

# 函数

aborn

2016-12-04

## Contents

<b>1</b>	<b>什么是函数？</b>	<b>1</b>
<b>2</b>	<b>定义函数</b>	<b>1</b>
2.1	检查一个函数是否定义 . . . . .	1
2.2	函数参数 . . . . .	2
<b>3</b>	<b>函数调用</b>	<b>2</b>
3.1	funcall . . . . .	2
3.2	apply . . . . .	2
3.3	映射函数 (Mapping Functions) . . . . .	3
<b>4</b>	<b>匿名函数</b>	<b>3</b>
4.1	lambda 宏 . . . . .	3
4.2	function 特殊表达式 . . . . .	4
<b>5</b>	<b>获取函数单元内容</b>	<b>4</b>
5.1	symbol-function . . . . .	4
5.2	fboundp . . . . .	4
<b>6</b>	<b>特殊表达式 (Special Forms) 和宏</b>	<b>5</b>
6.1	内建函数 (primitive) . . . . .	5
6.2	special form . . . . .	5

---

## 1 什么是函数？

函数是有传入参数的可计算的单元。每个函数的计算结果为函数返回值。大部分计算机语言里，每个函数有其自己函数名。从严格意义来说，lisp 函数

是没有名字的。lisp 函数其本质是一个对象，该对象可关联到一个标识符（本书把 Symbol 翻译成标识符），这个标识符就是函数名。

## 2 定义函数

定义一个函数的语法如下：

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

### 2.1 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 ***
```

### 2.2 函数参数

有些参数是可选的，当用户没有传时，设置一个默认值，下面是一个例子：

```
(defun cookbook/fun-option-parameter (a &optional b &rest e)
  (when (null b)
    (message "parameter b is not provided")
    (setq b "ddd")) ;; set to default value
  (message "a=%s, b=%s" a b))
```

函数 `cookbook/fun-option-parameter` 中，`a` 为必传参数，`b` 为可选择参数，`e` 为其余参数，当实际传入的参数大于 2 时，其他参数将组成一个 list 绑定到 `e` 上。

## 3 函数调用

最通用的函数的调用方式是对 list 进行求值，如对 list (`concat "a" "b"`) 进行求值，相当于用参数“a”和“b”调用函数 `concat`。这种方式用在你清楚程序上下文中调用哪个函数、传递哪个参数。但有时候你需要在程序运行时才决定调用哪个函数。针对这种情况，Emacs Lisp 提供了另外两种方式 `funcall` 和 `apply`。其中 `apply` 一般用在运行时行决定传递多少个参数的情况。

### 3.1 funcall

funcall 它的语法如下：

```
funcall function &rest arguments
```

这里 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，对 funcall 来说它不知道具体的求值过程。同时请注意第一个参数 **function** 必须为一个 Lisp 函数或者原生函数，不能为特殊表达式 (Special Forms) 和宏，但可以为匿名函数 (lambda 表达式)。

下面为一个例子：

```
(setq f 'list)          ;; list
(funcall f 'x 'y 'z)    ;; (x y z)
```

### 3.2 apply

apply 的定义如下：

```
apply function &rest arguments
```

apply 与 funcall 作用一样，唯独有一点不一样：它的 arguments 是一个对象列表，每个对象作为单独的参数传入，如下例子：

```
(setq f 'list)          ;; list
(apply f 'x 'y 'z)      ;; Wrong type argument: listp, z
(apply '+ 1 2 '(3 4))   ;; 10
```

### 3.3 映射函数 (Mapping Functions)

映射函数操作是指对一个列表或者集合逐个执行指定函数，这节介绍几个常的映射函数：mapcar, mapc, 和 mapconcat。

```
mapcar function sequence
```

这个函数功能有与 javascript 里的 array.map 操作类型，对 **sequence** 里的每个元素执行 function 操作，返回操作结果列表。这个函数应用非常广泛，以下几个应用举例：

```
(mapcar 'car '((a b) (c d) (e f)))  ;; (a c e)
(mapcar '1+ [1 2 3])                 ;; (2 3 4)
(mapcar 'string "abc")                ;; ("a" "b" "c")
```

`mapc` 与 `mapcar` 调用方式一样，唯一不同的点是它始终返回的是 `sequence`。

`mapconcat function sequence separator`

`mapconcat` 对 `sequence` 里的每个元素调用 `function` 最后将结果拼接成一个字符串作为返回值，采用 `separator` 作为拼接符。

## 4 匿名函数

在 `elisp` 里有三种方式可以定义匿名函数：`lambda` 宏、`function` 特殊表达式、`#'` 可读语法。

### 4.1 `lambda` 宏

它的定义如下：

```
lambda args [doc] [interactive] body. . .
```

这个宏返回一个匿名函数，实际上这个宏是自引用 (self-quoting)。

```
(lambda (x) (* x x)) ;; (lambda (x) (* x x))
```

下面是另一个例子：

```
(lambda (x)
  "Return the hyperbolic cosine of X."
  (* 0.5 (+ (exp x) (exp (- x)))))
```

上面的表达式被计算成一个函数对象。

### 4.2 `function` 特殊表达式

定义如下：

```
function function-object
```

这是一个特殊表达式 (Special Forms)，表示对 `function-object` 不作求值操作。其实在实际使用中我们往往采用它的简写 `#'`，因此下面三个是等价的：

```
(lambda (x) (* x x))
(function (lambda (x) (* x x)))
#'(lambda (x) (* x x))
```

## 5 获取函数单元内容

当我们把一个标识符 (Symbol) 定义为函数，其本质是将函数对象存储在标签符号对应的函数单元（标识符还有一个变量单元用于存储变量），下面是介绍函数单元处理方法：

### 5.1 symbol-function

定义如下：

```
symbol-function symbol
```

这个函数返回标识符 symbol 对应的函数对象，它不校验返回的函数是否为合法的函数。如果 symbol 的函数单元为空，返回 nil。

### 5.2 fboundp

用于判断 symbol 对应的函数单元是否为 nil

```
fboundp symbol
```

当 symbol 在函数单元有一个对象时返回 t，否则返回 nil。

## 6 特殊表达式 (Special Forms) 和宏

有些与函数看起来很像的类型，它们也接受参数，同时计算出结果。但在 Elisp 里，他们不被当成函数，下面给出简单介绍：

### 6.1 内建函数 (primitive)

是用 C 语言写的，可被调用的函数；

### 6.2 special form

一种类型的内建函数，如 if, and 和 while