Próximo:Expressões lambda, Acima:Funções [Conteúdo][Índice]

13.1 O que é uma função?

Em um sentido geral, uma função é uma regra para realizar uma computação dados valores de entrada chamados *argumentos* . O resultado da computação é chamado de *valor* ou *valor de retorno* da função. A computação também pode ter efeitos colaterais, como mudanças duradouras nos valores das variáveis ou no conteúdo das estruturas de dados (consulte Definição de efeito colateral). Uma *função pura* é uma função que, além de não ter efeitos colaterais, sempre retorna o mesmo valor para a mesma combinação de argumentos, independentemente de fatores externos, como tipo de máquina ou estado do sistema.

Na maioria das linguagens de computador, cada função tem um nome. Mas em Lisp, uma função no sentido estrito não tem nome: é um objeto que pode *opcionalmente* ser associado a um símbolo (por exemplo, `car`) que serve como nome da função. Consulte Nomes de funções . Quando uma função recebe um nome, geralmente também nos referimos a esse símbolo como uma "função" (por exemplo, nos referimos a "a função `car`"). Neste manual, a distinção entre um nome de função e o próprio objeto de função geralmente não é importante, mas tomaremos nota onde for relevante.

Certos objetos do tipo função, chamados *de formulários especiais* e *macros* , também aceitam argumentos para realizar cálculos. No entanto, conforme explicado abaixo, essas não são consideradas funções no Emacs Lisp.

Aqui estão termos importantes para funções e objetos semelhantes a funções:

expressão lambda

Uma função (no sentido estrito, ou seja, um objeto de função) que é escrita em Lisp. Estes são descritos na seção a seguir. Consulte Expressões lambda .

primitivo

Uma função que pode ser chamada de Lisp, mas na verdade é escrita em C. Primitivas também são chamadas *de funções* internas ou *subrs* . Exemplos incluem funções como `care` `append`. Além disso, todas as formas especiais (veja abaixo) também são consideradas primitivas.

Normalmente, uma função é implementada como primitiva porque é uma parte fundamental do Lisp (por exemplo, `car`), ou porque fornece uma interface de baixo nível para serviços do sistema operacional, ou porque precisa ser executada rapidamente. Ao contrário das funções definidas em Lisp, as primitivas podem ser modificadas ou adicionadas apenas alterando as fontes C e recompilando o Emacs. Veja Escrevendo primitivas do Emacs .

forma especial

Uma primitiva que é como uma função, mas não avalia todos os seus argumentos da maneira usual. Ele pode avaliar apenas alguns dos argumentos, ou pode avaliá-los em uma ordem incomum, ou várias vezes. Exemplos incluem `if`, `and`, e `while`. Consulte Formulários Especiais .

macro

Uma construção definida em Lisp, que difere de uma função na medida em que traduz uma expressão Lisp em outra expressão que deve ser avaliada em vez da expressão original. As macros permitem que os programadores de Lisp façam o tipo de coisa que os formulários especiais podem fazer. Consulte Macros .

comando

Um objeto que pode ser invocado por meio da `command-execute` primitiva, geralmente devido ao usuário digitar uma sequência de teclas *vinculada* a esse comando. Consulte Chamada interativa . Um comando geralmente é uma função; se a função é escrita em Lisp, ela é transformada em comando por um `interactive`formulário na definição da função (veja Definindo Comandos). Comandos que são funções também podem ser chamados de expressões Lisp, assim como outras funções.

Macros de teclado (strings e vetores) também são comandos, embora não sejam funções. Consulte Macros de teclado . Dizemos que um símbolo é um comando se sua célula de função contém um comando (veja Componentes de Símbolo); tal *comando nomeado* pode ser invocado com `M-x`.

fecho

Um objeto de função que é muito parecido com uma expressão lambda, exceto que também inclui um ambiente de associações de variáveis léxicas. Consulte Fechamentos .

função de código de byte

Uma função que foi compilada pelo compilador de bytes. Consulte Tipo de código de byte .

objeto de carregamento automático

Um espaço reservado para uma função real. Se o objeto autoload for chamado, o Emacs carrega o arquivo contendo a definição da função real e então chama a função real. Consulte Carregamento automático .

Você pode usar a função `functionp`para testar se um objeto é uma função:

Função: *objeto* functionp

Esta função retorna `t`se o *objeto* for qualquer tipo de função, ou seja, pode ser passado para `funcall`. Observe que `functionp`retorna `t`para símbolos que são nomes de função e retorna `nil`para formulários especiais.

Também é possível descobrir quantos argumentos uma função arbitrária espera:

Função: função *func* -arity

Esta função fornece informações sobre a lista de argumentos da *função* especificada . O valor retornado é uma célula cons da forma , onde *min* é o número mínimo de argumentos e *max* é o número máximo de argumentos ou o símbolo para funções com argumentos ou o símbolo se a *função* for uma forma especial. `(`*`min`*` . `*`max`*`)`many`&`restunevalled

Observe que essa função pode retornar resultados imprecisos em algumas situações, como as seguintes:

- Funções definidas usando `apply-partially`(ver apply-partially).

- Funções que são aconselhadas a usar `advice-add`(consulte Aconselhando Funções Nomeadas).

- Funções que determinam a lista de argumentos dinamicamente, como parte de seu código.

Ao contrário `functionp`de , as próximas três funções *não* tratam um símbolo como sua definição de função.

Função: *objeto* subrp

Esta função retorna `t` se o *objeto* for uma função interna (ou seja, uma primitiva Lisp).

```
(subrp 'mensagem); messageé um símbolo,
     ⇒ nil ;   não um objeto subr.
(subrp (símbolo-função 'mensagem)))
     ⇒ t
```

Função: *objeto* byte-code-function-p

Esta função retorna `t` se o *objeto* for uma função de código de byte. Por exemplo:

```
(função de código de byte-p (função de símbolo 'próxima linha))
     ⇒ t
```

Função: subr- *arity subr*

Isso funciona como `func-arity`, mas apenas para funções internas e sem indireção de símbolo. Ele sinaliza um erro para funções não integradas. Recomendamos usar `func-arity`em vez disso.

Próximo:Expressões lambda, Acima:Funções [Conteúdo][Índice]