基本数据类型

aborn

2016-12-21

Contents

1 Lisp 的数据类型 **1**

2 标识类型 (Symbols) **1**
2.1 标识类型的组成 1
2.1.1 名字 1
2.1.2 变量值 1
2.1.3 函数 2
2.1.4 属性列表 2
2.2 定义标识类型 2
2.2.1 defvar 和 devconst 2
2.2.2 defun 2
2.2.3 defmacro 2
2.3 标识符操作函数 2
2.3.1 make-symbol 2
2.3.2 intern 3
2.4 标识符属性 3
2.4.1 get symbol property 3
2.4.2 put symbol property value 3
2.4.3 标准标识符属性 3

3 列表 **3**
3.1 关联列表 alist (Association Lists) 4
3.1.1 关联列表操作 4
3.2 属性列表 plist (Property Lists) 4
3.2.1 属性列表的操作 4
3.3 对列表进行排序 5

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 标识类型 (Symbols)

标识类型是一种有唯一名的 (命名) 对象。它常用于变量及函数名。判断一个对象是否为标识类型用 **symbolp object** 方法。

2.1 标识类型的组成

每个标识对象有四部分组成，每部分称之为单元，每个单元指向其他对象。

2.1.1 名字

即标识对象名，获取标识对象名函数为 (symbol-name symbol)

2.1.2 变量值

当标识对象作为变量时的值

2.1.3 函数

标识函数定义，函数单元可保存另一个标识对象、或者 keymap、或者一个键盘宏

2.1.4 属性列表

标识对象的属性列表 (plist)，获取属性列表函数为 (symbol-plist symbol)

注意，其中 **名字**为字符串类型，不可改变，其他三个组成部分可被赋值为任意 lisp 对象。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。

2.2 定义标识类型

定义标识对象是一种特殊的 lisp 表达式，它表示将标识类型用于特殊用途。

2.2.1 defvar 和 devconst

它们是一种特殊表达式（Special Forms），它定义一个标识作为全局变量。实际应用中往往使用 *setq*，它可以将任意变量值绑定到标识对象。

2.2.2 defun

用于定义函数，它的作用是创建一个 lambda 表达式，并将其存储在标识对象的函数单元里。

2.2.3 defmacro

定义标识符为宏，创建一个宏对象并前对象保存在函数单元里。

2.3 标识符操作函数

常见的与标识类型相关的函数有 **make-symbol** 和 **intern**

2.3.1 make-symbol

```
make-symbol name
```

这个函数返回一个新的标识对象，它的名字是 **name** (必须为字符串)

2.3.2 intern

```
intern name &optional obarray
```

这个函数返回一个被绑定的名字为 **name** 的标识对象。如果标识符不在变量 **obarray** 对应的对象数组 (obarray) 里，创建一个新的，并加入到对象对象数组里。当无 obarray 参数时，采用全局的对象数组 obarray。

2.4 标识符属性

标识符属性记录了标识符的额外信息，下面的函数是对标签符属性进行操作：

2.4.1 get symbol property

获取标识符属性为 property 的属性值，属性不存在返回 nil

2.4.2 put symbol property value

设置标识符属性 property 的值为 value，如果之前存在相同的属性名，其值将被覆盖。这个函数返回 value。下面是一些例子：

```
(put 'fly 'verb 'transitive)              ;; 'transitive
(put 'fly 'noun '(a buzzing little bug)) ;;  (a buzzing little bug)
(get 'fly 'verb)       ;;   transitive
(symbol-plist 'fly)   ;;  (verb transitive noun (a buzzing little bug))
```

2.4.3 标准标识符属性

下面列的一些标准标识符属性用于 emacs 的特殊用途

1. :advertised-binding 用于函数的 key 的绑定
2. interactive-form 用于交互函数，不要手工设置它，通过 **interactive** 特殊表达式来设置它
3. disabled 如果不为 nil，对应的函数不能作为命令
4. theme-face 用于主题设置

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 '<))**