Próximo:Formulário de declaração, Anterior:Funções obsoletas, Acima:Funções [Conteúdo][Índice]

13.13 Funções Inline

Uma *função inline* é uma função que funciona como uma função comum, exceto por uma coisa: quando você compila uma chamada para a função (consulte Compilação de Byte), a definição da função é expandida para o chamador.

A maneira simples de definir uma função inline é escrever `defsubst`em vez de `defun`. O resto da definição parece a mesma coisa, mas usando `defsubst`diz para torná-la inline para compilação de bytes.

Macro: defsubst ***name args [doc] [declare] [interactive] body…***

> Esta macro define uma função embutida. Sua sintaxe é exatamente a mesma de `defun`(veja Definindo Funções).

Tornar uma função embutida geralmente faz com que suas chamadas de função sejam executadas mais rapidamente. Mas também tem desvantagens. Por um lado, reduz a flexibilidade; se você alterar a definição da função, as chamadas já embutidas ainda usarão a definição antiga até que você as recompile.

Outra desvantagem é que fazer uma função grande embutida pode aumentar o tamanho do código compilado tanto nos arquivos quanto na memória. Uma vez que a vantagem de velocidade das funções inline é maior para funções pequenas, você geralmente não deve fazer grandes funções inline.

Além disso, as funções inline não se comportam bem em relação à depuração, rastreamento e aconselhamento (consulte Funções de aconselhamento). Como a facilidade de depuração e a flexibilidade de redefinir funções são recursos importantes do Emacs, você não deve fazer uma função inline, mesmo que seja pequena, a menos que sua velocidade seja realmente crucial e você tenha cronometrado o código para verificar se o uso `defun`realmente tem desempenho problemas.

Depois que uma função inline é definida, sua expansão inline pode ser executada posteriormente no mesmo arquivo, assim como as macros.

É possível usar `defmacro`para definir uma macro para expandir no mesmo código que uma função inline executaria (consulte Macros). Mas a macro seria limitada ao uso direto em expressões - uma macro não pode ser chamada com `apply`, `mapcar`e assim por diante. Além disso, é preciso algum trabalho para converter uma função comum em uma macro. Para convertê-lo em uma função inline é fácil; basta substituir `defun`por `defsubst`. Como cada argumento de uma função embutida é avaliado exatamente uma vez, você não precisa se preocupar com quantas vezes o corpo usa os argumentos, como acontece com as macros.

Alternativamente, você pode definir uma função fornecendo o código que irá inline-la como uma macro do compilador. As macros a seguir tornam isso possível.

Macro: define-inline ***name args [doc] [declare] body…***

> Defina um *nome* de função fornecendo o código que faz sua inserção, como uma macro do compilador. A função aceitará os argumentos da lista de argumentos *e* terá o *corpo* especificado .
>
> Se presente, *doc* deve ser a string de documentação da função (veja Documentação da Função); *declare* , se presente, deve ser um `declare`formulário (consulte Declare Form) especificando os metadados da função.

As funções definidas via `define-inline`têm várias vantagens em relação às macros definidas por `defsubst`ou `defmacro`:

- Eles podem ser passados para `mapcar`(veja Funções de Mapeamento).

- Eles são mais eficientes.

- Podem ser usados como *formulários de lugar* para armazenar valores (ver Variáveis Generalizadas).

- Eles se comportam de uma maneira mais previsível do que `cl-defsubst` (consulte Listas de Argumentos em Extensões de Lisp Comuns para GNU Emacs Lisp).

Como `defmacro`, uma função embutida com `define-inline` herda as regras de escopo, dinâmicas ou léxicas, do site de chamada. Consulte Escopo variável .

As macros a seguir devem ser usadas no corpo de uma função definida por `define-inline`.

Macro: *expressão* de aspas inline

Expressão de cotação para `define-inline`. Isso é semelhante ao backquote (consulte Backquote), mas cita código e aceita apenas `,`, não `,@`.

Macro: inline-levals *(ligações…) corpo…*

Isso é semelhante a `let`(consulte Variáveis locais): ele configura variáveis locais conforme especificado por *bindings* e, em seguida, avalia *body* com essas associações em vigor. Cada elemento de *ligações* deve ser um símbolo ou uma lista da forma ; o resultado é avaliar *expr* e vincular *var* ao resultado. A cauda das *ligações* pode ser ou um símbolo que deve conter uma lista de argumentos, caso em que cada argumento é avaliado e o símbolo é vinculado à lista resultante. `(`*`var expr`*`)nil`

Macro: *expressão* inline-const-p

Retorna non- `nil`se o valor da *expressão* já for conhecido.

Macro: *expressão* inline-const-val

Retorna o valor da *expressão* .

Macro: formato de erro embutido *e argumentos restantes*

Sinalize um erro, formatando *os argumentos* de acordo com o *formato* .

Aqui está um exemplo de uso `define-inline`:

```
(definir myaccessor embutido (obj)
  (inline-levals (obj)
    (aspas em linha (if (foo-p ,obj) (aref (cdr ,obj) 3) (aref ,obj 2)))))
```

Isso é equivalente a

```
(defsubst myaccessor (obj)
  (se (foo-p obj) (aref (cdr obj) 3) (aref obj 2)))
```

Próximo:Formulário de declaração, Anterior:Funções obsoletas, Acima:Funções [Conteúdo][Índice]