Próximo:Saídas não locais, Anterior:Iteração, Acima:Estruturas de controle [Conteúdo][Índice]

11.6 Geradores

Um *gerador* é uma função que produz um fluxo potencialmente infinito de valores. Cada vez que a função produz um valor, ela se suspende e espera que um chamador solicite o próximo valor.

Macro: iter-defun *name args [doc] [declare] [interactive] body…*

`iter-defun`define uma função geradora. Uma função geradora tem a mesma assinatura de uma função normal, mas funciona de forma diferente. Em vez de executar *body* quando chamado, uma função geradora retorna um objeto iterador. Esse iterador executa *o corpo* para gerar valores, emitindo um valor e pausando onde `iter-yield`dou `iter-yield-from`aparece. Quando *o corpo* retorna normalmente, `iter-next`sinaliza `iter-end-of-sequence`com o resultado do *corpo* como seus dados de condição.

Qualquer tipo de código Lisp é válido dentro de *body* , mas `iter-yield`não `iter-yield-from`pode aparecer dentro `unwind-protect`de formulários.

Macro: iter-lambda *args [doc] [interactive] body…*

`iter-lambda`produz uma função geradora sem nome que funciona exatamente como uma função geradora produzida com `iter-defun`.

Macro: *valor* de rendimento iterativo

Quando aparece dentro de uma função geradora, `iter-yield` indica que o iterador atual deve pausar e retornar o *valor* de `iter-next`. `iter-yield`avalia o `value`parâmetro da próxima chamada para `iter-next`.

Macro: iter-yield-from *iterator*

`iter-yield-from`produz todos os valores que o *iterador* produz e avalia o valor que a função geradora do *iterador retorna normalmente.* Enquanto tiver controle, o *iterador* recebe valores enviados ao iterador usando `iter-next`.

Para usar uma função geradora, primeiro chame-a normalmente, produzindo um objeto *iterador* . Um iterador é uma instância específica de um gerador. Em seguida, use `iter-next`para recuperar valores desse iterador. Quando não há mais valores para extrair de um iterador, `iter-next`gera uma `iter-end-of-sequence`condição com o valor final do iterador.

É importante observar que os corpos da função do gerador são executados apenas dentro de chamadas para `iter-next`. Uma chamada para uma função definida com `iter-defun`produz um iterador; você deve conduzir este iterador `iter-next`para que algo interessante aconteça. Cada chamada para uma função geradora produz um iterador *diferente , cada um com seu próprio estado.*

Função: *valor do iterador* seguinte

Recupere o próximo valor do *iterator* . Se não houver mais valores a serem gerados (porque a função geradora do *iterador* retornou), `iter-next`sinaliza a `iter-end-of-sequence` condição; o valor de dados associado a esta condição é o valor com o qual a função geradora do *iterador* retornou.

value é enviado para o iterador e se torna o valor para o qual `iter-yield`davalia. *value* é ignorado para a primeira `iter-next`chamada para um determinado iterador, pois no início da função generator do *iterator* , a função generator não está avaliando nenhum `iter-yield`formulário.

Função: iter-close *iterator*

Se o *iterador* estiver suspenso dentro de um `unwind-protect`'s `bodyforme` se tornar inacessível, o Emacs eventualmente executará manipuladores de desenrolamento após uma passagem de coleta de lixo. (Observe que `iter-yield`é ilegal dentro `unwind-protect`de um `unwindforms`.) Para garantir que esses manipuladores sejam executados antes disso, use `iter-close`.

Algumas funções de conveniência são fornecidas para facilitar o trabalho com iteradores:

Macro: iter-do *(var iterator) corpo …*

Execute *body* com *var* vinculado a cada valor que o *iterador* produz.

O recurso de loop Common Lisp também contém recursos para trabalhar com iteradores. Consulte Loop Facility em Common Lisp Extensions .

O trecho de código a seguir demonstra alguns princípios importantes do trabalho com iteradores.

```
(requer 'gerador)
(iter-defun meu-iter (x)
  (rendimento iter (1+ (rendimento iter (1+ x)))))
   ;; Retornar normalmente
  -1)

(let* ((iter (meu-iter 5)))
       (iter2 (meu-iter 0)))
  ;; Impressões 6
  (imprimir (iter-próximo iter))
  ;; Impressões 9
  (imprimir (iter-próximo iter 8))
  ;; Imprime 1; iter e iter2 têm estados distintos
  (imprimir (iter-próximo iter2 nil))

  ;; Esperamos que a sequência iter termine agora
  (condição-caso x
      (iter-próximo iter)
    (iter-fim-de-sequência
      ;; Imprime -1, que my-iter retornou normalmente
      (imprimir (cdr x))))))
```

Próximo:Saídas não locais, Anterior:Iteração, Acima:Estruturas de controle [Conteúdo][Índice]