第16章 配置你的“.emacs”文件

“不要只为了喜欢 Emacs 而喜欢它”——这句看似悖论的话其实就是 GNU Emacs 的秘密所在。通常的 Emacs 是一个通用的工具。大多数使用它的人都根据自己的要求来定制适合自己的 Emacs。

GNU Emacs 的大部分是用 Emacs Lisp 编写的，这意味着通过在Emacs Lisp中编写表达式，你能够改变或者扩充 Emacs。

也有些人喜欢 Emacs 的默认配置。毕竟，当你编辑一个 C 语言文件时，Emacs就进入 C 语言模式；当你编辑一个 Fortran 语言文件时，Emacs 就进入 Fortran 模式；而当你编辑一个无格式的文件时，Emacs 就进入基本模式。如果你不知道谁将使用 Emacs，这种适应性很有意义。有谁知道哪种用户希望用 Emacs 编辑无格式的文件呢？Emacs 的基本模式就是针对编辑这种文件的，就像 C 模式是默认针对 C 语言文件的一样。但是当你确实知道将使用 Emacs 的人就是你自己时，那么你自己定制 Emacs 环境就有意义了。

例如，当我编写一个没有特定目的普通文件时，我很少希望进入基本模式，我希望进入文本模式。这就是为什么我要定制 Emacs 的原因：使它适合我。

通过编写或者改编“~/.emacs”文件，你就能够定制并扩充 Emacs。这是你个人的初始化文件，它的内容是由 Emacs Lisp 编写的，其作用是告诉 Emacs 做些什么。

这一章描述一个简单的“.emacs”文件，关于这个文件的更多信息参见《*GNU Emacs* 技术手册》以及《*GNU Emacs Lisp* 技术手册》的有关章节。

16.1 全站点的初始化文件

除了你个人的初始化文件之外，如果存在全站点初始化文件，Emacs 将自动加载各种不同的全站点初始化文件。这些文件的格式与你个人的初始化文件的格式是一样的。但是这些文件供所有人加载。

最常见的情况是，“site-load.el”和“site-init.el”两个全站点初始化文件被加载到 Emacs 中，然后如果创建了一个转储版本，随后就转储这两个文件。(Emacs 的转储版本可以加载得更快，然而，一旦一个文件被加载和转储了，对这个文件的改变就不会改变 Emacs，除非你自己来加载这个文件或者重新转储它。参见《*GNU Emacs Lisp* 技术手册》中“建立 Emacs”一节和“INSTALL”文件。)

其他三个全站点初始化文件在你每一次使用 Emacs 时就会被自动地加载(如果这三个文件存在的话)。这些文件中，“site-start.el”文件在用户个人的初始化文件“.emacs”加载之前被加载，“default.el”以及终端类型文件都是在用户个人的初始化文件“.emacs”加载之后被加载。

用户个人的初始化文件“.emacs”中的设置以及定义将覆盖“site-start.el”文件

（如果它存在的话）中与之有冲突的设置和定义。但是“default.el”以及终端类型文件中的设置和定义将覆盖用户个人的初始化文件中有冲突的设置和定义。（你将终端类型文件中的term-file-prefix项设置成nil，就可以避免终端类型文件的干扰。参见16.10节，“一个简单的功能扩充”。）

Emacs的发行版本中的“INSTALL”文件描述了“site-init.el”和“site-load.el”这两个文件。

“loadup.el”、“startup.el”以及“loaddefs.el”文件控制Emacs初始化文件的加载过程。这些文件都位于Emacs发行版本的“lisp”目录中，都值得好好研究。

“loaddefs.el”文件包含了许多建议，如你个人的初始化文件中应当放些什么内容，或者在一个全站点初始化文件中应当放些什么内容，等等。

16.2 为一项任务设置变量

我的Emacs第19.23版中有392个可选项，可以用edit-options命令来设置它们。这些“可选项”无外乎就是一些变量，与我们前面已经看到的用defvar定义的变量没有什么两样。

Emacs判断一个变量是否是可以设置的，是通过观看这个变量的说明文档字符串中的第一个字符来决定的；如果说明文档字符串中的第一个字符是一个星号“*”，这个变量就是用户可以自行设置的选项。（参见8.4节，“用defvar初始化变量”。）

edit-options命令列出了当人们在编写Emacs Lisp函数库时Emacs中所有可以重新设置的变量。它提供了一个易于使用的界面来重新设置这些变量。

另一方面，使用edit-options命令来设置的可选项只在你的编辑会话中有效。这些新的值并不永久保存供其他会话使用。Emacs每一次启动，它都读入其源代码中的初始定义的变量。要实现在不同会话之间永久地保存一个变化的设置，你需要在“.emacs”或者其他初始化文件中使用setq表达式来设置。

对我来说，edit-options命令的主要用法是用来建议哪些变量应当由我自己在初始化文件“.emacs”中设置。我强烈要求你仔细浏览一遍这个变量列表。

关于这个问题的更多信息，参见《*GNU Emacs技术手册*》中“编辑变量的值”一节。

16.3 开始改变“.emacs”文件

当你启动Emacs时，它加载你个人的初始化文件“.emacs”，除非你在命令行用“-q”参数告诉它不要加载这个文件。（emacs -q命令使你进入普通模式）。

“.emacs”文件包含了Lisp表达式。通常，这无非是一些设置变量的表达式，有时也有函数定义表达式。

关于初始化文件的简短描述，参见《*GNU Emacs技术手册*》中的“初始化文件‘~/.emacs’”一节。

这一章仔细讲解一些共同的东西，但是会对一个完整的、我长期使用的“.emacs”文件作一个通盘介绍。

这个文件的第一个部分由注释组成：用于提醒自己。现在，当然，我记得这些东西，但是

当我刚接触 Emacs 时，并不记得这么多。

```
;;;; Bob's .emacs file
; Robert J. Chassell
; 26 September 1985
```

瞧这个日期！我使用这个文件已经有相当长的历史了。我从那时起不断增加其内容。

```
; Each section in this file is introduced by a
; line beginning with four semicolons; and each
; entry is introduced by a line beginning with
; three semicolons.
```

这一段描述 Emacs Lisp 中注释的通常惯例。每一个以一个分号开始的行都是一个注释行，两个、三个或者四个分号分别是节和小节的标记。（参见《*GNU Emacs Lisp* 技术手册》中的“注释”一节。）

```
;;;; The Help Key
; Control-h is the help key;
; after typing control-h, type a letter to
; indicate the subject about which you want help.
; For an explanation of the help facility,
; type control-h three times in a row.
```

记住，连续键入 C-h 三次就可得到帮助信息。

```
; To find out about any mode, type control-h m
; while in that mode.  For example, to find out
; about mail mode, enter mail mode and then type
; control-h m.
```

这是“Mode help”（模式帮助）提示，就像我自己称呼的这样，是非常有用的。通常，它告诉你所有你需要知道的东西。

当然，你无需在你自己的初始化文件“.emacs”中加入这样的注释。我将这些注释加入到我的初始化文件中是因为我总是忘记模式帮助以及注释的惯例——但是我可以记得来这里看一看让自己得到提醒。

16.4 文本和自动填充模式

现在开始学习开启“Text mode”(文本模式)和“Auto Fill”（自动填充）模式的部分。

```
;;; Text mode and Auto Fill mode
; The next two lines put Emacs into Text mode
; and Auto Fill mode, and are for writers who
; want to start writing prose rather than code.

(setq default-major-mode 'text-mode)
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

这是“.emacs”文件的第一部分，这一部分没有别的功能，无非是为了提醒一个健忘的

人而已。

括号中的这两行告诉 Emacs，当找到一个文件时就开启文本模式，除非那个文件应当进入别的模式，如 C 模式等。

当 Emacs 读入一个文件时，它查看这个文件名的后缀（如果有后缀的话）。（文件名后缀是以“.”开始的那个部分）。如果文件是以“.c”结尾的，或者以“.h”结尾，那么 Emacs 就进入 C 模式。同样，Emacs 查看这个文件的第一个不是空白的行，如果这一行是“-*-C-*-”，那么 Emacs 也进入 C 模式。Emacs 拥有一个后缀列表并自动地使用它。另外，如果缓冲区的最后一页中有一个局部变量列表，Emacs 就查看缓冲区最后一页中的这个局部变量列表，根据它的内容来决定进入何种模式。

有关更多的信息，请参见《*GNU Emacs* 技术手册》中的“如何选择主要的模式”一节和“文件中的局部变量”一节。

现在回到初始化文件“.emacs”中。

后面的一行就列在下面，它是如何工作的呢?

```
(setq default-major-mode 'text-mode)
```

这一行很短，但是它确实是一个 Emacs Lisp 表达式。

我们已经熟悉了setq 函数，它将后续的变量 default-major-mode 设置为text-mode。在 text-mode 之前的单引号是告诉 Emacs 直接处理 text-mode 变量，而不是其内容。参见 1.9节“给一个变量赋值”可以得到关于 setq的资料。值得重提的主要的一点是，在“emacs”文件中设置一个变量的过程与在 Emacs 其他地方设置变量的过程没有任何区别。

第二行是：

```
(add-hook 'text-mode-hook 'turn-on-auto-fill)
```

在这一行，add-hook 命令将 turn-on-auto-fill 加到变量 text-mode-hook之后。而 turn-on-auto-fill 是一个程序的名字，它启动自动填充模式。

Emacs 每次启动文本模式，它都运行这些命令改变文本模式的某些属性。因此，Emacs 每次启动文本模式，Emacs 同时也启动了自动填充模式。

简而言之，当你编辑一个文件时，上面的第一行使 Emacs 进入文本模式，除非这个文件的后缀名、文件的第一个非空的行或者局部变量告诉 Emacs 进入其他模式。

其他动作当中的文本模式为用户设置语法表以方便用户。在文本模式中，Emacs 认为省略号是单词的一个部分就像构词要素字符一样，但是 Emacs 并不认为句点或者一个空格是单词的一个部分。因而，M-f 使你移过“it's”。另一方面，在 C模式下，M-f 则停在“it's”的“t”字母之后。

第二行使 Emacs 在开启文本模式的时候，同时开启自动填充模式。在自动填充模式中，Emacs 自动地分行使超出屏幕的过多的文本自动转移到下一行。Emacs在单词之间分行，而不是在单词中分行。

当自动填充模式被关闭时，如果你不断输入字符，它们就不断地接在这行的后面，而不换行。根据你设置的 truncate-lines 的值，你输入的单词要么显示在你的屏幕之外，要么被显示出来，当然这些单词是以相当难看和难以阅读的格式在一个连续行显示的。

16.5 邮件别名

下面是一个“开启”邮件别名的 setq 表达式，当然还包括不少提示信息：

```
;;; Mail mode
; To enter mail mode, type `C-x m'
; To enter RMAIL (for reading mail),
; type `M-x rmail'

(setq mail-aliases t)
```

这个 setq 表达式设置变量 mail-aliases的值为 t。因为t就是指“真”，这一行意思就是说，“对，使用邮件别名。”

使用邮件别名可以很方便地用简短的名字代替很长的邮件地址或者邮件地址列表。存放你的邮件别名的文件是“~/.mailrc”。使用别名的方法如下所示：

```
alias geo george@foobar.wiz.edu
```

当你给 George 一个消息时，只要指出地址 “geo” 即可；邮件程序会自动将 “geo” 别名扩展成它完整的邮件地址。

16.6 缩排模式

默认的情况下，如果要格式化一个区域，Emacs 在多个空格的地方插入制表符来代替。（例如，可能用 indent-region 命令一次性地缩排多行文本。）在终端屏幕上或者普通打印时，使用制表符将是很好的。但是当你使用 TEX或者 Texinfo 时，由于TEX忽略制表符，因此使用制表符的话将得到错误缩排的输出。

下面的表达式关闭制表符缩排模式：

```
;;; Prevent Extraneous Tabs
(setq-default indent-tabs-mode nil)
```

注意，上面的表达式值用 setq-default 函数而不是前面看到的 setq 函数。setq-default 命令只是在缓冲区中设置值而不是为变量设置它自己的局部值。

参见《*GNU Emacs* 技术手册》中的“制表符和空格”一节以及“文件中的局部变量一节”。

16.7 一些绑定键

现在该讲讲一些个人的绑定键了：

```
;;; Compare windows
(global-set-key "\C-cw" 'compare-windows)
```

compare-windows 是一个极有用的命令，这个命令将你当前缓冲区中的文本与下一个窗口中的文本进行比较。它将位点置于每一个窗口的开始进行比较，只要能够匹配就尽可能使位点移过整个缓冲区。我总是使用这个命令。

这个表达式同样显示了如何设置一个全局性的绑定键。这种设置对所有模式都是有效的。

设置全局性的绑定键的命令就是 global-set-key。这个命令之后就是绑定键。在一个

“.emacs”文件中，键的绑定是这样完成的：\C-c 代表“control-c”，这是指“同时按下 control 键和 c 键。其中的 w 是指“按下w键”。绑定键是由双引号包围的。在文档中，可以这样写：C-c w。（如果绑定 meta 键(如 M-c)而不是 control 键，应该这样写：\M-c。）关于这个问题的详细资料，可以参见《*GNU Emacs* 技术手册》中的“在用户初始化文件中重新绑定键”一节。

由这些键激活的命令是 compare-windows。注意，compare-windows 是由一个单引号开始的，否则 Emacs 将首先试图对这个符号求值来决定它的值。

双引号、C 字符前的反斜线以及单引号，都是键绑定过程的三个必要的部分，而这往往是我经常忘记的。幸运的是，我已经开始记得应当看一看自己的初始化文件，并采纳其中的提示。这就是注释的妙处。

对于这个绑定键本身：C-c w，它是由前缀键和一个单字符组成的。在这个例子中，前缀键就是 C-c，单字符就是w。这个键的集合，C-c 和其后的 w 键，是保留给用户使用的。如果你编写一个表达式来扩充 Emacs 的功能，请避免使用这些键。你可以创建一个类似 C-c C-w 的绑定键。否则，我们将会“用光”留给自己使用的键。

下面是另外一个键绑定及其注释：

```
;;; Keybinding for 'occur'
; I use occur a lot, so let's bind it to a key:
(global-set-key "\C-co" 'occur)
```

其中的 occur 命令显示当前缓冲区中包含了与某个正则表达式匹配内容的所有行。匹配的行显示在一个称为“*Occur*”的缓冲区中。这个缓冲区是作为一个菜单来显示结果的。

要取消一个键的绑定，下面的表达式将告诉你如何做。取消键绑定之后，这些键就不再起作用了。

```
;;; Unbind 'C-x f'
(global-unset-key "\C-xf")
```

之所以要取消键绑定，是因为我发现当我要键入 C-x C-f 的时候不可避免地键入了 C-x f。也就是说，当我要查找一个文件的时候，总是不小心设置了不是我所需要的文本宽度。因为我几乎从不重新设置默认的文本宽度，因此我简单地取消了这个键绑定。

下面的表达式重新绑定一个已经存在的键绑定：

```
;;; Rebind 'C-x C-b' for 'buffer-menu'
(global-set-key "\C-x\C-b" 'buffer-menu)
```

默认的情况是，C-x C-b 运行了 list-buffer 命令。这个命令在另外一个窗口中列出所有的缓冲区。因为我几乎总是要在这个窗口中做点什么，因此我更喜欢 buffer-menu 命令。这个命令不仅列出所有缓冲区，而且将位点移动到那个缓冲区中。

16.8 加载文件

GNU Emacs 社团中的许多人已经为 Emacs 编写了大量的扩展功能。随着时间的流逝，这些扩展功能经常被包含在新发行的版本之中，例如，Calendar 和 Diary 包现在都是标准的 Emacs 第19版的一部分了。但是它们曾经并不是标准的 Emacs第18版中的一部分。

（calc 是一个计算器，我认为这是 Emacs 的一个重要部分，应该成为 Emacs 的一个标准发行版的部分，只是它太大了而成为了一个独立的软件包。）

能够使用 load 命令对一个完整的文件求值，并因此将这个文件中所有函数和变量安装到 Emacs 中。例如，

```
(load "~/emacs/kfill")
```

对这个表达式求值，即从用户个人目录的“ emacs”子目录中加载“kfill.el”文件。（或者如果存在的话，加载这个文件的字节编译文件“kfill.elc”速度会更快。）

（“kfill.el”是 Bob Weiner 从 Kyle E. Jones 的“filladapt.el”包中整理出来的，并且它提供从新闻或者邮件消息，以及从Lisp C++和 PostScript 甚至 shell 注释中获得的缩排的、清晰文本的功能。我经常使用它们，也希望它能被打包进入标准的发行版本中。）

如果你要加载许多扩充功能包，就像我这样，无需精确指定扩充文件的准确路径，只要像上面那样指定它们作为 Emacs 的load-path 一部分的目录即可。然后，当 Emacs 加载一个文件时，它将查询这个目录以及它的默认的目录列表。（默认的目录列表是 Emacs 安装时在“paths.h”文件中指定的。）

下面的命令将你的“~/emacs”目录增加到已经存在的加载目录中：

```
;;; Emacs Load Path
(setq load-path (cons "~/emacs" load-path))
```

顺便说一下，load-library 是 load函数的交互接口，这个函数的完整形式如下所示：

```
(defun load-library (library)
  "Load the library named LIBRARY.
This is an interface to the function 'load'."
  (interactive "sLoad library: ")
  (load library))
```

这个函数的函数名，load-library，来自于人们使用“library”（库）来作为“file”(文件)的同义语。load-library 命令的源代码在“files.el”库中。

有另外一个交互命令与 load-file 命令完成相似的工作。参见《*GNU Emacs技术手册*》中的“Emacs 的 Lisp 代码库”，可以得到关于 load-library 和 load-file 之间差别的信息。

16.9 自动加载

除了通过加载包含指定函数的文件来实现函数的加载和安装，以及通过对函数定义求值来实现函数的安装之外，你还能够在不真正安装函数代码的情况下使用这个函数。这个函数是在它第一次被调用的时候安装的。这称为*自动加载*。

当你执行一个自动加载函数时，Emacs 自动地对包含这个函数的文件求值，然后调用这个函数。

Emacs 使用自动加载函数时执行得更快，因为自动加载函数的函数库不是直接被加载的，你在第一次使用这个函数的时候需要稍微等待一会儿，这是因为包含这个函数的文件正在被求值。

不常用的函数经常是自动加载的函数。“loaddefs.el”库包含了几百个自动加载的函数，从bookmark-set 到 workstar-mode。当然，你可能经常使用一个“不常用”的函数。在这种情况下，你应当在自己的“.emacs”初始化文件中用一个 load 表达式加载包含这个函数的文件。

在我的 Emacs 第19.23版的“.emacs”初始化文件中，我一共加载了17个库，这些库包含了原本要被自动加载的函数。(实际上，当我创建我的 Emacs 时，我应当将它包含在我的“转储”Emacs 中。但是我忘了。关于“转储”的更详细的介绍，参见《*GNU Emacs Lisp 技术手册*》中的“创建 Emacs”一节。)

你也可以将自动加载函数的相关文件包含在你个人的“.emacs”初始化文件的自动加载的表达式中。autoload 是一个内置的函数，这个函数接收五个参量。其中的最后三个是可选的。第一个参量是被自动加载的函数名，第二个参量是被加载的文件名。第三个参量是为这个函数编写的文档。而第四个参量是告之这个函数是否能被交互地调用。最后一个参量，也就是第五个参量，告诉对象是什么类型的—— autoload (自动加载) 函数可以处理函数也可以处理键图和宏。(默认情况下是函数)

下面是一个典型的例子：

```
(autoload 'html-helper-mode
  "html-helper-mode" "Edit HTML documents" t)
```

这个表达式从 html-helper-mode.el 文件中(或者如果存在的话就从 html-helper-mode.elc 文件中加载)自动加载 html-helper-mode 函数。这个文件必须是在由 load-path 定义的一个目录中。函数的文档说，这是一个帮助你用 HTML 语言编辑文档的模式。键入 M-x html-helper-mode，你能够交互地调用这个模式。(你需要复制自动加载表达式中的函数文档，因为这个函数还没有加载，因此它的文档也没有。)

参见《*GNU Emacs Lisp 技术手册*》的“自动加载”一节，可以得到更多的信息。

16.10 一个简单的功能扩充：line-to-top-of-window

这是一个对 Emacs 的简单的功能扩充，这个扩充使某一行的位置移动到窗口的顶端。我总是使用这个函数以使文本易于阅读。

你可以将下面的代码放进一个独立的文件中，并从你个人的“.emacs”初始化文件中加载它，或者你可以将这个函数包含在你个人的“.emacs”初始化文件中。

下面就是这个函数的定义：

```
;;; Line to top of window;
;;; replace three keystroke sequence  C-u 0 C-l
(defun line-to-top-of-window ()
  "Move the line point is on to top of window."
  (interactive)
  (recenter 0))
```

现在就要绑定组合键了。

虽然大部分 Emacs第18版的“.emacs”文件在第19版下工作得很好，但是它们之间还是有

区别的（当然，在第19版中有一些新的特性）。

在第19版的 Emacs 中，你可以编写这样的函数："[f6]"。在第18版中，你必须由键盘指定键序列。例如，对于一个 Zenith 29 键盘而言，当我按下它的第六个功能键的时候，键盘发送 ESC P 键序；对于一个 Ann Arbor Ambassador 型的键盘，则发送一个 ESC O F 键序列。这些键序列应被分别写成"\eP"和"\eOF"。

在我的第18版的".emacs"初始化文件中，我将 line-to-top-of-window 绑定到一个键上，这个键依赖于终端的类型：

```
(defun z29-key-bindings ()
  "Function keybindings for Z29 terminal."
  ;; ...
  (global-set-key "\eP" 'line-to-top-of-window))
(defun aaa-key-bindings ()
  "Function keybindings for Ann Arbor Ambassador"
  ;; ...
  (global-set-key "\eOF" 'line-to-top-of-window))
```

(你可以通过键入功能键来发现各个功能键的键值，然后键入 C-h l (view-lossage) 来显示最后 100 个键序列的值。)

在定义了绑定键之后，我对一个表达式求值，这个求值是根据我所使用的终端的类型从各种键序列中选择出来的。然而，在这样做之前，我关闭了预先定义的、与终端类型相关的默认绑定键，因为如果它们冲突的话就会覆盖在".emacs"中定义的键序列。

```
;;; Turn Off Predefined Terminal Keybindings

; The following turns off the predefined
; terminal-specific keybindings such as tbe
; vt100 keybindings in lisp/term/vt100.el.
; If there are no predefined terminal
; keybindings, or if you like them,
; comment this out.

(setq term-file-prefix nil)
```

下面是选择表达式本身：

```
(let ((term (getenv "TERM")))
  (cond
   ((equal term "z29") (z29-key-bindings))
   ((equal term "aaa") (aaa-key-bindings))
   (t (message
       "No binding for terminal type %s."
       term))))
```

在第19版中，功能键（包括鼠标事件和非 ASCII 字符）都是写在方括号中的，而不要引号。将 line-to-top-of-window 绑定到 F6 功能键上：

```
(global-set-key [f6] 'line-to-top-of-window)
```

简单多了！

关于绑定键的更详细的信息，参见《*GNU Emacs* 技术手册》中的“在你的初始化文件中重新绑定键”一节。

如果同时运行 Emacs 的第18版和第19版两个版本，可以用下面的条件表达式选择其中之一求值：

```
(if (string=
     (int-to-string 18)
     (substring (emacs-version) 10 12))
    ;; evaluate version 18 code
    (progn
      ... )
  ;; else evaluate version 19 code
  ...
```

16.11 键图

Emacs 使用键图(*keymaps*)来记录什么键调用什么命令。特定的模式，如 C 模式或者文本模式，都有它们自己的键图。与模式有关的键图将覆盖由所有缓冲区共享的全局键图。

`global-set-key` 函数的功能是绑定、或者重新绑定全局键图。例如，下面的键绑定表达式将 `C-c C-l` 绑定到 `line-to-top-of-window` 函数：

```
(global-set-key "\C-c\C-l" 'line-to-top-of-window))
```

与模式有关的键图是用 `define-key` 函数绑定的，它接受一个指定的键图、键以及命令作为其参量。例如，我的“`.emacs`”初始化文件包含下面的表达式，这些表达式将命令 `texinfo-insert-@group` 绑定到 `C-c C-c g`：

```
(define-key texinfo-mode-map "\C-c\C-cg"
  'texinfo-insert-@group)
```

`texinfo-insert-@group` 函数本身是对 Texinfo 模式的一个小的功能扩充，这个表达式将“`@group`”插入到一个 Texinfo 文件中。我总是用这个命令，而且喜欢键入 `C-c C-c g` 而不喜欢键入 `@group`。(“`@group`”和它的对应物“`@end group`”是将一页中的文本组织起来的两个命令，在这本书中许多的多行例子都是由这两个命令“`@group…@end group`”组织起来的。)

下面是 `texinfo-insert-@group` 函数的定义：

```
(defun texinfo-insert-@group ()
  "Insert the string @group in a Texinfo buffer."
  (interactive)
  (beginning-of-line)
  (insert "@group\n"))
```

(当然，我可以使用缩写模式来节省打字输入，而不是编写一个函数来插入一个单词，但是

我更喜欢与 Texinfo 模式中的绑定键一致的键序列。）

你将在“loaddefs.el”中看到大量的 define-key 表达式，也将在不同的模式库（如“c-mode.el”和“lisp-mode.el”中看到这些表达式）。

关于这方面的更多的信息，参见《*GNU Emacs* 技术手册》中的“定制键绑定”一节和《*GNU Emacs Lisp* 技术手册》中的“键图”一节。

16.12 X11的颜色

当你在 MIT X Window 系统上使用 Emacs 第19版及以上的版本时，你就能够定义屏幕显示的颜色。（所有前面的例子既能在第19版上运行，也能在第18版上运行，但是这一节讲述的内容只能在 第19版上起作用。）

我讨厌自己系统中的默认颜色，因此我要自己指定颜色。

在我的系统中，大部分对颜色的定义放在不同的 X 初始化文件之中。同时我在自己的“.emacs”初始化文件中做了注释，提醒我自己所做的修改：

```
;; I use TWM for window manager;
;; my ~/.xsession file specifies:
;    xsetroot -solid navyblue -fg white
```

实际上，X Window 系统的根目录根本不是 Emacs 的一部分，但是我喜欢这种提示。

```
;; My ~/.Xresources file specifies:
;     XTerm*Background:    sky blue
;     XTerm*Foreground:    white
;     emacs*geometry:      =80x40+100+0
;     emacs*background:    blue
;     emacs*foreground:    grey97
;     emacs*cursorColor:   white
;     emacs*pointerColor:  white
```

下面是在我的“./emacs”初始化文件中的设置这些变量的表达式：

```
;;; Set highlighting colors for isearch and drag
(set-face-foreground 'highlight "white" )
(set-face-background 'highlight "slate blue")
(set-face-background 'region    "slate blue")
(set-face-background
 'secondary-selection "turquoise")

;; Set calendar highlighting colors
(setq calendar-load-hook
      '(lambda ()
         (set-face-foreground 'diary-face   "skyblue")
         (set-face-background 'holiday-face "slate blue")
         (set-face-foreground 'holiday-face "white")))
```

不同的蓝色阴影使我的眼睛感觉很柔和，并使我避免看到屏幕的闪烁。

16.13 V19中的小技巧

这里列出的是第19版中设置的一些小技巧：

1) 自动根据需要改变小缓冲区的大小：

```
(resize-minibuffer-mode 1)
(setq resize-minibuffer-mode t)
```

2)开启对查询字符串的高亮显示：

```
(setq search-highlight t)
```

3)将每一个框架都设置为显示一个菜单条，并在你的鼠标移动到它上面时弹出菜单：

```
(setq default-frame-alist
      '((menu-bar-lines . 1)
        (auto-lower . t)
        (auto-raise . t)))
```

4) 设置鼠标的形状和颜色：

```
; Cursor shapes are defined in
; '/usr/include/X11/cursorfont.h';
; for example, the 'target' cursor is number 128;
; the 'top_left_arrow' cursor is number 132.
(let ((mpointer (x-get-resource "*mpointer"
                                "*emacs*mpointer")))
  ;; If you have not set your mouse pointer
  ;;     then sent it, otherwise leave as is:
  (if (eq mpointer nil)
      (setq mpointer "132")) ; top_left_arrow
  (setq x-pointer-shape (string-to-int mpointer))
  (set-mouse-color "white"))
```

16.14 修改模式行

最后，还有一个特性是我喜欢的：修改模式行。

因为我有时在一个网络系统上工作，因此我经常将平时正常显示在模式行的左边部分的“Emacs:”用我当时的系统名替代——否则的话，我会忘记自己究竟在使用哪一台计算机。另外，我总是列出默认的目录以免不知道自己处于什么位置，并且我开启行位点开关，使其显示“Line”值。我的“.emacs”初始化文件如下所示：

```
(setq mode-line-system-identification
  (substring (system-name) 0
             (string-match "\\..+" (system-name))))

(setq default-mode-line-format
      (list ""
            'mode-line-modified
            "<"
            'mode-line-system-identification
```

```
"> "
"%14b"
" "
'default-directory
" "
"%[("
'mode-name
'minor-mode-alist
"%n"
'mode-line-process
")%]--"
 "Line %l--"
'(-3 . "%P")
"-%-"))

;; Start with new default.
(setq mode-line-format default-mode-line-format)
```

我设置默认的模式行格式以允许不同的模式(如 Info)正常覆盖它。这个列表的许多元素的意义都是不言自明的：mode-line-modified 变量告诉缓冲区是否已被修改，mode-name 变量告诉模式的名称，等等。

其中的“%14b”显示当前缓冲区的名称（使用我们熟悉的 buffer-name 函数），其中的“14”定义最大显示的字符数是 14。当一个缓冲区的名称的字符数低于 14 个字符时，就用空格添满。“%[”和“%]”符号使得在每一个递归的编辑层次中使用一对方括号。“%n”则在变窄开启时显示“Narrow”。“%P”告诉你在窗口底部之上的部分占整个缓冲区的百分比，或者告诉你在“TOP”（顶端）、“Bottom”（底端）或“All”（全部）。（小写的“p”字符告诉你，在窗口顶部之上的部分所占的百分比）。“%-”则插入足够的破折号来填满一行。

对于 Emacs第19版及其后的版本，可以使用 frame-title-format 来设置一个Emacs 框架的标题。这个变量与 mode-line-format 变量有同样的结构。

模式行的格式在《*GNU Emacs Lisp 技术手册*》中的“模式行格式”一节中有详细的描述。

记住，“不要只为了喜欢 Emacs 而喜欢它”——你自己的 Emacs 可以与默认的Emacs拥有不同的颜色、不同的命令以及不同的绑定键。

另一方面，如果要进入一个没有任何定制的普通 Emacs 环境，键入：

```
emacs -q
```

就行了。这个命令将使 Emacs 不加载你个人的“~/.emacs”初始化文件。仅提供一个普通的默认的 Emacs，其他什么也没有。