控制结构

aborn

2018-05-08

Contents

---

1 控制结构

Lisp 程序由一系列表达式结成，Lisp 解释器解释并执行这些表达式。在执行这些表达式过程中用到了控制结构，Lisp 里的控制结构都是特殊表达式(Special Forms)。最简单的控制结构是顺序执行，也是符合人的书写和线性习惯。其他控制结构有：条件语句、迭代。

2 顺序结构

顺序结构是最简单的控件结构，如果想自己定义顺序结构，可以采用 **progn** 这个特殊表达式:

```
(progn a b c ...)
```

它的执行结构是最后一句的结果。与之类似有另一个特殊表达式 **(prog1 form1 forms...)** 它也是顺序执行，不过它的返回值是 form1 的返回值。同时还有一个特殊表达式 **(prog2 form1 form2 forms...)** 它效果也是一样，不过它返回的是 form2 的值。

3 条件语句

ELisp 提供四种条件语句：if、when、unless 和 cond

3.1 if

if 语句跟其他语言的 if 语言类似，它的结构如下：

```
(if condition then-form else-forms...)
```

这里有一点要引起注意的是当 condition 为 nil, 并且没有给定 else-forms 时，if 返回的是 nil。

3.2 when

when 是 if 的变体，是当没有 else-forms 的特殊情况：

```
(when condition then-forms. . .)
```

3.3 unless

unless 也是 if 的一个变体，是当没有 then-form 的特殊情况：

```
(unless condition forms...)
```

3.4 cond

cond 是一种选择条件语句，每一个条件语句必须是一个列表，其中列表的头 (car clause) 是条件，列表的其他部分是执行语句。cond 的执行过程是按顺序执行，对每个条件语句 clause，先对条件部分进行求值，如果条件的执行结果不是 nil，说明条件满足，则接下来执行条件语句的主体部分，最后返回主体部分的执行结果作为 cond 的结果，其他部分的条件语句则被忽略。

```
(cond ((numberp x) x)
      ((stringp x) x)
      ((bufferp x)
       (setq temporary-hack x) ; multiple body-forms
       (buffer-name x))        ; in one clause
      ((symbolp x) (symbol-value x)))
```

有时候当前面所有的条件语句都没有“命中”时，可以采用 t 进行默认处理，下面是一个例子：

```
(setq a 5)
(cond ((eq a 'hack) 'foo)
      (t "default"))           ;;   "default"
```

4 迭代语句

迭代在程序语言里表示重复执行某段代码，举例来说，如果你想对 list 的每个元素重复执行相同的计算，这就是一个迭代过程。

4.1 while

while 的定义如下：

```
(while condition forms...)
```

while 首先对 condition 进行求值操作，如果结果不是 nil，则执行 forms 里语句；接下来再次对 condition 进行求值，如果不是 nil，则执行 forms 里的语句；这个过程不断重复直到 condition 的求值为 nil。

```
(setq num 0) ;;  0
(while (< num 4)
  (princ (format "Iteration %d." num))
  (setq num (1+ num)))
```

4.2 dolist

dolist 的定义如下：

```
dolist (var list [result]) body...
```

dolist 对 list 里的每个元素执行 body 里的语操作，这里绑定 list 里的每个元素到 var 作为局部变量。最后返回 result，当 result 省略时，返回 nil。下面是一个例子：

```
(defun reverse (list)
   (let (value)
    (dolist (elt list value)
      (setq value (cons elt value)))))
```

4.3 dotimes

dotimes 的定义如下：

```
dotimes (var count [result]) body...
```

它的作用与 dolist 很类似，它从 0(包含) 到 count(不包含) 执行 body 语句，将当前的值绑定到 var，返回 result 作为结果。下面是一个例子：

```
(dotimes (i 100)
 (insert "I will not obey absurd orders\n"))
```