Próximo:Geradores, Anterior:Condicional de correspondência de padrões, Acima:Estruturas de controle

[Conteúdo][Índice]

11.5 Iteração

Iteração significa executar parte de um programa repetidamente. Por exemplo, você pode querer repetir algum cálculo uma vez para cada elemento de uma lista, ou uma vez para cada inteiro de 0 a *n* . Você pode fazer isso no Emacs Lisp com o formulário especial `while`:

Forma Especial: enquanto *a condição se forma…*

`while`primeiro avalia a *condição* . Se o resultado for não- `nil`, ele avalia os *formulários* em ordem textual. Em seguida, ele reavalia a *condição* e, se o resultado for não- `nil`, avalia os *formulários* novamente. Esse processo se repete até que a *condição* seja avaliada como `nil`.

Não há limite para o número de iterações que podem ocorrer. O loop continuará até que uma das *condições* seja avaliada `nil`ou até que um erro ou `throw`salte dele (consulte Saídas não locais).

O valor de um `while`formulário é sempre `nil`.

```
(conjunto número 0)
     ⇒ 0
(enquanto (< num 4)
  (princ (formato "Iteração %d." num))
  (setq num (1+ num)))
     -| Iteração 0.
     -| Iteração 1.
     -| Iteração 2.
     -| Iteração 3.
     ⇒ nada
```

Para escrever um loop repeat-until, que irá executar algo em cada iteração e então fazer o end-test, coloque o corpo seguido do end-test em a `progn`como o primeiro argumento de `while`, conforme mostrado aqui:

```
(enquanto (prog
          (linha de avanço 1)
          (não (olhando para "^$")))))
```

Isso avança uma linha e continua se movendo por linhas até atingir uma linha vazia. É peculiar na medida em que o `while`não tem corpo, apenas o teste final (que também faz o trabalho real de mover o ponto).

As macros `dolist`e `dotimes`fornecem maneiras convenientes de escrever dois tipos comuns de loops.

Macro: dolist *(var list [resultado]) corpo…*

Essa construção executa *body* uma vez para cada elemento de *list* , vinculando a variável *var* localmente para manter o elemento atual. Em seguida, ele retorna o valor da avaliação de *result* , ou `nil`se *result* for omitido. Por exemplo, aqui está como você pode usar `dolist`para definir a `reverse`função:

```
(defun reverse (lista)
  (deixe (valor)
    (dolist (valor da lista elt)
      (valor setq (valor cons elt)))))
```

Macro: dotimes ***(var count [resultado]) corpo…***

Essa construção executa *body* uma vez para cada inteiro de 0 (inclusive) a *count* (exclusivo), vinculando a variável *var* ao inteiro para a iteração atual. Em seguida, ele retorna o valor da avaliação de *result* , ou `nil`se *result* for omitido. O uso do *resultado* está obsoleto. Aqui está um exemplo de uso `dotimes`para fazer algo 100 vezes:

```
(dotimes (i 100)
  (insira "Não obedecerei a ordens absurdas\n"))
```

Próximo:Geradores, Anterior:Condicional de correspondência de padrões, Acima:Estruturas de controle

[Conteúdo][Índice]