Próximo:Onde definido, Anterior:Carregamento repetido, Acima:Carregando [Conteúdo][Índice]

16.7 Recursos

`provide`e `require`são uma alternativa `autoload`para carregar arquivos automaticamente. *Eles funcionam em termos de recursos* nomeados . O carregamento automático é acionado chamando uma função específica, mas um recurso é carregado na primeira vez que outro programa o solicita pelo nome.

Um nome de recurso é um símbolo que representa uma coleção de funções, variáveis, etc. O arquivo que os define deve *fornecer* o recurso. Outro programa que os usa pode garantir que eles sejam definidos *exigindo* o recurso. Isso carrega o arquivo de definições se ainda não tiver sido carregado.

Para exigir a presença de um recurso, chame `require`com o nome do recurso como argumento. `require`olha na variável global `features`para ver se o recurso desejado já foi fornecido. Caso contrário, ele carrega o recurso do arquivo apropriado. Este arquivo deve chamar `provide`no nível superior para adicionar o recurso `features`; se não o fizer, `require`sinaliza um erro.

Por exemplo, em`idlwave.el`, a definição de `idlwave-complete-filename`inclui o seguinte código:

```
(defun idlwave-nome-do-arquivo-completo ()
  "Use o material de comint para completar um nome de arquivo."
   (requer 'comint)
   (let* ((comint-file-name-chars "~/A-Za-z0-9+@:_.$#%={}\\-")
          (comint-completion-addsuffix nil)
          ...)
      (comint-dynamic-complete-filename)))
```

A expressão `(require 'comint)`carrega o arquivo`comint.el` se ainda não foi carregado, garantindo que `comint-dynamic-complete-filename`esteja definido. Os recursos normalmente são nomeados após os arquivos que os fornecem, de modo que `require`não precisam receber o nome do arquivo. (Observe que é importante que a `require`instrução esteja fora do corpo do `let`. Carregar uma biblioteca enquanto suas variáveis são vinculadas ao let pode ter consequências não intencionais, ou seja, as variáveis se tornam desvinculadas após a saída do let.)

O`comint.el`O arquivo contém a seguinte expressão de nível superior:

```
(fornecer 'comint)
```

Isso aumenta `comint`a lista global `features`, de modo que `(require 'comint)`a partir de agora saberá que nada precisa ser feito.

Quando `require`é usado no nível superior em um arquivo, ele entra em vigor quando você compila esse arquivo em bytes (consulte Compilação de bytes), bem como quando você o carrega. Isso ocorre caso o pacote necessário contenha macros que o compilador de bytes deve conhecer. Também evita avisos do compilador de bytes para funções e variáveis definidas no arquivo carregado com `require`.

Embora as chamadas de nível superior `require`sejam avaliadas durante a compilação de bytes, `provide`as chamadas não são. Portanto, você pode garantir que um arquivo de definições seja carregado antes de ser

compilado por byte incluindo um `provide`seguido de um `require`para o mesmo recurso, como no exemplo a seguir.

```
(fornecer 'meu-recurso); Ignorado pelo compilador de bytes,
                       ;   avaliado por load.
(requer 'meu-recurso); Avaliado pelo compilador de bytes.
```

O compilador ignora o `provide`, então processa o `require`carregando o arquivo em questão. O carregamento do arquivo executa a `provide`chamada, portanto, a `require`chamada subsequente não faz nada quando o arquivo é carregado.

Função: fornecer *recursos e sub-recursos opcionais*

Esta função anuncia que o *recurso* agora está carregado, ou sendo carregado, na sessão atual do Emacs. Isso significa que as facilidades associadas ao *recurso* estão ou estarão disponíveis para outros programas Lisp.

O efeito direto de chamar `provide`é adicionar *um recurso* à frente de `features`se ele ainda não estiver nessa lista e chamar qualquer `eval-after-load`código que esteja esperando por ele (consulte Ganchos para carregamento). *O recurso* de argumento deve ser um símbolo. *recurso*`provide` de retorno .

Se fornecidos, *os sub* -recursos devem ser uma lista de símbolos indicando um conjunto de sub-recursos específicos fornecidos por esta versão do *recurso* . Você pode testar a presença de um sub-recurso usando `featurep`. A ideia de sub-recursos é que você os usa quando um pacote (que é um *recurso*) é complexo o suficiente para torná-lo útil para dar nomes a várias partes ou funcionalidades do pacote, que podem ou não ser carregadas, ou podem ou não estar presente em uma determinada versão. Consulte Teste de recursos de rede , para obter um exemplo.

```
recursos
     ⇒ (bar bish)

(fornecer 'foo)
     ⇒ fo
recursos
     ⇒ (foo bar bish)
```

Quando um arquivo é carregado para atender a um carregamento automático e é interrompido devido a um erro na avaliação de seu conteúdo, quaisquer definições de função ou `provide`chamadas que ocorreram durante o carregamento são desfeitas. Consulte Carregamento automático .

Função: requer *recurso e nome de arquivo opcional noerror*

Esta função verifica se *o recurso* está presente na sessão atual do Emacs (usando ; veja abaixo). *O recurso* de argumento deve ser um símbolo. `(featurep feature)`

Se o recurso não estiver presente, `require`carrega o nome do *arquivo* com `load`. Se o nome do *arquivo não for fornecido, o nome do recurso* de símbolo será usado como o nome do arquivo base a ser carregado. No entanto, neste caso, `require`insiste em encontrar *o recurso* com um ' adicionado`.el`' ou '`.elc`' sufixo (possivelmente estendido com um sufixo de compressão); um arquivo cujo nome seja apenas *feature* não será usado. (A variável `load-suffixes`especifica os sufixos Lisp exatos necessários.)

Se *noerror* for non- `nil`, isso suprime erros do carregamento real do arquivo. Nesse caso, `require`retorna `nil` se o carregamento do arquivo falhar. Normalmente, `require`retorna *o recurso* .

Se o carregamento do arquivo for bem-sucedido, mas não fornecer *recurso* , `require`sinalizará um erro sobre o recurso ausente.

Função: recurso featurep *e sub-recurso opcional*

Esta função retorna `t`se *o recurso* foi fornecido na sessão atual do Emacs (ou seja, se *o* recurso é *um membro* de . `features`) da propriedade do símbolo de *recurso .)*`nil`ou`subfeature`

Variável: características

O valor desta variável é uma lista de símbolos que são os recursos carregados na sessão atual do Emacs. Cada símbolo foi colocado nesta lista com uma chamada para `provide`. A ordem dos elementos na `features`lista não é significativa.

Próximo:Onde definido, Anterior:Carregamento repetido, Acima:Carregando [Conteúdo][Índice]