Próximo:Compilando Macros, Anterior:Macro simples, Acima:Macros [Conteúdo][Índice]

14.2 Expansão de uma chamada de macro

Uma chamada de macro se parece com uma chamada de função, pois é uma lista que começa com o nome da macro. O restante dos elementos da lista são os argumentos da macro.

A avaliação da chamada da macro começa como a avaliação de uma chamada de função, exceto por uma diferença crucial: os argumentos da macro são as expressões reais que aparecem na chamada da macro. Eles não são avaliados antes de serem atribuídos à definição da macro. Por outro lado, os argumentos de uma função são resultados da avaliação dos elementos da lista de chamadas de função.

Tendo obtido os argumentos, Lisp invoca a definição de macro assim como uma função é invocada. As variáveis de argumento da macro estão vinculadas aos valores de argumento da chamada da macro ou a uma lista deles no caso de um `&rest`argumento. E o corpo da macro executa e retorna seu valor da mesma forma que o corpo da função.

A segunda diferença crucial entre macros e funções é que o valor retornado pelo corpo da macro é uma expressão Lisp alternativa, também conhecida como *expansão* da macro. O interpretador Lisp avalia a expansão assim que ela volta da macro.

Como a expansão é avaliada da maneira normal, ela pode conter chamadas para outras macros. Pode até ser uma chamada para a mesma macro, embora isso seja incomum.

Observe que o Emacs tenta expandir macros ao carregar um arquivo Lisp não compilado. Isso nem sempre é possível, mas se for, acelera a execução subsequente. Consulte Como os programas são carregados .

Você pode ver a expansão de uma determinada chamada de macro chamando `macroexpand`.

Função: macroexpandir *formulário e ambiente opcional*

> Esta função expande *form* , se for uma chamada de macro. Se o resultado for outra chamada de macro, ela é expandida por sua vez, até que resulte algo que não seja uma chamada de macro. Esse é o valor retornado por `macroexpand`. Se o *formulário* não for uma chamada de macro para começar, ele será retornado conforme fornecido.
>
> Observe que `macroexpand`não examina as subexpressões de *forma* (embora algumas definições de macro possam fazê-lo). Mesmo que sejam chamadas de macro, `macroexpand`não as expande.
>
> A função `macroexpand`não expande chamadas para funções embutidas. Normalmente não há necessidade disso, pois uma chamada para uma função inline não é mais difícil de entender do que uma chamada para uma função comum.
>
> Se *o ambiente* for fornecido, ele especificará uma lista de definições de macro que sombreiam as macros atualmente definidas. A compilação de bytes usa esse recurso.

```
(defmacro inc (var)
    (lista 'setq var (lista '1+ var)))


(macroexpandir '(inc r))
     ⇒ (setq r (1+ r))
```

```
(defmacro inc2 (var1 var2)
    (list 'progn (list 'inc var1) (list 'inc var2)))

(macroexpandir '(inc2 rs))
     ⇒ (progn (inc r) (inc s)) ; incnão expandido aqui.
```

Função: macroexpandir-todos os *formulários e ambiente opcional*

`macroexpand-all`expande macros como `macroexpand`, mas procurará e expandirá todas as macros no *formulário* , não apenas no nível superior. Se nenhuma macro for expandida, o valor de retorno `eq` deve *formar* .

Repetindo o exemplo usado `macroexpand`acima com `macroexpand-all`, vemos que `macroexpand-all` *expande* as chamadas incorporadas para `inc`:

```
(macroexpandir-tudo '(inc2 rs))
     ⇒ (progn (setq r (1+ r)) (setq s (1+ s)))
```

Função: formulário macroexpand-1 *e ambiente opcional*

Esta função expande macros como `macroexpand`, mas realiza apenas uma etapa da expansão: se o resultado for outra chamada de macro, `macroexpand-1`não a expandirá.

Próximo:Compilando Macros, Anterior:Macro simples, Acima:Macros [Conteúdo][Índice]