Próximo:Valor padrão, Anterior:Introdução ao Buffer-Local, Acima:Variáveis locais de buffer [Conteúdo][Índice]

12.11.2 Criando e Excluindo Ligações Buffer-Local

Comando: variável make-local- ***variable***

Esta função cria uma ligação buffer-local no buffer atual para *variável* (um símbolo). Outros buffers não são afetados. O valor retornado é *variável* .

O valor de buffer-local da *variável* começa como a mesma *variável* de valor que tinha anteriormente. Se *a variável* foi nula, ela permanece nula.

```
;; Em buffer 'b1':
(setq foo 5); Afeta todos os buffers.
     ⇒ 5
(fazer variável local 'foo) ; Agora é local em 'b1'.
     ⇒ fo
fo; Isso não mudou
     ⇒ 5 ;   O valor que.
(setq foo 6); Altere o valor
     ⇒ 6 ;   dentro 'b1'.
foo
     ⇒ 6


;; Em buffer 'b2', o valor não mudou.
(com buffer de corrente "b2"
  foo)
     ⇒ 5
```

Fazer uma variável buffer-local dentro de uma `let`-binding para essa variável não funciona de maneira confiável, a menos que o buffer no qual você faz isso não esteja atualizado na entrada ou na saída do arquivo `let`. Isso ocorre porque `let`não faz distinção entre diferentes tipos de ligações; ele sabe apenas para qual variável a ligação foi feita.

É um erro fazer uma constante ou uma variável somente leitura buffer-local. Consulte Variáveis Constantes .

Se a variável for terminal-local (veja Múltiplos Terminais), esta função sinaliza um erro. Essas variáveis também não podem ter ligações de buffer-local.

Atenção: não use `make-local-variable`para uma variável de gancho. As variáveis de gancho são automaticamente feitas como buffer-local conforme necessário se você usar o argumento *local*`add-hook` para ou `remove-hook`.

Macro: pares setq-local ***&rest***

pairs é uma lista de pares de variáveis e valores. Essa macro cria uma ligação local de buffer no buffer atual para cada uma das variáveis e fornece a elas um valor local de buffer. É equivalente a chamar `make-local-variable`seguido por `setq`para cada uma das variáveis. As variáveis devem ser símbolos sem aspas.

```
(setq-local var1 "valor1"
            var2 "valor2")
```

Comando: variável make- *variable* -buffer-local

Essa função marca a *variável* (um símbolo) automaticamente como buffer-local, de modo que qualquer tentativa subsequente de defini-la a tornará local para o buffer atual no momento. Ao contrário `make-local-variable`de , com o qual muitas vezes é confundido, isso não pode ser desfeito e afeta o comportamento da variável em todos os buffers.

Um problema peculiar desse recurso é que vincular a variável (com `let`ou outras construções de vinculação) não cria uma vinculação local de buffer para ela. Apenas definindo a variável (com `set`ou `setq`), enquanto a variável não tiver uma `let`ligação -style que foi feita no buffer atual, faz isso.

Se a *variável* não tiver um valor padrão, chamar esse comando fornecerá um valor padrão de `nil`. Se a *variável* já tiver um valor padrão, esse valor permanece inalterado. A chamada subsequente `makunbound`da *variável* resultará em um valor local de buffer nulo e deixará o valor padrão inalterado.

O valor retornado é *variável* .

É um erro fazer uma constante ou uma variável somente leitura buffer-local. Consulte Variáveis Constantes .

Aviso: Não assuma que você deve usar `make-variable-buffer-local`para variáveis de opção do usuário, simplesmente porque os usuários *podem* querer personalizá-las de forma diferente em diferentes buffers. Os usuários podem tornar qualquer variável local, quando quiserem. É melhor deixar a escolha para eles.

A hora de usar `make-variable-buffer-local`é quando é crucial que dois buffers nunca compartilhem a mesma ligação. Por exemplo, quando uma variável é usada para propósitos internos em um programa Lisp que depende de ter valores separados em buffers separados, então usar `make-variable-buffer-local`pode ser a melhor solução.

Macro: valor da variável local defvar *& docstring opcional*

Esta macro define a *variável* como uma variável com valor inicial *e* docstring , e a marca automaticamente como buffer-local. É equivalente a chamar `defvar`seguido por `make-variable-buffer-local`. *variável* deve ser um símbolo sem aspas.

Função: variável local-variável-p *& buffer opcional*

Isso retorna `t`se *a variável* for buffer-local no buffer *buffer* (o padrão é o buffer atual); caso contrário, `nil`.

Função: variável local-if-set-p *variável & buffer opcional*

Isso retorna `t`se a *variável* tiver um valor buffer-local em buffer *buffer* ou for automaticamente buffer-local. Caso contrário, ele retorna `nil`. Se omitido ou `nil`, o *buffer* assume como padrão o buffer atual.

Função: buffer *variável* de valor local de buffer

Esta função retorna a ligação buffer-local da *variável* (um símbolo) no buffer *buffer* . Se a *variável* não tiver uma ligação buffer-local em buffer *buffer* , ela retornará o valor padrão (consulte Valor padrão) da *variável* .

Função: variáveis locais de *buffer e buffer opcional*

Esta função retorna uma lista que descreve as variáveis locais de buffer em buffer *buffer* . (Se *buffer* for omitido, o buffer atual será usado.) Normalmente, cada elemento da lista tem a forma , onde *sym* é uma variável local de buffer (um símbolo) e *val* é seu valor local de buffer. Mas quando a ligação buffer-local de uma variável no *buffer* é void, seu elemento de lista é apenas *sym* . `(sym . val)`

```
(criar variável local 'foobar)
(makunbound 'foobar)
(fazer-variável local 'bind-me)
(setq me ligar 69)
(setq lcl (variáveis locais de buffer))
    ;; Primeiro, variáveis internas locais em todos os buffers:
⇒ ((marca ativa . nil)
    (buffer-undo-list . nil)
    (modo-nome . "Fundamental")
    …
    ;; Em seguida, variáveis locais de buffer não incorporadas.
    ;; Este é buffer-local e void:
    foobar
    ;; Este é buffer-local e nonvoid:
    (ligar-me. 69))
```

Observe que armazenar novos valores nos CDRs das células cons nesta lista *não* altera os valores locais de buffer das variáveis.

Comando: variável kill-local- *variable*

Esta função exclui a ligação local do buffer (se houver) para a *variável* (um símbolo) no buffer atual. Como resultado, a associação padrão da *variável* se torna visível nesse buffer. Isso normalmente resulta em uma mudança no valor da *variável* , já que o valor padrão geralmente é diferente do valor local do buffer que acabou de ser eliminado.

Se você eliminar a ligação buffer-local de uma variável que automaticamente se torna buffer-local quando definida, isso torna o valor padrão visível no buffer atual. No entanto, se você definir a variável novamente, isso criará novamente uma ligação local de buffer para ela.

`kill-local-variable`retorna *variável* .

Essa função é um comando porque às vezes é útil matar uma variável local de buffer interativamente, assim como é útil criar variáveis locais de buffer interativamente.

Função: kill-all-local-variables

Esta função elimina todas as ligações de variáveis locais de buffer do buffer atual, exceto para variáveis marcadas como permanentes e funções de gancho locais que não possuem uma `nil` `permanent-local-hook` propriedade (consulte Configurando Ganchos). Como resultado, o buffer verá os valores padrão da maioria das variáveis.

Essa função também redefine algumas outras informações relativas ao buffer: ela define o mapa de teclas local como `nil`, a tabela de sintaxe com o valor de `(standard-syntax-table)`, a tabela de casos com o valor de `(standard-case-table)`, e a tabela abreviada com o valor de `fundamental-mode-abbrev-table`.

A primeira coisa que esta função faz é executar o gancho normal `change-major-mode-hook`(veja abaixo).

Cada comando de modo principal começa chamando esta função, que tem o efeito de alternar para o modo fundamental e apagar a maioria dos efeitos do modo principal anterior. Para garantir que isso funcione, as variáveis que os modos principais definem não devem ser marcadas como permanentes.

`kill-all-local-variables`retorna `nil`.

Variável: change-major-mode-hook

A função `kill-all-local-variables`executa esse gancho normal antes de fazer qualquer outra coisa. Isso dá aos modos principais uma maneira de organizar algo especial a ser feito se o usuário alternar para um modo principal diferente. Também é útil para modos secundários específicos de buffer que devem ser esquecidos se o usuário alterar o modo principal.

Para obter melhores resultados, torne essa variável buffer-local, para que ela desapareça depois de fazer seu trabalho e não interfira no modo principal subsequente. Veja Ganchos .

Uma variável local de buffer é *permanente* se o nome da variável (um símbolo) tiver uma `permanent-local`propriedade que não seja `nil`. Tais variáveis não são afetadas por `kill-all-local-variables`, e suas ligações locais, portanto, não são apagadas alterando os modos principais. Locais permanentes são apropriados para dados relativos à origem do arquivo ou como salvá-lo, em vez de como editar o conteúdo.

Próximo:Valor padrão, Anterior:Introdução ao Buffer-Local, Acima:Variáveis locais de buffer [Conteúdo][Índice]