Próximo:Listas de propriedades, Anterior:Conjuntos e listas, Acima:Listas [Conteúdo][Índice]

5.8 Listas de Associações

Uma *lista de associações* , ou *simplesmente alist* , registra um mapeamento de chaves para valores. É uma lista de células cons chamadas *associações* : o CAR de cada célula cons é a *chave* e o CDR é o *valor associado* . [5]

Aqui está um exemplo de uma lista. A chave `pine`está associada ao valor `cones`; a chave `oak`está associada a `acorns`; e a chave `maple`está associada a `seeds`.

```
((pinhas)
 (carvalho. bolotas)
 (sementes de bordo.))
```

Tanto os valores quanto as chaves em uma lista podem ser quaisquer objetos Lisp. Por exemplo, na lista a seguir, o símbolo `a`está associado ao número `1`e a string `"b"`está associada à *lista* `(2 3)` , que é o CDR do elemento alista:

```
((a . 1) ("b" 2 3))
```

Às vezes é melhor projetar um alist para armazenar o valor associado no CAR do CDR do elemento. Aqui está um exemplo de tal lista:

```
((rosa vermelha) (lírio branco) (amarelo de botão de ouro))
```

Aqui consideramos `red`como o valor associado a `rose`. Uma vantagem desse tipo de lista é que você pode armazenar outras informações relacionadas – até mesmo uma lista de outros itens – no CDR do CDR . Uma desvantagem é que você não pode usar `rassq`(veja abaixo) para encontrar o elemento que contém um determinado valor. Quando nenhuma dessas considerações é importante, a escolha é uma questão de gosto, contanto que você seja consistente com qualquer lista.

A mesma lista mostrada acima pode ser considerada como tendo o valor associado no CDR do elemento; o valor associado `rose`seria a lista `(red)`.

As listas de associações costumam ser usadas para registrar informações que, de outra forma, você poderia manter em uma pilha, pois novas associações podem ser adicionadas facilmente à frente da lista. Ao pesquisar uma lista de associações por uma associação com uma determinada chave, a primeira encontrada é retornada, se houver mais de uma.

No Emacs Lisp, não é *um* erro se um elemento de uma lista de associações não for uma célula de contras. As funções de busca de lista simplesmente ignoram tais elementos. Muitas outras versões do Lisp sinalizam erros nesses casos.

Observe que as listas de propriedades são semelhantes às listas de associações em vários aspectos. Uma lista de propriedades se comporta como uma lista de associações na qual cada chave pode ocorrer apenas

uma vez. Consulte Listas de propriedades , para uma comparação de listas de propriedades e listas de associações.

Função: assoc *key alist & opcional testfn*

Esta função retorna a primeira associação para *key* em *alist* , comparando *key* com os elementos de alist usando *testfn* se for non- `nil` `equal`caso contrário (consulte Equality Predicates). Retorna `nil`se nenhuma associação em uma lista tiver um *CAR* igual a *chave* . Por exemplo:

```
(setq árvores '((pinheiro . cones) (carvalho . bolotas) (bordo . sementes)
     ⇒ ((pinheiro . pinhas) (carvalho . bolotas) (bordo . sementes))
(assoc 'carvalhos)
     ⇒ (carvalho . bolotas)
(cdr (assoc 'carvalhos))
     ⇒ bolotas
(assoc 'árvores de bétula)
     ⇒ nada
```

Aqui está outro exemplo, no qual as chaves e os valores não são símbolos:

```
(setq agulhas por cluster
      '((2 "Pinho Austríaco" "Pinho Vermelho")
        (3 "Pitch Pine")
        (5 "White Pine")))

(cdr (associação de 3 agulhas por cluster))
     ⇒ ("Pitch Pine")
(cdr (associação de 2 agulhas por cluster))
     ⇒ ("Pinho Austríaco" "Pinho Vermelho")
```

A função `assoc-string`é muito parecida `assoc`, exceto que ela ignora certas diferenças entre strings. Consulte Comparação de Texto .

Função: lista de *valores* rassoc

Esta função retorna a primeira associação com valor *value* em *alist* . Ele retorna `nil`se nenhuma associação em uma lista tiver um *CDR* para *value* . `equal`

`rassoc`é como `assoc`, exceto que compara o CDR de cada associação *alista* em vez do CAR . Você pode pensar nisso como reverse `assoc`, encontrando a chave para um determinado valor.

Função: lista de *teclas* assq

Esta função é semelhante `assoc`a que retorna a primeira associação para *key* em *alist* , mas faz a comparação usando `eq`. `assq`retorna `nil`se nenhuma associação em uma lista tiver um *CAR* para *chave* . Esta função é usada com mais frequência do que , pois é mais rápida do que e a maioria dos listas usa símbolos como teclas. Consulte Predicados de igualdade . `eq``assoc``eq``equal`

```
(setq árvores '((pinheiro . cones) (carvalho . bolotas) (bordo . sementes)
     ⇒ ((pinheiro . pinhas) (carvalho . bolotas) (bordo . sementes))
(assq 'pinheiros)
     ⇒ (pinhas . pinhas)
```

Por outro lado, `assq`geralmente não é útil em listas onde as teclas podem não ser símbolos:

```
(seq folhas
      '(("folhas simples" . carvalho)
        ("folhas compostas" . castanha-da-índia)))

(assq "folhas simples" folhas)
     ⇒ Não especificado; pode ser nilou ("simple leaves" . oak).
(assoc "folhas simples" folhas)
     ⇒ ("folhas simples" . carvalho)
```

Função: alist-get *key alist &opcional default remove testfn*

Esta função é semelhante a `assq`. Ele encontra a primeira associação comparando *key* com elementos *alist* e, se encontrada, retorna o *valor* dessa associação. Se nenhuma associação for encontrada, a função retornará *default* . A comparação de *chave* com elementos *alist* usa a função especificada por *testfn* , padronizando para . `(`*`key . value`*`)eq`

Esta é uma variável generalizada (consulte Variáveis Generalizadas) que pode ser usada para alterar um valor com `setf`. Ao usá-lo para definir um valor, o argumento opcional *remove* non `nil` -significa remover a associação da *chave de uma lista* se o novo valor for `eql`o *padrão* .

Função: lista de *valores* rassq

Esta função retorna a primeira associação com valor *value* em *alist* . Ele retorna `nil`se nenhuma associação em uma lista tiver um *CDR* para *value* . `eq`

`rassq`é como `assq`, exceto que compara o CDR de cada associação *alista* em vez do CAR . Você pode pensar nisso como reverse `assq`, encontrando a chave para um determinado valor.

Por exemplo:

```
(setq árvores '((pinheiro . cones) (carvalho . bolotas) (bordo . sementes)

(rassq 'árvores de bolotas)
     ⇒ (carvalho . bolotas)
(árvores de esporos rassq)
     ⇒ nada
```

`rassq`não pode procurar um valor armazenado no CAR do CDR de um elemento:

```
(setq cores '((rosa vermelha) (lírio branco) (amarelo de botão de ouro)))

(rassq 'cores brancas)
     ⇒ nada
```

Neste caso, o CDR da associação `(lily white)`não é o símbolo `white`, mas sim a lista `(white)`. Isso fica mais claro se a associação for escrita em notação de par pontilhado:

```
(lírio branco) ≡ (lírio . (branco))
```

Função: lista de teclas assoc-default *e padrão de teste opcional*

Esta função procura em uma lista *uma* correspondência para *key* . Para cada elemento de *alist* , ele compara o elemento (se for um átomo) ou o CAR do elemento (se for um contra) com *key* , chamando *test* com dois argumentos: o elemento ou seu CAR e *key* . Os argumentos são passados nessa ordem para que você possa obter resultados úteis usando uma lista que contém expressões regulares (consulte Pesquisa Regexp). Se *test* for omitido ou , será usado para comparação. `string-matchnilequal`

Se um elemento alist corresponder *a chave* por esse critério, `assoc-default`retornará um valor com base nesse elemento. Se o elemento for um contra, o valor será o CDR do elemento . Caso contrário, o valor de retorno é *padrão* .

Se nenhum elemento de lista corresponder *a key* , `assoc-default`retorna `nil`.

Função: *lista* de cópias

Esta função retorna uma cópia profunda de dois níveis de *alist* : ela cria uma nova cópia de cada associação, para que você possa alterar as associações do novo alist sem alterar o antigo.

```
(setq agulhas por cluster
      '((2 . ("Pinho Austríaco" "Pinho Vermelho"))
        (3. ("Pitch Pine"))
        (5 . ("White Pine"))))
⇒
((2 "Pinho Austríaco" "Pinho Vermelho")
 (3 "Pitch Pine")
 (5 "Pinheiro Branco"))

(setq copy (agulhas de lista de cópias por cluster))
⇒
((2 "Pinho Austríaco" "Pinho Vermelho")
 (3 "Pitch Pine")
 (5 "Pinheiro Branco"))

(eq cópia de agulhas por cluster)
     ⇒ nada
(cópia de agulhas por cluster iguais)
     ⇒ t
(eq (agulhas de carro por cluster) (cópia de carro))
     ⇒ nada
(cdr (carro (cdr agulhas por cluster)))
     ⇒ ("Pitch Pine")
(eq (cdr (carro (cdr agulhas por cluster)))
    (cdr (carro (cópia de cdr))))
     ⇒ t
```

Este exemplo mostra como `copy-alist`é possível alterar as associações de uma cópia sem afetar a outra:

```
(setcdr (cópia assq 3) '("Martian Vacuum Pine"))
(cdr (assq 3 agulhas por cluster))
     ⇒ ("Pitch Pine")
```

Função: lista de *chaves* assq-delete-all

Esta função exclui de uma *lista* todos os elementos cujo CAR deve ser `eq`a *tecla* , como se você `delq`excluísse cada um desses elementos um por um. Ele retorna o alist abreviado e geralmente

modifica a estrutura de lista original de *alist* . Para resultados corretos, use o valor de retorno de `assq-delete-all`em vez de consultar o valor salvo de *alist* .

```
(setq alist (lista '(foo 1) '(bar 2) '(foo 3) '(perde 4)))
     ⇒ ((foo 1) (bar 2) (foo 3) (perde 4))
(assq-delete-all 'foo alist)
     ⇒ ((bar 2) (perder 4))
uma lista
     ⇒ ((foo 1) (bar 2) (perde 4))
```

Função: lista de teclas assoc-delete-all *& teste opcional*

Esta função é semelhante `assq-delete-all`, exceto que aceita um *teste* de argumento opcional , uma função de predicado para comparar as chaves em uma *lista* . Se omitido ou `nil`, *o teste* assume como padrão `equal`. Como `assq-delete-all`, esta função geralmente modifica a estrutura de lista original de *alist* .

Função: rassq-delete-all *value alist*

Esta função exclui de uma *lista* todos os elementos cujo CDR deve eqter *valor* . Ele retorna o alist abreviado e geralmente modifica a estrutura de lista original de *alist* . `rassq-delete-all`é como `assq-delete-all`, exceto que compara o CDR de cada associação *alista* em vez do CAR .

Macro: *corpo do alista* let-alist

Cria uma ligação para cada símbolo usado como chaves na lista de associações *alist* , prefixada com ponto. Isso pode ser útil ao acessar vários itens na mesma lista de associações e é melhor compreendido por meio de um exemplo simples:

```
(setq cores '((rosa . vermelho) (lírio . branco) (buttercup . amarelo)))
(cores letalistas
  (if (eq .rose 'red)
      .lírio))
=> branco
```

O *corpo* é inspecionado no momento da compilação, e apenas os símbolos que aparecem no *corpo* com um '.' como o primeiro caractere no nome do símbolo será vinculado. A localização das chaves é feita com `assq`, e o `cdr`valor de retorno this `assq`é atribuído como o valor da associação.

As listas de associações aninhadas são suportadas:

```
(setq colors '((rosa . vermelho) (lírio (beladona . amarelo) (brindisi . rosa))))
(cores letalistas
  (if (eq .rose 'red)
      .lily.belladonna))
=> amarelo
```

O aninhamento `let-alist`um dentro do outro é permitido, mas o código no interno `let-alist`não pode acessar as variáveis vinculadas pelo externo `let-alist`.

Notas de rodapé

(5)

Esse uso de “chave” não está relacionado ao termo “sequência de teclas”; significa um valor usado para pesquisar um item em uma tabela. Nesse caso, a tabela é a lista e as associações de lista são os itens.

Próximo:Listas de propriedades, Anterior:Conjuntos e listas, Acima:Listas [Conteúdo][Índice]