Anterior:Palavras-chave de tipo, Acima:Tipos de personalização [Conteúdo][Índice]

15.4.5 Definindo Novos Tipos

Nas seções anteriores, descrevemos como construir especificações de tipo elaboradas para `defcustom`. Em alguns casos, você pode querer dar um nome a tal especificação de tipo. O caso óbvio é quando você está usando o mesmo tipo para muitas opções de usuário: em vez de repetir a especificação para cada opção, você pode dar um nome à especificação de tipo e usar esse nome cada `defcustom`. O outro caso é quando o valor de uma opção do usuário é uma estrutura de dados recursiva. Para possibilitar que um tipo de dados se refira a si mesmo, ele precisa ter um nome.

Como os tipos personalizados são implementados como widgets, a maneira de definir um novo tipo de personalização é definir um novo widget. Não vamos descrever a interface do widget aqui em detalhes, veja Introdução na Biblioteca de Widgets do Emacs , para isso. Em vez disso, vamos demonstrar a funcionalidade mínima necessária para definir novos tipos de personalização por meio de um exemplo simples.

```
(defina-widget 'binary-tree-of-string' preguiçoso
  "Uma árvore binária feita de cons-cells e strings."
  :deslocamento 4
  :tag "Nó"
  :type '(escolha (string :tag "Folha" :valor "")
                  (contras :tag "Interior"
                        :valor ("" . "")
                        árvore-de-string binária
                        árvore-de-string binária)))

(defcustom foo-bar ""
  "Variável de amostra contendo uma árvore binária de strings."
  :type 'binary-tree-of-string)
```

A função para definir um novo widget é chamada `define-widget`. O primeiro argumento é o símbolo que queremos criar um novo tipo de widget. O segundo argumento é um símbolo que representa um widget existente, o novo widget será definido em termos de diferença do widget existente. Para fins de definição de novos tipos de customização, o `lazy`widget é perfeito, pois aceita um `:type`argumento de palavra-chave com a mesma sintaxe do argumento de palavra-chave `defcustom`com o mesmo nome. O terceiro argumento é uma string de documentação para o novo widget. Você poderá ver essa string com o comando. `M-x widget-browse RET binary-tree-of-string RET`

Após esses argumentos obrigatórios, seguem os argumentos da palavra-chave. O mais importante é `:type`, que descreve o tipo de dados que queremos combinar com este widget. Aqui a `binary-tree-of-string`é descrito como sendo uma string ou uma cons-cell cujo car e cdr são ambos `binary-tree-of-string`. Observe a referência ao tipo de widget que estamos atualmente definindo. O `:tag` atributo é uma string para nomear o widget na interface do usuário, e o `:offset`argumento existe para garantir que os nós filho sejam recuados quatro espaços em relação ao nó pai, tornando a estrutura de árvore aparente no buffer de customização.

O `defcustom`mostra como o novo widget pode ser usado como um tipo de personalização comum.

A razão para o nome `lazy`é que os outros widgets compostos convertem seus widgets inferiores para o formato interno quando o widget é instanciado em um buffer. Essa conversão é recursiva, portanto, os

widgets inferiores converterão *seus* widgets inferiores. Se a própria estrutura de dados for recursiva, essa conversão será uma recursão infinita. O `lazy`widget evita a recursão: ele converte seu `:type`argumento apenas quando necessário.

Anterior:Palavras-chave de tipo, Acima:Tipos de personalização [Conteúdo][Índice]