply

apply 的语法如下：

```
apply function &rest arguments
```

它与 funcall 功能类似，唯一不同的是它的 arguments 是一个列表对象。

3 文件

3.1 文件及访问

文件是操作系统永久保存数据的单元，为了编辑文件，我们必要告诉 Emacs 去读取一个文件，并将文件的内容保存在一个 Buffer 里，这样 Buffer 与文件就关联在一起。下面介绍与文件访问相关的函数，由于历史原因这些函数的命令都是以 **find-** 开头的，不是以 **visit-** 开头。

3.1.1 打开文件

如果想在 buffer 里打开一个文件，其命令是 **find-file** (C-x C-f)。当文件已经在 buffer 中存在时，这个命令返回文件对应的 buffer。如果当前没有 buffer 对应文件，则，创建一个 buffer，并将其文件内容读到 buffer 中，并返回这个 buffer。字义如下：

```
find-file filename &optional wildcards
```

这个函数有一个对应的 hook 变量，叫 **find-file-hook** 它的值是一个函数列表。这些函数在文件被打开后依次执行。

3.1.2 文件保存

文件被载入到 `buffer` 后，我们可以对其进行修改；修改完了后，将内容保存回文件，其对应的函数为：

```
save-buffer &optional backup-option
```

文件保存对应有两个 `hook` 变量，为：**before-save-hook** 和 **after-save-hook** 分别表示保存前的 `hook` 函数列表和保存后的 `hook` 函数列表。与之类似的还有一个函数 **write-file**

```
write-file filename &optional confirm
```

这个函数的功能是将当前 `buffer` 的内容写入到 `filename` 对应的文件中，并将当前 `buffer` 与这个文件进行关联

3.2 文件基本信息函数

下面介绍一些与文件基本信息相关的函数

1. 文件是否存在

```
file-exists-p lename
```

与之类似的有：**file-readable-p**、**file-executable-p**、**file-writable-p**、**file-directory-p** 这几个函数。

3.3 文件与目录

判断文件是否在一个目录下，怎么做？

```
file-in-directory-p file dir
```

如果 `file` 是一个在目录 `dir` 或者 `dir` 子目录下的文件，则返回 `t`。如果 `file` 与 `dir` 处于同一目录，也返回 `t`。如果想列出一个目录下的所有文件，那就要用到 **directory-files** 这个函数，其定义如下：

```
directory-files directory &optional full-name match-regexp nosort
```

这个函数按字母顺序返回目录 `directory` 下的所有文件。参数 `full-name` 不为 `nil` 时，则返回每个文件的绝对路径，否则返回相对路径。`match-regexp` 如果不是 `nil`，该函数返回只与 `match-regexp` 相匹配的文件列表。`nosort` 如果不为 `nil`，则不按字母排序。

3.3.1 创建、复制和删除目录

对目录的创建、复制和删除都有相关的处理函数，下面一一介绍：

`make-directory dirname &optional parents`

make-directory 创建一个目录名为 `dirname` 的目录

3.4 文件名

下面介绍一些与文件名操作有关的函数

`file-name-directory lename`

file-name-directory 返回的文件名里的目录部分，如果文件名里没有包含目录部分，则返回 `nil`。与这个函数对应的一个函数为 **file-name-nondirectory**，它返回非目录部分。

3.4.1 文件名扩展

expand-file-name 这个函数将文件名转成绝对文件名：

`expand-file-name filename &optional directory`

如果 `directory` 参数存在，将 `filename` 作为其相对路径，否则使用 **default-directory** 变量。这个函数在写 `elisp` 代码时经常用到，下面是一些例子：

```
(expand-file-name "foo")
"/xcssun/users/rms/lewis/foo"
(expand-file-name "../foo")
"/xcssun/users/rms/foo"
(expand-file-name "foo" "/usr/spool/")  "/usr/spool/foo"
```

4 org 实践

4.1 org 模式简介

Emacs 的 `org-mode` 可用于记笔记、管理自己的待办事项 (TODO lists)，同时，也可用于管理项目。它是一个高效的纯文本编辑系统。

4.2 文档结构

Org 是基于 Outline-mode，并提供灵活的命令编辑结构化的文档。其文档结构语法跟 `markdown` 很类似。

4.2.1 目录结构

Org 的目录结构在每行最左边以星号标记，星号越多，标题层级越深。下面是一些例子：

```
\* 一级目录
\** 二级目录
\*** 三级目录
\* 另一个一级目录
```

4.2.2 显示与隐藏

目录结构下的内容可以隐藏起来，通常用采用 **TAB** 和 *S-TAB* 这两个命令来切换。

4.2.3 列表

Org 提供三种类型的列表：有序列表、无序列表和描述列表

1. 有序列表以 '1.' 或者 '1)'
2. 无序列表以 '-', '+' 或者 '*'
3. 描述列表

4.2.4 块结构

在 Org 文档中，加入代码块这种类型的块结构，都是采用 begin...end 这种模式，下面是一个例子：

```
\#+BEGIN_EXAMPLE
\#+END_EXAMPLE
```

4.3 表格

4.4 超链接

Org 模式提供了比较好用的超链接方式，可以链接到普通网页、文件、email 等。

4.4.1 链接格式

Org 模式支持两种链接，即，内部链接和外部链接。它们有相同的格式：

[[链接]][描述]] 或 当只有链接没有描述 [[链接]]

一旦链接编辑完成，在 org 模式下，只显示 **描述**部分，而不会显示整体（后一种是只显示链接）。为了编辑链接和描述，需要通过快捷键 **C-c C-l** 来完成（注意：编辑结束后按 Enter 完成修改操作）。

1. 内部链接 内部链接是指向当前文件的链接，它的链接格式：

[[#链接ID]]

其中 **链接 ID** 是文档中唯一的标识 ID

2. 外部链接 Org 支持的外部链接有很多中形式，如文件、网页、新闻组、电子邮件信息、BBDB 数据条目等。它们以一个短的标识字符串打头，紧接着是一个冒号，冒号后面没有空格字符。
3. 链接处理相关命令
 - org-store-link 保存当前位置的一个链接，以备后面插入使用
 - org-insert-link 插入链接，绑定的快捷键为 C-c C-l，如果光标正在一个链接上，那么 C-c C-l 的行为是编辑这个链接及其描述

4.5 待办事项

Org 模式用来管理自己的 TODO list 非常方便

4.6 日程表 (Agenda View)

我们可以用 Org 来安排自己的行程

4.6.1 日程文件 (Agenda files)

变量 org-agenda-files 保存了一个文件列表，这些文件用来记录日程，下面是一些操作函数：C-c [将当前文件加入到 agenda 文件列表最前页面
org-agenda-file-to-front C-c] 将当前文件从 agenda 文件列表中删除 org-remove-file

4.6.2 分发按键

默认采用 **C-c a**，接下的默认的命令有：

- a 创建一个日程
- t/T 创建一个 TODO items
- L 对当前文件生成 timeline

4.6.3 内建 Agenda 视图

4.6.4 计划 Schedule

用 org 来安排日程

- org-schedule 将当前 TODO 添加计划时间

4.7 Org 快速记录

有时候，突然想到一些待办事项，或者一些突发的灵感。这时，我们想用 emacs 快速记录它，Org-Capture 提供这个好用的功能。它的前身是 org-remember.el（注：从 org 8.0 开始，org-remember 被 org-capture）替代。

4.7.1 如何使用 org-capture?

快速记录的命令为 **M-x org-capture**，默认绑定的快捷键为 **C-c c**。当这个命令被调用后，你可以使用自己定义好的模板快速创建记录。一旦完成内容的输入，按下 **C-c C-c** (org-capture-finalize)，来完成。然后，你就能继续做你当下的事。如果想跳转到刚刚创建的记录的 buffer，用 **C-u C-c C-c** 来完成。如果想中途中止输入，只要按下 **C-c C-k** (org-capture-kill)。

4.7.2 org 条目复制与移动

有时候，我们想将当前的某条目转移到其他文件或者其他项目里。这时，我们会用到 org-copy 和 org-refile 这两个命令。它们对应的快捷键分别是 **C-c M-w** 及 **C-c C-w**。这里有一个问题是，目标文件如何配置？目录文件的配置由一个变量决定，**org-refile-targets**，我自己的配置如下：

```
(setq org-refile-targets
      '((nil :maxlevel . 3)          ;; 当前文件的最大层级
        (aborn-gtd-files :maxlevel . 3)))
```

注意：我这时将文件放在 aborn-gtd-files 文件列表里。

4.7.3 记录模板

记录的模板为一个列表变量，org-capture-templates，列表的每条记录由如下几段组成：

```
("t" "Todo" entry (file+headline (expand-file-name org-default-notes-file org-directory)
    "* TODO %?\n  创建于:%T  %i\n")
```

1. 快捷键 如例子中的那样，”t”表示对应按键 t 这个快捷键。它能帮助我们快速地选中哪条模板进行快速记录。
2. 描述 接下来是一段简单的描述
3. 类型 第三段表示类型，有五种类型：entry item checkitem table-line plain

- entry 普通的 Org 结点，保证目标文件为 org-mode 文件，插入的时候将作为目录结点的子结点

(如果没有，将做为顶级结点)；

- item 与 entry 类似，不同点在于它的目标文件可以为简单的纯文本文件；
- checkitem 复选条目；
- table-line 在目标文件中的第一个 table 中插入新行；
- plain 纯文本记录

4. 目标文件 第四个字段配置目标文件

5. 模板 第五个字段表示模板，模板参数 含义如下：

- %t 只有日期的时间戳
- %T 日期 + 时间的时间戳
- %u,%U 如上，只不过它们是 inactive 的
- %i 初始化文本，当前上下文将作为初始化文本

6. 属性 properties 最后一个字段表示属性列表，支持以下属性配置：

- :prepend 一般一个记录条目插入在目标文件的最后，这个属性可以将条目插入在最前
- :immediate-finish 立刻完成，没有交互
- :clock-in 对这个条目设置闹钟
- :kill-buffer 如果目标文件没有相应的访问 buffer, 插入后，自动关闭 buffer

4.8 Org 的导出功能

Org 文件支持导出多种格式的目标文件，如 ASCII 文件、HTML 文件 (用于发布为 Web)、PDF 文档等。

4.8.1 导出的 Dispatcher

任何导出命令都有一个前缀按键，我们称之为 Dispatcher，为 **C-c C-e**

4.9 org-capture.el

Org 8.0 以后版本采用 org-capture.el 取代原有的 org-remember.el

5 书签

5.1 emacs 的书签功能

emacs 的书签用于记录你在文件中的阅读位置。它有点类似寄存器，跟寄存器一样，因为它也能记录位置位置。但同寄存器有两点不一样：1. 它有更长的名字；2. 当 emacs 关闭的时候，它会自动持久化到磁盘。

5.1.1 设置一个书签

当我们阅读一个很长的文档，没能一口气读完时。我们希望记住当前文档的最后阅读的位置，以便下次再用 emacs 阅读的时候能快速定位到。那么，我们设置一个书签，通过 **bookmark-set** 对应快捷键为 **C-x r m**

5.1.2 列出保存的书签

bookmark-bmenu-list 对应快捷键为 **C-x r l**，它将打开一个 *Bookmark List* 的 buffer 同时列出所有保存的书签。

1. 书签列表 *Bookmark List* 在 *Bookmark List* 这个 buffer 里，有以下快捷键可以使用：
 - a 显示当前书签的标注信息；
 - A 在另一个 buffer 中显示所有书签的所有标注信息；
 - d 标记书签，以便用来删除 (x - 执行删除)；
 - e 编辑当前书签的标注信息；
 - m 标记书签，以便用于进一步显示和其他操作 (v - 访问这个书签)；

- o 选中当前书签，并显示在另一个 window 中;
- C-o 在另一个 window 中切换到当前这个书签;
- r 重命名当前书签;
- w 将当前书签的位置显示在 minibuffer 里。

5.1.3 跳转到一个书签

使用 **bookmark-jump** 函数，可以跳转到一个特定的书签，它绑定的快捷键为 **C-x r b**。如果你的 emacs 中安装了helm 这个插件，你也可以使用 **helm-bookmarks** 这个命令来快速查找书签，并跳转到书签位置。

1. helm-bookmarks 通过 helm-bookmarks 命令来查找并跳转书签如下图:
2. 修改默认排序 书签查找和跳转的时候，默认的书签排序是按字母排序的。如果想将最近访问的书签放在最前面，将下面代码添加到你的 emacs 配置文件中。

```
(defadvice bookmark-jump (after bookmark-jump activate)
  (let ((latest (bookmark-get-bookmark bookmark)))
    (setq bookmark-alist (delq latest bookmark-alist))
    (add-to-list 'bookmark-alist latest)))
```

5.1.4 删除一个书签

删除一个书签对应的命令为 **bookmark-delete**。

5.1.5 保存书签

最新版本 emacs (老版本的书签保存在 `~/.emacs.bmk`)，在退出的时候会自动保存书签。如果想手动保存书签的话，可以采用 **bookmark-save** 这个函数命令。默认的情况，emacs 会将书签保存在 **bookmark-default-file** 变量对应的文件中。在我的机器中，对应的文件如下：

```
ELISP> bookmark-default-file
"/Users/aborn/.emacs.d/.cache/bookmarks"
ELISP>
```

5.1.6 其他设置

有一个变量 **bookmark-save-flag**。如果这个变量的值为一个数值，它表示修改（或新增）多少次书签后，emacs 会自动保存书签到磁盘。当这个变量的值被设置为 1 时，每次对 bookmark 的改动，emacs 就会自动保存内容到磁盘相应位置（这样可以防止 emacs 突然 crash 时 bookmark 的丢失）。如果这个值设置为 nil，表示 emacs 不会主动保存 bookmark，除非用户手动调用 **M-x bookmark-save**。

5.1.7 bookmark+

bookmark+ 是对 bookmark 的一个扩展的包。它有更多的功能：

1. 原始的 bookmark 只能对文件位置记录,bookmark+ 对孤立的 buffer(没有关联文件的 buffer) 也能保存书签;
2. 支持对书签进行打 tag;
3. 对文档的某个区域保存为书签，而不仅仅是某个位置;
4. 记录了每个书签的访问次数，及最后一次的访问时间，可以基于它们排序;
5. 多个书签可以有相同的名字;
6. 可以对函数、变量等加书签。

更多功能请参考: <https://www.emacswiki.org/emacs/BookmarkPlus#Bookmark%2b>

6 dired 实践

6.1 dired 文件管理

dired 的全称为 Directory Edit，即目录编辑，是一个非常老的模式。是 Emacs 下的一个文件管理神器！进入当前文件的 dired 文件管理，*M-x dired*。

6.1.1 常用命令

1. 光标移动命令
 - n 下移
 - p 上移

2. 文件操作

- **C** 拷贝文件, dired-recursive-copies 变量决定了拷贝的类型, 一般为 top
- **D** 删除文件, 类似的有一个 dired-recursive-deletes 变量可以控制递归删除
- **R** 重命名或者移动文件
- **D** 删除文件或者目录
- **+** 创建目录
- **Z** gzip 压缩文件
- **w** 复制文件名 (**C-u** 则复制相对于 dired 当前目录的相对目录)
- **A** 对文件进行正则表达式搜索, 会在第一个匹配的地方停下, 然后使用 **M-**, 搜索下一个匹配。

3. 其他命令

- **RET** 打开文件或者目录
- **g** 刷新当前 dired buffer
- **k** 隐藏不想显示出来的文件
- **q** 退出

6.1.2 标记与操作

dired 可以对多个文件进行标记, 然后进行批量操作。一个典型的是采用 **d** 对当前文件打上删除标记, 然后使用 **x** 命令来删除所有标记的文件。

1. 标记操作命令

- **m** 以星标记当前文件
- ****** 标记所有可执行文件
- ***@** 标记所有符号链接
- ***/** 标记所有目录 (不包括. 和..)
- ***s** 标记所有文件 (不包括. 和..)
- ***.** 标记具有给定扩展名的文件
- **% m REGEXP <RET>** 或 *** % REGEXP <RET>** 标记所有匹配到给定的正则表达式的文件。
- **% g REGEXP <RET>** 标记所有文件内容匹配到给定的正则表达式的文件。

2. 其他标记相关命令

- `u` 去除当前行的标记
- `U` 去除所有标记

6.1.3 批量执行 Shell 命令

在 `dired` 模式下，可以对标记的文件批量执行 `shell` 命令（如果没有标记文件，则对当前文件执行 `shell`），运行命令 **`dired-do-shell-command`**（绑定的快捷键为 `!`），相应的它有一个对应的异步操作的命令 **`dired-do-async-shell-command`**（绑定的快捷键为 `&`）。

6.1.4 `dired` 的扩展

1. `diredful` `diredful` 可使得不同的文件显示不同的颜色，是一个非常好的扩展
2. `dired-icon` `dired-icon` 根据文件类型显示相应 `icon`

7 magit 实践

7.1 magit 模式简介

`magit` 是 `emacs` 下版本管理的强大武器

7.2 常用命令

- **`magit-dispatch-popup`** 命令分发器，在 `spacemacs` 里绑定到 **`M-m g m`**
- **`magit-diff`** 相当于 `git diff`，当进入 `diff-buffer` 后按 `g` 更新之
- **`magit-status`** 相当于 `git status`，进入 `status-buffer` 后按 `s` 添加文件或文件夹到本地仓库
- **`magit-checkout`** 切换分支
- **`magit-branch-and-checkout`** 从当前分支切一个新的分支

7.3 分支操作

常用的分支操作如下：

- (magit-branch-delete) **b k** 删除一个或多个（本地）分支
- (magit-branch-rename) **b r** 对当前 Branch 进行重命名
- (magit-get-current-branch) 获取当前分支名

8 包管理

8.1 Emacs 的 Package-Mode

当通过 `*M-x list-package*` 命令打开一个 `*Package*` 的 Buffer, 它有如下命令:

1. `i` 标识安装 (`u` 取消标识)
2. `x` 执行安装操作
3. `d` 标识删除 (`x` 执行删除操作)
4. `U` 标识要更新的 package
5. `~` 标识所有废弃包
6. `M-x package-autoremove` 删除那些无用的旧包

8.2 包列表

1. `elisp-slime-nav` 写 `elisp` 代码时, 可用于跳转到函数的定义