Anterior:Erros, Acima:Saídas não locais [Conteúdo][Índice]

11.7.4 Limpeza de Saídas Não Locais

A `unwind-protect`construção é essencial sempre que você coloca temporariamente uma estrutura de dados em um estado inconsistente; ele permite que você torne os dados consistentes novamente no caso de um erro ou lançamento. (Outra construção de limpeza mais específica que é usada apenas para alterações no conteúdo do buffer é o grupo de alterações atômicas; Alterações Atômicas .)

Forma especial: *formas de limpeza de forma corporal* descontrair-proteger …

`unwind-protect`executa *o body-form* com a garantia de que os *cleanup-forms* serão avaliados se o controle sair *do body-form* , não importa como isso aconteça. *body-form* pode completar normalmente, ou executar um `throw`out of the `unwind-protect`, ou causar um erro; em todos os casos, os *formulários de limpeza* serão avaliados.

Se *body-form* terminar normalmente, `unwind-protect`retorna o valor de *body-form* , após avaliar o *cleanup-forms* . Se *body-form* não terminar, `unwind-protect`não retorna nenhum valor no sentido normal.

Somente *a forma do corpo* é protegida pelo `unwind-protect`. Se algum dos próprios *formulários de limpeza* sair não localmente (através de um `throw`ou de um erro), *não*`unwind-protect` é garantido que o restante deles seja avaliado. Se a falha de um dos *formulários de limpeza* tiver o potencial de causar problemas, proteja-o com outro em torno desse formulário. `unwind-protect`

O número de formulários atualmente ativos `unwind-protect`conta, junto com o número de associações de variáveis locais, em relação ao limite `max-specpdl-size`(consulte Variáveis locais).

Por exemplo, aqui fazemos um buffer invisível para uso temporário, e certifique-se de matá-lo antes de terminar:

```
(let ((buffer (get-buffer-create " *temp*")))
  (com buffer de buffer de corrente
    (descontrair-proteger
        forma corporal
      (buffer kill-buffer))))
```

Você pode pensar que poderíamos escrever `(kill-buffer (current-buffer))`e dispensar a variável `buffer`. No entanto, a maneira mostrada acima é mais segura, se *a forma do corpo* receber um erro após alternar para um buffer diferente! (Como alternativa, você pode escrever um *body-form*`save-current-buffer` , para garantir que o buffer temporário se torne atual novamente a tempo de matá-lo.)

O Emacs inclui uma macro padrão chamada `with-temp-buffer`que se expande para mais ou menos o código mostrado acima (veja Current Buffer). Várias das macros definidas neste manual são utilizadas `unwind-protect`desta forma.

Aqui está um exemplo real derivado de um pacote FTP. Ele cria um processo (consulte Processos) para tentar estabelecer uma conexão com uma máquina remota. Como a função `ftp-login`é altamente suscetível a inúmeros problemas que o redator da função não pode prever, ela é protegida com um formulário que garante a exclusão do processo em caso de falha. Caso contrário, o Emacs pode ficar cheio de subprocessos inúteis.

```
(deixe ((ganhar nada))
  (descontrair-proteger
      (prog
        (processo setq (arquivo host ftp-setup-buffer))
        (if (setq win (senha do usuário do host do processo ftp-login))
            (mensagem "Logado")
          (erro "Falha no login FTP")))
    (ou ganhar (e processar (processo de exclusão do processo)))))
```

Este exemplo tem um pequeno bug: se o usuário digitar `C-g`para sair, e a saída acontecer imediatamente após a função `ftp-setup-buffer`retornar, mas antes da variável `process`ser definida, o processo não será encerrado. Não há uma maneira fácil de corrigir esse bug, mas pelo menos é muito improvável.

Anterior:Erros, Acima:Saídas não locais [Conteúdo][Índice]