第八章　求值规则

至此，elisp 中最常见的数据类型已经介绍完了。我们可以真正开始学习怎样写一个 elisp 程序。如果想深入了解一下 lisp 是如何工作的，不妨先花些时间看看 lisp 的求值过程。当然忽略这一部分也是可以的，因为我觉得这个求值规则是那么自然，以至于你会认为它就是应该这样的。

求值是 lisp 解释器的核心，理解了求值过程也就学会了 lisp 编程的一半。正因为这样，我有点担心自己说得不清楚或者理解错误，会误导了你。所以如果真想深入了解的话，还是自己看 info elisp - Evaluation 这一章吧。

一个要求值的 lisp 对象被称为表达式（ form）。所有的表达式可以分为三种：符号、列表和其它类型（废话）。下面一一说明各种表达式的求值规则。

第一种表达式是最简单的，自求值表达式。前面说过数字、字符串、向量都是自求值表达式。还有两个特殊的符号 t 和 nil 也可以看成是自求值表达式。

第二种表达式是符号。符号的求值结果就是符号的值。如果它没有值，就会出现 void-variable 的错误。

第三种表达式是列表表达式。而列表表达式又可以根据第一个元素分为函数调用、宏调用和特殊表达式（ special form）三种。列表的第一个表达式如果是一个符号，解释器会查找这个表达式的函数值。如果函数值是另一个符号，则会继续查找这个符号的函数值。这称为“ symbol function indirection”。最后直到某个符号的函数值是一个 lisp 函数（ lambda 表达式）、byte-code 函数、原子函数（ primitive function）、宏、特殊表达式或 autoload 对象。如果不是这些类型，比如某个符号的函数值是前面出现的某个符号导致无限循环，或者某个符号函数值为空，都会导致一个错误 invalid-function。

这个函数显示 indirection function：

```
(symbol-function 'car)                  ; => #<subr car>
(fset 'first 'car)                      ; => car
(fset 'erste 'first)                    ; => first
(erste '(1 2 3))                        ; => 1
```

对于第一个元素是 lisp 函数对象、byte-code 对象和原子函数时，这个列表也称为函数调用（ funtion call）。对这样的列表求值时，先对列表中其它元素先求值，求值的结果作为函数调用的真正参数。然后使用 apply 函数用这些参数调用函数。如果函数是用 lisp 写的，可以理解为把参数和变量绑定到函数后，对函数体顺序求值，返回最后一个 form 的值。

如果第一个元素是一个宏对象，列表里的其它元素不会立即求值，而是根据宏定义进行扩展。如果扩展后还是一个宏调用，则会继续扩展下去，直到扩展的结果不再是一个宏调用为止。例如：

```
(defmacro cadr (x)
  (list 'car (list 'cdr x)))
```

这样`(cadr (assq 'handler list))`扩展后成为`(car (cdr (assq 'handler list)))`。

第一个元素如果是一个特殊表达式时，它的参数可能并不会全求值。这些特殊表达式通常是用于控制结构或者变量绑定。每个特殊表达式都有对应的求值规则。这在下面会提到。

最后用这个伪代码来说明一下 elisp 中的求值规则：

```
(defun (eval exp)
  (cond
   ((numberp exp) exp)
   ((stringp exp) exp)
   ((arrayp exp) exp)
   ((symbolp exp) (symbol-value exp))
   ((special-form-p (car exp))
    (eval-special-form exp))
   ((fboundp (car exp))
    (apply (car exp) (cdr exp)))
   (t
    (error "Unknown expression type -- EVAL %S" exp))))
```