Próximo:Funções genéricas, Anterior:Funções de mapeamento, Acima:Funções [Conteúdo][Índice]

13.7 Funções Anônimas

Embora as funções geralmente sejam definidas com `defun`nomes e nomes ao mesmo tempo, às vezes é conveniente usar uma expressão lambda explícita — uma *função anônima* . As funções anônimas são válidas onde quer que os nomes das funções estejam. Eles são frequentemente atribuídos como valores de variáveis ou como argumentos para funções; por exemplo, você pode passar um como argumento de *função*`mapcar` para , que aplica essa função a cada elemento de uma lista (consulte Funções de mapeamento). Veja o exemplo describe-symbols , para um exemplo realista disso.

Ao definir uma expressão lambda que deve ser usada como uma função anônima, você pode, em princípio, usar qualquer método para construir a lista. Mas normalmente você deve usar a `lambda`macro, ou o `function`formulário especial, ou a `#'`sintaxe de leitura:

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

Essa macro retorna uma função anônima com lista de argumentos *args* , string de documentação *doc* (se houver), especificação *interativa interativa* (se houver) e formas de corpo fornecidas por *body* .

Sob vinculação dinâmica, essa macro efetivamente torna `lambda` os formulários autocitados: avaliar um formulário cujo CAR é `lambda` produz o próprio formulário:

```
(lambda (x) (* xx))
     ⇒ (lambda (x) (* xx))
```

Observe que ao avaliar sob vinculação léxica, o resultado é um objeto closure (consulte Closures).

O `lambda`formulário tem um outro efeito: ele diz ao avaliador do Emacs e ao compilador de bytes que seu argumento é uma função, usando `function`como uma sub-rotina (veja abaixo).

Forma especial: função *função-objeto*

Esta forma especial retorna *o objeto-função* sem avaliá-lo. Nisso, é semelhante a `quote`(consulte Cotação). Mas, diferentemente `quote`de , também serve como uma nota para o avaliador e compilador de bytes do Emacs que *o objeto-função* deve ser usado como uma função. Assumindo *que o objeto de função* é uma expressão lambda válida, isso tem dois efeitos:

- Quando o código é compilado por byte, *o objeto de função* é compilado em um objeto de função de código de byte (consulte Compilação de Byte).
- Quando a associação léxica está habilitada, *o objeto de função* é convertido em um encerramento. Consulte Fechamentos .

Quando *o objeto-função* é um símbolo e o código é compilado por byte, o compilador de bytes avisará se essa função não estiver definida ou não for conhecida em tempo de execução.

A sintaxe de leitura `#'`é uma abreviação para usar `function`. As seguintes formas são todas equivalentes:

```
(lambda (x) (* xx))
(função (lambda (x) (* xx)))
#'(lambda (x) (* xx))
```

No exemplo a seguir, definimos uma `change-property` função que recebe uma função como seu terceiro argumento, seguida por uma `double-property`função que faz uso `change-property`passando a ela uma função anônima:

```
(propriedade de mudança defun (função de prop de símbolo)
  (let ((valor (obter símbolo prop)))
    (coloque o símbolo prop (valor da função funcall))))


(propriedade dupla defun (propriedade do símbolo)
  (propriedade de símbolo de propriedade de mudança (lambda (x) (* 2 x))))
```

Observe que não citamos o `lambda`formulário.

Se você compilar o código acima, a função anônima também será compilada. Isso não aconteceria se, digamos, você tivesse construído a função anônima citando-a como uma lista:

```
(propriedade dupla defun (propriedade do símbolo)
  (alterar símbolo de propriedade prop '(lambda (x) (* 2 x))))
```

Nesse caso, a função anônima é mantida como uma expressão lambda no código compilado. O compilador de bytes não pode assumir que esta lista é uma função, mesmo que pareça com uma, pois não sabe que `change-property`pretende usá-la como uma função.

Próximo:Funções genéricas, Anterior:Funções de mapeamento, Acima:Funções [Conteúdo][Índice]