Anterior:Configurando Variáveis Generalizadas, Acima:Variáveis Generalizadas [Conteúdo][Índice]

12.17.2 Definindo novos `setf`formulários

Esta seção descreve como definir novos formulários que `setf`podem operar.

Macro: gv-define-simple-setter *name setter e fix-return opcional*

Essa macro permite definir facilmente `setf`métodos para casos simples. *name* é o nome de uma função, macro ou formulário especial. Você pode usar esta macro sempre que *name tiver uma função setter* diretamente correspondente que a atualize, por exemplo, `(gv-define-simple-setter car setcar)`.

Esta macro traduz uma chamada do formulário

```
(setf ( nome  args …) valor )
```

para dentro

```
( setter  args … valor )
```

Essa `setf`chamada é documentada para retornar *valor* . Isso não é problema com, por exemplo, `care setcar`, porque `setcar`retorna o valor que ele definiu. Se sua função *setter* não retornar *value* , use um non- `nil`value para o argumento *fix-return*`gv-define-simple-setter` de . Isso se expande para algo equivalente a

```
(deixe (( valor temporário ))
  ( setter  args ... temp)
  temperatura)
```

garantindo assim que retorna o resultado correto.

Macro: gv-define-setter *nome arglist & resto corpo*

Esta macro permite `setf`expansões mais complexas do que a forma anterior. Você pode precisar usar este formulário, por exemplo, se não houver uma função setter simples para chamar, ou se houver uma, mas ela exigir argumentos diferentes para o formulário de lugar.

Essa macro expande o formulário primeiro vinculando os formulários de argumento de acordo com *arglist* e, em seguida, executando *body* . *body* deve retornar um formulário Lisp que faz a atribuição e, finalmente, retorna o valor que foi definido. Um exemplo de uso desta macro é: `(setf (`*name args*`…) `*value*`)setf(`*value args*`…)`

```
(gv-define-setter caar (val x) `(setcar (car ,x) ,val))
```

Macro: *manipulador de nomes* gv-define-expander

Para mais controle sobre a expansão, a `gv-define-expander` macro pode ser usada. Por exemplo, um settable `substring`pode ser implementado desta forma:

```
(substring gv-define-expander
  (lambda (coloque de &opcional para)
    (gv-letplace (getter setter) lugar
      (macroexp-let2* nil ((começar de) (finalizar))
        (funcall do `(substring ,getter ,start ,end)
                 (lambda (v)
                   (funcall setter `(cl--set-substring
                                     ,getter ,começo ,fim ,v))))))))
```

Macro: gv-letplace ***(getter setter) place & rest body***

A macro `gv-letplace`pode ser útil na definição de macros com desempenho semelhante ao `setf`; por exemplo, a `incf`macro do Common Lisp poderia ser implementada desta forma:

```
(defmacro incf (lugar &opcional n)
  (gv-letplace (getter setter) lugar
    (macroexp-let2 nil v (ou n 1)
      (funcall setter `(+ ,v ,getter)))))
```

getter será vinculado a uma expressão copiável que retorna o valor de *place* . *setter* será vinculado a uma função que recebe uma expressão *v* e retorna uma nova expressão que define *place* como *v* . *body* deve retornar uma expressão Emacs Lisp manipulando o *lugar* via *getter* e *setter* .

Consulte o arquivo de origem`gv.el`para mais detalhes.

Nota de Common Lisp: Common Lisp define outra maneira de especificar o `setf`comportamento de uma função, ou seja, `setf`funções, cujos nomes são listas em vez de símbolos. Por exemplo, define a função que é usada quando é aplicada a . O Emacs não suporta isso. É um erro de tempo de compilação para usar em um formulário que ainda não tenha uma expansão apropriada definida. Em Common Lisp, isso não é um erro, pois a função pode ser definida posteriormente. `(setf` *`name`*`)` `(defun (setf foo) …)`setffoosetf`(setf` *`func`*`)`

Anterior:Configurando Variáveis Generalizadas, Acima:Variáveis Generalizadas [Conteúdo][Índice]