第2章 求值实践

在学习用 Emacs Lisp 编写函数定义之前，花很少一点时间对已经学习过的各种表达式进行求值是很有帮助的。这些表达式是以函数作为其第一个（经常是仅有的）元素的列表。因为其中一些与缓冲区相联系的函数既简单又有趣，所以我们就从它们开始讲起。在这一章中，将对这样的一些表达式求值。在另一章中，我们将学习另外几段与缓冲区有关的函数的代码，看看那些函数是如何被编写的。

每当在 Emacs Lisp 中发出一个编辑命令时，比如一个移动光标或滚动屏幕的命令，就是在对一个表达式求值，这个表达式的第一个元素就是一个函数。这就是 Emacs 的工作方式。

当你击键时，你使 Lisp 解释器对一个表达式求值，于是你就得到了结果。即使是键入普通文本也是对 Emacs Lisp 的一个函数求值。在这种情况下，就是使用了 `self-insert-command` 函数，这个函数仅仅插入你输入的字符。通过键入键序列进行求值的函数被称为*交互函数*，或者是命令。如何使一个函数变成交互函数将在如何编写函数定义一章中讲解，参见3.3节，“使函数成为交互函数”。

除了键入键盘命令外，我们已经看到第二种对表达式求值的方法：将光标置于列表后，并键入 `C-x C-e`。在这一章的余下部分，我们就将这样做。对表达式求值还有其他方法，这些方法将在后续章节中描述。

除了用于求值实践外，在下面几节中演示的函数本身都是很重要的。学习这些函数可帮助你弄清：缓冲区和文件的区别、如何切换至一个缓冲区以及如何确定在其中的位置。

2.1 缓冲区名

`buffer-name` 和 `buffer-file-name`这两个函数显示文件和缓冲区之间的区别。当对表达式（`buffer-name`）求值时，缓冲区的名称将在回显区中出现。当对（`buffer-file-name`）表达式求值时，缓冲区所指的那个文件的名称将在回显区中出现。通常情况下，由（`buffer-name`）返回的名称与（`buffer-name`）所指的文件名称相同，由（`buffer-file-name`）返回的名称是文件完整的路径名。

文件和缓冲区是两个不同的实体。文件是永久记录在计算机中的信息（除非你删除了它）。而缓冲区则是 Emacs 内部的信息，它在 Emacs 编辑会话结束时(或当取消缓冲区时)就消失了。通常情况下，缓冲区包含了从文件中拷贝过来的信息，我们称这个缓冲区正在“访问”那个文件。这份拷贝正是你加工或修改的对象。对这个缓冲区的改动不会改变那个文件，除非你保存了这个缓冲区。当你保存这个缓冲区时，缓冲区中的内容被拷贝到文件中去，因此被永久地保存下来。

如果在 GNU Emacs 的 Info 中阅读本教程，可以通过将光标置于下面的表达式后并键入 `C-x C-e` 来对它们求值：

```
(buffer-name)

(buffer-file-name)
```

当我这样做时，“introduction.texinfo”是对（buffer-name）求值所返回的值，而“/gnu/work/intro/introduction.texinfo”是对（buffer-file-name）求值所返回的值。前者是缓冲区的名字，而后者是文件的名字。（在表达式中，括号告诉 Lisp 解释器将buffer-name 和buffer-file-name当做函数处理；如果没有括号，则解释器将它们当做变量来对这些符号求值。参见1.7节，“变量”。）

尽管缓冲区和文件有这些区别，但是你会经常发现人们在说一个缓冲区的时候是指文件，或者反过来。在实践中绝大多数人会说：“我正在编辑一个文件”，而不是说“我正在编辑很快就要存入文件的一个缓冲区”。从人们谈话的上下文中几乎总是可以知道人们真正所指的东西。然而，当处理计算机程序时，你在头脑中清楚地意识到这两者的区别是非常重要的，因为计算机可没有人那么聪明。

“缓冲区”一词来自于这个词可被用作“缓解碰撞力的软垫”之意。在早期的计算机里，缓冲区在文件和中央处理器（CPU）之间的相互作用中起缓和的作用。那时，磁鼓和磁带用于保存文件，它们和 CPU 是彼此很不相同的设备，各自以其固有的速度高速运行。缓冲区使它们能够共同高效地工作。最终，缓冲区演变成为一个中间部件，一个临时存放区，计算机的工作就是在这里进行的。这种变化就像一个小海港成长为一个大城市一样：原来它仅仅是尚未装上船的货物的临时仓库，后来以其自身条件发展成为一个商业和文化中心。

并不是所有的缓冲区都与文件联系在一起。例如，当键入 emacs 命令启动一个 Emacs 会话时，没有给出任何文件，Emacs 将在屏幕上启动一个“*scracth*”(草稿)缓冲区。这个缓冲区并没有访问任何文件。类似地，一个“*help*”(帮助)缓冲区也不与任何文件相关联。

如果切换到“*scratch*”缓冲区，键入（buffer-name），将光标置于列表之后，并键入 C-x C-e 对这个表达式求值，“*scratch*”这个名字将显示在回显区中。“*scratch*”就是这个缓冲区的名字。然而，如果输入（buffer-file-name）并对它求值，nil 将显示在回显区中。nil 一词来自于拉丁语，意指“什么都没有”（空）。在这种情况下，它是指“*scratch*”缓冲区没有与任何文件关联。（在Lisp 中，nil 也用于指“假”，或者用作空列表（）的同义语。）

顺便提一下，如果你在“*scratch*”缓冲区中并希望由一个表达式返回的值出现在缓冲区中而不是出现在回显区中，键入 C-u C-x C-e 而不是键入 C-x C-e。这将使返回值显示在表达式的后面。缓冲区看起来如下所示：

```
(buffer-name)"*scratch*"
```

你无法在 Info 中完成上面的工作，因为 Info 是只读的，它不允许你改变缓冲区中的内容。但是你可以在任意能够进行编辑的缓冲区中这样做；当编写代码或者文档时（例如我写作本书时），这个特性是非常有用的。

2.2 获得缓冲区

buffer-name函数返回缓冲区的名字。为了获得缓冲区本身，需要另外一个函数：

current-buffer。如果在代码中使用这个函数，得到的将是这个缓冲区本身。

一个名字与名字所指的对象或实体是互不相同的。你不是你的名字。你是一个用名字指向的人。如果你要求与 George 讲话，有人给你一张印有“G”“e”“o”“r”“g”“e”这几个字母的名片，你可能被搞糊涂，你不会满意的。你不是要对这个名字讲话，而是要与这个名字所指的那个人讲话。缓冲区就与此类似：草稿缓冲区的名字是“*scratch*”，但是这个名字本身不是缓冲区。为了得到缓冲区本身，你需要使用一个函数，如 current-buffer。

然而，这里带有一点复杂性：如果在缓冲区中的一个表达式内对 current-buffer 求值，就像我们将要做的那样，你所看到的是打印出来的这个缓冲区对应的名字，而没有缓冲区的内容。Emacs 这样做有两个理由：缓冲区可能有数千行长——显示起来太长了，不方便；而且，另外一个缓冲区可能有同样的内容，只是名字不一样而已，将它们之间区分开来是很重要的。

下面是包含这个current-buffer函数的一个表达式：

```
(current-buffer)
```

如果用常规的办法对这个表达式求值，“#<buffer *info*>”将显示在回显区中。这个特殊的格式表明这个缓冲区本身被返回了，而不仅仅是其名字。

顺便说一下，可以在一个程序中输入一个数或者符号，但却不能用这种方法得到缓冲区的打印表示：得到缓冲区本身的唯一方法是用一个函数，如 current-buffer函数。

一个相关的函数是 other-buffer。这个函数返回最近使用过的缓冲区，而不是当前使用的那个缓冲区。如果最近经常对“*scratch*”缓冲区不停地来回切换，那么 other-buffer 函数将返回那个缓冲区。

对下面的表达式求值就可以看到这一点：

```
(other-buffer)
```

应该看到“#<buffer *scratch*>”或者最近从其中切换回来的缓冲区的名字显示在回显区中。

2.3 切换缓冲区

当other-buffer函数被一个函数用作参量时，这个other-buffer 函数实际上提供了一个缓冲区。通过使用 other-buffer函数和 switcb-to-buffer 函数来切换到另外一个缓冲区，我们将看到这一点。

但是，先简单介绍一下 switch-to-buffer 函数。当在Info和草稿缓冲区“*scratch*”之间来回切换来对 (buffer-name) 表达式求值时，很可能要键入键序列C-x b，并当在小缓冲区中提示要求你输入希望切换到的缓冲区的名字时输入“*scratch*”。键序列 C-x b，使 Lisp 解释器对交互性的 Emacs Lisp 函数 switch-to-buffer 求值。正像我们在前面讲的，这就是 Emacs 的工作方式：不同的键序列调用和运行不同的函数。例如，C-f 调用forward-char 函数，M-e 调用 forward-sentence 函数，等等。

在一个表达式中写入 switch-to-buffer 函数，并给它一个要切换到的缓冲区，就可以像 C-x b 那样切换缓冲区了。

以下就是完成这个任务的 Lisp 表达式：

```
(switch-to-buffer (other-buffer))
```

符号 switch-to-buffer 是这个列表的第一个元素，因此 Lisp 解释器将它视作成一个函数，并执行这个函数的指令。但在这样做之前，解释器将注意到 other-buffer 在一个括号内，因此先处理这个列表。other-buffer 是这个列表的第一个元素（在这种情况下也是仅有的一个元素），因此 Lisp 解释器调用和运行这个函数。它返回另外一个缓冲区。下一步，解释器运行 switch-to-buffer 函数，将另外这个缓冲区作为一个参量传送给它，这后面一个缓冲区就是 Emacs 要切换到的缓冲区。如果你在 Info 中阅读这本教程，现在就可试一下，求这个表达式的值。（要返回的话，键入 C-x b RET。）

在这本教程的后续的编程例子中，将更多地看到 set-buffer 函数而不是switch-to-buffer 函数。这是因为人和计算机程序之间的一个差别：人有眼睛，并希望在他们工作的计算机终端上看到缓冲区。这是如此的直观，几乎不言自明。然而，计算机程序没有眼睛，当计算机程序工作在一个缓冲区时，缓冲区无需在屏幕上显示出来。

switch-to-buffer 函数是为人设计的，它完成两件不同的事情：一是切换到 Emacs 关注的缓冲区；一是从当前显示在窗口中的缓冲区切换到一个新的缓冲区。另一方面，set-buffer 函数只做一件事：它将计算机的注意力切换到另外一个不同的缓冲区。屏幕上显示的缓冲区并不改变（当然，直到命令运行完之前一般这不会发生任何事情）。

这里，我们已经接触了另外一个术语：调用（*call*）。当对第一个元素是一个函数的列表求值时，就是在调用那个函数。这个词的使用来自这样的概念，函数作为一个实体，如果“呼叫”它，它可以为你做某些事情——就像水管工人在你呼叫他时能帮你补漏一样。

2.4 缓冲区大小和位点的定位

最后，来看看几个相当简单的函数：buffer-size、point、point-min 和 point-max。这些函数给出缓冲区大小以及其中的位点的位置等信息。

buffer-size 函数给出当前缓冲区的大小，也就是，这个函数返回关于这个缓冲区中字符数的计数。

```
(buffer-size)
```

可以用通常的办法，即将光标置于这个表达式后面并键入 C-x C-e 来对这个表达式求值。

在 Emacs 中，光标所在的当前位置被称为“位点”（*point*）。表达式（point）返回一个数字，这个数字给出光标所处的位置，即从这个缓冲区首字符开始到光标所在位置之间的字符数。

用通常的办法对下面的表达式求值，你可以看看光标在这个缓冲区中当前位点的字符计数：

```
(point)
```

当我写到这里时，point 函数的返回值是 65724。point 函数将经常出现在本书后面的例子中。

当然，位点的值依赖于它在缓冲区中的位置。如果在这个点对 point 函数求值，返回数值会更大些：

```
(point)
```

对我而言，在这个位置中的位点的值是 66043，这意味着在这两个表达式之间有 319个字符（包括空格）。

point-min函数与 point 函数有点类似，但是它返回在当前缓冲区中位点的最小可能值。除非设置了变窄（*narrowing*），这个值一般就是 1。（变窄是一种自我限制的机制，限制用户或者一个程序只能对缓冲区的一部分进行操作。参见第 6 章，“变窄和增宽”。）与此类似，函数 point-max 返回在当前缓冲区中位点的最大可能值。

2.5 练习

找一个文件，对它进行操作，将光标移动到缓冲区的中间部分。找出它的缓冲区名、文件名、长度、和你在文件（其实是缓冲区）中的位置。