第6章 变窄和增宽

“变窄”(narrowing)是Emacs的一个特性，这个特性允许你让 Emacs 关注于一个缓冲区的特定部分，而不会在无意中更改缓冲区的其他部分。变窄一般都是没有开启的，因为它会将新手弄得不知所措。

采用变窄技术之后，缓冲区的其余部分就变成不可见的了，就像它们并不存在一样。这样做有一个好处，例如，要在缓冲区的某个部分而不是在别的部分替换一个单词：首先将那个部分隔离出来，替换工作就在缓冲区的这个变窄的部分进行，而不在缓冲区的其余部分进行。查询也是在缓冲区的一个变窄的部分进行，而不是在缓冲区的其他部分进行。因此，如果你正在修改一个文档，你可以使自己严格限制在要修改的那个部分，而不要跑到其他部分去。只要用变窄技术就可以实现这一点。

然而，变窄确实使缓冲区的其余部分不可见，这会使那些无意中设置了变窄功能的人惊恐不安，并认为他们已经删除了文档的一部分。更有甚者，undo 命令（这个命令经常绑定到 C-x u）无法取消变窄开启（或者不应当），因此，如果人们不知道可以用 widen 命令使其余部分意新变成可见的，他们就会非常绝望。(在 Emacs 第18版中，widen 命令一般绑定到C-x w；而在第19版中，则绑定到 C-x n w。)

变窄技术不仅对人有用，而且对 Lisp 解释器也同样有用。通常，一个 Emacs Lisp 函数被设计成针对缓冲区的一个区域操作，或者反过来说，Emacs Lisp 函数需要在一个变窄开启的缓冲区中执行。例如，如果一个缓冲区设置了变窄开启，what-line 函数从缓冲区中取消变窄开启，然后完成它本身的工作，随后再恢复缓冲区中原来的变窄开启。另一方面，由 what-line 调用的 count-line 函数则用变窄技术来将这个函数的执行范围限制到缓冲区中你感兴趣的那个部分，并随后恢复原来的状态。

6.1 save-restriction特殊表

在 Emacs Lisp 中，能用save-restriction特殊表来跟踪变窄开启的部分。当 Lisp 解释器遇到 save-restriction 特殊表时，它执行这个表达式中的代码，并恢复这些代码导致的变窄开启的变更。例如，如果缓冲区本来是变窄开启的，而 save-restriction表达式中的代码取消了变窄开启，save-restriction 特殊表就返回变窄开启的缓冲区部分。在 what-line 命令中，缓冲区中可能包含的变窄开启都可以用紧跟在 save-restriction 特殊表后而的widen 命令取消。在这个函数执行完之前，所有的变窄开启都被恢复了。

save-restriction 表达式的模板很简单：

```
(save-restriction
  body... )
```

save-restriction 特殊表的主体是一个或多个表达式，这些表达式将被Lisp 解释器逐一求值。

最后，有一点值得提醒一下：当你同时使用 save-excursion和 save-restriction 时（并且是一个紧接着另一个使用时），应当在外层使用 save-excursion。如果采用了相反的次序，就会在调用 save-excursion 之后无法记录缓冲区中变窄开启的标记。因而，当同时使用这两个特殊表时，应当采用类似下面的顺序：

```
(save-excursion
  (save-restriction
    body...))
```

6.2 what-line函数

what-line 命令告诉你光标所在的行数。这个函数展示了如何使用save-excursion 和 save-restriction 命令。下面是这个函数的全部代码：

```
(defun what-line ()
  "Print the current line number (in the buffer) of point."
  (interactive)
  (save-restriction
    (widen)
    (save-excursion
      (beginning-of-line)
      (message "Line %d"
               (1+ (count-lines 1 (point)))))))
```

这个函数有一个文档说明行，并且该函数就像你希望的那样，也是一个交互函数。函数定义中接下来的两行使用了 save-restriction 和 widen 函数。

save-restriction 特殊表判断当前缓冲区是否设置了变窄开启，如果设置了，就在 save-restriction 特殊表中的所有代码执行完之后恢复变窄开启。

上面的代码中，save-restriction 特殊表之后紧跟了一个 widen 命令。当what-line 被调用时，这个函数取消当前缓冲区中可能有的所有的变窄开启标记。（其中的变窄开启标记由 save-restriction 特殊表记录下来。）这个增宽操作使对行的计数从缓冲区的开始处进行。否则，计数就被局限在缓冲区的可见部分。所有原有的变窄开启在 save-restriction 特殊表执行完时被恢复。

在 widen 命令之后紧跟着 save-excursion 特殊表，它保存光标的位置（即位点），并在此作一个标记，当 save-excursion 特殊表中的代码使用beginning-of-line 函数来移动位点之后再恢复它们。

（注意，(widen) 表达式夹在 save-restriction 和 save-excursion 之间。当你编写连续含有两个 save-…的表达式时，总是要将 save-excursion 写在最外层。）

what-line 函数的最后两行代码对缓冲区中行数进行计数，然后在回显区中打印这个数。

```
(message "Line %d"
         (1+ (count-lines 1 (point)))))))
```

这个 message 函数在 Emacs 屏幕底部输出一行消息。该函数的第一个参量是引号中的字

符串。然而，这个字符串可以包含如“%d”、“%s”或者“%c”这样的控制符，以打印跟在字符串后面的参量。“%d”将后续的参量作为十进制数输出。因此这个消息将输出如“Line 243”这类的消息。

代替“%d”而输出的数是由函数的最后一行计算得到的：

```
(1+ (count-lines 1 (point)))
```

这个表达式所做的工作，就是从缓冲区的开始位置(由1表示)计数，直到位点处（point），并对最后的计数加 1。（1+ 函数就是对其参量加 1。）我们之所以加 1，是因为第 2 行只是在第 1 行前面 1 行。而且 count-lines 只对当前行前面的行数计数。）

在 count-lines 求值完成时，消息输出在回显区，save-excursion 恢复位点和标记到它们原来的位置；而如果有变窄开启，save-restriction 则恢复变窄开启的原有标记。

6.3 练习：变窄

编写一个函数，这个函数在即使设置了变窄开启而使缓冲区的前一半不可见的情况下也能显示出当前缓冲区的头 60 个字符。要在显示完成之后恢复位点、标记和变窄开启等相关设置。对这个练习题，要使用 save-restriction、widen、goto-char、point-min、buffer-substring、message 和其他函数，真可以算得上是一个大杂烩！