Próximo:Tipos de personalização, Anterior:Definições de grupo, Acima:Costumização [Conteúdo][Índice]

15.3 Definindo Variáveis de Personalização

Variáveis personalizáveis , também chamadas *de opções do usuário* , são variáveis Lisp globais cujos valores podem ser definidos por meio da interface Personalizar. Ao contrário de outras variáveis globais, que são definidas com `defvar`(consulte Definindo variáveis), as variáveis personalizáveis são definidas usando a `defcustom`macro. Além de chamar `defvar`como uma sub-rotina, `defcustom`informa como a variável deve ser exibida na interface Personalizar, os valores que ela pode assumir, etc.

Macro: doc padrão da opção defcustom *[valor da palavra-chave]…*

Essa macro declara a *opção* como uma opção do usuário (ou seja, uma variável personalizável). Você não deve citar a *opção* .

O argumento *padrão* é uma expressão que especifica o valor padrão para *opção* . Avaliar o `defcustom`formulário avalia *standard* , mas não necessariamente vincula a opção a esse valor. Se a *opção* já tiver um valor padrão, ela será deixada inalterada. Se o usuário já salvou uma personalização para *option* , o valor personalizado do usuário é instalado como o valor padrão. Caso contrário, o resultado da avaliação do *padrão* é instalado como valor padrão.

Como `defvar`, essa macro marca `option`como uma variável especial, o que significa que ela deve sempre ser vinculada dinamicamente. Se a *opção* já estiver vinculada lexicalmente, essa vinculação léxica permanecerá em vigor até que a construção de vinculação seja encerrada. Consulte Escopo variável .

O padrão de expressão também pode ser avaliado em vários outros momentos - sempre que o recurso de personalização precisar saber o valor padrão da *opção* . Portanto, certifique-se de usar uma expressão que seja inofensiva para avaliar a qualquer momento.

O argumento *doc* especifica a string de documentação para a variável.

Se a `defcustom`não especificar qualquer , será usado `:group`o último grupo definido com no mesmo arquivo. `defgroup`Dessa forma, a maioria `defcustom`não precisa de um `:group`.

Quando você avalia um `defcustom`formulário com o `C-M-x`modo Emacs Lisp (`eval-defun`), um recurso especial do `eval-defun` organiza para definir a variável incondicionalmente, sem testar se seu valor é nulo. (O mesmo recurso se aplica a `defvar`, veja Definindo Variáveis .) Usar `eval-defun`em um defcustom que já está definido chama a `:set`função (veja abaixo), se houver uma.

Se você colocar a `defcustom`em um arquivo Emacs Lisp pré-carregado (veja Construindo o Emacs), o valor padrão instalado na hora do dump pode estar incorreto, por exemplo, porque outra variável da qual ela depende ainda não recebeu o valor correto. Nesse caso, use `custom-reevaluate-setting`, descrito abaixo, para reavaliar o valor padrão após a inicialização do Emacs.

Além das palavras-chave listadas em Palavras- chave comuns , essa macro aceita as seguintes palavras-chave:

`:type` *`type`*

Use *type* como o tipo de dados para esta opção. Ele especifica quais valores são legítimos e como exibir o valor (consulte Tipos de personalização). Cada `defcustom`deve especificar um valor para esta palavra-chave.

`:options` *`value-list`*

Especifique a lista de valores razoáveis para uso nesta opção. O usuário não está restrito a usar apenas esses valores, mas eles são oferecidos como alternativas convenientes.

Isso é significativo apenas para determinados tipos, atualmente incluindo `hook`, `pliste` `alist`. Consulte a definição dos tipos individuais para obter uma descrição de como usar o `:options`.

`:set` *`setfunction`*

Especifique *setfunction* como a forma de alterar o valor desta opção ao usar a interface Personalizar. A função *setfunction* deve receber dois argumentos, um símbolo (o nome da opção) e o novo valor, e deve fazer o que for necessário para atualizar o valor adequadamente para esta opção (o que pode não significar simplesmente definir a opção como uma variável Lisp); de preferência, porém, não deve modificar seu argumento de valor de forma destrutiva. O padrão para *setfunction* é `set-default`.

Se você especificar essa palavra-chave, a string de documentação da variável deve descrever como fazer o mesmo trabalho em código Lisp escrito à mão.

`:get` *`getfunction`*

Especifique *getfunction* como a forma de extrair o valor desta opção. A função *getfunction* deve receber um argumento, um símbolo, e deve retornar qualquer valor que customize deve ser usado como o valor atual para esse símbolo (que não precisa ser o valor Lisp do símbolo). O padrão é `default-value`.

Você precisa realmente entender o funcionamento do Custom para usar `:get`corretamente. Destina-se a valores que são tratados em Custom como variáveis, mas não são realmente armazenados em variáveis Lisp. É quase certo que é um erro especificar *getfunction* para um valor que realmente está armazenado em uma variável Lisp.

`:initialize` *`function`*

function deve ser uma função usada para inicializar a variável quando o `defcustom`é avaliado. Deve receber dois argumentos, o nome da opção (um símbolo) e o valor. Aqui estão algumas funções predefinidas destinadas a serem usadas dessa maneira:

`custom-initialize-set`

Use a `:set`função da variável para inicializar a variável, mas não a reinicialize se já não for nula.

`custom-initialize-default`

Como `custom-initialize-set`, mas use a função `set-default`para definir a variável, em vez da `:set`função da variável. Esta é a escolha usual para uma variável cuja `:set`função habilita ou desabilita um modo menor; com essa escolha, definir a variável não chamará a função de modo secundário, mas personalizar a variável fará isso.

`custom-initialize-reset`

Sempre use a `:set`função para inicializar a variável. Se a variável já for não nula, redefina-a chamando a `:set` função usando o valor atual (retornado pelo `:get`método). Esta é a função padrão `:initialize`.

`custom-initialize-changed`

Use a `:set`função para inicializar a variável, caso já esteja configurada ou tenha sido customizada; caso contrário, basta usar `set-default`.

`custom-initialize-delay`

Esta função se comporta como `custom-initialize-set`, mas atrasa a inicialização real para o próximo início do Emacs. Isso deve ser usado em arquivos que são pré-carregados (ou para variáveis carregadas automaticamente), para que a inicialização seja feita no contexto de tempo de execução e não no contexto de tempo de compilação. Isso também tem o efeito colateral de que a inicialização (atrasada) é executada com a `:set`função. Consulte Construindo o Emacs .

`:local` *`value`*

Se o *valor* for `t`, marque a *opção* como automaticamente buffer-local; se o valor for `permanent`, também defina a propriedade da *opção* como . Consulte Criando Buffer Local . `permanent-localt`

`:risky` *`value`*

Defina a `risky-local-variable`propriedade da variável como *valor* (consulte Variáveis locais de arquivo).

`:safe` *`function`*

Defina a `safe-local-variable`propriedade da variável para *funcionar* (consulte File Local Variables).

`:set-after` *`variables`*

Ao definir as variáveis de acordo com as personalizações salvas, certifique-se de definir as variáveis das *variáveis* antes desta; isto é, retarde a configuração desta variável até que as outras tenham sido tratadas. Use `:set-after`se definir esta variável não funcionará corretamente, a menos que essas outras variáveis já tenham seus valores pretendidos.

É útil especificar a palavra- `:require`chave para uma opção que ativa um determinado recurso. Isso faz com que o Emacs carregue o recurso, se ainda não estiver carregado, sempre que a opção for definida. Consulte Palavras-chave comuns . Aqui está um exemplo:

```
(defcustom frobnicate-automaticamente nil
  "Não-nil significa frobnicar automaticamente todos os buffers."
  :type 'boolean
  :require 'frobnicate-mode
  :grupo 'frobnicate)
```

Se um item de personalização tiver um tipo como `hook`ou `alist`, que suporte `:options`, você poderá adicionar valores adicionais à lista de fora da `defcustom`declaração chamando `custom-add-frequent-value`. Por exemplo, se você definir uma função `my-lisp-mode-initialization`destinada a ser chamada de `emacs-lisp-mode-hook`, talvez queira adicioná-la à lista de valores razoáveis para `emacs-lisp-mode-hook`, mas não editando sua definição. Você pode fazer assim:

```
(custom-add-frequent-value 'emacs-lisp-mode-hook
   'my-lisp-mode-initialization)
```

Função: valor de *símbolo* de valor de adição de costume

Para o símbolo de opção de personalização , adicione *valor* à lista de valores razoáveis.

O efeito preciso de adicionar um valor depende do tipo de personalização do *símbolo* .

Internamente, `defcustom`usa a propriedade symbol `standard-value`para registrar a expressão para o valor padrão, `saved-value`para registrar o valor salvo pelo usuário com o buffer de personalização e `customized-value`para registrar o valor definido pelo usuário com o buffer de personalização, mas não salvo. Consulte Propriedades do símbolo . Além disso, há `themed-value`, que é usado para registrar o valor definido por um tema (consulte Temas personalizados). Essas propriedades são listas, cujo carro é uma expressão que avalia o valor.

Função: *símbolo* de configuração de reavaliar personalizado

Esta função reavalia o valor padrão de *symbol* , que deve ser uma opção do usuário declarada via `defcustom`. Se a variável foi personalizada, esta função reavalia o valor salvo. Em seguida, ele define a opção do usuário para esse valor (usando a `:set`propriedade da opção, se estiver definida).

Isso é útil para opções personalizáveis que são definidas antes que seu valor possa ser calculado corretamente. Por exemplo, durante a inicialização, o Emacs chama essa função para algumas opções de usuário que foram definidas em arquivos Emacs Lisp pré-carregados, mas cujos valores iniciais dependem de informações disponíveis apenas em tempo de execução.

Função: variável personalizada-p *arg*

Esta função retorna non - `nil`se *arg* é uma variável personalizável. Uma variável personalizável é uma variável que possui a propriedade `standard-value`ou `custom-autoload`(geralmente significando que foi declarada com `defcustom`), ou um alias para outra variável personalizável.

Próximo:Tipos de personalização, Anterior:Definições de grupo, Acima:Costumização [Conteúdo][Índice]