

基本数据类型

aborn

2016-12-21

Contents

1	Lisp 的数据类型	1
2	符号 (symbol)	1
3	列表	1
3.1	关联列表 alist (Association Lists)	1
3.1.1	关联列表操作	2
3.2	属性列表 plist (Property Lists)	2
3.2.1	属性列表的操作	2
3.3	对列表进行排序	3

1 Lisp 的数据类型

Lisp 的对象至少属于一种数据类型。Emacs 里最基础的数据类型称之为原始类型 (primitive type)，这些原始类型包括整型、浮点、cons、符号 (symbol)、字符串、数组、哈希表 (hash-table)、subr、二进制编码函数 (byte-code function)，再加上一些特殊的类型，如 buffer。同时，每种原始类型都有一个对应的函数去校验对象是否属于其类型。

2 符号 (symbol)

符号类型是一种命名对象，可以用作变量及函数名，其值为属性列表 (plist)。一个以冒号开头的符号类型称之为 keyword symbol，它常用于常量类型。

3 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。

3.1 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 操作即拿到相关关系的 value。

3.1.1 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
```

3.2 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。

3.2.1 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))
```

3.3 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 '<)**)