Próximo:Funções obsoletas, Anterior:Fechamentos, Acima:Funções [Conteúdo][Índice]

13.11 Aconselhando as funções do Lisp do Emacs

Quando você precisa modificar uma função definida em outra biblioteca, ou quando você precisa modificar um gancho como *foo*`-function`, um filtro de processo ou basicamente qualquer variável ou campo de objeto que contém um valor de função, você pode usar a função setter apropriada, como `fset`ou `defun`para funções nomeadas, `setq`para variáveis de gancho ou `set-process-filter`para filtros de processo, mas esses geralmente são muito grosseiros, descartando completamente o valor anterior.

O recurso de *aconselhamento* permite adicionar à definição existente de uma função, *aconselhando a função* . Este é um método mais limpo do que redefinir toda a função.

O sistema de conselhos do Emacs fornece dois conjuntos de primitivas para isso: o conjunto principal, para valores de função mantidos em variáveis e campos de objeto (com as primitivas correspondentes sendo `add-function`e `remove-function`) e outro conjunto em camadas sobre ele para funções nomeadas (com as primitivas principais sendo `advice-add`e `advice-remove`).

Como um exemplo trivial, veja como adicionar um conselho que modificará o valor de retorno de uma função toda vez que for chamada:

```
(defun my-double (x)
  (*x2))
(defun meu-aumento (x)
  (+ x 1))
(advice-add 'my-double :filter-return #'my-increase)
```

Depois de adicionar este aviso, se você chamar `my-double`com '3', o valor de retorno será '7'. Para remover este conselho, diga

```
(advice-remove 'my-double #'my-increase)
```

Um exemplo mais avançado seria rastrear as chamadas para o filtro de processo de um processo *proc* :

```
(função defun my-tracing-string (proc string)
  (mensagem "Proc %S recebeu %S" proc string))

(add-function:before (process-filter proc ) #'my-tracing-function)
```

Isso fará com que a saída do processo seja passada `my-tracing-function` antes de ser passada para o filtro do processo original. `my-tracing-function` recebe os mesmos argumentos que a função original. Quando terminar, você pode reverter para o comportamento não rastreado com:

```
(remove-function (process-filter proc ) #'my-tracing-function)
```

Da mesma forma, se você quiser rastrear a execução da função chamada `display-buffer`, você pode usar:

```
(defun his-tracing-function (orig-fun & rest args)
  (mensagem "display-buffer chamado com argumentos %S" argumentos)
  (let ((res (aplicar argumentos orig-fun)))
    (mensagem "display-buffer retornou %S" res)
    res))

(advice-add 'display-buffer :around #'sua-função de rastreamento)
```

Aqui, `his-tracing-function`é chamado em vez da função original e recebe a função original (adicionalmente aos argumentos dessa função) como argumento, para que possa chamá-la se e quando precisar. Quando você estiver cansado de ver essa saída, poderá reverter para o comportamento não rastreado com:

```
(advice-remove 'display-buffer #'sua-função de rastreamento)
```

Os argumentos `:before`e `:around`usados nos exemplos acima especificam como as duas funções são compostas, pois existem muitas maneiras diferentes de fazê-lo. A função adicionada também é chamada de *conselho* .

Próximo:Funções obsoletas, Anterior:Fechamentos, Acima:Funções [Conteúdo][Índice]